Università della Calabria

Dipartimento di Elettronica, Informatica e Sistemistica

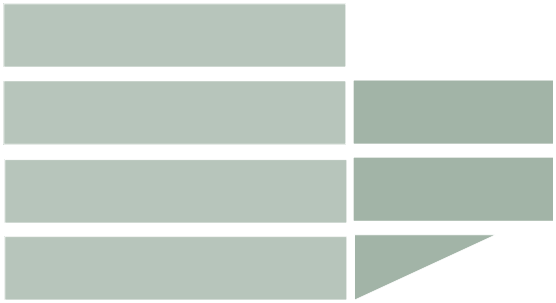
Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XXIV Ciclo

Tesi di Dottorato

A Domain-Specific Approach for Programming Wireless Body Sensor Network Systems

Raffaele Gravina



Università della Calabria

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

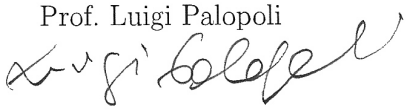
XXIV Ciclo

Tesi di Dottorato

A Domain-Specific Approach for Programming Wireless Body Sensor Network Systems

Raffaele Gravina

Coordinatore
Prof. Luigi Palopoli



Supervisore
Prof. Giancarlo Fortino

Dipartimento di Elettronica, Informatica e Sistemistica
Novembre 2011

Settore Scientifico Disciplinare: ING-INF/05

Acknowledgments

Getting a PhD is not only an academic achievement, but the completion of a maturation process both at professional and personal level. Many things have contributed to this process, some of which I learned at my Department, some important others I learned during the time spent in the unforgettable and diverse environment called Berkeley.

This PhD thesis would not have been possible without the support of many people.

First and foremost, I would like to thank my advisor Professor Giancarlo Fortino. His vision, the precious suggestions, the commitment to research, and his invaluable support determined the success of most of my research work. He helped me understanding strengths and weakness of my contributions, providing key ingredients to make my efforts more focused and robust.

A special thank goes to the professors I met at U.C. Berkeley, Ruzena Bajcsy, Roozbeh Jafari, and Edmund Seto. Their great intuition and collaboration had a significant impact on my research path and results.

A very important factor to the development of my research was played by the relationship with industry. During my visit at the Telecom Italia WSN Lab at Berkeley, I met great people to whom I am thankful for the freedom they left me in exploring new research directions; Marco Sgroi and Fabio Bellifemine also gave me fundamental insights for finding practical applications to my research work, without which it would have been less valuable and pleasant. I also would like to thank Roberta Giannantonio and Alessia Salmeri for the uncountable technical discussions, and the great time we had in Berkeley together.

I would like to thank my PhD colleague, but most of all, my friend Antonio Guerrieri, who shared with me many moments of joy (both at a professional and personal level), and the times of frustration (that necessarily happen during the years of a PhD).

A special thank also goes to the many students of our research group, and the ones visiting the WSN Lab. In particular I would like to mention Francesco Aiello, Alessandro Andreoli, Andrea Caligiuri, Stefano Galzarano,

Vitali Loseu, Philip Kuryloski, and Ville-Pekka Seppä. Their precious help in many projects, their support, and collaboration is something I will never forget.

I would like to thank my mother Mariella, my father Gianni, and all my family for their immense support and love.

Finally, I thank my wife Marianna, who most of all knows how many sacrifices my PhD required, especially when thousands of kilometers of land and water keep a family separated.

To my wife, and our lovely daughter

Abstract

The progress of science and medicine during the last years has contributed to significantly increase the average life expectancy. The increase of elderly population will have a large impact especially on the health care system. Furthermore, especially in more developed countries, there is an always growing interest in maintaining, and improving the quality of life.

Wireless Body Sensor Networks (BSNs) can contribute to improve the quality of health care services. BSNs involve wireless wearable physiological sensors applied to the human body for strictly medical and non medical purposes. They can enhance many human-centered application domains such as e-Health, sport and wellness, and even social applications such as physical/virtual social interactions.

However, there are still open issues that limit their wide diffusion in real life; primarily, the programming complexity of these systems, due to lack of high-level software abstractions, and to hardware constraints of wearable devices. In contrast to 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 this aim, this thesis proposes a novel domain-specific approach for programming signal-processing intensive BSN applications. The definition of this approach resulted in a domain-specific programming framework named *SPINE* (Signal Processing in Node Environment) which is one important contribution of this thesis, along with other interesting contributions derived from enhancements and variants to the main proposal. Additionally, to provide validation and performance evaluation of the proposed approach, a number of BSN applications (including human activity monitoring, physical energy expenditure estimation, emotional stress detection, and step-counting) have been developed atop SPINE. These research prototypes showed the effectiveness and efficiency of the proposed approach and improved their respective state-of-the-art. Finally, a *Platform-Based Design* (PBD) methodology, which is widely adopted for the design of traditional embedded systems, is proposed for the design of BSN systems.

Riassunto

Il progresso della scienza e della medicina ha contribuito negli ultimi anni a incrementare significativamente l'aspettativa di vita media. L'aumento della popolazione anziana avrà un forte impatto specialmente sul sistema di assistenza sanitaria. Inoltre, soprattutto nei paesi più sviluppati, si sta assistendo ad un interesse sempre più crescente verso il mantenimento ed il miglioramento della propria qualità di vita.

Le *Reti di Sensori Wireless Indossabili* (BSN) possono contribuire a migliorare la qualità dei servizi sanitari. Le WBSN consistono di sensori fisiologici wireless indossabili applicati al corpo umano. Questa tecnologia può migliorare diversi domini applicativi quali l'e-Health, lo sport ed il fitness, ma anche applicazioni di interazione sociale (fisica e virtuale).

Purtroppo alcuni problemi tutt'ora aperti stanno limitando una capillare diffusione delle BSN nella vita quotidiana; uno dei più critici riguarda la complessità di programmazione di questi sistemi, sia a causa della mancanza di appropriate astrazioni software di alto livello che per le forti limitazioni delle risorse fisiche dei dispositivi sensoristici indossabili. Per affrontare questo problema recentemente sta emergendo un nuovo paradigma di programmazione basato sul concetto di framework domain-specific, che si contrappone alle tecniche di programmazione più tradizionali basate su sviluppo ad-hoc o su middleware general-purpose.

Seguendo questa linea di ricerca, questa tesi propone un innovativo approccio *domain-specific* per la programmazione di applicazioni su BSN. Questo approccio ha portato alla realizzazione di un framework di programmazione domain-specific chiamato *SPINE* (Signal Processing in Node Environment) che rappresenta un importante contributo, insieme ad altri contributi derivati da estensioni e varianti della proposta principale. Per ottenere una validazione ed una valutazione di performance dell'approccio proposto, sono state sviluppate, utilizzando SPINE, diverse applicazioni BSN (tra cui un sistema di monitoraggio di attività fisiche, un'applicazione per la stima del consumo calorico, un sistema per il rilevamento di stati di stress emotivo ed un conta passi intelligente). Questi prototipi di ricerca hanno dimostrato l'effiacia ed efficienza

dell'approccio proposto ed hanno contribuito a migliorare lo stato dell'arte. Infine, per supportare la progettazione di sistemi BSN viene proposta una metodologia di tipo *Platform-Based Design* (PBD), già ampiamente diffusa per la progettazione di sistemi embedded tradizionali.

Contents

List of Figures

List of Tables

List of Abbreviations

A-SPINE	Agent-SPINE
ADC	Analog Digital Converter
ADMR	Adaptive Demand-Driven Multicast Routing
AFME	Agent Factory Micro Edition
ANS	Autonomic Nervous System
API	Application Programming Interface
BDI	Belief, Desire, and Intention
BLE	Bluetooth Low Power
BM	Buffer Manager
BPM	Beat Per Minute
BS	Base-Station
BSN	Body Sensor Network
C-SPINE	Collaborative-SPINE
CBQ	CodeBlue Query
CBSN	Collaborative BSN
CIBO	C-SPINE Inter-BSN Over-The-Air
CLDC	Constrained Limited Device Configuration
CPOD	Crew Physiologic Observation Device
CSMA	Carrier Sense Multiple Access
CSMA/CA	CSMA / Collision Avoidance
ECA	Event-Condition-Action
ECAA	ECA-based Automata
ECG	Electrocardiogram
ED	Event Dispatcher
EEG	Electroencephalogram
EIP	Electrical Impedance Plethysmography
EMG	Electromyogram
FIFO	First In First Out
FIR	Finite Input Response
GF	Global Functions
GPL	General Public License

GSR	Galvanic Skin Response
GV	Global Variables
HRV	Heart Rate Variability
HW	Hardware
IDE	Integrated Development Environment
ISM	Industrial Scientific and Medical
KNN	K-Nearest Neighbor
LF	Local Functions
LV	Local Variables
MA	Mobile Agent
MAC	Medium Access Control
MACC	Mobile Agent Communication Channel
MAEE	Mobile Agent Execution Engine
MAMM	Mobile Agent Migration Manager
MAN	Mobile Agent Naming
MAPS	Mobile Agent Platform for Sun SPOTs
MiLAN	Middleware Linking Applications and Networks
MPSM	Multi-Plane State Machine
NesC	Network Embedded Systems C
OLED	Organic Light-Emitting Diode
OS	Operating System
OTA	Over-The-Air
P2P	Peer-to-Peer
PBD	Platform-Based Design
PDA	Personal Digital Assistant
QoS	Quality of Service
RAM	Random Access Memory
RK	Resource Kernel
RM	Resource Manager
RMS	Root Mean Square
ROM	Read Only Memory
RPC	Remote Procedure Call
RRi	(Heart) R-peak to R-peak intervals
RX	Reception
SNAPP	Sensor Network Implementation Platform
SNIP	Sensor Network Ad-hoc Protocol Platform
SNR	Signal-to-Noise Ratio
SNSP	Sensor Network Service Platform
SPI	Serial Port Interface
SPINE	Signal Processing in Node Environment
SW	Software
TDMA	Time Division Multiple Access
TITAN	Tiny Task Network
TM	Timer Manager
TX	Transmission

UML	Unified Modeling Language
VM	Virtual Machine
VS	Virtual Sensor
VSM	Virtual Sensor Manager
WPAN	Wireless Personal Area Network
WS	Wireless Sensor
WSN	Wireless Sensor Network
ZDO	ZigBee Device Object

1

Motivation, Objectives and Organization of the Thesis

1.1 Motivation

The progress of science and medicine during the last years has contributed to significantly increase the average life expectancy. According to recent studies, in 2050 life expectancy will be 80 and 85 years respectively for men and women and the worldwide population over 65 is projected to increase from 500 million to one billion in 2030. The increase of elderly population will have a large impact especially on the health care system. At the same time, especially in the more developed countries, there is an always growing interest in maintaining, and improving the quality of life, and consequently health and wellness.

ICT technologies and, in particular, *Wireless Body Sensor Networks*(BSNs) possess enormous potential for changing people's daily lives. This technology has gained much interest world-wide, and in Europe in particular. For instance, inside the European Community FP7 - Challenge 5 "ICT for Health, Ageing Well, Inclusion and Governance" there is a specific section on *Personal Health Systems*.

A *BSN* is a collection of wearable (and 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, muscular tension, breathing rate and volume, skin conductivity, and brain activity. The local personal device is typically a smart-phone or a PC, and allows for real-time monitoring as well as long-term remote storage and off-line analysis.

BSNs, therefore, involve wireless wearable physiological sensors applied to the human body for strictly medical and non medical purposes. BSNs allow for continuous and non-invasive measurement of body movements and physiological parameters during the daily life, as these tiny wireless sensors are placed on the skin, and sometimes in the garments.

BSNs can enhance many human-centered application domains such as early detection or prevention of diseases (heart attacks, Parkinson, diabetes, etc.), elderly assistance at home, e-fitness, post-trauma rehabilitation after surgeries, motion and gestures detection, cognitive and emotional recognition, medical assistance in disaster events, etc. 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 domain in which BSNs have a central relevance; applications in such domain aim at monitoring 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 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 so far, none of them have reached the market yet. One of the biggest driving factors for this delay is due to their design and implementation approaches. It is clear that to fully exploit the potential of BSNs in these domains and enable a broader range of advanced assisted living and health-care services, additional research is needed, in particular focusing on two main directions:

1. conceiving new programming abstractions, techniques, and methodologies for improving system design and prototyping;
2. implementing real-time, power efficient distributed signal processing algorithms for data interpretation on wireless nodes that are very resource limited and have to meet hard requirements in terms of wearability and battery duration as well as computational and storage resources.

Implementing real-time, power efficient distributed signal processing algorithms on wireless nodes remains, indeed, 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 pre-defined, 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 functionality and features.

To date, almost all the BSN applications have been developed following two main approaches: low-level programming and general-purpose middleware for Wireless Sensor Networks (WSNs). 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. 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-level 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, as general-purpose frameworks for WSNs are designed to support a wide variety of application scenarios, they are either too general (lacking of abstractions typically needed for BSN systems), or demand too many resources to realize efficient BSN-specific applications.

To this perspective, an emerging research topic is devoted to the design of novel BSN programming approaches based on the concept of *domain-specific* frameworks. The research contributions proposed in this thesis are founded on this concept, with the aim of fulfilling some of the key issues that are currently limiting the full exploitation of the BSN technology.

1.2 Objectives of the Thesis

This thesis proposes a novel domain-specific approach for programming signal-processing intensive BSN applications.

The definition of this approach resulted in a domain-specific programming framework named *SPINE* (Signal Processing in Node Environment) which represents the first contribution of the thesis. Additional interesting research works which are the result of enhancements and variant to the original framework, are discussed in this thesis as well. They include a task-oriented redesign of SPINE, an enhancement for supporting collaborative BSNs, a multi-agent model for BSN programming, and a programming paradigm based on the concept of Virtual Sensors.

Although a number of surveys and review works in the context of BSNs have been published so far, this thesis required a novel analysis of the state-of-the-art which focuses on development tools and middlewares for programming BSN applications, along with a systematic identification of the fundamental requirements and properties which should be satisfied by an effective and efficient BSN programming framework. To the best of our knowledge, such analytic review work was lacking in the literature, and, therefore, it can be considered as the second contribution of this thesis.

The third contribution is represented by a number of case studies developed through the proposed framework. These research prototypes showed the effectiveness and efficiency of the proposed approach and improved their respective state-of-the-art. Specifically, the case studies are: (a) a physical

activity monitoring system, (b) a step-counter, (c) a physical energy expenditure system, and (d) an emotional stress detector. The physical activity monitoring system reaches very high recognition accuracy with fewer wearable sensor nodes than other works. The step-counter application relies on a novel algorithm which is the only one able to correctly recognize the steps taken during walking from healthy, elderly and people affected by walking disabilities. The physical energy expenditure system is able to estimate the calories burnt during daily activities in real-time, without any assumption on the orientation of the worn motion sensor. Finally, the emotional stress detection relies on a wireless system, and a monitoring application that, by means of time-domain heart-rate analysis, provides a stress index using only ten minutes of observations.

The fourth contribution is the definition of a *Platform-Based Design* (PBD) methodology specifically tailored for the BSN domain. PBD has shown to be very effective in traditional embedded system design, both at academic and industrial level. An application of the PBD to the WSN domain has been proposed in the past, providing case studies for building and industrial monitoring. However, this is the first time that PBD is shown to properly address also the system level design of BSNs.

1.3 Structure of the Thesis

This thesis is organized as follows.

Chapter 2 contains a review of the BSN domain. It includes a brief overview of its current hardware technologies, common physical sensors, wearable sensor node platforms, communication protocol standards, and embedded operating systems. The discussion is focused, however, on a review of the state-of-the-art development tools and middlewares for programming BSN applications. A detailed analysis of the requirements and properties for an effective BSN programming framework is also reported, and used as an objective reference for the comparison of the state-of-the-art middlewares.

In Chapter 3, *SPINE*, a novel domain-specific framework for rapid prototyping of BSN applications is presented. The most important contributions of this framework are presented, and analyzed. The design choices and the architecture of both node-side and coordinator-side components of SPINE are described in detail. Furthermore, an in-depth performance evaluation has been carried out, and the main results reported. Finally, the most relevant enhancements and variants to the core branch of the framework are overviewed, including (i) *C-SPINE*, an enhancements for supporting Collaborative BSNs, (ii) *A-SPINE*, an agent-based variant of SPINE, (iii) *SPINE2*, a task-oriented re-design of SPINE, and (iv) the concept of *Virtual Sensors* based on SPINE.

Chapter 4 describes some interesting research case studies of SPINE which include (i) human activity recognition, (ii) step-counter, (iii) physical energy expenditure, and (iv) emotional stress detection. The proposed systems are

based on wireless wearable sensor nodes, and programmed atop the SPINE framework.

In Chapter 5, a *Platform-Based Design* methodology specifically focused on the BSN domain is introduced. The proposed methodology is exemplified through a case study for physical activity recognition based on a *Template Matching* approach.

Finally, Chapter 6 includes a summary of the main results of this thesis, along with some concluding remarks, and comments on possible future research directions that can derive from the work here presented. For the sake of completeness, a list of the publications related to the thesis is also reported.

2

Related Work

Body Sensor Networks (BSNs) possess an enormous potential for changing people's daily lives as they can enhance many human-centered application domains.

This chapter presents an overview of the current state-of-the-art and examines the concepts and the hardware/software technologies behind the BSNs, focusing, in particular, on the software frameworks for programming BSN applications that have been proposed so far.

2.1 Introduction

BSNs systems involve wireless wearable physiological sensors applied to the human body for strictly medical or non medical purposes. This area is particularly dense of interest because foreseen real-world applications of BSNs aim to improve the quality of life by enabling at low cost continuous and real-time non-invasive medical assistance. Applications where BSNs could be greatly useful include early detection or prevention of diseases (e.g. heart attacks, Parkinson, diabetes, asthma), elderly assistance at home (e.g. fall detection, pills reminder), e-fitness, rehabilitation after surgeries (e.g. knee or elbow rehabilitation), motion and gestures detection (e.g. for interactive gaming), cognitive and emotional recognition (e.g. for driving assistance or social interactions), medical assistance in disaster events (e.g. terrorist attacks, earthquakes, wild fires), etc.

The design and programming of applications based on BSNs are complex tasks. That is mainly due to the challenge of implementing signal processing intensive algorithms for data interpretation on wireless nodes that are very resource limited and have to meet hard requirements in terms of wearability and battery duration as well as computational, and storage resources. This is challenging because BSNs applications usually require high sensor data sampling rates which endangers real-time data processing and transmission capabilities as computational power and available bandwidth of the BSN infrastructure

is generally scarce. Indeed, efficient implementation of BSN applications requires appropriate allocation of the limited resources on the nodes in terms of power, memory and processing. This is especially critical in signal processing systems, which usually have large amounts of data to process and transmit.

Current embedded operating systems do not address such high level and complex requirements as they mainly focus on hardware abstraction, power management, routing, security and synchronization algorithms, and sometimes on general-purpose data structures, dynamic memory management, and multi-tasking. That being said, new programming abstractions, methodologies, and tools are needed to support application developers during design space exploration, and prototyping.

This chapter provides a survey on the current hardware/software architectures of the BSN domain. In particular, a classification of the main BSN programming approaches will be given, and the different strategies will be compared on the basis of a novel taxonomy that is proposed in this thesis.

It is worth noting that, to the best of our knowledge, this is a completely novel state-of-the-art work in the BSN domain.

A few survey and review works on wearable sensor-based systems have been published to date. For example, in [1] the focus of the survey is on the functional perspective of the analyzed systems (i.e. what kind of applications they target) as well as on the hardware architecture. In this survey, systems are classified in commercial products and research projects, and also grouped by different approaches on the hardware platforms: wired electrodes-based, smart textiles, wireless mote-based, based on commercial smart-phones and sensors. A comprehensive review of the sensors and their application use is also provided. Another frequently cited survey work on BSNs is [2]. The focus is again on the hardware components (antennas, in particular) and the application scenarios. Described projects are classified in (i) in-body (implantable), (ii) on-body medical and (iii) on-body non medical systems.

However, none of the aforementioned works take deeply into account the software design and the programming approaches of such technologies. From a software standpoint, just the communication protocols and the embedded operating systems have been taken in consideration. Rather, they focus on the hardware and the application use cases. Nevertheless, a significant issue in this field is related to the programming of the embedded devices, due to their resource constraints and the limited availability of software abstractions and tools.

Therefore, the purposes of this chapter are to:

- identify what are the approaches that best fit the complex task of programming BSN-based systems;
- analyze actual implementations to evaluate their development level according to their potentials;

- provide possible direction for further research in the fields that these systems show a lack of performance.

2.2 State-of-the-art on BSNs

2.2.1 Hardware

In this section, a brief overview on the hardware aspects (physical architecture, sensors, and commercial platforms) of a typical wearable sensor node will be given.

2.2.1.1 Physical architecture of a sensor node

A sensor node is capable of gathering sensory information, performing some processing, and communicating with other connected nodes in the network. To perform these functionalities, sensor nodes require a specific hardware architecture (see Figure 2.1). In particular, the main hardware components are:

- a *microcontroller*, for local processing (it also contains a limited amount of RAM memory);
- a *transceiver*, to communicate with other nodes;
- *external memory*, for local persistent storage of sensor data;
- *power source* (typically a battery or a solar panel), to provide energy to the circuitry;
- one or more *transducers* (physical sensors), to acquire raw data from the surrounding environment.

2.2.1.2 Sensors

BSNs support many application domains. As a consequence, extremely diverse physical sensors can be found in BSN systems. The most widely used sensors are:

- skin/chest electrodes for electrocardiogram (ECG) and electrical impedance plethysmography (EIP) to monitor, respectively, heart rate and respiration rate;
- arm cuffs for blood pressure;
- pulse oximeters for oxygen saturation and heart rate monitoring;
- galvanic skin response (GSR) for perspiration (sweating) and emotional recognition;
- microphones for voice, ambient, and heart sounds (placed on the chest or earlobe);
- strip-base glucose meters for blood glucose level monitoring;

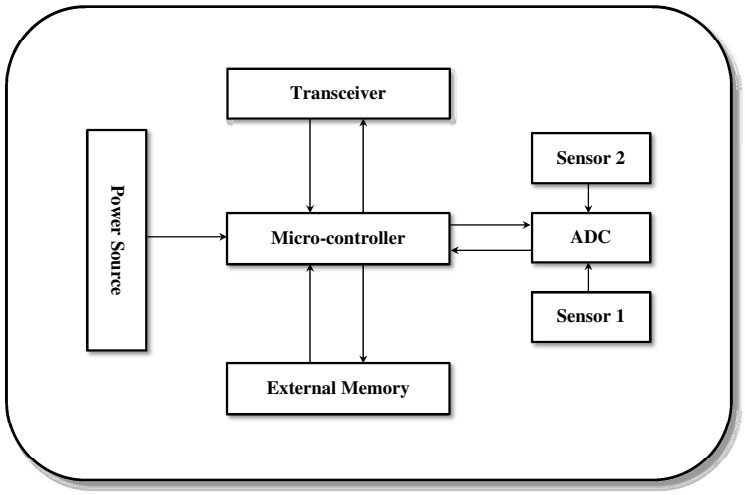


Fig. 2.1. Typical hardware architecture of a sensor node.

- skin electrodes for electromyogram (EMG) to monitor muscle activity;
- scalp-placed electrodes for electroencephalogram (EEG) to monitor brain activity;
- accelerometer, gyroscope, pressure sensors for body movements and applied forces.

2.2.1.3 Commercial Platforms

A comprehensive analysis on the commercial sensor platforms used for BSN applications is out of the scope of this section. However, to provide a general idea on the current status, a brief list is summarized in Table 2.1. An interesting survey on sensor network platforms can be found in [3].

2.2.2 Communication

This section briefly introduces the most popular communication protocols in the BSN domain, and its typical network topologies.

2.2.2.1 IEEE 802.15.4 / ZigBee

IEEE 802.15.4 [4] is to date the most widely adopted standard in the wireless sensor network domain. Indeed, it is intended to offer the fundamental lower network layers (physical and MAC) of *Wireless Personal Area Networks* (WPANs) focusing on low-cost, low-speed ubiquitous communication between

Table 2.1. List of commercial sensor node platforms.

Sensor Platform	MCU	Tranceiver	Code/Data Memory	External Memory	Language
BTnode	ATmega 128L	CC1000, Bluetooth	180/64KB	128 KB	C, nesC/ TinyOS
Epic mote	TI MSP430	CC2420	48/10 KB	2 MB Flash	nesC/ TinyOS
MicaZ	ATmega 128	CC2420	128/4 KB	512 KB	nesC/ TinyOS
Shimmer	TI MSP430	CC2420, Bluetooth	48/10 KB	2 GB microSD	nesC/ TinyOS
SunSPOT	ARM 920T	CC2420	512 KB	4 MB Flash	JavaME
TelosB Tmote Sky	TI MSP430	CC2420	48/10 KB	1 MB Flash	C, nesC/ TinyOS
Waspmote	ATmega 1281	ZigBee	128/8 KB	2 GB microSD	C

devices. The emphasis is on very low cost communication of nearby devices with little to no underlying infrastructure.

The basic protocol conceives a 10-meter communications range with a transfer rate of 250 kbit/s. Tradeoffs are possible to favor more radically embedded devices with even lower power requirements, through the definition of several physical layers. Lower transfer rates of 20 and 40 kbit/s were initially defined, with the 100 kbit/s rate being added in the current revision. Even lower rates can be considered with the resulting effect on power consumption.

As aforementioned, the main identifying feature of 802.15.4 is the importance of achieving extremely low manufacturing and operation costs and technological simplicity, without sacrificing flexibility or generality. Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through *CSMA/CA* and integrated support for secure communications. Devices also include power management functions such as link quality and energy detection. 802.15.4 operates on one of three possible unlicensed frequency bands:

- 868.0-868.6 MHz: Europe, allows one communication channel (2003, 2006);
- 902-928 MHz: North America, up to ten channels (2003), extended to thirty (2006);
- 2400-2483.5 MHz: worldwide use, up to sixteen channels (2003, 2006).

To complete the IEEE 802.15.4 standard, the ZigBee [5] protocol has been realized. ZigBee is a low-cost, low-power, wireless mesh network standard built upon the physical layer and medium access control defined in the 802.15.4. It is intended to be simpler and less expensive than other WPANs such as Bluetooth. ZigBee chip vendors typically sell integrated radios and microcontrollers with between 60 KB and 256 KB flash memory.

The ZigBee network layer natively supports both star and tree typical networks, and generic mesh networks. Every network must have one coordinator device. In particular, within star networks, the coordinator must be the central node. Both trees and meshes allows the use of ZigBee routers to extend communication at the network level.

The ZigBee specification completes the 802.15.4 standard by adding four main components:

1. network layer;
2. application layer;
3. *ZigBee device objects* (ZDO's);
4. manufacturer-defined application objects which allow for customization and favor total integration.

Besides adding two high-level network layers to the underlying structure, the most significant improvement is the introduction of ZDO's. These are responsible for a number of tasks, which include keeping of device roles, management of requests to join a network, device discovery and security.

2.2.2.2 Bluetooth / Bluetooth Low Energy

Bluetooth [6] is a proprietary open wireless technology standard for exchanging data over short distances (using short wavelength radio transmissions in the ISM band from 2400-2480 MHz) from fixed and mobile devices, creating WPANs with high levels of security.

Bluetooth uses a radio technology called frequency-hopping spread spectrum, which chops up the data being sent and transmits chunks of it on up to 79 bands (1 MHz each; centered from 2402 to 2480 MHz) in the range 2,400-2,483.5 MHz (allowing for guard bands).

Bluetooth is a *packet-based* protocol with a master-slave structure. One master may communicate with up to 7 slaves in a *piconet*; all devices share the master's clock. Packet exchange is based on the basic clock, defined by the master.

The Bluetooth Core Specification also provides for the connection of two or more piconets to form a *scatternet*, in which certain devices simultaneously play the master role in one piconet and the slave role in another.

Although being designed for WPANs, the first versions of Bluetooth are actually suitable only for BSN systems that do not require long battery life before recharging. This is because the power consumption profile of Bluetooth

is significantly higher compared with 802.15.4. Other factors limiting the use of Bluetooth in the BSN domain are the high communication latency (typically around 100ms), and the long set-up time (that can reach 6s).

To overcome these limitation, Bluetooth recently released the 4.0 version that has been called ***Bluetooth Low Energy*** (BLE) [7]. One of the BLE design driving factors is the specific support for applications such as healthcare, sport, and fitness. Promoter for such applications is the Bluetooth Special Interest Group in cooperation with the Continua Health Alliance.

BLE operates in the same spectrum range (2402-2480 MHz) as classic bluetooth, but uses a different set of channels. Instead of 79 1 MHz wide channels, BLE has 40 2 MHz wide channels.

BLE is designed with two equally important implementation alternatives: *single-mode* and *dual-mode*. Small devices like watches, and sports sensors based on a single-mode BLE implementation will take advantage of the low-power consumption, and low-production costs. However, pure BLE is not backward compatible with the classic Bluetooth protocol. In dual-mode implementations, instead, the new low energy functionality is integrated into Classic Bluetooth circuitry. The architecture will share Classic Bluetooth technology radio and antenna, enhancing currently chips with the new low energy stack.

2.2.2.3 ANT

ANT [8] is an ultra-low power wireless communications protocol stack operating in the 2.4 GHz band. A typical ANT protocol transceiver comes pre-loaded with the protocol software and must be controlled by an application processor. It is characterized by a low computational overhead, and high efficiency resulting in low power consumption by the radios supporting the protocol.

Similarly to BLE, ANT, among other WSN application scenarios, has been targeted for sport, wellness, and home health monitoring. To date, indeed, ANT has been adopted in a number of commercial wrist-mounted instrumentation, heart rate monitoring, speed and distance monitoring, bike computers, health and wellness monitoring devices.

2.2.2.4 IEEE 802.15 WPAN Task Group 6 (TG6) - Body Area Networks

The IEEE 802.15 Task Group 6 (BAN) [9] is developing a communication standard specifically optimized for low power devices operating on, in or around the human body to serve a variety of applications including medical, consumer electronics, personal entertainment, and others.

Compared to IEEE 802.15.4, IEEE 802.15.6 focuses specifically on BSNs, addressing their identifying characteristics such as shorter communication range (the standard supports a range of 2-5 meters), and larger data rate (up to 10Mbps), which helps decreasing power consumption, and meeting safety and bio-friendly requirements, since the working environment is related to human health.

2.2.2.5 Network Topologies

The most common network topologies adopted in the BSN domain are the following:

- peer-to-peer;
- star;
- mesh;
- clustered.

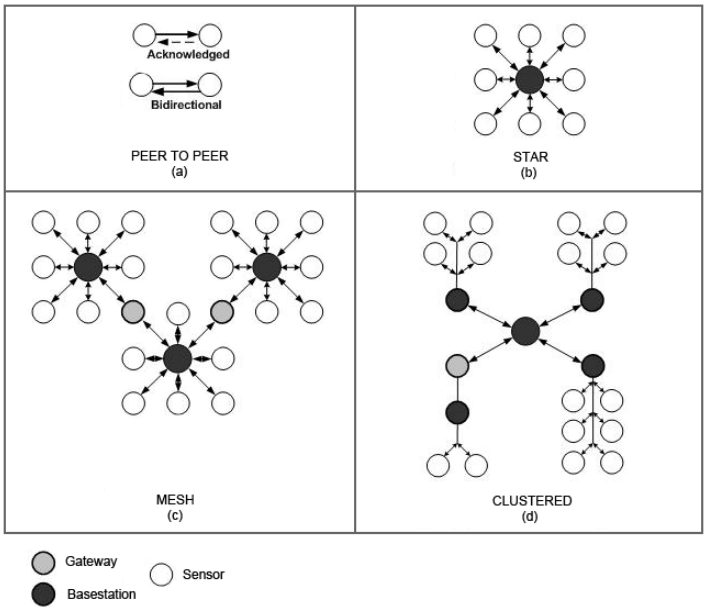


Fig. 2.2. Common BSN topologies.

The *peer-to-peer* (P2P) topology (Figure 2.2(a)) reflects BSN systems that do not rely on coordinator station to operate. It is worth noting that a pure P2P topology is never used in practice today. In fact, even for systems in which the sensor nodes interacts directly, adopting a decentralized communication paradigm to reach a certain shared goal, there is always one node (at least) that plays the role of interfacing with the user, to receive commands, and provide some sort of feedback for the events generated by the BSN.

The most common topology for a BSN system is actually the *star* (Figure 2.2(b)). Here, the base-station (BS) (i.e. the coordinator device such as a smart-phone, a PDA or also a PC) acts as the center of the star and it is in charge of configuring the remote sensor nodes (which do not communicate among each other directly), and gathering the sensory information.

These topologies are used for personal BSN applications (e.g. health monitoring, wellness, or sport) that do not need to interact with other BSNs.

The *mesh* topology (Figure 2.2(c)) is an extension of the star, where multiple BSNs may interact, and even collaborate, through the existence of an underlying infrastructure consisting of gateway nodes necessary to enable the communication among BSs.

A somewhat similar topology is the *clustered* (Figure 2.2(d)). Here, however, different BSNs may communicate without relying necessarily on a fixed infrastructure. In other words, the BSs are able to communicate directly, typically in a peer-to-peer fashion.

Mesh and clustered topologies are adopted in complex systems which involve different BSNs to communicate among each other. Depending on the specific application, they are often referred as **Collaborative BSNs**, and are discussed in more details in Section 3.10.1.1.

2.2.3 Operating systems

The design of operating systems for WSNs deviates from traditional operating system design due to significant and specific characteristics of WSNs like constrained resources, high dynamics and inaccessible deployment.

This section provides a brief overview of the most popular operating systems for WSN nodes. A comprehensive survey can be found in [10].

2.2.3.1 TinyOS / nesC

TinyOS [11] is an *event-driven* operating system which provides a programming environment for embedded systems. It has a component-based execution model implemented in the nesC language [12] which has a very low memory footprint.

TinyOS concurrency model is based on commands, asynchronous events, deferred computation called tasks and *split-phase* interfaces. The function invocation (as command) and its completion (as event) are separated into two phases in interfaces provided by TinyOS. Application user has to write the handler which should be invoked on triggering of an event. Commands and Event handlers may post a task, which is executed by the TinyOS FIFO scheduler. These tasks are non preemptive and run to completion. However tasks can be preempted by events but not by other tasks. Data race conflicts that arise due to preemption can be solved using atomic sections.

Radio communication in TinyOS follows the *Active Messages* [13] model, in which each packet on the network specifies a handler ID that will be invoked on recipient nodes. The handler ID as an integer that is carried in the header of the message. When a message is received, the receive event associated with that handler ID is signaled. Different sensor nodes can associate different receive events with the same handler ID.

2.2.3.2 Contiki

Contiki [14] is an operating systems which combines the advantages of both events and threads. It has primarily an event-driven model but supports multi-threading as an optional application level library. Application can link this library if it needs multi-threading. Events are classified as asynchronous and synchronous in Contiki. Synchronous events are scheduled immediately and asynchronous events are scheduled later. A polling mechanism is used to avoid race conditions.

In Contiki device drivers and sensors data handling are implemented as service. Communication is also implemented as a service in order to enable run-time replacement. Implementing communication as a service also provides for multiple communication stacks to be loaded simultaneously. Each service has an interface and implementation components. Applications are aware of only the interfaces. This way, the service implementation can be changed also at run time. This is done by stub library which is linked with the application for accessing services.

