Università della Calabria

Dipartimento di Elettronica, Informatica e Sistemistica

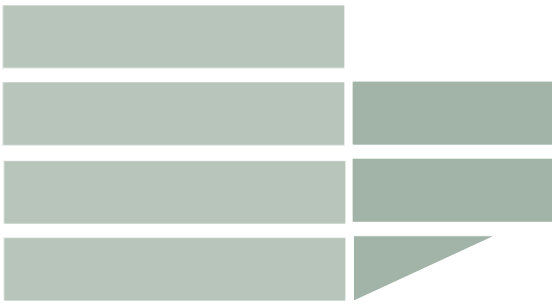
Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XXIV Ciclo

Tesi di Dottorato

High-level Frameworks for the Development of Wireless Sensor Network Applications

Antonio Guerrieri



Università della Calabria

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XXIV Ciclo

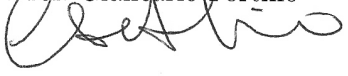
Tesi di Dottorato

High-level Frameworks for the Development of Wireless Sensor Network Applications

Antonio Guerrieri

Coordinatore
Prof. Luigi Palopoli

Supervisore
Prof. Giancarlo Fortino



Dipartimento di Elettronica, Informatica e Sistemistica

Novembre 2011

Settore Scientifico Disciplinare: ING-INF/05

Abstract

Wireless Sensor Networks (WSNs) are emerging as powerful platforms for distributed embedded computing supporting a variety of high-impact applications. A WSN is a group of small devices (nodes) capable to sample the real world through sensors, actuate commands through actuators, elaborate data on the node, and send messages to other nodes through radio communication. However, programming WSN applications is a complex task that requires suitable paradigms and technologies capable of supporting the specific characteristics of such networks which uniquely integrate distributed sensing, computation and communication.

This thesis aims at providing new paradigms to support the development of WSN applications through both a domain-specific and a general-purpose approach. In particular, this thesis provides three main contributions. The first is related to the analysis, design and realization of a domain-specific framework for heterogeneous WSNs for flexible and efficient distributed sensing and actuation in buildings called Building Management Framework (BMF). BMF provides fast WSN reconfiguration, in-node processing algorithms, multi-hop networks, and multi-platform support, a programming abstraction to dynamically catch the morphology of buildings, actuators support, and an extensible human computer interface. The second contribution refers to the analysis, design and realization of a general-purpose mobile agent system for WSN, namely MAPS (Multi Agent Platform for SunSPOT). MAPS allows an effective Java-based development of agents and agent-based applications for WSNs by integrating agent oriented, event-driven and state-based programming paradigms. Finally, the third contribution regards the analysis, design and realization of a domain-specific framework for rapid prototyping of platform independent Wireless Body Sensor Network (WBSN) applications, namely SPINE2 (signal processing in-node environment version 2). SPINE2 aims at supporting the development of WSN applications raising the level of the used programming abstractions by providing a task-oriented programming model.

Riassunto

Le Wireless Sensor Networks (WSNs), o reti di sensori wireless, stanno emergendo come potenti piattaforme di calcolo distribuito capaci di supportare una grande varietà di applicazioni ad alto impatto tecnologico. Una WSN è un gruppo di piccoli dispositivi (nodi) capaci di campionare il mondo reale tramite sensori, attuare comandi tramite attuatori, elaborare dati sui nodi e spedire messaggi ad altri dispositivi tramite comunicazione radio. Programmare applicazioni per WSN è un lavoro complesso che richiede paradigmi e tecnologie adeguati in grado di sostenere le caratteristiche di tali reti che esigono di integrare assieme tecniche di sensing, computazione e comunicazione distribuite.

Questa tesi si propone di fornire nuovi paradigmi per supportare lo sviluppo di applicazioni per WSN sia tramite un approccio domain-specific, che tramite un approccio general-purpose. In particolare, questa tesi proporrà tre contributi principali. Il primo riguarda l'analisi, la progettazione e la realizzazione di un framework domain-specific per WSN eterogenee per la gestione efficiente, flessibile e distribuita di edifici, chiamato Building Management Framework (BMF). BMF offre la possibilità di riconfigurare dinamicamente una WSN e di eseguire algoritmi di elaborazione direttamente sui nodi, supporta attuatori, reti multi-hop e multi-platform, fornisce un'astrazione di programmazione appositamente concepita per catturare la morfologia degli edifici e una interfaccia facilmente estensibile. Il secondo contributo si riferisce alla progettazione, analisi e realizzazione di un sistema general-purpose ad agenti mobili per WSN, chiamato MAPS (Multi Agent Platform for SunSPOT). Tale sistema permette lo sviluppo efficace di agenti e di applicazioni basate su agenti Java. MAPS integra i paradigmi di programmazione agent oriented, event-driven e state-based. Infine, il terzo contributo consiste nella progettazione, analisi e realizzazione di un framework domain-specific per la prototipazione rapida di applicazioni per Wireless Body Sensor Network (WBSN), chiamato SPINE2 (signal processing in-node environment version 2). SPINE2 mira a supportare lo sviluppo di applicazioni innalzando il livello delle astrazioni di programmazione fornendone un modello task-oriented.

I have learned to work hard.

I have grown as an individual.

Thanks to those who have supported me in this process.

Contents

List of Figures

List of Tables

1

Motivation, Objectives and Organization of the Thesis

1.1 Motivation

Due to recent advances in electronics and communication technologies, Wireless Sensor Networks (WSNs) have been introduced and are currently emerging as a disruptive technology enabling and supporting next generation ubiquitous and pervasive computing scenarios. WSNs have a high potential to support a variety of high-impact applications such as disaster / crime prevention and military applications, environmental applications, health-care applications and smart spaces. However, programming WSNs is a complex task due to the limited capabilities (processing, memory and transmission range) and energy resources of each sensor node as well as the lack of reliability of the radio channel. Moreover, WSN programming is usually application-specific (or more generally domain-specific) and requires tradeoffs in terms of task complexity, resource usage and communication patterns.

A problem of the WSN development is the lack of methodologies and programming tools to support a fast prototyping of applications on top of them. So, designing and programming applications for such networks is tedious and error prone because the need to integrate together in an application: network logics, signal processing algorithms, sensor and actuator drivers and logics, always paying attention to the limited resources of the nodes.

Thus, to support rapid development and deployment of WSN applications flexible, WSN-aware programming paradigms are needed. They can allow reducing design time through modularity and reuse while offering solutions that are optimized for the target domain.

1.2 Objectives of the Thesis

The aim of the proposed thesis is the definition of high level frameworks and tools dealing with the above mentioned issues and supporting rapid development of WSN applications. In particular:

- The first contribution is related to the requirements analysis, the design, the implementation and experimentation of a Building Management Framework (BMF). The BMF is a domain-specific framework implemented for both heterogeneous Wireless Sensor and Actuator Networks (WSAN) nodes and more capable base station (PC, plug computer, smartphone, PDA, etc.) for flexible and efficient distributed sensing and actuation in buildings. BMF addresses specific building requirements that are not addressed in general-purpose frameworks for WSNs. In particular, BMF provides fast WSAN reconfiguration, in-node processing algorithms, multi hop networks and multi-platform support, a building programming abstraction to dynamically catch the morphology of buildings, actuators support and an extensible human computer interface. The main application of the BMF is the building indoor management and, in particular: the energy monitoring, analysing data coming from the sensors to understand the energy spent in a building; the behavioral monitoring, understanding the behavior of people in building; the space monitoring, understanding the use of the spaces in building; intelligent actuation, using actuators to achieve specific aims (e.g. energy saving, maximization of the comfort in the building).

- The second contribution refers to the design, implementation and experimentation of MAPS (Mobile Agent Platform for Sun SPOT), an innovative Java-based framework for wireless sensor networks based on Sun SPOT technology. MAPS enables an effective Java-based development of agents and agent-based applications by integrating agent oriented, event-driven and state-based programming paradigms. Performance evaluation of MAPS has been carried out to evaluate not only MAPS per se but also the degree of maturity of the Sun SPOT technology for supporting (mobile) agent-based applications and systems. Finally, MAPS has been used for the development of several real case studied in the health-care and building management domains.

- The third contribution regards the definition of the requirements, the design, the implementation and experimentation of SPINE2, an evolution of the SPINE (Signal Processing In-Node Environment) framework based on the C-language, which aims at supporting the development of sensor platform-independent WSN applications. The goal of SPINE2 is to reach a very high platform independency for C-like programmable sensor platforms (e.g. TinyOS, Ember, ZStack) and raise the level of the provided programming abstractions from platform-specific to platform-independent. SPINE2 offers a task-oriented model for programming the sensor nodes of a collaborative WSN. In particular tasks can be dynamically discovered, created, activated, scheduled and controlled by the coordinator on each sensor node in order to fulfill a goal-directed overall task of the distributed system implemented by the network of sensor nodes.

1.3 Organization of the Thesis

This thesis is organized as follows:

- Chapter 2 defines WSNs illustrating the hardware used, the most common communication protocols, operating systems and programming languages. Moreover, the chapter introduces the principal existing frameworks and middlewares and shows some existing applications for WSNs.

- Chapter 3 lists the main requirements for a building management system, introduces the Building Management Framework (BMF) describing its main components and processing levels and compares BMF with some related work. Moreover, the chapter presents the SmartEnLab case study and discusses an off-line energy data analysis, a performance evaluation of the BMF-based building wireless sensor networks, and an estimation of the duration of a WSAN running the BMF.

- Chapter 4 explains the deep correlation between mobile agents and WSNs, presents the requirements, architecture, agent programming model and implementation of MAPS and compares it with some related work. Moreover, the chapter shows the performance evaluation of MAPS carried out through micro-benchmarks and illustrates a real case study developed through MAPS for the real-time monitoring in smart buildings.

- Chapter 5 analyzes the evolution of the SPINE framework and compares it with some related work. Moreover, the chapter categorizes and discusses interesting approaches which can effectively support platform-independent development of WSN applications and shows the current efforts towards the definition of SPINE2 which aims at supporting the development of platform-independent WSN applications. Finally, the chapter presents a case study that tests the effectiveness of the SPINE2 framework.

- Finally, Chapter 6 presents a summary of the main results of this thesis, along with some concluding remarks. Afterwards, a list of the publications related to the thesis, and possible future research works that can derive from the work here presented are shown.

2

Background: State-of-the-art of WSNs

Wireless sensor networks (WSNs) [1] have gained worldwide attention in recent years, particularly with the proliferation in Micro-Electro-Mechanical Systems (MEMS) [2] technology which has facilitated the development of smart sensors. These sensors are small, with limited processing and computing resources, and they are inexpensive compared to traditional sensors. These sensor nodes can sense, measure, and gather information from the environment and, based on some local decision process, they can transmit the sensed data to the user.

Smart sensor nodes (also called motes) are low power devices equipped with one or more sensors, a processor, memory, a power supply, a radio, and an actuator. A variety of mechanical, thermal, biological, chemical, optical, and magnetic sensors may be attached to the sensor node to measure properties of the environment. Since the sensor nodes have limited memory and are typically deployed in difficult-to-access locations, a radio is implemented for wireless communication to transfer the data to a base station (e.g., a laptop, a personal handheld device, or an access point to a fixed infrastructure).

WSNs have great potential for many applications in scenarios such as military target tracking and surveillance, natural disaster relief, biomedical health monitoring, and hazardous environment exploration and seismic sensing. In military target tracking and surveillance, a WSN can assist in intrusion detection and identification [3]. Specific examples include spatially-correlated and coordinated troop and tank movements. With natural disasters, sensor nodes can sense and detect the environment to forecast disasters before they occur. In biomedical applications, surgical implants of sensors can help monitor a patient's health [4]. For seismic sensing, ad hoc deployment of sensors along the volcanic area can detect the development of earthquakes and eruptions [5].

Unlike traditional networks, a WSN has its own design and resource constraints. Resource constraints include a limited amount of energy, short communication range, low bandwidth, and limited processing and storage in each node. Design constraints are application dependent and are based on the mon-

itored environment. The environment plays a key role in determining the size of the network, the deployment scheme, and the network topology. The size of the network varies with the monitored environment.

