Enabling Effective Programming and Flexible Management of Efficient Body Sensor Network Applications

Giancarlo Fortino, *Member, IEEE*, Roberta Giannantonio, Raffaele Gravina, Philip Kuryloski, and Roozbeh Jafari, *Senior Member, IEEE*

***Abstract*—Wireless body sensor networks (BSNs) possess enormous potential for changing people's daily lives. They can enhance many human-centered application domains such as m-Health, sport and wellness, and human-centered applications that involve physical/virtual social interactions. However, there are still challenging issues that limit their wide diffusion in real life: primarily, the programming complexity of these systems, due to the lack of high-level software abstractions, and the hardware constraints of wearable devices. In contrast with low-level programming and general-purpose middleware, domain-specific frameworks are an emerging programming paradigm designed to fulfill the lack of suitable BSN programming support with proper abstraction layers. This paper analyzes the most important requirements for an effective BSN-specific software framework, enabling efficient signal-processing applications. Specifically, we present signal processing in node environment (SPINE), an open-source programming framework, designed to support rapid and flexible prototyping and management of BSN applications. We describe how SPINE efficiently addresses the identified requirements while providing performance analysis on the most common hardware/software sensor platforms. We also report a few high-impact BSN applications that have been entirely implemented using SPINE to demonstrate practical examples of its effectiveness and flexibility. This development experience has notably led to the definition of a SPINE-based design methodology for BSN applications. Finally, lessons learned from the development of such applications and from feedback received by the SPINE community are discussed.**

***Index Terms*—Design methods, human-centered applications, sensor programming frameworks, signal processing in node environment (SPINE), wireless body sensor networks (BSNs).**

Manuscript received November 3, 2011; revised April 15, 2012 and July 28, 2012; accepted August 8, 2012. Date of current version December 21, 2012. This work was supported in part by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with Contract FP7-2007-2-224053. This paper was recommended by Associate Editor L. Zhang of the former IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews (2011 Impact Factor: 2.009).

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

R. Giannantonio is with the Telecom Italia, 10015 Turin, Italy (e-mail: roberta.giannantonio@telecomitalia.it).

P. Kuryloski is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: philip.kuryloski@gmail.com).

R. Jafari is with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: rjafari@utdallas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCC.2012.2215852

I. Introduction

A WIRELESS body sensor network (BSN) is a collection of wearable (programmable) sensor nodes communicating with a local personal device. The sensor nodes have computation, storage, and wireless transmission capabilities, a limited energy source (i.e., battery), and different sensing capabilities depending on the physical transducer(s) they are equipped with. Common physiological sensing dimensions include body motion, skin temperature, heart rate, skin conductivity, and brain activity. The local personal device is typically a smartphone or a PC, and allows for real-time monitoring, as well as long-term remote storage and off-line analysis. Very diverse application scenarios are enabled by BSN technologies, although m-Health applications are probably the most emblematic example. BSN systems can be used to directly monitor several vital signs continuously and noninvasively, as tiny wireless sensors are placed on the skin and sometimes in the garments. These signals can, in turn, infer many diseases (e.g., cardiovascular or neurodegenerative) at an early stage. Furthermore, BSNs are great enablers for many other application domains such as e-Sport, e-Fitness, and e-Wellness, where the objective is not specifically related to disease detection and/or monitoring, but rather to help people maintain physical and mental wellness. E-Factory is also an emerging scenario in which BSNs have a central relevance; these applications aim to monitor employees activities, such as in production chains, to both help ensure safety, and to guide proper assembly of the product. BSNs can play an important role also for social physical/virtual interactions as they could monitor emotional states of people while they meet, and enable certain services depending on (mutual) emotion reactions.

However, although several BSN-based research prototypes have been proposed to date [1], [2], almost none of them have reached the market at this time. One of the main limiting factors is related to the programming complexity of these systems. Implementing real-time power-efficient distributed signal-processing algorithms on wireless nodes that are very resource limited and have to meet stringent requirements in terms of wearability (including battery duration) remains extremely challenging and complex. Such algorithms are the basis for the end-user applications of these devices. Yet, the software abstractions provided natively by the current sensor node operating systems and development environments suffer a lack of predefined, customizable, and easy-to-use sensing, signal processing, communication, and storage functionalities.

Consequently, BSN developers must devote significant development time to what would be considered low-level programming details, rather than focus on new and unique core application functionalities and features. To date, almost all the BSN applications have been developed following two main approaches: low-level programming and general-purpose middleware.

The most common approach consists of developing prototype applications on BSN nodes as a monolithic block intertwining low-level services, reusable components, and application-specific logic [10]–[17]. As a result, the developed software is poorly reusable and difficult to modify. Moreover, the risk of implementation errors is significant, and debugging can be a very time-consuming process. Depending on individual developer skill, the main advantage is represented by manual code optimization and efficiency.

The second approach is based on general-purpose frameworks for wireless sensor networks. Such a framework is a software layer consisting of a set of services implemented across a network. It hides the complexity of low system and network layers and provides proper abstractions and interfaces to the upper layers. Application developers can then focus on application logic without dealing with the implementation details of the underlying services, and development time is generally shortened. However, current general-purpose frameworks for WSNs such as Agilla [3], DFuse [4], Milan [5], TAG [6], or Mires [7] are designed to support a wide variety of application scenarios. As a result, they are either too general (lacking abstractions typically needed for BSN systems) or demand too many resources to realize efficient BSN-specific applications.

An emerging research topic is devoted to the design of novel BSN programming approaches based on the concept of domain-specific frameworks [8] which represent a concrete application of the domain-specific modeling [9]. A domain-specific framework is a collection of cooperating software entities that together define a generic or template solution to a family of domain-specific application requirements. Since such frameworks are tailored around a well-defined domain, they enable faster prototyping times and help realizing more effective and efficient applications in the target domain. This paper discusses, in detail, the most important requirements for an effective domain-specific framework for BSNs. It then presents the signal processing in node environment (SPINE) framework, a domain-specific framework able to support rapid development of efficient BSN applications, and describes how SPINE addresses these requirements.

The remainder of this paper is organized as follows. Section II discusses the requirements for a domain-specific framework for BSN applications. Section III overviews some related work. Section IV presents the SPINE Framework, analyzing its distinctive features from different perspectives (architecture, high-level data processing, and multiplatform support). Section V focuses on thoroughly evaluating the SPINE performance atop the most used hardware/software sensor platforms and also with respect to well-known correlated frameworks. Section VI briefly introduces some enabling BSN applications that have been developed through SPINE to show the SPINE effectiveness in supporting BSN systems. In Section VII, we propose a design method based on SPINE and derived from the well-known platform-based design (PBD) methodology for the development of SPINE-based BSN systems. Section VIII discusses the lesson learned with SPINE, summarizing interesting feedback received from the SPINE community. Finally, some conclusive remarks are drawn and future works delineated.

TABLE I
COMMON TASKS OF BSN APPLICATIONS

TASK	DESCRIPTION
SENSOR SAMPLING	The sensor sampling process represents the first step for developing a BSN application. Selecting the appropriate sampling time to satisfy the application requirements is important, as it determines the amount of raw data generated and processed (and to a certain degree, energy consumed). The proper execution of the application may depend on this parameter; often, a minimum sampling time is required to allow a sensor to accurately capture a particular phenomena.
IN-NODE DATA PROCESSING	Classifier algorithms very rarely use raw data. Instead, attributes (or features) are extracted on sample data windows and used to detect events and classify activities. Extracting features directly on the wireless nodes allows for reduction of radio usage, as resulting summary data are sent instead of raw data values.
SENSOR CONFIGURATION AT RUN-TIME	Support for runtime configuration (enabling, disabling, setting the sampling rate) on the available sensors of a node is often very useful. Application requirements can change over time; for instance, under certain circumstances, a sensor may be sampled at a lower rate, or its data not needed at all. Therefore, supporting runtime sensor configuration allows dynamic application behavior.
NODE SYNCHRO-NIZATION	Many BSN-oriented signal-processing algorithms require sensors on multiple nodes to be sampled in unified time intervals, to ensure consensus of time in observing underlying events. Nodes are often kept synchronized to in turn allow synchronized sampling of sensors and joint processing data at the coordinator.
DUTY-CYCLING	Duty cycling is a mechanism for controlling radio power, to reduce power consumption of a sensor node, thus increasing its battery lifetime. Radio duty cycling must be tuned very carefully in order to minimize energy use, but allow sufficient transmission of data.
APPLICATION LEVEL COMMUNICATION PROTOCOL	A specific application level communication protocol is needed to support the interaction among sensor nodes (if needed) and between sensor nodes and the coordinator. The communication involves sensor node discovery and service advertisement, requests for sensing and signal processing, raw and preprocessed sensor data transmission, and event delivery.
HIGH-LEVEL PROCESSING	Often, the end-user BSN applications do much more than plotting sensor data into graphs. They require the interpretation of asynchronous events and periodic data coming from sensors in high-level knowledge. This implies decision support (classification) algorithms that extract meaningful information from such events and data.

II. REQUIREMENTS FOR A DOMAIN-SPECIFIC BODY SENSOR NETWORK FRAMEWORK

Typically, BSN applications share several common tasks on top of which specific application components can be developed at both sensor-node side and coordinator side. Table I describes such common tasks that we have identified by examining in depth the state of the art of BSN applications: sensor sampling, in-node data (pre)processing, sensor (re)configuration at run-time, node synchronization, duty-cycling mechanisms, application-level communication protocols, and high-level processing at the coordinator side.

Thus, a framework for the development of BSN applications should provide suitable programming abstractions and tools to effectively and efficiently support the identified common tasks. Moreover, a software framework designed to support fast prototyping of efficient BSN applications should meet

TABLE II
REQUIREMENTS FOR BSN FRAMEWORKS

REQUIREMENT	HIGH-LEVEL TECHNIQUES
Programming Effectiveness	Programming Abstractions, Software Engineering Methods, Debugging and Testing Tools
System Efficiency	Resource Management Optimization
System Interoperability	Application-Level Communication Protocol and Adapter for Heterogeneous Sensor Inclusion
System Usability	GUI-based flexible management of the BSN system, PC and smartphone-based Coordinator
Privacy Support	Data encryption and authentication

specific (functional and nonfunctional) requirements in terms of effectiveness, efficiency, and usability. In particular, such a BSN-oriented framework should facilitate the development of well-structured and resource-efficient applications with less effort in terms of development time and application programming complexity. Resulting developed code should be more modular and easier to maintain, and with usable tools for sensor-node-side and coordinator-side application management. A system interoperability requirement is desirable, which allows the integrated use of heterogeneous sensor platforms in the same BSN application. Finally, privacy is a very important nonfunctional requirement in the context of BSN applications, as they usually involve processing and communication of data acquired from the human body. Such data are inherently personally identifiable and may be medically relevant and, therefore, highly privacy sensitive. In the following, such requirements are extensively discussed. Table II summarizes the main identified requirements for a BSN-oriented application development framework along with key techniques/mechanisms, which the framework should incorporate, capable of fulfilling them.