2.2.3.3 MantisOS

MantisOS [15] is a *thread-driven* operating system model for sensor networks. Thread-driven model gives flexibility in writing applications as the developer is not concerned about task size which is mandatory in event-driven model. Execution of an application involves spawning multiple threads. Network stack and scheduler are also implemented as threads just like an application. Apart from these threads, there is an *idle* thread which runs when all other threads are blocked. Idle thread invokes the required power management routines. To maintain threads, kernel maintains a thread table that consists of thread priority, pointer to thread handler and other information about the thread. Scheduling between the threads is done by means of scheduler that follows priority based scheduling algorithm with round-robin semantics. Race conditions are avoided by using binary and counting semaphores. In MantisOS, communication is realized as a layered network stack. The stack is designed to minimize memory buffer allocation through layers; it supports layer three and above, i.e. network layer routing, transport layer, and application layer, while MAC protocol support is performed by another communication layer, which is located in a separate lower layer of the OS, distinct from and below the user-level networking stack.

One of the main drawbacks of adopting a thread-driven model leads MantisOS to suffer from the overheads of context switching and the memory allocated (in the form of stack) per each thread.

2.2.3.4 NanoRK

Nano-RK [16] is a reservation-based, *multi-tasking*, energy-aware real-time operating system for WSNs, and it is implemented as an extension of the

Resource-Kernel (RK) [17] paradigm. Tasks are associated with priorities, and higher-priority tasks always preempt the lower priority ones. For time sensitive tasks in applications, it implements rate-monotonic scheduling algorithm for the tasks, so that the deadlines of the tasks are fulfilled. Applications can define their resource requirements and deadlines to meet.

Nano-RK contains a lightweight network protocol stack that allows for port-based communication. Since the network stack is tightly integrated with the OS and execution/communication information is available, optimizations using global application knowledge such as automatic packet aggregation, network reservations, and buffer management are possible. An incoming data packet triggers an interrupt that handles the arrival of the packet. Packet transmissions are handled by a periodic network task responsible for servicing all outgoing packets of all tasks.

2.2.3.5 Java Squawk VM

Squawk is a Java micro edition virtual machine (VM) for embedded system and small devices [18]. It is different from most of the other virtual machines for the Java platform (which are written in low level native languages such as C/C++ and assembler) because Squawk's core is mostly written in Java. A Java implementation provides ease of portability, and a seamless integration of virtual machine and application resources such as objects, threads, and operating-system interfaces.

The Squawk VM targets small, resource constrained devices, and enables Java for micro-embedded development. The *SunSPOT* [19] is, probably, the most popular sensor platform built on this VM.

The fundamental programming paradigm is based on the concept of *Isolate*. An isolate is a mechanism by which an application is represented as an object. In Squawk, one or more applications can run in the single VM completely isolated from each other.

In addition to the standard semantics of isolates, the Squawk implementation has one very unique extra feature among the WSN operating systems: the *isolate migration*. An isolate running on one Squawk VM instance can be paused, serialized to a file or over a network connection and restarted in another Squawk VM instance.

The current version of the Sun SPOT SDK uses the GCF (Generic Connection Framework) to provide radio communication between SPOTs, routed via multiple hops if necessary. In particular, two protocols, implemented on top of the MAC layer of the 802.15.4 implementation, are available: the *radiostream* protocol and the *radiogram* protocol. The radiostream protocol provides reliable, buffered, stream-based communication between two devices. The radiogram protocol provides datagram-based communication between two devices and broadcast communications. This protocol provides no guarantees about delivery or ordering.

2.2.3.6 Z-Stack

Z-Stack [20] is the Texas Instrument ZigBee compliant protocol stack for a growing portfolio of IEEE 802.15.4 products and platforms. Z-Stack supports both ZigBee and ZigBee PRO feature sets on the CC2530 System-on-Chip, and MSP430+CC2520 platforms. In particular, it supports the Smart Energy and Home Automation profiles. A very interesting feature of this environment is the support for over-the-air sensor node programming.

2.2.4 Applications

As aforementioned, BSNs enable a very wide range of application scenarios. We can categorize them into different application domains:

- e-Health;
- e-Emergency;
- e-Entertainment;
- e-Sport;
- e-Factory;
- e-Sociality.

e-Health applications include physical activity recognition, gait analysis, post-trauma rehabilitation after surgeries, cardiac and respiratory diseases prevention and early detection, remote elderly assistance and monitoring, sleep quality monitoring and sleep apnea detection, and emotion recognition [21].

e-Emergency applications include BSN systems to support firefighters, response teams in large scale disasters due to earthquakes, landslides, terrorist attacks, etc. [22].

e-Entertainment domain refers to human-computer interaction systems typically based of BSNs for real-time motion and gesture recognition [23].

e-Sport applications are related to the e-Health domain, although they have a non-medical focus. Specifically, this domain includes personal e-fitness applications for amateur and professional athletes, as well as enterprise systems for fitness clubs and sport teams offering advanced performance monitoring services for their athletes [24].

e-Factory is an emerging and very promising domain involving industrial process management and monitoring, and workers safety and collaboration support [25].

Finally, *e-Sociality* applications may use BSN technologies to recognize user emotions and cognitive states to enable new forms of social interactions with friends and colleagues. An interesting example is given by a system that involves the interaction between two people's BSNs to detect handshakes and, subsequently monitor their social and emotional interactions [26].

2.3 Development tools and middlewares

This section describes the main methodologies and tools for developing BSN applications.

A novel contribution of this thesis is related to the identification of the key requirements, techniques, and properties that should be satisfied by an effective middleware for the development of BSN systems. This classification has been helpful to provide a detailed taxonomy of the state-of-the-art middlewares for BSNs.

2.3.1 Classification of BSN programming approaches

Programming BSN applications is a complex task mainly due to the hard resource constraints of wearable devices and to the lack of proper and easy to use software abstractions.

Three main programming methodologies are adopted for the development of BSN applications; they are based on:

1. *application-specific* approach;
2. *general-purpose* middleware for WSN;
3. *domain-specific* frameworks for BSN.

The *application-specific* approach consists of developing prototype applications on BSN nodes as a monolithic block assembling low level services, reusable components and application-specific logic. As a result, the software is poorly reusable and difficult to extend. Moreover, the risk of introducing bugs is significantly high and the debugging can be a very time-consuming process.

The second approach is based on *general-purpose middleware*. A middleware is a software layer consisting of a set of services implemented on the sensor platform and sometimes across a network. It hides the complexity of low system and network layers and provides proper abstractions and interfaces to the application layers. In this way application developers can focus on the application logic without dealing with the implementation details of the underlying services. As a consequence, development time is generally shortened. Furthermore, if the middleware is well optimized, the overall system could even reach higher performance. While in traditional distributed systems a number of general-purpose middlewares such as CORBA, DCOM and RMI have been widely used thanks to their ability to work well for very different applications, current general-purpose middlewares for WSN (e.g. Agilla [27], DFuse [28], TAG [29], Mires [30]) are usually too general to be effective or demand too many resources to be implemented on sensor node platforms.

The third programming approach combines the best characteristics of the other two: it is based on frameworks that include *domain-specific* libraries and tools that can be easily reused for multiple applications of a selected domain. This approach, named domain-specific, allows reducing design time

through modularity and reuse, while offering solutions that are optimized for the target domain. For example, most BSN systems need signal processing intensive tasks such as signal filtering, feature extraction, decision support tools. Security and data encryption must be taken into account as most of the communication involves sensible medical data of the user, such as his heart condition, the glucose level, the physical activity being performed, or simply his location. Device discovery and advertising are considered very useful, due to the variety of physical sensors and functionalities supported by the wearable devices. Scalability and flexibility are also important, to allow higher performance (e.g. more accurate event classification having availability of more data sources) and easier updates in the system requirements. Last but not least, many BSN applications need to handle multiple sensor signals at the same time, which is sometime referred as sensor data fusion and context awareness (e.g. accelerometer, electrocardiogram and location data to identify physical activity).

The prototyping of BSN applications can be significantly facilitated by a domain-specific framework with libraries and protocols that allow to implement signal processing tasks efficiently and that address the aforementioned features.

In the following, for each of the identified categories, a brief description of some of the most relevant examples is provided.

2.3.2 BSN Systems implemented with the Application- Specific approach

Most of the literature on BSN systems focus on the functional and system perspective, and a few details are typically given on the software design and architecture; sometime the software side of the system is completely left uncovered. In the absence of a systematic description of such aspects, we have to assume that the software engineering has not been given enough importance while developing the system. Hence, we can definitely state that currently most of the BSN implementations are based on application-specific code.

In the following, two recent BSN research projects and two BSN-inspired commercial products will be briefly introduced. A summary of some literature BSN systems is reported in Table 2.2. A comprehensive overview of several BSN applications can be found in [1][31].

2.3.2.1 Real-time Arousal Monitor

An interesting research work regarding the design of a real-time arousal monitor has been presented in [32]. *Arousal* is a physiological and psychological state of being awake or reactive to stimuli. The goal of the study aims to monitor the level of arousal on a continuous scale, and continuously in time. It is different from other works in that it focuses on light-weight algorithms

Table 2.2. Summary of some well-known BSN systems.

Project Title	Application Domain	Sensors Involved	Hardware Description	Node Platform	Communication Protocol	OS / Programing Language
Real-time Arousal Monitor	Emotion recognition	ECG, Respiration, Temp., GSR	Chest-belt, skin electrodes, wearable monitor station, USB dongle	custom	Sensors connected through wires	n.a. / C-like
LifeGuard	Medical monitoring in space and extreme environments	ECG, Blood Pressure, Respiration, Temp., Accelerometer, SpO_2	Custom microcontroller device, commercial bio-sensors	*XPod* signal conditioning unit	*Bluetooth*	n.a.
Fitbit®	Physical activity, sleep quality	Accelerometer	Waist/wrist-worn device, PC USB dongle	*Fitbit®* node	RF proprietary	n.a.
VitalSense®	In-and on-body Temperature, Physical Activity, Heart monitoring	Temp., ECG, Respiration, Accelerometer	Custom wearable monitor station, wireless sensors, skin electrodes, ingestible capsule	*VitalSense®* monitor	RF proprietary	Windows Mobile
LiveNet	Parkinson symptom, epilepsy seizure detection	ECG, Blood Pressure, Respiration, Temp., EMG, GSR, SpO_2	PDA, microcontroller board	custom physiological sensing board	wires, 2.4GHz radio, GPRS	Linux (on PDA)
AMON	Cardiac-respiratory diseases	ECG, Blood Pressure, Tem., Accel., SpO_2	Wrist-worn device	custom wrist-worn device	Sensors connected through wires - GSM/UMTS	C-like / JAVA (on the server station)
MyHeart	Prevention and detection of cardio vascular diseases	ECG, Respiration, Accelerometer	PDA, Textile sensors, chest-belt	Proprietary monitoring station	conductive yarns, Bluetooth, GSM	Windows Mobile (on the PDA)
Human++	General health monitoring	ECG, EMG, EEG	Low-power BSN nodes	ASIC	2.4GHz radio / UWB modulation	n.a.
HealthGear	Sleep apnea detection	Heart Rate, SpO_2	Custom sensing board, commercial sensors, cellphone	custom wearable station	Bluetooth	Windows Mobile (on the mobile phone)
TeleMuse®	Medical care and research	ECG, EMG, GSR	*Zigbee* wireless motes	proprietary	IEEE 802.15.4 / *Zigbee*	C-like
Polar® Heart Rate Monitor	Fitness and exercise	Heart Rate, altimeter	Wireless chest-belt, watch monitor	proprietary watch monitor	*Polar OwnCode®* (5 kHz) – coded transmission	n.a.

that yield a continuous estimation of the non-discretized arousal level, having the potential of being integrated as a part of the BSN.

The system detects four signals that are known to be directly influenced by a subject's state of arousal through the activation of the *Autonomic Nervous System* (ANS) - being ECG, respiration, skin conductance, and skin temperature. ECG is recorded through a proprietary chest belt for bio-potential read-out. Also integrated in the belt is a piezoelectric film sensor used to measure respiration. Skin conductance is measured at the base of two fingers, by measuring the electrical current that flows as a result of applying a constant voltage. Skin temperature is measured at the wrist, by using a commercially available digital infrared thermometer module.

Data is received wireless by the basestation that is connected to a PC via USB. An interface to Matlab has been realized to enable a platform for quick development and verification of real-time physiological signal processing algorithms. Interpolation of missing data is also present.

The main weakness of this proposal is the lack of a good reference which makes difficult to quantify the quality of the analysis. Furthermore, all experiments have been performed in a controlled environment. Further experiments

are needed before conclusions can be drawn about the extension of the results to non-controlled environments.

2.3.2.2 LifeGuard

Another research work that is worth mentioning is *LifeGuard* [33]. The goal of this effort was to design a small, light weight, wearable, ergonomic device for a NASA research that not only records and streams a comprehensive set of diagnostic-quality physiologic parameters, but can also record body position and orientation, acceleration in three axes, and can be used to mark events. This feature set, combined with wearability, alarm indicators, fault detections, and the ability to stream data to hand-held Bluetooth-enabled devices, forms a compact and reliable system.

The LifeGuard system consists of the "Crew Physiologic Observation Device" (CPOD) and a portable basestation computer. The CPOD device, the core component of the system, is a custom-made, small, lightweight, easy-to-use device that is worn on the body along with the physiologic sensors described below. It is capable of logging physiologic data as well as transmitting data to a portable basestation computer for display purposes and further processing. Most physiologic parameters (ambient and skin temperature, ECG, respiration rate, pulse oximetry, blood pressure) supported by LifeGuard are measured with sensors that are external to the CPOD wearable device, and that can be connected to it via wired plugs. The only sensors that are integrated into the CPOD are the accelerometers.

The user cannot change the basic operational mode of the CPOD. The authors motivate this design choice as an additional security that the system meets the high reliability standards of medical monitoring devices. However, the 6 hours streaming (or 20 hours logging) battery lifetime make LifeGuard hardly usable in non-critical environments. Additionally, as the system cannot be re-configured easily, re-programming is necessary every time some of the tunable system properties (the sensors to enable, or the sensor sampling rate to name a few) need to be modified.

2.3.2.3 FitBit

Fitbit [34] is a promising commercial product thanks to its relatively low cost, the various supported features, and its graceful cosmetic design.

Fitbit consists of a wearable small sensor device, a basestation connected to a PC, and a web-based application used to record, visualize and analyze collected data. Through an embedded accelerometer, Fitbit allows the estimation of number of steps taken during walking or running, the distance traveled, the calories burned during physical activities such as walking, jogging, and other daily life activities. It provides statistics about daily levels of activity (sedentary, lightly active, fairy active, very active). All these information are available in real-time on a tiny OLED display placed on one side of

the wearable device. Fitbit also gives information about the quality of sleep (e.g. how long did it take to fall asleep, how long was the sleep, how many times the user has awakened).

To work properly, it must be worn on the waist or the chest, or, while sleeping, with the provided wristband. Recorded data are temporarily stored locally up to 7 days, and transmitted via wireless to the basestation automatically as soon as the wearable device comes in the range of 15 feet. The declared battery lifetime is 5-10 days of continuous use. Because each device has an unique ID, a single basestation can be used to gather information from multiple Fitbit devices in the nearby. An Internet connection is needed to send the recorded data to a dedicated server. An intuitive and user-friendly web-based application allows the user tracking historical statistics, and visualizing graphically his/her personal progress from time to time. Although the system does not provide any way to dump data to the PC, the website will have an extensive XML and JSON API to access most of stored data.

2.3.2.4 VitalSense

Another commercially available physiologic monitoring system is *VitalSense* [35]. The VitalSense system includes different types of wireless sensors, a small monitor equipped with control buttons and a graphical display, and a software running on the PC.

VitalSense is designed to monitor temperature, in active or inactive subjects and in indoor and outdoor environments. Each VitalSense monitor can receive transmissions from up to 10 miniature, wireless, temperature sensors. Core temperature is sensed by small ingestible capsules. Dermal temperatures are recorded from hypoallergenic adhesive dermal patches. Both sensor types are disposable, but designed for multi-day use under demanding physical and environmental conditions. In addition, the VitalSense telemetric physiological monitoring system can be interfaced to the so called *VitalSense-XHR* sensor: a compact device that transmits over the air Heart Rate and Respiration Rate (which is derived from the ECG). This sensor enables researchers to monitor heart rate and respiratory rate on moving subjects. The XHR is a chest-worn wireless water-resistant physiological monitor that incorporates an ECG-signal processor. The rechargeable battery in the XHR provides four days of battery life on a full charge.

Furthermore, VitalSense can also be interconnected with the *Actical* device. Caloric expenditure, and counting of the steps taken during walking are calculated from movement recorded by the Actical device worn on the waist.

VitalSense is supported by a software utility that communicates with the portable monitor station via an RS-232 cable. The software enables to setup the monitor and initialize the sensors. It also allows real-time temperature data monitoring as well as retrieval - and offline visualization - of recorded data from the monitor.

2.3.3 Domain-specific frameworks for BSNs

Domain-specific frameworks are novel software systems following an approach in the middle between application-specific code and general-purpose middleware. They specifically address and standardize the core challenges of WSN design within a particular application domain. While maintaining high efficiency, such frameworks allow for a more effective development of customized applications with little or no additional hardware configuration and with the provision of high-level programming abstractions tailored for the reference application domain.

2.3.3.1 CodeBlue

One of the most relevant, and probably the first attempt to define a general platform able to support various BSN applications is *CodeBlue* [36]. CodeBlue consists of a set of hardware wearable medical sensor nodes (pulse oximeter, ECG, EMG, accelerometer, gyroscope) based on the Telos and MicaZ motes and a software framework running on TinyOS specifically designed for integrating these 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. CodeBlue is based on a *publish/subscribe*-based data routing framework in which sensors publish relevant data to a specific channel and end-user devices subscribe to channels of interest. It includes a naming scheme, a multi-hop communication protocol, authentication and encryption capabilities, location tracking and in-network filtering and aggregation. CodeBlue provides end-user devices with a query interface for retrieving data from previously discovered sensor nodes.

The focus of the framework is 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. That explains the relevance given to the use of a multi-hop routing protocol.

Because publishers and subscribers are not necessarily within radio range, the CodeBlue routing layer uses the *Adaptive Demand-Driven Multicast Routing* (ADMR) protocol [37]. In order for CodeBlue nodes to discover each other and determine the capabilities of each sensor device, a simple discovery protocol is layered on top of the ADMR framework. Each CodeBlue node periodically publishes metadata about itself, including node ID and sensor types that it supports, to the broadcast channel. Receiving devices that wish to learn about other nodes in the network can subscribe to the broadcast channel to receive this information. The *CodeBlue Query* (CBQ) layer allows receiving devices to establish communication pathways by specifying the sensors, data rates, and optional filter conditions that should be used for data transfer. A CBQ query is generated by an end-user device and instructs CodeBlue nodes to publish data that meets the query conditions on a specific ADMR channel.

Queries can be issued using the provided end-user GUI. A Query is specified by a set of node IDs that should report data for this query, the sensor type representing a specific physiological sensor, the sampling grate, an optional count of the total number of samples to retrieve from each node, and an optional filter predicate which can be used to suppress transmission of sensor data when the predicate condition is not met. It must be noted, however, that no more than two sub expressions can be included in the predicate.

A *GenericSensor* interface is used to abstract the details of acquiring data from each sensor type. Like the standard TinyOS ADC interface, GenericSensor provides a simple split-phase interface. The set of sensor types supported by CBQ on a particular device is configured at compile time with a set of programmer-s flags. These flags cause the appropriate sensor modules to be automatically wired to the CBQ component and included in the sensor metadata advertisements. In this way the binary for a sensor node will only include the components necessary for the sensors actually present. CodeBlue also provides a Java-based GUI that is intended to be easy for medical personnel to use and which provides enough details on patient status and location to identify trends.

Although CodeBlue provides a sensor driver abstraction architecture allowing an easy integration of new sensors within the system, selection of sensor types or physical node identifiers as data sources, tuning of the data rate and definition of threshold-based filters to avoid unnecessary data being transmitted, it does not allow inserting complex signal processing functionalities into the sensor nodes. It supports just simple threshold-based triggers on the sensor readings that do not give enough flexibility for the variety of requirements of the BSN applications.

2.3.3.2 RehabSPOT

RehabSPOT [38] is a customizable wireless networked body sensor platform for physical rehabilitation. RehabSPOT is built on top of the *SunSPOT* sensor platform [19] from Sun Microsystems. The platform consists of a number of SunSPOT nodes attached to various parts of human body, and a SunSPOT basestation connected to a PC. To enforce a high degree of system configurability and reliability, both the wearable nodes and basestation are powered by a flexible software architecture. The RehabSPOT software architecture includes the following features:

- A sensor management module enabling sensor addition/removal and adjustable sampling rate for each sensor during runtime;
- An exception handler inside each SunSPOT node for sensor failure detection;
- A device discovery manager installed in both remote nodes and basestation for dynamic network construction during runtime;
- Use of the on-board flash memory for multi-hop routing and local data storage;

- Adaptive data collection and display according to the number of remote nodes and the types of active sensors.

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 functions during runtime.

RehabSPOT is organized in a three-tier architecture. The first tier consists of all the remote nodes that are organized as a standalone mesh network. A basestation connected to PC along with remote nodes in its neighborhood compose the second tier. This tier forms a star network where the basestation acts as the master node. The basestation streams the data to the program running on the PC for real-time display and on-line processing. The third tier relies on the established Internet infrastructure. In this tier, data stored inside the PC can be transmitted to a remote server for further processing.

The system hardware includes a custom-designed signal conditioning accessory board to facilitate interfacing external sensors with the wearable SunSPOT nodes. The system software is based on client-server architecture.

The server program is installed and running on the 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 basestation to support dynamic BSN construction has been introduced. Each message contains a source address, a message type-code, and data payload. The size of payload varies among different message types. The communication security is enforced by utilizing an efficient pure Java cryptographic library which supports key exchange and digital signatures based on the Elliptic Curve Cryptography (ECC).

2.3.4 General-purpose frameworks for WSNs applied to BSNs

A number of frameworks for generic WSNs have been proposed so far. The goal of such frameworks is to raise the abstraction level given by the current operating systems for WSNs by providing developers with programming models and paradigms (e.g. based on agents or tasks), and general-purpose libraries and APIs to reduce the application development and testing time.

This section describes the general-purpose frameworks for WSNs that have been customized and used to develop BSN-based applications.

2.3.4.1 TITAN

Titan (Tiny Task Network) [39] is a general-purpose middleware that supports implementation and execution of context recognition algorithms in dynamic

WSN environments. Titan represents data processing by a data flow from sensors to recognition result. The data is processed by tasks, which implement computations like classifiers or filters. The tasks and their data flow interconnections define a task network, which runs on the sensor network as a whole. The tasks are mapped onto the single sensor nodes according to the sensors and the processing resources they provide.

Titan dynamically reprograms the sensor network to exchange context recognition algorithms, handle defective nodes, variations in available processing power, or broken communication links. It has been designed to run on resource constrained sensor nodes and implemented in TinyOS on Tmote Sky motes.

The goal of Titan is to provide a mechanism to dynamically configure a data processing task network on a wireless sensor node network. Titan provides a set of tasks, of which each implements some signal processing function. Connections transport the data from one task to another. Together, the tasks and connections form a task network, which describes the application to be run on the wireless sensor network. Programming data processing in this abstraction is likely to be more intuitive and less error prone than writing sequential code. The inner working of every processing task have to be thoroughly checked only once. This can be done in an isolated way and reduces the complexity of debugging. Tasks have a set of input ports, from which they read data, and a set of output ports to which task processing results are delivered. Connections deliver data from a task output port to a task input port and store the data as packets in FIFO queues. The application is issued by a master node. It analyzes the task network and splits it into task sub-networks, which are to be executed on the individual nodes. The connections between tasks on different nodes is maintained by sending the packets in messages via a wireless link protocol.

2.3.4.2 AFME

AFME (Agent Factory Micro Edition) [40] is a minimized footprint intelligent agent platform for ubiquitous devices, originally developed for use with 3G mobile phones, and recently deployed for SunSPOT sensor nodes.

AFME is based on the *Belief, Desire, and Intention* (BDI) constructs [41], and on a declarative agent programming language in conjunction with imperative components. AFME agents are imbued with mechanisms that enable them to interact with their environment. Agents perceive and act upon the environment through perceptors (software sensor components) and actuators respectively. Perceptors and actuators represent the interface between the agent and the environment and are implemented in Java.

AFME also supports a form of agent migration. Migration refers to the process of transferring an agent from one platform to another. Agent migration is often classified as either *strong* or *weak*. This classification is related to the amount of information transferred when an agent moves. The strongest form

of migration possible requires the transfer of the entire internal stack of the process in question. This is not possible in heterogeneous environments, where the internal stacks are implemented differently. In such environments, weaker forms of mobility are necessary.

Truly strong migration is not possible in Java, since much of an application state is under the control of the JVM. Within AFME, support is only provided for the transfer of the agent mental state. Any classes required by the agent must already be present at the destination. This is because the Constrained Limited Device Configuration (CLDC) does not contain an API for dynamically loading foreign objects. In the Squawk JVM, it is possible to migrate an application to another Squawk enabled device. Squawk implements an isolate mechanism, which can be used for a type of code migration. However, Isolate migration is not used in AFME as with isolates, it would be necessary to migrate the entire application or platform, rather than just a single agent.

AFME has been used for the development of an activity monitoring system based on BSNs [42].

2.3.4.3 MiLAN

MiLAN (Middleware Linking Applications and Networks) [43] is a middleware for sensor networks whose goal is maximizing application lifetime while providing application QoS by controlling and optimizing network as well as sensors.

One of the application scenarios where the authors believe it can be useful is the medical monitoring. Early studies involved the introduction of MiLAN in the University of Rochester - Center for Future Health's Smart Medical Home project, particularly for heart monitoring.

The distinctive feature of its approach is the integration of network control into the middleware, enabling application-directed network reconfiguration. From the application, Milan receives a set of performance specifications with respect to different system components as well as information specifying how different applications should interact. From the network, Milan monitors for available components and overall resources such as power consumption and bandwidth. Combining this information in some optimal manner, Milan continuously adapts the network configuration to best meet the application needs and balance performance for cost.

The fundamental concept of MiLAN is that while the policy of how to manage and control the dynamic network should be left to the application, the mechanisms for implementing the policy should reside at the middleware. Indeed, the aim of this generic middleware is to provide this type of policy/mechanism split by allowing the application to specify its low-level data needs, how these needs vary over time, and how its performance is affected by different sets of input data. Knowing this information allows the middleware

to manipulate the network configuration, taking into account the trade-offs between application performance, data availability, and energy consumption.

An implementation of MiLAN has been provided for Bluetooth and IEEE 802.11 wireless networks. However, although the aim of the framework is to manage QoS in distributed sensor networks, the framework implementation runs in a centralized fashion.

2.3.5 Requirements, techniques and properties for BSN programming frameworks

A software framework designed for supporting fast prototyping of BSN applications should meet specific requirements in terms of effectiveness, efficiency, and usability. Adopting a BSN-specific framework should result, with less efforts in terms of development time and application-level programming complexity, in accurate and efficient applications, whose source code is more modular and easier to maintain.

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 2.3 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 runtime, 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 for supporting fast prototyping of efficient BSN applications should meet specific (functional and non-functional) 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 non-functional requirement in the context of BSN applications, as they usually involve processing and communication of data acquired from the human body. Such data is inherently personally identifiable, and may be medically relevant and therefore highly privacy sensitive.

In the following, such requirements are extensively discussed. Table 2.4 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.

Table 2.3. 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 (PRE)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 SYNCHRONIZATION	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.

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:

Table 2.4. 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

- *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: (i) 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 multi-channel sensor (e.g. only some of the axes of a three-axial accelerometer or gyroscope); (ii) flexible data structures to easily accommodate different data types (e.g. short, int, long values), so that, for instance, the same buffer might be configured for storing data from sensors that produce samples with different word-length; (iii) flexible communication API, as different applications may require different data payload structure and length; (iv) 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 to allow customized trade-off 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.
- *Software engineering methods* aim at supporting 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. FIR 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, user data transmission).

- *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 interoperate 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 smart-phone, 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.

2.3.6 Comparison of the WSN/BSN programming frameworks

The in depth analysis of the requirements, properties, and techniques that are absolutely necessary for a framework to be effective and efficient in supporting BSN applications programming, has been helpful for providing an accurate comparison of the current state-of-the-art on the WSN/BSN programming frameworks. To the best of our knowledge, this is a novel research contribution in this context.

The result of the comparison is reported in Table 2.5.

2.3.7 Summary

This chapter provided an overview of the current state-of-the-art of the BSN domain. Most popular hardware architectures, communication protocols and standards, and operating systems have been briefly introduced. Furthermore, a classification of the current BSN programming approaches has been described, and the most relevant research projects of each approach have been analyzed. Particular emphasis has been given to the domain-specific frameworks for BSNs and to general-purpose frameworks for WSNs applied to the BSN domain. Results of an in-depth analysis of these frameworks is reported and a taxonomy of key requirements, techniques and properties that emerged to be of fundamental importance during the design of an effective and efficient BSN programming framework have been described in detail. Finally, the analyzed frameworks have been compared on the basis of such identified key parameters.

Table 2.5. Comparison of the WSN/BSN programming frameworks.

	CodeBlue	RehabSPOT	TITAN	AFME	MiLAN
Programming abstractions					
Tunable Sensor Drivers	✓	✓	✓	✓	n.a.
Data Structures flexibility					n.a.
Flexible Communication API			✓	✓	n.a.
Parameterized Processing functions		✓	✓		
Tunable power management					✓
Software engineering methods					
Predefined in-node signal filters	✓	✓			
Predefined in-node feature extractors		✓	✓		
Embedded support for classification algorithms			✓		
Built-in Application level communication protocol (messages format, encoding, decoding)	✓	✓	✓	✓	✓
Encryption and authentication		✓			
Debugging mechanisms					
Simulators	✓	✓	✓	✓	✓
Debugger tools		✓			
Resource Optimization					
Optimized Code footprint	✓		✓		n.a.
Reduced memory usage	✓		✓		n.a.
Optimized Processing time		✓	✓		n.a.
Communication					
Communication among different nodes of the same SW platform (e.g. TelosB and MicaZ on TinyOS) and radio standard	✓	✓	✓		✓
Heterogeneous network formation of nodes programmable with the same language (e.g. C language)					
Interoperability among homogeneous BSN coordinators					
Interoperability among heterogeneous networks (e.g. Internet through sockets)		✓			

3

The *SPINE* Framework

This chapter describes the *SPINE* Framework, a novel domain-specific framework for supporting rapid development of BSN applications, along with its enhancements, and variants proposed so far.

3.1 Introduction

The analysis of the state-of-the-art on the BSN domain has highlighted that the development of BSN applications is to date a complex task also due to the lack of programming frameworks with dedicated support to the distinctive requirements of BSN systems.

To support the programming of optimized BSN applications while minimizing the development time and effort, we have designed and realized SPINE (Signal Processing In Node Environment) [44]: an open-source domain-specific programming framework [45] for BSNs.

SPINE has been designed around the requirements defined in Section 2.3.5 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 pre-defined 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 re-programming off-line the sensor nodes before switching from an application to another.

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. However, it is worth noting that as SPINE uses an application-level protocol, it may rely on an underlying network layer which supports multi-hop, so that the physical network can actually involve communication between the coordinator and nodes that are more than one hop away.

3.2 Network Architecture

The BSN architecture supported by SPINE includes multiple sensor nodes and one coordinator node (see Figure 3.1). The coordinator manages the network, collects and analyzes the data received from the sensor nodes, and acts as a gateway to connect the BSN with wide area networks for remote data access. Sensor nodes measure local physical parameters and send raw or processed data to the coordinator. Currently SPINE supports BSNs with star topology, which is a requirement for BSN applications, where sensor nodes communicate only with the coordinator. However, the framework can be easily extended to support also direct and multi-hop communications among sensor nodes.

In the current version of SPINE (version 1.3) a sensor node can only be associated with a single coordinator; a possible extension is to allow nodes to be associated and communicate with multiple coordinators. A scenario where such architecture could be used is when a patient wearing body sensors moves across locations; in this case such sensors should connect to a different coordinator at each different location.

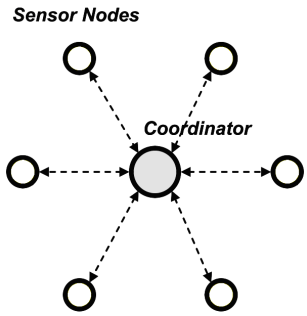


Fig. 3.1. SPINE Network architecture.

3.3 High-Level software Architecture

The SPINE framework consists of two main components:

1. *SPINE Node*, and
2. *SPINE Coordinator*.

The former is implemented in the sensor platform-specific embedded programming language and runs on the sensor nodes; the latter is implemented in Java and runs on the coordinator station. The functional architecture of SPINE is shown in Figure 3.2.

The *SPINE Node* is organized in five interacting macro functional components:

- The "*Sensor Node Manager*" is responsible for the general interactions among the Sensing Management, Signal Processing, and Communication components, and dispatches requests coming from the remote coordinator to the appropriate block.
- The "*Communication*" block handles reception and transmission of messages over-the-air, and managing radio duty cycling.
- The "*Sensing Management*" block 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. It interacts with the "Buffering" block to store the sensor readings. It also contains a sensor registry where compiled sensor drivers self-register at boot-time, to allow other components to retrieve the available sensor list.
- The "*Buffering*" block consists of a set of circular buffers dedicated to store the sensor sample data. It provides two mechanisms to access the sensor data: (i) upon requests using getter functions, and (ii) using listeners that are notified when new data from sensor/s of interest is available.
- The "*Signal Processing*" block involves a flexible, customizable and expansible set of signal processing functionalities such as math aggregators, filters, and threshold-based alarms that can be applied to any sensor data streams. This block retrieves the data to be processed from the sensor buffers, and report the results back over-the-air to the coordinator.

The *SPINE Coordinator* is organized in five components:

- The "*Communication*" block has similar functionalities of the corresponding block in the sensor node, and it has been designed to load the proper radio module adapter (e.g. for TinyOS motes or SunSPOT devices) 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.