Research in WSNs aims to meet the above constraints by introducing new design concepts, creating or improving existing protocols, building new applications, and developing new algorithms.

In this chapter, an overview, that is not intended to be exhaustive, on WSNs is given. In particular, Section 2.1 analyzes the main components of a sensor node, Section 2.2 shows some of the available communication standards, Section 2.3 points out the most important operating systems for WSN, Section 2.4 shows a classification of Middlewares/Frameworks to support the development, maintenance, deployment and execution of applications over WSN. Finally, Section 2.5 gives an overview on applications for WSN.

2.1 Hardware

A sensor node is a node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network.

The main components of a sensor node are microcontroller, transceiver, external memory, power source and one or more sensors (Figure 2.1).

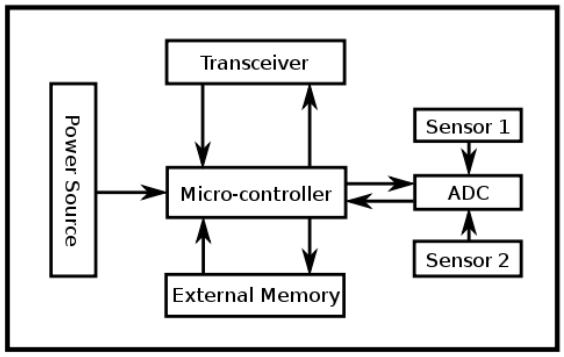


Fig. 2.1. The typical architecture of a sensor node.

2.1.1 Microcontroller

The microcontroller performs tasks, processes data and controls the functionality of other components in the sensor node. Most of the microcontroller used in WSNs are produced by Atmel [6], Cypress [7], and Texas Instruments [8].

Alternatives that can be used as a controller are general purpose desktop microprocessors, digital signal processors, FPGAs and ASICs. Anyway, microcontrollers are used in most of embedded systems such as sensor nodes because of its low cost, flexibility to connect to other devices, ease of programming, and low power consumption. A general purpose microprocessor generally has a higher power consumption than a microcontroller, therefore it is often not considered a suitable choice for a sensor node.

2.1.2 Transceiver

Sensor nodes often make use of ISM (industrial, scientific and medical) band which gives free radio, spectrum allocation and global availability. The possible choices of wireless transmission media are Radio frequency (RF), Optical communication (Laser) and Infrared. Radio frequency based communication is the most relevant that fits most of the WSN applications. WSNs tend to use license-free communication frequencies: 173 MHz, 433 MHz, 868 MHz, 915 MHz, and 2.4 GHz. The functionality of both transmitter and receiver are combined into a single device known as transceivers.

2.1.3 External memory

From an energy perspective, the most relevant kind of memory for a WSN node is the on-chip memory of a microcontroller. However, many sensor nodes are equipped with specific flash memories. Flash memories are used due to their cost and storage capacity.

2.1.4 Power source

The sensor node consumes power for sensing, communicating and data processing. More energy is required for data communication than any other process. Power is stored either in batteries or capacitors. Batteries, both rechargeable and non-rechargeable, are the main source of power supply for sensor nodes. Current studies are creating sensors that are able to renew their energy by ambient energy harvesting. The main sources of ambient energy considered suitable for use with WSNs are solar, mechanical (vibration or strain) and thermal energy [9].

2.1.5 Sensors

Sensors are hardware devices that produce a measurable response to a change in a physical condition like temperature or pressure. Sensors measure physical data of the parameter to be monitored. The continual analog signal produced by the sensors is digitized by an analog-to-digital converter and sent to controllers for further processing. Sensors are classified into three categories:

- passive, omni-directional sensors;
- passive, narrow-beam sensors;
- active sensors.

Passive sensors sense the data without actually manipulating the environment by active probing. They are self powered; that is, energy is needed only to amplify their analog signal. Active sensors actively probe the environment, for example, a sonar or radar sensor, and they require continuous energy from a power source. Narrow-beam sensors have a well-defined notion of direction of measurement, similar to a camera. Omni-directional sensors have no notion of direction involved in their measurements.

2.2 Communication

Wireless sensor networks communication standards have been developed with the key design requirement for low power consumption. The standard defines the functions and protocols necessary for sensor nodes to interface with a variety of networks. Some of these standards include IEEE 802.15.4 [10], ZigBee [11], WirelessHART [12], ISA100.11 [13], IETF 6LoWPAN [14], IEEE 802.15.3 [15]. The following subsections describe these standards in more detail.

2.2.1 IEEE 802.15.4

IEEE 802.15.4 [10] is the proposed standard for low rate wireless personal area networks (LRWPANs). IEEE 802.15.4 focuses on low cost of deployment, low complexity, and low power consumption. IEEE 802.15.4 is designed for wireless sensor applications that require short range communication to maximize battery life. The standard allows the formation of the star and peer-to-peer topology for communication between network devices. Devices in the star topology communicate with a central controller while in the peer-to-peer topology ad hoc and self-configuring networks can be formed.

The physical layer supports 868/915 MHz low bands and 2.4 GHz high bands.

Wireless sensor applications using IEEE 802.15.4 include residential, industrial, and environment monitoring, control and automation.

2.2.2 ZigBee

ZigBee [11] defines the higher layer communication protocols built on the IEEE 802.15.4 standards for LR-PANs. ZigBee is a simple, low cost, and low power wireless communication technology used in embedded applications. ZigBee devices can form mesh networks connecting hundreds to thousands of

devices together. ZigBee devices use very little power and can operate on a cell battery for many years. The ZigBee standard was publicly available on June 2005.

2.2.3 WirelessHART

The WirelessHART [12] standard provides a wireless network communication protocol for process measurement and control applications. The standard is based on IEEE 802.15.4 for low power 2.4 GHz operation. WirelessHART is compatible with all existing devices, tools, and systems. WirelessHART standards were released to the industry in September 2007.

2.2.4 ISA100.11a

ISA100.11a [13] standard is designed for low data rate wireless monitoring and process automation applications. It defines the specifications for the OSI layer, security, and system management. The standard focuses on low energy consumption, scalability, infrastructure, robustness, and interoperability with other wireless devices. ISA100.11a networks use only 2.4 GHz radio and channel hopping to increase reliability and minimize interference.

2.2.5 6LoWPAN

IPv6-based Low power Wireless Personal Area Networks [14] enables IPv6 packets communication over an IEEE 802.15.4 based network. Low power device can communicate directly with IP devices using IPbased protocols. Using 6LoWPAN, low power devices have all the benefits of IP communication and management. 6LoWPAN standard provides an adaptation layer, new packet format, and address management. Because IPv6 packet sizes are much larger than the frame size of IEEE 802.15.4, an adaptation layer is used. The adaptation layer carries out the functionality for header compression. With header compression, smaller packets are created to fit into an IEEE 802.15.4 frame size. Address management mechanism handles the forming of device addresses for communication. 6LoWPAN is designed for applications with low data rate devices that requires Internet communication.

2.2.6 IEEE 802.15.3

IEEE 802.15.3 [15] is a physical and MAC layer standard for high data rate WPAN. It is designed to support real-time multi-media streaming of video and music. IEEE 802.15.3 operates on a 2.4 GHz radio and has data rates starting from 11 Mbps to 55 Mbps.

2.3 Operating systems

As already stated, sensor nodes are generally low-cost, resources constrained devices with limitations in memory size and computational capability; all these restricted characteristics have to be considered when designing an application and mainly when designing an operating system. In the following, the most important operating systems for WSNs are reported with a brief description of their features. A more complete survey about operating systems for WSNs can be found in [16] and [17].

2.3.1 TinyOS

TinyOS [18] [19] is an open source, flexible, component based, and application-specific operating system designed for sensor networks. TinyOS can support concurrent programs with very low memory requirements. The OS has a footprint that fits in 400 bytes. The TinyOS component library includes network protocols, distributed services, sensor drivers, and data acquisition tools.

TinyOS uses the component model and, according to the requirements of an application, different components are glued together with the scheduler to compose a static image that runs on the mote. A component is an independent computational entity that exposes one or more interfaces. Components have three computational abstractions: commands, events, and tasks. Mechanisms for inter-component communication are commands and events. Tasks are used to express intra-component concurrency. A command is a request to perform some service, while the event signals the completion of the service. TinyOS provides a single shared stack and there is no separation between kernel space and user space. Figure 2.2 shows a simplified version of the TinyOS architecture.

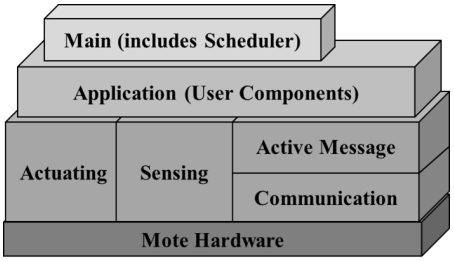


Fig. 2.2. Simplified TinyOS Architecture.

2.3.2 Contiki

Contiki [20], is a lightweight open source OS written in C for WSN sensor nodes. Contiki is a highly portable OS and it is build around an event-driven

kernel. Contiki provides preemptive multitasking that can be used at the individual process level. A typical Contiki configuration consumes 2 kilobytes of RAM and 40 kilobytes of ROM. A full Contiki installation includes features like: multitasking kernel, preemptive multithreading, proto-threads, TCP/IP networking, IPv6, a Graphical User Interface, a web browser, a personal web server, a simple telnet client, a screensaver, and virtual network computing.

The Contiki OS follows a modular architecture. At the kernel level it follows the event driven model, but it provides optional threading facilities to individual processes. The Contiki kernel comprises of a lightweight event scheduler that dispatches events to running processes. Process execution is triggered by events dispatched by the kernel to the processes or by a polling mechanism. The polling mechanism is used to avoid race conditions. Any scheduled event will run to completion, however, event handlers can use internal mechanisms for preemption.

Two kinds of events are supported by Contiki OS: asynchronous and synchronous events. The difference between the two is that synchronous events are dispatched immediately to the target process that causes it to be scheduled. On the other hand asynchronous events are more like deferred procedure calls that are en-queued and dispatched later to the target process.

The polling mechanism used in Contiki can be seen as high-priority events that are scheduled in between each asynchronous event. When a poll is scheduled, all processes that implement a poll handler are called in order of their priority.

All OS facilities e.g., sensor data handling, communication, and device drivers are provided in the form of services. Each service has its interface and implementation. Applications using a particular service need to know the service interface. An application is not concerned about the implementation of a service. Figure 2.3 shows the block diagram of the Contiki OS architecture, as given in [21].

2.3.3 MANTIS

The MultimodAl system for NeTworks of In-situ wireless Sensors (MANTIS) provides a new multithreaded operating system for WSNs. MANTIS is a lightweight and energy efficient operating system. It has a footprint of 500 bytes, which includes kernel, scheduler, and network stack. The MANTIS Operating System (MOS) key feature is that it is portable across multiple platforms, i.e., MOS applications can be tested on a PDA or a PC [22]. Afterwards, the application can be ported to the sensor node. MOS also supports remote management of sensor nodes through dynamic programming. MOS is written in C and it supports application development in C.

MOS follows a layered architectural design as shown in Figure 2.4. Services provided by the OS are implemented in layers. Each layer acts as an enhanced virtual machine to the layers above.

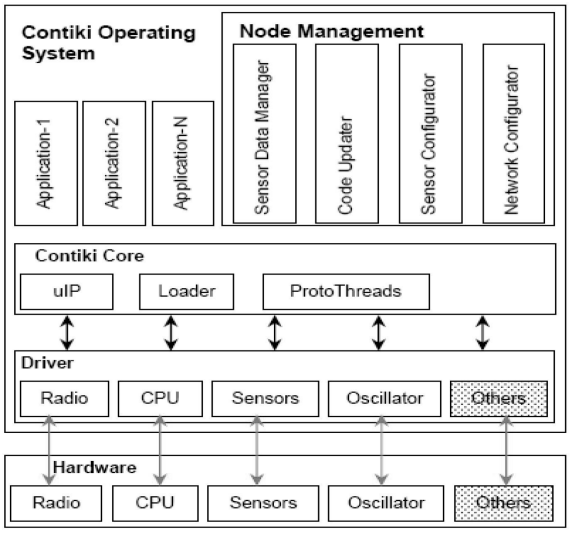


Fig. 2.3. Contiki Architecture.