Programming Effectiveness refers to the ability of the software framework to provide effective and specific support for the programming, debugging, and testing of BSNs applications. It is enabled by suitable programming abstractions, software engineering methods, and debugging and testing tools. In particular:

1) *Programming abstractions* refer to development paradigms, programming interfaces, and built-in functionalities that provide easier access to the platform physical resources (e.g., sensing, storage, and communication), and higher level functionalities that help developers focus on core application aspects. In particular, the following BSN programming abstractions should be made available: a) tunable sensor drivers, as it is often necessary to adjust (sometimes during run-time) the sampling rate, sensitivity, and range, or to enable a subset of channels of a multichannel sensor (e.g., only some of the axes of a three-axial accelerometer or gyroscope); b) flexible data structures to easily accommodate different data types (e.g., *short, int*, or *long* values) so that, for instance, the same buffer might be configured for storing data from sensors that produce samples with different word-length; c) flexible communication application programming interface (API), as different applications may require different data payload structure and length; d) parameterized processing functions, to set the inputs of the functions without hard-coding their values (e.g., to allow run-time configuration of feature extractions on variable signal windows). Finally, built-in tunable power management schemes allow for customized tradeoff between performance, reliability, and system lifetime. They are often intended to improve the lifetime of the sensor node, and allow, for instance, the radio duty cycling (that may drastically reduce energy consumption), reducing the sensor/s sampling and processing (which typically reduce energy consumption at the cost of lower performance), or disabling data transmission over-the-air and enabling local storage.
2) *Software engineering* methods aim to support rapid prototyping of BSN applications through the use of component-based approaches. They include properties such as robust isolation among software modules that enhances code reusability, and testing of individual modules. In particular, the availability of predefined (ready to use) software components that are common to most of the BSN applications, along with well-defined techniques through which assemble them, is critical to obtain prototypes in a short period of time. The main components often used in BSN applications are signal filters (e.g., finite impulse response filters) to clean or amplify a signal, feature extractors (e.g., average, variance, zero crossing, and signal slope) to reduce the amount of data to be transmitted, classification algorithms (e.g., K-NN, decision trees) useful as decision support tools, and an application-level communication protocol (that includes, e.g., nodes/services discovery, failure notification, and user data transmission).
3) *Debugging and Testing tools* allow for compile and run-time assessment of the functional correctness of the BSN application under-development. They are both extremely useful to help developers find errors that can be revealed only during run-time. Debugger tools are useful to locate the cause of a known erroneous application behavior, while testing tools are used to help verify the correctness of software components, as well as find specific situations in which the program crash or executes unexpectedly. Such tools may be offered by the development environment and can be based on simulators or step-by-step debuggers that are able to track the state of the application at each instruction.

System Efficiency refers the quality with which energy, memory, and computational resources in the system are managed, particularly with respect to the resource limited wearable sensor nodes. It is important to optimize the code footprint (i.e., reduced code segment memory needs) and reduce RAM usage for sensor node binaries, as common BSN nodes use a microcontroller as their CPU. Thus, optimizing the signal-processing algorithms that run on the sensor node is essential.

System Interoperability refers to the ability of using and making collaborate devices based on different hardware/software technologies. It includes the possibility of communication between different nodes of the same software platform (e.g., TelosB and Micaz motes on TinyOS), the ability of the

system to allow heterogeneous network formation of nodes that are programmable using the same language, the interoperability among homogeneous BSN coordinators, and, finally, the ability of the system to inter-operate with heterogeneous networks (e.g., Internet through sockets or XML RPC). It can be enabled by an application-level communication protocol and communication adapters for heterogeneous sensor inclusion that jointly allow the simultaneous use of devices based heterogeneous hardware/software technologies.

System Usability indicates the property of a system to be user-friendly for both end-users, and developers/designers. It is typically supported by GUI-based flexible management of the BSN along with a coordinator running either on PC or smartphone so that the BSN can be (re)configured remotely without manually coding the instructions, but through intuitive graphical interfaces.

Privacy Support refers to the ability of a system to protect a user's confidential information (such as physiological signals). Encryption and authentication mechanisms allow the system to maintain the secrecy of such information and, in turn, ensure that it is released only to authorized entities. Privacy protection is critical in real-life scenarios and can only be achieved when supported by all sources and channels of privacy sensitive information. As the source of privacy sensitive information, BSNs should adopt strong privacy protection features and controls.

III. Related Work

A. e-Health Body Sensor Network Systems

BSNs are receiving great interest from both academy and industry as a novel technology for improving health-care systems and, as a consequence, quality of life. Practical e-health scenarios where BSNs could be effective include physical activity recognition, physical rehabilitation, cardiac and respiratory diseases prevention and early detection, emotion recognition, remote elderly assistance and monitoring, sleep quality monitoring and sleep apnea detection, gait analysis, and Parkinson's symptoms. A wide range of such application prototypes has been proposed, although most of them have not hit the market yet. Here, we present some of the most representative and pioneering research efforts and prototypes.

An innovative physical activity monitoring system is presented in [10]. The system is based on the eWatch, a multisensor platform that can be worn in several body positions (such as at the wrist, ankle, waist, and trousers pocket). Multiple activities (sitting, standing, walking, ascending and descending stairs, and running) are recognized in real time and stored into the device for later analysis. The in-node classifier algorithm is a decision tree fed with time-domain features extracted online from the raw readings of a two-axis accelerometer and a light sensor. Other projects [11]–[13] aim to recognize more complex activities, including movements (such as drinking, brushing the teeth, and writing), and hand or facial gestures, but they combine data from multiple sensor nodes placed in different body positions rather than using a single multisensor unit like the eWatch.

MyHeart [14] is a system for cardiovascular diseases prevention and detection. The sensors to detect the electrocardiogram (ECG) signal are embedded into garments and an on-body processing unit is in charge to perform signal processing to enhance the ECG signal and reduce artifacts introduced by body motion. A single-axis accelerometer placed into the on-body processing unit is used to perform a real-time activity classification, whose results are taken into account for the correct interpretation of the ECG variations. The system also features a Bluetooth interface to a Java-powered mobile phone where the end-user application is running. The application allows the user to interact with the BSN and, when Internet connectivity is detected, forwards the acquired signals to a remote monitoring center consulted by professional care-givers.

Emotion recognition can be of interest, for instance, in human–computer interaction systems, and as a support to specialists and patients affected by autism. In [15], a study on using BSN technology to recognize human emotions has been presented. The system uses the BodyMedia SenseWear Arm-band to extract physiological signals needed by the classification algorithm to detect sadness, anger, surprise, fear, frustration, and amusement. In particular, galvanic skin response (GSR), heart rate, and skin temperature are involved in the recognition task. Other projects process only the GSR signal (however, high accuracy is reached only to detect arousal) or use facial expression to derive emotions [16], [17], although video cameras or EMG electrodes placed on the face are required in this case. In [18], a pilot study for monitoring Parkinson's symptoms and motor complications using wearable sensors is described. Eight wireless accelerometer sensors are placed on the left and right arms and legs. In-node feature extraction is performed on the acceleration signals to evaluate the severity of Parkinson's symptoms, by capturing movements associated with tremor, bradykinesia, and dyskinesia; such estimations could be used to facilitate the administration of medications. The authors stress on the importance of defining such methods and algorithms that are feasible to be implemented on wearable nodes, to improve system lifetime and enable long-term patients monitoring.

All the described systems have been implemented using the application-specific approach, which leads issues such as complex and time-consuming maintenance and upgrade, as the code produced tends to be less reusable. Furthermore, developers had to implement from scratch functionalities that would have been offered by a domain-specific framework such as SPINE.

B. Frameworks Supporting Body Sensor Network Application Programming

The literature on domain-specific frameworks for BSN programming is rather small. Indeed, just a few frameworks have been proposed to date. This is partially due to the lack of standardization for BSN sensor node hardware and software platforms, due to the fact that the BSN-based e-health domain is still quite novel, and due to the application-driven programming approach taken by many BSN developers that do not stress the need for higher level software abstraction and toolkits.

One of the most relevant and, probably, the first attempt to define a general platform able to support various BSN applications is CodeBlue [19]. Originally, it was designed to address a wide range of medical scenarios, such as monitoring patients in

hospitals or victims of a disaster scene, where both patients/victims and doctors/rescuers may move and not necessarily be in direct radio range all the time. CodeBlue consists of a set of hardware wearable medical sensor nodes (pulse oximeter, ECG, EMG, accelerometer, and gyroscope) based on the TelosB and MicaZ motes and on a software framework running atop the TinyOS operating system [20] specifically designed for integrating wireless medical sensor nodes and other devices, such as PDAs and PCs. The CodeBlue framework allows these devices to discover each other, report events, and establish communications. It includes a naming scheme, a multihop communication protocol, authentication and encryption capabilities, location tracking and in-network filtering, and aggregation.

A more recent example is RehabSPOT [21], a customizable wireless networked body sensor platform for physical rehabilitation. RehabSPOT is built on top of SunSPOT technology [22] from Oracle (formerly Sun Microsystems). The fundamental idea behind RehabSPOT is that instead of downloading different programs onto different sensor nodes, RehabSPOT-based BSNs run a uniform program on all wearable nodes, although they may perform different sensing and/or signal-processing functions during run-time. The system software is based on a client–server architecture. The server program is installed and running on a PC, while the client program is installed in the remote nodes. The communication between client and server programs follows the message passing distributed computing paradigm by leveraging the computation power embedded inside the remote nodes. A lightweight protocol for device discovery at both remote nodes and base station to support dynamic BSN construction has been introduced.

SPINE improves the current state of the art, fulfilling the main limitations of the aforementioned frameworks. CodeBlue, for instance, does not provide for complex and customizable processing functionalities. RehabSPOT, instead, is available only for the SunSPOT platform and, hence, does not guarantee sufficient system heterogeneity.

In addition to the related work presented previously, a few general-purpose middlewares for WSNs have been customized to develop BSN-based health-care applications. Specifically, Titan [23] and MAPS [24] have been used to develop a real-time physical activity recognition prototype based on body-worn motion sensors. Titan supports implementation and execution of context recognition algorithms in dynamic WSN environments by representing data processing by a data flow from sensors to recognition results. The data are processed by tasks that are interconnected to define a task network. It has been designed to run on resource constrained sensor nodes and implemented in TinyOS on Tmote Sky motes. MAPS is a novel Java-based framework for wireless sensor networks based on SunSPOT technology which enables agent-oriented programming of WSN applications. MAPS has been appositely defined for resource-constrained sensor nodes. It adopts component-based lightweight agent server architecture to avoid heavy concurrency models as well as lightweight architecture for efficient agent execution, and migration. It also features minimal core services involving agent migration, sensor resource capability access (actuators, input signalers, flash memory, and battery), agent naming, agent communication, and timing.

TABLE III
COMPARISON AMONG DIFFERENT DOMAIN-SPECIFIC AND GENERAL-PURPOSE FRAMEWORKS

	CodeBlue	RehabSPOT	MAPS	TITAN	SPINE
Programming Effectiveness					
Programming Abstractions	partial support	partial support	✓	✓	✓
Software Engineering Methods		partial support	partial support	partial support	✓
Debugging and Testing Tools		✓	✓		✓
System Efficiency					
Resource Management Optimization	✓			✓	✓
System Interoperability					
Application-Level Communication Protocol	✓	✓	✓	partial support	✓
Adapter for Heterogeneous Sensor Inclusion					✓
System Usability					
GUI-based flexible management of the BSN system	partial support	partial support			✓
PC and smartphone-based Coordinator	partial support	✓	partial support	✓	✓
Privacy Support					
Data encryption and authentication		✓		✓	✓

Table III summarizes the comparison of the domain-specific frameworks described above against SPINE on the basis of the identified requirements and high-level techniques for an effective BSN programming framework.

IV. SIGNAL PROCESSING IN NODE ENVIRONMENT FRAMEWORK

SPINE [25], [26] is an open-source software framework [27] for fast prototyping of applications based on BSNs. It has been designed around the requirements defined in Section II to maximize its effectiveness for the development of applications in the BSN domain. In particular, SPINE provides support for distributed signal-processing intensive BSN applications by a wide set of predefined physiological sensors, in-node and on-coordinator signal-processing utilities, flexible data transmission, and optimized network/resource management. SPINE has a powerful and well-designed modular structure that allows easy integration of new custom-designed sensor drivers and processing functions, as well as flexible tailoring and customization of its built-in features as developers deem necessary. One fundamental idea behind SPINE is the reuse of software components to allow different end-user applications to configure sensor nodes at run-time based on the application-specific requirements so that the same embedded code can be used for several applications without reprogramming. SPINE natively supports logical star-topology sensor networks, where the edges are represented by the wearable sensor nodes, and the center is a smart coordinator station. As SPINE uses an application-level protocol, it may rely on a network layer which supports multihop so that the physical network can actually involve communication between the coordinator and nodes that are more than one hop away.