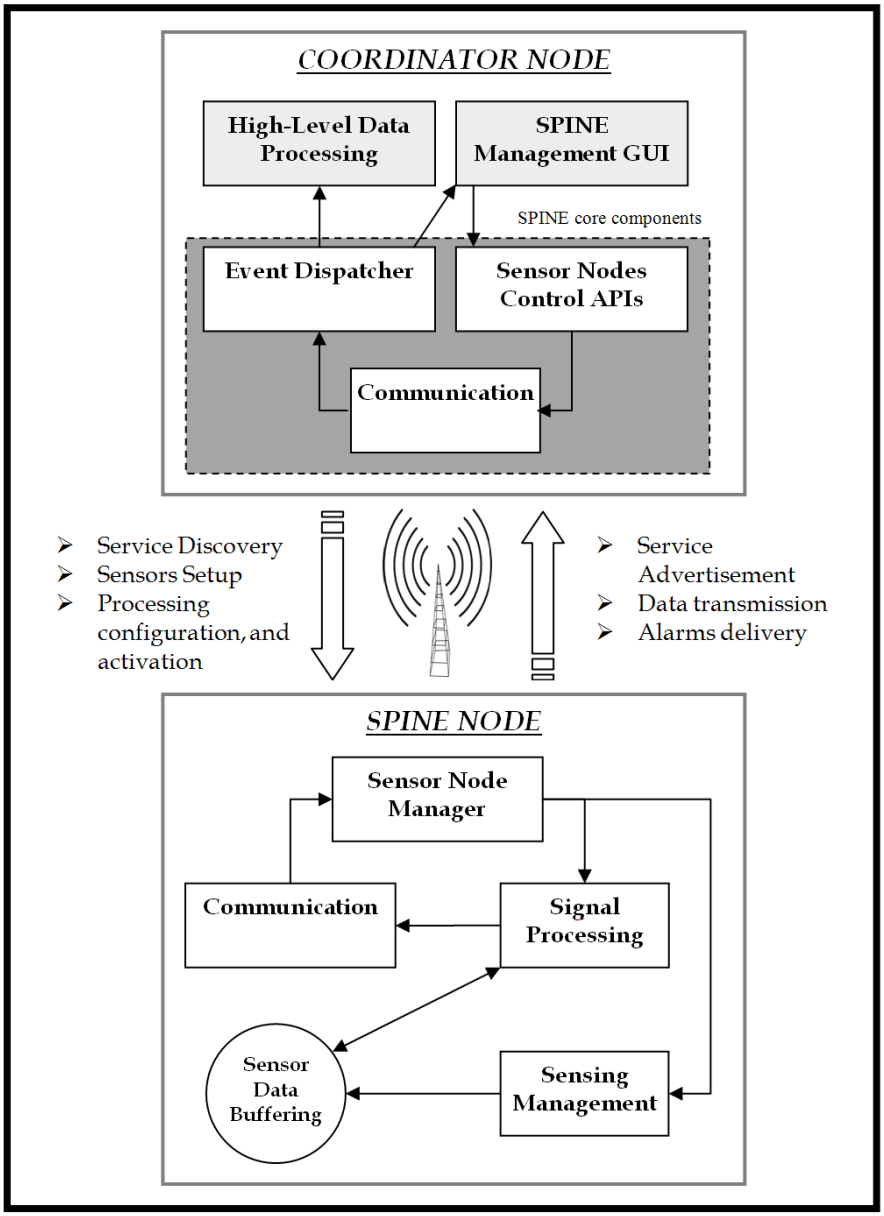


Fig. 3.2. The SPINE high-level Functional Architecture.

- The "*Sensor Nodes Control APIs*" 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 3.1. Functionalities exposed by SPINE at the coordinator station.

FUNCTIONALITY	DESCRIPTION
DISCOVERY	Allows for the discovery of nodes and their supported sensors and processing capabilities.
SENSING SETUP	For each node, allows independent specification of sampling rates for multiple sensors.
RAW SENSOR READING REQUEST	Enables periodical or one-shot transmission of raw readings from one or multiple sensors. To avoid wasted bandwidth and energy, readings are grouped before being transmitted.
ON-NODE SIGNAL PROCESSING ACTIVATION	For each node, enables one or multiple in-node (periodic or trigger-based) signal processing functionalities independently. The specific processing functions may be set up over-the-air via additional parameters.
HIGH-LEVEL DATA PROCESSING	Offers a wide set of feature selection and classification algorithms, and specific data structures to process the received data.

- The "*Event Dispatcher*" forwards various events (e.g. new nodes discovered, alarm or user data messages) to the set of registered listeners.
- The "*High-Level Data Processing*", which is described in more details in paragraph 3.9.4, 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 [46] Data Mining toolkit to allow the use of its feature selection and pattern classification algorithms from within SPINE.
- The "*SPINE Management GUI*" is a graphical add-on tool that allow configuration of remote sensor nodes using a user-friendly interface (rather than programmatically). It contains a simple textual logging function for events generated by remote nodes and received by the underlying SPINE coordinator (e.g. discovery advertisement packets, data messages).

From a programming point of view, 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, particular care has been given to simplify the extension and customization of the framework itself. In particular, it is very straightforward to add software support for new, custom-defined sensor drivers and developers can easily inter-

act with the sensor nodes through a simple Java API. The main functionalities exposed by this API are summarized in Table 3.1.

3.4 Main tunable parameters

At *compile time*, SPINE allows developers to tune a number of parameters of the sensor node. In particular, it is possible to specify which sensor drivers and processing functions must be included in the compilation, how many buffers must be allocated for the sensors, and which size such buffers must have.

At *run-time*, SPINE allows to tune several parameters of the sensor nodes and activate and deactivate (periodic, and threshold-based) processing functions.

Each available sensor of a remote node, is initially idle, but it can be set remotely to start sampling at any time. Sampling time and time scale (e.g. *ms*, *sec*, *min*) are, hence, tunable parameters.

Any computable processing function (e.g. feature extractors or alarms) is active by default, but it can be started or stopped remotely. Typical tunable processing parameters include data window size, type of feature extractors to compute on a given sensor, type and value of a threshold that would trigger alarms on given sensed data. It is also possible to enable/disable a simple *TDMA* (Time Division Multiple Access) communication protocol, and a "radio low power" mode through a duty cycle mechanism.

SPINE also introduces an optional encryption service that enables the remote motes to communicate securely with the base-station node.

3.5 SPINE application-level communication protocol

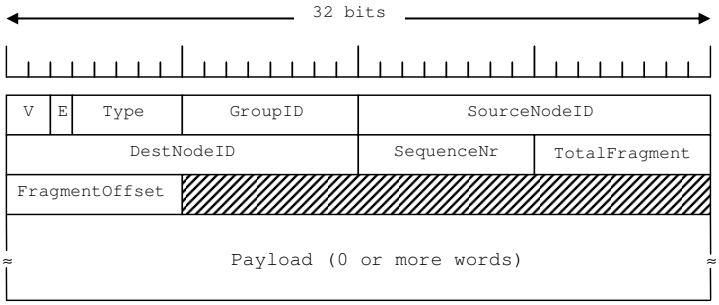
The SPINE framework includes an application-level communication protocol to manage the bidirectional communication between nodes and the coordinator. The SPINE communication protocol works at the application-level and is independent from the underlying network and data-link layers. The current architecture includes an optional TDMA scheme and a radio duty cycling mechanism.

A general communication scheme has been defined and is supported by a set of standard messages summarized in Table 3.2 whereas the structure of the general SPINE message is reported in Figure 3.3. Messages can be directed from the coordinator to a node (C $\rightarrow$ N) or from a node to the coordinator (N $\rightarrow$ C). While service messages have a fixed format, user messages can be easily customized to better fit application needs. Moreover, developers have also the possibility to extend the framework with new user-defined messages.

The communication scheme is usually initiated by a *Node Discovery* phase. Several discovery techniques can be adopted. For example, the coordinator can

Table 3.2. Standard messages of the SPINE Protocol exchanged between Coordinator (C) and Node (N).

	Direction		*Parameters*
	$C \rightarrow N$	$N \rightarrow C$	
Service Discovery	•		NONE
Service Advertisement		•	< sensors list, services list >
Set-Up Sensor	•		< sensor code, sensor parameters >
Set-Up Service	•		< service code, service parameters >
Activate Service	•		< service code >
De-activate Service	•		< service code >
Data (raw or processed)		•	< service code, data >
Start processing	•		< radio configuration >
Reset (node / network)	•		NONE
System notification		•	< notification type, notification details >



V = Version of the protocol
E = Extension flag signalling an extension of the message
Type = Type of SPINE message
GroupID = Identifier of the group
SourceNodeID = Identifier of the transmitting node
DestNodeID = Identifier of the receiving node
SequenceNr = Sequence number of the messages
Total Fragments = Number of total fragments of the SPINE message
Fragment Offset = Offset of the fragment with respect to the reference message

Fig. 3.3. Structure of the general SPINE message.

broadcast a service request (*Service Discovery* message) and wait for advertisements from active nodes within range. To keep the network information updated, service requests can be sent on a periodical basis. A *Service Advertisement*, sent from a sensor node upon reception of a Service Discovery message, includes information about the node hardware, in particular regarding the available sensors, and the node services which can be processing functions (e.g. mean, max, min, variance, total energy, entropy) and/or alarms (sensed

Table 3.3. Default services and alarms which can be activated in the SPINE node

FEATURE	DESCRIPTION
Max	Maximum value computed on a sample window.
Min	Minimum value computed on a sample window.
Range	Maximum displacement value computed on a sample window.
Mean	Average value computed on a sample window.
Amplitude	(Maximum-Mean) value computed on a sample window.
RMS	RMS value computed on a sample window.
St. Deviation	Standard Deviation value computed on a sample window.
Total Energy	Cross-axial magnitude computed on a sample window. It takes into account multiple sensor channels, if any..
Variance	Variance value computed on a sample window.
Mode	Most frequent value computed on a sample window.
Median	Median value computed on a sample window (central value of the ordered window buffer).
Vector Magnitude	Magnitude of a sample window (sum of the squares of the window elements).
Entropy	Entropy computed on a sample window.
ALARM	
Above	Triggers when a given sensor signal (or a computed feature) exceeds the specified threshold.
Below	Triggers when a given sensor signal (or a computed feature) goes below the specified threshold.
Within	Triggers when a given sensor signal (or a computed feature) is within the range of the specified thresholds (min, max).
Outside	Triggers when a given sensor signal (or a computed feature) exceeds the range specified by the thresholds (min, max).
OTHER	
Buffered Raw Data	Buffers into a single over-the-air message a given number of sensor readings. Useful to reduce radio usage and bandwidth consumption when collecting raw data.
Kcal	Computes activity counts every second. The final physical energy expenditure estimation must, however, be computed at the coordinator (see Section 4.3). It can be computed only if the node is equipped with an accelerometer sensor.
Pitch	*Pitch* angle estimation computed on a sample window. It can be computed only if the node is equipped with an accelerometer sensor.
Roll	*Roll* angle estimation computed on a sample window. It can be computed only if the node is equipped with an accelerometer sensor.

data exceeding thresholds). Table 3.3 reports the services and alarms that SPINE provides by default.

It is worth noting that many services can be set-up to process periodically, on-demand only, or even on event basis. Hence, the data flow generated by the service is dependent on the service itself and on the specific dynamic configuration requested by the application on that service. For instance, a raw data service can be set-up to transmit sensor data periodically, one-shot, or

only when a given threshold has been exceeded. According to the information collected, the user application on the coordinator must first set-up the sensors of interest by specifying: sensor identification code, sampling time, and time scale. Once sensors have been set-up, the user application will typically set-up the desired services (if the service requires a preliminary set-up phase) and subsequently will activate them. Services can be dynamically activated and, if necessary, de-activation is also supported.

Set-up, activation and de-activation of a service involve the specification of certain parameters which usually vary from service to service. Thus, corresponding messages have been structured with a dynamic and partly service-specific format. As aforementioned, the SPINE framework provides native services whereas new services can be easily integrated. In this case, the developer must enhance the framework with a specific format for the set-up, activation and de-activation request messages.

Once each node is fully set-up, the user application is finally ready to start working by broadcasting a start message. From this time on, the nodes will start sensing and processing according to the activated services. The data produced by the running services of each node are transmitted in data messages to the coordinator which in turn forwards the message content to the user application. The data message has a well defined format, but its payload semantic is clearly dependent from the service that generated that data and, again, if it is a user-defined service, coding and decoding of the payload content is up to the developer. To notify the coordinator of system events such as low battery warnings, processing overhead errors or bad requests, nodes can issue defined system event notification messages. Single nodes or the whole network can be reset by the coordinator if requested by the user application. Information such as sensor and service advertisements, error and warning types and details, are all exchanged in form of numerical codes which must be shared between nodes and coordinator.

To provide a concrete example of a typical messages exchange between the coordinator and a node, Figure 3.4 shows the sequence of transmission within a scenario where a user application is looking for a node equipped with a given sensor to request some in-node processing of the data sensed by that sensor. The user application, through the coordinator side of SPINE, broadcasts a Service Discovery to check if a node with the required sensor is found in the nearby. An active node replies to the service discovery with a Service Advertisement. The information contained in the advertisement message is sufficient to the user application to understand if that node has the sensor(s) and the signal processing functionalities required. In that case, the application can proceed by first setting-up the sensor with the desired sampling rate. Then, the application will set-up the specific function(s) and subsequently will activate them. The node configuration phase is now complete and the application issues a *Start* message which includes configuration for radio behaviors of the sensor nodes (enabling/disabling of TDMA, number of nodes for TDMA initialization, and radio activity of type always on or duty cycle). Upon re-

ception of the start message, the node reacts by starting the sampling timer on the sensor and the processing mechanism, and transmitting the results as they are available.

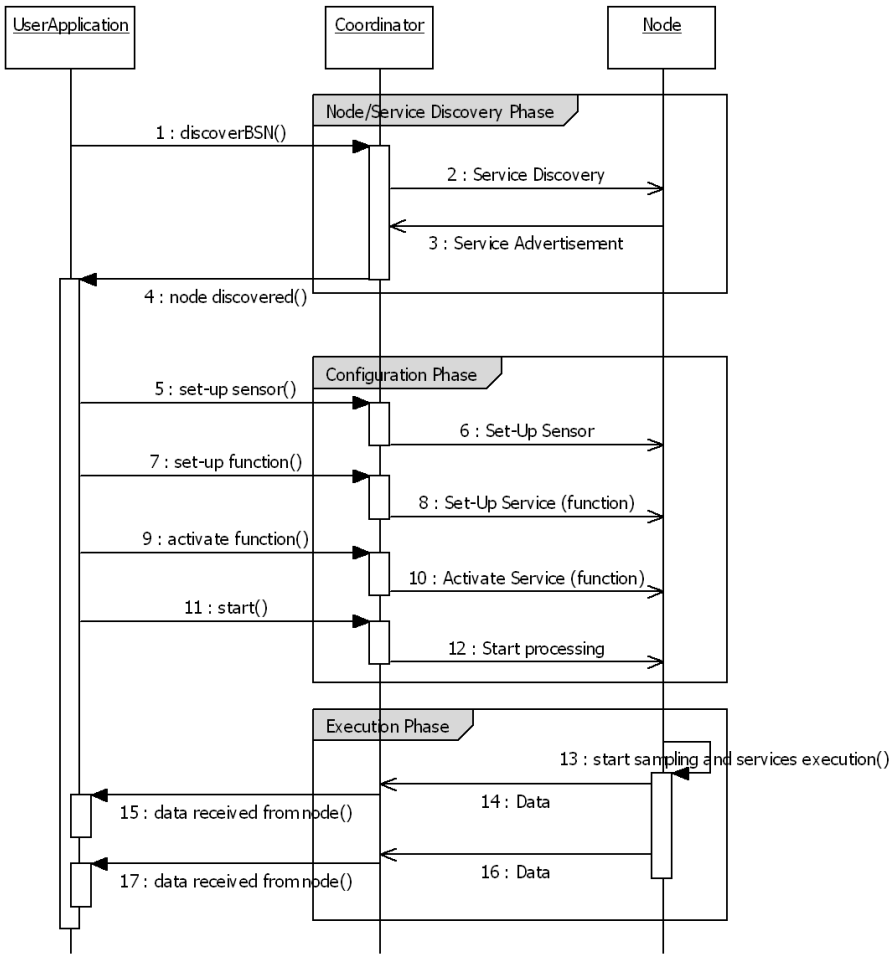


Fig. 3.4. Example of communication between User application, Coordinator and Sensor Node.

An important mechanism of SPINE is the *radio duty cycling*, a simple runtime mechanism to help reducing the power consumption due to the radio usage. When enabled by the coordinator, the sensor nodes turn on the radio only when they need to transmit data over the air. In order to receive messages, before turning off the radio after the successful transmission of a message, the radio is kept listening to incoming messages for a given period of time (duty

cycling timer), which can be set at compile time (usually in the order of a few milliseconds). If a message is received, then the timer is reset. The coordinator, which keeps a queue of the messages to be sent to each node, sequentially sends each message for a given node immediately after receiving a packet. If an ack is received for a sent message, the coordinator removes that message from the queue of messages ready to be sent.

3.6 Multi-platform Support

The variety of hardware platforms, sensors, programming languages and operating systems supported by SPINE enables a great degree of heterogeneity. This allows for a very flexible and usable framework in different BSN application scenarios (e.g. e-Health, e-Fitness/Wellness, 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 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 using the Texas Instruments Z-Stack, and for the Java-based SunSPOT nodes are also available.

Many physical sensors (accelerometers, gyroscopes, Electrocardiogram, Electro impedance 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, while the driver for the accelerometer is available by default, developers may integrate drivers for further sensors (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 to personal computers/work-stations. 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 [47] implementation of SPINE has been developed. It has been tested on several smartphones based on Android 2.2 or above, communicating 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. Bluetooth communication is also supported by the main SPINE version, using the open-source BlueCove API [48] on PCs (Windows, and Linux only), and the Android Bluetooth API [49] on devices running Android 2.0 or above.

Finally, SPINE has been also ported on an *emulator* tool that virtualizes SPINE-enabled sensor nodes. The tool allows emulation of a set of nodes forming a BSN and requires a dataset for each node. The data set 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, data sets from real sensors can be used to objectively validate and compare different processing algorithms or hardware set-ups. 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.

3.7 The Node-Side module

The node side of the SPINE functional architecture presented in section 3.3 is described in the following, by detailing the developed TinyOS-based architecture, and discussing the performance evaluation results.

3.7.1 Software-architecture in TinyOS

The flexible and effective TinyOS software architecture of SPINE is presented in the following. The presentation of the static architecture is aided by UML class diagrams, while UML sequence diagrams are used to show key dynamic interactions among software components. The whole architecture has been graphically shown in three separate class diagrams (see Figures 3.5, 3.7, and 3.9). For the sake of clarity, logical components are reported as single blocks, when they are self explaining. The stereotype $\ll$*component*$\gg$ has been used to represent TinyOS entities which typically consist of three components: an interface for publishing the component commands and events, a configuration for wiring external components and a module which implements the component commands and that can generate the defined events of the component interface. The stereotype $\ll$*is wired to*$\gg$ indicates the TinyOS wiring operation "->".

3.7.1.1 Sensing

The class diagram in Figure 3.5 shows the architecture of the *SPINE Sensor Controller* (or sensing functional block). To enhance extendibility, access to the sensors drivers has been decoupled by the introduction of the *SensorBoard Controller* and the *Sensor* interface. Thus, actual sensors are addressed by unique codes representing their abstraction. In particular, by using *parameterized* Sensor interfaces, the SensorBoard Controller module is itself independent of the actual sensors; wiring the actual sensor drivers (e.g. `HilAccelerometerSensorC`) to the parameterized Sensor interface is left to the SensorBoard Controller configuration component, which is much easier to edit or extend. Each sensor driver module (e.g. `HilAccelerometerSensorP`) must register its unique sensor code to the *Sensor Registry* in orded to be inserted into the Service Advertisement message. The SensorBoard Controller module uses timers for the sampling operations of the available sensors.

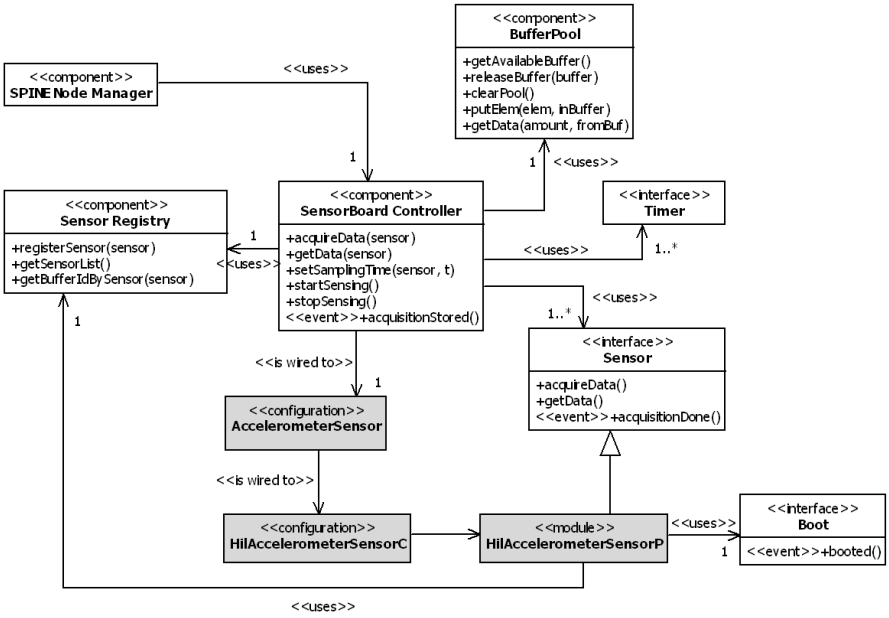


Fig. 3.5. Class diagram of the sensing logical block.

In TinyOS sensor data are typically gathered in "*split-phase*", which means that an operation request and its correlated response are separated with a callback mechanism. Hence, as shown in the sequence diagram of Figure 3.6, when a given sensor timer fires, the SensorBoard Controller requests the corresponding sensor to start data acquisition. When the data is ready, the sensor driver notifies the SensorBoard Controller which, in turn, can get the new reading.

Finally, to decouple data sources (the sensors) by data consumers (the processing services), sensor readings are stored by the SensorBoard Controller in an ad-hoc *Buffer Pool*. The Buffer Pool is internally implemented as a set of circular buffers. Mappings between buffers and sensors are stored in the Sensor Registry.

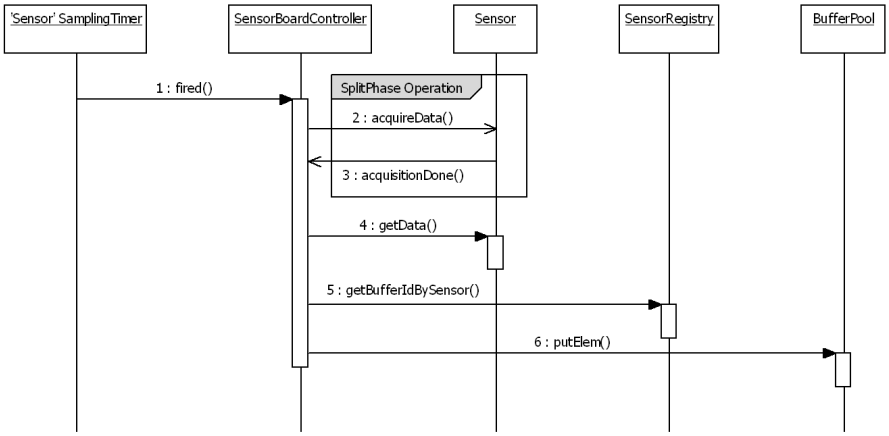


Fig. 3.6. Sequence diagram of the sensing process.

3.7.1.2 Processing

The *SPINE Processing Manager* (or processing block) (see Figure 3.7) relies on a similar schema adopted for the sensing block to support fast extensions of the processing functions provided by the framework. In fact, the *Function Manager* handles the actual implemented functions through the parameterized interface Function. *Features* are particular types of processing functions which are applied on windows of sensed data. In particular, they are characterized by the following parameters: *Window*, which is the number of buffered data samples on which the function is applied, and *Sliding%*, which is the percentage of shift on the buffered data samples with respect to the Window. For instance, if Window=40 and Sliding%=50, the function is computed on 40 acquired samples composed of the last 20 previously acquired samples and the first 20 newly acquired samples.

A specific type of function is the `FeatureEngine` which handles particular math functions named *features* (e.g., max, min, standard deviation, etc). In particular, the FeatureEngine acts as a dispatcher for accessing the various feature extractors components and as an aggregator if multiple features are requested to be computed on the same data. Functions are invoked through the Function Manager by unique codes.

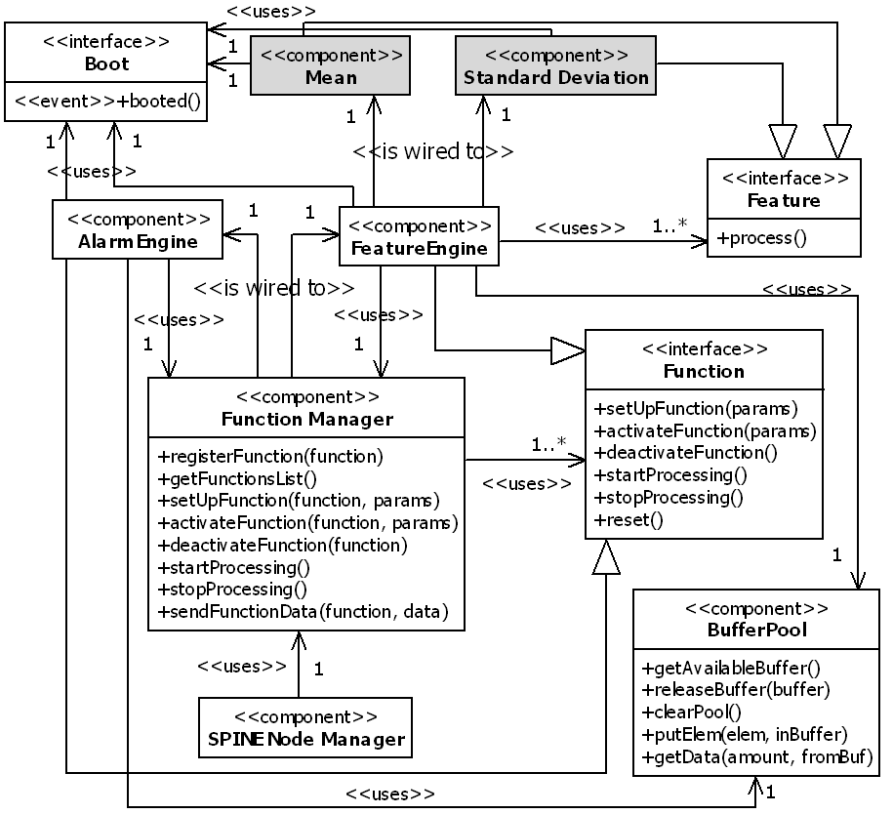


Fig. 3.7. Class diagram of the processing logical block.

Besides centralizing the access to the various functions, the Function Manager mainly provides a data transmission command which masks the presence of lower level transmission services to the functions. This choice is motivated by standardization issues of data messages: to guarantee the creation of standard messages, function data are encapsulated by the Function Manager as the payload of a new SPINE data message. It should be noted that the Function interface is strictly part of the framework core and is used to generalize the concept of processing function, such as feature extractors, alarms, sensor data filters and pre-processors, and even simple on-line classifiers. The Feature interface, instead, has the solely scope of decoupling the actual feature extractors from the FeatureEngine.

Figure 3.8 shows the complete sequence of steps from buffer data fetching to features extraction computation. As aforementioned, processing functions are completely decoupled from the sensor data generation; to get the proper data frames, a function, as soon as a new computation is required, accesses the Sensor Registry to get the buffer ID associated to the sensor of interest;

then, it obtains the desired data amount on that buffer through the BufferPool component and computes the processing.

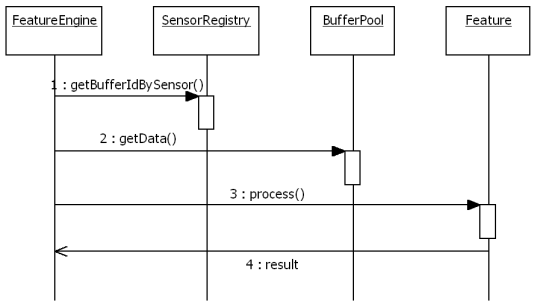


Fig. 3.8. Sequence diagram of a feature processing.

3.7.1.3 Communication

The *SPINE Node Communication Manager* (or communication block) is presented in Figure 3.9 to explain how messages transmission and reception are handled at the SPINE node side. At the lowest level, the `RadioController` masks the TinyOS components for controlling the physical radio states (e.g. turn on/switch off) and handling packets transmission and reception.

The RadioController exposes the send command to transmit a stream of bytes, which are not further manipulated. Users of the RadioController are not aware of the current state of the radio module, which is transparently managed by the RadioController.

Additionally, the RadioController signals the event of the reception of a new packet. Such events are captured by the Packet Manager which verifies whether the received packet is a SPINE message and, if that is the case, generates a new `spineMsgReceived` event. SPINE messages are decoded and encoded by different components one for each message type defined. The *Packet Manager*, after having recognized the type, which is contained into the SPINE header of each message, dispatches the encoding (if it is an outgoing message) or the decoding (if it is an incoming message) to the proper codec component. The dispatching is implemented through parameterized interfaces: `InPacket` for incoming messages and `OutPacket` for outgoing messages.

The sequence diagram in Figure 3.10 shows how the reception of a new message is captured and processed by SPINE. In the example, a user application sends over-the-air a *Set-Up Sensor* request. The packet is received by the RadioController which forwards it "as is" to the Packet Manager. The Packet Manager checks, by processing the packet header, that it is a valid SPINE message and requests the message decoding to the proper decoder;

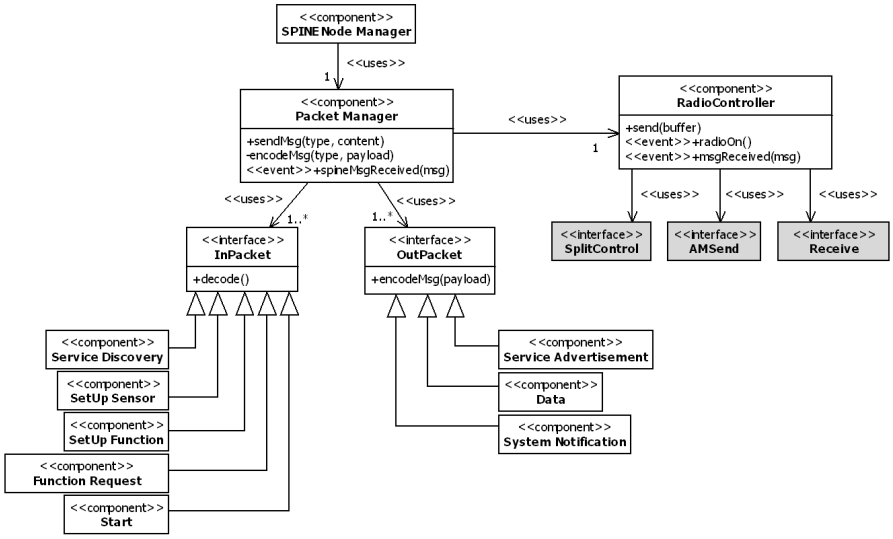


Fig. 3.9. Class diagram of the communication logical block.

then it generates a *spineMsgReceived* event, notifying the *SPINE Node Manager* with the message type. The SPINE Node Manager handles this event by invoking an internal procedure to process the message parameters coming from the SetUp Sensor decoder component. Finally, it invokes the SensorBoard Controller to set-up the sampling timer needed to drive the sensing operation of the given sensor.

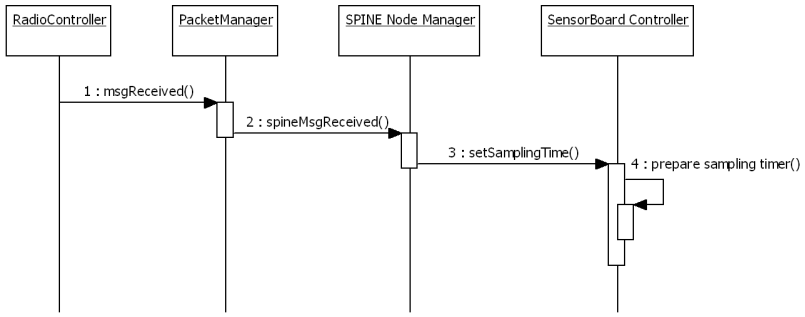


Fig. 3.10. Sequence diagram of a message reception and handling.

An example of message transmission is shown in Figure 3.11. After having computed the activated features, the FeatureEngine sends back the results by invoking the `sendFunctionData` command of the Function Manager; the Function Manager generates a new data transmission request to the Packet

Manager which, in turn, encodes this request into a Data message and invokes the Radio Controller to transmit it over-the-air.

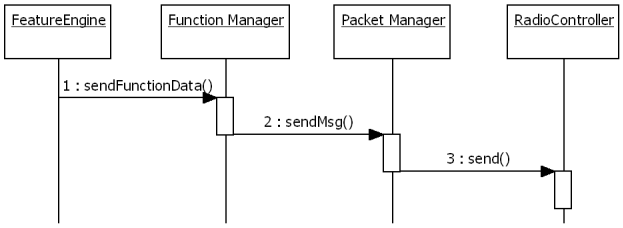


Fig. 3.11. Sequence diagram of a data message transmission.

3.8 Performance Evaluation

An extensive performance evaluation of the SPINE framework was carried out, which included all the supported sensor node platforms. It included:

a. measurement of the execution time of signal processing functionalities;
b. memory usage of the framework;
c. communication bandwidth usage;
d. energy consumption under a given application profile.

In addition, some of the significant functionalities of SPINE have been re-implemented in specific ad-hoc applications in TinyOS to evaluate how much overhead in terms of computation, memory, bandwidth, and energy requirements is added by SPINE.

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

3.8.1 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 for identifying 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 for calculating 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, a simple physical activity recognition system has been implemented on Shimmer motes [50] (that natively embed a 3-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 Figure 3.12). Therefore, for this class of applications, it is 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.

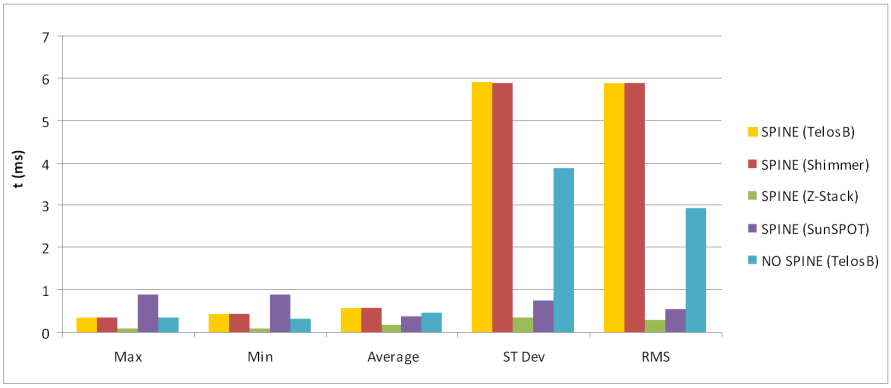


Fig. 3.12. Execution time of selected in-node functions computed on different sensor platforms using Sampling Time = 20Hz, Window = 40 samples, Shift = 20 samples.

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 sen-

sor 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.

Figure 3.12 summarize 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). 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.

3.8.2 Memory Requirements

The memory footprint of SPINE has been analyzed on different platforms. In particular, the drivers for a motion sensor board (which mounts an accelerometer sensor) have been wired to the core framework, and have been plugged the feature extractors and the threshold-based alarms signal processing units.

Results summarized in Table 3.4 show that the framework is very 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, in particular, the 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 reducing the ROM footprint by reconfiguring the framework very easily to remove from the compilation all the standard sensing, and processing functionalities that are not needed for specific applications.

