From Modeling to Implementation of Virtual Sensors in Body Sensor Networks

Nikhil Raveendranathan, Stefano Galzarano, Vitali Loseu, Raffaele Gravina, Roberta Giannantonio, Marco Sgroi, Roozbeh Jafari, and Giancarlo Fortino

***Abstract*—Body Sensor Networks (BSNs) represent an emerging technology which has received much attention recently due to its enormous potential to enable remote, real-time, continuous and non-invasive monitoring of people in health-care, entertainment, fitness, sport, social interaction. Signal processing for BSNs usually comprises of multiple levels of data abstraction, from raw sensor data to data calculated from processing steps such as feature extraction and classification. This paper presents a multi-layer task model based on the concept of Virtual Sensors to improve architecture modularity and design reusability. Virtual Sensors are abstractions of components of BSN systems that include sensor sampling and processing tasks and provide data upon external requests. The Virtual Sensor model implementation relies on SPINE2, an open source domain-specific framework that is designed to support distributed sensing operations and signal processing for wireless sensor networks and enables code reusability, efficiency, and application interoperability. The proposed model is applied in the context of gait analysis through wearable sensors. A gait analysis system is developed according to a SPINE2-based Virtual Sensor architecture and experimentally evaluated. Obtained results confirm that great effectiveness can be achieved in designing and implementing BSN applications through the Virtual Sensor approach while maintaining high efficiency and accuracy.**

***Index Terms*—Body sensor networks, SPINE, signal processing, virtual sensors.**

I. Introduction

THE wide variety of potential applications for Wireless Sensor Networks (WSNs) in conjunction with severe resource constraints [1] makes difficult to design an operating system that is both sufficiently lightweight and powerful. Because of resource abundance, traditional desktop operating systems and environments provide APIs for many different problems and domains. However, operating systems for WSNs, such as TinyOS [2], provide only minimal facilities, such as communication and process scheduling. For problems which share common structure, the abstraction provided by a software framework can enhance developer productivity and exploit problem- or domain-specific optimizations.

Middleware exploiting such structure for conventional wireless sensor networks includes TinyLIME [3], GSN (Global Sensor Network) [4], and ATAG (Abstract Task Graph) [5]. In the context of the wireless networks the idea has been explored in several works [4]–[6]. Authors in these works propose generic abstractions for all system components, which allow for easier integration of new components and functionality on the system. However, in specific application domains they usually are not so effective in providing efficient and optimized support to specific tasks at communication, sensor and processing levels. In fact, they aim at programming general-purpose WSN applications and not domain-specific WSN applications so that customization and tailoring need to be carried out by programmers which not only increases programming efforts but also likely introduces not optimized components.

Body Sensor Networks (BSNs) represent a new application domain which has received much attention recently due to potential benefits for health-care, entertainment, fitness, sport, social interaction. There are several properties of BSNs not present in standard WSN deployments, including single-hop networking, a significantly more powerful node called base station (usually a PC or a smartphone/PDA), and applications requiring extensive signal processing and pattern recognition. These properties offer considerable potential for optimization and simplification. Among the domain-specific frameworks (Codeblue [7], Titan [8], SPINE [9]–[11] and RehabSpot [12]) available for the development of BSN applications, the SPINE framework [9], [10] has distinctive features in terms of effectiveness, efficiency and usability that allows for a more rapid prototyping of efficient solutions of signal processing on BSN sensor nodes. In particular, the SPINE framework allows for implementation of multiple signal processing functions, either on top of each other or running independently on top of the sampled sensor data. Furthermore, SPINE also facilitates selective activation of these functions at run-time offering the flexibility to vary the complexity at the node side. Although frameworks such as SPINE provide effective and efficient abstractions for BSN application development, the exploitation of the concept of virtual sensor, recently introduced also in the context of wireless sensor networks [6], could further enhance the modeling effectiveness of the solutions provided.

In this paper, we present a task abstraction layer specifically conceived for BSNs, called BSN-oriented Virtual Sensors

Manuscript received November 08, 2010; revised January 23, 2011, February 08, 2011; accepted February 09, 2011. Date of publication February 28, 2011; date of current version February 01, 2012. This work was supported in part by the Cooperating Objects Network of Excellence (CONET), funded by the European Commission under FP7 under Contract FP7-2007-2-224053, and in part by Telecom Italia. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Aime Lay-Ekakille.

N. Raveendranathan and V. Loseu are with the Embedded Systems and Signal Processing Lab, University of Texas at Dallas, Dallas, TX 75080 USA.

S. Galzarano, R. Gravina and G. Fortino are with the Department of Electronics, Informatics, and Systems, University of Calabria, Rende 87036, Italy (e-mail: g.fortino@unical.it).

R. Giannantonio is with the Telecom Italia, Turin 112, Italy.

M. Sgroi is with the So.Tel, Rome, Italy.

Digital Object Identifier 10.1109/JSEN.2011.2121059

(BVS), which allows for multiple layers or stages of processing. We describe the implementation of BVS through SPINE2 [13], a new version of SPINE based on a task-oriented model. Specifically, the application of BVS is explained by developing a module based on Hidden Markov Model (HMM) to extract temporal parameters from gait with the final aim to define a Gait Virtual Sensor useful for continuous and real-time postural analysis of assisted livings.

The rest of this paper is organized as follows. In Section II, middleware approaches similar to virtual sensors in the field of WSN are discussed. Section III describes the BSN-oriented Virtual Sensor Architecture in detail. Section IV covers the SPINE2 framework highlighting its architecture, the main task-oriented programming abstractions that it offers and the Virtual Sensor implementation through SPINE2 tasks. Section V introduces the application context, details the Gait Virtual Sensor modeling and implementation and discusses the results obtained. Finally concluding remarks are provided and directions of future work delineated.

II. Related Work

The concept of Virtual Sensor has been investigated in a few research works [6], [14]–[16] in the context of WSNs.

In [6], authors define Virtual Sensors as software sensors that provide indirect measurements of abstract conditions by combining sensed data from a group of heterogeneous physical sensors. In particular, a virtual sensor is specified through four parameters: (i) input data types, which are physical (low-level) data types required to compute the desired abstract measurement; (ii) aggregator, which is a generic function defined to operate over the specific (possibly heterogeneous) input data types to calculate the desired measurement; (iii) resulting data type, which is the abstract measurement type that is a result of the aggregation; (iv) aggregation frequency, namely the frequency with which this aggregation should be made. This frequency determines how consistent the aggregated value is with actual conditions (i.e., more frequently updated aggregations reflect the environment more accurately but generate more communication overhead). Authors also present a middleware for programming WSN applications designed around the defined virtual sensor abstraction. Specifically, the middleware, implemented in nesC on TinyOS, provides an API for programming virtual sensors, services for sensor discovery and communication with data sources, and facilities for the virtual sensor deployment. Although the proposed approach is well suited for general-purpose WSN applications, it should be further customized to actually support the characteristic abstractions of BSN applications. In fact, the approach does not consider fine-grained, signal-processing-oriented virtual sensors and is mainly based on query-driven virtual sensors (based on a pull model) instead of data-driven virtual sensors (based on a push model) that would have higher performance in BSNs.

