Remote Programming of Biomedical Smart Sensors

David Naranjo[2,1], Laura M. Roa[1,2], Javier Reina-Tosina[3,2], and Miguel A. Estudillo-Valderrama[2,1]

[1] Biomedical Engineering Group, University of Seville, Seville, Spain
[2] CIBER de Bioingeniería, Biomateriales y Nanomedicina (CIBER-BBN), Spain
[3] Dept. of Signal Theory and Communications, University of Seville, Seville, Spain
{dnaranjo,lroa,jreina,m.estudillo}@us.es

Abstract. This paper proposes a processing architecture and a programming framework for the remote and seamless update of the algorithms used in the context of biomedical smart sensors. The generic processing architecture provides, among others, the following facilities to a Body Sensor Network in a seamless way to the user beyond functional modularity: 1) direct and immediate update with new and improved versions of the algorithms, 2) personalization of algorithms, 3) adaptability to the user context, 4) remote test of algorithms, 5) hardware reusability and sustainability, 6) parallel execution of several monitoring applications in one device, 7) structural modularity. Due to its simplicity, the proposed technique takes advantage over other solutions employed in applications that impose severe limitations on hardware/software resources of the devices, which may result in lower cost and size of them. The results obtained on two heart monitoring applications shows the viability of the proposed scheme.

Keywords: Biomedical smart sensor, Body Sensor Network, code dissemination, remote programming, structural modularity.

1 Introduction

Population ageing and the rise of chronic diseases [1] make necessary the application of new technologies for the improvement of patients welfare and the sustainability of the associated costs. Body Sensor Networks (BSN) are a promising solution in this context, because they make possible the real-time unobstructive ubiquitous monitoring of patient's health status and the detection of emergencies [2, 3]. This kind of sensor networks pose important restrictions in terms of size, computation, communication, energy consumption and data storage, related with the requirements of portability, transparency and long battery lifetime, which are even more evident when dealing with implanted sensors [4,5].

In order to reduce consumption in communications, many authors propose diverse functional modularity approaches like an initial processing within the intelligent sensors so as to reduce the number of data sent [2,6–8]. For this purpose,

B. Godara and K.S. Nikita (Eds.): MobiHealth 2012, LNICST 61, pp. 199–206, 2013.

it is necessary to provide the customization of BSNs to adapt its functionality to changes in patient's medical condition, context, activity, individual needs or lifestyles [3,6,9]. However, despite the fact that the spread of code technique is widely used in generic wireless sensor networks [10, 11], there are few authors who have considered the possibility of a remote and user-seamless update of the software in the context of BSNs.

The vast majority of the solutions for BSNs are developed in *m*otes and use frameworks based on the TinyOS operating system to support dynamic modification of code with different levels of abstraction [2, 6, 12]: A model-driven development in which the models are translated into platform-dependent code, virtual machines with byte code interpreters running on TinyOS, or programming abstractions free of low level details. However, when the restrictions of size, cost and energy limit the use of microcontrollers with the ability to integrate an operating system, even as light as TinyOS, the functional modularity faces poor performance due to the structural hardware/ software (HW/SW) constraints of the arquitecture, and many times other architectures are required for the remote programming of sensors. In such situations, a possible solution could be the sending of firmware with a very reduced set of instructions [13], which has the disadvantage that they are limited to the specific application for which they have been developed and may fail to provide sufficient flexibility to develop algorithms out of the functional domain of the instruction set.

In the present paper, it is proposed a processing architecture and programming framework designed to consume the fewest resources in smart sensors with very limited hardware-software capabilities. This architecture provides the ability to remotely upgrade the software of intelligent sensors (dissemination code) transparently to the user, a framework for running multiple applications in parallel within a single device, and the minimization of energy consumption in communications. As a proof-of-concept, the proposed scheme has been validated in a laboratory setup on an ECG virtual sensor, since both the processing and the definition of the observed characteristics of such signals are necessary to make a customization and adaptation to the activity and user context [3, 9].