3.8.3 Energy Consumption

To evaluate energy consumption, an application profile that was supported both by SPINE and implemented with hard-coded logic in TinyOS has been defined. In particular, a three-axis accelerometer sensor (attached to the node

Table 3.4. Memory requirements of a SPINE configuration (motion sensor board with feature extractors and threshold-based alarms) on different sensor node platforms.

Application Profile	RAM (Kb used/available)	ROM (Kb used/available)
SPINE on *TelosB*	3.7 / 10	33.5 / 48
SPINE on *Shimmer*	4.4 / 10	40 / 48
SPINE on *Shimmer using BT*	4.3 / 10	34.4 / 48
SPINE on *Z-Stack*	3.9 / 8	95.9 / 128
SPINE on *SunSPOT*	79 / 512	75 / 4096
Ad-hoc appl. on *TelosB*	1.3 / 10	16.1 / 48

platform) is sampled at 20 Hz and a sequence of 20 sensor readings is transmitted to the coordinator by using a single message, so that the radio is actively used to transmit one message every second.

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, it 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 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, one second, as the sensor is sampled at 20Hz, 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 (the TelosB, which normally use two AA alkaline batteries, has been opportunely modified) has the greatest capacity among the selected platforms (see Table 3.5). Although TelosB and 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.

3.8.4 Communication Bandwidth

To analyze the bandwidth usage, we refer to the same application profile used for the energy consumption evaluation. Hence, it is supposed to send over-the-air, once a second, 20 readings of a three-axis accelerometer, where each single axis sample takes 2 bytes. On platforms using the IEEE 802.15.4 CC2420 radio, SPINE automatically fragments this message as the total number of

Table 3.5. Energy consumption of the application profile (motion sensor board with feature extractors and threshold-based alarms) on different platforms.

Application Profile	Average Power Consumption	Battery	Lifetime
SPINE on *TelosB*	6.6 mW/s	650mAh	101 h
SPINE on *Shimmer*	13.9 mW/s	280mAh	21 h
SPINE on *Shimmer using BT*	87.8 mW/s	280mAh	3 h
SPINE on *Z-Stack*	11.2 mW/s	650mAh	60 h
SPINE on *SunSPOT*	84.2 mW/s	720mAh	9 h
Ad-hoc appl. on *TelosB*	73.7 mW/s	650mAh	9 h

bytes to send is greater than the TX buffer (which is 128 bytes). Results are summarized in Table 3.6. 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 bitrate. With the Shimmer, it is possible to use the Bluetooth radio, and decrease the overhead incurred by fragmentation. Finally, as expected, an ad-hoc 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 a packet-specific headers.

Table 3.6. Bandwidth of a SPINE configuration (motion sensor board with feature extractors and threshold-based alarms) on different platforms.

Application Profile	Bitrate
SPINE on *TelosB*	178 byte/s
SPINE on *Shimmer*	178 byte/s
SPINE on *Shimmer using BT*	150 byte/s
SPINE on *Z-Stack*	160 byte/s
SPINE on *SunSPOT*	168 byte/s
Ad-hoc appl. on *TelosB*	152 byte/s

Table 3.7 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, the measurements confirm that using Bluetooth on the Shimmer allows for shorter delays (3 times shorter than transmitting the same packet using the 802.15.4 radio).

Table 3.7. Average transmission delay for sending 28 bytes on different platforms.

Radio Type	SPINE TelosB	SPINE Shimmer	SPINE Z-Stack	SPINE SunSPOT	NO SPINE TelosB
IEEE 802.15.4	10,07 ms	10,04 ms	0,61 ms	67,2 ms	9,98 ms
BLUETOOTH	N/A	3,05 ms	N/A	N/A	N/A

3.8.5 An Analysis of the Development Effectiveness and Performance

This section is devoted to provide an analysis of the development effectiveness and performance of the proposed framework. In particular, the analysis is carried out with respect to an application developed without any high-level framework and centered on an approach based on the processing of sensed data at the base station side only.

The development effectiveness of SPINE have been evaluated by means of a simple case study which is represented by a human activity recognition system. The application has been implemented atop SPINE (see Section 4.1), and another application with the same functionalities has been implemented following an application-specific approach.

The basic components as well as the ones specifically related to the developed human activity monitoring system were analyzed. The critical development points have been identified by defining the percentage of efforts for developing each component (both at node and coordinator level (see Figure 3.13).

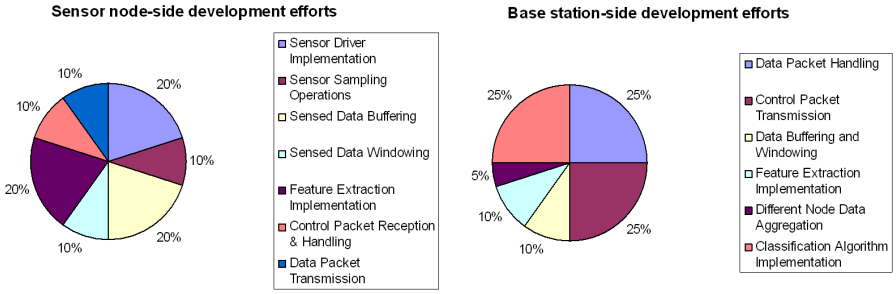


Fig. 3.13. Development efforts for typical BSN applications at sensor node-side and base station-side.

As can be seen, without SPINE, all the components had to be implemented from scratch, whereas using SPINE, only what is strictly specific to the application had to be programmed, since most of the required services (e.g. for sensing, and communicating) are already available in the libraries and easily

configurable. The saved development efforts are 100% at the node side, and 80% at the base-station side. This consideration shows that a notable improvement can be obtained by adopting SPINE for the development of BSN applications.

3.9 The Coordinator-Side module

3.9.1 Software-architecture in Java

As aforementioned, the SPINE architecture consists of two entities, one on the sensor nodes and the other on a coordinator station. Such coordinator can be a desktop computer, a laptop but even a PDA or a smart-phone. The coordinator provides the end-user application with an access point to the wireless BSN. Thus, its main tasks are controlling the remote nodes and capturing the various messages and events generated, according to the user-application needs.

The design of the SPINE coordinator is driven by principles of lightweight, ease of use and portability. The idea is to provide a small set of high-level operations by which the BSN can be controlled and an effective set of events to let the application be notified of information coming from the BSN. To enhance portability, the Java language was adopted. Java is supported PCs, and by most of the current PDAs and smart-phones. Hence, a careful use of Java libraries and paradigms allows fast porting, e.g. from a PC implementation of the SPINE coordinator to a mobile phone. In particular, the implementation is based on the 1.4 version of Java and makes use of data structures and libraries which are available both for desktop and mobile edition of the Java Virtual Machine.

Before discussing a simplified architecture of the coordinator-side of SPINE, it is important to observe that none of the existing computers and mobile phones have a native wireless interface for communicating with most of the sensor node platforms as they are typically based on the IEEE 802.15.4 communication protocol, rather than Wi-Fi or Bluetooth. Hence, a portion of the implementation is dependent to the particular base-station module attached to the coordinator node for allowing the physical communication with the BSN nodes.

Figure 3.14 shows a simplified Package Diagram of the SPINE Coordinator. The *SPINE Core* package includes the *SPINE Manager* class contained in the *Commands API* and used by end-user applications for issuing commands to the BSN. Moreover the SPINE manager is responsible of capturing low-level messages and nodes events through the *Event Listener* to notify registered applications with higher-level events and messages content. Additionally, the SPINE Core package contains tables of constants codes, such as sensors and functions codes, which must be aligned with the ones present on the nodes.

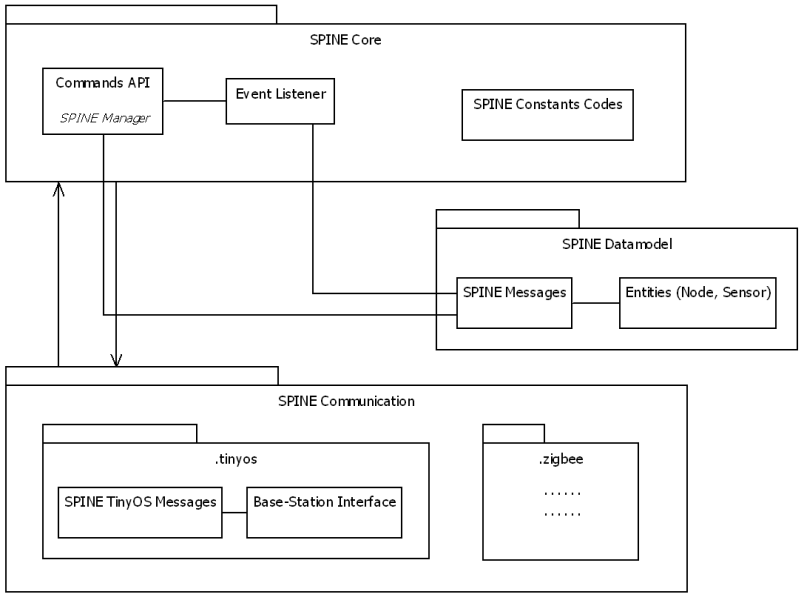


Fig. 3.14. Simplified Package Diagram of the SPINE Coordinator.

The *SPINE Datamodel* package contains classes which represent the high-level, platform independent SPINE Messages as well as abstractions of BSN nodes and sensor itself (e.g. a `Node` object will be characterized by attributes as its type and ID and built-in sensors lists and available processing functions lists).

End-user applications are only aware of the SPINE Core and Datamodel packages and hence are completely decoupled by specific implementation of the SPINE Messages and communication procedures of the currently available sensor node platform.

The SPINE Communication package is composed of a *send/receive* interface and some components implementing that interface according to the specific base-station platform and that represent the high-level SPINE Messages in platform-specific messages. Thus, most of the work for porting an implementation of the SPINE Coordinator on a different platform is to be carried out in this package.

3.9.2 BSN runtime configuration APIs

SPINE provides, on the Coordinator, a set of simple Java APIs to effectively support the development of BSN applications. In summary, the main exposed functionalities are the following:

- discovery request for the surrounding sensor nodes;

- configuration of a function of a given node (a function-dependent parameter set is passed along with the request);
- configuration of a specific sensor (e.g. to set the sampling rate) of a given node;
- request for an "immediate one-shot" sampling on a given sensor node;
- activation of a function (or even function sub-routine) on a given sensor node;
- de-activation of a function (or even function sub-routine) on a given sensor node;
- request for starting the configured, and activated sensing and processing functionalities (this is a broadcast request to "start" the entire network at the same time);
- software reset (i.e. re-boot) of the entire connected sensor network.

3.9.3 BSN event handlers

The SPINE Coordinator notifies the applications of the various events generated by the BSN nodes through a classic mechanisms based on the *Listener* design pattern. Applications using SPINE, therefore, have to register their interest to be notified of the discovery of sensor nodes, the reception of data messages, and so on. In particular, the most important events that may be forwarded to registered applications are the following:

- discovery of a new sensor node (it is generated when an advertisement message from a BSN node is received, and contains information on its sensing and processing capabilities);
- end of the sensor network discovery procedure (it provides a list of all the sensor nodes discovered so far);
- reception of new data (either raw sensor readings or processed information) from a specific node;
- reception of a service message (e.g. notification of warning or error situations) from a specific node.

3.9.4 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* with flexible and reusable Java code.

It is designed to simplify the integration of SPINE in Signal Processing or Data Mining 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 Figure 3.15. Each of them provides a number of default implementations the developer may choose. In addition, a very modular architecture allows for easy integration of further custom-defined components.

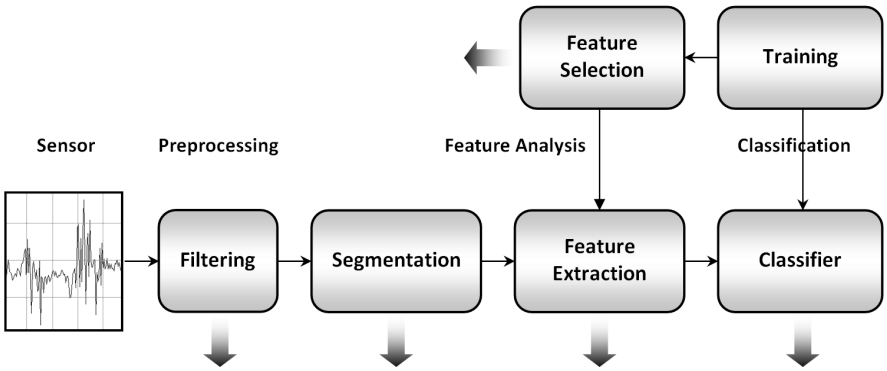


Fig. 3.15. Data processing chain supported by the SPINE High-level Data Processing module.

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 also 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, the proposed module is also integrated with the *WEKA* Data Mining toolkit [46]. This brings great advantages to SPINE as developing such 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.

3.10 SPINE enhancements and variants

The research efforts conducted for the development of the SPINE framework, also founded the basis for interesting enhancements and variants of SPINE that are relevant from both a research, and a technical point of view. These enhancements and variants cover additional relevant aspects of the development of BSN applications.

This section describes the essential concepts of these research results.

3.10.1 C-SPINE

BSNs are being mainly applied to monitor single assisted livings so the systems proposed so far are typically based on a star topology composed of a set of wireless sensors coordinated by a base station (usually a PDA/smartphone). Such systems efficiently support programming of sensor nodes and communication between sensor nodes and the base station to develop remote monitoring applications.

However, different kinds of BSN architectures can be envisaged in different application domains in which monitoring of single assisted livings is not enough to fulfill the application requirements. Such application domains include e-Health, e-Emergency, e-Entertainment, e-Sport, e-Factory, and e-Social.

Collaborative Body Sensor Networks (CBSNs), a new type of BSN architecture, can be therefore defined to allow BSNs to interact with each other to support the development of collaborative applications, where not only single assisted living monitoring is needed, but also data exchange and cooperative processing among BSNs can be triggered and managed.

3.10.1.1 Novel Interaction Models

To support the many requirements of the diverse BSN scenario, different logical network architectures of BSN systems can be designed. We refer to logical architectures as the focus of this thesis is in fact related to the final actors of the system that generate and consume data, although the actual low-level data flow may require intermediate transmissions (e.g. via multi-hop routing).

As a consequence, common interaction models in the BSN domain have been identified; some of them are actually novel, in the sense that they have not been formally described before. Figure 3.16 depicts the most important ones.

Figure 3.16(a) represents the "*single body - single base station* (BS)" interaction model, where a single BSN worn by a user interacts with one BS, typically represented by his personal smart-phone or computer. This logical topology is very common for a very wide range of applications, such as e-Health and e-Fitness, where user health conditions are locally processed, stored, visualized, and possibly forwarded remotely for further analysis.

Figure 3.16(b) refers to the "*single body - multiple BSs*" model that is realized for applications where a single BSN communicates with multiple BSs. This could be the case of a complex in-building application where the BSN of the user interacts with different BSs with regards to the specific context, e.g. with a smart-phone for health monitoring and (possibly concurrently) with a game console for high interactive game controls.

Figure 3.16(c) represents the "*multiple body - single BS*" model. This is typical in mobile medical applications e.g. for family doctors that use their personal computer for monitoring patients wearing a BSN as they come to the doctor office. Another application that requires this particular interaction model may be an e-Entertainment scenario where a group of friends, wearing specific body wireless sensors, interact with a game console or a TV for some kind of enhanced social interaction.

Finally, Figure 3.16(d) is referred to the "*multiple body - multiple BSs*" interaction model, which is related to complex BSN systems involving communications among many BSs and BSNs. Application scenarios include emergency response to disasters such as earthquakes, terrorist attacks, wild fires, or big car accidents, where the rescue team(s) may place medical wireless sensors to the victims to monitor their conditions. Another example is given by e-Factory applications, where collaborative BSNs aids a group of workers collaborating to a shared goal.

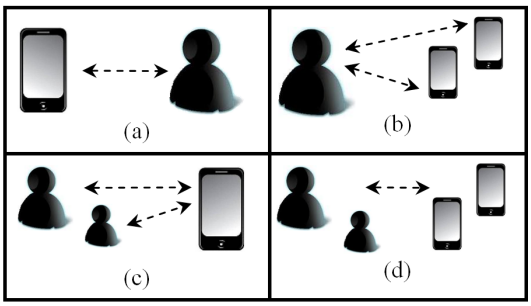


Fig. 3.16. Logical network architectures of BSN systems: (a) *single body - single BS*, (b) *single body - multiple BS*, (c) *multiple body - single BS*, (d) *multiple body - multiple BS*.

3.10.1.2 Collaborative BSNs

We have identified a reference architecture of CBSN which consists of two parts: the network architecture and the software architecture. While the former includes the communication layer (basic and application-specific interaction protocols), the latter defines types and tasks of the functional components that carry out system management and execute application-specific activities.

The network architecture of CBSNs is portrayed in Figure 3.17. Each BSN is composed of a base station (BS) and a set of wireless sensors (WSs). The BS communicates with its WSs through an intra-BSN over-the-air (OTA) protocol and with the BSs of other BSNs through a set of inter-BSN OTA protocols.

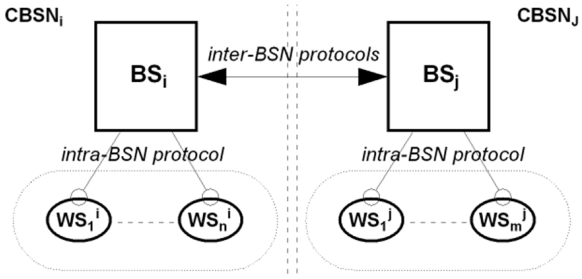


Fig. 3.17. Reference network architecture of CBSNs.

The intra-BSN OTA protocol supports discovery, configuration and control of WSs as well as data transmission from WSs to BS. The discovery function discovers the WS belonging to the BSN and retrieves the services (sensing, actuating, computing, communication, storing) that each WS offers. The configuration function allows setting up the services of the discovered WSs. The control function enables activation, monitoring and control of the configured WS services. Finally, the data transmission function allows WSs to send raw data and/or pre-processed data to the BS and/or to other WSs of the same BSN.

The inter-BSN OTA protocols are divided into two categories: basic and application-specific. The basic protocols include:

- the proximity detection protocol;
- the service description protocol;
- the service activation protocol.

The *proximity detection protocol* allows for detection of neighbor BSNs and management of the detected BSNs. The *service description protocol* provides on-demand information about the available services and/or applications that BSNs can offer to each other. The *service activation protocol* supports collaborative service selection and activation. The application-specific protocols support interaction between specific applications running on CBSNs.

Figure 3.18 shows a high-level interaction diagram involving proximity detection, service description and generic service activation and execution between two CBSNs. In particular, as soon as a CBSN detects a neighbor CBSN (proximity detection phase), a service description request is sent (service discovery phase) and a manually or automatically selected service is activated

(service selection and activation phase), and then executed (service execution phase).

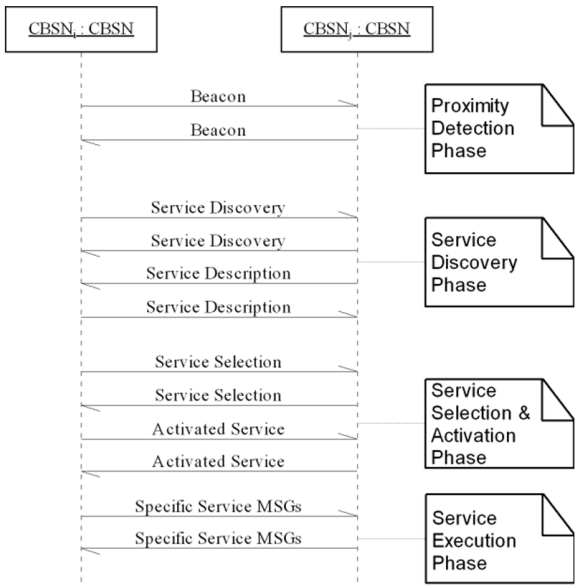


Fig. 3.18. High-level interaction among CBSNs.

The functional architecture of CBSNs (see Figure 3.19) is inherently service-oriented even though its actual implementation depends on the chosen paradigm and technology. In particular, Figure 3.19 shows the collaborative layer, installed at the BS-side, which consists of the following main components:

- *CBSN Manager*, which manages the proximity detection, the service discovery and the service activation phases. Service activation can be either automatic (i.e. a service is activated as soon as it is discovered) or on-demand (i.e. a discovered service is activated only if the owner of the CBSN agrees to). Automatic activation can be based on service classification and mutual acquaintance relations among CBSN owners.
- *Service-specific Managers* (SS Managers), which manage the specific services through inter-BSN service-specific protocols and the intra-BSN protocol for WS interaction.

3.10.1.3 Collaborative SPINE

The CBSN reference architectures have been integrated within the SPINE framework to define a SPINE-based CBSN middleware, named *Collaborative SPINE* (C-SPINE) [51].

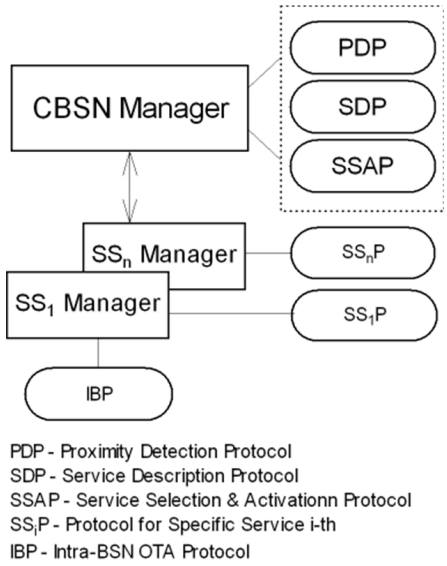


Fig. 3.19. CBSN software architecture layers.

The architecture of C-SPINE, which is portrayed in Figure 3.20, integrates the SPINE WS, SPINE BS, and CBSN architectural components. The SPINE WS and BS architectural components have been detailed in Section 3.7 and 3.9.

The *SPINE BS-BS Communication* component implements and manages the C-SPINE Inter-BSN OTA (*CIBO*) Protocol to provide an effective communication layer to the basic and applicationspecific services. This communication layer abstracts away the specific lower-level communication protocol that can be actually used (see Figure 3.21) through adapters. The communication layer is designed around the concepts of Message, *Communication Provider* and *Message Handler*. The Communication Provider exports methods that allow configuring the BS to receive packets of a certain message type and discard other types. During the configuration, one or more Message Handlers are passed to the Communication Provider so that it can notify the reception of a packet corresponding to the defined message ones. To send a packet, the Communication Provider exports the send method, which also permits to specify the packet recipients and is independent of the message type. This mechanism allows high-level client components to handle incoming packets with separate routines. According to this layer, the basic steps to define an interaction protocol are:

- Creation of a new message type that is associated to the protocol;
- Creation of a new set of packets belonging to this message type (i.e. the protocol messages);

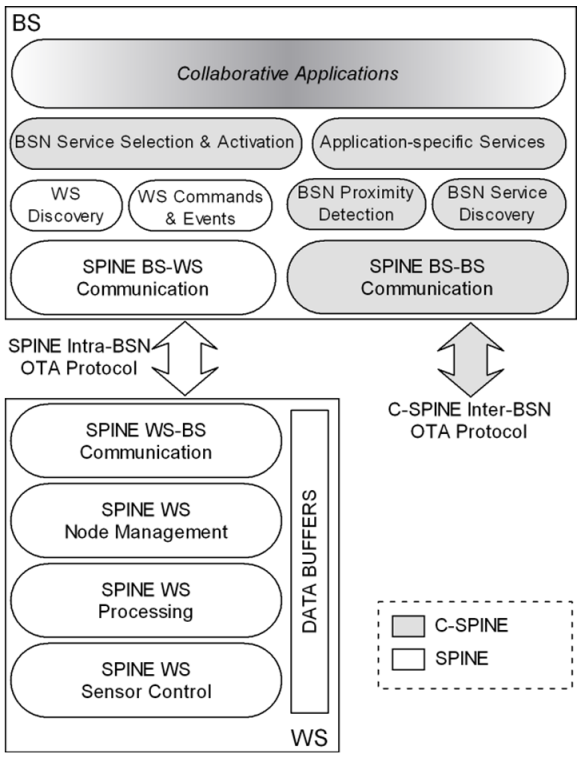


Fig. 3.20. C-SPINE Architecture.

- Creation of a new Handler for handling this message type (i.e. all packets of this message type);
- Registration of the defined Handler with the Communication Provider to handle the packets of the defined protocol.

The CIBO protocol has been defined according to this process and specifically supports the basic services of C-SPINE: BSN proximity detection, BSN service discovery, and BSN service selection and activation.

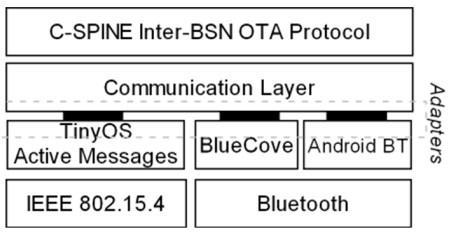


Fig. 3.21. C-SPINE Communication Layer.

3.10.2 An Agent-oriented design of SPINE: A-SPINE

In this section, an agent-oriented redesign of SPINE (*A-SPINE*), partially based on *MAPS* (Mobile Agent Platform for Sun SPOTs), is proposed [42]. The MAPS frameworks [52, 53] is briefly described in Appendix A).

We believe that the exploitation of the agent-oriented programming paradigm to develop BSN applications is certainly promising, as demonstrated by the application of agent technology in several other key application domains [54].

The coordinator of A-SPINE is composed on a JADE-based [55] enhancement of the SPINE coordinator. It allows configuring the sensing process, and receiving sensed data features. The sensor node software is based on MAPS.

The main research contribution of this work is, therefore, the design, implementation and evaluation of a novel agent-oriented system based on MAPS and a opportunely modified version of SPINE.

3.10.2.1 The A-SPINE Architecture

Figure 3.22 shows the architecture of A-SPINE through a class diagram. In particular, the core agents are defined in the following.

Base station-side:

- The *CoordinatorAgent* is responsible for managing the set of nodes of the sensor network under control. Management involves configuring and monitoring nodes;
- The *ApplicationAgents* are agents implementing application-specific or domain-specific logics;
- The *CommunicatorAgent* allows the CoordinatorAgent and the ApplicationAgents to interact with the sensor nodes through an efficient over-the-air application-level protocol.

Sensor node-side:

- The *SensorManagerAgent* manages the sensor/actuator resources of the node through specific *SensorAgents* able to interact with specific sensors (temperature, light, accelerometer, etc);
- The *CommunicationManagerAgent* manages the communication with the CommunicatorAgent and among the CommunicationManagerAgents, located at different sensor nodes, by means of specific CommunicationAgents;
- The *ProcessingManagerAgent* supports one or more local processing tasks or parts of global processing tasks through ProcessingAgents. They are able to perform computation on sensed data (e.g. feature extraction) and data aggregation.

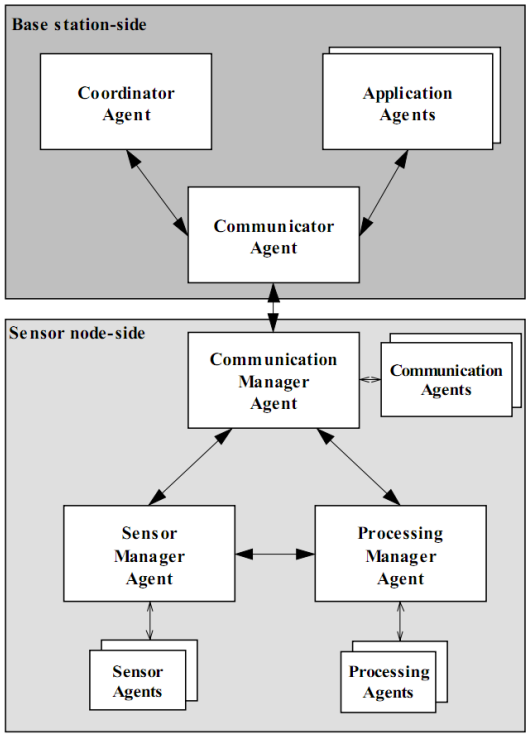


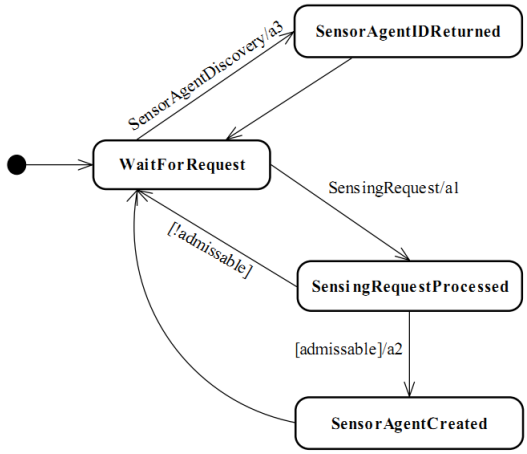
Fig. 3.22. The A-SPINE Architecture.

3.10.2.2 A MAPS-based design of A-SPINE

This paragraph describes the behaviors of the sensor node-side agents of A-SPINE designed through MAPS.

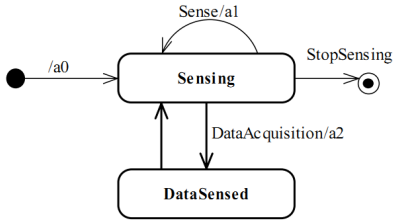
The behavior of the *SensorManagerAgent* consists of the state machine shown in Figure 3.23. It basically handles two events: `SensingRequest` and `SensorAgentDiscovery`. A SensingRequest can be admissible or not depending on the requested sensors: whether or not it is already in use (see action a1). In the former case, the SensorManagerAgent creates a SensorAgent with the sensing configuration parameters passed in the SensingRequest event (see action a2). A DiscoverySensorAgent event requests the identifier of the SensorAgent, if existing, attached to a given sensor type (see action a3).

The behavior of the *SensorAgent* is described by the state machine depicted in Figure 3.24. In particular, the `Sense` event is driven by a timer set according to the sensing sampling rate (see action a0). When the Sense is received, the sense operation is issued (see action a1) and, after data acquisition, sensed data are buffered into the data acquisition buffer/s of the SensorAgent (see action a2).




```
a1: SensingRequest e = (SensingRequest)evt;
    if (sensorAgents.get(e.getSensorType())==null)
       admissable=true;
    else admissable=false;
a2: create(genAgentID(),"SensorAgent", e.getConfParams(),
           local);
a3: SensorAgentType sensorType = (DiscoverySensorAgent)
                                 evt.getSensorAgentType();
AID sensorAgentID=sensorAgents.get(sensorType);
    send(self(),(DiscoverySensorAgent)evt.getSource(),
         "AgentID", <SAI=sensorAgentID>, true);
```

Fig. 3.23. The SensorManagerAgent behavior.

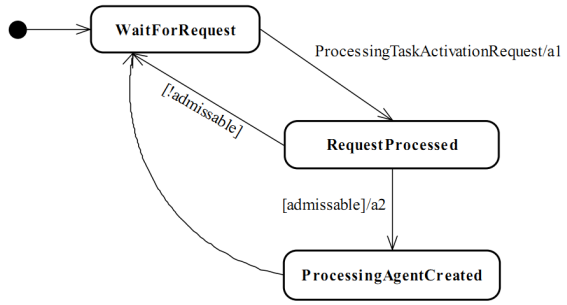



```
a0: Sense timer =new Sense(self(), self(),
                           Event.TMR_EXPIRED, Event.NOW);
    timerID = setTimer(self(), samplingTime, timer);
a1: DataAcquisition dataEvt = new DataAcquisition(self(),
                    self(), Event.SENSOR_TYPE, Event.NOW);
    sense(dataEvt);
a2: DataAcquisition e = (DataAcquisition)evt;
    buffer(e.getData());
```

Fig. 3.24. The SensorAgent behavior.

The behavior of the *ProcessingManagerAgent* is described by the state machine depicted in Figure 3.25. When the `ProcessingTaskActivationRequest` arrives, the ProcessingManagerAgent interprets the request and, if the request is admissible, creates a *ProcessingAgent* and links it to its input and output

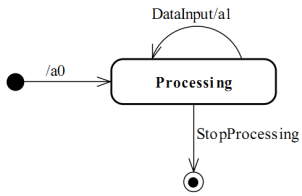
agents (i.e. agents providing data input to and receiving data output from the created ProcessingAgent).




```
a1: ProcessingTaskActivationRequest e =
                  (ProcessingTaskActivationRequest)evt;
    admissable=check(e.getRequest());
a2: String agentType=interpret(e.getRequest());
    AID agID = genAgentID();
    create(agID,agentType, e.getConfParams(), local);
    link(agID);
```

Fig. 3.25. The ProcessingManagerAgent behavior.

The state machine of the behavior of the *ProcessingAgent* is reported in Figure 3.26. After initialization (see action a0), the ProcessingAgent is able to receive DataInput events from its input agents and process them (see action a1). After processing, the output is sent to the attached data output ProcessingAgents.

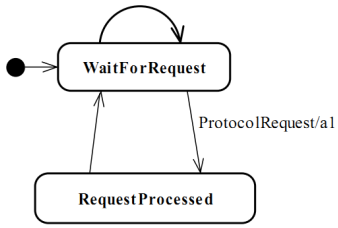



```
a0: initProcessing();
a1: DataInput e = (DataInput)evt;
    process(e.getData());
```

Fig. 3.26. The ProcessingAgent behavior.

The basic behavior of the *CommunicationManagerAgent* is described by the state machine depicted in Figure 3.27. The `ProtocolRequest` event encapsulates the packet of the interaction protocol with the CommunicatorAgent at the base-station; once the event is received the CommunicationManagerAgent processes it according to the protocol (see action a1) or routes it to the target manager agent, if it is not able to handle it. A ProtocolRequest involving data

transmission from the node to the base station or to another node can also be requested by a *SenderAgent*.




```
a1: ProtocolRequest e=(ProtocolRequest)evt;
    process(e.getRequestType());
```

Fig. 3.27. The CommunicationManagerAgent behavior.

3.10.3 SPINE2

SPINE2 [56, 57] represents a significant variant to the original framework. While retaining the same fundamental goals and concepts, SPINE2 has been, instead, designed around a *task-oriented* paradigm.

This paradigm provides a way to conceive a graphical high-level behavior of the application, like the one shown in Figure 3.28. In this particular example, the max, mean and min values of a series of temperature data coming from a sensor are evaluated, and the results are transmitted to another node or to the coordinator.

As a consequence to the simple definition, this choice implies the design and the implementation of an appropriate runtime system for interpreting and executing these applications specifications.

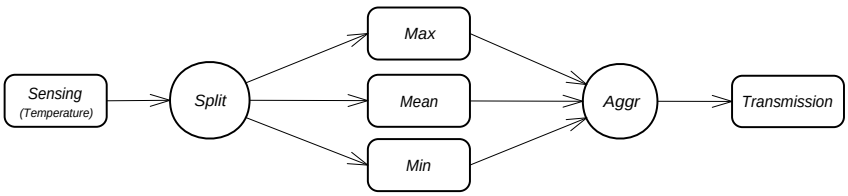


Fig. 3.28. Example of a task-oriented application.