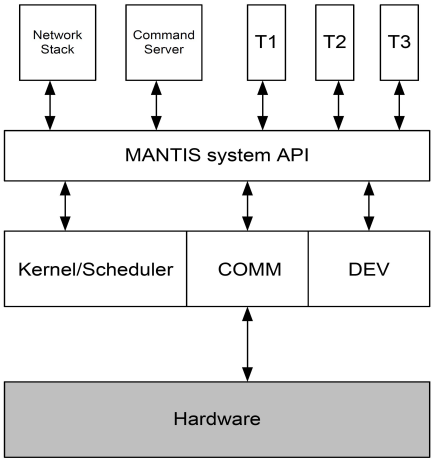


Fig. 2.4. MANTIS OS Architecture.

2.3.4 Nano-RK

Nano-RK [23] is a fixed, preemptive multitasking real-time OS for WSNs. The design goals for Nano-RK are multitasking, support for multi-hop networking, support for priority-based scheduling, timeliness and schedulability, extended WSN lifetime, application resource usage limits, and small footprint. Nano-RK uses 2 Kb of RAM and 18 Kb of ROM. Nano-RK provides support for CPU, sensors, and network bandwidth reservations. Nano-RK supports hard and

soft real-time applications by the means of different real-time scheduling algorithms, e.g., rate monotonic scheduling and rate harmonized scheduling [24].

Nano-RK follows the monolithic kernel architecture model. Due to its real-time nature, the authors of Nano-RK emphasis the use of a static design time framework i.e., task priorities, deadlines, period, and their reservations should be assigned offline, so that admission control procedures can be applied efficiently. By choosing this static approach, one can determine whether the task deadlines can be met in the overall system design or not. Application programmers can change different parameters (deadline, period, CPU reservation, and network bandwidth reservation) associated with the tasks to arrive at a configuration that meets the overall objectives. Nano-RK also provides APIs through which task parameters can be configured at run-time, but its use is discouraged, especially when a task represents hard real-time jobs. Figure 2.5 shows the Nano-RK architecture.

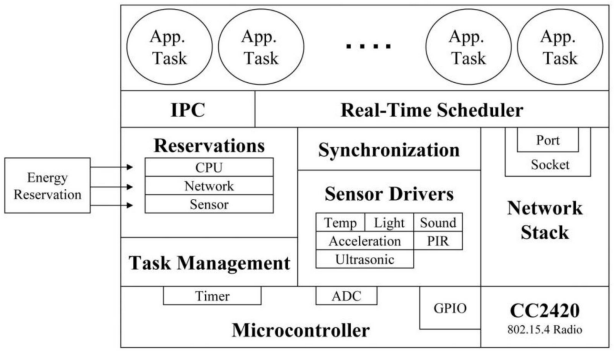


Fig. 2.5. Nano-RK Architecture.

2.3.5 LiteOS

LiteOS [25] is a Unix-like operating system designed for WSNs at the University of Illinois at Urbana-Champaign. The motivations behind the design of LiteOS are to provide a Unix-like OS for WSN, provide system programmers with a familiar programming paradigm (thread-based programming mode, although it provides support to register event handlers using callbacks), a hierarchical file system, support for object-oriented programming in the form of LiteC++, and a Unix-like shell. The footprint of LiteOS is small enough to run on nodes having an 8 MHz CPU, 128 bytes of program flash, and 4 Kbytes of RAM.

LiteOS follows a modular architecture design. LiteOS is partitioned into three subsystems: LiteShell, LiteFS, and the Kernel. LiteShell is a Unix-like

shell that provides support for shell commands for file management, process management, debugging, and devices. LiteFS mounts all neighboring sensor nodes as a file. LiteFS mounts a sensor network as a directory and then lists all one hop sensor nodes as a file. The LiteOS kernel provides concurrency in the form of multithreading, provides support for dynamic loading, uses round robin and priority scheduling, allows programmers to register event handlers through callback functions, and provides synchronization support. Figure 2.6 shows the architecture of LiteOS.

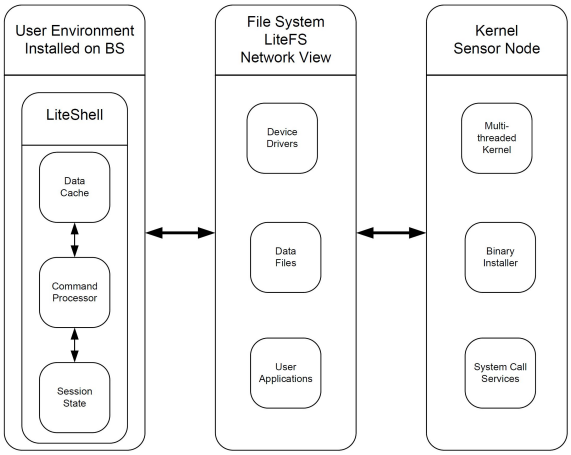


Fig. 2.6. LiteOS Architecture.

2.4 Frameworks and Middleware

Wireless Sensor Networks have found more and more applications in a variety of pervasive computing environments. However, how to support the development, maintenance, deployment and execution of applications over WSNs remains to be a nontrivial and challenging task, mainly because of the gap between the high level requirements from pervasive computing applications and the underlying operation of WSNs. Middlewares and Frameworks for WSN can help bridge the gap and remove impediments. In recent years, research has been carried out to study these instruments from different aspects and for different purposes. Most of the proposed solutions fit into one of the following categories:

- *virtual machines approaches*;
- *databases approaches*;

- *agents approaches*;
- *application driven (domain specific) approaches*;
- *message-oriented approaches*.

2.4.1 Virtual Machines approaches

Virtual machines (VMs) are an approach useful to virtualize real hardware, intermediate program representation or bytecode interpretation. Several works are developed using virtual machine approach.

Maté [26] is probably the first virtual machine for resource constrained sensor nodes. Its stack-based bytecode interpreter was introduced as an alternative way to program and reprogram WSNs based on TinyOS. As Maté uses a specific bytecode (named tinyscript), that is dense and concise, it means that complex programs could be written in a few lines of tinyscript, saving resources like memory and energy, especially during transmission over the air. In order to reprogram (infect) the network, Maté implements a viral program mechanism.

MagnetOS [27] is a distributed operating system for sensor and ad hoc networks that abstracts the whole network as a single, unified Java Virtual Machine, exposing a runtime environment for java programs. MagnetOS applications are specified as regular Java programs. An application partitioning tool takes the monolithic Java application and converts it into components. Then the components are automatically and transparently distributed through the network, based on components communication/interaction, to reduce energy consumption and increase network longevity. The components use Java Remote Method Invocation (JRMI) for inter-component communication. As MagnetOS needs Java heavy mechanisms, like JRMI, it targets devices with better resources than the standard motes.

Squawk VM [28] [29] is a small Java virtual machine written mostly in Java that runs without an operating system on a wireless sensor platform. Squawk translates standard class file into an internal pre-linked, position independent format that is compact and allows for efficient execution of bytecodes that have been placed into a read-only memory. In addition, Squawk implements an application isolation mechanism whereby applications are represented as object and are therefore treated as first class objects (i.e., they can be reified). Application isolation also enables Squawk to run multiple applications at once with all immutable state being shared between the applications. Mutable state is not shared. The combination of these features reduce the memory footprint of the VM, making it ideal for deployment on small devices. Squawk provides a wireless API that allows developers to write applications for WSNs. Authentication of deployed files on the wireless device and migration of applications between devices is also performed by the VM. The Squawk VM

was specifically designed for the Sun Small Programmable Object Technology (SunSPOT) wireless device, a device developed at Sun Microsystems Laboratories for experimentation with wireless sensor and actuator applications.

2.4.2 Databases Approaches

The database programming support virtualizes the WSNs as a distributed database, where user can use SQL like languages to interact with the network. Examples of solutions that adopt this approach are COUGAR [30], SINA [31], TinyDB [32] and SwissQM [33]. Some of the work in this area has been on pure sensor database systems, which essentially provide a distributed database solution appropriate for resource-constrained sensor networks, focusing on efficient query routing and processing. COUGAR, TinyDB and SwissQM fall into this category. SINA differs in that it uses an SQL-like language for expressing queries, but also provides other functions which are outside the scope of traditional database systems.

A functional comparison of TinyDB and SwissQM with reference to the Building domain is explained in Section 3.3.

2.4.3 Agents Approaches

The agent approach promotes the modular programming in order to facilitate the distribution of code through mobile agents.

Due to the currently available resource-constrained sensor nodes and related operating systems, building flexible and efficient agent systems for WSNs is a very complex task. Very few systems for WSNs have to date been proposed and actually implemented. The most significant ones are SensorWare [34], Agilla [35] and actorNet [36]

The mentioned works are described in Section 4.3 and compared to a novel Java-based Agent framework for wireless sensor networks, that is presented in chapter 4.

2.4.4 Application Driven (Domain Specific) Approaches

Application driven is another programming support approach that focuses on tuning the network in order to support the application requirements given by a domain specific domain.

Novel domain specific frameworks are presented in the following. In particular, a domain specific framework for Building domain is presented in chapter 3 while a domain specific framework for wireless body sensor networks is described in chapter 5.

2.4.5 Message-Oriented Approaches

The message-oriented approach constitutes a communication model to facilitate the message passing between nodes and the sink nodes. This approach applies the publish-subscribe mechanism to a distributed sensor network. It can support asynchronous communication by allowing a loose coupling between the sender and the receiver. This approach is particularly suitable in pervasive environments where most applications are based on events.

Mires [37] proposes an asynchronous communication model that is suitable for WSN applications where event driven environments are quite common. Mires incorporates characteristics of message-oriented approach by allowing applications communicate in a publish/subscribe way.

2.5 Applications

With reference to the survey from Yick, Mukherjee and Ghosal [38], WSN applications can be classified into two categories: monitoring and tracking (see Figure 2.7). Monitoring applications include indoor/outdoor environmental monitoring, health and wellness monitoring, power monitoring, inventory location monitoring, factory and process automation, and seismic and structural monitoring. Tracking applications include tracking objects, animals, humans, and vehicles. While there are many different applications, below a few example applications that have been deployed and tested in the real environment are described.

PinPtr [3] is an experimental counter-sniper system developed to detect and locate shooters. The system utilizes a dense deployment of sensors to detect and measure the time of arrival of muzzle blasts and shock waves from a shot. Sensors route their measurements to a base station (e.g., a laptop or PDA) to compute the shooter's location.

Macroscope of redwood [39] is a case study of a WSN that monitors and records the redwood trees in Sonoma, California. Each sensor node measures air temperature, relative humidity, and photo-synthetically-active solar radiation. Sensor nodes are placed at different heights of the tree. Plant biologists track changes of spatial gradients in the microclimate around a redwood tree and validate their biological theories.

Underwater monitoring study in [40] developed a platform for underwater sensor networks to be used for longterm monitoring of coral reefs and fisheries. The sensor network consists of static and mobile underwater sensor nodes. The nodes communicate via point-to-point links using high speed optical communications.