In [14] a framework for building virtual sensors and actuators in wireless sensors and actuators networks is presented. In particular, authors propose virtual nodes as a programming abstraction simplifying the development of decentralized WSN applications. The data acquired by a set of sensors can be collected, processed according to an application-provided aggregation function, and then perceived as the reading of a single virtual sensor. On the other hand, a virtual actuator provides a single entry point for distributing commands to a set of real actuator nodes. The set of physical nodes to be abstracted into a virtual one is specified using logical neighborhoods [17]. In particular, a virtual sensor is specified as a set of inputs from neighbor sensors or virtual sensors and an aggregation function processing the received inputs and producing an output. Virtual sensor and actuators are programmed using a high-level programming language (an extension of SPIDEY) and then translated into nesC/TinyOS through the SPIDEY translator purposely extended. Although the approach allows for hierarchical composition of virtual sensors, it is more suitable for environmental monitoring or building automation rather than for BSN as it does not provide BSN-oriented signal-processing-intensive abstractions and optimized fine-grain virtual sensors.

Authors in [16] present SenQ, a multilayer embedded query system, which enables user-driven and peer-to-peer in-network query submitted by wearable interfaces and other resource-constrained devices. Complex virtual sensors and user-created streams can be dynamically discovered and shared, and SenQ is extensible to new sensors and processing algorithms. In particular, SenQ enables a developer to use its embedded query submission capabilities to collect data streams from local and remote sensors for custom processing, and export the results as a virtual sensor. This encapsulates the complex, hierarchical stream processing as a low-level sensor type that can be discovered, queried, and viewed as any other. Although SenQ provides mechanisms for defining low-level virtual sensors, and was used to monitor assisted livings through BSNs, a well-defined virtual sensor programming abstraction is not offered.

The concept of Virtual Sensor Networks (VSNs) has also been described in multiple works [15], [18], [19]. In [18], authors present VIP Bridge that aims at connecting heterogeneous sensor networks with IP based wired/wireless networks, and integrate these sensor networks into one VSN. This allows for discovering and querying of sensor nodes located in different and heterogeneous sensor networks. Jayasumana *et al.* [19] present another approach to the definition and implementation of VSNs that provides protocol support for the formation, usage, adaptation and maintenance of subsets of sensors collaborating on specific tasks. VSNs can also enable applications that involve dynamically varying subsets of sensor nodes collaborating tightly to achieve the desired outcomes, while relying on the remaining nodes to achieve connectivity and overcome the deployment and resource constraints. In [15], an agent-based approach for VSNs is proposed. Agents support VSN connectivity, coverage, membership, formation as well as power management. The latter is conducted at a finer level than simply a binary sleep-active decision through dynamic coordination among agents resident on different nodes.

In the context of our paper, such proposals can be customized for enabling the integration of BSNs to have networks of BSNs on which Virtual Sensors can be created for monitoring not only a given person but also an even large group of people wearing

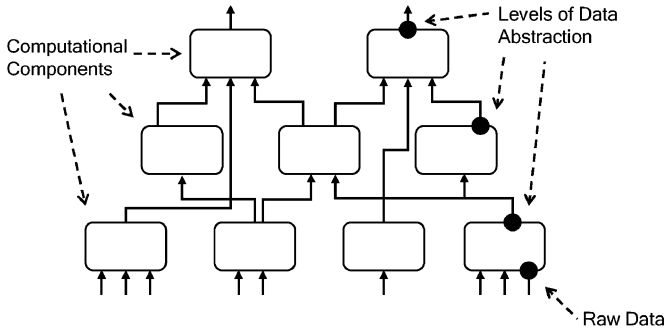

Fig. 1. Multi-layer signal processing.

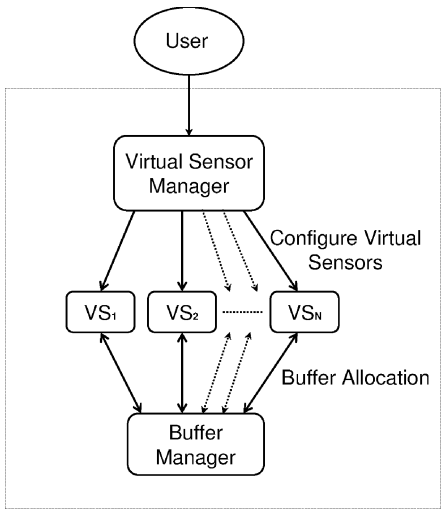

Fig. 2. BVS architecture.

heterogeneous BSNs and, notably, their gesture-initiated interactions [20].

III. BSN-Oriented Virtual Sensor Architecture

Physical sensors map an observed physical quantity, such as temperature, acceleration, or sound, onto a data value and produce an output. The output is generated when inputs change, as the result of an event, or in response to a (timed) request. Physical sensors are transducers converting values from one form to another using physical processes. Signal processing algorithms convert values using digital processes. This observed similarity is the motivation behind the virtual sensor abstraction. Every processing task can be represented as a virtual sensor. Therefore, if we consider a complete BSN system, we can model its data processing part as a multi-level hierarchy of virtual sensors as shown in Fig. 1. Moreover, virtual sensors may be implemented directly in a programming language, or as networks of already existing virtual sensors.

Fig. 2 shows the defined BSN-oriented virtual sensor system architecture. A user requests certain outputs given specified inputs. This request is handled by the Virtual Sensor Manager, which configures a set of virtual sensors to handle the computational task. Virtual sensors use the Buffer Manager to setup communication through the use of efficient buffers. Once configured, the system is activated, and virtual sensors cooperate to produce the final outputs.

Virtual sensors are defined in Section III-A. Details of the virtual sensor manager operation are described in Section III-B. Finally, buffer manager operation is described in Section III-C.

A. Virtual Sensor Definition

Software frameworks are usually introduced to provide programmers with abstractions to isolate them from low-level implementation details. Virtual sensors provide a new level of abstraction at the software level by allowing signal processing tasks to be defined and composed easily. Furthermore, VS abstractions allow signal processing tasks to be modified or changed at design or runtime without affecting the rest of the system. In Fig. 1 every component represents a processing task applied to a stream of data originated from physical sensors and can be modeled as a virtual sensor. The output of each virtual sensor is defined by a set of inputs and its configuration. More formally, a virtual sensor i, denoted as VS_i, is defined as

$$\mathrm{VS}_i = \{I_i, O_i, C_i\} \tag{1}$$

where I_i denotes the set of inputs, O_i denotes the set of outputs, and C_i denotes the configuration of VS_i. The configuration of each virtual sensor defines the type of its inputs and outputs, the particular implementation used for a given computational task, and a set of parameters required for a particular implementation. In particular, C_i is defined as

$$C_i = \{\vec{t}_{\mathrm{in}}, \vec{t}_{\mathrm{out}}, d, p\} \tag{2}$$

where $\vec{t}_{\mathrm{in}}$ is a vector that describes the types of inputs I_i, $\vec{t}_{\mathrm{out}}$ is defined similarly for the outputs O_i, d represents the specific VS implementation, and p denotes the VS configuration parameters. In particular, if the user does not specify d, the Virtual Sensor Manager (described below) will select the implementation.