The task-oriented paradigm adopted by this variant of SPINE comes along with a high-level language, which exposes a set of constructs expressly defined for supporting such a approach. These constructs represent the elementary concepts through which users specify the behavior of the applications. This

modeling language is shown to be simple, yet expressive enough for becoming the preferred method used for describing a typical distributed application aimed to data and signal processing. Moreover, its intrinsic nature allows the required reusability and reconfigurability of the application to be satisfied. The details and the benefits in using the task-oriented paradigm are described in the following section.

The main motivation that has led to a task-oriented architecture is that most of the current middlewares provide high-level services for data collection and querying but they do not allow to define an explicit data flow processing which is very useful in many application domains, such as context recognition, health monitoring and medical assistance. Furthermore, differently from many others middleware, this new framework is conceived to support an application definition method that avoid users the burden of writing any programming code.

With respect to this methodology, an application can be simply specified as a set of tasks connected together. Each task represents a particular activity, such as a sensing operation, a processing function or a radio data transmission. The user has only to select a certain number of tasks from a set of available ones on the basis of the application requirements. Afterwards, the user has to link together pairs of tasks with a connection if necessary, so that the output result of the one correspond to the data input of the other. In this way, the set of connected tasks form a direct graph which defines the work flows performing a series of operations on the sensor data and so represents the high-level description of the whole application.

Typically, a data processing application supported by the framework consists in (1) accomplishing the needed sensor readings, (2) passing the sensed data to processing functions which carry out some signal processing operation, and (3) sending result to other nodes of the network (eventually for further data elaboration).

It is worth noting that, as the framework supports a distributed data processing, users can decide where every task forming the application is allocated over the sensor network. This is shown in Figure 3.29. Each single task is performed on a particular node, guaranteeing that the execution of the application is maintained well balanced. Depending on the different features of the nodes constituting the network, the user can allocate the tasks requiring more resources to node providing more computational capabilities.

For what concern the software architecture, the framework is composed of two components, one is implemented on the *coordinator* of the WSN, the other is implemented on the *sensor nodes*. The former is a Java application running on a laptop or a hand-held device through which the user configures and manages the sensor network and the task-application to be deployed on it. The latter represents the middleware engine running on top of the sensor node operating system. It is responsible of handling the messages coming from the coordinator which are used, among the other things, for configuring the

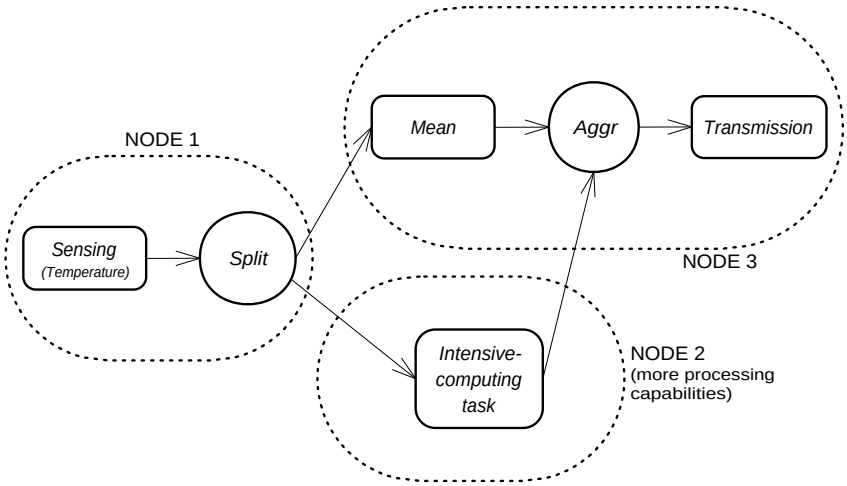


Fig. 3.29. Application example having tasks instantiated on different nodes.

portion of the user application assigned to the node. It is also in charge of managing and executing the tasks that are instantiated on the node.

Currently, the node-side part of the framework has been implemented and tested on the TelosB sensor platform running TinyOS 2.x, and on a Z-Stack compliant sensor platform. However, one of the distinctive characteristics of SPINE2 is its platform independence that allows for fast porting to others C-like software architectures.

3.10.3.1 The task-oriented approach

The way of modeling a sensor network application through a task-oriented methodology aims at providing an abstract description of the actual application running on the nodes by omitting low-level details and thus reducing complexity which is usually inherent in such a distributed software.

The basic blocks contained in this formalism are:

- *tasks*, and
- *task-connections*.

A task represents a well defined node activity which can consist, for instance, in a processing operation rather than a data transmission or a sensor reading. Tasks execute *atomically* among each other. However, asynchronous events, such as a radio message reception or a timer expiration, may preempt a running task.

A task-connection represents a relationship between tasks which generally consists in having some kind of dependency, such as temporal and data dependency. Furthermore, these tasks relationships are semantically consistent thanks to the well-defined input and output interfaces.

Such a system representation, which capture both data and control flow, allows for a better application definition that in turn will lead to effective scheduling activities and in general to a more efficient system implementations. Designing an application as a composition of elementary blocks with fixed interfaces enables rapid application reconfiguration as well as more simple application maintenance.

Moreover, a system adopting a task-oriented approach can easily be enhanced in functionality, by simply adding new task definitions which represent further computing capabilities. Most important, adding these new blocks will not imply the need of changing definition of any others, neither it needs the underlying task management software to be modified. This is achieved because tasks are decoupled among each other, and the only relation point is through their input/output interfaces (i.e. data they need/provide).

3.10.3.2 Main characteristics of SPINE2

This section describes the main aspects characterizing SPINE2.

Platform independence and quick portability: these are two very important factors to consider because the success and the wide diffusion of a middleware depend on how many platforms it can support. And this is particularly true for sensor networks, considering that at the present low-cost mass production has permitted the development of a wide variety of sensors platforms, and also it is not unusual that a single application may be deployed on a WSN including different sensor architectures (heterogeneous sensor network). So, one of the requirement for an application development tool is that of being predisposed, since its design phase, for a rapid and simple portability process towards different sensor architectures (considering both hardware and software).
At the present most of the sensor platforms (and their operating systems) supports the C programming language. However, simply using C to implement framework is not enough for reaching the platform independence and for enhancing its portability. Therefore, the node-side software architecture is conceived for decoupling the task runtime logic from all what is concerned with services and features provided by the operating system of a particular platform. For this purpose, the *software layering* approach has been adopted (see Figure 3.30).
According to such approach, the node-side framework is designed so that a set of "core modules", developed in C and representing the actual runtime system, constitutes the part of the software which can be used on every C-like sensor platform without the need for any changes. Along with these modules, other components constitute the *platform-dependent* part of the architecture, and they represent the adaptation interfaces between the core runtime system and the services and resources (sensors, timers, communications) provided by the underlying environment system of a particular target sensor platform (such as TinyOS, or Z-Stack). To make a

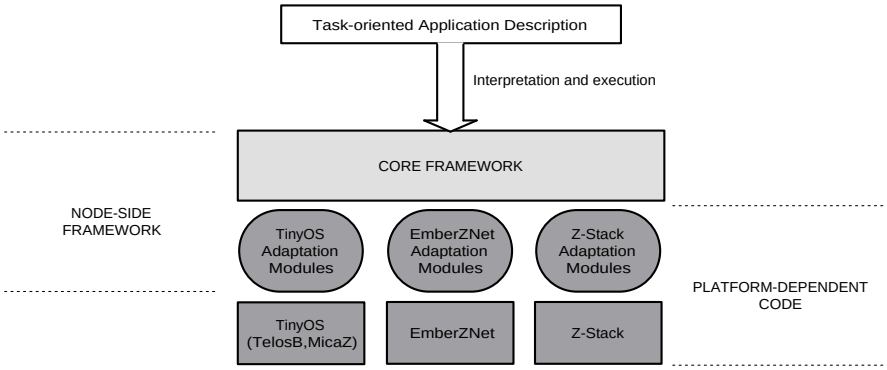


Fig. 3.30. The *Software Layering* approach for developing the framework.

porting of the framework to a new sensor platform, the latter components are the only software that a developer has to provide.

Extensibility: a middleware should provide a way for allowing developers to easily improve it with possible enhancements because a constraint in this sense may limit its use in the future. The chosen task-oriented design methodology is a perfect example of how is possible a straightforward approach for adding new functionalities beside to the existing ones. This is done by simply defining new tasks which represent further computing capabilities and developers do not have to change the underlying runtime logic or the other task definitions, thanks to the fact that the runtime does not care about what tasks do whereas every task is decoupled with each other. The framework also allows a convenient mechanism for supporting new hardware resources such as sensor types or actuators.

Modularity: concerning the design of complex software systems, it is always useful to conceive an architecture composed by several modules, each of them devoted to a particular purpose and interacting each others through well-defined interfaces. The approach of defining a modular entity constituted by different and independent functional blocks allows a more rapid implementation time, a more effective software maintenance, and improvement of functionalities. For example, it may be possible that future requirements need a different way for managing the memory or the tasks execution. Thanks to the modularity, the modifications made by the framework developers affects only the correspondent modules without interfering with the rest of the architecture.

Flexibility for the final user: the success of a development tool also depends on the flexibility that it provides to the application developer by avoiding, as much as possible, constraints, and limitations during design, and implementation. In the particular case of SPINE2, a user can define how many tasks he needs, setting values for their parameters with a wide freedom of choice and maintaining a high abstraction on the real capabil-

ities of the node. The only limitation is dictated by the actual amount of physical resources on the node.

3.10.3.3 SPINE2 Tasks

The various typologies of *task* represent the fundamental elements of the language. They can be subdivided into two main categories: *data-processing tasks* and *data-routing tasks*. The former tasks perform functions related to data processing and execution control, whereas the latter ones provide store-and-forward and data replication functionalities.

The hierarchy of the different types of task defined so far are shown, along with their purposes.

- *TimedTask*, is the super-category that includes every temporized task:
  TimingTask: allows to define timers for timing other tasks.
  SensingTask: defines sensing operations on a sensor node and include a timer for setting the sampling time.
- *FunctionalTask*, includes tasks for data manipulation:
  ProcessingTask: performs data processing functions and algorithms, allowing to specify the type of operation to accomplish; particular operations are the so called "feature extractions" which are mathematical function applied to a data series, such as Mean, Variance, etc.
  TransmissionTask: allows an explicit transmission of data generated by other tasks, sending them to a specific addressee node. Generally it is used for sending data and information to the coordinator whereas, implicit data transmissions take place in the case of connected tasks located into different nodes.
- *FlashingTask*, this category allows to use the on-board flash memory:
  StoringTask: stores data coming from its input on the flash memory.
  LoadingTask: retrieves data from the flash for being used by other tasks of the application.
- *DataRoutingTask*, comprises:
  SplitTask: duplicates data of its input to every output links for making them available to other tasks.
  AggregationTask: collects data coming from its multiple inputs carrying them to output.
  HistoricalAggregationTask: similar to the previous task but supporting a series of aggregation operations over the time, before bring them to output.

3.11 Virtual Sensors based on SPINE

This section presents a *multi-layer task model* based on the concept of *Virtual Sensors* to improve architecture modularity and design reusability [56].

Virtual Sensors (VSs) are abstractions of components of BSN systems that include sensor sampling and processing tasks and provide data upon external requests. The Virtual Sensor model implementation relies on SPINE2, presented in Section 3.10.3.

3.11.1 BSN-oriented Virtual Sensor Architecture

Physical sensors map an observed physical quantity, such as temperature, acceleration, or sound, onto a data value and produce an output. The output is generated when inputs change, as the result of an event, or in response to a (timed) request. Physical sensors are transducers converting values from one form to another using physical processes. Signal processing algorithms convert values using digital processes. This observed similarity is the motivation behind the virtual sensor abstraction.

Every processing task can be represented as a virtual sensor. Therefore, considering a complete BSN system, its data processing part can be modeled as a multi-level hierarchy of virtual sensors, as shown in Figure 3.31. Moreover, virtual sensors may be implemented directly in a programming language, or as networks of already existing virtual sensors.

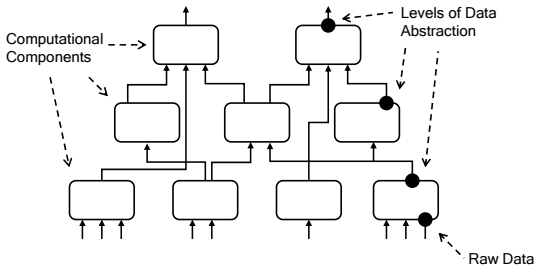


Fig. 3.31. Multi-layer Signal Processing

Figure 3.32 shows the defined BSN-oriented virtual sensor system architecture. A user requests certain outputs given specified inputs. This request is handled by the *Virtual Sensor Manager*, which configures a set of virtual sensors to handle the computational task. Virtual sensors use the *Buffer Manager* to setup communication through the use of efficient buffers. Once configured, the system is activated, and virtual sensors cooperate to produce the final outputs.

Virtual sensors are defined in the following section. Details of the virtual sensor manager operation are described in Section 3.11.1.2. Finally, buffer manager operation is described in Section 3.11.1.3.

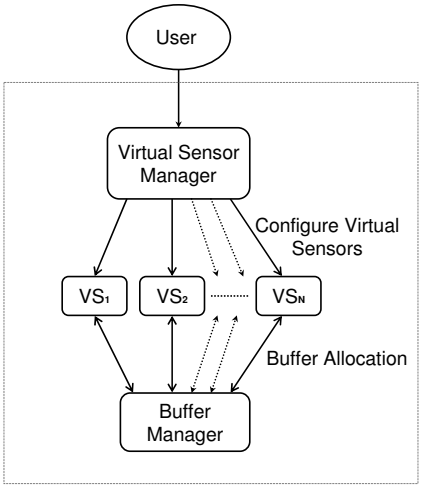


Fig. 3.32. BVS Architecture

3.11.1.1 Virtual Sensor Definition

Software frameworks are usually introduced to provide programmers with abstractions to isolate them from low-level implementation details. Virtual sensors provide a new level of abstraction at the software level by allowing signal processing tasks to be defined and composed easily. Furthermore, VS abstractions allow signal processing tasks to be modified or changed at design or runtime without affecting the rest of the system. In Figure 3.31 every component represents a processing task applied to a stream of data originated from physical sensors and can be modeled as a virtual sensor. The output of each virtual sensor is defined by a set of inputs and its configuration. More formally, a virtual sensor i, denoted as VS_i, is defined as:

$$VS_i = \{I_i, O_i, C_i\} \tag{3.1}$$

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

$$C_i = \{\vec{t}_{in}, \vec{t}_{out}, d, p\} \tag{3.2}$$

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

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

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

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

$$VS^* =< VS_1, VS_2, ...VS_n >= \{I^*, O^*, C^*, L\} \tag{3.3}$$

where $I^* \subseteq I_1 \cup I_2 ... \cup I_n$, $O^* \subseteq O_1 \cup O_2 ... \cup O_n$, $C^* = \{C_1, C_2, ...C_n\}$, and L is the set of links connecting outputs and inputs of $\{VS_1,...,VS_n\}$

3.11.1.2 Virtual Sensor Manager

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

The current configuration of a virtual sensor may be invalidated by changes in its inputs or connections with other virtual sensors, therefore reinitialization could happen at any time. For example, Figure 3.33(a) describes a system that takes a temperature reading in Fahrenheit, and a heart rate in beats per minute. In Figure 3.33(b) a new thermometer, that produces output in Celsius, is introduced. VS_1 has to be reconfigured to handle such change. To be able to configure/reconfigure the system at run time, the VSM manages a table that maps each available combination of possible inputs and outputs to the appropriate virtual sensor implementation. This can be represented by the set A. Each entry $a \in A$ is defined as:

$$a = \{\vec{t}_{in}, \vec{t}_{out}, \psi\} \tag{3.4}$$

where ψ is a particular virtual sensor implementation.

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

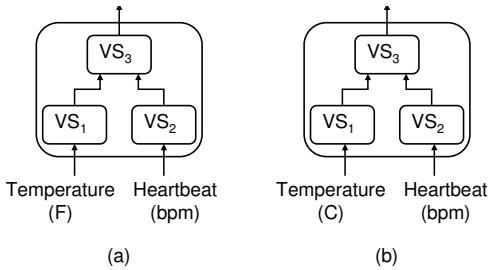


Fig. 3.33. Example of Input Modification in Virtual Sensors

While individual virtual sensors do not hold any information about other virtual sensors, the overall system relies on their cooperation. At the beginning of the system execution, the VSM receives the VS topology configuration graph. Based on the requirements of the topology configuration, the VSM initializes the appropriate VSs and connects them as required. Input and output types are a property of each virtual sensor. An output of one of the virtual sensors can also be an input of another virtual sensor. For example, in Figure 3.34, configuration of VS_3 and VS_4 depends on the input they receive from VS_1. To simplify the configuration and reconfiguration process, the VSM initializes VSs in a specific order, to meet the requirement that each virtual sensor cannot be created until all inputs are configured. This ordering can be determined with a topographical sort of the topology configuration graph.

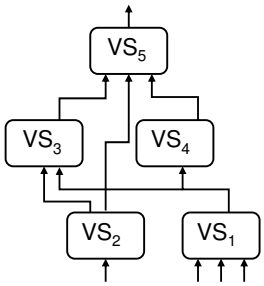


Fig. 3.34. Example of Input/Output Dependency in Virtual Sensors

3.11.1.3 Buffer Manager

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

When a virtual sensor is created and configured, it initiates a data buffer for its output. The virtual sensor contacts the BM and requests the creation of a buffer sufficient to hold its output. The BM allocates a circular buffer of the required size and returns the `bufferID`. This `bufferID` is propagated by the VSM to other virtual sensors that are interested in data of this particular buffer. To read from a buffer, a virtual sensor must register with the buffer as a reader, specifying the number of samples it can consume at a time. Every time the producer writes to the buffer, the BM checks if the buffer has enough information for any of the readers, and signals them when they can access the data. Figure 3.35 shows an overview of the BM operation. In particular, it shows that BM keeps track of buffers by ID, tracking the point where the producer (e.g. W) is writing to, and where each individual reader (e.g. R_1, R_2) is reading from. If the producer VS is reconfigured, and its output is changed, the BM removes the buffer that is associated with the previous output and initiates a new buffer, based on the new configuration information.

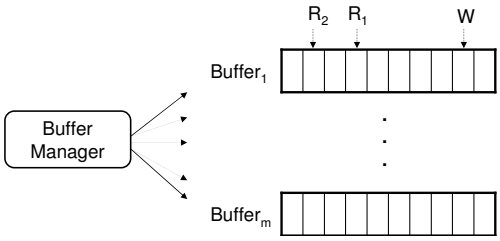


Fig. 3.35. Buffer Manager Overview

3.11.1.4 SPINE2-based Virtual Sensors

The Virtual Sensor architecture described in Section 3.11.1 is straightforwardly implemented through the SPINE2 framework. In Figure 3.36 basic conversion schemas for the translation of Virtual Sensors into SPINE2 task-oriented applications are shown. In particular, only simple (flat) virtual sensors have been taken into consideration as it is quite intuitive to translate a virtual sensor defined as composition of flat virtual sensors.

In the most simple case, a virtual sensor defined as a basic functional block incorporating some kind of operation on its single input can be translated into

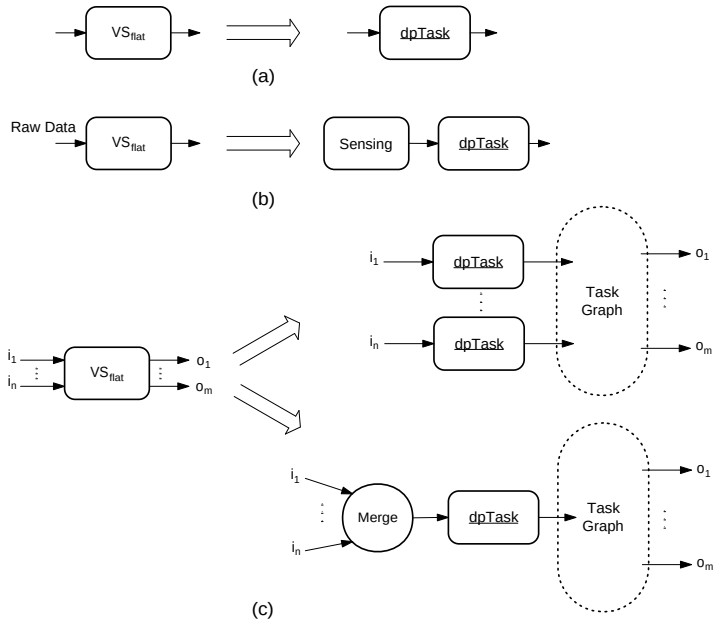


Fig. 3.36. Translation of Virtual Sensors into SPINE2 task-oriented models

a SPINE2 *data-processing task* (see Figure 3.36(a)). In fact, a generic SPINE2 data-processing task (such as *dpTask*) is defined as a functional component having a single input and a single output, differently from the data-routing task. Obviously, the operations that have to be performed by the task (specified by its configuration) depend on the actual functionalities of the virtual sensor. If the virtual sensor does not have a generic input but raw data (such as data coming form a hardware sensor on a wireless node), the corresponding translation includes the introduction of a SensingTask, specifically configured for representing the digital data source (see Figure 3.36(b)). Finally, Figure 3.36(c) shows the translation of a simple virtual sensor having multiple inputs and outputs. In this case, the corresponding SPINE2 tasks can be configured in several ways on the basis of the actual definition of the virtual sensor and of the type description of its inputs/outputs. In particular, two different translations are shown. In the first one, there is a single data-processing task for each input. These tasks, along with the not-specified Task Graph, carry out the overall computational operation performed by the virtual sensor. Conversely, in the other translation, inputs are merged by a single data-routing task (namely, the MergeTask) and provided to a generic task graph. In either case, the more complex the function defined for the virtual sensor gets, the more complex the set of actual interconnected tasks would be. Of course, there could exist a more generic SPINE2 translation in which some of the

inputs merge on an AggregationTask, the other ones become inputs of data-processing tasks.

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

3.12 Summary

In this Chapter, *SPINE*, a domain-specific programming framework, has been presented. One of the main achievements of 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 without off-line re-programming before switching from an application to another. Furthermore, thanks to its modular component-based design approach, SPINE enables a great degree of heterogeneity, and a wide variety of hardware platforms, sensors, programming languages and operating systems are supported. This allows for a very flexible and usable framework in different BSN application scenarios, where, due to specific requirements, only certain platforms or operating systems might be used.

Additionally, other interesting contributions which are the result of enhancements and variants to the original proposed framework, have presented here. They include a task-oriented framework re-design of SPINE (*SPINE2*), an enhancement for supporting collaborating BSNs (*C-SPINE*), a multi-agent model for BSN programming (*A-SPINE*), and a programming paradigm based on the concept of *Virtual Sensors*. Particularly relevant is the task-oriented re-design in which we achieved the significant result of *platform-independence* for most of the framework components, and the multi-agent model as we showed that, at least when implemented on more powerful sensor nodes, it allows for even more flexibility, also thanks to the code mobility which can be more naturally enabled by means of the *mobile agent* concept.

4

BSN Research prototypes implemented using SPINE

This chapter emphasizes how the proposed SPINE framework is actually able to support the development of heterogeneous health-care applications based on reusable subsystems. Indeed, one of the main goal of SPINE is to provide a flexible architecture that can support variety of practical applications without the need for costly redeployment of the code running on sensor nodes.

This chapter therefore introduces some interesting research BSN systems that have been developed atop SPINE. Furthermore, each of the proposed applications improves the current state-of-the-art, as described in the following.

4.1 Physical Activity Recognition

The human activity monitoring system prototype here presented is able to recognize postures (*lying*, *sitting*, and *standing still*) and a few movements (*walking*, and *jumping*) of a person; furthermore it can detect if the assisted living has fallen and is unable to stand-up.

The wearable nodes are based on the Tmote Sky platform to which is attached a custom sensor-board (SPINE sensor-board) including a 3-axis accelerometer and two 2-axis gyroscopes. The nodes are powered by a standard 3.7V, 600mAh Li-Ion battery). The end-user application is implemented in the Java language and runs on top of a SPINE coordinator laptop to which is attached a Tmote Sky, acting as a base-station bridge, connected via USB port.

The activity recognition system prototype relies on a classifier that takes accelerometer data measured by sensors placed on the waist and on the thigh of the monitored subject and recognizes the movements defined in a training phase. Among the classification algorithms available in the literature, a *K-Nearest Neighbor* [58] (KNN)-based classifier has been selected.

The prototype provides a default training set and a graphical wizard to let the user build his own training set to enhance recognition accuracy. The significant features, that will be eventually activated on the sensor nodes to

Table 4.1. Posture/Movement recognition accuracy.

Sitting	Standing	Lying	Walking	Falling
96%	92%	98%	94%	100%

classify the movements, are selected using an offline *sequential forward floating selection* (SFFS) [59] algorithm. Experimental results have shown that, given a certain training set, the classification accuracy is not significantly affected by the K value nor by the distance metric used by the classifier. This is because, thanks to the accurate selection of the signal features, the activities instances form clusters that are internally very dense, and well separated among each other. Therefore, the classfier parameters have been selected as follows:

- K = 1;
- Metric distance: *Manhattan*.

For the feature selection, the accuracy has been calculated with a shift of 50% of the data window, using half of the dataset for training and half to test the classifier.

The resultant most significant features are:

- *waist node*: mean on the accelerometer axes XYZ, min value and max value on the accelerometer axis X;
- *leg node*: min value on the accelerometer axis X.

As previously mentioned, the proposed system also includes a *fall detection* module which is implemented on the waist sensor node and can be activated/deactivated at run-time. When the fall detector is active, every time that new accelerometer data are acquired, one of the threshold-based functions checks if the *total energy* (as defined in Table 3.3) of the accelerometer signals, exceeds an empirically-evaluated threshold. If so, it sends an alarm message back to the coordinator to inform the user application. False alarms are drastically reduced by a simple mechanism implemented directly at the end-user application: as soon as it receives a fall-detected message, the system waits the recognition of the next seven postures of the person; only if it evaluates 4 out of 7 lying positions, an emergency message is reported to the user attention.

An interesting functionality of the prototype is a simple tool for adding new, user-defined activities among the default ones. The tool drives the user through a simple procedure for acquiring the necessary training data which are then stored in the global data set.

Although the objective of this prototype concerned mainly in testing the SPINE framework in a semi-realistic use case, the overall performance (see Table 4.1) reached by the recognition system is considerably high, with an average posture/movement classification accuracy of 97%. The fall detection

algorithm is quite accurate as well, as it is able to detected almost every falls, and reaching a low percentage of false alarms.

4.2 Step-counter

This section describes an innovative step-counter algorithm, whose main design requirements are the followings:

- Use of accelerometer data;
- Energy and computation efficient design to support embedded implementations;
- Use of a single sensor node, placed on the waist (below the navel);
- General-purpose algorithm, to be used by healthy people as well as elderly and/or people with disabilities;
- No need for "ad-personam" calibration;
- High average accuracy (*robustness*).

Several real walk data on different subjects have been collected and studied before starting the algorithm design. The subjects were asked to walk naturally, and to increase/decrease the walking speed occasionally. In particular, a single three-axis accelerometer sensor node was placed on the waist while recording. The sensor has been sampled at 40Hz.

In summary, the preliminary conclusions from this analysis are:

- An off-line downsampling to 20Hz showed that the signal is still well characterized;
- The lateral acceleration presents a poor SNR (Signal-to-Noise Ratio) as the waist swing during walking is heavily influenced by noise;
- Frontal (horizontal) and vertical acceleration present both very useful signals; for both, it looks roughly sinusoidal, because the waist is interested by vertical accelerations/decelerations when the feet hit the ground, and by horizontal accelerations/decelerations when the body swings frontally while walking;
- Interestingly, the horizontal acceleration signal looks generally cleaner than the vertical acceleration signal.

A number of approaches have been considered and evaluated, eventually converging to the proposed solution. To simplify the development, debugging, and evaluation, the implementations have been initially programmed in *Matlab*. Only *integer-math* computations were used, so allowing for a more straightforward embedded implementation (as the target embedded platform is based on a microcontroller with no hardware support for floating point operations).

A very simple "*fixed threshold-range*" technique has been studied at the beginning. The thresholds were empirically determined by observing several real walk data on different subjects. In this approach, a step is detected if the

instantaneous raw acceleration stays within the defined range. To remove multiple detections of the same step, the algorithm sleeps for the shortest "step-time", measured from the available observations. Although the algorithm is very simple, it showed high precision ($> 90\%$ in lab experiments) while applied to healthy people, regardless the gender, weight, height, and walking speed. However, when applied to elderly people, it behaves very poorly, with almost no steps detected. Intuitively, that is due to less pronounced movements of walking elderly people that lead to lower accelerations of the waist.

An enhanced algorithm followed a different approach. Rather than defining *a priori* the threshold range, it is possible to determine the best threshold for the monitored subject by running a search algorithm that requires a set of walking data of that subject, the number of steps inside that recording, and the required accuracy. The threshold search is performed iterating the step-counter algorithm previously described on the walking data, trying a decreasing threshold (initially set to the upper accelerometer scale value) and comparing the number of detected steps against the actual number. The search stops when the required accuracy has been reached or the number of detected steps exceeds the actual number, and the threshold found is returned. This implementation performs overall better than the first one, but requires a training phase and the manual counting of the steps during the training. If the counting is incorrect and/or during the recording a significant number of data packets get lost, the threshold search can be compromised.

The first algorithm does not recognize effectively the steps of elderly people. On the other hand, the second algorithm needs a preliminary training phase. Furthermore, they both work on the raw accelerometer data.

To overcome these limitations, a third algorithm has been defined, borrowing some ideas from the previous ones. At the same time, additional real walk data have been recorded from hospitalized subjects (elderly and people with walk disabilities).

It is worth reminding that the frontal acceleration of the waist presents a signal roughly sinusoidal while walking. In few words, the new algorithm attempts at detecting steps by identifying the decreasing segment (*falling edge*) which corresponds to the last fraction of a step movement.

A step is characterized by time constraints (it cannot be "too" fast or "too" slow). However, walk patterns change from people to people and even for the same person it might change from time to time; hence, the amplitude of the acquired signal can vary significantly.

To simplify the pattern recognition, the raw acceleration is first processed with a *smoothing filter* which removes the high frequency components. Then, it looks for local maximums. When a local max is found, it looks for a local minimum. After the local min is also found, the candidate segment is identified as well. Two features are then extracted and used to determine whether the candidate belongs to a real step or not. The candidate is classified as *step* (i) if it has an acceleration drop within a certain range (specified by a "*tolerance*" parameter around a threshold), and (ii) if the time it lasted is within a certain

interval. More specifically, the pre-processing is a 9-point windowed smoothing filter which uses gaussian kernels. Because they are applied to a digital signal, the sum of the kernels must be 1. Furthermore, because the algorithm work on integer-math, they are scaled so that decimal factors are removed.

The following Gaussian kernels have been selected:

- { *5*, *30*, *104*, *220*, *282*, *220*, *104*, *30*, *5* }

The threshold is coarsely initialized, but it is automatically adapted while steps are recognized. In particular, it is continuously updated with the average of the last 10 acceleration drops that were classified as steps.

Since the threshold is just coarsely initialized at the beginning and because even following steps can occasionally present significant variance in their vigor, the updated threshold is adjusted with parametric coefficients to specify the amplitude range in which a candidate must fall.

Finally, to reduce "false positive" recognitions, e.g. due to sudden shocks or slow tilts of the sensor, the time elapsed between the local max and min (which it is simply determined as the product between the number of samples of the segment and the sampling time) must be longer than the "*minimun step time*" and shorter than the "*maximum step time*". Both values have been determined empirically from the available observations. In particular, based on the available walking data available, the following values have been identified:

- $mininum\ step\ time = 350ms$
- $maximun\ step\ time = 2000ms$.

Figure 4.1 shows the block diagram of the proposed algorithm.

The proposed algorithm has been initially evaluated on the computer, and finally implemented on a wireless sensor node running SPINE. For this application, the node-side of SPINE has been extended with the proposed algorithm. Every time the node detects a step, it communicates to its coordinator the total numer of steps taken so far (this to avoid mis-counting due to lost packets). On the coordinator, very minor additions have been made to the core framework, and a simple graphical application shows in real time the number of steps.

As a running example of the algorithm, Figure 4.2 shows the raw data of the frontal acceleration of the waist during normal walking of a healthy subject, sampled at 20Hz. The "end" of each step (which then brings to the beginning of the following one with the other foot), corresponds roughly to the sharp low spikes in the plot.

Figure 4.3 shows the result of the Gaussian filtering of the data in Figure 4.2; the small black dots in Figure 4.3 correspond to where the step-counter algorithm has detected steps.

Although the strength of the steps sometime change significantly, the algorithm still detects properly all the steps.

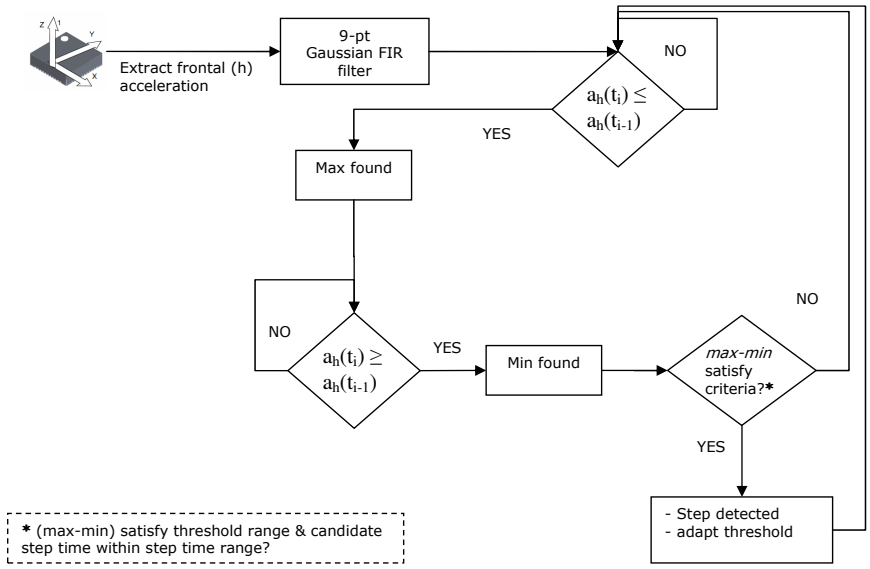


Fig. 4.1. Block diagram of the step-counter algorithm.

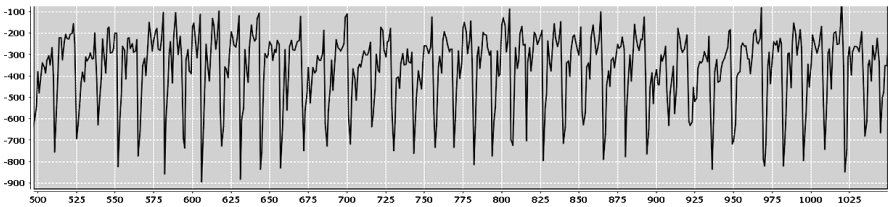


Fig. 4.2. Raw Data of the frontal (horizontal) acceleration of the waist during normal walking.