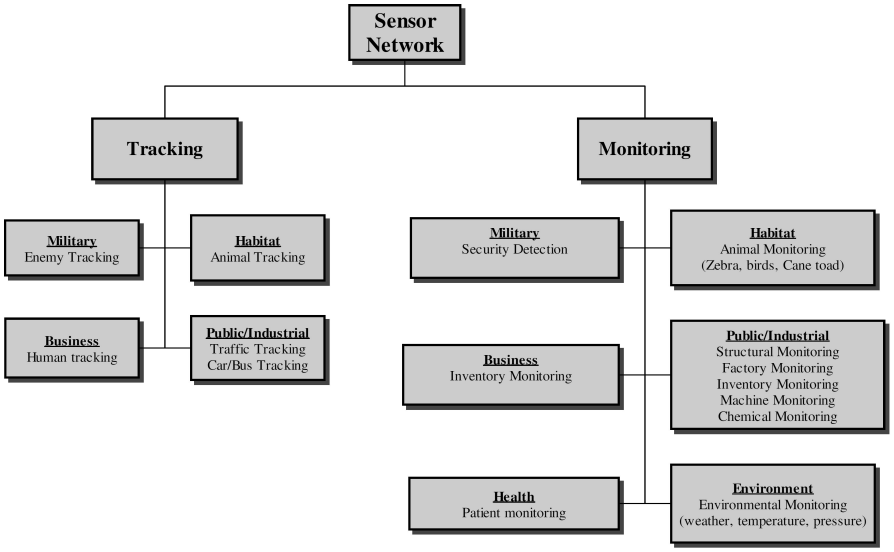


Fig. 2.7. Overview of sensor applications.

Cyclops [41] is a small camera device that bridges the gap between computationally-constrained sensor nodes and complimentary metal-oxide semiconductor (CMOS) imagers. This work provides sensor technology with CMOS imaging. With CMOS imaging, humans can (1) exploit a different perspective of the physical world which cannot be seen by human vision, and (2) identify their importance. Cyclops attempts to interface between a camera module and a lightweight sensor node. Cyclops contains programmable logic and memory circuits with high speed data transfer. It contains a microcontroller to interface with the outside world. Cyclops is useful in a number of applications that require high speed processing or high resolution images.

Volcanic monitoring [5] with WSN can help accelerate the deployment, installation, and maintenance process. WSN equipments are smaller, lighter, and consume less power. The challenges of a WSN application for volcanic data collection include reliable event detection, efficient data collection, high data rates, and sparse deployment of nodes. Given these challenges, a network consisting of 16 sensor nodes was deployed on Volcan Reventador in northern Ecuador. Each sensor node was equipped with an external omnidirectional antenna, a seismometer, a microphone, and a custom hardware interface board. 14 sensor nodes were equipped with a single axis Geospace Industrial GS-11 Geophone with corner frequency of 4.5 Hz while the other two sensor nodes carried triaxial Geospace Industries GS-1 seismometers with corner frequencies of 1 Hz. Sensor nodes are placed approximately 200400 m apart from each other. Nodes relay data via multi-hop routing to a gateway

node. The gateway node connected to a long-distance FreeWave radio modem transmits the collected data to the base station. During network operation, each sensor node samples two or four channels of seismo-acoustic data at 100 Hz. The data is stored in local flash memory. When an interesting event occurs, the node will route a message to the base station. If multiple nodes report the same event, then data is collected from the nodes in a round-robin fashion. When data collection is completed, the nodes return to sampling and storing sensor data locally. study.

Health monitoring applications [42] using WSN can improve the existing health care and patient monitoring. Many prototypes have been developed for applications such as infant monitoring, alerting the deaf, blood pressure monitoring and tracking, and fire-fighter vital sign monitoring. In the following some examples are reported.

Because many infants die from sudden infant death syndrome (SIDS) each year, *Sleep Safe* [43] is designed for monitoring infants while they sleep. It detects the sleeping position of an infant and alerts the parent when the infant is lying on its stomach. Sleep Safe consists of two sensor motes. One node is attached to an infant's clothing and a node is connected to base station computer. The former node has a three-axis accelerometer for sensing the infant's position relative to gravity. This node periodically sends packets to the base station for processing. Based on the size of the sensing window and the threshold set by the user, the data is processed to determine if the infant is on his back.

Baby Glove [43] prototype is designed to monitor vitals. Baby Glove is a swaddling baby wrap with sensors that can monitor an infant's temperature, hydration, and pulse rate. A mote is connected to the swaddling wrap to transmit the data to the base station. Like Sleep Safe, an alert is sent to the parent if the analyzed data exceeds the health settings.

FireLine [42] is a wireless heart rate sensing system. It is used to monitor a fire fighter's heart rate in real-time to detect any abnormality and stress. FireLine consists of a mote, a custom made heart rate sensor board, and three re-usable electrodes. All these components are embedded into a shirt that a fire fighter will wear underneath all his protective gears. The readings are taken from the mote and then transferred to the base station. If the fire fighter's heart rate is increasing too high, an alert is sent.

Heart@Home [42] is a wireless blood pressure monitor and tracking system. Heart@Home uses a mote located inside a wrist cuff which is connected to a pressure sensor. A user's blood pressure and heart rate is computed using the oscillometric method. The mote records the reading and sends it to a basestation. A software application processes the data and provides a graph of the user's blood pressure and heart rate over time.

LISTSENse [42] enables the hearing impaired to be informed of the audible information in their environment. A user carries the base station with him.

The base station consists of a vibrator and LEDs. Transmitter motes are place near objects (e.g., smoke alarm and doorbell) that can be heard. Transmitter motes consist of an omni-directional condenser microphone. They periodically sample the microphone signal at a rate of 20 Hz. If the signal is greater than the reference signal, an encrypted activation message is sent to the user. The base station receiving the message actives the vibrator and its LED lights to warn the user. The user must press the acknowledge button to deactivate the alert.

Finally, among tracking applications of animals, *ZebraNet* [44] system is a mobile wireless sensor network used to track animal migrations. ZebraNet is composed of sensor nodes built into the zebra's collar. The node consists of a 16-bit TI microcontroller, 4 Mbits off-chip flash memory, a 900 MHz radio, and a GPS unit. Positional readings are taking using the GPS and sent multi-hop across zebras to the base station. The goal is to accurately log each zebra's position and use them for analysis. A total of 610 zebra collars were deployed at the Sweetwaters game reserve in central Kenya. A set of movement data was collected during this study. From the data, the biologists can better understand the zebra movements during the day and night.

3

Building Management Framework

Future buildings will be constantly monitored and managed through intelligent systems that allow having information about the building health, keeping a good comfort level for the building inhabitants and optimizing the energy spent. Despite many WSN programming frameworks have been to date developed and, in some cases, applied to support monitoring of buildings, none of them possesses all the specific features needed to develop WSN-based building applications. In this chapter a multi-platform domain specific framework based on Wireless Sensor and Actuator Networks (WSANs) for enabling efficient and effective management of buildings is presented. The proposed Building Management Framework (BMF) provides powerful abstractions that capture the morphology of buildings to allow for the rapid development and flexible management of pervasive building monitoring applications. The functionalities of the framework are shown in an emblematic case study concerning the SmartEnergyLab that is an effective operating scenario related to the monitoring of the usage of workstations in laboratories and offices. Finally, a performance evaluation of a WSAN running the BMF in terms of bandwidth usage and system lifetime is shown.

3.1 Introduction

Nowadays, as buildings inhabitants are growing their awareness about technology, they expect that future buildings will be "smart" in supporting their need [45]. In particular, the inhabitants of a building would be sure about the structural health of their buildings, they would require the presence of mechanisms supporting building automation for increasing the comfort levels and monitoring of the building energy usage to automatically actuate energy saving strategies. This would be possible only with the pervasive and intelligent control of all the spaces in a building carried out through sensors and actuators deployed inside buildings and on the building structure.

To achieve this goal, the use of wireless sensor and actuator networks (WSANs) [46] to audit buildings and control equipment represents a viable and more flexible solution to traditional building monitoring and actuating systems (BMAS), which require retrofitting the whole building and therefore are difficult to implement in existing structures. In contrast, WSAN-based solutions for monitoring buildings and controlling equipment, such as electrical devices, heating, ventilation and cooling (HVAC), can be installed in existing structures with minimal efforts. This should enable a more effective monitoring of building structure condition, and building space and energy (electricity, gas, water) usage while facilitating the design of techniques for intelligent actuation of devices in buildings. In order to achieve this, WSAN-based building auditing necessitates devising a dedicated management framework for: (1) the management of a range of cooperating networked entities in the different parts of the structure; (2) the capture of the morphology of the buildings to correlate sensed data to a portion of the building; (3) the adaptive management of sensing and actuation; (4) the management of networks communication to allow different duty cycle for nodes powered through the mains; (5) the low and high level programmability of the network; (6) the fast deployment of concurrent applications at runtime.

Several academic works in literature can be modified to work with such requirements. In fact, research proposals like TinyDB [32], SwissQM [33], Abstract Regions (AR) [47] and FiGaRo [48] can be used to monitor buildings, provide a simplified way to query data from the nodes, use in-network processing, and multi-hop support for large networks. However, existing frameworks have significant drawbacks and miss key features that hinder the applicability in a smart building context. In particular, they do not implement methods to capture the morphology of buildings, do not provide multi-platform support, and do not allow using actuators. In addition, they provide users with features like SQL-like query interpreters, in-network data fusion, and shared variables that in a Building Management domain could be unnecessary and charge the nodes with complex and onerous algorithms.

On the other hand, there are some specific industrial solutions for Building Management; such as the Crossbow Ecowizard System [49] or the EpiSensor SiCA for Building Management [50] that are fixed frameworks but lack of adaptability and advanced features such as data filtering and processing and, especially, they do not provide customizability of the system.

This chapter proposes BMF, Building Management Framework [51] [52], which is a domain-specific framework based on WSANs (Wireless Sensor and Actuator Networks) for enabling proactive monitoring of spaces and control of devices/equipments. The aim of the BMF is to overcome the limits of existing frameworks by providing: (i) flexible and efficient management of large sets of cooperating networked sensors and actuators; (ii) abstractions for logical and physical node group organization to specifically capture the morphology of buildings; (iii) intelligent sensing and actuation techniques; (iv) integration

of heterogeneous WSANs; (v) flexible system programming at low- and high-level; (vi) fast deployment of different applications through message exchange.

BMF is organized in two processing layers: Low-Level Processing (LLP) and High-Level Processing (HLP). LLP resides at sensor node level and provides sensor-based services, while HLP resides at the basestation level and provides application based services.

The contribution of the framework is twofold: through the HLP the BMF assists the application developer with an extensible set of OSGi Bundles [53] that facilitate the management of an heterogeneous WSAN spanned in buildings and the collection of data from the network. Through the LLP the BMF is a network practitioner in the sense that facilitates the deployment of heterogeneous nodes, management of network operations, and network maintenance. This is achieved by providing a set of embedded functionalities and a corresponding API at PC side, namely high-level processing. At the LLP side, BMF is currently implemented in TinyOS [19] specifically supporting the interoperation of several types of TelosB such as Crossbow, Moteiv, KMotes [54], as well as Tyndall [55] and Epic [56] platforms with heterogeneous sensing capabilities, and in Java (J2ME) for the SunSPOT [57] platform. The use of BMF is demonstrated through SmartEnLab, a BMF-based application for energy monitoring in computer laboratory environments. SmartEnLab is useful not only to highlight the advantages provided by BMF in terms of application programming effectiveness, deployment flexibility, run-time reconfiguration and system performance, but also shows the usefulness of WSAN in the context of energy profiling in office environments.

The rest of the chapter is organized as follows: section 3.2 introduces the main requirements for a building management system. Section 3.3 analyses related work to the BMF highlighting the unsuitability of some existing frameworks when used in the context of building monitoring. Section 3.4 introduces the BMF and describes its main components and processing levels. Section 3.5 presents the SmartEnLab application and discusses an off-line energy data analysis, a performance evaluation of the BMF-based building wireless sensor networks, and an estimation of the duration of a WSAN running the BMF.

3.2 Requirements

This section analyses all the requirements of a Building Management System (BMS) based on WSANs and used as comparison parameters:

1. *Fast Reconfiguration.* While some frameworks allow to quickly configure nodes in a WSN through packets sent over the air, some other works require the user to write embedded code for sensors that is difficult to realize and requires an expensive and slow deployment phase. We believe that a dynamic network requires tools for a fast (re)configuration to switch over

different applications at runtime. Moreover, the packets sent to configure the nodes have to be optimized to save communication bandwidth and carry out a fast reconfiguration. Moreover, an effective query language for node programming has to be defined for WSAN programming at run-time.