This definition provides high modularity for application design. In fact, different configurations of the same virtual sensor can be easily substituted without requiring changes in the rest of the design. This property therefore enables a component-based approach for application development in which an application is assembled out of well defined components appositely interconnected. Moreover, it can be used when environmental changes require a new implementation of a particular signal processing component for a given application. Alternative implementations do not need to be loaded into main memory at all times. They can be stored in flash memory, or transferred over the air upon request.

VS can be further composed to create higher-level VS. This allows to define multiple abstraction levels that capture the successive processing and interpretation of sensor data and system components that perform data fusion. High-level VS identify abstractions that are useful to support code modularity and reusability. In fact, if an implementation of a VS is replaced with another one, where one or more VS components are changed but the interface is the same, there is no need to change the rest of the system.

More formally, the composition of n Virtual Sensors to form an higher-level VS can be defined as follows:

$$\mathrm{VS}^* = \langle \mathrm{VS}_1, \mathrm{VS}_2, \ldots \mathrm{VS}_n \rangle = \{I^*, O^*, C^*, L\} \tag{3}$$

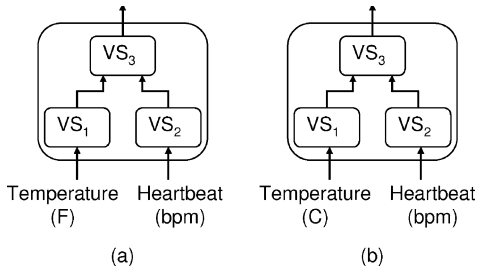

Fig. 3. Example of input modification in virtual sensors.

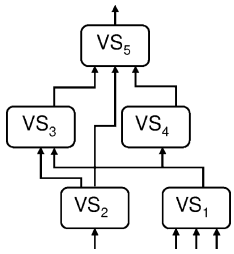

Fig. 4. Example of input/output dependency in virtual sensors.

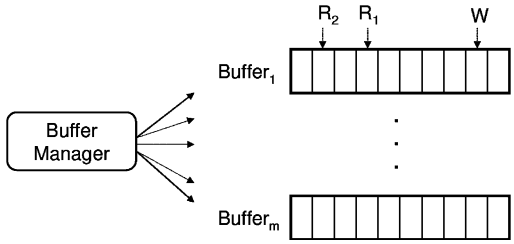

Fig. 5. Buffer manager overview.

where $I^* \subseteq I_1 \cup I_2 \cdots \cup I_n$, $O^* \subseteq O_1 \cup O_2 \cdots \cup O_n$, $C^* = \{C_1, C_2, \ldots C_n\}$, and L is the set of links connecting outputs and inputs of $\{\mathrm{VS}_1, \ldots, \mathrm{VS}_n\}$

B. Virtual Sensor Manager

Once all virtual sensors are configured, no additional control is required during execution. However, configuration requires significant support from the Virtual Sensor Manager (VSM). The VSM is responsible for creating and configuring virtual sensors and connections among virtual sensors. The following subsections will describe the main functionalities of the VSM (virtual sensor configuration and overall system configuration).

1) Virtual Sensor Configuration: the current configuration of a virtual sensor may be invalidated by changes in its inputs or connections with other virtual sensors, therefore reinitialization could happen at any time. For example, Fig. 3(a) describes a system that takes a temperature reading in Fahrenheit, and a heart rate in beats per minute. In Fig. 3(b) a new thermometer, that produces output in Celsius, is introduced. VS_1 has to be reconfigured to handle such change. To be able to configure/reconfigure the system at run time, the VSM manages a table that maps each available combination of possible inputs and outputs to the appropriate virtual sensor implementation. This can be represented by the set A. Each entry $a \in A$ is defined as:

$$a = \{\vec{t}_{\mathrm{in}}, \vec{t}_{\mathrm{out}}, \psi\} \tag{4}$$

where ψ is a particular virtual sensor implementation.

If the modification is not drastic enough to require changing the virtual sensor implementation, reconfiguration can alter parameters of a given implementation. During the configuration of a virtual sensor, VSM includes the address of the selected virtual sensor implementation and the required configuration parameters.

2) Overall System Configuration: while individual virtual sensors do not hold any information about other virtual sensors, the overall system relies on their cooperation. At the beginning of the system execution, the VSM receives the VS topology configuration graph. Based on the requirements of the topology configuration, the VSM initializes the appropriate VSs and connects them as required. Input and output types are a property of each virtual sensor. An output of one of the virtual sensors can also be an input of another virtual sensor. For example, in Fig. 4, configuration of VS_3 and VS_4 depends on the input they receive from VS_1. To simplify the configuration and reconfiguration process, the VSM initializes VSs in a specific order, to meet the requirement that each virtual sensor cannot be created until all inputs are configured. This ordering can be determined with a topographical sort of the topology configuration graph.

C. Buffer Manager

Signal processing for BSNs often relies on combining data from multiple sources and locations. As a result, virtual sensors can have multiple inputs from different sensor nodes. To avoid synchronization issues, virtual sensors implicitly use buffers for communication. The Buffer Manager (BM) controls dynamic buffer allocation and manages data flow in the system.

When a virtual sensor is created and configured, it initiates a data buffer for its output. The virtual sensor contacts the BM and requests the creation of a buffer sufficient to hold its output. The BM allocates a circular buffer of the required size and returns the `bufferID`. This `bufferID` is propagated by the VSM to other virtual sensors that are interested in data of this particular buffer. To read from a buffer, a virtual sensor must register with the buffer as a reader, specifying the number of samples it can consume at a time. Every time the producer writes to the buffer, the BM checks if the buffer has enough information for any of the readers, and signals them when they can access the data. Fig. 5 shows an overview of the BM operation. In particular, it shows that BM keeps track of buffers by ID, tracking the point where the producer (e.g., W) is writing to, and where each individual reader (e.g., R_1, R_2) is reading from. If the producer VS is reconfigured, and its output is changed, the BM removes the buffer that is associated with the previous output and initiates a new buffer, based on the new configuration information.

IV. Implementing Virtual Sensors in SPINE2

A. The SPINE2 Framework: An Overview

SPINE2 [13], [21] is a novel framework specifically designed for the development of signal processing applications on WSNs through a programming abstraction based on the *task-oriented paradigm*. The framework provides a set of graphical constructs allowing for the representation of the application behavior by

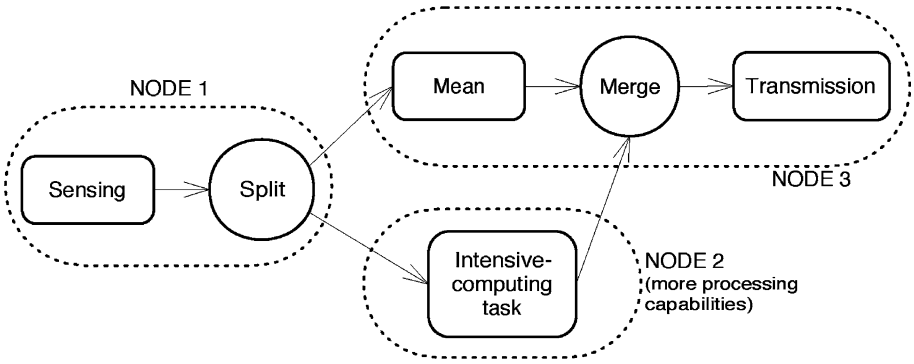

Fig. 6. Example of SPINE2 task-oriented distributed application.

abstracting away low-level details. According to the task-oriented paradigm, an application can be effectively specified as a set of interconnected tasks. Each task represents a particular activity, such as a sensing operation, a processing function or a radio data transmission. A task connection represents a relationship between tasks which generally consists of a temporal or a data dependency. Designing an application as a composition of elementary blocks with well-defined interfaces enables a more rapid application development, run-time reconfiguration and a simplified software maintenance.