1.1 Distributed Processing Architecture

In the proposed approach, smart sensors are devices able to send, wirelessly and unobtrusively, the monitored physiological information. The smart sensors perform a first information processing, in order to distribute the processing load among all the devices, abstract and compress the captured information to send only the relevant data and execute a first detection of events related with the physiological variables under monitorization. A wearable device with more computational and energy resources, referred to as Decision-Analysis Device (DAD), manages the information provided by all sensors and connects, if necessary, with the Remote Telehealthcare Center (RTC) [14] (see Figure 1).

The design of the smart sensors follows a modular scheme in order to facilitate the integration of new technologies in the devices, both for processing and communications tasks:

1. Sensor device: it constitutes the acquisition element of the physiological monitoring signals.
2. Communications module: Unit responsible for the transmission of biomedical information that releases the processing module from all the tasks associated with communications.
3. Processing unit: The intelligence of the sensor device is provided by the processing modules (PMs) that are executed in real-time and in parallel within the processing unit of the smart sensor. Each PM is capable of transmitting the captured information or the result of its processing. This data are structured into information samples generated with a given sampling frequency, which can be set via commands. In the normal operating modes of the PMs, no data are sent until the sensor device detects an alert event after processing the monitored physiological variables. This way, the overall system consumption is minimized [14].

2 Remote Programming Mechanism

Taking into account that PM interfaces are perfectly defined, the addition or removal of a new module does not affect to the rest of them, and thus the system integrity is not affected. In addition, these modules are designed to work in parallel, therefore they can cooperate easily. The developed modular scheme also allows updating the devices functionalities in order to adapt them to the user information needs. In the case of the smart sensor, the addition, update or removal of PMs can be developed by means of a firmware update of the PMs within the smart sensor (adaptable functionality, personalization and medium and context adaptation) which could be remotely performed in real time. In order to enable these characteristics, without forgetting that the smart sensor processing capabilities are limited by the device size and power consumption, a software architecture that allows the execution and the optimum management of the PMs has been researched and developed.

According to the proposed paradigm, the data memory (related to microcontroller RAM) is divided into the following blocks (see Figure 1):

1. DAT_G: related to the global variables for the device performance management.
2. DAT_I: this data block stores the information needed to manage the queues of data employed by the PMs. DAT_I is divided in turn into several subblocks, one per queue, each of which consists of the initial and final address of the queue in the data memory, the address where the next data will enter in the queue and the number of items in the queue.
3. DAT_C: area of memory where different data queues are implemented.

The information is organized into queues to facilitate its transference in this modular architecture. Several procedures to extract data from the queues, read indexed information or enter new data are also considered. The proposed scheme uses the following queues:

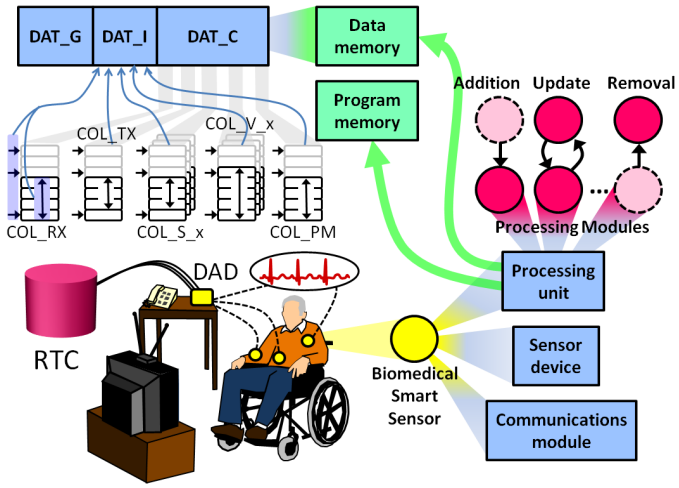

Fig. 1. Structural modularity in the proposed distributed monitoring system and data memory of the processing unit