The remainder of this section is organized as follows: Section IV-A describes in detail the software architecture of

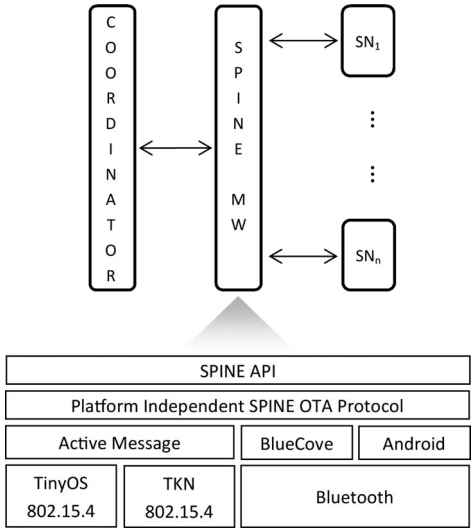

Fig. 1. SPINE middleware architecture.

the SPINE framework, Section IV-B focuses on one of its most recent and promising components that is the high-level data-processing module, while Section IV-C describes the great degree of heterogeneity of SPINE in terms of supported sensor node and coordinator device platforms.

A. Architecture

Fig. 1 depicts a general overview of the SPINE architecture. The SPINE middleware (MW) spans to both the coordinator device and the wearable sensor nodes forming the BSN. Both at coordinator and sensor node, SPINE provides an API to manage the BSN application which relies on a platform-independent communication protocol layer that we have specifically designed for SPINE. This abstraction layer is implemented by different platform-dependent communication adapters (supporting IEEE 802.15.4 and Bluetooth) which are dynamically loaded at the coordinator which are linked at compile time at sensor node level.

Figs. 2 and 3 depict, respectively, the two main components of SPINE: SPINE Node(s) and SPINE Coordinator. The former is implemented in the sensor platform-specific embedded programming language and runs on each sensor node forming the BSN; the latter is implemented in Java and runs on the coordinator device.

The *SPINE Node* is organized in four interacting components.

1) The "Sensor Node Manager" is responsible for the general interactions among the sensing management, signal processing, and communication modules, and dispatches requests coming from the remote coordinator to the appropriate block.
2) The "Communication" block handles reception and transmission of messages over-the-air, and manages radio duty cycling. It is composed by inbound packets decoders (i.e., service discovery, setup sensor request, set up function request, function (de)activation request, and start and reset computation requests) and outbound packets encoders (service advertisement, acknowledgment packet, sensor readings, and processed data message). Each received or sent message is initially handled by the radio controller which provides a common interface regardless on the specific radio chip adapter.
3) The "Sensing Management" block consists of the sensorboard controller component which acts as a general interface to the physical sensors of the platform, setting up timers when periodical sensing is requested by the remote coordinator, or simply performing one-shot reading to the requested sensors. The sensor-board controller manages, independently from the specific hardware specifications, all the supported sensor drivers (currently three-axis accelerometer, two-axis gyroscope, 4-leads ECG, GSR, respiration rate, environmental temperature, light, and humidity) through a list of sensor interfaces. This design approach guarantees high modularity and efficient

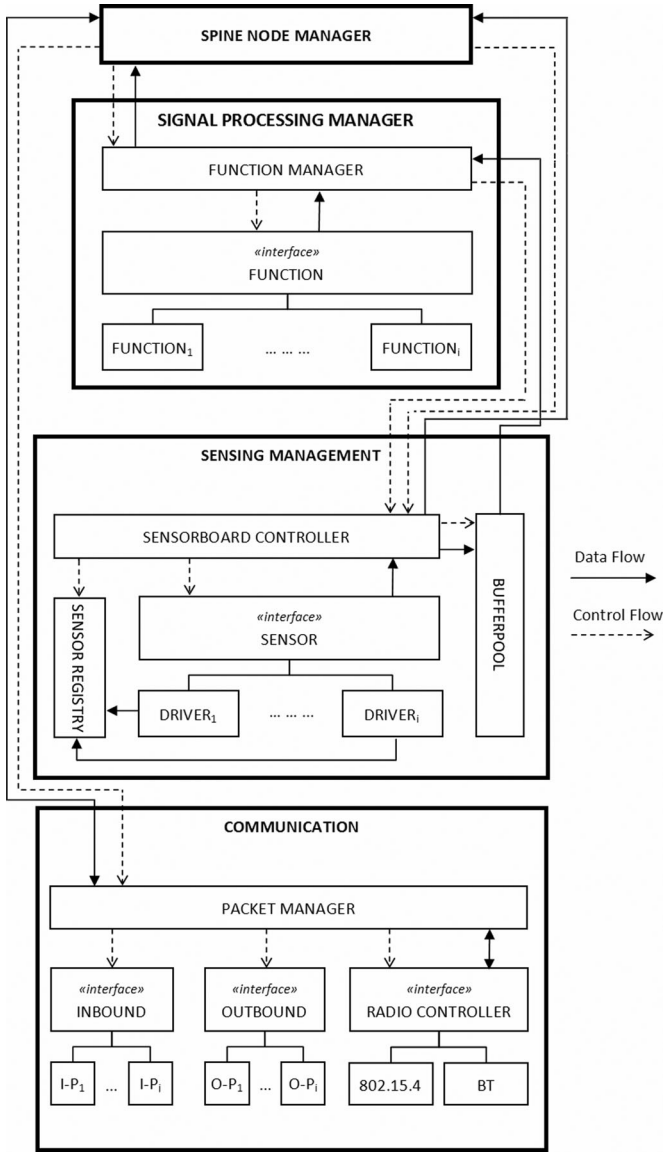

Fig. 2. SPINE node software architecture.

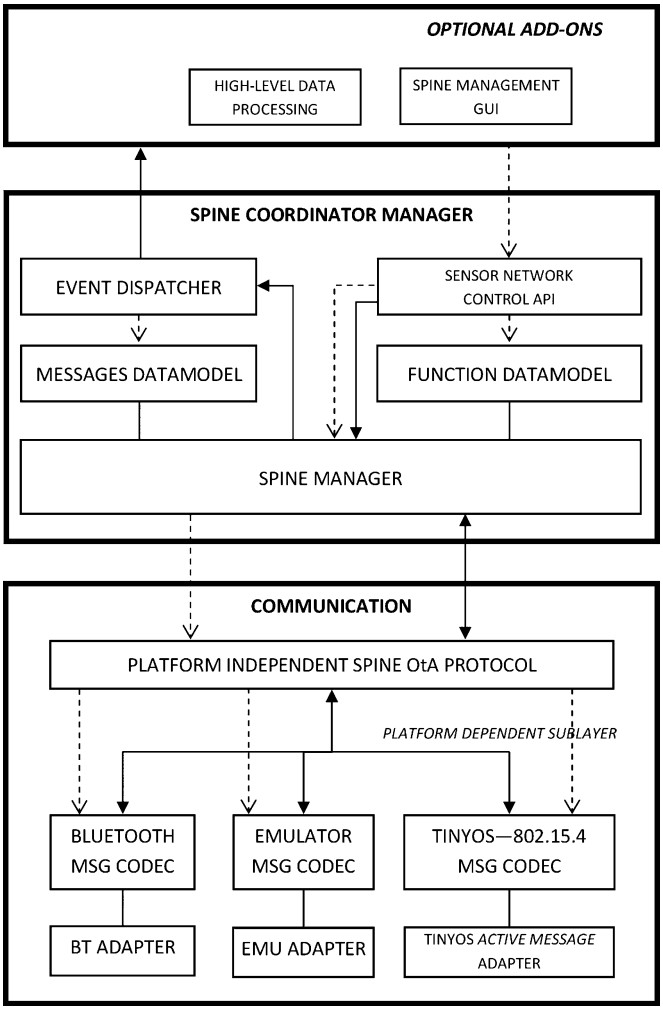

Fig. 3. SPINE coordinator software architecture.

customization to properly support the hardware sensing capabilities of the target platform. The sensor-board controller uses the BufferPool component to store the sensor readings so that they become available to the signal-processing block. Specifically, the BufferPool consists of a set of circular buffers and provides two mechanisms to access the sensor data: 1) upon requests using getter functions, and 2) using listeners that are notified when new data from sensor/s of interest are available. The sensing management block, finally, contains a sensor registry where each compiled sensor driver self-registers at boot-time, to allow other components to retrieve the available sensor list.

4) The "signal-processing" block uses the function manager to handle a flexible, customizable, and expansible set of signal-processing functionalities such as math aggregators (currently, max, min, amplitude, mean, variance, standard deviation, entropy, and signal energy), filters, and threshold-based alarms that can be applied to any sensor data streams. Again, to simplify customization and extension with custom-developed functionalities, the function manager engine uses an efficient design approach, based on a list of function interfaces which abstract any type of processing tasks. The signal-processing block retrieves the data to be processed from the BufferPool and, by interacting with the sensor node manager and the packet manager, it reports the results over-the-air back to the coordinator.

TABLE IV
API EXPOSED BY SPINE AT THE COORDINATOR STATION

Functionality	Description
discoveryBsn	Allows for the discovery of nodes and their supported sensing and processing capabilities.
setupSensor	For each node, allows independent specification of sampling rates for multiple sensors.
setupFunction	For each node, allows for preliminary configuration of processing functionalities that will be eventually activated.
activateFunction	For each node, enables one or multiple in-node (periodic or trigger-based) signal processing functionalities independently. The specific processing functions may be activated over-the-air via additional parameters.
startBsn	Issues a broadcast message to the BSN to command a synchronized start of the sensing and processing functionalities that have been already setup.
resetBsn	Issues a broadcast message to the BSN to command a synchronized reset to the initial idle state.
Event	**Description**
newNodeDiscovered	Invoked by the SPINE Manager to its registered listeners when a new BSN node is discovered
discoveryCompleted	Invoked by the SPINE Manager to its registered listeners at the end of the BSN discovery procedure.
dataReceived	Invoked by the SPINE Manager to its registered listeners when new user data sent from a specified node are received.
serviceMessageReceived	Invoked by the SPINE Manager to its registered listeners when a service message (e.g. warning or error notifications) is received from a specified node.

The *SPINE Coordinator* is organized in the two macrocomponents.

1) The "communication" block has similar functionalities of the corresponding block in the sensor node and has been designed to load the proper radio module adapter (i.e., for TinyOS 802.15.4 motes, Bluetooth devices, and for emulated SPINE nodes) dynamically. This component has the important function of abstracting the logical interactions between the coordinator and the BSN from the low-level transmissions that depend on the actual platform technology being used. This is obtained by decoupling the communication service interface from its platform-dependent implementation layer.
2) The "SPINE coordinator manager" block represents the most superficial layer and is the only one the SPINE applications will be based on. It includes the sensor network control API and the event dispatcher. The former is an interface exposed to the end-user application developers to manage the underlying BSN (e.g., to activate sensor sampling and on-node signal processing to certain nodes). Table IV summarizes the most relevant functionalities provided by the control API. The latter forwards events such as new nodes discovered, alarm or user data messages

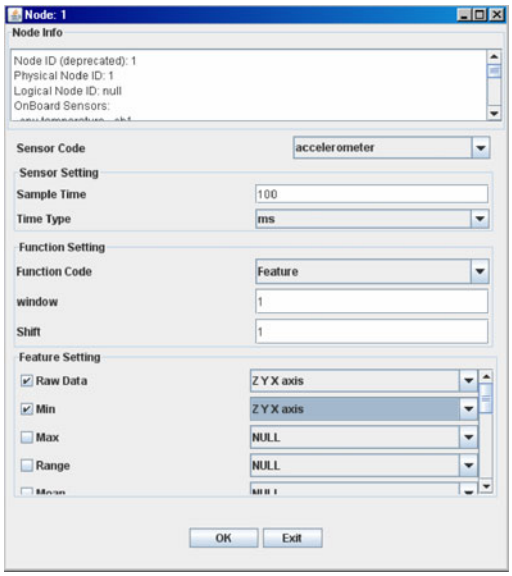

Fig. 4. Java desktop implementation of the SPINE management GUI (sensor-node configuration dialog window).

to the registered listeners implemented by the SPINE applications.