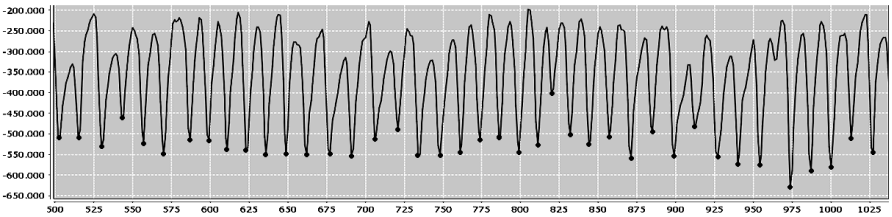


Fig. 4.3. Result of the Gaussian filtering of the data shown in Figure 4.2.

The proposed step-counter algorithm has been tested on several subjects, from healthy young people to elderly and/or with disabilities. Over 40 tests on 8 healthy subjects (both males and females, with different height and weight) and 6 hospitalized subjects (with post-stroke disabilities, walking using crutch

or wheels walkers, or simply elderly), have shown overall error (detected steps vs actual steps) of 12%.

The worst under-estimation was 27%, and the worst over-estimation was 18%. It is worth noting, however, that the subjects were not video recorded during their walking data collection; hence, it runs out that is sometimes difficult to validate the number of reported steps (manually counted by direct observation during walking) in each experiment. Furthermore, some lost packets caused discrepancy with the number of reported steps against the number of recorded ones.

In conclusion, the main contribution of the proposed algorithm is the ability to provide a good estimate of the steps taken by people with disabilities and/or using crutch or wheels walkers. To the best of our knowledge, this is the only system being able to work properly for the latter category of users.

4.3 Real-time Physical Energy Expenditure

Accurate measurements of physical activity are important for obesity research and intervention programs, for fitness and wellness applications, and so on. Physical activity assessment may be used to establish baselines and changes that occur over time. Quantitative assessments may be used to gauge whether recommended levels of regular physical activity are being met, such as 60 and 30 minute guidelines for daily moderate intensity activity established by the National Academy of Sciences [60] and U.S. Surgeon General [61], respectively. Conversely, measurements of physical inactivity associated with sedentary lifestyles can equally be useful in estimating risk for overweight and obesity.

To provide this quantitative assessment, we developed an energy expenditure algorithm based on previous research conducted by Chen and Sun in which 125 adult subjects wore a 3-axis accelerometer at the waist position for two 24 hour periods in a controlled air-tight environment, where whole-room indirect calorimetry was computed based on 1-minute measurements of O2 consumption and CO2 production [62]. Energy estimates from triaxial accelerometry were found to be well-correlated (Pearson's r = 0.959) with total energy expenditure measured from the room. The same study used generalized linear and nonlinear models to estimate energy expenditure from the raw activity counts along the vertical and horizontal axes.

The problem of many accelerometry-based approaches is an assumption that the sensor is oriented correctly to give accurate activity counts along the relevant axes. This can be difficult to guarantee, particularly for obese subjects, where the sensor may tilt on the waist with activity or changes in posture.

Our Kcal algorithm, instead, improves the current state-of-the-art, by providing a *dynamic compensation of the gravity vector* affecting the accelerometer readings. We first applied time-averaging and vector projection to obtain

vertical and horizontal axes regardless of sensor orientation [63]. Briefly, the approach isolates perturbations around a time-averaged (or smoothed) acceleration vector, which indicates the direction of gravity. We compute a vector projection onto this time-averaged representation of gravity to obtain activity counts along the vertical axis, and through a vector subtraction obtain the counts in the horizontal axis. Hence, given the smoothed acceleration vector v that approximates the gravity vector, and an acceleration vector at a given time a, the perturbation is:

$$d = a - v \tag{4.1}$$

The vertical component of this perturbation is computed through vector projection as:

$$p = \left(\frac{d \cdot v}{v \cdot v}\right) \cdot v \tag{4.2}$$

The horizontal vector is the subtraction of p from the perturbation vector d:

$$h = d - p \tag{4.3}$$

Counts along the vertical and horizontal axes, computed as summations of vector magnitude over a period of time, were used as input into the Chen and Sun generalized models described above.

The nonlinear equation accounts for variations due to subject weight and gender. Briefly, the equation for energy expenditure EE (in KJ) is based on horizontal and vertical activity counts, H and V for the k-th minute, respectively:

$$EE(k) = aH^{p_1} + bV^{p_2} \tag{4.4}$$

And where a and b are generalized estimates based on the subject's weight w (in Kg) (original equation from [63] differentiates by gender):

$$a = (12.81w + 843.22)/1000 \tag{4.5}$$
$$b = (38.9w + 10.06)/1000 \tag{4.6}$$
$$p_1 = (2.66w + 146.72)/1000 \tag{4.7}$$
$$p_2 = (-3.85w + 968.28)/1000 \tag{4.8}$$

We have implemented the proposed algorithm using the SPINE framework. It partially runs on the sensor node, where we added a new processing functionalities to compute the activity counts, and partially atop the SPINE coordinator, where a graphical application, using the activity counts received every seconds by the sensor node, computes the final estimation of the energy expenditure using the formulas shown above after collecting one minute of observations.

4.4 Emotional Stress Detection

The *Heart Rate Variability* (HRV) is based on the analysis of the R-peak to R-peak intervals (*RR-intervals* - RR_i) of the electrocardiogram (ECG) signal in the time and/or frequency domains. Doctors and psychologists are increasingly recognizing the importance of HRV.

A number of studies have demonstrated that patients with anxiety, phobias and post-traumatic stress disorder consistently show lower HRV, even when not exposed to a trauma related prompt. Importantly, this relationship exists independently of age, gender, trait anxiety, cardio-respiratory fitness, heart rate, blood pressure and respiration rate.

This section presents a toolkit based on BSN for the time-domain HRV analysis, named *SPINE-HRV* [64]. The SPINE-HRV is composed of a wearable heart activity monitoring system which continuously acquires the RR-intervals, and a processing application developed using the SPINE framework. The RR-intervals are processed using the SPINE framework at the coordinator through a time-domain analysis of HRV.

The analysis provides seven common parameters known in medical literature to help cardiologists in the diagnosis related to several heart diseases. In particular, SPINE-HRV is applied for stress detection of people during activities in their everyday life.

Monitoring the stress it relevant as many studies show connections between long-term exposure to stress and risk factors for cardiovascular diseases [65, 66].

The main contribution of the proposed system relies in its comfortable wearability, robustness to noise due to body movements and its ability to identify emotional stress in real-time, with no need to rely on off-line analysis.

4.4.1 Hardware

The hardware architecture of our system in composed of a wireless chest band, a wireless wearable node and a coordinator station. The wireless chest band detects heart beats and transmits a pulse message over the air each time a heart beat has been detected. It does not require manual power-on nor software configuration. The wearable node is a Telosb mote equipped with a custom board that has a dedicated receiver for the heart beat messages sent by the chest band. Specifically, the wearable node runs the TinyOS operating system and is powered by the SPINE framework. The coordinator station is a PC running a Java application built atop SPINE, which allows bidirectional communication to setup the wearable node and retrieve the heart beats.

4.4.2 Software

The wearable mote runs the SPINE framework, which has been extended with a custom defined processing function to support the custom sensor board.

Once enabled by the SPINE coordinator, the processing function on the wearable node starts timestamping the heart beat events, to transmits back to the coordinator the RR_i values.

The RR_i data is used by a Java application built atop SPINE, to compute the average heart beat rate expressed in beat per minute (BPM), the maximum and minimum heart rate, and to analyze the stress level of the monitored subject.

4.4.3 Stress analysis engine

The heart rate is computed from the RR_i values, sent by the wearable node and expressed in milliseconds. It worth noting that we assume a reliable communication between the wireless chest band and the wearable node. The system is able to detect most of the times when heart-beat packets are dropped due to radio interfence or out-of-range. We decided not to interpolate dropped RR_i messages to avoid further bias while executing the analysis.

We use a 20-point moving average filter over the inter-beat intervals. Maximum and minimum heart rate, however, are computed instantaneously by dividing the current RR_i received from the wireless node by one minute.

The stress level of the subject is refreshed every ten minutes (previous works have shown that this is the minimum collection time to get significant results [67]). Our approach is based only on a time-domain analysis, which is fair enough to evaluate the stress condition as demonstrated in [67].

Specifically, $\overline{RR_j}$ (computed by averaging on 15 heartbeats) proportional to $\overline{HR}$, *SDNN*, *RMSSD*, and *pNN50* are computed as follows:

$$\overline{RR_j} = \frac{1}{15}\sum_{j=1}^{15} RR_j \tag{4.9}$$

$$SDNN = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(RR_j - \overline{RR})^2} \tag{4.10}$$

$$RMSSD = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N-1}(RR_{j+1} - RR_j)^2} \tag{4.11}$$

$$pNN50 = \frac{NN50}{N-1} \times 100 \tag{4.12}$$

where RR_j denotes the value of j^{th} RR interval and N is the total number of successive intervals. *SDNN* is the primary measure used to quantify HRV change, since *SDNN* reflects all the cyclic components responsible for variability in the period of recording. Under negative emotions, the activation of *Autonomic Nervous System* (ANS) is decreased compared to positive emotions; hence, higher SDNN is often an indicator for ANS activation.

The proposed work focuses on determining whether the monitored subject is under emotional stress. It is a *decision problem* that has been solved with

Table 4.2. Stress threshold for HRV parameters.

Feature	Threshold	Unit
HR	>85	1/min
pNN50	<7	%
SDNN	<55	ms
RMSSD	<45	ms

a threshold-based approach. Table 4.2 reports the threshold values extracted from the results found in [67]. The final decision is made on a simple majority vote: if three out of the four features exceed the threshold, the current emotional condition is classified as *stressed*.

4.5 Summary

The main goal of SPINE is to provide BSN developers with support for rapid prototyping of signal-processing applications. In SPINE, sensors and common processing blocks, such as math aggregators and threshold-based alarms, can be configured independently and connected together arbitrarily at run-time based on external controls.

One of the key advantages of SPINE is the ability to satisfy diverse application needs at run-time, avoiding, in most situations, the costly redeployment of the code running on the remote sensing devices.

Such an approach also allows heterogeneous applications to be built atop the same basic software components, enhancing code reusability and, more importantly, removes the need for redeploying the node-side code based on a particular application.

This property is very desirable especially in real world scenarios. For instance, a doctor could use SPINE nodes that are equipped with accelerometers and a suitable coordinator device (e.g. a smartphone), to monitor weekly energy expenditure of a patient. The same nodes could be used later with another patient, for instance, in a rehabilitation scenario, as long as the proper application software is available on the doctor's coordinator device.

In this chapter, the SPINE framework has been showed to support heterogeneous health-care applications without redeployment of the code running on the nodes. The flexibility of SPINE has been demonstrated by describing four different case studies (physical activity detection, step counting, energy expenditure estimation, and emotional stress detection) which all exploit the same sensor node hardware and software. Obvisouly, in the general case, to support different applications, the wearable sensing node(s) must be equipped with all the required physical sensors.

5

Platform-Based Design methodology for BSNs

5.1 Introduction

In the context of WSN, the design usually follows a "*bottom-up*" approach such that these systems are realized choosing the hardware components first, then the communication protocols, and finally implementing ad-hoc applications on top of the underlying infrastructure.

In this thesis, a different approach for the development of BSN applications based on the concept of domain-specific frameworks has been presented. This can be considered a "*top-down*" approach as the proposed application-level framework provides a set of programming abstractions and services without any intrinsic assumptions on the underlying protocol stacks and hardware platforms. However, to obtain the final system, which includes obviously both the hardware (e.g. the physical sensors/transducers) and the software, the designer is not fully guided during the process of requirement and resource optimization (such as system lifetime and cost), and must rely on his skills and expertise in the specific application domain.

A yet different strategy to support rigorous design methodology, reliable system design, and true interoperability between different applications as well as between different implementation platforms, based on a Platform Based Design (PBD) [68] approach, has been recently proposed [69, 70].

PBD is a "*meet-in-the-middle*" design methodology, where system constraints are refined top-down, while implementation characteristics including performances such as delay and power consumption are abstracted bottom-up. The two parts are essential for selecting a good implementation via a design exploration phase that meets the constraints while estimating the performance of the candidate implementations. PBD relies on a clear identification of layers of abstraction, on a modeling strategy that captures uniformly functionality and architecture of the design, and on tools that map two contiguous layers, verify that the selected architectures satisfy constraints, and identify drawbacks and strengths of the design.

In [69, 70], this methodology has been applied to WSNs, and three layers of abstraction and relative platforms are identified: a service platform at the application layer, a protocol platform to describe the protocol stacks, and an implementation platform for the hardware nodes. The proposed methodology has been validated using case studies from WSN-based building monitoring and industrial monitoring applications. However, a specific research on a PBD approach to design BSN systems has never been faced before.

This Chapter proposes a specialization of the methodology presented in [69] focused to the context of BSNs.

5.2 Platform-Based Design

According to the PBD, a design is composed by a sequence of steps that lead the initial high level description all the way down to the implementation. Each step is a refinement process that takes 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 replacing each block of the higher level description, with blocks (or composition of blocks) from a lower level description. Among the possible lower level implementations, the methodology selects one that satisfies the constraints coming from the higher level description, while optimizing according to some cost function. For each layer of abstraction, these design blocks, together with a description of their interfaces and performance, are stored in a library, called *platform*. The higher the initial level of abstraction, the easier is expressing the functionality and constraints as well as catching design errors early, but the more difficult is to reach quickly a high-quality implementation due to the semantic gap between specification and implementation. Differently from classical top-down or bottom-up approaches, in PBD each step is a combination of both approaches where application constraints are refined in a top-down fashion, architecture performance are abstracted in a bottom up fashion, and a "*meet-in-the-middle*" phase decides the final implementation solving a constrained optimization problem. The formalization of the PBD methodology is based on the Agent Algebra [71].

The Agent algebra can be used to describe formally the process of successive refinement in a platform-based design methodology. There, refinement is interpreted as the concretization of a *function* in terms of the elements of a platform. The process of design consists of evaluating the performance of different kinds of instances in the platform by mapping the functionality onto its different elements. The implementation is then chosen on the basis of a cost function. Three distinct domains of agents are used to characterize the process of mapping and performance evaluation. The first two are used to represent the platform and the function, while the third, called the *common semantic domain*, is an intermediate domain that is used to map the function onto a platform instance. A platform, depicted on the right in Figure 5.1, corresponds

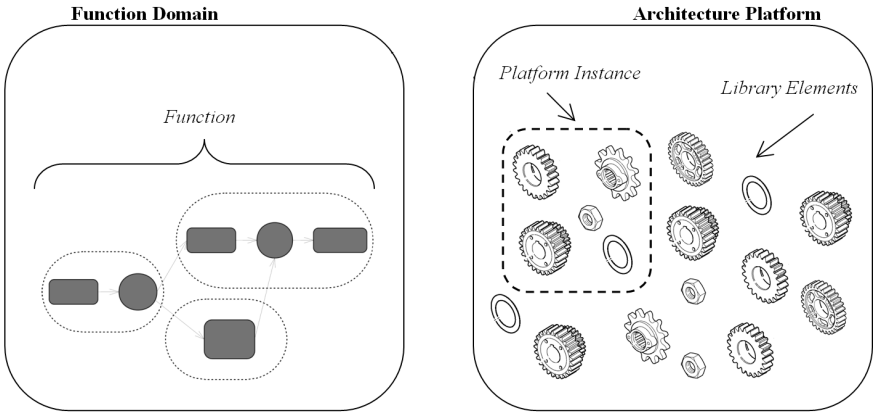


Fig. 5.1. Architecture and function platforms.

to the implementation search space. On the other hand, the function, depicted on the left in Figure 5.1, is represented in an agent algebra called the specification domain. Here, the desired function may be represented denotationally as the collective behavior of a composition of agents, or it may retain its structure in terms of a particular topology of simpler functions. The denotational representation is typically used at the beginning of the platform-based design process, when no information on the structure of the implementation is available. Conversely, after the first mapping, the subsequent refinement steps are started from the mapped platform instance, which is taken as the specification. Thus, a common semantic domain, described in the following, is used as the specification domain. However, contrary to the mapping process that is used to select one particular instance among several, when viewed as a representation of a function, the mapped instance is a specification and is therefore fixed. The function and the platform come together in an intermediate representation called the common semantic domain. This domain plays the role of a common refinement and is used to combine the properties of both the platform and the specification domain that are relevant to the mapping process. The domains are related through conservative approximations.

In particular, the inverse of the conservative approximation is defined at the function to evaluate. The function is therefore mapped onto the common semantic domain as shown in Figure 5.2. This mapping also includes all the refinements of the function that are consistent with performance constraints, which can be interpreted in the semantic domain. If the platform includes programmable elements, the correspondence between the platform and the common semantic domain is typically more complex.

In this case, each platform instance may be used to implement a variety of functions or behaviors. Each of these functions is in turn represented as one agent in the common semantic domain. A platform instance is therefore projected onto the common semantic domain by considering the collection of

agents that can be implemented by the particular instance. This projection, represented by the rays that originate from the platform in Figure 5.2, may or may not have a greatest element. If it does, the greatest element represents the nondeterministic choice of any of the functions that are implementable by the instance.

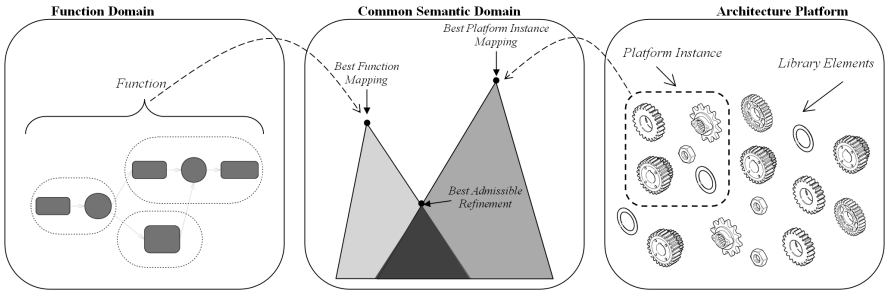


Fig. 5.2. Mapping of function and architecture.

The common semantic domain is partitioned into four different areas. We are interested in the intersection between the refinements of the function and the functions that are implementable by the platform instance. This area is marked "Admissible Refinements" in Figure 5.2. Each of the admissible refinements encodes a particular mapping of the components of the function onto the services offered by the selected platform instance. These can often be seen as the *covering* of the function through the elements of the platform library. Of all these agents, those that are closer to the greatest element are more likely offer the most flexibility in the implementation. Once a suitable implementation has been chosen (possibly by considering different platform instances), the same refinement process is iterated to descend to an even more concrete level of abstraction. The new function is thus the intersection between the behavior of the original function and the structure imposed by the platform. The process continues recursively at increasingly detailed levels of abstraction to reach the final implementation.

5.3 PBD for BSNs

In this section, the different platforms introduced in [69] are specialized for the BSN domain.

Three layers of abstractions are defined, each of them associated to a corresponding platform as follows:

1. The *Sensor Network Service Platform* (SNSP) represents the highest level, and is used by the end user to describe the application;

2. The *Sensor Network Ad-hoc Protocol Platform* (SNAPP) is the middle layer, and is used to describe the different communication protocols;
3. The *Sensor Network Implementation Platform* (SNIP) represents the lowest level, and is used to describe the different hardware nodes.

5.3.1 The Sensor Network Service Platform

The goal of the SNSP is to introduce an abstraction layer for the application description. A properly defined application interface captures all the possible services that can be used by any BSN application and supported by any physical sensor network platform. To perform its functionality, a controller (algorithm) has to be able to read the state of the environment. In a BSN, controllers do so by relying on communication and coordination among a set of distinct elements to sense and control the assisted living.

The role of the SNSP is to provide a logical abstraction for these communication and coordination functions. This approach allows the designer to specify the application while abstracting away the specific details of the communication mechanisms (routing strategies, MAC protocols, physical channel characteristics) thereby making possible for the application designer to focus on the task of developing the application-specific control algorithms. In particular, the SNSP is a collection of data sensing and processing functions that cooperate to provide the following services:

- *query service* (QS), used by controllers to get information from other components;
- *command service* (CS), used by controllers to configure functionalities (e.g. sensing, or processing) of other components;
- *timing/synchronization service* (TSS), used by components to agree on a common time;
- *location service* (LS), used by components to learn their location (e.g. local position on the body);
- *concept repository service* (CRS), which maintains a map of the capabilities of the deployed system and it is used by all the components to maintain a common consistent definition of the concepts that they agreed upon during the network operation.
- *privacy and security service* (PSS), used to protect user confidential information (such as physiological signals), and to guarantee access to such information only to authorized entities.

The CRS is essential for enhancing system reconfigurability and interoperability. The repository includes definitions of relevant global concepts such as the physical quantities that can be acquired (e.g. body movements, heart rate, skin temperature). Additionally, It allows for collecting information about the capabilities of the system (i.e. which sensing and processing services it provides and at which quality and cost). The repository could be dynamically

updated during the network operations, depending on the sensor nodes that might be turned on/off.

Access to the SNSP services is provided to the application through a set of APIs. Following the Agent Algebra formulation, the SNSP can be characterized by defining a set of agents and how they can be composed to create an instance. An instance of this platform is also called *Application*. There are three types of agents:

1. *Services*: these are the previously described macro-instructions used to achieve a specific goal within an algorithm;
2. *Conditional Blocks*: used to relate time- or data-triggered events to specific decision on service requests.
3. *Directional Links*: they abstract the sequentiality of two services or conditional blocks.

Services or conditional statements can be composed only if a directional link is declared between the two. Consequently, at the highest level of abstraction, applications can be described using *flowcharts*.

5.3.2 The Sensor Network Implementation Platform

The Sensor Network Implementation Platform (SNIP) is a library of physical nodes that can be used to support the application. A physical node is a collection of physical resources such as:

- sensing devices (i.e. transducers);
- processing units (e.g. micro-controller or dedicated processing chips);
- external memory (typically for local data storage);
- energy sources (i.e. battery);
- communication devices (i.e. the radio);
- LEDs and/or LCD screen (for visual feedbacks to the user);
- clocks.

In particular, the main physical parameters of a node are:

- list of physical sensors attached to node;
- maximum sampling rate for each attached sensor;
- memory available for the application, and for data storage;
- clock frequency range;
- clock accuracy and stability;
- level of available energy;
- cost of sensing (energy);
- cost of computation (memory, time, energy);
- cost of communication (energy);
- transmission rate, and range.

Examples of physical nodes have been given in Section 2.2.

Using the algebraic approach, the SNIP can be defined as a library whose agents are the hardware nodes, the coordinator devices, and bidirectional links. The hardware nodes and base stations are characterized by their physical resources and possibly by their location on the body. An instances of this platform is called a *Topology*. In a topology, physical components can be connected only using links. A link represents the capability of communication between two physical components. Restrictions on the possibility of linking directly two components reflect the reachability due to their radio interface and distance. For example Shimmer motes [50] and commercial smart-phones can be linked since they both use Bluetooth, while TelosB cannot be directly linked to a smart-phone, but a path between the two can exist only if there is also a third component (such as a bridge device) that is able to support both radio interfaces.

5.3.3 The Sensor Network Ad-hoc Protocol Platform

Choosing the architecture of the SNIP and mapping the functional specification of the system onto it are critical steps in sensor network design. To facilitate this process, the Sensor Network Ad-hoc Protocol Platform (SNAPP) is a proper intermediate level of abstraction. The SNAPP is a library of communication standards and MAC protocols. These are "*parametrized protocols*", meaning that their structure is specified, but their working point is determined by a set of parameters. In general, these parameters are the free parameters of the protocol that can be easily tuned by the application developer. For example, the access probability in a p-persistent CSMA scheme, or the wake up rate of the nodes for the un-beaconed version of the IEEE 802.15.4. The value of these parameters is obtained as the solution of a constrained optimization problem, where the constraints are derived from the latency, error rate, sensing requirements of the application while the cost function is the energy consumption. The energy consumption is estimated based on an abstraction of the physical properties of the candidate hardware platform. Any protocol can be included in the SNAPP as long as the end to end delay distribution and energy consumption performance of the protocol are characterized.

However, differently from the general WSN systems, most of the BSN applications require simple one-hop physical sensor networks (tipically, star-topology; see Section 3.10.1.1). As a consequence, the modeling effort for characterizing the end-to-end (E2E) delay distribution is limited to the analysis of the single hop performance of the protocol. Specifically, parameters such as the number of nodes forming the BSN, the wake up rate, and the distribution of the number of transmission attempts before observing a successful packet exchange must be evaluated.

The mathematical framework allows for capturing the requirements of the design functionality and performance as a constrained optimization problem.

The solution to this problem provides the parameters to derive the final protocol implementation. Once the trade-off equations are derived and solved as an optimization problem, all the protocol parameters are automatically synthesized. The use of parameterized protocols allows to effectively restrict the large design space to a few parameters. In addition, since the protocols are developed with a specific mathematical model in mind, it is ease to evaluate the effects of changing these parameters on the overall network performance. This predictive ability prevents the need for extensive simulation and allows for the ease of comparison with other protocols.

5.4 A case study: Activity Recognition based on *Template Matching*

Action recognition is a classification problem with the goal of detecting transitional actions such as "Sit to Stand", "Walking", and "Kneeling". It can be used for a variety of applications such as activity monitoring, gait analysis, and diagnosis of many movement disorders such as Parkinson's and Alzheimer's diseases.

Unfortunately, designing power-aware signal processing algorithms for action recognition is challenging as special care needs to be taken to maintain acceptable classification accuracy while minimizing power consumption as well as improving wearability of the system. As a consequence, trade-offs must be made between power consumption and classification accuracy while designing the system.

Reducing the amount of active nodes is a common approach for power optimization and wearability enhancement in BSNs. Recognizing more complex or multitude of actions requires more sensor nodes. However, in most cases, only a subset of sensors is needed to recognize individual actions. Keeping those sensors operational whose values are not considered by the system, is wasteful. Thus, it is beneficial to determine the minimal set of nodes that can identify a given full action set.

Differently from most of related work, our approach to solve this classification problem combines two performance criteria: accuracy and power consumption. Furthermore, while most of the previous works in sensor node selection minimize the number of active nodes, we examine individual sensors that are embedded within each sensor node. Specifically, our classification approach is based on *Template Matching*. Figure 5.3 shows a high level block diagram of the classification based on similarity score provided by the template matching function. Template matching is a technique most used in image processing to find portions of an image, called *template*, which are of interest. The goal is to find the portions of the image that are most similar to the template [72]. Instead of using images, we use the signals coming from the inertial sensors and compare such signals with a previously calculated template. There are different ways to calculate the similarity of an incoming signal

with a predefined template of interest, including Sum of Squared Differences, Euclidean Distance, and Normalized Cross Correlation (NCC).

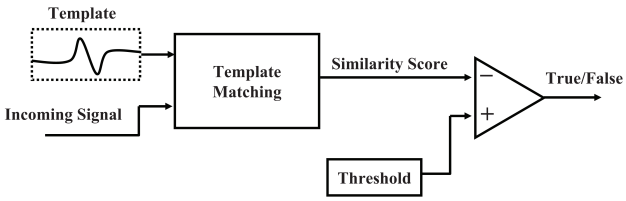


Fig. 5.3. Template matching for classification.

In our classification model, results generated by individual sensor nodes are combined to provide a final classification decision. This classifier combination is a classic example of ensemble learning. In statistics and machine learning, the ensemble learning refers to the method of combining multiple learned models a predictive model that is more accurate that the constituent models. The basilar assumption is that the resulting classifier is at least as accurate as the most accurate weak classifier. The most well-known techniques for ensemble learning include Weighted Majority, Boosting, Bagging and Stacked Generalization [73]. We use a specific type of boosting adapted from the well-known AdaBoost algorithm.

Classical AdaBoost learns from a set of weak classifiers and boost classification performance by allocating weights to the individual local classifiers. The weights specify contribution of each sensor to the classification task. There are two main drawbacks with this approach:

1. AdaBoost examines all classifiers even if they provide less informative data for classification. Contribution of different sensors to action recognition varies depending on the placement of the sensor, type of inertial information, and actions of interest. Practically, body-worn sensors can produce redundant or correlated information when a movement occurs. For instance, signals from nodes placed on the "upper arm" and "lower fore arm" correlate during an upper body movement. Also, data provided by a node placed on the "leg" may not contribute to upper body movements such as "Sit to Stand". Consequently, a more intelligent learning algorithm is required to select only those sensors that contribute best to the classification accuracy.
2. AdaBoost does not take into account the power requirements of the individual weak classifiers while learning from weak classifiers. In fact, the weights assigned by the AdaBoost account only for the accuracy of the classifiers. However, power consumption of the classifiers varies depending on the type of sensors, computing complexity, and data communication requirements. For example, gyroscopes generally consume much more power than accelerometers. Therefore, the optimal subset of the classifiers

is selected by making appropriate design trade-off between accuracy and power consumption.

5.4.1 Problem Formulation

Our system aims to detect a target action of interest, $\widehat{\alpha}$, associated with a particular template. Therefore, the system consists of several binary weak classifiers each contributing to detection of the target action using a template matching approach. To build a classification framework, we assume that there are a total of m non-target actions, $A = \{a_1, a_2, \cdots, a_m\}$, that may occurs during the operation of the system.

Definition 1 (Sensor Node). A sensor node, s_i, is a physical wearable node that has limited processing power and storage and can communicate within certain range, and is composed of l sensors that capture inertial data from human movements. Each sensor node is placed on a specific location on the body.

Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set of n sensor nodes that are used to distinguish between the target action and non-target actions.

Definition 2 (Weak Classifier). Each sensor node, s_i, consists of l weak classifiers, $C_i = \{C_{i1}, C_{i2}, \cdots, C_{il}\}$. Each classifier C_{ik} is associated with one of the sensors available on s_i, and is a binary classifier that operates based on template matching and makes a classification decision using the similarity score obtained from NCC. The classifier determines whether or not an incoming signal is classified as the target action.

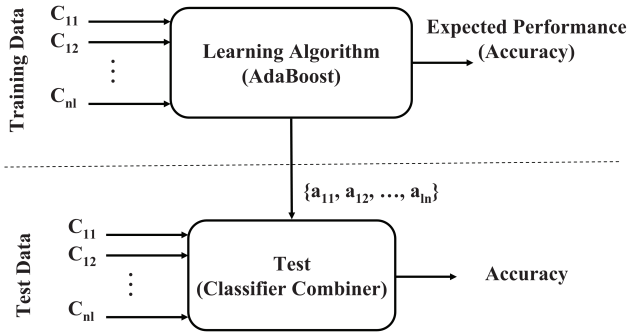


Fig. 5.4. Learning algorithm and classifier combiner during training and test.

Given the n nodes and l sensors per node, the system has a total of $T = n \times l$ weak classifiers. Figure 5.4 shows how learning parameters (e.g. weights $\alpha_{11}, \cdots, \alpha_{nl}$) are generated during training. As shown in the figure, the learning algorithm can also provide an estimate of the expected accuracy of

the entire classification, α. The set of sensors/classifiers that are used for learning would essentially determine the accuracy of the system. To calculate power consumption of the set of active classifiers, we consider two sources of power consumption, namely computation and communication costs. We assume that each classifier is associated with a specific sensor (e.g. x-accelerometer, y-gyroscope) and therefore has a fixed computation cost depending on whether or not it is activated.

Definition 3 (Computation Cost). For each classifier C_{ij}, we define w_{ij} as the computation cost associated with power consumption of the corresponding sensor. This value is a priori known and has a non-zero value for active classifiers while it is zero for non-active sensors. Thus, the total computation costs is given by:

$$P_{comp} = \sum_{i=1}^{n} \sum_{j=1}^{l} x_{ij} w_{ij} \tag{5.1}$$

where x_{ij} denotes in classifier C_{ij} is active.

$$x_{ij} = \begin{cases} 1, & \text{if classifier } C_{ij} \text{ is active} \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

Definition 4 (Communication Cost). For a set of weak classifiers used for learning, the communication cost is given by:

$$P_{comm} = \sum_{i=1}^{n} f\left(\sum_{j=1}^{l} x_{ij} b_{ij} \right) \tag{5.3}$$

where $f(.)$ denotes the communication cost due to transmission of certain amount of data, and x_{ij} denotes if classifier C_{ij} is activated, and b_{ij} represents the amount of data that is generated by classifier C_{ij} and needs to be transmitted to the basestation.

We note that the communication costs are calculated for each sensor node rather than individual sensors/classifiers. This is mainly due to combining results of all active classifiers at each node prior to transmission to the basestation. In other words, energy cost of communications is calculated collectively for all active sensors at each node. The power consumption of the system due to activation of a set of weak classifiers used for learning is then given by:

$$Z = P_{comp} + P_{comm} \tag{5.4}$$

Problem 1. Given a set of $T = \cup C_i = \{C_{11}, C_{12}, \cdots, C_{nl}\}$ classifiers, and also data units b_{ij} and computation cost w_{ij} for each classifier C_{ij}, the problem of Minimum Cost Classifier Selection (MCCS) is to find a subset of C_{ij} with

minimum total cost while a lower bound of $\alpha \geq F$ on the overall accuracy is met. Therefore, the optimization problem can be written as follows:

$$\text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{l} x_{ij} w_{ij} + \sum_{i=1}^{n} f \left(\sum_{j=1}^{l} x_{ij} b_{ij} \right) \tag{5.5}$$

subject to:

$$\alpha \geq F \tag{5.6}$$

5.4.2 Applying the PDB methodology

To drive the design flow of the case study system, the SNSP, SNIP, and SNAPP platforms defined in Section 5.3 sre used. The synthesis is composed by a sequence of successive refinements that starts with a functional description of the application and delivers at the other end a network of programmed wireless sensor nodes and a coordinator device.

However, to provide a higher abstraction layer for the functional description of the system, a further platform, called the Virtual Connectivity Platform (VCP) is used. A VCP $\mathcal{Q}^{vcf}$ is composed of three types of library elements: the set of Virtual Sensors $\mathcal{S}_v$, Virtual Controllers $\mathcal{C}_v$, and bidirectional links $\mathcal{L}$. A virtual sensor $s_v \in \mathcal{S}_v$ is an abstraction of a sensing area and will be later on refined in a set of physical sensor nodes. Similarly, a virtual controller $c_v \in \mathcal{C}_v$ is an abstraction of a computation capability and in general can be mapped on physical sensor nodes, on the coordinator device, or partially on both of them. Virtual components and links can be composed to form more complex elements. It is not possible to directly connect virtual components, but links must be used.

During the first step of the design flow, the classification algorithm of the case study system can be easily described as an instance of VCP. Specifically, it is possible to use very intuitively the concept of virtual sensor defined in Section 3.11 as a formal model for both the Virtual Sensors and Virtual Controllers in VCP. Alternatively, the task-oriented model supported by SPINE2 is another natural way to formalize the functional behavior of the system. Of course, there are many possible refinements which represent the same functional model. However, among these possible refinements it is chosen the "highest", which is the one with the minimum number of virtual components and looser sensing and communication requirements. This instance is called r_g and this is the starting point for the rest of the synthesis flow.