1. COL_RX: a queue that stores the data received from the DAD.
2. COL_TX: a queue that stores the information to be sent to the DAD.
3. COL_S_x ($x = 1..N\_PM$, N_PM defined below) a queue for each PM where the information samples associated with the PM are stored.
4. COL_V_x ($x = 1..N\_PM$): a queue for each PM that stores the current values of the variables used by the PM. At the beginning of the PM execution, the data are transferred to the spaces reserved for the auxiliary variables. Then, these updated data return to the queue at the end of the execution. Since the PMs work only with auxiliary variables, the generic use of the device's memory capacity is powered.
5. COL_PM: this queue stores the information needed for the implementation of the different processing modules. It consists of a first field that indicates the number of PMs (N_PM), followed by N_PM blocks which are broken down into the following elements: the PM identifier (unique for each PM development), the initial program memory address of the PM, the number of 8 bytes blocks that compound the PM, the memory data address where the DAT_I information sub-block of the COL_S_x queue associated with the PM starts, and the address of the data memory where the DAT_I information sub-block of the COL_V_x queue starts.

On the other hand, the program memory (nonvolatile) is organized into the following blocks to facilitate the implementation of the modular software (see Figure 2):

1. $PROG\_Gen$: related to the non-modifiable generic code of the device, which is responsible for the administration and management of its overall operation as well as the PMs implemented on it.

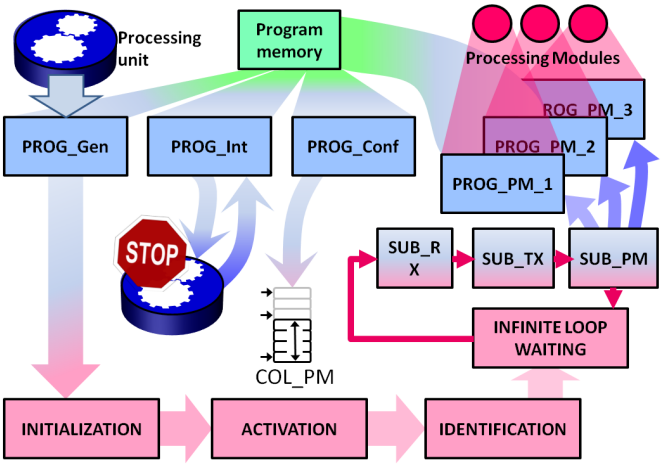

Fig. 2. Structural modularity in program memory and processing unit operation

2. *PROG_Int*: where the non-modifiable interrupt routines that control the peripheral of the smart sensor processing unit are implemented (timers, sensor data capture, transmissions, etc.).
3. *PROG_Conf*: where the code responsible for the setting of the appropriate parameters in the *COL_PM* queue after a device reset is placed, to ensure the correct management of the PMs. This code is remotely changed every time a PM is added, modified or deleted.
4. *PROG_PM_x* ($x = 1..N\_PM$): the *N_PM* modifiable memory blocks are subsequently located containing the operation code of the PMs, one per block.

The device operation is based on the execution of the following operating system algorithm corresponding to the generic code stored in *PROG_Gen*:

1. INITIALIZATION: setup of initial global parameters and device peripherals initialization.
2. ACTIVATION: activation of the executing code stored in *PROG_Conf* for the configuration of *COL_PM* queue.
3. IDENTIFICATION: process in which the smart sensor transmits the contents of the *PROG_Conf* queue to the DAD. Thus, the DAD knows the device memory map and identifies the PMs implemented in the smart sensor. This identification can also be activated through a command.
4. INFINITE LOOP WAITING: where the following subroutines are activated whenever a data from the sensor is received:
   (a) *SUB_RX*: subroutine that receives data from the DAD, which are stored in the *COL_RX* queue. The transceiver develops a low-power transmission protocol that enables the smart sensor PM to transparently communicate with the DAD. A detailed explanation of this communication protocol exceeds the scope of this paper. If the reception of a command

is completed, its processing starts. If the command is for a PM, it is introduced at the end of the COL_V_x queue associated with the PM.

(b) SUB_TX: subroutine that transmits the data to the DAD in the case that COL_TX queue is not empty.