In addition to the core components, SPINE currently provides two "optional add-ons" modules.

1) The "high-level data processing" enables signal processing and pattern recognition on the coordinator node. Using the SPINE distributed computing architecture, this important module supports the design and implementation of new applications by providing highly generalized interfaces for data preprocessing, feature extraction and selection, signal processing, and pattern classification. It is designed to simplify the integration of SPINE in signal processing or data mining environments, providing functionality such as automatic network configuration, aggregate data collection, and graphical configuration. It also includes a bridge to the popular WEKA [28] data mining toolkit to allow the use of its feature selection and pattern classification algorithms from within SPINE.
2) The "SPINE management GUI" is a graphical add-on tool that allows configuration of remote sensor nodes using a user-friendly interface (rather than by programming). It also contains a simple textual logging function for events generated by remote nodes and received by the underlying SPINE coordinator (e.g., discovery advertisement packets and data messages). Snapshots of the desktop and Android implementation of this GUI are shown in Figs. 4 and 5, respectively.

From a programming perspective, SPINE provides abstraction layers for the node discovery, sensing, signal processing, and data transmission over-the-air. Apart from providing built-in support for given sensors and processing, SPINE has been designed such that it is very straightforward to add software support for new, custom-defined sensor drivers, and developers can easily interact with the sensor nodes through a simple Java API. The main functionalities exposed by this API are

Fig. 5. Android implementation of the SPINE management GUI (sensor and function configuration dialog windows).

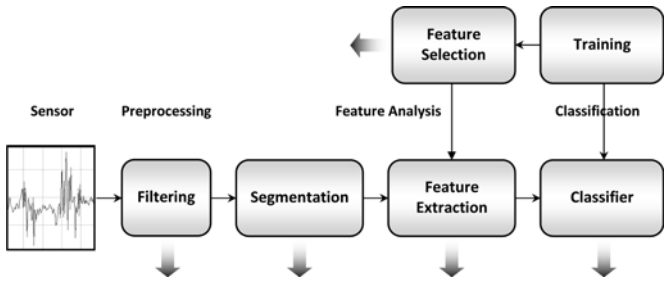

Fig. 6. Data-processing chain supported by the SPINE high-level data-processing module.

summarized in Table IV. In the following sections, the high-level processing and the multiplatform support enabled by SPINE are described in more detail.

B. High-Level Data Processing

This module has been organized as an optional SPINE plug-in and represents a powerful extension to the core framework as it provides more complex signal processing and decision support functionalities (e.g., pattern recognition, classification, etc.) that are intended to be performed at the coordinator. It provides signal processing and pattern recognition (or data mining) with flexible and reusable Java code. It is designed to simplify the integration of SPINE in signal processing and pattern recognition environments providing more application-oriented functions such as automatic network configuration, aggregate data collection, and graphical configuration. The module provides complete support during all the steps, from sensor data acquisition up to classification, as shown in Fig. 6.

Fig. 7 depicts a schematic layered view of the high-level data-processing module. The lower layers represent the WSN and the way the module communicates with it (i.e., through SPINE). On top of them, a set of converters manipulate data coming from SPINE to obtain a more powerful and manageable representation. In particular, the module is based on the concept of *Datasets* and *Signals*, which are objects defined in the data layer. This enables signal processors and data mining tools to transparently work on data coming from the sensor nodes. One

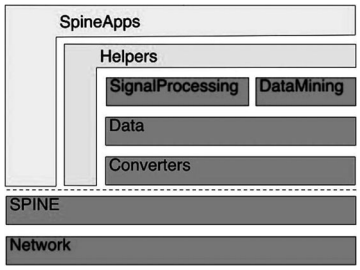

Fig. 7. High-level data-processing layered software architecture.

of the key features of this module is the capability of converting data in various formats such as the comma-separated values or the attribute-relation file format document format (used by Weka). Finally, a collection of helper classes has been defined to speed up the implementation of the most common tasks of SPINE applications. These helpers have been designed to reduce as much as possible the amount of code needed to configure SPINE networks for specific uses. They aggregate data from sensors that can be conveniently rearranged in any of the structures aforementioned.

Each provides a number of default implementations the developer may choose. In addition, a modular architecture allows for easy integration of further custom-defined components.

Sensor data are retrieved using the core functionality of SPINE, and then converted into more convenient data structures that reflect the concepts of signals and datasets. Optionally, developers may apply filtering and segmentation to the collected signals. Feature Extraction algorithms have also been provided, as they become useful when the application developers do not choose to perform in-node feature extraction. Furthermore, several feature selection techniques are available to identify optimal subsets of the extracted features that are later used for the classification phase. The classification phase is widely supported, also allowing for classifier algorithm training. A few algorithms have been implemented, and, if needed, the developer may easily integrate further classifiers. Moreover, our module is also integrated with the WEKA data mining toolkit. This brings great advantages to SPINE as developing machine learning algorithms is extremely time consuming. The choice of WEKA is motivated by its very wide academic and industrial community, and because it is freely distributed under the GPL license.

C. Multiplatform Support

The variety of hardware platforms, sensors, programming languages, and operating systems supported by SPINE enables a great degree of heterogeneity (see Fig. 8). This allows for a very flexible and usable framework in different BSN application scenarios (e.g., e-Health, e-Fitness/Wellness, and e-Factory), where, due to specific requirements, only certain platforms or operating systems might be used.

At the sensor node level, SPINE supports the most diffused wireless sensor platforms. The TinyOS implementation runs on TelosB/Tmote Sky, MicaZ, and Shimmer (both the IEEE 802.15.4 radio and Bluetooth radio are supported on Shimmer). This implementation also provides an optional security function

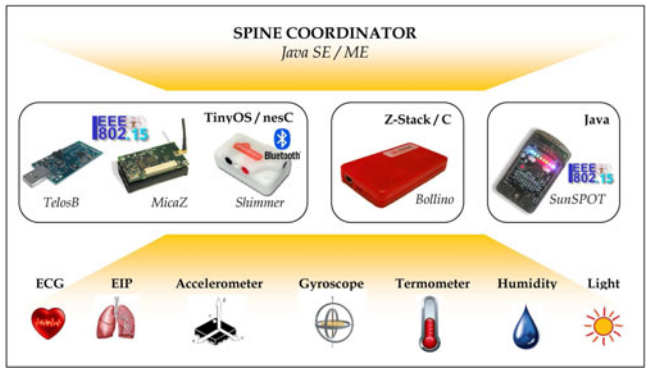

Fig. 8. Heterogeneous physical sensors and sensor platforms supported by SPINE.

TABLE V
SPINE-TESTED MOBILE PERSONAL DEVICES

Device	CPU	RAM	Misc.
HTC NexusOne	1GHz, Qualcomm Snapdragon QSD 8250	512 MB	Android 2.x. MicroSD, up to 32 GB.
Samsung Galaxy S	1GHz, ARM Cortex-A8	512 MB	Android 2.x. MicroSD, up to 32 GB.
Nokia N95	332 MHz, TI OMAP 2420 (ARM11-based)	128 MB	Symbian OS v9.2, S60 rel. 3. MicroSD, up to 32GB
Nokia 6120	369, MHz ARM11	64 MB	Symbian OS v9.2, S60 rel. 3.1. MicroSD, up to 8GB.

that is based on hardware AES-128 encryption of the CC2420 radio (used on the Telosb/Tmote Sky, MicaZ, and Shimmer platforms). In addition, SPINE implementations for ZigBee devices (Telecom Italia "Bollino" equipped with a Chipcon CC2530 SoC) using the Texas Instruments Z-Stack, and for the Java-based SunSPOT nodes, are also available. Many physical sensors (accelerometers, gyroscopes, ECG, electroimpedance plethysmography, temperature, humidity, and light) are supported by default as their drivers are distributed along with the core framework components, and simple signal-processing operations (e.g., mathematical aggregation functions such as max, min, average, and standard deviation) are implemented directly at the sensor node level. As previously mentioned, the set of predefined sensing and processing functionalities can be easily extended. For instance, for the Shimmer platform [29], while the driver for the accelerometer is available by default, developers may integrate further drivers for other sensors that can be attached to the Shimmer (e.g., the gyroscope, ECG, EMG, or the magnetometer). In addition to the built-in processing functions, others may be implemented and integrated into the SPINE framework depending on specific application needs; for instance, it is very simple to integrate additional feature extractors such as the zero crossing or the first derivative, and even simple classifier algorithms such as a small decision trees.

At the coordinator level, SPINE supports heterogeneous devices, spanning from smartphones and PDAs (see Table V for details) to personal computers/workstations. Windows and Linux-based PCs/workstations are widely supported through the Java SE implementation of the framework and the availability of the lower level components for communicating with the remote

sensor nodes. A Java ME porting has also been carried out so that many Java powered smartphones and PDAs can be used as SPINE coordinators. More recently, an Android implementation of SPINE has been developed. It has been tested on several smartphones based on Android 2.2 or above which communicate with Shimmer nodes over Bluetooth. Furthermore, an implementation with limited functionalities using the QT development environment has been realized. It runs on Symbian and Windows, and enables Bluetooth communication with Shimmer nodes using the third-party QBluetooth library. Supporting smartphones as coordinator devices is often crucial for many real-time health monitoring applications especially when the system must be able to properly work continuously and outdoor, where is not possible to rely on fixed infrastructures. Many modern smartphones and tablets easily have sufficient computation and storage capabilities to support different classes of BSN applications (e.g., activity, cardiac and respiratory monitoring) as they can be equipped with 1-GHz (and more) CPUs, possibly dual-core, with hundreds megabytes of RAM and gigabytes of external flash storage drives. Furthermore, through their Internet connectivity capabilities, they can transmit sensed and/or processed data to remote servers and cloud services for offline and long-term analysis [30].

Finally, SPINE has been also ported on an emulator tool, which is called SPINE Emulator, that virtualizes SPINE-enabled sensor nodes. The tool allows emulation of a set of nodes forming a wireless BSN and requires a dataset for each node. The dataset can be built using a provided data collector tool which records data from real sensor nodes. Hence, a particular emulated node is virtually equipped with sensors determined by the given dataset. There are several advantages of using a SPINE emulator. For instance, processing functionality can be tested in the emulated environment first, to simplify the debug process. Furthermore, datasets from real sensors can be used to objectively validate and compare different processing algorithms or hardware setups. Finally, the emulator and a standard dataset can be used by interested developers to investigate the potential of the SPINE framework itself, even if they do not have suitable physical sensor nodes.

V. Evaluating Signal Processing in Node Environment Performance on Multiple Platforms

An extensive performance evaluation of the SPINE framework was carried out, which included all the supported sensor node platforms. It included measurement of the execution time of signal-processing functionalities, memory usage of the framework, and bandwidth and energy consumption under a given application profile. We have, in fact, defined a common benchmark scenario that was supported both by SPINE and implemented with hard-coded logic in TinyOS. In particular, the application profile corresponds to a three-axis accelerometer sensor (attached to the node platform) that is sampled at 20 Hz, and a sequence of 20 sensor readings is transmitted to the coordinator device by using a single message so that the radio is actively used to transmit one message every second. In addition, we have reimplemented some of the significant functionalities of SPINE in specific hard-coded applications in TinyOS, in order to evaluate how much overhead in terms of computation, memory, and energy requirements is added by SPINE.