2. *In-node Processing.* Performing in-node processing allows computing and sending synthetic data on the network giving the opportunity to send less raw data packets so saving both bandwidth and energy on the nodes (as the radio is the sensor component that wastes more energy) and also allowing more nodes to join the network. In a Building WSAN, the in-node processing is more useful than the in-network processing (data fusion, shared variables, and so forth) which provides functionalities that are most of the time useless in a Building Management domain, charges the nodes with onerous algorithms and increases configuration data exchange among nodes.

3. *Multi-hop Support.* To cover whole buildings, a framework must provide support for multi-hop networks relying for example on specific data centric or hierarchical protocols [58].

4. *Multi-platform Support.* Using different platforms together in a network allows to take advantage of different types of sensor nodes with different and more suitable features (e.g. the use of sensor boards already developed for different platforms).

5. *Methods to dynamically capture the morphology of buildings (Building Programming Abstraction).* In a building, the nodes of a WSAN can be deployed everywhere. It is very important to design a Building Programming Abstraction that allows to store where the nodes are placed in a building in terms of room (e.g. kitchen, sitting room, and bathroom) and function (e.g.: ambient temperature, lighting system, and radiator monitoring). A Building Programming Abstraction allows to capture the morphology of a building at basestation side and allows to make a node conscious about its position in the building.

6. *Support for actuators.* In order to apply policies to achieve some goals like comfort or energy saving in a building, it's of primary importance for some nodes of the network the control of actuators. Through actuators, a user can remotely send commands to objects (appliances, lights, radiators) or a software can intelligently apply some rules.

7. *Deploy management through Human Computer Interface (HCI).* A user-friendly graphical interface to easily configure the network is fundamental. In particular, it allows saving programming time and effectively visualizing data from the WSANs. Moreover, the interface has to be extensible to allow users to add high-level functionalities without programming a new interface or re-implementing the one provided.

3.3 State-of-the-Art and Related Work

This section focuses on existing works discussing the applicability in the context of buildings. We first overview the main characteristics and functionalities of TinyDB [32], SwissQM [33], Abstract Regions (AR) [47] and FiGaRo [48], which are the four most interesting academic proposals related to BMF, and then compare them and BMF on the basis of the identified requirements.

TinyDB is defined as a distributed query processing system for extracting information from a network of sensors. TinyDB has many of the features of a traditional query processor (e.g.,the ability to select, join, project, and aggregate data), but also incorporates a number of other features designed to minimize power consumption via acquisitional techniques. SwissQM is intended as the "next generation architecture for data acquisition and data processing in sensor networks". It is a query executor with optimized performance and is based on a specialized virtual machine (specialized in data processing rather than a generic platform for application development) that runs optimized byte code rather than queries. AR is a programming model with the goal of simplifying the application design by providing a set of programming primitives for sensor networks that abstract the details of low-level communication, data sharing, and collective operations. Finally, FiGaRo is a programming model supported by an efficient run-time system and distributed protocols, collectively enabling a fine-grained control over what is being reconfigured, and where.

TinyDB and SwissQM give support to node reconfiguration at runtime. While AR and FiGaRo require users to write embedded code for the nodes and support an expensive configuration phase, TinyDB and SwissQM provide functionalities to set requests to nodes through queries. In particular, TinyDB uses long configuration packets (50 bytes) because nodes receive the whole query whereas SwissQM requests are shorter because a bytecode representing the query is sent to the nodes (36 bytes). In contrast, BMF avoids SQL-like queries and uses optimized request messages where configuration packets sent on the network are significantly shorter and optimized, between 9 and 22 bytes + 10 bytes of multi-hop protocol header.

TinyDB, SwissQM, AR and FiGaRo support multi-hop and give the possibility to use standard in-node processing algorithms on sampled data (e.g. average, variance, RMS).They provide in-network algorithms that are hardly used in a building context because they strongly increase exchange of packets (making node execution heavier) for the algorithm execution without having significant benefits. In particular, TinyDB implements an in-network processing optimized to merge data from different nodes when possible, SwissQM follows the idea used in TinyDB aggregating data during the route to the base station and AR provides in-network services like data sharing and data reduction where the reduction operator takes a shared variable key and an associative operator (such as sum, max, or min) and reduces the shared variable across nodes in the region, storing the result in a shared variable. These

features are useful for WSN applications that allow trading data precision and communication responsivity for a reduction of packet transmitted. However, in the context of building such features are less important while the framework should maximize data precision and responsiveness, for example to actuate appliances or report an anomaly somewhere in the building. In addition, system extensibility to several kinds of sensor platforms is key as sensors will likely be from several vendors and with different capabilities.

Among the considered works, only the BMF offers a multi-platform support. In particular, TinyDB only works on Mica motes with TinyOS 1.1, an old version of SwissQM can only use Mica2 motes with TinyOS 1.1 while a new one supports only TelosB with TinyOS 2.0.2, AR exclusively runs on Mica motes, FiGaRo was only tested on TelosB with the Contiki OS. BMF has been designed to be fully extensible and platform-independent; it already supports TelosB, Tyndall25 and Epic motes with TinyOS 2.1 and SunSPOT nodes with Java.

The provision of methods to dynamically capture the morphology of buildings is a limited, sometimes missing, feature in the works under examination. For example, TinyDB and SwissQM do not explicitly support Building Programming Abstractions. In AR the concept of Regions of nodes in a neighbourhood defined as radio hops and useful to exchange information among neighbours, is implemented. However, program the network taking into account the morphology of a building can be hardly achieved considering that radio range-based neighborhood. FiGaRo allows to define groups using boolean predicates over nodes attributes. Although this can be used to define a building morphology, in FiGaRo groups are defined statically before deployment and cannot be updated at runtime. This represents a significant drawbacks for building applications as sensor nodes are often relocated and belong to several groups (e.g. a node can be both in the group of heat monitoring and in the group of the corridor monitoring).

In contrast, BMF supports a Building Programming Abstraction to explicitly capture the morphology of a building based on high-level dynamic groups. Groups can be used to address the nodes through their logical or physical characteristics (place in which the node is positioned, type of sensor available on the node, appliance which the node is monitoring, and so forth). In such a way, many nodes can be simply addressed by the same packet and, consequently, there is no need to explicitly send the same request to many addressees. If this type of aggregation is dynamic and the network changes, a request will be transparently sent to the updated group of nodes.

Moreover, BMF is the only work that allows to control actuators such as ACme actuator nodes [56] or Tyndall REAM nodes [59].

Among the analysed works, only SwissQM and the BMF provide an interface allowing users not only to simply configure the WSAN (like TinyDB does) but also to add high-level functionalities without programming a new interface or re-implementing the one provided.

In conclusion, considering the analysed requirements together with the domain specific nature of BMF, the proposed framework results to be the best option to manage a Building WSAN while the related works show some limitations in the context of buildings as they have been originally designed with general-purpose features for WSN-based monitoring applications which do not specifically fit the considered context.

In Table 3.1, a comparison summary between the BMF and the analysed academic related work is reported.

Table 3.1. Comparison between the BMF and some academic related work.

	TinyDB	**SwissQM**	**BMF**	**Abstract Regions**	**FiGaRo**
Description	Query Processor for Sensor Networks	VM on nodes + gateway which parses queries written in different languages.	Application Level Framework	Programming Model (Abstraction Layer on TinyOS)	Programming Model
Fast Reconfiguration	YES by sending SQL-like queries over the air	YES by sending bytecode representing queries	YES by sending Request-4-Tuple packets	NO users have to write embedded code and deploy it in a configuration phase	NO users have to write embedded code and deploy it in a configuration phase
In-node processing	YES	YES	YES	YES	YES
Multi Hop Support	YES	YES	YES.	YES	YES
Multi-platform Support	NO only supported Mica motes with TinyOS 1.1	NO only supported Mica2 motes with TinyOS 1.1	YES fully support for TelosB, Tyndall25, Epic motes with TinyOS 2.1, and for SunSPOT	NO only supported Mica motes	NO tested only on TelosB with Contiki OS
Methods to dynamically capture the morphology of buildings (Building Programming Abstraction)	NO	NO	YES	NO	NO
Support for actuators	NO	NO	YES	Not explicitly supported	Not explicitly supported
Effective Human Computer Interface	NO Human Computer Interface provided, but it's not extensible	YES OSGi Bundles-based Human Computer Interface	YES OSGi Bundles-based Human Computer Interface	NO	N/A

With regards to industrial solutions for building management that can be correlated to BMF, the most representative are briefly described as follows:

- *The Arch Rock Energy Optimizer* [60] is an Energy Optimizer that provides easy-to-install wireless submetering hardware and a rich web application that helps users to tune and improve building energy usage. Arch Rock wireless sensing nodes enable the real-time monitoring of electricity, gas and water usage as well as ambient indoor or outdoor conditions such as temperature, humidity and light.

- *The Delta Dore Building Management Systems* [61] offers solutions suitable for any type of building. It addresses both service buildings (large office buildings, health establishments, medium and large stores, local governments, hotels, schools, leisure parks, museums) and industrial buildings (agri-food industry, aeronautics industry, quarries, laboratories, electronics industry). The Delta Dore monitoring systems are user-friendly, upgradeable, and provide comprehensive, real-time data. The systems regulate technical equipment while optimizing energy.

- *The Trend IQ Assured* [62] is Trend's commitment to provide building owners with the capability to manage their energy usage and environmental conditions - reducing energy wastage and associated cost, whilst maintaining optimum comfort conditions and maximising plant availability. Designed to deliver immediate benefits, IQ ASSURED is a structured program of optimising, upgrading and maintaining a Building Energy Management System. They provide an on-line software package that automatically collects and analyses utility meter readings and other data monitored. It is able to constantly compare actual and expected energy usage profiles and generate exception reports identifying incidences of energy waste. The system also allows clients to directly access energy performance data over Internet.

- *The Sentilla Energy Manager 3.0* [63] is a non-invasive, software-only solution for managing energy in data centers. By providing a holistic view, Sentilla Energy Manager bridges the IT and facilities gap, providing the world's first comprehensive view of where, when, and why energy is used in the data center.

- *The EpiSensor SiCA for Building Management* [50] system can be used to track electricity, water and natural gas usage in commercial buildings via the web. The system can also remotely control loads and monitor environmental parameters such as ambient temperature and humidity using easy to install wireless nodes. The data produced by these nodes can be used to improve building energy efficiency by identifying areas of wasteful energy usage. Using EpiSensor's SiCA software, wireless nodes can be remotely controlled and monitored from anywhere in the world via the web.

- *The Crossbow EcoWizard System* [49] enables users to analyse and monitor energy consumption in real-time using intuitive visualization tools allowing intelligent and efficient energy conservation. Using leading-edge wireless sensor networking technology, EcoWizard offers various energy measurement modes while significantly reducing the installation costs associated with current solutions. The developed data viewer provides simple graphical displays of past and present energy consumption (electricity,

temperature, etc.). The system uses sensors for electric power consumption, water, gas, temperature, humidity, light, etc. including pulse input wireless nodes enabling data acquisition from typical gas or water gauges.

Network reconfiguration is very limited for industrial solutions. While EcoWizard nodes send the values from the sensors once turned on, SiCA and BMF nodes can be fast reconfigured at runtime by specific queries which are respectively called "commands" and "Request 4-Tuple" (see Section 3.4 for details).

Both EcoWizard and the SiCA execute no data aggregation on node while the BMF can perform in-node processing of the sensed data. They all support multi-hop network, in particular EcoWizard uses Xmesh protocol, SiCA makes use Zigbee Pro protocol and BMF relies on Collection and Dissemination TinyOS protocols or on SunSPOT proprietary protocol.

BMF is the only one to be multi-platform which is key in building context and recognized by Episensor that is working with some third part vendors to integrate other nodes in SiCA.

Likewise BMF, SiCA allows to manage the network by dynamic groups to catch the morphology of buildings. In SiCA groups can be declared and nodes can be joined to them. EcoWizard nodes have no prevision for functionalities to allow managing of the network.