The main characteristics of the SPINE2 framework are:

- *Platform independence and quick portability*: the framework lends itself to be rapidly and simply portable onto different C-like sensor architectures. Specifically, only the components constituting the platform-dependent part of the framework architecture (such as sensors and radio drivers) have to be provided for supporting a new target sensor platform.
- *Extensibility*: the defined task-oriented paradigm makes it possible to a straightforward addition of new functionalities beside the already existing ones. This is carried out by defining new tasks, which represent further computing capabilities, without any change to the underlying run-time logic or to the other task definitions.
- *Modularity*: the framework architecture composed of several and independent functional modules allows for a more rapid implementation time and a more effective software maintenance and improvement. For example, it may be possible that future requirements need a different way for managing the tasks execution. Thanks to modularity, the modifications made by the framework developers affects only the correspondent modules without the risk of causing damages to the rest of the architecture.

An example of SPINE2 task-oriented application description is depicted in Fig. 6. The set of interconnected tasks forms a direct graph representing chains of operations that include sensor data acquisition, data processing and merged result transmission. It is worth noting that, as the framework manages the deployment phase of a distributed application, users can decide which node of the wireless sensor network each task is allocated on. Therefore, as shown in the example, the user can allocate a task requiring more resources onto nodes providing more computational capabilities.

The SPINE2 task-oriented definition language consists of three main task categories: *data-processing tasks*, *data-routing tasks*, and *time-driven tasks*. The first category makes it available functions related to data processing and archiving, the second one provides store-and-forward and data replication functionalities, whereas the latter offers a mechanism for time-driven tasks. Every task is coupled with a description consisting of a set of configuration parameters defining the behavioral characteristics of a given instance of a task type. The defined task types are the following.

- *TimingTask*: allows to define timers for timing the execution of other tasks.
- *SensingTask*: defines sensing operations on a sensor node and includes a timer for setting the sensor sampling time.
- *ProcessingTask*: performs data processing functions and algorithms; particular operations are the so called "feature extractions" which are mathematical function applied to a data stream, such as mean, variance, etc.
- *TransmissionTask*: allows the transmission of data generated by other tasks, sending them to a specific addressee node. Generally, it is used for sending data and information to the coordinator whereas, implicit data transmission takes place in the case of connected tasks allocated onto different nodes.
- *StoringTask*: stores data into the flash memory.
- *LoadingTask*: retrieves data from the flash memory to be used by other tasks of the application.
- *SplitTask*: duplicates incoming data to every output links to make them available to other tasks.
- *MergeTask*: merge data coming from its multiple inputs into its single output.
- *HistoricalMergeTask*: is similar to the MergeTask but it supports a sequence of merge operations over the time, before outputting them.

The architecture of the framework is split into two components: one component runs on the coordinator of the WSN, the other one is executed on the sensor nodes. The coordinator side of the framework is a Java application through which the sensor network is managed and the task-oriented application defined. Moreover, it gathers pre-elaborated data coming from sensor nodes and eventually passes them to a specific application for more complex data processing and visualization. The sensor node side is adapted on top of the sensor node operating system.

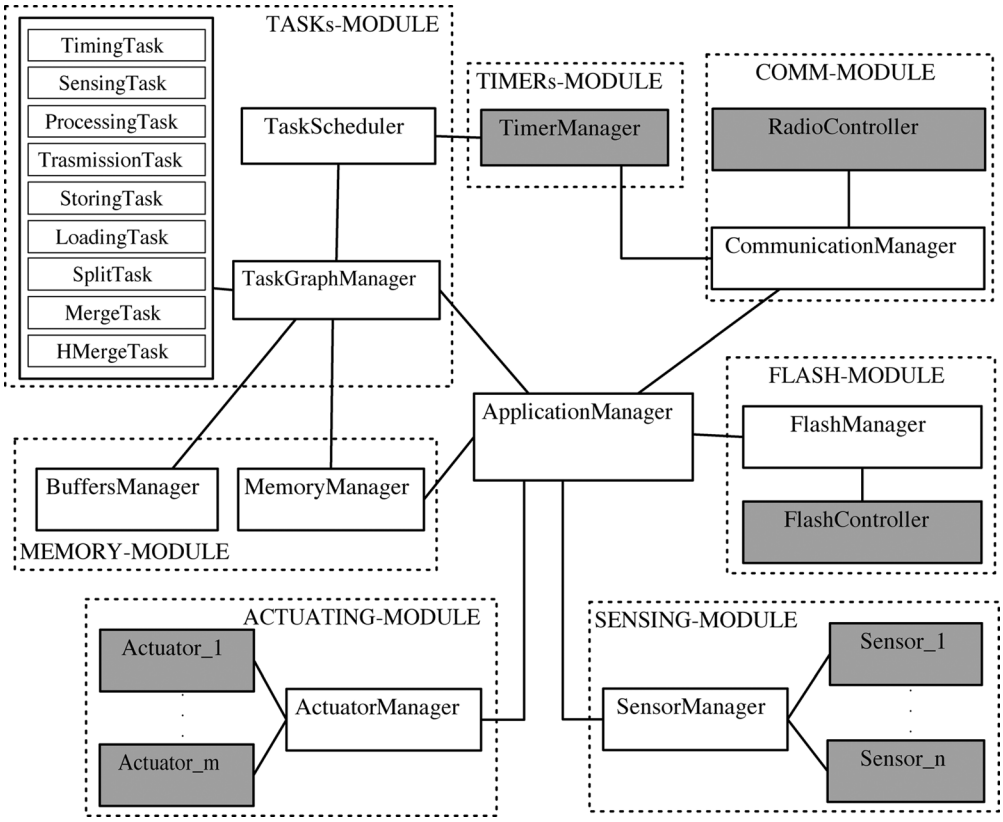

Fig. 7. SPINE2 node-side software architecture.

In particular, it carries out two main functions: (i) handling of the messages coming from the coordinator, such as the ones used for configuring the subset of tasks assigned to the node, and coming from other nodes; (ii) interpretation and execution of the task specifications.

According to a component-based approach, the node-side software architecture (see Fig. 7) is structured as a set of independent but interacting components, each of which has been carefully designed to provide a well-defined functionality. In particular, a subset of these components (the colorless ones) represents the *core framework* of the node-side system. They are implemented in the C language, so that they can be easily ported to practically every "C-like" compatible sensor platforms, without the need for changing their code. However, the very high portability of the framework is not only enabled by the use of the C language, but mainly by supporting a strong software decoupling between the platform-independent runtime execution logic (such as task management, application-level message handling and on-board sensor abstraction management) and the platform-dependent components (the gray colored ones) needed for accessing the services and the resources provided by the specific platform on which the software architecture is running. The platform-dependent components have to be specifically provided for each further new sensor platform and cannot be therefore reused for different hardware/software technologies. In practice, they represent adaptation components (or platform-specific drivers) which have the function of bridging the core with a particular operating system/environment through well-defined interfaces.