(c) SUB_PM: subroutine responsible for the sequential execution of the PMs by means of the information stored in the COL_PM queue. Each PM begins with the execution of the received commands associated with it, if there is any, and then continues with the information processing in order to generate the information samples and perform the detection of events. Finally, it ends with the storage of the data to be transmitted to the DAD in the COL_TX queue, if necessary.

The procedure to add, edit or delete a PM is initiated by sending a programming command, which suspends the device operation until the end of the writing process in program memory. Afterwards, the smart sensor is reset.

3 Validation of the Proposed Technique

In order to validate the proposed framework a virtual ECG sensor has been implemented consisting of a PIC18F2431 microcontroller from Microchip as the processing unit, a CC2430 Chipcom transceiver that corresponds to the communications module and an algorithm in the microcontroller core that simulates the periodical reception of ECG signals. A processing module for the detection of heartbeats has been developed, which is an important parameter for the detection of tachycardia, bradycardia and fibrillation, and a processing module to estimate PQ interval duration, which is of relevance in the detection of auricle-ventricular conduction defects, atrioventricular blocks or blocks of His bundle branches [9]. Both modules generate data samples at a rate of 500 Hz and transmit data continuously. The following experimental setup was developed:

1. A transceiver located 3m apart from the virtual sensor wirelessly sent the processing module to detect the heartbeat (about 400 bytes).
2. After about 4 seconds, the device began sending information samples of heart rate.
3. Afterwards, the processing module to estimate the duration of PQ interval was wirelessly sent (about 1 Kbyte).
4. After about 7 seconds, the device began sending information samples of heart rate and the estimated duration of the PQ interval.
5. Later on, a command was wirelessly sent to remove the processing module for detecting the heartbeat, and almost immediately, the device began sending only information samples corresponding to PQ interval.

These update times are comparable to others obtained by other authors with similar code sizes and greater resources in the devices [3, 15], including generic Wireless Sensor Networks [10] (see Table 1).

Table 1. Comparison of the results obtained with related works

	Code size	Update time	Description	Resources
This work	1 KB	7 sec	Parallel processing modules managed by a generic code	ROM: 984 B RAM: 46 B (Generic code)
[3, 15]	9 KB	10 sec	Command interpreter (MedOS) that acts as a virtual machine upon FreeRTOS	ROM: 6 KB (MedOS + FreeRTOS)
[10]	10 KB	30 sec	Code dissemination system (Seluge) for wireless sensor networks running TinyOS	ROM: 45258 B RAM: 2278 B (only Seluge)

4 Conclusion

The results show the feasibility of the processing architecture and programming framework proposed as a method for updating remotely software from intelligent sensors (code dissemination) transparently to the user and to the execution of multiple applications in parallel within one device. The addition, update or removal of new functionalities does not affect to the rest of them, or the system integrity. Due to its simplicity and compared to other proposals, the proposed scheme allows to be used for smart sensors with very limited HW/SW resources, such as the case of implants. Furthermore, the approaches of other authors address the problem through a functional modularity, whereas in the present work goes further, also establishing a structural modality that affects to the system architecture. In addition, the referred generic processing architecture provides, among others, the following facilities to a BSN in a seamless way to the user: 1) update capability of the processing algorithms to include any improvement or modification that may arise as a result of future research; 2) personalization of algorithms to remotely fit the particular characteristics of the user; 3) adaptability to the context in which the user is monitored or his/her vital signs; 4) remote test of algorithms; 5) hardware reusability for different applications so as to obtain lower development costs and a greater sustainability of the devices.

Future works will be the development of security mechanisms to prevent malicious software modification, and guarantee data protection and error-free and robust execution of processing algorithms. Once established such mechanisms, the system will undergo a comprehensive experimental study to verify and validate the operation of the programming and processing architecture, detect errors, correct and document them, in accordance with the regulations for medical devices (FDA or European Commission). Furthermore, the proposed system enables the software update to correct problems identified a posteriori, according to the regulations, but with the advantage of being performed immediately.