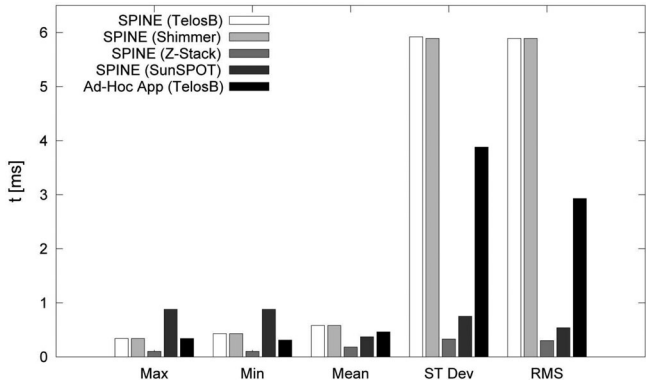

Fig. 9. Execution time of selected in-node functions computed on different sensor platforms using sampling time = 20 Hz, window = 40 samples, and shift = 20 samples.

This analysis is important as it provides the quantitative cost paid for the advantage of drastically reduced development time of BSN applications. As highlighted in the following sections, results from the performance evaluation are a crucial support for SPINE developers that need to achieve high system efficiency.

A. Function Execution Time

Measuring the execution time of some key operations of SPINE running on the supported sensor platforms is important, not only to compare the platforms themselves, but also to identify the upper bound of the sampling rate and transmission rate considering the time needed to process the sensor readings and transmitting the results over-the-air. To ensure the reliability of in-node processing, developers must ensure that the execution time to calculate the most computational-intensive feature is shorter than the sampling period of the associated sensor. This is necessary to avoid potential sampling inconsistencies or overwrites of sensor data before they can be processed. Especially, while using TinyOS, sensor sampling and buffering may be blocked until a feature processing is completed, leading to an inconstant sampling rate.

To provide a practical example, we implemented a simple physical activity recognition system based on Shimmer motes (that natively embed a three-axial accelerometer) placed on the human body. Most of the basic human movements and postures, such as walking or sitting, have loose requirements in terms of sensor sampling rate and feature processing. Accelerometer sampling rate between 20 and 40 Hz, and processing of features such as the average, max, and min value over sensor data windows of 40–100 samples with 25–50% overlap are more than sufficient to enable the classification of simple activities. Calculating such features over 100 samples using the SPINE framework on the Shimmer platform takes much less than the sampling period of 25 ms if the sensor is sampled at 40 Hz (see Fig. 9). Therefore, for this class of applications, it is

TABLE VI
MEMORY REQUIREMENTS, ENERGY CONSUMPTION, BANDWIDTH, AND AVERAGE TRANSMISSION DELAY (FOR SENDING 28 BYTES) OF THE BENCHMARK SCENARIO (MOTION SENSOR BOARD WITH FEATURE EXTRACTORS AND THRESHOLD-BASED ALARMS) ON DIFFERENT PLATFORMS

	Memory Requirements		*Energy Consumption*			*Bandwidth*	*Transmission Delay*	
Application Profile	**RAM (Kb used/av.)**	**ROM (Kb used/av.)**	**Average Power Consumption**	**Battery**	**Lifetime**	**Bitrate**	**802.15.4**	**Bluetooth**
SPINE on *TelosB*	3.7 / 10	33.5 / 48	6.6 mW/s	650mAh	101 h	178 byte/s	10,07	N/A
SPINE on *Shimmer*	4.4 / 10	40.0 / 48	13.9 mW/s	280mAh	21 h	178 byte/s	10,04	N/A
SPINE on *Shimmer Bluetooth*	4.3 / 10	34.4 / 48	87.8 mW/s	280mAh	3 h	150 byte/s	N/A	3,05
SPINE on *Z-Stack*	3.9 / 8	95.9 / 128	11.2 mW/s	650mAh	60 h	160 byte/s	0,61	N/A
SPINE on *SunSPOT*	79.0 / 512	75.0 / 4096	84.2 mW/s	720mAh	9 h	168 byte/s	67,20	N/A
A-hoc application on *TelosB*	1.3 / 10	16.1 / 48	73.7 mW/s	650mAh	9 h	152 byte/s	9,98	N/A
CodeBlue on *TelosB*	3.4 / 10	36.6 / 48	74.2 mW/s	650mAh	9 h	186 byte/s	10,25	N/A
TITAN on *TelosB*	9.0 / 10	38.7 / 48	18.7 mW/s	650mAh	36 h	158 byte/s	10,43	N/A

actually more convenient to enable in-node processing as constant and continuous sensor sampling is always achieved. However, a similar analysis may lead the application designer to choose slower sampling rate requirements, or perform the sensor data processing entirely at the coordinator if in-node processing will interfere with the sensor sampling.

To boost framework performance, SPINE implements an advanced execution mode that can be enabled for certain features. While features are normally computed upon arrival of the last sample in the window associated with that feature, some features such as maximum, minimum, and average can be computed incrementally with each sample. For instance, it is possible to use an average feature implementation that splits the computation in elementary processing steps consisting of a simple summation of a new sensor reading to a total, and a single division of the total by the window size when the data window must slide. Many other features may be split in partial processing steps that are distributed during the sensor sampling and that are individually faster than equivalent monolithic implementation, hence allowing for higher sampling rates.

In addition, the comparison of SPINE processing operations with applications implemented *ad hoc* to execute them without any other overhead (to achieve highest performance) provides an estimation of the overhead introduced by SPINE. Fig. 9 summarizes some relevant time measurements. Z-Stack and SunSPOT platforms are based on a more powerful microcontroller which allows for faster execution times on the feature calculations. However, significant packet transmission overhead, caused by a tall network stack, can be observed on the SunSPOT. TelosB and Shimmer show identical results as they are based on a very similar hardware/software architecture (they have the same microcontroller and 802.15.4 radio, and both run TinyOS). Among the analyzed functions, standard deviation and root mean square are the most time consuming, and, in particular, they take longer on TelosB and Shimmer due to the less performing microcontroller of these platforms (note that we did not use any hardware multipliers for these experiments). As expected, the *ad hoc* implementation on the TelosB performs better because the modularity of SPINE adds some overhead incurred by the communication among its components.

B. Memory Usage

The memory footprint of SPINE was analyzed on different platforms. In particular, we have included with the core framework the drivers for a motion sensor board (which mounts an accelerometer sensor), the feature extractors, and the threshold-based alarms signal-processing units. Results summarized in Table VI show that the framework is well optimized and provides enough free memory for custom-developed extensions to the framework, and for different tradeoffs while dimensioning the size of the sensor data buffers (which affects RAM usage). It is worth noting that complex extensions of the framework, configured to compile with the standard built-in functionalities, may not be installed on selected platforms if the required binary code resulting from the compilation process is greater than the available ROM size. The modular architecture of SPINE, however, helps developers reduce the ROM footprint by easily reconfiguring the framework to exclude any standard sensing and processing functionalities that are not needed for specific applications.

C. Energy Consumption

To better understand the energy consumption results, it is worth noting that while SPINE provides a built-in low-power radio mode for TinyOS sensor platforms with IEEE 802.15.4 radio, this is not available if using the Shimmer Bluetooth radio. The low-power radio mode was not implemented on the hard-coded logic application.

To obtain the actual power consumption of the whole platform, a professional digital oscilloscope connected to the motes was used. The average power consumption is computed as the weighted average consumption between the radio usage (message transmission and listening for incoming packets) time and the sensing/processing time during one cycle (that is, in this case, 1 s, as the sensor is sampled at 20 Hz, and the system

waits 20 samples before packing them into a message that is eventually transmitted).

Results show that the lowest average power consumption is achieved with SPINE running on the TelosB platform, which also results in the longest lifetime since the available Li-ion battery (we have modified the standard TelosB which normally use two AA alkaline batteries) has the greatest capacity among the selected platforms (see Table VI). Although the TelosB and the Shimmer platforms are both based on the same microcontroller (the Texas Instrument MSP430F1611) and radio (the Chipcon CC2420), there is a significant difference in the power consumption among the two platforms. This is due to the accelerometer used by the Shimmer, which consumes about six times more than the one mounted on the custom motion board of the TelosB.

D. Communication Bandwidth

To analyze the bandwidth usage, we refer to the same benchmark scenario defined at the beginning of Section V. Therefore, we send over-the-air, once a second, 20 readings of a three-axis accelerometer, where each single-axis sample takes 2 bytes. Consequently, the total user data to be transmitted each second are 60 samples, totaling 120 bytes. On platforms using the IEEE 802.15.4 CC2420 radio, SPINE automatically fragments this message as the total number of bytes to send is greater than the TX buffer (which is 128 bytes). Results are summarized in Table VI. This includes the TelosB, Shimmer, and SunSPOT. Using the Z-Stack platform is possible to transmit the whole message into a single packet, resulting in a lower bit rate. With the Shimmer, it is possible to use the Bluetooth radio and decrease the overhead incurred by fragmentation. Finally, as expected, a hard-coded implementation of the application allows for a significantly reduced number of bytes transmitted with respect of a SPINE-based implementation because the framework must add to each packet generic and packet-specific headers. Table VI reports the average time to transmit over-the-air a packet of 28 bytes using different sensor platforms. Results show that SPINE does not introduce a relevant overhead with respect to the application-specific implementation on TinyOS. On the SunSPOT, instead, the underlying VM components and a more sophisticated low-level communication model cause a significantly longer transmission time. Finally, as expected, our measurements confirm that using Bluetooth on the Shimmer allows for shorter delays (three times shorter than transmitting the same packet using the 802.15.4 radio).

E. Comparison Between Signal Processing in Node Environment, CodeBlue, and TITAN

In addition to the comparison of the benchmark scenario among different platforms running the SPINE framework, we have implemented our benchmark using the CodeBlue and TITAN frameworks (see Section III) and compiled it for the TelosB platform. This allowed us to analyze and compare the performance and programming effectiveness of the three frameworks to address a simple yet meaningful BSN application. From a performance point of view, as summarized in Table VI, the SPINE and CodeBlue implementations of the benchmark have similar memory and bandwidth requirements, while the SPINE implementation performs much better in terms of system lifetime; the TITAN implementation, instead, has a RAM footprint almost three times bigger and, although performing better than CodeBlue in terms of energy consumption, its lifetime is three times shorter than the SPINE implementation. In conclusion, with its domain-specific design and low-power mechanisms, SPINE is more suitable to realize memory- and power-efficient BSN applications.

This comparison has been useful also to compare the three frameworks in terms of programming effectiveness on the BSN domain. In particular, the application profile has been implemented using SPINE "as is," without the need for modifying or extending any framework modules, which is a significant result. This is also because SPINE provides a flexible and expressive application level protocol, a wide range of predefined sensing and processing functionalities, built-in support for sensor data buffered transmission and low-level packet segmentation. Conversely, both the CodeBlue and TITAN implementations required extensions to the core modules, thus delaying the prototyping time.

VI. Signal-Processing-in-Node-Environment-Based Applications

In SPINE, sensors can be set up, activated, and disabled dynamically, and their output can be arbitrarily connected to the online processing at run-time based on external controls. Of course, to support different applications, the wearable sensor nodes must be equipped with all the required physical sensors. For instance, a doctor could use SPINE nodes that are equipped with accelerometers and a suitable coordinator device, such as a smartphone or a tablet, to monitor weekly energy expenditure of a patient. The same nodes could be used later with another patient in a rehabilitation scenario, as long as the proper application software is available on the coordinator node.

We present six prototypical BSN systems (activity recognition, physical rehabilitation, gait analysis, Kcal expenditure, emotional stress detector, and handshake detection), that we have been built atop SPINE, to show the practical flexibility of our framework. These applications configure the SPINE network differently to meet their requirements; however, the same hardware devices and node-side software have been used in each case.

A. Activity Recognition

Physical activity recognition is one of the fundamental building blocks of many BSN applications [31]. It is necessary to monitor daily activity levels for wellness applications; it may help identifying abnormal heart rate variations, e.g., by correlating the rate variations with the current activity being performed, and it can be applied in highly interactive computer games, to cite a few scenarios. Our prototype [32] uses only two wireless motion sensor nodes placed on the waist and on the thigh of wearer, and a personal smartphone running the end-user application which is able to detect four basic activities (lying down, sitting, standing, and walking). This is achieved with or without

an individual training phase, and with an overall average accuracy of about 98%. Furthermore, the system may also report the number of steps during walking and notify via SMS or registered voice call in the event of accidental falls that may potentially lead to dangerous situations (e.g., if, after a detected fall, the system recognizes the subject is lying down for several minutes). Finally, the application also provides an incremental learning system that users may use to increase the set of activities that can be recognized (e.g., to add a "kicking" or "jumping" activity).