Currently, the adaptation components are available for supporting the development of SPINE2 task-oriented applications on TinyOS-based platforms [2] and the Z-Stack architecture [22], namely the ZigBee-compliant implementation provided by Texas Instruments.

B. SPINE2-Based Virtual Sensors

The Virtual Sensor architecture described in Section III is straightforwardly implemented through the SPINE2 framework. In Fig. 8 basic conversion schemas for the translation of Virtual Sensors into SPINE2 task-oriented applications are shown. In particular, only simple (flat) virtual sensors have been taken into consideration as it is quite intuitive to translate a virtual sensor defined as composition of flat virtual sensors.

In the most simple case, a virtual sensor defined as a basic functional block incorporating some kind of operation on its single input can be translated into a SPINE2 *data-processing task* (see Fig. 8(a)). In fact, a generic SPINE2 data-processing task (such as *dpTask*) is defined as a functional component having a single input and a single output, differently from the data-routing task. Obviously, the operations that have to be performed by the task (specified by its configuration) depend on the actual functionalities of the virtual sensor. If the virtual sensor does not have a generic input but raw data (such as data coming form an hardware sensor on a wireless node), the corresponding translation includes the introduction of a SensingTask, specifically configured for representing the digital data source (see Fig. 8(b)). Finally, Fig. 8(c) shows the translation of a simple virtual sensor having multiple inputs and outputs.

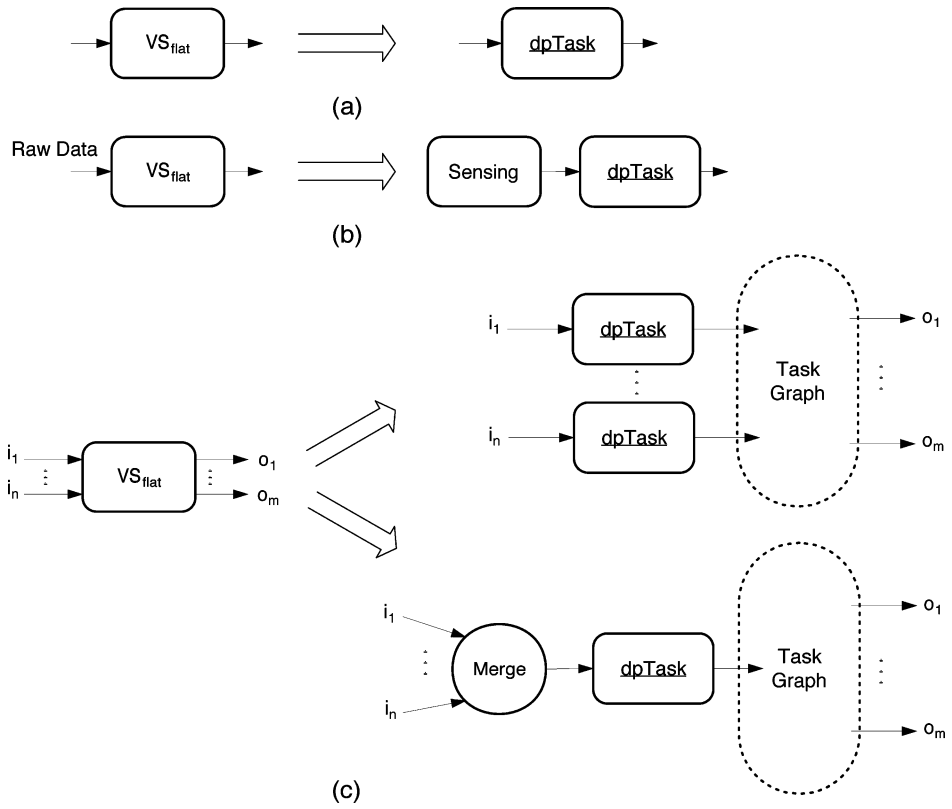

Fig. 8. Translation of Virtual Sensors into SPINE2 task-oriented models.

In this case, the corresponding SPINE2 tasks can be configured in several ways on the basis of the actual definition of the virtual sensor and of the type description of its inputs/outputs. In particular, two different translations are shown. In the first one, we have a single data-processing task for each input. These tasks, along with the not-specified Task Graph, carry out the overall computational operation performed by the virtual sensor. Conversely, in the other translation, inputs are merged by a single data-routing task (namely, the MergeTask) and provided to a generic task graph. In either case, the more complex the function defined for the virtual sensor gets, the more complex the set of actual interconnected tasks would be. Of course, there could exist a more generic SPINE2 translation in which some of the inputs merge on an AggregationTask, the other ones become inputs of data-processing tasks.

It is worth noting that the two application modeling abstractions, virtual sensors and tasks, have strong similarities. In fact, both of them enables creation of applications in a modular and easily reconfigurable way by using elementary functional blocks (virtual sensors or tasks) which do not have any functional couplings with each others. This is due to the fact that they have no knowledge of the provenance of their inputs nor the destination of their outputs.

V. A Gait Virtual Sensor

As a test application for virtual sensors in SPINE2, we implemented a virtual sensor-based system for automatic event annotation based on a left-right Hidden Markov Model (HMM) [23]. The HMM associates each data sample with a state. States typically consist of multiple data samples and the transition from one state to another starts with an event which consists of a single sample. By training the model, we can identify these key events within a movement. Walking being a cyclical activity, events and states repeat over a period of time.

The HMM annotation system consists of four parts:

1) Sampling of acceleration data from physical sensors. We used LIS3LV02DQ MEMS based accelerometer from ST Microelectronics for the data collection. The data is collected at a sampling frequency of 20 Hz which is sufficient for slow movements like walking. The data is copied into the buffer pool allocated by the Buffer Manager as it gets collected.
2) Pre-filtering of sampled data. The main purpose of pre-filtering is to remove any noise from the original data. The first stage of pre-filtering involves calculating the 5-point moving average for removing high frequency noise. This data is then normalized by subtracting 100-point mean and dividing by 100-point standard deviation to get the final pre-filtered data.
3) Extraction of features from filtered data. The pre-filtered data stream is processed by the feature extractor and the first derivative, second derivative and the existence of any peaks is calculated [23]. These three features, along with the prefiltered data sample is passed on to the next virtual sensor.
4) HMM-based annotation of events from the extracted features. The annotation is done based on the probability of occurrence of a state. This requires training of the model to create the tables for mapping the features to the probability of observing a state. In addition, we also consider the