Furthermore, the EcoWizard System does not support actuators that instead can be used in SiCA and in BMF. Both EcoWizard and SiCA provides no API to allow developers to build custom user interfaces. In stark contrast BMF can be extended using specific Java APIs or OSGi Bundles. Table 3.2 summarizes the comparison between EcoWizard, SiCA and BMF.

In conclusion, this analysis underlines that some commercial systems have some functions similar to BMF. A major difference is that BMF can be simply extended not only at the basestation side but also at the node side. BMF optimizes the amount of data sent over the network and allows to use platforms from different vendors. Some other systems, like EcoWizard from Crossbow, are very useful to monitor a building but they have no mechanism to reduce communication overhead, capture the morphology of buildings, and to provide an extensible interface for system reconfiguration at runtime.

3.4 The Building Management Framework

The Building Management Framework (BMF) is a domain-specific framework implemented for both WSAN nodes and more capable devices at basestation side such as PC, plug computer, smartphone, PDA. It allows flexible and efficient distributed sensing and actuation in buildings. BMF fully addresses all the requirements identified in Section 3.2, which are not comprehensively addressed in currently available general-purpose application frameworks for

Table 3.2. Comparison between the Crossbow Ecowizard System, the EpiSensor SiCA for Building Management and the BMF.

	Crossbow Ecowizard System	**EpiSensor SiCA for Building Management**	**BMF**
Description	Building Monitoring System	Building Management System	Application Level Framework
Fast Reconfiguration	NO nodes cannot be reconfigured, they just send the data they sample	YES sending "commands to the nodes"	YES sending Request 4-Tuple packets
In-node processing	NO	NO	YES
Multi Hop Support	YES using Xmesh protocol	YES using Zigbee Pro	YES using Collection and Dissemination TinyOS protocols or the SunSPOT proprietary protocol
Multi-platform support	NO	NO working to integrate nodes from different vendors	YES already supported TelosB, Tyndall25, Epic motes with TinyOS 2.1, SunSPOT
Methods to dynamically catch the morphology of buildings (Building Programming Abstraction)	NO	YES	YES based on high level dynamic groups
Support for actuators	NO	YES	YES
Effective Human Computer Interface	NO	YES	YES

WSNs. In particular, BMF provides fast reconfiguration, in-node processing algorithms, multi hop networks and multi-platform support, a building programming abstraction to dynamically catch the morphology of buildings, actuators support and an extensible interface.

In Figure 3.1 are shown the main functionalities of the BMF through a layered representation divided into BS-Side and Node-Side layers.

The BS-side component includes the following layers:

- *Heterogeneous Platform Support layer* incorporates a set of adapters that allow interfacing the system with different types of sensor / actuator platforms. An adapter is linked to a specific hardware device able to communicate with a specific sensor platform in the network.
- *WSAN Management layer* allows to fully manage a WSAN cluster. This layer supports packet coding / decoding according to the BMF application-level protocol (see Section 3.4.2.4 for details) and packet transmission / reception to / from the WSAN cluster. Moreover, this layer supports device discovery within the cluster.
- *Group Organization layer* provides group-based programming of sensors and actuators, tracking of nodes, groups in the system, management of node configurations and group compositions. Node organization in groups is specifically defined to capture the morphology of buildings. Nodes belong to groups depending on their physical (location) or logical (operation type) characteristics.

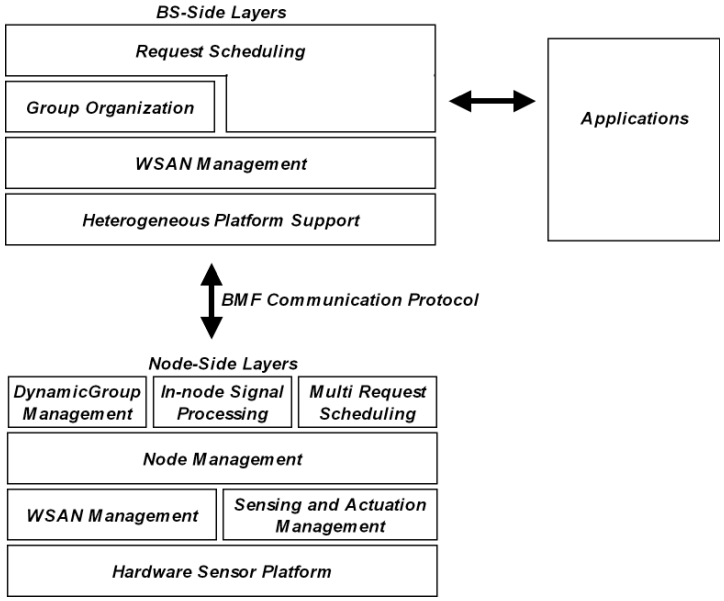


Fig. 3.1. The BMF layered architecture.

- *Request Scheduling layer* allows the support for higher-level application-specific requests. Through this layer, a BS can ask for the execution of specific tasks to single or multiple nodes or groups of nodes. Moreover, this layer keeps track of the requests submitted to the system, waits for data from the nodes and passes them to the requesting applications. An application can be implemented on top of this layer to fully manage a WSAN using the BMF.

The Node-side part of the BMF is designed around the following layers:

- *Hardware Sensor Platform layer* allows to access the hardware sensor / actuator platform. In particular, this layer facilitates the configuration of the platform specific drivers and the use of the radio.
- *WSAN Management layer* manages the communication between the node and the reference BS according to the BMF application-level protocol and among the nodes in the neighbourhood through the multi-hop network protocol provided by the node sensor platform.
- *Sensing and Actuation Management layer* allows to acquire data from sensors and execute on actuators. In particular, this layer allows to control and plugin sensors / actuators from different vendors.
- *Node Management layer* is the core of the Node-side BMF and allows to coordinate all the layers for task execution. In particular, on one side it

handles events from the lower layers every time that a network packet arrives or data from sensor / actuator are available, and on the other side it manages events from the upper layers every time that data are processed or a stored request has to be executed.

- *Dynamic Group Management layer* provides node grouping functionalities. A node can belong to several groups at the same time and its membership can be dynamically updated on the basis of requests from the user.
- *In-node Signal Processing layer* allows the node to execute signal processing functions on data acquired from sensors [64]. It can compute simple aggregation functions (e.g. mean, min, max, variance, RMS) and more complex user-defined functions on buffers of acquired data.
- *Multi Request Scheduling layer* allows the scheduling of sensing and actuation requests. In particular, it stores the requests from the user and schedules them according to their execution rate.

The BMF is implemented and tested on TelosB, Tyndall25, and Epic sensor platforms based on TinyOS and on the Java SunSPOT platforms.

In the following subsections the BMF is detailed. In particular, section 3.4.1 explains the dynamic organization of the BMF network using groups; Section 3.4.2 analyses the two processing levels composing the BMF, namely Low-Level Processing (LLP) and High-Level Processing (HLP), shows the BMF Management GUI explaining its functionalities, and presents the Building Management Framework Communication Protocol (BMFCP), an application level communication protocol that supports the communication between HLP and LLP; finally, Section 3.4.3 analyses the application scenarios in which BMF can be successfully applied.

3.4.1 Sensor Network Organization and Programming

Sensor Network Organization and Programming

The BMF dynamically organizes the nodes in groups. Formally, a group is defined as a non-empty set of nodes, as highlighted in Expression 3.1.

$$G_i = \{node_1^i, node_2^i, \ldots, node_n^i\} \neq \emptyset \tag{3.1}$$

In particular, a group formalizes either logical or physical properties of nodes. Examples of properties are: location (e.g. dining room, bedroom, kitchen), monitoring activity (e.g. heater monitoring), the presence of a specific sensing device onboard (e.g. all the nodes with passive infrared detector). Indeed, a node can belong to more than one group. Each node knows its group membership through the reception of configuration packets sent by the user through the BS. This allows a group membership be dynamically changed at any time.

Operating on groups properties yields high flexibility when addressing nodes. In particular, BMF uses dynamic node addressing scheme to send packets to a specific node, a group of nodes or a compound group. Packet transmission to several nodes at the same time can be very useful to save bandwidth. The node-addressing scheme is formalized with the Expression 3.2, where N is an element of the set of nodes in the building sensor network (N^+ is a sequence of nodes), G is an element from the set of the groups, STO is a set theory operator (e.g. union, intersection, difference) and NOT is the negation operator. After node configuration, through the dynamic node addressing, BMF allows submitting queries to single nodes or (compound) groups.

$$N^+|([NOT]G[STO[NOT]G]^*) \tag{3.2}$$

Figure 3.2 provides examples of addressing based on the scheme in Expression 3.2. In particular, the left column of the legend shows the symbols used in the map and representing nodes and groups. The right column shows examples of compound groups realized using Intersection, Union and Negation operators.

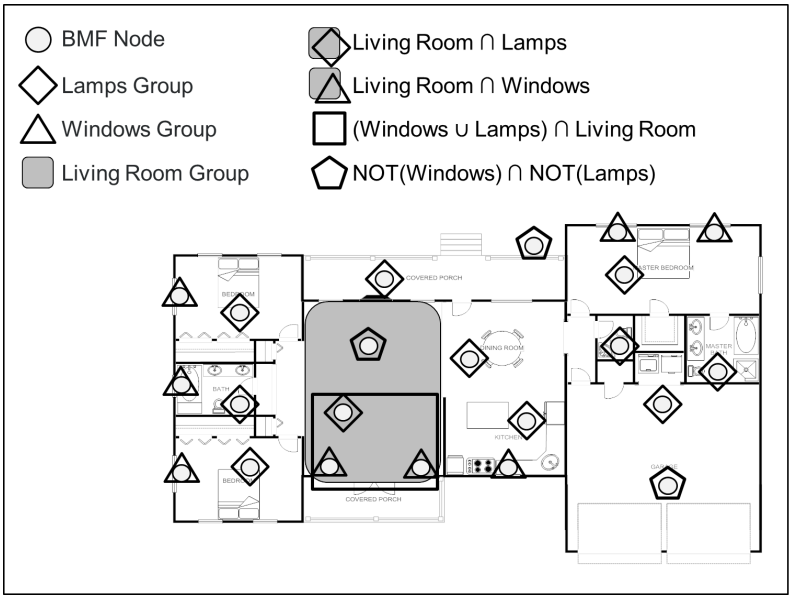


Fig. 3.2. Examples of groups and compound groups.

Sensing and actuation activity of the building sensor network can be programmed through queries, namely 4-tuple-requests, sent from the BS to nodes. A request R is formalized as the quadruple in Expression 3.3 where $OBJECT$

is a specific sensor or actuator belonging to a node, $ACTION$ is the action to be executed on $OBJECT$, $RATE$ is the frequency of each executed $ACTION$, $LIFETIME$ is the length of time over which these actions are to be reiterated. In particular: (i) if $OBJECT$ is a sensor, $ACTION$ can be a request for either raw sensed data or threshold notification (e.g. if the sensed data overtakes a given threshold, a notification is sent); (ii) if the $OBJECT$ is an actuator, $ACTION$ represents the actuation of a specific parameter of the actuator (e.g. in case of a led, $ACTION$ can activate the led toggling). The 4-tuple-requests have been introduced and chosen so to cover the majority of request types for building management apps.

$$R =< OBJECT, ACTION, RATE, LIFETIME > \tag{3.3}$$

3.4.2 Software Architecture

BMF is organized into two levels of processing logics: Low-Level Processing (LLP) and High-Level Processing (HLP). While the LLP resides on sensor nodes, HLP resides at the BS side, which is usually a laptop, a workstation, a PDA, a plug computer or a smartphone. LLP and HLP communicate according to an application-level Building Management Framework Communication Protocol (BMFCP). The BMFCP is described in detail in Section 3.4.2.4.