The next step is to refine the requirement graph r_g into a networked topology nt. The goal is to substitute the virtual components with an adequate set of physical components. To do this, the Sensor Network Implementation Platform is used. Selecting an agent in $\mathcal{Q}^{snip}$ and mapping it to $\mathcal{Q}^{nt}$, means selecting the hardware platform for the sensor node (taking into account the built-in physical sensors and the ones that may be attached to it) and a lower

bound on the density of nodes. The mapping gives a set of agents that may or may not intersect the set of agents obtained by mapping r_g to $\mathcal{Q}^{nt}$. This intersection represents all the topologies with a sufficient number of interoperable nodes (equipped with proper physical sensors) that are good enough to satisfy the sensing requirements. The set of solutions that are unfeasible, e.g. due to budget and wearability constraints (the overall solution has too many nodes) must then by removed. Among the remaining solutions, choosing the right one to propagate down in the synthesis process is not trivial. Given a number of nodes, the solution with the loosest sensing and communication requirements is chosen. However it is difficult to understand at this level what is a good number of nodes. On one hand, using more nodes may improve the classification accuracy and the system lifetime as nodes can duty cycle e.g. according to the set of actions that have to be recognized. On the other hand, more nodes involve a higher cost of the solution and lead to a more invasive system. However, the energy consumption cannot be estimated until the communication protocol is decided and this happens at the next step of the design flow. Consequently, we start with a solution that is relatively high in the space of the possible solutions (i.e. with a low number of nodes), and after the communication protocol is mapped, evaluate if the energy consumption per node is satisfactory for a good lifetime of the network. If that is not the case, go back and select another possible solution with more nodes. This step consists in mapping the classification algorithm into the hardware platform of the controller. This can be performed with a mapping tool like Metropolis [74, 75], which is a design environment that was developed to support Platform Based Design. The advantage of using Metropolis is that it supports any type of model of computation for the functional description.

The last step is concerned with associating a communication protocol to the physical components such that the communication requirements are satisfied and the energy consumption minimized. To drive this step the Sensor Network Ad-Hoc Protocol Platform (SNAPP) $\mathcal{Q}^{snapp}$ is used. The library elements of the SNAPP are MAC protocols. The common semantic domain in this step is represented by the instantiated network domain $\mathcal{Q}^{in}$. An instantiated network is an operational BSN, i.e. a network of physical nodes with a communication protocol. By mapping the selected networked topology onto this common domain all the possible instantiated networks that satisfy the given requirements on accuracy are obtained. By mapping a SNAPP instance all the possible instantiated networks that use the selected protocol with all the feasible combinations of the free parameters are obtained. The intersection between the two mappings gives all the possible instantiated networks that use the selected protocol and satisfy the given communication constraints. Among these solutions, the one that minimizes the energy consumption is selected. At this point, it is possible to evaluate if the synthesized solution can comply with the lifetime requirements of the network. If that is the case, the process ends, otherwise it is necessary to get back to the networked topology domain and select an instance with more nodes. This final refinement is obtained as

the solution of a constrained optimization problem, where the constraint is the classification accuracy requirement while the cost function to be minimized is the energy consumption that is estimated based on an abstraction of the physical properties of the candidate hardware platform. This step can be effectively guided using a specific mapping framework like Rialto [69].

This step maps the communication protocol on the physical nodes. Since the communication protocols of the SNAPP are already described in a distributed fashion, the parametrized code for each node can be easily developed using the software interface of the nodes.

5.4.3 Summary

This chapter introduced a specialization of a PBD methodology for system level design of BSN applications. First, the PBD approach has been briefly described. Then, a PBD methology, previously proposed for the design of WSN systems, has been specialized for the more specific BSN domain. Finally, the methodology is applied to a case study related to the human activity recognition problem solved with a technique inspired to a template matching approach from the image processing domain.

6

Conclusions, Publications and Future Work

6.1 Conclusions

Wireless *Body Sensor Networks* (BSNs) allow for continuous and non-invasive measurement of human body movements and physiological parameters by means of tiny wireless physiological sensor devices applied to the skin and in garments. BSNs possess enormous potential for changing people's daily lives as the can significantly enhance many human-centered domains not only strictly related to medical applications (such as early detection and prevention of diseases, and medical assistance at home) but also to sport, fitness, and many other scenarios. However, although several BSN-based research prototypes have been proposed so far, none of them have reached the market yet. One of the biggest driving factors for this delay is due to their software design and implementation approaches. This thesis provided innovative and effective solutions to this issue, particularly focused on programming abstractions, techniques, and methodologies for improving design and prototyping of BSN systems.

During the development of this thesis, several contributions have been provided to the BSN research area.

The first important contribution is the definition of a domain-specific approach for programming BSN systems, which resulted in a domain-specific programming framework named *SPINE* (Signal Processing in Node Environment). One of the main achievements of 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 without off-line reprogramming before switching from an application to another. Furthermore, thanks to its modular component-based design approach, SPINE enables high degree of heterogeneity, and a wide variety of hardware platforms, sensors, programming languages, and operating systems are supported. This allows for a very flexible and usable framework in different BSN application scenar-

ios, where, due to specific requirements, only certain platforms or operating systems might be used.

Inside the thesis, Section 2 has been devoted to the review of the current BSN state-of-the-art, with a specific emphasis on development tools and middlewares for programming BSN applications, along with a systematic identification of the fundamental requirement and properties that an effective and efficient BSN programming framework should satisfy. To the best of our knowledge, such a review work was missing before, and, therefore, it can be considered an important contribution of this thesis as well.

To provide quantitative validation and performance evaluation of the proposed framework, a number of BSN research prototypes have been implemented atop SPINE. One of the main outcomes of having used SPINE is a significant reduction of the development efforts: 100% at the node side, and 80% at the coordinator side. This shows that a notable improvement can be obtained by adopting SPINE for the development of BSN applications.

Some considerations must be devoted to the proposed case studies as they improved the current state-of-the-art. The physical activity monitoring system reaches an overall average recognition accuracy of 97% using only two wearable motion sensor nodes, a fewer number than the most relevant works. The step-counter application relies on a novel algorithm which, to the best of our knowledge, is the only one able to correctly recognize the steps taken during walking from healthy, elderly, and people affected by walking disabilities, limiting the average step detection error to only 12%. Our physical energy expenditure system is able to estimate the calories burnt during daily activities in real-time without assuming fixed orientation of the worn motion sensor, which represented an important improvement to the state-of-the-art. Finally, our emotional stress detection relies on a wireless sensor system, and a monitoring application that, by means of time-domain heart-rate analysis, provides a stress index using only ten minutes of observations.

Additional contributions presented in this thesis are the result of enhancements and variants to the original proposed framework SPINE. They include a *task-oriented* framework re-design of SPINE, an enhancement for supporting *Collaborative BSNs*, a *multi-agent* model for BSN programming, and a programming paradigm based on the concept of *Virtual Sensors*. Particularly relevant is the task-oriented re-design of SPINE through which the significant result of *platform-independence* is achieved for most of the framework components, and the multi-agent model as we showed that, at least when implemented on more powerful sensor nodes, it allows for even more flexibility, also thanks to the code migration which can be more naturally enabled by means of the *mobile agent* concept.

Finally, a further contribution of this thesis is a specialization of the *Platform-Based Design* (PBD) methodology for the BSN domain. PBD has shown to be very effective in traditional embedded system design, both at academic and industrial level. However, although an application of the PBD to the WSN domain has been proposed in the past, delivering case studies for

building and industrial monitoring, this is the first time that PBD is shown to properly address also the design of BSN systems.

6.2 Publications Related with this Thesis

The research work related to this thesis has resulted in 21 publications, including:

- 6 journal articles (4 with ISI impact factor);
- 12 conference papers;
- 2 book chapters;
- 1 conference posters.

In the following, the publications are organized according to the section of the thesis where the contents have been discussed, and a brief description of each publication is provided.

6.2.1 SPINE

- **SPINE: A domain-specific framework for rapid prototyping of WBSN applications** [44]:

  F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi. SPINE: A domain-specific framework for rapid prototyping of WBSN applications. *Software: Practice & Experience*, 41(3):237-265, March 2011.

  This paper is a significant extension of [76]. It presents in detail the core SPINE framework, and describes its unique features through the development of a case study which consists of a BSN system for monitoring human physical activities in real-time.

- **SPINE-based application development on heterogeneous Wireless Body Sensor Networks** [77]:

  G. Fortino, S. Galzarano, R. Giannantonio, R. Gravina, and A. Guerrieri. SPINE-based application development on heterogeneous Wireless Body Sensor Networks. *International Journal of Computing*, 9(1):80-89, 2010.

  This paper proposes an approach based on the SPINE frameworks (SPINE-1.x and SPINE2) for the programming of signal processing applications on heterogeneous wireless sensor platforms. In particular, it presents two integrable approaches, based on the proposed frameworks, that allow for the development of applications for BSNs constituted by heterogeneous sensor nodes.

- **DexterNet: An open platform for heterogeneous Body Sensor Networks and its applications** [78] :

  P. Kuryloski, A. Giani, R. Giannantonio, K. Gilani, R. Gravina, V. Seppä, E. Seto, V. Shia, C.Wang, P. Yan, A.Y. Yang, J. Hyttinen, S. Sastry, S.Wicker, and R. Bajcsy. DexterNet: An

open platform for heterogeneous Body Sensor Networks and its applications. In *Proceedings of the International Conference on Body Sensor Networks*, BSN 2009, pages 92-97. IEEE Computer Society, June 2009.
The paper presents an open-source platform for BSNs called *DexterNet*. The system supports real-time, persistent human monitoring in both indoor and outdoor environments. The platform utilizes a three-layer architecture to control heterogeneous body sensors.

- **Performance analysis of an activity monitoring system using the SPINE framework** [79]:
R. Giannantonio, R. Gravina, P. Kuryloski, V. Seppä, F. Bellifemine, J. Hyttinen, and M. Sgroi. Performance analysis of an activity monitoring system using the SPINE framework. In *Proceedings of the 3rd International Conference on Pervasive Computing Technologies for Healthcare*, Pervasive Health 2009, pages 1-8. IEEE Press, April 2009.
This paper describes important implementation parameters of the SPINE framework, such as processing time, memory, bandwidth usage and power consumption that are most relevant for application developers to set tunable parameters and analyze system performance. It also presents performance and resource usage of a SPINE-based posture recognition system for elderly health monitoring.

- **SPINE: Signal Processing In Node Environment** [80]:
R. Giannantonio, F. Bellifemine, R. Gravina, A. Guerrieri, G. Fortino, and M. Sgroi. SPINE: Signal Processing In Node Environment. In *Proceedings of the 1st European TinyOS Technology Exchange*, ETTX 2009, February 2009.
This poster provides a quick overview to some relevant aspects of the TinyOS implementation of the SPINE framework.

6.2.2 SPINE enhancements and variants

- **Collaborative Body Sensor Networks** [51]:
A. Augimeri, G. Fortino, S. Galzarano, and R. Gravina. Collaborative Body Sensor Networks. In *Proceedings of the IEEE International Conference on Systems, Man and Cybertnetics*, SMC 2011. IEEE Press, October 2011.
This paper proposes reference architectures and a SPINE-based middleware for Collaborative Body Sensor Networks (CBSNs) that can enable new smart wearable systems in the context of physical pervasive computing environments. It also presents a collaborative emotion detection system, integrating heart rate sensing with handshake detection, developed through C-SPINE, and experimentally analyzed.

- **An agent-based signal processing in-node environment for real-time human activity monitoring based on Wireless Body Sensor Networks** [42]:
F. Aiello, F. Bellifemine, G. Fortino, S. Galzarano, and R. Gravina. An agent-based signal

processing in-node environment for real-time human activity monitoring based on Wireless Body Sensor Networks. *Journal of Engineering Applications of Artificial Intelligence*, 24(7):1147-1161, 2011.

This paper proposes an application of the MAPS framework for the development of a real-time BSN-based system for human activity monitoring. The agent-oriented programming abstractions provided by MAPS allow effective and rapid prototyping of the sensor-side software. The coordinator relies on a JADE-based enhancement of the SPINE coordinator, while the sensor nodes run the standard MAPS agents.

- **Programming signal processing applications on heterogeneous wireless sensor platforms** [81]:

  L. Buondonno, G. Fortino, S. Galzarano, R. Giannantonio, A. Giordano, R. Gravina, and A. Guerrieri. Programming signal processing applications on heterogeneous wireless sensor platforms. In *Proceedings of the 5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IDAACS 2009, pages 682-687. IEEE Press, September 2009.

  This paper presents SPINE2, a framework for the programming of signal processing applications on heterogeneous wireless sensor platforms. The approach is exemplified through a human activity recognition system based on a BSN composed of two types of sensor nodes, heterogeneous with respect to base software and hardware.

- **From modeling to implementation of Virtual Sensors in Body Sensor Networks** [56]:

  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 Journal*, doi:10.1109/JSEN.2011.2121059, 2011.

  This paper presents a multi-layer task model based on the concept of Virtual Sensors to improve architecture modularity, and design reusability. Virtual Sensors are abstractions of components of BSN systems that include sensor sampling and processing tasks and provide data upon external requests. The proposed model is applied in the context of gait analysis through wearable sensors.

- **Implementation of Virtual Sensors in Body Sensor Networks with the SPINE framework** [82]:

  N. Raveendranathan, V. Loseu, E. Guenterberg, R. Giannantonio, R. Gravina, M. Sgroi, and R. Jafari. Implementation of Virtual Sensors in Body Sensor Networks with the SPINE framework. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems*, SIES 2009, pages 124-127. IEEE Press, July 2009.

  This paper presents an extension of the SPINE Framework based on the concept of Virtual Sensors (VS) which includes a new buffer management scheme that facilitates the VS implementation.

6.2.3 MAPS and agent-based WSN programming frameworks

- **An analysis of Java-based mobile agent platforms for Wireless Sensor Networks** [83]:

  F. Aiello, G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. An analysis of Java-based mobile agent platforms for Wireless Sensor Networks. *Multi-Agent and GRID Systems*, to appear:1-30, 2011.

  This paper proposes an in-depth analysis of the only two available Java-based mobile agent platforms for WSNs: Mobile Agent Platform for SunSPOT (MAPS) and Agent Factory Micro Edition (AFME). In particular, the architecture, programming model and basic performance of MAPS and AFME are described and compared. Moreover, a simple yet effective case study concerning a mobile agent-based monitoring system for remote sensing and aggregation is proposed.

- **A Java-based agent platform for programming Wireless Sensor Networks** [53]:

  F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri. A Java-based agent platform for programming Wireless Sensor Networks. *The Computer Journal*, 54(3):439-454, 2011.

  This paper presents the design, implementation, and experimentation of MAPS (Mobile Agent Platform for SunSPOT), an innovative Java-based framework for WSNs based on SunSPOT technology which enables agent-oriented programming of WSN applications. Agent programming with MAPS is presented through both a simple example related to mobile agent-based monitoring of a sensor node and a more complex case study for real-time human activity recognition based on BSNs. Moreover, a performance evaluation of MAPS carried out by computing micro-benchmarks, related to agent communication, creation and migration, is illustrated.

- **Agent-based development of Wireless Sensor Network applications** [84]:

  G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. Agent-based development of Wireless Sensor Network applications. In *Proceedings of the 12th Workshop on Objects and Agents*, WOA 2011. CEUR Workshop Proceedings, July 2011.

  This paper promotes the use of the agent paradigm for the development of WSN applications. It provides motivations about synergies between agents and WSNs, and a brief overview about agent technology for WSNs. Requirements, and guidelines for the design of full-fledged agent-oriented methodologies for programming WSN applications are also provided.

- **Signal processing in-node frameworks for Wireless Body Sensor Networks: from low-level to high-level approaches** [85]:

  F. Aiello, G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. *Wireless Body Area Networks: Technology, Implementation and Applications*, chapter 5 - Signal processing in-node frameworks for Wireless Body Sensor Networks: from low-level to high-level approaches,

pages 1-23. Pan Stanford publishing, 2011.
This book chapter proposes a high-level approach based on the agent-oriented programming model to flexibly design and efficiently implement signal processing in-node environments supporting WBAN applications. The approach is exemplified through a case study concerning a real-time human activity monitoring system which is developed through two different agent-based frameworks: MAPS and AFME. A comparison of the effectiveness and efficiency of the developed systems is finally presented.

- **An agent-based signal processing in-node environment for real-time human activity monitoring based on Wireless Body Sensor Networks** [86]:
F. Aiello, F. Bellifemine, G. Fortino, R. Gravina, and A. Guerrieri. An agent-based signal processing in-node environment for real-time human activity monitoring based on Wireless Body Sensor Networks. In *Proceedings of the 1st International Workshop on Infrastructures and Tools for Multiagent Systems, jointly held with the 9th International Conference on Autonomous Agents and Multi-Agent Systems*, ITMAS 2010, May 2010.
This paper proposes an application of MAPS for the development of a real-time WBSN-based system for human activity monitoring. The experimentation phase of the prototype is also described, along with a performance evaluation analysis.

- **MAPS: a mobile agent platform for Java Sun Spots** [87]:
F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri. MAPS: a mobile agent platform for Java Sun Spots. In *Proceedings of the 3rd Workshop on Agent Technology for Sensor Networks, jointly held with the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, ATSN 2009, May 2009.
This paper emphasizes the importance of the mobile agents approach in the WSN domain. Due to their intrinsic characteristics mobile agents may provide more benefits in the context of WSNs than in conventional distributed environments. The discussion is supported through the description, analysis, and evaluation of a case study application of the MAPS framework.

6.2.4 BSN Applications

- **Continuous, real-time monitoring of assisted livings through Wireless Body Sensor Networks** [88]:
D.L. Carni', G. Fortino, D. Grimaldi, R. Gravina, A. Guerrieri, and F. Lamonaca. Continuous, real-time monitoring of assisted livings through Wireless Body Sensor Networks. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IDAACS 2011. IEEE Press, September 2011.
This paper proposes the BSNs as an enabling technology for a rich variety of application domains, from e-Health to e-Factory. The paper describes

reference network architectures, effective programming frameworks and novel applications in important application domains for BSNs.

- **Enabling multiple BSN applications using the SPINE framework** [21]:

  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 *Proceedings of the International Conference on Body Sensor Networks*, BSN 2010, pages 228-233. IEEE Computer Society, June 2010.

  This paper describes a number of practical case studies of the SPINE framework, including gait analysis, physical rehabilitation support system, human activity recognition, and physical energy expenditure. The paper, therefore, emphasizes how SPINE is an effective support to the development of heterogeneous health-care applications based on reusable software subsystems. It also presents a SPINE sensor-node emulator that supports the first phase of the signal processing flow design, when the actual hardware devices may not be available.

- **SPINE-HRV: a BSN-based toolkit for heart rate variability analysis in the time-domain** [64]:

  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. *Wearable and Autonomous Biomedical Devices and Systems: New issues and Characterization - Lecture Notes on Electrical Engineering*, 75:369-389, 2010.

  This book chapter is a significant extension of [89]. It presents a toolkit based on body sensor networks (BSN) for the time-domain HRV analysis, named SPINE-HRV. The SPINE-HRV is composed of a wearable heart activity monitoring system to continuously acquire the RR-intervals, and a processing application developed using the SPINE framework. The chapter describes an application of SPINE-HRV for stress detection of people during activities in their everyday life. Experimentations carried out by monitoring subjects in specific activities have shown the effectiveness of SPINE-HRV in detecting stress.

- **Time-domain heart rate variability analysis with the SPINE-HRV** [89]:

  A. Andreoli, R. Gravina, R. Giannantonio, P. Pierleoni, and G. Fortino. Time-domain heart rate variability analysis with the SPINE-HRV. In *Proceedings of the 1st International Workshop on SigProcessing (Light-weight Signal Processing for Computationally Intensive BSN Applications)*, PETRA 2010. ACM Press, June 2010.

  This paper presents SPINE-HRV, a toolkit for time-domain Heart Rate Variability (HRV) analysis. The toolkit is composed of a wearable Heart Activity Monitoring System to acquire the R-peak to R-peak intervals, and a processing application developed using the SPINE framework.

- **Opportunistic strategies for lightweight signal processing for BSN** [90]:

  E. Seto, M. Eladio, A.Y. Yang, P. Yan, R. Gravina, I. Lin, C. Wang, M. Roy, V. Shia, and R. Bajcsy. Opportunistic strategies for lightweight signal processing for BSN. *In Proceedings of the 1st International Workshop on SigProcessing (Light-weight Signal Processing for Computationally Intensive BSN Applications)*, PETRA 2010. ACM Press, June 2010.

  This paper presents a mobile platform for BSNs based on a smartphone for lightweight signal processing of sensor mote data. The platform allows for local processing of data at both the sensor mote and smartphone levels, reducing the overhead of data transmission to remote services. It discusses how the smartphone platform not only provides the ability for wearable signal processing, but it allows for opportunistic sensing strategies, in which many of the onboard sensors and capabilities of modern smartphones may be collected and fused with body sensor data to provide environmental and social context.

6.3 Future Work

Some of the research directions related with this thesis deserving further efforts are still being explored. An interesting research activity is currently devoted to enhance C-SPINE with proximity algorithms and mechanisms which take into account users' physical activities. Another relevant on-going work is focused on re-investing some of the research contribution presented in this thesis (especially related to the proposed BSN case studies) into industrial patents.

On the basis of the achieved results and on the on-going research activities, a number of new research directions have been envisaged.

A promising future work is related with the integration of the proposed framework with a cloud-based server-side system (e.g. for control and long-term remote analysis). Furthermore, as in the near future personal data generated by the BSN will be surely connected to users' social networks, in ways that we can't necessarily anticipate now, it would be interesting to explore new policy models for BSN-data privacy and publicness.

Another interesting research direction is the application of the *Autonomic Computing* in the context of BSNs. In particular, it should be analyzed the convenience to extend the proposed framework with an autonomic plane, a way for separating out the provision of *self-*$*$ properties from the BSN application logic. The Autonomic Computing is a paradigm born as a response to the increasingly complexity of managing computing systems. It faces the problem by introducing a series of *self-*$*$ properties (self-configuration, self-healing, self-optimization, and self-protection) into complex systems, through which such systems can be capable of performing several self-management actions without any human intervention.

Finally, the proposed PBD methodology surely deserves additional research efforts, e.g. devoted at providing a different Service Platform, based on the concept of *software agents* rather than on *virtual sensors*.

A

MAPS

MAPS [53, 87, 91] is an innovative Java-based framework purposely developed on SunSPOT technology [19] for enabling agent-oriented programming of WSN applications.

In this Appendix, requirements, architecture (at system and agent level), and programming model of the MAPS Framework are described.

A.1 Requirements

The MAPS framework has been appositely defined for resource-constrained sensor nodes; in particular its requirements are the following:

- *Lightweight agent server architecture.* The agent server architecture must be lightweight, which implies the avoidance of heavy concurrency models and, therefore, the exploitation of cooperative concurrency to run agents.
- *Lightweight agent architecture.* The agent architecture must also be lightweight so that agents can be efficiently executed and migrated.
- *Minimal core services.* The main core services must be: agent migration, sensing capability access, agent naming, agent communication and timing. The agent migration service allows an agent to be moved from one sensor node to another by retaining the code, data and execution state. The sensing capability access service allows agents to access the sensing capabilities of the sensor node and, more generally, its resources (actuators, input signalers, flash memory). The agent naming service provides a region-based namespace for agent identifiers and agent locations. The agent communication service allows local and remote one-hop/multi-hop message-based communications among agents. The timing service allows agents to set timers for timing their actions.
- *Plug-in-based architecture extensions.* Any other service must be defined in terms of one or more dynamically installable components (or plug-ins) implemented as single mobile agents or cooperating mobile agents.

- *Layer-oriented mobile agents.* Mobile agents may be natively characterized on the basis of the layer to which they belong: application, middleware and network layer. They should also be able to locally interact to enable cross-layering.

A.2 Agent server architecture

The designed sensor node architecture is shown in Figure A.1.

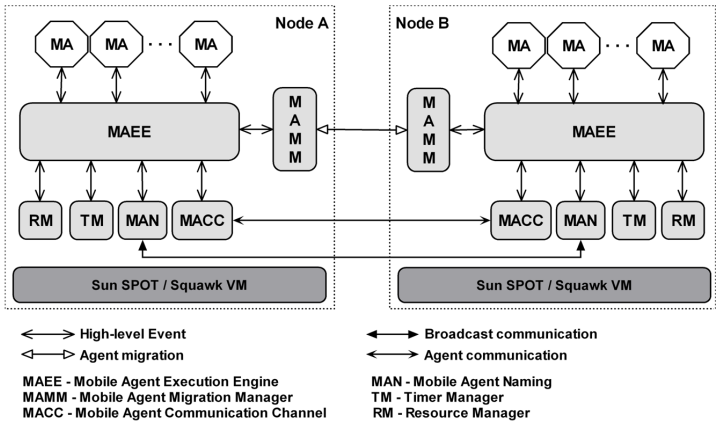


Fig. A.1. The architecture of MAPS.

The architecture is based on components that interact through events.

The choice to design the architecture according to a component- and event-based approach is motivated by the effectiveness that such a kind of architecture has demonstrated for sensor node programming. In fact, the TinyOS operating system [11], the *de facto* standard for wireless sensor node platforms, relies on this kind of architecture. In particular, the main components are the following:

1. *Mobile agent (MA).* The MAs are computing components which are differentiated on the basis of the layer (application, middleware and network) at which they perform tasks. Application layer MAs incorporate application-level logic performing sensor monitoring, actuator control, data filtering/aggregation, high-level event detection, application-level protocols etc. Middleware layer MAs perform middleware-level tasks such as distributed data fusion, discovery protocols for agents, data and sensors, scope management etc. Network layer MAs mainly implement transport (e.g. data dissemination) and network (e.g. multi-hop routing) protocols. Agents at different layers can locally interact to implement cross-layering.

2. *Mobile agent execution engine (MAEE).* The MAEE is the component that supports the execution of agents by means of an event-based scheduler enabling cooperative concurrency. The MAEE handles each event emitted by or to be delivered at MAs through decoupling event queues. The MAEE interacts with the other core components to fulfill service requests (message transmission, sensor reading, timer setting etc.) issued by the MAs.
3. *Mobile agent migration manager (MAMM).* The MAMM component supports the migration of agents from one sensor node to another. In particular, the MAMM is able to: (i) serialize an MA into a message and send it to the target sensor node and (ii) receive a message containing a serialized MA, de-serialize and activate it. The agent serialization format includes the code, data and execution state.
4. *Mobile agent communication channel (MACC).* The MACC component enables inter-agent communications based on asynchronous messages. Messages can be unicast, multicast or broadcast.
5. *Mobile agent naming (MAN).* The MAN component provides agent naming based on proxies and regions [92] to support the MAMM and MACC components in their operations. The MAN also manages the (dynamic) list of the neighbor sensor nodes.
6. *Timer manager (TM).* The TM component provides the timer service that allows for the management of timers to be used for timing MA operations.
7. *Resource manager (RM).* The RM component provides access to the sensor node resources: sensors/actuators, battery and flash memory.

A.3 Agent programming model

The architecture of an MA is modeled as a multi-plane state machine communicating through events (see Figure A.2).

This architecture allows exploiting the benefits derived from three paradigms for WSN programming: event-driven programming [12], state-based programming [93] and agent-based programming [94]. Moreover, it enables role-based programming, an important paradigm for agents, as agents behave differently according to the role they can assume during their life cycle [95]. In particular the architecture consists of:

- *Global variables (GV).* The GV component represents the data of the MA including the MA identity.
- *Global functions (GF).* The GF component consists of a set of supporting functions which can access GV but can invoke neither core primitives nor other functions.
- *Multi-plane state machine (MPSM).* The MPSM component consists of a set of planes. Each plane may represent the behavior of the MA in a specific role. In particular a plane is composed of:

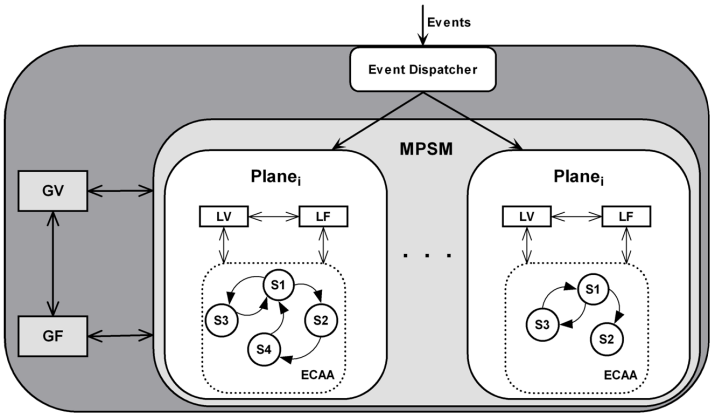


Fig. A.2. MAPS agent model.

- *Local variables (LV).* The LV component represents the local data of a plane.
- *Local functions (LF).* The LF component consists of a set of local plane supporting functions which can access LV but can invoke neither core primitives nor other functions.
- *ECA-based automata (ECAA).* The ECAA component represents the dynamic behavior of the MA in that plane and is composed of states and mutually exclusive transitions among states. Transitions are labeled by ECA rules: E[C]/A, where E is the event name, [C] is a boolean expression based on the GV and LV variables, and A is the atomic action. A transition t is triggered if t originates from the current state (i.e. the state in which the ECAA component is), the event with the event name E occurs and [C] holds. When the transition fires, A is first executed and, then, the state transition takes place. In particular, the atomic action can use GV, GF, LV and LF for performing computations and, particularly, invoking the core primitives to asynchronously emit one or more events. The delivery of an event is asynchronous and can occur only when the ECAA is idle, i.e. the handling of the last delivered event (ED) is completed.

• *Event dispatcher (ED).* The ED component dispatches the event delivered by the MAEE to one or more planes according to the events that the planes are able to handle. In particular, if an event must be dispatched to more than one plane, the event dispatching is appositely serialized.

A.4 Implementation

The implementation of MAPS is a real challenge due to the constrained resources of the current sensor nodes. Nevertheless, an actual implementation

could be done in nesC/TinyOS on TelosB motes or in Java on SunSPOT nodes. Although the implementations of the currently available mobile agent frameworks for WSN have to date been carried out in nesC/TinyOS (e.g. through the Mate' virtual machine [96]), we believe that the object-oriented features offered by the Sun SPOT technology could provide more flexibility and extendability as well as easiness of development for an efficient implementation of the proposed framework. Specifically, the offered features are the following:

- *Java programming language.* Sensor node software is programmed in the Java language by using Java standard libraries and specific Sun SPOT libraries such as main Sun SPOT board classes, sensor board transducer classes and Squawk operating environment classes.
- *NetBeans IDE for software development.* The IDE fully supports code editing, compilation, deployment and execution for SunSPOTs. This enables a more rapid software prototyping.
- *Single-hop/multi-hop and reliable/unreliable communications.* The current version of the Sun SPOT SDK uses the GCF (Generic Connection Framework) to provide radio communication between SPOTs, routed via multiple hops if necessary. Two protocols are available: the radiostream protocol and the radiogram protocol. The radiostream protocol provides reliable, buffered, stream-based communication between two devices.The radiogram protocol provides datagram-based communication between two devices and broadcast communications. This protocol provides no guarantees about delivery or ordering. Datagrams sent over more than one hop could be silently lost, be delivered more than once and be delivered out of sequence. Datagrams sent over a single hop will not be silently lost or delivered out of sequence, but they could be delivered more than once.The protocols are implemented on top of the MAC layer of the 802.15.4 implementation.
- *Easy access to the sensor node devices (sensors, flash memory, timer, battery).* The Sun SPOT device libraries contains drivers to easily access and use the following: the on-board LED, the PIO, AIC, USART and Timer-Counter devices in the AT91 package, the CC2420 radio chip (in the form of an IEEE 802.15.4 Physical interface), an IEEE 802.15.4 MAC layer, an SPI interface (used for communication with the CC2420 and off-board SPI devices) and an interface to the flash memory.
- *Code migration support.* An Isolate is a mechanism by which an application is represented as an object. In Squawk, one or more applications can run in the single JVM. Conceptually, each application is completely isolated from all other applications. The Squawk implementation has the interesting feature of Isolate migration, i.e. an Isolate running on one Squawk VM instance can be paused, serialized to a file or over a network connection and restarted in another Squawk VM instance.

A.5 An agent-based system for monitoring human activity

This section describes a MAPS-based implementation [42] of the real-time human activity monitoring discussed in Section 4.1. The system is able to recognize postures (e.g. lying down, sitting and standing still) and movements (e.g. walking) of assisted livings. It is designed and implemented with MAPS at the sensor node side and through Java and JADE at the coordinator side.

A.5.1 Design and Implementation

The architecture of the system, shown in Figure A.3, is organized into one coordinator and two sensor nodes.

The coordinator side (see Figure A.3) is based on a JADE agent that incorporates two modules of the Java-based SPINE coordinator, which are the SPINE Manager and the SPINE Listener (see Section 3.9). In particular, the SPINE Manager is used by end-user applications (e.g. real-time activity monitoring application) for sending commands to the sensor nodes. Moreover, the SPINE Manager is responsible of capturing low-level messages and events sent from the nodes through the SPINE Listener, which integrates several sensor platform-specific SPINE communication modules (e.g. TinyOS and Z-Stack), to notify registered applications with higher-level events and message content. A SPINE communication module is composed of a send/receive interface and additional components which implement such interface according to the specific sensor platform and that formalize the high-level SPINE messages in sensor platform-specific messages. In this work, the SPINE Listener has been enhanced with a new MAPS/Sun SPOT communication module to configure and communicate with MAPS-based sensor nodes. Such module translates high-level SPINE messages formatted according to the SPINE communication protocol into lower-level MAPS/Sun SPOT messages through its transmitter component and vice versa through its receiver component. The JADE agent coordinator also integrates an application-specific logic for the synchronization of the two sensors (see below). The SPINE-based real-time activity monitoring application was thus completely reused as well as the SPINE Manager, only the SPINE Listener was modified to account for such enhancement.

The sensor node side (see Figure A.3) is based on two Java Sun SPOTs sensors respectively positioned on the waist and the thigh of the monitored person. In particular, MAPS is resident on the sensor nodes and supports the execution of the WaistSensorAgent and the ThighSensorAgent, which have the following similar step-wise cyclic behavior:

1. *Sensing* the 3-axial accelerometer sensor according to a given sampling time (ST);

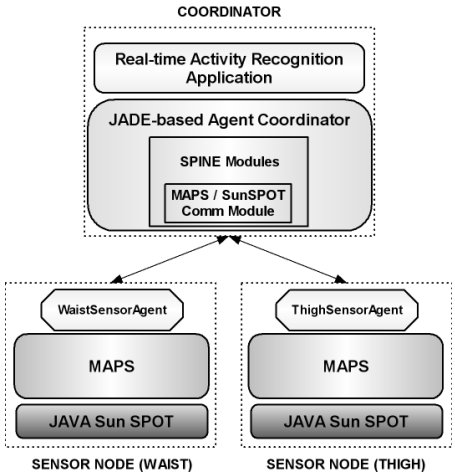


Fig. A.3. Architecture of the real-time activity monitoring system.

2. *Computation* of specific features on the acquired raw data according to the window (W) and shift (S) parameters. In particular, W is the sample size on which features are computed whereas S is the percentage of sliding on W (usually S is set to 50%);
3. Features *aggregation* and *transmission* to the coordinator;
4. Goto 1.

The agents differ in the specific computed features even though the W and S parameters are equally set. In particular, while the WaistSensorAgent computes the mean values for the accelerometer data sensed on the XYZ axes, the min and max values for data sensed on the X axis, the ThighSensorAgent calculates the min value for data sensed on the X axis.