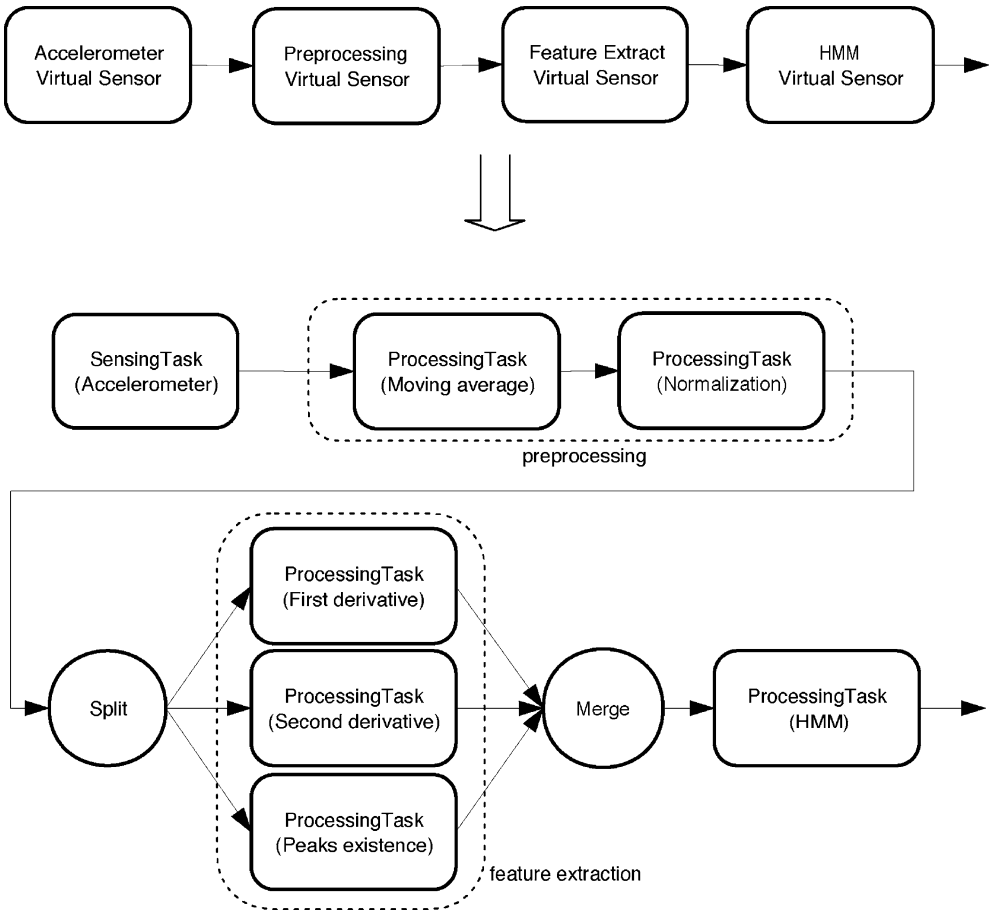

Fig. 9. GAIT analysis application.

probability of a state transition. These two tables are generated based on training which was done in MATLAB.

The system uses a single sensor node, and only one implementation was created for each virtual sensor.

The Gait Virtual Sensor is defined as a linear interconnection of four virtual sensors as described in Fig. 9. Arrows connecting virtual sensors denote the data flow from the producer to the consumer. The initial request is generated by the SPINE coordinator running on a PC. This request initiates the configuration of each of the four virtual sensors. Virtual sensors are created as described in Section III-B.1. Once all the virtual sensors have been initialized, the HMM virtual sensor starts producing output every sample. The sampling interval is defined for the Accelerometer VS, which acts as the data source for the whole system.

As described in Section IV-B, each Virtual Sensor can be implemented through one or more SPINE2 tasks, on the basis of its actual complexity. First of all, the presence of a Data Raw source, namely the accelerometer, requires the use of a SensingTask configured to acquire data from a real onboard sensor with proper settings (like the sampling rate). The Preprocessing VS, which performs preliminary computational functions can be translated into two ProcessingTasks sequentially interconnected with each of them in charge of carrying out a specific elementary operation. In a similar way, the Feature Extract VS can be decomposed into three further ProcessingTasks, individually devoted to compute a specific feature extraction function on the data coming from the previous preprocessing step. Since all these tasks require the same data, a Split data-routing task has to be used. As soon as all the features are computed, they have to be made available to the HMM ProcessingTask. This task, representing the actual implementation of the HMM VS, receives its input data from the Merge data-routing task, which is in charge of collecting the feature extraction results.

In the following subsections, we first describe some implementation details about GVS and other software modules implementing the same gait analysis functionality but developed in C and Matlab; we then analyze the obtained results and compare them with the results obtained with the C and Matab software modules.

A. Implementation

Fig. 9 shows the schemas of the GVS according to the higher level model and its translation based on SPINE2. In particular, as described in Section IV-B, each Virtual Sensor can be implemented using one or more SPINE2 tasks, on the basis of its actual complexity.

We started with a MATLAB model described in [23]. The code was simplified to match sensor node capabilities, e.g., using fixed-point math with limited precision. Based on this

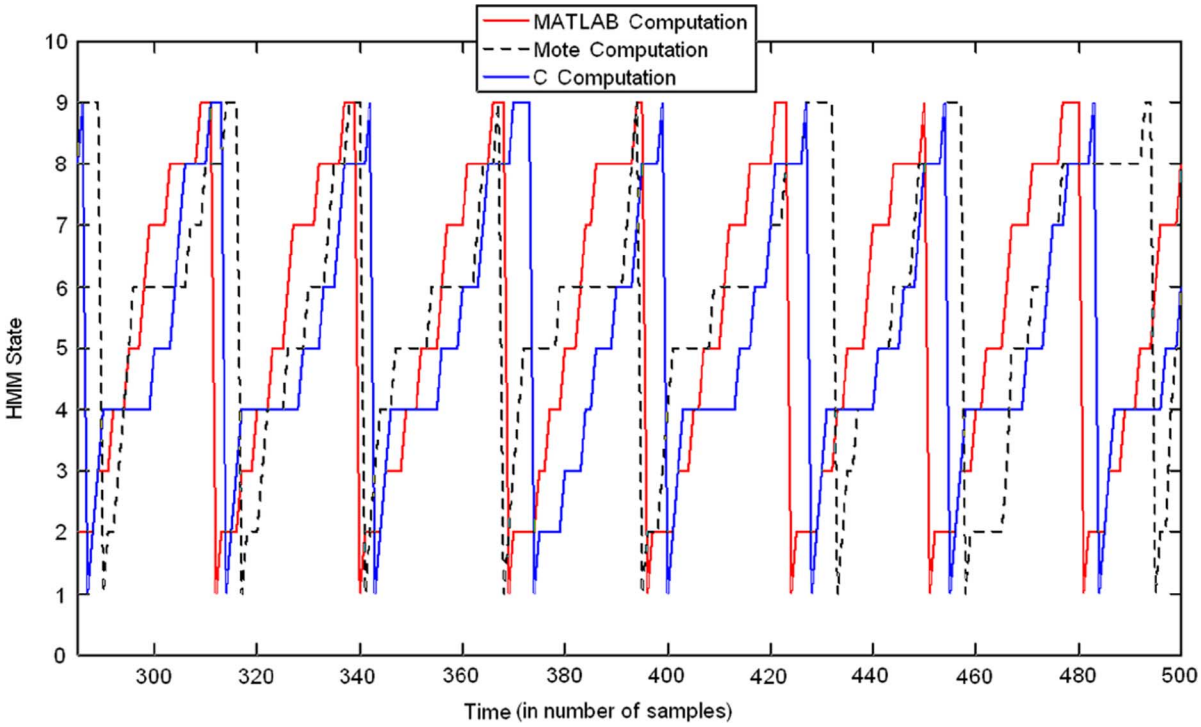

Fig. 10. Comparison of Matlab, C, and SPINE2 implementations of the Gait Virtual Sensor.

code, a version of the code was developed in C, and the outputs were compared. The model was retrained based on the updated code. After making changes to assure matching output between the C and MATLAB code, the C code was adapted to SPINE2 and ported to nesC for an ad hoc implementation on the sensor nodes. This allowed us to compare the effectiveness of using SPINE2 in the development of the virtual sensor with respect to the use of a lower level programming language like nesC.