The implementation of BMF has been carried out on the TinyOS [19] operating system through the nesC language [65] due to its availability for many sensor platforms and flexibility to meet specific application needs and on the SunSPOT [57] sensor platform that allows the programming of a WSAN through the flexible and more user-friendly Java language. The BS side of the framework is fully implemented in Java according to the OSGi Framework [53] mainly due to its programming effectiveness, modularity and possibility to plugin applications at runtime. The OSGi Framework is a modular system and service platform for Java that implements a complete dynamic component model. Building application components, called bundles, can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot of the application. A service registry allows bundles to know other bundles and their state. Bundles can use services offered by other bundles and explicitly depend on them. Implementing the OSGi framework means implementing a modularized environment. Some of the benefits that OSGi provides are: (i) Reuse: the OSGi component model makes it very easy to use many third party components in an application; (ii) suitable for the Real World: the OSGi framework is dynamic. It can update bundles on the fly and services can come and go; (iii) Dynamic Updates, the OSGi component model is a dynamic model. Bundles can be installed, started, stopped, updated, and uninstalled without bringing down the whole system; (iv) Secure, OSGi inherits Java security and makes it simpler to be used.

LLP, HLP and BMFCP are described in detail in the following subsections.

3.4.2.1 High Level Processing (HLP)

The HLP logic of BMF is implemented at the base station side and consists of a set of OSGi bundles.

The HLP has a strong modularity that permits to implement all the services needed in different bundles that communicate through the OSGi Framework [66]. This allows to change functions or part of them just stopping a bundle and starting a new one implementing the same services of the former. As shown in Figure 3.3, the defined core bundles are:

- The *Platform Bundle (PB)* represents a set of bundles that allow to interface the system with different types of platforms. Every PB is linked to a hardware component so to communicate with a specific WSAN platform. For example, in a building WSAN consisting of TinyOS and Sun SPOT nodes there will be two PBs to communicate with TinyOS and Sun SPOT platforms.
- The *Communication Bundle (CB)* uses specific platform services published by the PBs, to allow sending and receiving packets and enabling communication between bundles and a WSAN.
- The *Groups and Nodes Management Bundle (GNMB)* keeps track of nodes and groups in a WSAN and stores nodes configurations and groups compositions. GNMB is useful to save in one place data about a WSAN and share it among several bundles.
- The *Packet Manager Bundle (PMB)* allows the creation and the interpretation of low-level packets according to the Building Management Framework Communication Protocol (BMFCP) introduced and described in detail in Section 3.4.2.4.
- The *Network Manager Bundle (NMB)* allows to fully manage a WSAN through the use of CB, GNMB, and PMB. NMB can build, send, receive and interpret packets to/from a WSAN, manage the status of nodes and the membership of groups as requested by the high level bundles.
- The *Data Saving Bundle (DSB)* listens to all the data forwarded by the NMB from the network. The DSB is designed to save data to files or a database in order to track the evolution of a WSAN and to store data for future requests.
- The *Aggregation Bundle (AB)* is responsible for executing aggregations on data from the network. AB is useful to calculate, on the base station side, aggregated synthetic data on sensor values from different nodes of a WSAN.
- The *BMF Management GUI Bundle (MGUIB)* is a graphical user interface to allow the user to manage a WSAN, submitting requests, waiting for and visualizing data from the network and displaying charts about sensing operations (application details are in Section 3.4.2.2).

Besides the GUI Configuration application implemented in the MGUIB, a new application can be built using the listed service bundles that allow exploiting the WSAN, configuring, sending and receiving packets to/from it.

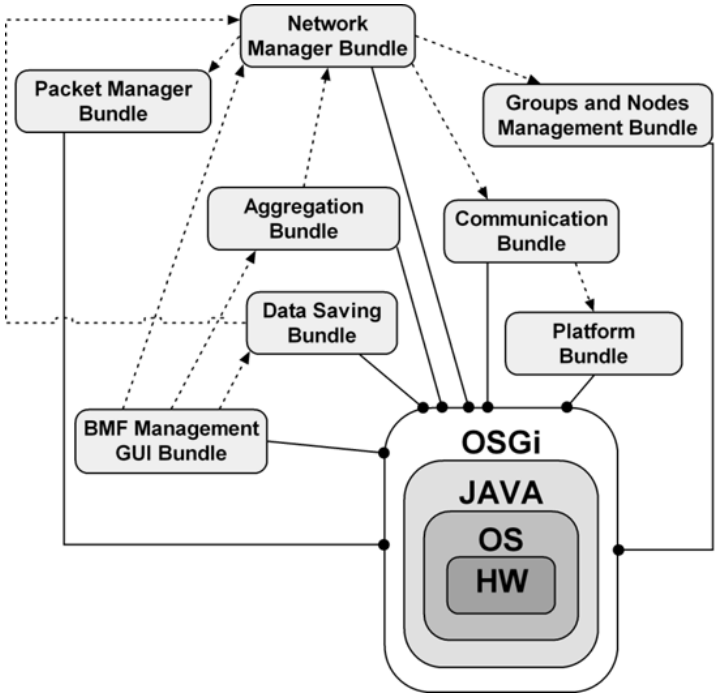


Fig. 3.3. OSGi Core Bundles of the BMF.

3.4.2.2 The BMF Management GUI

In order to demonstrate the capability of the framework, this section describes a BMF management interface to allow total control of a WSAN in a building. The interface, namely BMF Management GUI, is included in the BMF Management GUI Bundle and is shown in Figure 3.4.

The application is organized in five main sections supporting all the functionalities provided by the BMF at low and high level:

- *Nodes and groups management.* When a new BMF node appears in the network, it is automatically detected and added to the Node list tree. Once a node is detected, the GUI allows to visualize its properties, set its membership to groups, send a new request or reset it. In particular, the GUI features a few panels and functions to allow operating on the node:

  - the *Node Properties* panel shows an image of the node, all the sensors on its sensorboard and the groups which the node belongs to;
  - the *Set Membership* panel allows to add or delete a group to the membership of a node. A node membership can be updated anytime. A node can be also associated to a group by dragging and dropping the node icon on the corresponding group icon. If a membership changes, a packet is sent to the interested nodes to update their state;
  - the *New Request* panel allows sending specific requests to a destination node. The destination is preset to the selected node but it can be changed to a single node, a sequence of nodes, a group or a composition of groups (according to what is explained in Section 3.4.1). For each request, destination, name, rate of execution, duration, action to execute (sensing or actuation) and some action parameters can be set;
  - the *Reset Node* panel just asks to confirm if the node has to be reset.

  The group tree is organized similarly to the node tree with the difference that a group has to be explicitly created. The group functions that can be enabled are: visualization of nodes' group properties to see and change group belonging; sending of a new request; resetting of the group belonging; deleting a group.

- *Request management.* The request management panel allows displaying details of running or elapsed requests, reschedule them and print charts. In particular, all requests are logged in a "combo box". In addition, the "details" dialog box, allows a request to be re-scheduled or unscheduled. There is also the possibility to show all the request charts together or all the request details.

- *Maps and real time charts visualization.* The maps and real time charts panel is divided into two sections: (1) the first section is used to visualize a set of maps uploaded as image files to represent the space in which sensors are located; (2) the second section is used to visualize the charts related to the scheduled requests. The panel allows several maps to be loaded and nodes dragged and dropped from the node tree onto a map. Once a node is placed in a map, it can be deleted and moved and, also, context menu functions are still present.

- *Real-time data visualization console.* The console of the application allows visualizing the activity of the building sensor network in real-time. In particular, it displays when packets are sent to the WSAN and when ACKs or data packets are received from the nodes.

- *File* and *Saving menus.* These are used to close the application and manage saving options such as where to save and how, e.g. csv files, database, etc. By default, all the data from the network are saved in csv files which can be loaded and displayed on the GUI.

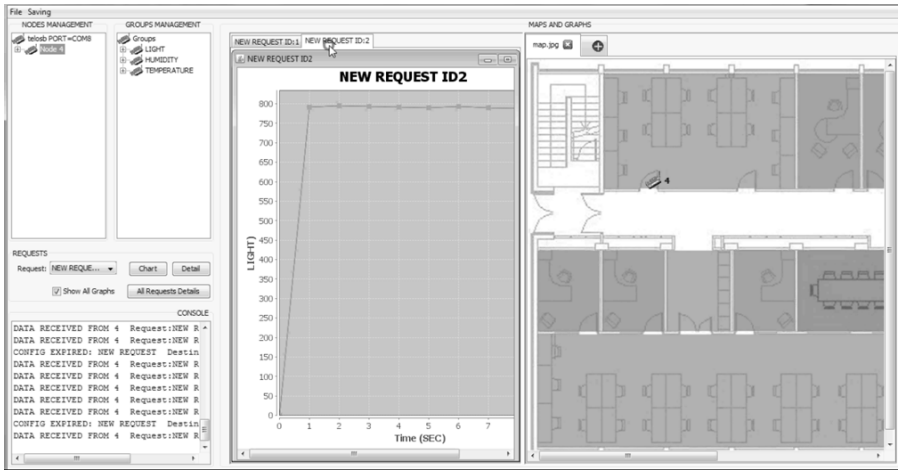


Fig. 3.4. BMF Management GUI Application.

3.4.2.3 Low Level Processing (LLP)

LLP is the node-level part of BMF and is implemented both in TinyOS 2.1, for TinyOS-based nodes, and in Java, for SunSPOT nodes. For space reason this paper focuses on the LLP for TinyOS, while the SunSPOT implementation has similar functionalities and components. As shown in Figure 3.5, LLP supports two types of logics:

- *Node Logic*, which provides acquisition of data from sensors, control of actuators, in-node signal processing; store and scheduling of requests;
- *Network Logic*, which provides downstream and upstream communication, packet parsing, packet formation, transmission handling, dynamic node addressing; group management.

The modular structure of LLP allows separating all key functions. This is useful in case of improvements of a single component, which would solely involve the replacement of the structure independently from other components. In particular:

- The *PacketManager (PaM)* handles packets of the BMF application level protocol, parses downstream packets and builds upstream packets to the BS.
- The *GroupManager (GM)* manages the group membership of a node. As a node can belong to several groups simultaneously, its membership can be dynamically updated on the basis of downstream packets sent.
- The *CommunicationManager (CM)* manages node's communication through downstream and upstream routing protocols. The current protocols used

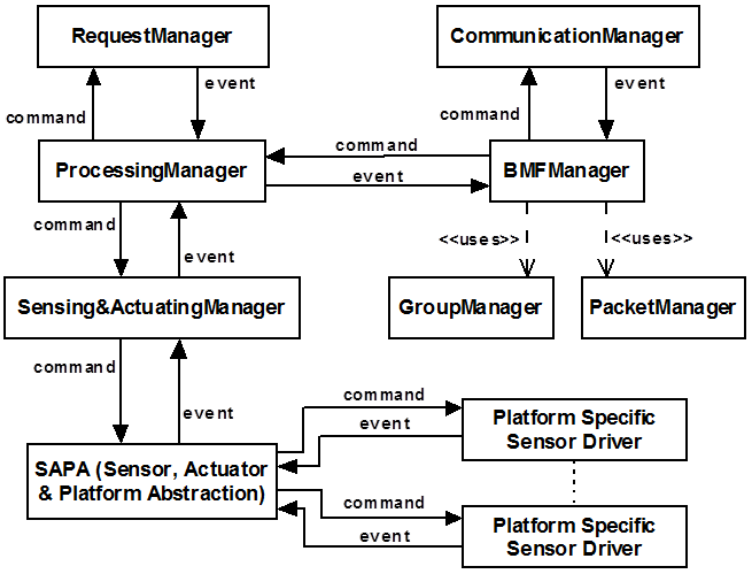


Fig. 3.5. The LLP Component Diagram.

by the CM are the TinyOS standars Collection and Dissemination protocols [67].

- The *BMFManager (BMFM)* is the broker component of the framework. BMFM receives events every time a packet arrives from the network and every time some data to be sent is ready. In the first case, it interprets the packet by means of PaM and dispatches the packet to the handling component (GM or PM); in the latter case, it builds the packet through PaM and dispatches it to CM to send it to the BS.
- The *RequestManager (RM)* stores the BS requests and schedules them according to their execution rate and lifetime.
- The *Sensing&ActuatingManager (SAM)* is a dispatcher that forwards sensor reading and actuation requests to the right sensor / actuator driver. It addresses sensors and actuators in a generic way, as it communicates with a generic platform version of the drivers.
- The *Sensor, Actuator & Platform Abstraction (SAPA)* is a layer that allows addressing different types of sensors/actuators in a platform independent way. SAPA guarantees interoperability between TinyOS-based platforms and sensing/actuation devices by creating a common API used to abstract from different low-level hardware sensor and actuator drivers. SAM exploits SAPA when a sensor is to be addressed. Thus, SAM only needs to request for a generic sensor, which is linked to the platform specific one at compilation time.