B. Physical Rehabilitation

Wearable wireless motion sensors are also used for physical rehabilitation purposes [33]. It is quite common to require repetitive physical exercises, for instance, to recover from a muscle strain, a limb fracture, or a surgery. Having real-time feedback about exercise performance quality would allow users to independently exercise properly without the need of a continuous professional assistance. In our physical rehabilitation application [34], we used SPINE nodes equipped with accelerometer sensors to monitor arm and leg movements. The application consists of monitoring leg and arm bending movements in real time and comparing them with the ones recorded during a setup phase. The application scenario consists of two steps, namely, setup and exercise phases. During the setup phase, the user wears two sensors on either leg or arm that needs to be exercised and performs the correct exercise under the guidance of rehabilitation professional. Meanwhile, the system records the data and stores them as reference exercise. Then, during the exercise phase, the user repeats the bending movement and is provided with a real-time feedback about how well the movement is done with respect to the stored reference exercise.

C. Gait Analysis

Gait analysis is the study of human walking [35]. Recently, it has been augmented by instrumentation for measuring body movements, body mechanics, and the activity of the muscles. Gait analysis is used to assess and treat individuals with conditions affecting their ability to walk. A novel method of gait analysis, using a hidden Markov model (HMM)-based technique to extract temporal parameters from gait, has demonstrated good results and the potential to be partially implemented on the hardware constrained sensing nodes [36], [37]. In our application prototype, the HMM classifier has been implemented in SPINE to run on the nodes in real time; however, an offline processing step is still required to train the algorithm. Given the trained HMM, SPINE nodes can classify the four walk phases (initial swing, mid-swing, terminal swing/initial stance, and mid-stance) and report back to the coordinator node the classification results. In our test case, we extracted heel-down and heel-lift events from a walking subject using a single wearable node at the waist equipped with a three-axis accelerometer sensor.

D. Kcal Expenditure

Physical activity is essential to overall human health [38]. Accurate and objective measurement of daily activity is needed to determine whether the suggested guidelines provided by the medical scientific community are met. Our open-source application [34] estimates the energy consumed, while performing daily basic activities, and has been implemented on top of SPINE. It uses a single sensor node equipped with a three-axis accelerometer, placed on the belt (or in a trousers pocket). Because accelerometer data are dynamically prefiltered, removing the gravity components, the mote can be arbitrarily oriented. Furthermore, measurements are robust against tilting of the mote inside the pocket. The node reports an activity count measurement to the coordinator once per second, and the coordinator computes an estimation of the energy expenditure (expressed in Kcal or KJ) every minute. The algorithm can be tuned with the subject gender and weight.

E. Emotional Stress Detector

Many studies show connections between long-term exposure to stress and risk factors and poorer immune functions and cardiovascular diseases. Although the exact mechanisms of how stress exerts these effects are not yet well known, a stress monitoring technique that would measure stress levels may significantly support the development of methods for the mitigation of long-term stress, and promote healthier life style. Heart rate variability (HRV) is among the most promising markers for this purpose [39]. HRV represents the variations in the beat-to-beat alteration in the heart rate. We have realized a custom sensor board (named *CardioShield*) with a specific receiver for a wireless chest belt (produced by Polar Electro) detecting heart beats. The board is mounted on a standard TelosB mote running SPINE, customized with a specific processing function that computes the inter-beat time interval. The prototype [40] running on the SPINE coordinator receives these timed events and performs an HRV analysis by extracting time-domain features on the inter-beat times over a 10-min window. Then, these features are fed to a simple threshold-based classifier to estimate whether the subject is stressed.

F. Handshake Detection

The handshake gesture is an important part of the social etiquette in many cultures. The automated detection of a handshake among people can enable wide range of pervasive computing scenarios; in particular, different types of information can be exchanged and processed among the handshaking persons, depending on the physical/logical contexts where they are located and on their mutual acquaintance. We have realized a handshake detection system based on a collaborative BSN [41] having a base station and a wrist-wearable, three-axial accelerometer-equipped wireless sensor node [42]. The node communicates with its corresponding base station only when a potential handshake-like movement is detected. Moreover, sampling on the sensor node is activated only when people are in sufficient proximity with each other so as to exchange networking

TABLE VII
SPINE EXTENSIONS REQUIRED BY THE PRESENTED APPLICATIONS

Application	SPINE Node-side	SPINE Coordinator-side
Activity Recognition	None	Off-line feature selection algorithm. Real-time K-Nearest Neighbor classifier.
Physical Rehab	None	Trigonometric function for upper and lower body joint angles estimation.
Gait Analysis	HMM on-line classifier (trained off-line using Matlab). Custom-defined gait data message encoder.	Custom-defined gait data message decoder.
Kcal Expenditure	Activity counts feature extraction.	Kcal estimation algorithm, based on activity counts values from wearable node.
Stress Detector	Driver adapter for Cardio-Shield sensor-board. RRi feature extraction.	Time-domain HRV analysis. Stress detector classifier.
Handshake Detection	Zero-crossing and Near-zero feature extraction. On-line local decision tree classifier for potential handshake detection.	Extension for Collaborative-BSN interaction model. Distributed decision tree classifier based on joint data from potential meeting people.

data through their BSNs. If the same potential handshake event is detected by the two interacting BSNs that have advertised each other, they cooperatively run a handshake detection protocol to recognize the handshake gesture. The recognition phase is based on two J48 tree classifiers: one executing on the node, which detects potential handshake, and the other running on the coordinator, which provides the final outcome by aggregating data from the interacting BSN. As the cooperative recognition algorithm requires the interaction between BSNs, we have extended the SPINE framework with a specific API to support bidirectional inter-BSN communication.

G. Signal-Processing-in-Node-Environment-Based Application Development Analysis

Table VII summarizes the main programming efforts in terms of customization and extensions of the basic SPINE framework that were required to implement the presented applications. Thanks to the specific support for common sensors and processing functions in the BSN domain, only the very application-specific logic (e.g., the required classifiers and specific algorithms) was added to SPINE. The only two exceptions are represented by the gait analysis system which required a custom-defined gait data message, along with the corresponding encoder and decoder and by the emotional stress detector for which a specific hardware sensor-board has been used which in turn required the integration into SPINE of its driver adapter.

VII. SIGNAL-PROCESSING-IN-NODE-ENVIRONMENT-BASED DESIGN METHODOLOGY

In the context of WSNs, and BSNs in particular, the design of systems usually follows a "*bottom-up*" approach such that such systems are developed choosing the hardware components first, then the communication protocols, and, finally, implementing *ad hoc* applications on top of the underlying infrastructure. Alternatively, the "*top-down*" approach is adopted when the designer chooses to start from the high-level application requirements and map them to an application-level framework, i.e., a set of programming abstractions and APIs, without any intrinsic assumptions on the underlying protocol stacks and hardware platforms.

Through the use of SPINE for the development of many different BSN applications/systems (see Section VI), we have defined a novel method to support rigorous BSN system design to have reliable systems, system efficiency, and true interoperability between different applications as well as between different implementation platforms. The SPINE-based design methodology (SPINE-based DM) is a BSN system design method based on the well-known PBD [43], in which the defined platforms are opportunely semi-instantiated.

In particular, the PBD methodology consists of a sequence of steps that leads the initial high-level description of a system to its final implementation. Each step is a refinement process that transforms the design from a higher level description to a lower level description that is progressively closer to the final implementation. This refinement step is obtained by *mapping* each block of the higher level description with lower level blocks (or set of blocks). The mapping choice is the result of a constrained optimization problem: the methodology selects the mapping that satisfies the constraints coming from the higher level description, while optimizing according to a cost function specified by the designer. In the PBD formalism, each layer of abstraction is referred as a *platform*.

Within the PBD-oriented SPINE-based DM, we identified three layers of abstraction and corresponding platforms: the *Service Platform* at the application layer, the *Protocol Platform* to describe the protocol stacks, and the *Implementation Platform* for the hardware devices. Each design integrates an instance of the three layers, thus containing both application, protocol and device layers. In particular, each design is a complete instance of the BSN system under-development at a given refinement step: high-level, detailed design (DD), or implementation.

However, differently from the pure PBD approach, some of our platforms are semi-instantiated and, as a consequence, the designer is guided to the development of a SPINE-based efficient BSN system. In particular, the service platform is represented by the high-level API exposed by the SPINE Framework (see Section IV). The high-level application functionalities can be mapped freely to the different and flexible services provided by SPINE through its API commands. The protocol platform supports and models two different protocol stacks: IEEE 802.15.4 and Bluetooth. The protocol platform is the last to be instantiated as the choice is almost always forced by the choice made by the designer at the implementation platform (specifically, it

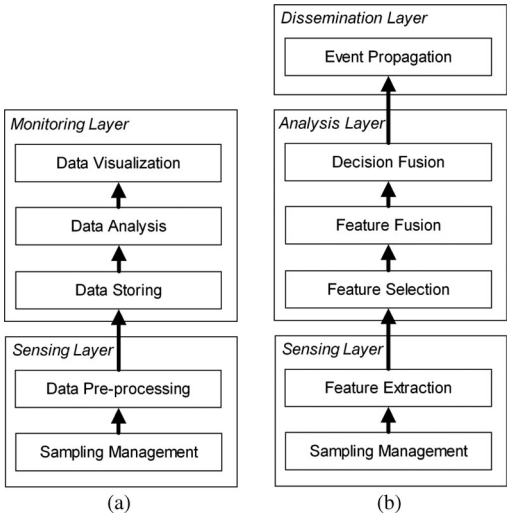

Fig. 10. Architectural schemas of design patterns: (a) sensor data collection for monitoring; (b) multisensor data fusion for detection/classification of events.

depends on the available type of radio on the target devices). The implementation platform provides a significant variety of hardware devices (both in terms of sensor nodes and coordinator units) the designer may choose according to low-level system requirements. This platform is also semi-instantiated, as we assume to make available only TinyOS-based sensor node architectures predeployed with the SPINE Framework, and Java-based and/or Android-powered devices/computers (e.g., laptops, smartphones, or tablets) that will be used as BSN coordinator units.

A. Pattern-Driven Application-Level Design

The application-level design of a SPINE-based BSN application can be produced by following different pattern-driven strategies. In the following, we discuss two frequent design patterns that we use in the development of SPINE-based applications (see Section VI):

1) *Sensor Data Collection for Monitoring*: This pattern is the simplest one as it supports the development of BSN applications which collect data from a set of wearable sensors into the coordinator that, in turns, can archive, analyze or simply display such data. The pattern architectural schema is shown in Fig. 10(a). The main components, which have to be specifically defined, are organized in two layers:
   a) the *Sensing layer*, where data are collected from the sensor nodes;
   b) the *Monitoring layer*, where data can be stored, analyzed, and visualized.

   Each of such layers can be implemented both at the sensor node and at the coordinator node (see Section IV). Specifically, at the sensing layer, the sampling management component provides (multiple) sensed data to the data preprocessing component that can preprocess incoming sensed data. At the monitoring layer, data can be stored (in RAM or in mass-memory) by the data-storing component, analyzed (by an application-specific logic) by the data analysis component, and displayed through the (application-specific) data visualization component. Each of such components can be optional.
2) *Multisensor Data Fusion for Detection/Classification of Events*: This pattern is an extension of the previous one specifically focusing on the detection and/or classification of events of interest, e.g., fall detection, activity recognition, stress detection, and handshake detection (see Section VI). The pattern architectural schema is shown in Fig. 10(b). The main components, which have to be specifically defined, are organized in three layers:
   a) the *Sensing layer*, where data are collected from the sensor nodes;
   b) the *Analysis layer*, where decisions are extracted from data;
   c) the *Dissemination layer*, where information is provided to application built atop the BSN.