The main challenge of adapting MATLAB models to SPINE and C implementation was the lack of hardware floating point multipliers in the microcontrollers of the wearable units, particularly MSP430 in our platform. Floating point operations are handled in software and are computationally expensive. Hence, we chose to approximate floating point operations in the original model with integer operations for our implementation. This technique appeared to be effective while from time to time generated inconsistencies due to the conversion error. Furthermore, when the range of floating point numbers were large, this technique would be less effective as it has to represent numbers with a large number of integer bits. To reduce the occurrence of these cases, we had to perform further optimization on the signal processing algorithm to reduce the ranges or reduce the effect of error due to conversion on the outcome of the algorithm.

Although the layered model of the defined virtual sensors allowed us to develop and verify one component at a time, one of the major limitation that we faced during the development was debugging. The development platforms available for TinyOS offer limited support for debugging. Due to this reason, the functionality testing and debugging was done using the C version. However, problems that we faced after porting the code on to the mote were much more serious and hard to debug. Problems like stack overflow due to excessive memory usage were extremely difficult to detect. We also faced issues due to high packet loss hindering the basestation-mote communication. Another major problem was the freezing of system due to higher processing overhead of one component over the others. This also caused interruptions in the communication. Another problem was the occurrence of "impossible" state transitions in the HMM. The root cause of the problem was determined as overflows. The HMM involved a large number of summations which periodically overflowed, leading to unpredictable results. Figuring out these problems required in-depth analysis of the code and careful optimizations and corrections on appropriate components. Reducing the memory footprint of the application was one of the major challenges that we faced during the whole development.

B. Analysis of Results

For our test case, we used this system to extract heel-down and heel-lift events from a walking subject. The sensor node contained a single tri-axial accelerometer sampling at 20 Hz (the highest sampling rate achievable while performing the processing steps). The event annotation was initially quite sensitive to sensor node misplacement, so we trained it with data from one subject with ten different trials. Each trial contained approximately 50 steps and had a slightly different sensor placement. This significantly increased the accuracy. The sensor node performed annotation and broadcast the raw samples so results between different implementations could be compared.

As can be seen in Fig. 10, the versions produce similar results. Most of the events were accurately predicted by our implementation barring the slight offsets occurring at times. There is a drastic difference observed between the sensor node output and the others on the far right side of Fig. 10. Also, the state

TABLE I
EVENT ANNOTATION RESULTS FOR DIFFERENT IMPLEMENTATIONS

Implementation	Stride Time (Smpl)	Stride Time (s)
MATLAB	28.21	1.41
C (PC)	28.04	1.40
SPINE2	28.04	1.40

transitions were not completely synchronized. For example, in the actual MATLAB implementation, state marked 6 is always momentary where as this does not happen in the other implementations. This is due to the fact that the model tries to catch up after a wrong state calculation. Consequently, the total time taken for a complete sequence of transitions remained almost the same for the whole experiment (on average, the difference is less than 10 ms). This time, which can also be defined as the time taken to reach the same state in the next cycle is termed as the stride time. Table I shows the average stride time measured for the three implementations.

VI. CONCLUSION

This paper describes a new approach to design of BSN applications that is based on the concept of virtual sensors and relies on the SPINE2 framework. The intuition behind the work is that in BSN applications there is need of new abstractions that capture the successive stages of the data processing and classification, including sensor data fusion. Virtual sensors allow abstracting even complex systems that behave like sensors, thus favoring code modularity and reuse. If a programmer needs to update an implementation of a virtual sensor or changing environmental conditions require that a different implementation strategy is adopted, it is sufficient to replace that portion of code without changing the rest of the system implementation. Virtual sensors also allow for the adoption of a simple coding style, where local sensors, remote sensors, or network abstractions can be treated in a uniform manner. The paper has presented an application of the concept of virtual sensors in the gait analysis domain aiming at enabling real-time activity and posture recognition. Results are twofold: (i) the exploitation of the SPINE2-based virtual sensor approach allowed to obtain a more rapid implementation of the gait system that by means of direct nesC programming; (ii) the efficiency and accuracy of the system is retained and is comparable with the C and Matlab implementations. More work is on-going to make the virtual sensor framework more powerful and flexible. In particular, over the air configuration is being added to allow reconfiguration of virtual sensors even at run-time. Moreover, other virtual sensors spanning from activity recognizers to human stress detectors are being developed.

ACKNOWLEDGMENT

This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2- 224053. This work is also supported by Telecom Italia through technical and financial contributions.

REFERENCES

[1] F. Lewis, "Wireless sensor networks," in *Smart Environments: Technologies, Protocols, and Applications*, D. J. Cook and S. K. Das, Eds. New York: Wiley, 2004, pp. 19–24.

[2] P. Levis *et al.*, "TinyOS: An operating system for sensor networks," *Ambient Intell.*, pp. 115–148, 2005.

[3] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. Murphy, G. Picco, and D. E. Informazione, "TinyLIME: Bridging mobile and sensor networks through middleware," in *Proc. 3rd IEEE Int. Conf. Pervasive Computing and Communications (PerCom)*, 2005, pp. 61–72.

[4] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *Proc. 32nd Int. Conf. Very Large Data Bases. VLDB Endowment*, 2006, pp. 1199–1202.

[5] A. Bakshi, V. Prasanna, J. Reich, and D. Larner, "The abstract task graph: A methodology for architecture-independent programming of networked sensor systems," in *Proc. Workshop on End-to-End Sense-Andrespond Systems (EESR), in Conjunction With MobiSys*, Jun. 5, 2005, pp. 19–24.

[6] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual sensors: Abstracting data from physical sensors," in *Proc. Int. Symp. World of Wireless, Mobile and Multimedia Networks*, 2006, pp. 587–592.

[7] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: Challenges and opportunities," *IEEE Pervasive Comput.*, vol. 3, no. 4, pp. 16–23, Oct. 2004.

[8] C. Lombriser, D. Roggen, M. Stäger, and G. Tröster, "Titan: A tiny task network for dynamically reconfigurable heterogeneous sensor networks," in *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, ser. Informatik aktuell. Berlin, Germany: Springer, 2007, pp. 127–138.

[9] F. Bellifemine, G. Fortino, R. Giannatonio, R. Gravina, A. Guerrieri, and M. Sgroi, "Spine: A domain-specific framework for rapid prototyping of wbsn applications," Software: Practice and Experience Sep. 7, 2010.

[10] R. Gravina, A. Guerrieri, G. Fortino, F. Bellifemine, R. Giannantonio, and M. Sgroi, "Development of body sensor network applications using spine," in *Proc. IEEE Int. Conf. Systems, Man Cybern.*, Oct. 2008, pp. 2810–2815.