Acknowledgments. This work was supported in part by the CIBER de Bioingeniería, Biomateriales y Nanomedicina (CIBER-BBN) and the intramural Grant

PERSONA, in part by the Instituto de Salud Carlos III under Grants PI082023 and PI11/00111, and in part by the Dirección General de Investigación, Tecnología y Empresa, Government of Andalucía, under Grants P08-TIC-04069 and TIC6214. CIBER-BBN is an initiative funded by the 6th National R&D&i Plan 2008-2011, Iniciativa Ingenio 2010, Consolider Program, CIBER Actions and financed by the Instituto de Salud Carlos III with assistance from the European Regional Development Fund.

References

1. Thorpe, K., Philyaw, M.: The Medicalization of Chronic Disease and Costs. Annu. Rev. Public Health 33, 409–423 (2012)
2. Raveendranathan, N., et al.: From Modeling to Implementation of Virtual Sensors in Body Sensor Networks. IEEE Sensors J. 12(3), 583–593 (2012)
3. de Barbosa, T.A., da Rocha, A.: A Smart System to Program Body Sensor Networks. In: 5th IEEE Int. Conf. on Intelligent Systems, pp. 168–172 (2010)
4. Ghasemzadeh, H., Loseu, V., Ostadabbas, S., Jafari, R.: Burst Communication by means of Buffer Allocation in Body Sensor Networks. IEEE J. Sel. Areas Commun. 28(7), 1073–1082 (2010)
5. Mitsch, S., Kurschl, W., Schoenboeck, J.: Modeling Distributed Signal Processing Applications. In: 6th Int. Workshop on Wearable and Implantable Body Sensor Networks, pp. 103–108 (2009)
6. Zhu, Y., Keoh, S.L., Sloman, M., Lupu, E.: A Lightweight Policy System for Body Sensor Networks. IEEE Trans. Netw. Service Manag. 6(3), 137–148 (2009)
7. Mondal, N., Zaman, S., Al Masud, A., Alam, J.: Comparisons of Maximum System Lifetime in Diverse Scenarios for Body Sensor Networks. In: 11th Int. Conf. on Computer and Information Technology, pp. 73–78 (2008)
8. Nabar, S., Walling, J., Poovendran, R.: Minimizing Energy Consumption in Body Sensor Networks Via Convex Optimization. In: Int. Conf. on Body Sensor Networks (BSN), pp. 62–67 (2010)
9. Augustyniak, P.: Autoadaptivity and Optimization in Distributed Ecg Interpretation. IEEE Trans. Inf. Technol. Biomed. 14(2), 394–400 (2010)
10. Hyun, S., Ning, P., Liu, A., Du, W.: Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks. In: Int. Conf. on Information Processing in Sensor Networks, pp. 445–456 (2008)
11. Miller, C., Poellabauer, C.: Paler: A Reliable Transport Protocol for Code Distribution in Large Sensor Networks. In: 5th IEEE Conf. on Sensor, Mesh and Ad Hoc Communications and Networks, pp. 206–214 (2008)
12. Kowalczuk, J., Vuran, M., Perez, L.: A Dual-Network Testbed for Wireless Sensor Applications. In: IEEE Global Telecommunications Conf., pp. 1–5 (2011)
13. Passama, R., Andreu, D., Guiraud, D.: Computer-Based Remote Programming and Control of Stimulation Units. In: 5th Int. IEEE/EMBS Conf. on Neural Engineering, pp. 538–541 (2011)
14. Naranjo, D., Roa, L., Reina, J., Estudillo, M.: Personalization and Adaptation to the Medium and Context in a Fall Detection System. IEEE Trans. Inf. Technol. Biomed. 16(2), 264–271 (2012)
15. Barbosa, T., Sene, I., da Rocha, A., Nascimento, F., Carvalho, H., Camapum, J.: Application-Oriented Programming Model for Sensor Networks Embedded in the Human Body. In: 28th Int. IEEE Conf. on Engineering in Medicine and Biology Society, pp. 6037–6040 (2006)