   Each of such layers can be implemented both at the sensor node and at the coordinator node (see Section IV). Specifically, at the sensing layer, the sampling management component provides (multiple) sensed data to the feature extraction component that process incoming data to compute specific features (e.g., max, min, dev std, etc). At the analysis layer, the feature selection component is able to select the most significant features and the feature fusion component puts the selection feature together. Finally, the decision fusion component is able to provide decision on the basis of the incoming set of features, e.g., classification of postures of humans on the basis of features computed on the accelerometer data coming from sensors worn on the human body (see Section VI-A). At the dissemination layer, the event propagation component sends the decision to application-level local and/or remote components.

Both the two presented patterns can be fully supported by the SPINE middleware (see Section IV).

B. System Parameters

The key parameters affecting BSN systems at different degrees of refinement (from high-level design (HLD) to implementation and deployment) can also be categorized into the three reference levels of the proposed method, as follows:

1) *Application Level:* At this level, the most important parameters are the system accuracy, reliability, and responsiveness. Accuracy is application specific and related to object/event classification, e.g., activity recognition accuracy (see Section VI-A) and stress detection accuracy (see Section VI-E). Reliability is often a key parameter especially for life-critical applications (e.g., early detection of heart attacks, fall prevention, or detection), although typically an expensive property to be guaranteed in any possible case. Responsiveness is intuitively related to the ability of the system to provide the necessary feedback to the user within acceptable times and is also application specific as it depends on the computation load to perform the main operation, e.g., computation of extension degrees in rehabilitation systems (see Section VI-B), detection

of activity in an activity recognition system (see Section VI-A), and detection of a handshake in a handshake detection system (see Section VI-F).

2) *Protocol Level:* At this level, the main affecting parameters are the bandwidth and delay which depend on the sensor sampling rates, sensor- and application-specific generated data, and obviously on the communication protocol itself. Moreover, specific synchronization requirements among sensor nodes can be accommodated by the selected protocol, e.g., by using a time-division multiplexing access technique, whereas more complex synchronization constraints are usually handled at application level.
3) *Device Level:* At this level, the main affecting parameters are the energy consumption, the required memory, and the required computing power (see Sections IV and V). The energy consumption depends on duty cycle, sensor type and sampling rates, communication activity (both in terms of radio transmission and reception), and application-specific signal processing. The required memory (i.e., RAM and Flash) depends on the software platform tailoring (i.e., which TinyOS and SPINE components are wired in), the sampling parameters (sampling rate), the buffering parameters for sensor data storing and computation (number of buffers, window and shift), and also on application-specific signal processing (e.g., mean, variance, classification trees, and total energy). Finally, the computing power essentially depends on the defined application-specific signal processing.

C. Process Schema

The schema of Fig. 11, sketched in the standard OMG SPEM (Software & Systems Process Engineering Metamodel specification) 2.0 notation [44], shows the SPINE-based platform design process schema. The process is iterative and consists of the following steps (which can be carried out by different roles, modeler, designer, and developer).

1) *Requirements Analysis* produces a set of functional and nonfunctional requirements that will drive the design activity.
2) HLD produces an HLD of the BSN system on the basis of the identified requirements. In particular, an HLD is an instance of the reference SPINE framework integrated with selected protocols and sensor/platforms.
3) *Performance Estimation of HLD* allows us to analyze the HLD by estimating the HLD performance by using available analytical/simulation methods (see the SPINE Emulator in Section IV-C). Results cannot be precise at this stage; however, they provide insights on the feasibility of the HLD to be translated into an effective DD. If the requirements are not fulfilled, the HLD activity is rexecuted.
4) DD produces the DD of the defined HLD. The HLD is refined at the different three layers of SPINE-based DM and, specifically, at application level by following the pattern-driven design described in the previous subsection.
5) *Performance Estimation of DD* allows us to analyze the DD by testing/estimating the DD performance by using

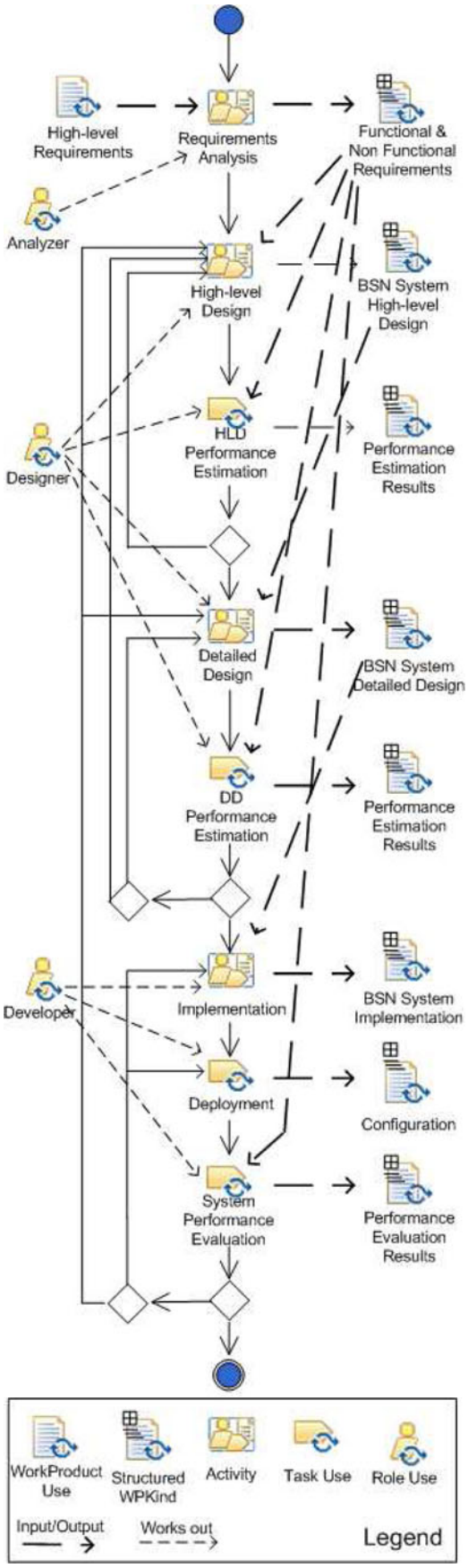

Fig. 11. SPINE DM process schema.

available analytical/simulation methods (see the SPINE Emulator in Section IV-C) and by mapping some components of the DD onto the device level and testing them. Results should be more precise than those obtained in the performance estimation HLD task; they should provide more valuable insights on the feasibility of the DD to be translated into an effective and efficient implementation. If the requirements are not fulfilled, either the DD or HLD activities are reexecuted.

6) *Implementation* produces the implementation of the defined DD. The BSN system is now ready to be deployed and fully executed and tested.
7) *Deployment* allows to define the deployment of the BSN systems by uploading the code and tuning the most important system parameters.
8) *System Performance Evaluation* allows to carefully test the BSN system and extract all performance measurements and results for its validation. The obtained results will provide a full-fledged test of the system. If the requirements are not fulfilled, either the implementation, DD, or HLD activities are reexecuted.

VIII. Lesson Learned

Releasing SPINE as an open-source framework, along with a dedicated website and developer mailing list [27], has been very helpful for creating a community of both contributors to the code and users of the framework. Among many others, the most relevant external contributions and feedback have come from the EECS Department of the University of California at Berkeley, the Aarhus University in Denmark, the CES Laboratory—ENIS in Tunisia, and the Department of Biomedical Engineering at Tampere University of Technology in Finland.

SPINE is being used in several research projects, spanning from human activity recognition, to physical rehabilitation, to heart-rate and respiratory monitoring, and even in an agriculture scenario for monitoring animals in a stable.

Also thanks to the feedback we received, we have summarized a number of perceived pros and cons while using the SPINE framework. The lack of a packet acknowledgment/retransmission system has been reported as probably the biggest limitation, as it can cause run-time sensor network configuration failure, especially when configuring many nodes in a noisy environment. Although SPINE actually provides a limited packet acknowledgment/retransmission mechanism, this is based on a very simple approach and available only for messages sent by the coordinator to the sensor nodes. The rationale of this design choice is that SPINE is an application-level framework; hence, network-level functionalities should be provided by the exploited sensor platforms (e.g., ZigBee provides packet acknowledgment/retransmission, as well as some medium access control (MAC) implementation in TinyOS). Another perceived limitation is due to the lack of a configuration mechanism that allow for starting up the sensing/processing routines on individual nodes. Currently, SPINE provides straightforward support for the configuration of different sensing and on-board processing functionalities on each individual sensor node, but the actual start-up of these functionalities is synchronized over all the nodes forming the BSN. Some users reported the need for persistent on-node storage, as in some application scenarios, the person wearing the system might not be within the radio range of the coordinator station all the time. It is worth noting, however, that the reference application scenario is typically based on the use of personal and portable BSN coordinator devices such as smartphones and tablets that users carry, e.g., in their pocket while using the system during their daily activities.

Fortunately, the main features of the SPINE framework appear to be widely acknowledged among the SPINE community as important and very useful to rapidly build BSN applications. In particular, the integration of new, custom-defined sensor drivers and signal-processing functionalities has found to be very easy and straightforward (see Section IV-A). The higher level communication APIs at both the sensor node and the coordinator side have also been mentioned frequently as an important advantage of SPINE. In one instance, a BSN prototype was reimplemented as a SPINE-based prototype, at savings in development time of 80%, as demonstrated with the implementation of the research prototypes shown in Section VI.

IX. Conclusion

In this paper, we have identified and discussed the fundamental requirements that a designer should take into account, while developing an effective and efficient domain-specific framework for programming BSN applications. We have described the SPINE middleware, which is an open-source domain-specific framework to support rapid development of BSN applications. We have also highlighted how the distinctive characteristics of the framework fully address all the identified requirements. In addition, we have emphasized the wide support of SPINE for heterogeneous sensor and coordinator platforms, which allows sufficient freedom for selecting the most appropriate hardware and software infrastructure during application design. We have also evaluated the performance of SPINE under different dimensions (execution time, memory usage, energy consumption, communication bandwidth) to demonstrate its programming effectiveness and practical usability on most of the popular sensor node platforms. Furthermore, we defined an application profile benchmark and implemented it on SPINE, CodeBlue, and Titan to compare the performance and the programming effectiveness of these frameworks in the BSN domain, showing that the system implemented with SPINE reaches higher performance with less programming efforts. We have also reported a few applications that have been built atop SPINE, such as physical activity recognition and rehabilitation support, handshake detection, emotional stress indication, physical energy expenditure estimation, and gait analysis. Although these application scenarios require different sensing and signal-processing capabilities, the flexibility and modularity of SPINE allowed uniform support for their development. With consideration of the feedback, suggestions, and critiques of many academic and industrial developers that used SPINE to implement BSN applications, we have provided a summary of lessons learned. Their feedback gave us important information about the perceived advantages and limitations of

using our framework, allowing us to define a number of critical improvements that are currently being finalized. In particular, the next release of SPINE will address the limitations that were reported to us, also including support for multi-base-station BSN configuration and sensor data gathering, persistent (on-demand and automatic) local sensor data storage when a BSN is out of the radio range of its coordinator, an optimized mechanism to support two-way message acknowledgment and retransmissions, and, for the TinyOS implementation, the use of the "TKN 15.4 MAC". TKN 15.4 MAC is a fully compliant implementation of the IEEE 802.15.4 standard, developed in the context of CONET (a EU-funded project under ICT, Framework 7) by the Telecommunication Networks Group at the Faculty of Electrical Engineering and Computer Science at the Technische Universität Berlin. In particular, multi-base-station BSN configuration and sensor data gathering would allow for much higher mobility and freedom of the users. It enables scenarios in which, for instance, multiple base stations are deployed as a backbone infrastructure, and human beings wearing a BSN may move in the environment without losing connectivity with the system, even though they are not carrying their personal coordinator device all the time. Finally, the SPINE-based DM is being released to the SPINE community to receive feedback of its application for the development of SPINE-based BSN systems.