[11] S. Iyengar, F. T. Bonda, R. Gravina, A. Guerrieri, G. Fortino, and A. Sangiovanni-Vincentelli, "A framework for creating healthcare monitoring applications using wireless body sensor networks," in *Proc. ICST 3rd Int. Conf. Body Area Networks, Ser. Bodynets'08*, 2008, pp. 8:1–8:2.

[12] M. Zhang and A. Sawchuk, "A customizable framework of body area sensor network for rehabilitation," in *Proc. 2nd Int. Symp. Appl. Sci. Biomed. Commun. Technol. ISABEL 2009*, Nov. 24–27, 2009, pp. 1–6.

[13] G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio, "Platformindependent development of collaborative wireless body sensor network applications: Spine2," in *Proc. IEEE Int. Conf. Syst., Man Cybern. SMC 2009*, Oct. 2009, pp. 3144–3150.

[14] P. Ciciriello, L. Mottola, and G. P. Picco, "Building virtual sensors and actuators over logical neighborhoods," in *Proc. Proc. MidSens'06: Int. Workshop Middleware for Sensor Netw.*, New York, NY, 2006, pp. 19–24, ACM.

[15] R. Tynan, G. O'Hare, M. J. O'Grady, and C. Muldoon, "Virtual sensor networks: An embedded agent approach," in *Proc. Int. Symp. Parallel and Distributed Process. Appl.*, 2008, pp. 926–932.

[16] A. Wood, L. Selavo, and J. Stankovic, "Senq: An embedded query system for streaming data in heterogeneous interactive wireless sensor networks," in *Distributed Computing in Sensor Systems*, S. Nikoletseas, B. Chlebus, D. Johnson, and B. Krishnamachari, Eds. Berlin/ Heidelberg, Germany: Springer, 2008, pp. 531–543, LNCS 5067.

[17] L. Mottola and G. P. Picco, "Programming wireless sensor networks with logical neighborhoods," presented at the 1st Int. Conf. Integr. Internet Ad Hoc and Sensor Networks, Ser. InterSense'06, New York, 2006, ACM.

[18] S. Lei, H. Xu, W. Xiaoling, Z. Lin, J. Cho, and S. Lee, "Vip bridge: Integrating several sensor networks into one virtual sensor network," Aug. 2006, pp. 2–2.

[19] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual sensor networks—A resource efficient approach for concurrent applications," in *Proc. 3rd Int. Conf. Inf. Technology: New Generations*, 2007, pp. 111–115.

[20] A. Augimeri, G. Fortino, M. Rege, V. Handziski, and A. Wolisz, "A cooperative approach for handshake detection based on body sensor networks," in *Proc. IEEE Int. Conf. Systems, Man Cybern. SMC 2010*, Oct. 2010, pp. 281–288.
[21] G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio, "Spine2: Developing bsn applications on heterogeneous sensor nodes," in *Proc. IEEE Int. Symp. Industrial Embedded Syst. SIES'09*, Jul. 2009, pp. 128–131.
[22] 2010, "The z-stack website," [Online]. Available: http://focus.ti.com/docs/toolsw/folders/print/z-stack.html
[23] E. Guenterberg, H. Ghasemezadeh, and R. Jafari, "A distributed hidden markov model for fine-grained annotation in body sensor networks," in *Proc. 6th Int. Workshop on Body Sensor Networks (BSN)*, 2009, pp. 339–344.

Nikhil Raveendranathan received the B.Tech. degree in electrical and electronics engineering from College of Engineering, Trivandrum affiliated to the University of Kerala, India, in 2006, and the M.S. degree in computer engineering from the University of Texas at Dallas in 2010.

He is currently a Software Developer at Research in Motion Ltd, Dallas, TX. His research interest is mainly on software architecture and optimization for Embedded Systems.

Stefano Galzarano received the B.S. and M.S. degrees both in computer engineering from the University of Calabria, Rende, Italy, in 2006 and 2009, respectively, where he is currently pursuing the Ph.D. degree in computer engineering.

His research interests are focused on high-level programming methods for wireless sensor networks.

Vitali Loseu received the B.S. and M.S. degrees both in computer science from the University of Texas at Dallas in 2007 and 2008, respectively, where he is currently pursuing the Ph.D. degree in computer engineering in the Embedded Systems and Signal Processing Lab (ESSP).

His research interests lie in system optimization for reconfigurable computing and biomedical data organization and mining.

Raffaele Gravina received the B.S. and M.S. degrees both in computer engineering from the University of Calabria, Rende, Italy, in 2004 and 2007, respectively, where he is currently pursuing the Ph.D. degree in computer engineering.

His research interests are focused on high-level programming methods for wireless sensor networks.

Roberta Giannantonio graduated in telecommunication engineering at Politecnico di Torino, Torino, Italy, with a master thesis about multicast in ad hoc wireless networks carried out at Trinity College in Dublin, Ireland.

Then, she joined Telecom Italia and since then she has been involved in research projects about Wireless Technologies. She has been active into the ZigBee Alliance and she has been visiting the WSN Lab under sponsorship of Telecom Italia, Berkeley, CA. She is now responsible for the SPINE open source project for distributed signal processing in Wireless Sensor Networks. She is coauthor of patents and papers about wireless technologies.

Marco Sgroi received the Laurea degree in electronic engineering (*summa cum laude)* from the Universita' di Roma La Sapienza, Rome, Italy, in 1994, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley in 1998 and 2002, respectively.

From 2003 to 2005 he was Researcher at DoCoMo Communications Laboratories Europe. From 2006 to 2009 he was Director of the WSN Lab sponsored by Pirelli and Telecom Italia in Berkeley. His main research interests are related to wireless sensors and embedded systems technologies. He is coauthor of over 30 technical papers and 3 patents.

Roozbeh Jafari received the B.Sc. degree in electrical engineering from Sharif University of Technology, Sharif, India, in 2000, the M.S. degree in electrical engineering from SUNY at Buffalo, and the M.S. and Ph.D. degrees in computer science from the University of California at Los Angeles, La Jolla, in 2002, 2004, and 2006 respectively.

He spent 2006-2007 in EECS department at UC Berkeley as a Postdoctoral Researcher. He is currently an Assistant Professor in Electrical Engineering at the University of Texas at Dallas. His research is primarily in the area of networked embedded system design and reconfigurable computing with emphasis on medical/biological applications, their signal processing and algorithm design. He is the director of ESSP Lab.

Giancarlo Fortino received the Ph.D. degree in computer engineering from the University of Calabria, Rende, Italy, in 2000.

He is an Associate Professor of Computer Science at the Department of Electronics, Computer Science and Systems of the University of Calabria, Cosenza, Italy. He has been a visiting researcher at the International Computer Science Institute (ICSI), Berkeley, CA, in 1997 and 1999. His research interests include distributed computing and networks, agent systems, agent oriented software engineering, wireless sensor networks, and streaming content distribution networks. He is author of more than 130 papers in international journals, conferences and books. He is also co-founder and president of SenSysCal S.r.l., a spin-off of University of Calabria, whose mission is the development of innovative systems and services based on wireless sensor networks for health care, energy management and structural health.

Dr. Fortino currently serves in the editorial board of the *Journal of Networks and Computer Applications* (Elsevier).