The interaction diagram depicted in Figure A.4 shows the interaction among the three agents costituting the real-time system: CoordinatorAgent, WaistSensorAgent and ThighSensorAgent. In particular, the CoordinatorAgent first sends one AGN_START event for each sensor agent to configure them with the sensing parameters (W, S and ST); then, it broadcasts the START event to start the sensing activity of the sensor agents. Sensor agents sends the DATA event to the CoordinatorAgent as soon as features are computed. If the CoordinatorAgent detects that the agents are not synchronized anymore, it sends the RESYNCH event to resynchronize them.

The behavior of the WaistSensorAgent is specified through 1-plane reported in Figure A.5 (the behavior of the ThighSensorAgent has the same structure but the computed features are different as discussed above). In particular, after an initialization action (A0) driven by the occurrence of the AGN_START event, the sensing plane goes into the WAIT4SENSING state.

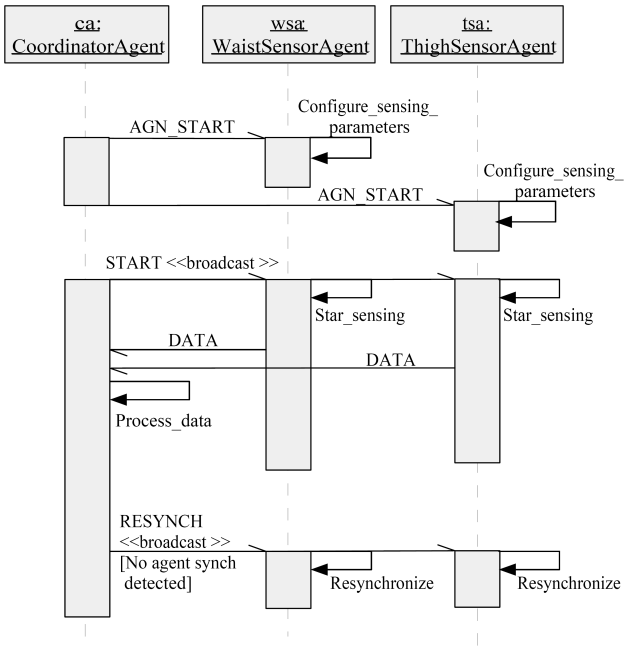


Fig. A.4. Agents interaction of the real-time activity monitoring system.

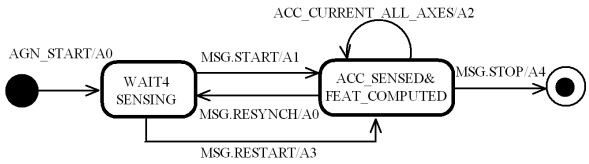


Fig. A.5. 1-plane behavior of the WaistSensorAgent.

The MSG.START event allows starting the sensing process by the execution of action A1, which in particular performs the following steps:

1. sensing parameters (W, S, ST), data acquisition buffers for XYZ channels of the accelerometer sensor (windowX, windowY, windowZ), and data buffers for feature calculation (windowFE4X, windowFE4Y, windowFE4Z) are initialized (see *initSensingParamsAndBuffers* function);
2. the timer is set for timing the data acquisition according to the ST parameter (see *timerSetForSensing* function and in particular the highly precise Sun SPOT timer is used);
3. a data acquisition is requested by submitting the ACC_CURRENT_ALL-_AXES event through the sense primitive (see *doSensing* function).

Once the data sample is acquired, the ACC_CURRENT_ALL_AXES event is sent back with the acquired data and the action A2 is executed; in particular:

1. the buffers are circularly filled with the proper values (see *bufferFilling* function);
2. the sampleCounter is incremented and the nextSampleIndex is incremented module W for the next data acquisition;
3. if S samples have been acquired, features are to be calculated, thus sampleCounter is reset, samples in the buffers are copied into the buffers for computing features, calculation of the features is carried out through the *meanMaxMin* function, and the aggregated results are sent to the base station by means of the MSG_TO_BASESTATION event appropriately constructed;
4. the timer is reset;
5. data acquisition is finally requested.

In the ACC_SENSED&FEAT_COMPUTED state the MSG.RESYNCH might be received for resynchronization purposes; it brings the sensing plane into the WAIT4SENSING state. The MSG.RESTART brings the sensing plane back into the ACC_SENSED&FEAT_COMPUTED state for (reconfiguring and) continuing the sensing process. The MSG.STOP eventually terminates the sensing process.

A.5.2 Performance evaluation

Two important issues to deal with are the timing of the sensing process in terms of admissible sampling rate and the synchronization between the operations of the two agents which is to be maintained within a maximum skew for not affecting the real-time monitoring. If such skew is overtaken, the two agents are to be re-synchronized. Indeed such two aspects are strictly correlated. In particular, as the sensor agents compute a different number of features, when the sampling rate is high, the agent computing more features (i.e. the WaistSensorAgent) takes more time to complete its operations for each S sample acquisition than the ThighSensorAgent. Re-synchronization is driven by the synchronization logic included in the developed MAPS/Sun SPOT comm module, which sends a resynchronization message (see the MSG.RESYNCH event in Figure A.5) as soon as it detects that the synchronization skew is greater than a given threshold. Detection is based on the skew time between the receptions of two messages sent by the agents that contain features referring to the same interval of S sample acquisition: if $skew >= P*S*ST$ then synchronize, where P is a percentage, S=0.5W, and ST is the sampling time. Thus, the evaluation aimed at analyzing the synchronization of the sensor agents and their monitoring continuity. The defined measurements are:

- The *Packet Pair Average Time* (PPAT), which is the average reception time between two consecutive pairs of synchronized packets (same logical

timestamp, see timestamp variable in Figure A.5) containing the computed features (see the MSG_TO_BASESTATION event in Figure A.5) sent by the sensor agents. PPAT should be ideally equals to ST*S, i.e. the packet pair arrives each monitoring period and so there is no de-synchronization in the average.

- The *Synchronization Packet Percentage* (SPP), which is the percentage of resynchronization packets (see RESYNCH event in Figure A.5), which are sent by the coordinator for re-synchronizing the sensor agents, calculated with respect to the total number of received feature packets. SPP should be as much as possible close to 0, i.e. a few or no resynchronizations are carried out and so the monitoring can be continuous as a resynch operation usually takes 600 ms.

In particular, the experiments were carried out by fixing ST (ms) = [25, 50, 100], W (samples) = [100, 80, 40, 20, 10], and P (%) = [5, 10, 25]. Each experiment took 15 minutes and 50 tests per experiment were carried out. The obtained values were averaged over the 50 tests performed (also the standard deviation is reported). The values of ST and W were chosen to evaluate the system under different operating conditions: from high (ST=25 ms, W=10, S=50% –> response time = 125 ms) to slow (ST=100 ms, W=100, S=50% –> response time = 5 s) system response times. The system response time can directly affect the accuracy of the human activity recognition as the higher is the frequency of refreshing the human activity status, the quicker is the capability of the system to capture human activity changes. Moreover the variation range of P% accommodates for small to medium skews.

Figure A.6 shows the obtained results for P=25% and P=5% by varying ST and W in the ranges defined above. As can be noticed, the system cannot support an ST=25 ms because PPAT is always greater than the ideal value and SPP is too high. This leads to a non continuous monitoring due to very frequent resynchronizations (SPP $\geq$15%). An ST=50 ms can be supported for P=25% and W$\geq$40 as SPP is maximum 8% so slightly impacting the monitoring continuity. The best results are obtained with ST=100 ms, P=25% and W$\geq$20; they guarantee monitoring continuity due to an SPP$\approx$ 0% and regularity as experimented PPAT $\approx$ ideal PPAT for W$\geq$20. If P=5% and W=[10, 20] or P=25% and W=10, an ST=100 ms is not a good value either because an out-of-limits skew frequently occurs.

It is worth noting that even though a lower ST would allow a more frequent monitoring, the considered human activities can be well captured by an ST=100 ms and W=20 (which implies a response time = 1 s) as demonstrated by the experimental results obtained from the carried-out real-time human activity monitoring.

To compare the efficiency of the MAPS- and SPINE-based implementations of the system, the experiments were carried out by fixing ST (ms) = [25, 50, 100], W (samples) = [40, 20], and P (%) = [5, 25]. Each experiment took 15 minutes and 50 tests per experiment were carried out. Figures A.7 and

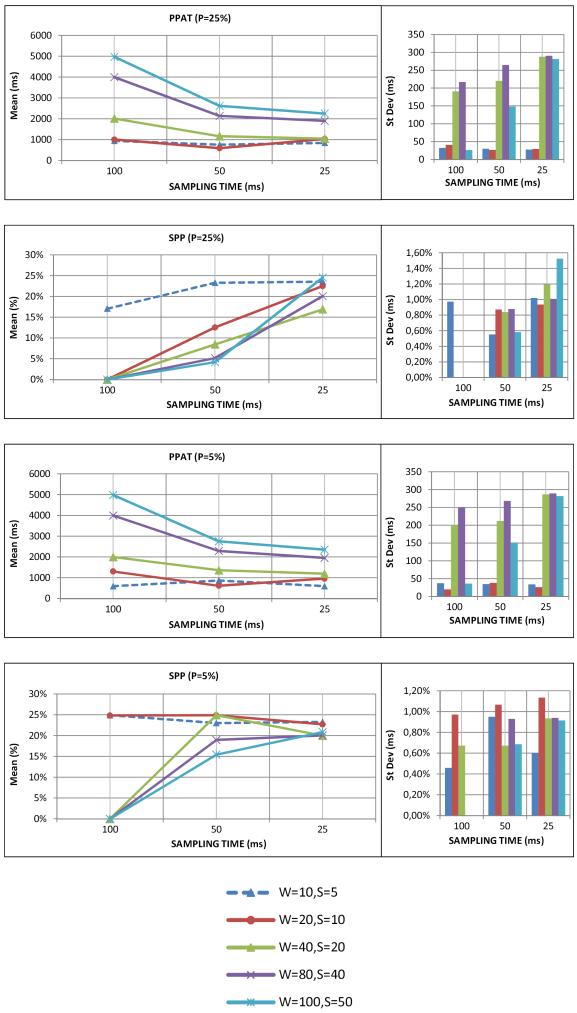


Fig. A.6. Analysis of the synchronization of the MAPS sensor agents: PPAT and SPP for P=25% and P=5% by varying W.

A.8 show the obtained results, which are the average values of the 50 tests (also the standard deviation is reported). As can be noticed, all the systems cannot support an ST=25ms because PPAT is always greater than the ideal value and SPP is too high. This leads to a non continuous monitoring due to the very frequent resynchronization (SPP≥20 for W=20 and S=10). The best results are obtained with ST=100ms, P=25% and W=20; they guarantee monitoring continuity due to an SPP≈0% and regularity as experimented PPAT≈ideal PPAT for W=20. If W=20 and P=5%, ST=100 ms is not a good

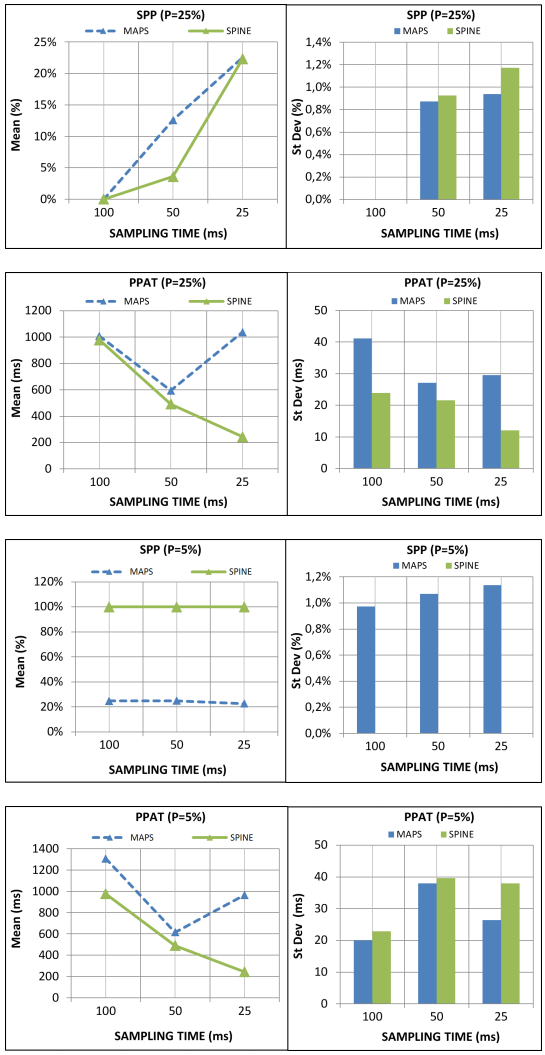


Fig. A.7. Comparison of the synchronization between MAPS and SPINE sensor agents: PPAT and SPP for P=25% and P=5% and W=20, S=10.

value either because an out-of-limits skew frequently occurs. SPINE performs better for the parameters ST=100, W=40, and P=25% whereas it has lower performance in the other cases. On the basis of the obtained results we can state that MAPS on Sun SPOT shows comparable performances with SPINE on TelosB sensors, which is a domain-specific framework for WBSNs, so confirming its suitability for supporting efficient WBSN applications.

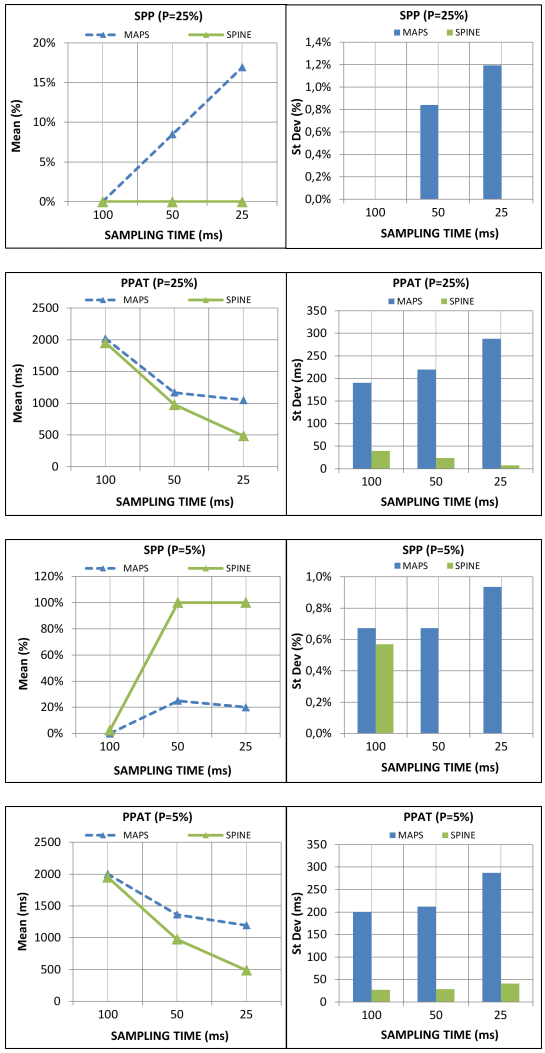


Fig. A.8. Comparison of the synchronization between MAPS and SPINE sensor agents: PPAT and SPP for P=25% and P=5% and W=40, S=20.

A.5.2.1 Recognition accuracy

The activity monitoring system integrates a classifier based on the K-Nearest Neighbor algorithm [58] that is capable of recognizing postures and movements defined in a training phase. The classifier was setup through a training phase and tested considering the following parameter setting: ST=100ms, W=20 (S=10), P=25%. Accordingly, the features (Min, Max and Mean) are computed on 20 sampled data every new 10 samples acquired by the sensors.

The training phase used a KNN-based classifier parameterized with K=1 and the Manhattan distance which performs quite well as classes (lying down, sitting, standing still and walking) are rather separate and scarcely affected by noise. The test phase is carried out by considering the pre-defined sequence of postures/movements represented by the state machine reported in Figure A.9.

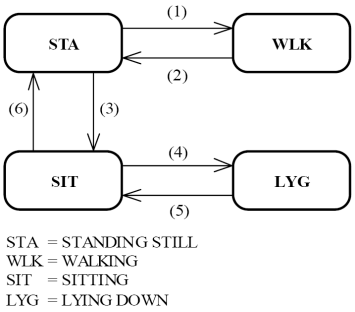


Fig. A.9. State machine of the pre-defined sequence of postures/movements.

Accordingly, the obtained classification accuracy results are reported in Figure A.10. As can be noted after a transitory period of 5 s from one state to another, all the postures/movements are recognized with an accuracy of 100%. The state transitions more difficult to recognize are STA→SIT, WLK→STA, and SIT→LYG, whereas the transition STA→WLK is recognized as soon as it occurs. The obtained results are good and encouraging if compared with other works in the literature which use more than two sensors on the human body to recognize activities [97].

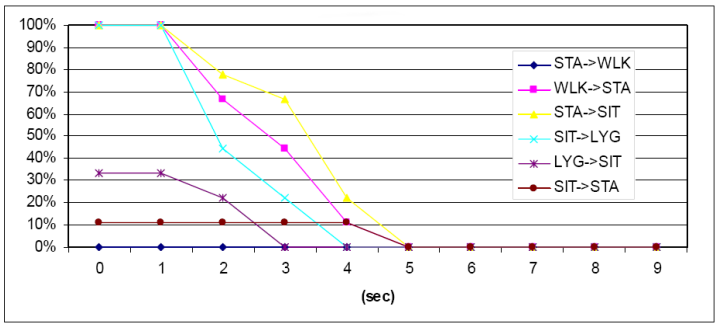


Fig. A.10. Percentage of mismatches vs. transitory time computed with ST=100 ms, W=20, P=25%.

References

[1] P. Alexandros and B. Nikolaos. A Survey on Wearable Sensor-Based Systems For Health Monitoring and Prognosis. *IEEE Transactions on Systems, Man and Cybernetics*, 40(1):1–12, 2010.

[2] S. Ullah, P. Khan, N. Ullah, S. Saleem, H. Higgins, and K.-S. Kwak. A Review of Wireless Body Area Networks for Medical Applications. *International Journal of Communications, Network and System Sciences*, 2(8):797–803, 2009.

[3] J. Yick, B. Mukherjee, and D. Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292–2330, 2008.

[4] IEEE 802.15.4 website. http://www.ieee802.org/15/pub/tg4.html, 2011.

[5] ZigBee website. http://www.zigbee.org, 2011.

[6] Bluetooth website. http://www.bluetooth.com, 2011.

[7] Bluetooth Low Energy website. http://www.bluetooth.com/pages/low-energy.aspx, 2011.

[8] ANT website. http://www.thisisant.com, 2011.

[9] IEEE 802.15 WPAN Task Group 6 website. http://ieee802.org/15/pub/tg6.html, 2011.

[10] Adi Mallikarjuna V. Reddy, A.V.U. Phani Kumar, D. Janakiram, and G. Ashok Kumar. Wireless Sensor Network Operating Systems: a Survey. *International Journal of Sensor Networks*, 5(4):236–255, August 2009.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. *ACM SIGPLAN Notices*, 35(11), November 2000.

[12] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The NesC language: A holistic approach to networked embedded systems. *ACM SIGPLAN Notices*, 38(5):1–11, May 2003.

[13] T. Von Eicken, D. Culler, S.-C. Goldstein, and K.-E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA'92, pages 256–266. ACM Press, May 1992.

[14] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN'04, pages 455–462. IEEE Computer Society, November 2004.

[15] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, 10(4):563–579, January 2005.

[16] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS 2005, pages 256–265. IEEE Computer Society, December 2005.

[17] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Readings in multimedia computing and networking. chapter Resource kernels: a resource-centric approach to real-time and multimedia systems, pages 476–490. Morgan Kaufmann Publishers Inc., 2001.

[18] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White. Java on the bare metal of wireless sensor devices: the Squawk Java virtual machine. In *Proceedings of the 2nd International Conference on Virtual Execution Environments*, VEE 2006, pages 78–88, June 2006.

[19] SunSPOT website. http://www.sunspotworld.com, 2011.

[20] Z-Stack website. http://www.ti.com/tool/z-stack, 2011.

[21] 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 *Proceedings of the International Conference on Body Sensor Networks*, BSN 2010, pages 228–233. IEEE Computer Society, June 2010.

[22] K. Lorincz, D.-J. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor Networks for Emergency Response: Challenges and Opportunities. *IEEE Pervasive Computing*, 3(4):16–23, October 2004.

[23] T. Terada and K. Tanaka. A framework for constructing entertainment contents using flash and wearable sensors. In *Proceedings of the 9th International Conference on Entertainment computing*, ICEC'10, pages 334–341. Springer-Verlag, 2010.

[24] S. Coyle, D. Morris, K. Lau, N. Moyna, and D. Diamond. Textile-based wearable sensors for assisting sports performance. In *Proceedings of the International Conference on Body Sensor Networks*, BSN 2009, pages 228–233. IEEE Computer Society, June 2010.

[25] J.-Y. Huang and C.-H. Tsai. A wearable computing environment for the security of a large-scale factory. In *Proceedings of the 12th International Conference on Human-computer interaction: interaction platforms and techniques*, HCI'07, pages 1113–1122. Springer-Verlag, July 2007.

[26] A. Augimeri, G. Fortino, M. Rege, V. Handzisky, and A. Wolisz. A Cooperative Approach for Handshake Detection based on Body Sensor Networks. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, SMC 2010, pages 281–288. IEEE Press, October 2010.

[27] C.-L. Fok, G.-C. Roman, and C. Lu. Mobile Agent Middleware for Sensor Networks: An Application Case Study. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN 2005, pages 382–387. IEEE Signal Processing Society, April 2005.

[28] R. Kumar, M. Wolenetz, B. Agarwalla, J.-S., P. Hutto, A. Paul, and U. Ramachandran. DFuse: A Framework for Distributed Data Fusion. In *Proceedings*

of the 1st International Conference on Embedded networked sensor systems, SenSys'03, pages 114–125. ACM Press, November 2003.

[29] S. Madden, M.-J. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, OSDI'02. ACM Press, December 2002.

[30] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, December 2005.

[31] Y. Hao and R. Foster. Wireless body sensor networks for health-monitoring applications. *Physiological Measurement*, 29(11):R27–R56, November 2009.

[32] B. Grundlehner, L. Brown, J. Penders, and G. Gyselinckx. The Design and Analysis of a Real-Time, Continuous Arousal Monitor. In *Proceedings of the 6th International Workshop on Wearable and Implantable Body Sensor Networks*, BSN 2009, pages 156–161. IEEE Press, June 2009.

[33] C.-W. Mundt, K.-N. Montgomery, U.-E. Udoh, V.-N. Barker, G.-C. Thonier, A.-M. Tellier, R.-D. Ricks, R.-B. Darling, Y.-D. Cagle, N.-A. Cabrol, S.-J. Ruoss, J.-L. Swain, J.-W. Hines, and G.-T. Kovacs. A Multiparameter Wearable Physiologic Monitoring System for Space and Terrestrial Applications. *IEEE Transactions on Information Technology in Biomedicine*, 9(3):382–391, September 2005.

[34] Fitbit website. http://www.fitbit.com, 2011.

[35] Vitalsense website. http://vitalsense.respironics.com, 2011.

[36] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *Proceedings of the MobiSys 2004 Workshop on Applications of Mobile Embedded Systems*, WAMES 2004. ACM Press, June 2004.

[37] J.-G. Jetcheva and D.-B. Johnson. Adaptive Demand-Driven Multicast Routingin Multi-Hop Wireless AdHoc Networks. In *Proceedings of the International Symposium on Mobile AdHoc Networking and Computing*, MobiHoc 2001, pages 33–44. ACM Press, October 2001.

[38] M. Zhang and A. Sawchuk. A Customizable Framework of Body Area Sensor Network for Rehabilitation. In *Proceedings of the 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies*, ISABEL 2009, pages 24–27. IEEE Press, November 2009.

[39] C. Lombriser, D. Roggen, M. Stager, and G. Troster. Titan: A Tiny Task Network for Dynamically Reconfigurable Heterogeneous Sensor Networks. In *Proceedings of the 15th Fachtagung Kommunikation in Verteilten Systemen*, KiVS 2007, pages 127–138. Springer, February 2007.

[40] C. Muldoon, G.-M.P. O'Hare, M.-J. O'Grady, and R. Tynan. Agent Migration and Communication in WSNs. In *Proceedings of the 9th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT 2008, pages 425–430. IEEE Computer Society, December 2008.

[41] A. S. Rao and M. P. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the 1st International Conference on Multi-Agent Systems*, ICMAS'95, pages 312–319. MIT Press, June 1995.

[42] F. Aiello, F. Bellifemine, G. Fortino, S. Galzarano, and R. Gravina. An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks. *Journal of Engineering Applications of Artificial Intelligence*, 24(7):1147–1161, 2011.

[43] W.-B. Heinzelman, A.-L. Murphy, H.-S. Carvalho, and M.-A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, Jan/Feb 2004.

[44] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi. SPINE: A domain-specific framework for rapid prototyping of WBSN applications. *Software: Practice & Experience*, 41(3):237–265, March 2011.

[45] SPINE website. http://spine.tilab.com, 2011.

[46] G. Holmes, A. Donkin, and I.H. Witten. Weka: A machine learning workbench. In *Proceedings of the 2nd Australia and New Zealand Conference on Intelligent Information Systems*, ANZIIS'94, pages 1269–1277. IEEE Press, 1994.

[47] Android website. http://www.android.com, 2011.

[48] Bluecove API website. http://bluecove.org, 2011.

[49] Android Bluetooth API. http://developer.android.com/guide/topics/wireless/bluetooth.html, 2011.

[50] Shimmer website. http://www.shimmer-research.com/, 2011.

[51] A. Augimeri, G. Fortino, S. Galzarano, and R. Gravina. Collaborative Body Sensor Networks. In *Proceedings of the IEEE International Conference on Systems, Man and Cybertnetics*, SMC 2011. IEEE Press, October 2011.

[52] F. Aiello, G. Fortino, and A. Guerrieri. Using Mobile Agents as Enabling Technology for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Sensor Technologies and Applications*, SENSORCOMM'08, pages 549–554. IEEE Computer Society, August 2008.

[53] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri. A Java-based Agent Platform for Programming Wireless Sensor Networks. *The Computer Journal*, 54(3):439–454, 2011.

[54] M. Luck, P. McBurney, and C. Preist. A Manifesto for Agent Technology: Towards Next Generation Computing. *Autonomous Agents and Multi-Agent Systems*, 9(3):203–252, 2004.

[55] JADE website. http://jade.tilab.com, 2011.

[56] 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 Journal*, doi:10.1109/JSEN.2011.2121059, 2011.

[57] F. Bellifemine, G. Fortino, R. Giannantonio, and A. Guerrieri. Platform-independent development of collaborative wireless body sensor network applications: SPINE2. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, SMC 2009, pages 3144–3150. IEEE Press, October 2009.

[58] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, January 1967.

[59] P. Pudil, J. Novovicova, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, November 1994.

[60] Food and National Academy of Sciences Nutrition Board. Dietary reference intakes for energy, carbohydrates, fiber, fat, protein and amino acids. *National Academy Press*, 2002.

[61] U.S. Department of Health & Human Services, Centers for Disease Control & Prevention, and The National Center for Chronic Disease Prevention & Health Promotion. Physical activity and health: A report of the Surgeon General, 1996.

[62] K.Y. Chen and M. Sun. Improving energy expenditure estimation by using a triaxial accelerometer. *Journal of Applied Physiology*, 83(6):2112–2122, 1997.

[63] D. Mizell. Using gravity to estimate accelerometer orientation. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, ISWC'03, pages 252–253. IEEE Computer Society, October 2003.

[64] 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. *Wearable and Autonomous Biomedical Devices and Systems: New issues and Characterization - Lecture Notes on Electrical Engineering*, 75:369–389, 2010.

[65] B.S. McEwen. Protective and Damaging Effects of Stress Mediators. *The New England Journal of Medicine*, 338(3):171–179, 1998.

[66] S.-C. Segerstrom and G.-E. Miller. Psychological stress and the human immune system: A meta-analytic study of 30 years of inquiry. *Psychological Bulletin*, 130(4):601–630, 2004.

[67] H.-K. Yang, J.-W. Lee, K.-H. Lee, Y.-J. Lee, K.-S. Kim, H.-J. Choi, and D.-J. Kim. Application for the wearable heart activity monitoring system: Analysis of the autonomic function of HRV. In *Proceedings of the 30th Annual International Conference on Engineering in Medicine and Biology Society*, EMBS 2008, pages 1258–1261. IEEE Press, August 2008.

[68] K. Keutzer, A.-R. Newton, J.-M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(12):1523–1543, December 2000.

[69] A. Bonivento. *Platform Based Design for Wireless Sensor Networks*. PhD thesis, University of California at Berkeley, 2007.

[70] A. Bonivento, L.-P. Carloni, and A. Sangiovanni-Vincentelli. Platform Based Design for Wireless Sensor Networks. *Mobile Networks and Applications*, 11(4):469–485, August 2006.

[71] R. Passerone. *Semantic Foundations for Heterogeneous Systems*. PhD thesis, University of California at Berkeley, 2004.

[72] L. Ma, Y. Sun, N. Feng, , and Z. Liu. Image Fast Template Matching Algorithm Based on Projection and Sequential Similarity Detection. In *Proceedings of the Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, IIH-MSP 2009, pages 957–960. IEEE, September 2009.

[73] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[74] F. Balarin, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Concurrent execution semantics and sequential simulation algorithms for the metropolis meta-model. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 13–18. ACM, May 2002.

[75] F. Balarin, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, M. Sgroi, and Y. Watanabe. Modeling and designing heterogeneous systems. *Concurrency and Hardware Design, Advances in Petri Nets*, pages 228–273, 2002.

[76] F. Bellifemine, G. Fortino, R. Giannantonio, R. Gravina, A. Guerrieri, and M. Sgroi. Development of Body Sensor Network Applications using SPINE. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, SMC 2008, pages 2810–2815. IEEE Press, October 2008.

[77] G. Fortino, S. Galzarano, R. Giannantonio, R. Gravina, and A. Guerrieri. SPINE-based Application Development on Heterogeneous Wireless Body Sensor Networks. *International Journal of Computing*, 9(1):80–89, 2010.

[78] P. Kuryloski, A. Giani, R. Giannantonio, K. Gilani, R. Gravina, V.-P. Seppä, E. Seto, V. Shia, C. Wang, P. Yan, A.-Y. Yang, J. Hyttinen, S. Sastry, S. Wicker, and R. Bajcsy. DexterNet: An Open Platform for Heterogeneous Body Sensor Networks and Its Applications. In *Proceedings of the International Conference on Body Sensor Networks*, BSN 2009, pages 92–97. IEEE Computer Society, June 2009.

[79] 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 *Proceedings of the 3rd International Conference on Pervasive Computing Technologies for Healthcare*, Pervasive Health 2009, pages 1–8. IEEE Press, April 2009.

[80] R. Giannantonio, F. Bellifemine, R. Gravina, A. Guerrieri, G. Fortino, and M. Sgroi. SPINE: Signal Processing In Node Environment. In *Proceedings of the 1st European TinyOS Technology Exchange*, ETTX 2009, February 2009.

[81] L. Buondonno, G. Fortino, S. Galzarano, R. Giannantonio, A. Giordano, R. Gravina, and A. Guerrieri. Programming Signal Processing Applications on Heterogeneous Wireless Sensor Platforms. In *Proceedings of the 5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IDAACS 2009, pages 682–687. IEEE Press, September 2009.

[82] N. Raveendranathan, V. Loseu, E. Guenterberg, R. Giannantonio, R. Gravina, M. Sgroi, and R. Jafari. Implementation of Virtual Sensors in Body Sensor Networks with the SPINE Framework. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems*, SIES 2009, pages 124–127. IEEE Press, July 2009.

[83] F. Aiello, G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. An Analysis of Java-based Mobile Agent Platforms for Wireless Sensor Networks. *Multi-Agent and GRID Systems*, to appear:1–30, 2011.

[84] G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. Agent-based Development of Wireless Sensor Network Applications. In *Proceedings of the 12th Workshop on Objects and Agents*, WOA 2011. CEUR Workshop Proceedings, July 2011.

[85] F. Aiello, G. Fortino, S. Galzarano, R. Gravina, and A. Guerrieri. *Wireless Body Area Networks: Technology, Implementation and Applications*, chapter 5 - Signal processing in-node frameworks for Wireless Body Sensor Networks: from low-level to high-level approaches, pages 1–23. Pan Stanford publishing, 2011.

[86] F. Aiello, F. Bellifemine, G. Fortino, R. Gravina, and A. Guerrieri. An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks. In *Proceedings of the 1st International Workshop on Infrastructures and Tools for Multiagent Systems, jointly held with 9th International Conference AAMAS*, ITMAS 2010, May 2010.

[87] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri. MAPS: a Mobile Agent Platform for Java Sun SPOTs. In *Proceedings of the 3rd Workshop on Agent Technology for Sensor Networks, jointly held with the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, ATSN 2009, May 2009.

[88] D.L. Carni', G. Fortino, D. Grimaldi, R. Gravina, A. Guerrieri, and F. Lamonaca. Continuous, Real-time Monitoring of Assisted Livings through Wireless Body Sensor Networks. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IDAACS 2011. IEEE Press, September 2011.

[89] A. Andreoli, R. Gravina, R. Giannantonio, P. Pierleoni, and G. Fortino. Time-Domain Heart Rate Variability Analysis with the SPINE-HRV. In *Proceedings of the 1st International Workshop on SigProcessing (Light-weight Signal Processing for Computationally Intensive BSN Applications)*, PETRA 2010. ACM Press, June 2010.

[90] E. Seto, M. Eladio, A.-Y. Yang, P. Yan, R. Gravina, I. Lin, C. Wang, M. Roy, V. Shia, and R. Bajcsy. Opportunistic Strategies for Lightweight Signal Processing for BSN. In *Proceedings of the 1st International Workshop on SigProcessing (Light-weight Signal Processing for Computationally Intensive BSN Applications)*, PETRA 2010. ACM Press, June 2010.

[91] MAPS website. http://maps.deis.unical.it, 2011.

[92] M. Welsh and G. Mainland. Programming Sensor Networks Using Abstract Regions. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, NSDI'04, pages 3–16. USENIX Association Berkeley, March 2004.

[93] O. Kasten and K. Romer. Beyond Event Handlers: Programming Wireless Sensors with Attributed State Machines. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN 2005, pages 45–52. IEEE Press, April 2005.

[94] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In *Proceedings of the 24th International Conference Distributed Computing Systems*, ICDCS'05, pages 653–662. IEEE Computer Society, June 2005.

[95] H. Zhu and R. Alkins. Towards Role-Based Programming. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, CSCW'06. ACM, November 2006.

[96] P. Levis and D. Culler. Mate': A Tiny Virtual Machine for Sensor Networks. In *Proceedings of the 10th International Conference Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pages 85–95. ACM Press, October 2002.

[97] U. Maurer, A. Smailagic, D.-P. Siewiorek, and M. Deisher. Activity Recognition and Monitoring Using Multiple Sensors on Different Body Positions. In *Proceedings of the 3rd International Workshop on Wearable and Implantable Body Sensor Networks*, BSN 2006, pages 113–116. IEEE Computer Society, 2006.