ACKNOWLEDGMENT

The authors would like thank all the SPINE community and, in particular, would like to mention F. Bellifemine and A. Salmeri (Telecom Italia), A. Sangiovanni-Vincentelli (University of California at Berkeley), S. Iyengar (Google Inc.), M. Sgroi (formerly Telecom Italia/Pirelli WSN Lab Berkeley), M. Kjeldsen (Aarhus University), W. Bouzayani (ENIS, Tunisia), and V.-P. Seppä (Tampere University of Technology) for their precious contribution and feedback on the framework.

REFERENCES

[1] Y. Hao and R. Foster, "Wireless body sensor networks for health-monitoring applications," *Physiol. Meas.*, vol. 29, no. 11, pp. R27–R56, Nov. 2009.

[2] P. Alexandros and B. Nikolaos, "A Survey on wearable sensor-based systems for health monitoring and prognosis," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 1, pp. 1–12, Jan. 2010.

[3] C.-L. Fok, G.-C. Roman, and C. Lu, "Mobile agent middleware for sensor networks: An application case study," in *Proc. 4th Int. Symp. Inf. Process. Sensor Netw.*, Apr. 2005, pp. 382–387.

[4] R. Kumar, M. Wolenetz, B. Agarwalla, J.-S., P. Hutto, A. Paul, and U. Ramachandran, "DFuse: A framework for distributed data fusion," in *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst.*, Nov. 2003, pp. 114–125.

[5] W.-B. Heinzelman, A.-L. Murphy, H.-S. Carvalho, and M.-A. Perillo, "Middleware to support sensor network applications," *IEEE Netw.*, vol. 18, no. 1, pp. 6–14, Jan./Feb. 2004.

[6] S. Madden, M.-J. Franklin, J. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad-hoc sensor networks," in *Proc. 5th Symp. Oper. Syst. Des. Implementation*, Dec. 2002, pp. 131–146.

[7] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: A publish/subscribe middleware for sensor networks," *Pers. Ubiquitous Comput.*, vol. 10, no. 1, pp. 37–44, Dec. 2005.

[8] W. Codenie, K. D. Hondt, P. Steyaert, and A. Vercammen, "From custom applications to domain-specific frameworks," *Commun. ACM*, vol. 40, no. 10, pp. 70–77, Oct. 1997.

[9] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *IEEE Comput.*, vol. 34, no. 11, pp. 44–51, Nov. 2001.

[10] U. Maurer, A. Smailagic, D.-P. Siewiorek, and M. Deisher, "Activity recognition and monitoring using multiple sensors on different body positions," in *Proc. Int. Workshop Wearable Implantable Body Sensor Netw.*, Apr. 2006, pp. 113–116.

[11] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster, "Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection," in *Proc. 5th Eur. Conf. Wirel. Sensor Netw.*, Jan. 2008, pp. 17–33.

[12] C. Zhu and W. Sheng, "Wearable sensor-based hand gesture and daily activity recognition for robot-assisted living," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 3, pp. 569–573, May 2011.

[13] W. Li, J. Bao, X. Fu, G. Fortino, and S. Galzarano, "Human postures recognition based on D-S evidence theory and multi-sensor data fusion," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2012, pp. 912–917.

[14] J. Luprano, J. Sola, S. Dasen, J.-M. Koller, and O. Chetelat, "Combination of body sensor networks and on-body signal processing algorithms: The practical case of MyHeart project," in *Proc. Int. Workshop Wearable Implantable Body Sensor Netw.*, Apr. 2006, pp. 76–79.

[15] C.-L. Lisetti and F. Nasoz, "Using noninvasive wearable computers to recognize human emotions from physiological signals," *EURASIP J. Appl. Signal Process.*, vol. 2004, pp. 1672–1687, Jan. 2004.

[16] G. Chanel, C. Rebetez, M. Bétrancourt, and T. Pun, "Emotion assessment from physiological signals for adaptation of game difficulty," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 6, pp. 1052–1063, Nov. 2011.

[17] C. Katsis, N. Katertsidis, G. Ganiatsas, and D. Fotiadis, "Toward emotion recognition in car-racing drivers: A biosignal processing approach," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 3, pp. 502–512, May 2008.

[18] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, D. Standaert, M. Akay, J. Dy, M. Welsh, and P. Bonato, "Monitoring motor fluctuations in patients with Parkinson's disease using wearable sensors," *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, no. 6, pp. 864–873, Nov. 2009.

[19] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, "CodeBlue: An ad hoc sensor network infrastructure for emergency medical care," in *Proc. Workshop Appl. Mobile Embedded Syst.*, Jun. 2004.

[20] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.-E. Culler, and K.-J. Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Not.*, vol. 35, no. 11, pp. 93–104, Nov. 2000.

[21] 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.*, Nov. 2009, pp. 24–27.

[22] SunSPOT website. (2011). [Online]. Available: http://www.sunspotworld.com

[23] C. Lombriser, D. Roggen, M. Stager, and G. Troster, "Titan: A tiny task network for dynamically reconfigurable heterogeneous sensor networks," in *Proc. 15th Fachtagung Kommunikation in Verteilten Syst.*, Feb. 2007, pp. 127–138.

[24] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri, "A java-based agent platform for programming wireless sensor networks," *Computer J.*, vol. 54, no. 3, pp. 439–454, 2011.

[25] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi, "SPINE: A domain-specific framework for rapid prototyping of WBSN applications," *Softw.: Pract. Exp.*, vol. 41, no. 3, pp. 237–265, Mar. 2011.

[26] 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. 3rd Int. Conf. Body Area Netw.*, Mar. 2008.

[27] SPINE website. (2012). [Online]. Available: http://spine.deis.unical.it

[28] G. Holmes, A. Donkin, and I. Witten, "Weka: A machine learning workbench," in *Proc. 2nd Aust. New Zealand Conf. Intell. Inf. Syst.*, 1994, pp. 1269–1277.

[29] Shimmer website. (2011). [Online]. Available: http://www.shimmer-research.com

[30] M. Boulos, S. Wheeler, C. Tavares, and R. Jones, "How smartphones are changing the face of mobile and participatory healthcare: An overview, with example from eCAALYX," *BioMed. Eng. OnLine*, vol. 10, no. 24, Apr. 2011.

[31] L. Wang, T. Gu, H. Chen, X. Tao, and J. Lu, "Real-time activity recognition in wireless body sensor networks: from simple gestures to complex

activities," in *Proc. 16th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2010, pp. 43–52.
[32] R. Giannantonio, R. Gravina, P. Kuryloski, V.-P. Seppä, F. Bellifemine, J. Hyttinen, and M. Sgroi, "Performance analysis of an activity monitoring system using the SPINE framework," in *Proc. 3rd Int. Conf. Pervasive Comput. Technol. Healthc.*, Apr. 2009, pp. 1–8.
[33] E. Jovanov, A. Milenkovic, C. Otto, and P. de Groen, "A Wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation," *J. NeuroEng. Rehab.*, vol. 2, no. 6, Mar. 2005.
[34] R. Gravina, A. Andreoli, A. Salmeri, L. Buondonno, N. Raveendranathank, V. Loseu, R. Giannantonio, E. Seto, and G. Fortino, "Enabling multiple BSN applications using the SPINE framework," in *Proc. Int. Conf. Body Sensor Netw.*, Jun. 2010, pp. 228–233.
[35] J. Perry, *Gait Analysis: Normal and Pathological Function*. Thorofare, NJ: Slack Inc., 1992.
[36] E. Guenterberg, H. Ghasemzadeh, and R. Jafari, "A distributed hidden Markov model for fine-grained annotation in body sensor networks," in *Proc. 6th Int. Workshop Wearable Implantable Body Sensor Netw.*, Jun. 2009, pp. 339–344.
[37] N. Raveendranathan, S. Galzarano, V. Loseu, R. Gravina, R. Giannantonio, M. Sgroi, R. Jafari, and G. Fortino, "From modeling to implementation of virtual sensors in body sensor networks," *IEEE Sensors J.*, vol. 12, no. 3, pp. 583–593, Mar. 2012.
[38] D.-E. Warburton, C.-W. Nicol, and S.-S. Bredin, "Health benefits of physical activity: the evidence," *Can. Med. Assoc. J.*, vol. 174, no. 6, pp. 801–809, Mar. 2006.
[39] L. Bernardi, J. Wdowczyk-Szulc, C. Valenti, S. Castoldi, C. Passino, G. Spadacini, and P. Sleight, "Effects of controlled breathing, mental activity and mental stress with or without verbalization on heart rate variability," *J. Amer. Coll. Cardiol.*, vol. 35, no. 3, pp. 1462–1469, May 2000.
[40] A. Andreoli, R. Gravina, R. Giannantonio, P. Pierleoni, and G. Fortino, "SPINE-HRV: A BSN-based toolkit for heart rate variability analysis in the time-domain," in *Wearable and Autonomous Biomedical Devices and Systems for Smart Environment*, (Lecture Notes in Electrical Engineering. vol. 75, Berlin, Germany: Springer, 2010, pp. 369–389.
[41] A. Augimeri, G. Fortino, S. Galzarano, and R. Gravina, "Collaborative body sensor networks," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2011, pp. 3427–3432.
[42] A. Augimeri, G. Fortino, M. Rege, V. Handzisky, and A. Wolisz, "A cooperative approach for handshake detection based on body sensor networks," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2010, pp. 281–288.
[43] K. Keutzer, A.-R. Newton, J.-M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 12, pp. 1523–1543, Dec. 2000.
[44] Software & Systems Process Engineering Metamodel specification (SPEM) Version 2.0. (2008). [Online]. Available: http://www.omg.org/spec/spem/2.0/

Giancarlo Fortino (M'02) received the Ph.D. degree in computer engineering from the University of Calabria, Rende, Italy, in 2000.

He is currently an Associate Professor of computer science with the Department of Electronics, Computer Science and Systems, University of Calabria. He was a Visiting Researcher with the International Computer Science Institute, Berkeley, CA, in 1997 and 1999. He is the author of more than 150 papers in international journals, conferences, and books. He is also co-founder and President of SenSysCal S.r.l., a spin-off of the 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. His research interests include distributed computing and networks, agent systems, agent-oriented software engineering, wireless sensor networks, and streaming content distribution networks.

Dr. Fortino currently serves in the Editorial Board of the *Journal of Networks and Computer Applications* (Elsevier). He is also member of the IEEE Systems, Man, and Cybernetics Society.

Roberta Giannantonio received the M.S. degree in telecommunication engineering from Politecnico di Torino, Torino, Italy.

She then joined Telecom Italia, Turin, Italy, where she is currently involved in research projects about wireless technologies. She is active into the ZigBee Alliance and is visiting the Telecom Italia WSN Lab, Berkeley, CA. She is the coauthor of patents and papers on wireless technologies.

Raffaele Gravina received the Ph.D. degree in computer engineering from the University of Calabria, Italy, in 2012.

His research interests are focused on high-level programming methods for WSNs. He is the main designer of the SPINE Framework and responsible for the open-source contributions. He spent two years as researcher at the Telecom Italia WSN Laboratory at Berkeley, California. He is involved in several research projects on WSNs, including MAPS and the REWSN Cluster of CONET FP7. He is co-founder of SenSysCal S.r.l.

Philip Kuryloski received the Ph.D. degree in electrical and computer engineering from Cornell University, Ithaca, NY, in 2009

He is currently a Postdoctoral Researcher with the University of California, Berkeley. His research interests include the design and development of privacy and security features for wireless networked sensing systems.

Roozbeh Jafari (M'98–SM'12) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2000, the M.S. degree in electrical engineering from the State University of New York, 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.

In 2006–2007, he was a Postdoctoral Researcher with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. He is currently an Assistant Professor with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, where he is the Director of the Embedded Systems and Signal Processing Lab. 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.