- The *ProcessingManager (PM)* manages all the requests for sensing and/or actuation coming from the network or from internal components. In particular, PM handles an internal queue of one-shot requests. According to the request type, PM can (i) interact with RM to schedule the request if it is periodical; (ii) interact with SAM to read a sensor or actuate a command; (iii) compute some function on the acquired data; (iv) check the sampled values to fire an alarm if set. Moreover, when data is ready to be sent to the BS, PM forwards them to the BMFM through an event.

In Figure 3.6 the components of LLP are shown according to a layered architectural style. It is worth noting that only RM, SAPA and CM are based on TinyOS components while SAM, PaM, PM, GM and BMFM are built on top of the other components. This means that the higher layers are also independent of the TinyOS components so that they can be easily ported to non-TinyOS platforms.

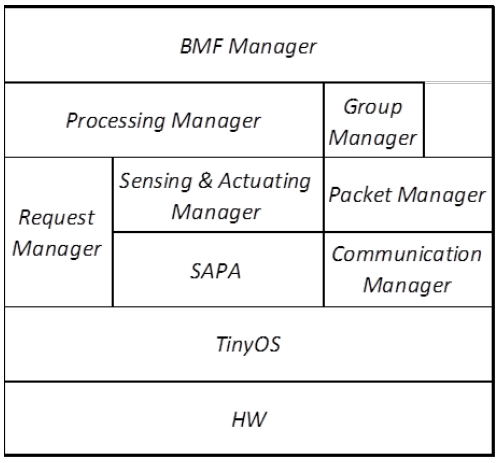


Fig. 3.6. The LLP Architectural Layers.

3.4.2.4 The Building Management Framework Communication Protocol

LLP and HLP interact through an application level communication protocol, namely Building Management Framework Communication Protocol (BMFCP). BMFCP supports the communication between HLP and LLP to configure and monitor the building sensor network in an effective manner. The packets exchanged are called BMFPackets that can be formed, depending on the specific request sent from the BS or the specific data sent from the sensor nodes, by different fields having a different amount of bytes. The BMFCP is

developed to transmit variable length packets containing only the meaningful bits of the significant fields. By doing so, the BMFCP optimizes transmissions saving battery of nodes and network bandwidth so allowing an improved sharing of the radio channel.

Table 3.3 shows all the packets with their parameters separated in Node-to-BS and BS-to-Node packets. Table 3.4 shows the predefined values of packet parameters. All Node-to-BS packets have a PKT_TYPE and a SENDER_ID field which shows the sender of the packet. Node-to-BS packets have no explicit destination fields because all upstream packets are implicitly addressed to the BS.

The BS-to-Node packets have a shared variable length header beginning with a PKT_TYPE field. Later in this section we will describe some fields representing what is formalized in Expression 3.2. An ADDRESSEE_TYPE field tells if the ADDRESSEE field is filled with node IDs or groups. Finally, groups can be combined in the ADDRESSEE field with the associative rules explained in Section 3.4.1.

As shown in Table 3.3, the framework implements eight different packets:

- The *Advertisement Packet (AD-PKT)* is a Node-to-BS packet with variable length sent when nodes are switched on. The AD-PKT is sent several times at random intervals until a packet from the BS is received so to ensure that the BS is online and received an AD-PKT from the current node. The purpose of this packet is to inform the BS about the presence of node in the network, the sensing/actuating abilities of the node itself, and the analytic available at the node in the case sensing devices are provided. The AD-PKT, besides the PKT_TYPE and SENDER_ID fields, has a SENSOR_TYPE/ACTUATOR_TYPE code that can be repeated to show all sensors and actuators connected to the node, and a FUNCTION field repeated as many times as the number of different functions that are available.
- The *Sensor Schedule Packet (SS-PKT)* is a BS-to-Node packet with variable length sent to configure the available sensing devices. The sensing can be matched with some analytic functions on node, as explained in section 3.1. The packet presents a field indicating the REQUEST_ID, namely the code of the request, and two couples of fields, PERIOD_TIMESCALE and PERIOD_VALUE, LIFETIME_TIMESCALE and LIFETIME_VALUE. These indicate timescale / value of the period at which a data is requested and the timescale / value of the duration of the current request respectively. Following are a SENSOR_TYPE with the code of the sensor required and a DATA_TYPE, indicating if the packet is asking for all the sensed (and eventually calculated) data or only for data that verifies a threshold. The THRESHOLD_TYPE field indicates if the packet requests for values over or under a THRESHOLD_VALUE or for all the times the THRESHOLD_VALUE is passed from up to down or vice versa.

Table 3.3. All the packets of the Building Management Framework Communication Protocol with their parameters.

Packet Name	*Direction*	*Parameters <KEY, VALUE>*
Advertisement Packet (AD-PKT).	Node-to-BS	<PKT_TYPE, AD-PKT> <SENDER_ID, VALUE> <SENSOR_TYPE, VALUE>* <ACTUATOR_TYPE, VALUE>* if exists(<SENSOR_TYPE, VALUE>*) <FUNCTION, VALUE>*
Sensor Schedule Packet (SS-PKT).	BS-to-Node	<PKT_TYPE, SS-PKT> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE> <REQUEST_ID, VALUE> <PERIOD_TIMESCALE, VALUE> <PERIOD_VALUE, VALUE> <LIFETIME_TIMESCALE, VALUE> <LIFETIME_VALUE, VALUE> <SENSOR_TYPE, VALUE> <DATA_TYPE, VALUE> <SYNTHETIC_DATA_TYPE, VALUE> if DATA_TYPE.VALUE == THRESHOLD_NOTIFICATION <THRESHOLD_TYPE, VALUE> <THRESHOLD_VALUE, VALUE>
Actuator Schedule Packet (AS-PKT).	BS-to-Node	<PKT_TYPE, AS-PKT> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE> <REQUEST_ID, VALUE> <PERIOD_TIMESCALE, VALUE> <PERIOD_VALUE, VALUE> <LIFETIME_TIMESCALE, VALUE> <LIFETIME_VALUE, VALUE> <ACTUATOR_TYPE, VALUE> <ACTUATOR_PARAM, VALUE>*
Unschedule Packet (U-PKT)	BS-to-Node	<PKT_TYPE, U-PKT> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE > <REQUEST_ID, VALUE>
Group Management Packet (GM-PKT)	BS-to-Node	<PKT_TYPE, GM-PKT> <ADDRESSEE_TYPE, VALUE > <ADDRESSEE, VALUE> <MEMBERSHIP_TYPE, VALUE> <MEMBERSHIP_COUNT, VALUE> If MEMBERSHIP_TYPE.VALUE != RESET <MEMBERSHIP_GROUPS, VALUE> endIf
Reset Packet (R-PKT)	BS-to-Node	<PKT_TYPE, R-PKT> <ADDRESSEE_TYPE, VALUE > <ADDRESSEE, VALUE >
Data Packet (D-PKT)	Node-to-BS	<PKT_TYPE, D-PKT> <SENDER_ID, VALUE> <TIMESTAMP, VALUE> <REQUEST_ID, VALUE> <RESULT, VALUE>
Ack Packet (A-PKT)	Node-to-BS	<PKT_TYPE, A-PKT> <SENDER_ID, VALUE> <PKT_TYPE_TO_ACK, VALUE> <ACK_PARAM, VALUE>

Table 3.4. Predefined values of the parameters in the Building Management Framework Communication Protocol packets.

Additional Parameter <KEY>	***Description***	***PREDEFINED VALUEs***
ADDRESSEE_TYPE	The type of addressee in the following	Node, List of Nodes, GROUP, GROUP_COMPOSITION
ADDRESSEE	The addressee of the packet	N+ | ([NOT] G [STO [NOT] G]*)
REQUEST_ID	The unique identifier of a request	No predefined value. Eight bit Integer
PERIOD_TIMESCALE	The timescale of the period	MSEC, SEC, MIN, HOUR, DAY
PERIOD_VALUE	The period of the request execution	No predefined value. Six bit Integer
LIFETIME_TIMESCALE	The lifetime of the request	MSEC, SEC, MIN, HOUR, DAY
LIFETIME_VALUE	The timescale of the request	No predefined value. Six bit Integer
SENSOR_TYPE	The specific sensor type	ACC_X, ACC_Y, ACC_Z, HUMIDITY, IR, LIGHT, MAGNETIC_X, MAGNETIC_Y, SOUND, TEMPERATURE, ELECTRICITY, INTERNAL_VOLTAGE
ACTUATOR_TYPE	The specific actuator type	LED
ACTUATOR_PARAM	An actuator parameter	If ACTUATOR_TYPE == LED LED_0_TOGGLE, LED_1_TOGGLE, LED_2_TOGGLE
DATA_TYPE	The data type of sensor readings	SENSED_DATA, THRESHOLD_NOTIFICATION
SYNTHETIC_DATA_TYPE	The synthetic data type of sensor readings. Data aggregation can be set.	NO_SYNTHETIC (RAW DATA), AVERAGE, MIN, MAX
THRESHOLD_TYPE	The threshold type applied on sensor reading	LOWER, BIGGER, TRANSITION
MEMBERSHIP_TYPE	The type of membership operation	UPDATE, ADD, DELETE, RESET
MEMBERSHIP_COUNT	The counter of the membership configuration sent	No predefined value. Eight bit Integer
FUNCTION	The type of in-node function computed on the sampled data	ELABORATION_AND_THRESHOLD_STANDARD, ELABORATION_STANDARD, THRESHOLD_STANDARD, AVERAGE, MIN, MAX, THRESHOLD_TYPE_LOWER, THRESHOLD_TYPE_BIGGER, THRESHOLD_TYPE_TRANSITION
TIMESTAMP	Timestamp of the transmitted data	No predefined value. Eight bit Integer
RESULT	Transmitted data	No predefined value. Variable length 8, 16 or 32 bit Integer
PKT_TYPE_TO_ACK	The packet type to ack	SS-PKT, AS-PKT, U-PKT, GM-PKT
ACK_PARAM	Type of ack	if PKT_TYPE_TO_ACK == SS-PKT || AS-PKT || U-PKT REQUEST_ID.VALUE if PKT_TYPE_TO_ACK == GM-PKT MEMBERSHIP_COUNT.VALUE

- The *Actuator Schedule Packet (AS-PKT)* is a BS-to-Node packet of variable length sent to configure actuation on them. In the packet there are the same timing fields as in the SS-PKT. Afterwards, there are an ACTUATOR_TYPE field with the code of the actuator and an ACTUATOR_PARAM which is filled with a value linked to the actuator type.

- The *Unschedule Packet (U-PKT)* is a BS-to-Node packet of variable length sent to unschedule an active sensing or actuation request on them. Apart from the header, the only field of this packet is the REQUEST_ID to request an unscheduling.
- The *Group Management Packet (GM-PKT)* is a BS-to-Node packet with variable length that is sent to set the groups to which the addressees have to belong to. In the packet, there is a MEMBERSHIP_TYPE field which indicates if the groups in the following have to be added to the actual membership, if the membership has to be completely updated, if some groups have to be deleted or if the group membership has to be reset. The field MEMBERSHIP_COUNT represents the counter of the GM-PKTs sent. MEMBERSHIP_GROUPS is a list of nodes which is present in the packet if the MEMBERSHIP_TYPE is different from "reset".
- The *Reset Packet (R-PKT)* is a BS-to-Node packet with variable length sent to restore addressees' initial state. This packet has no fields apart from the header.
- The *Data Packet (D-PKT)* is a Node-to-BS packet with variable length sent when some sensed and elaborated or just sensed data is ready. The SENDER_ID field reports the producer of the following data, the TIMESTAMP is a progressive number saying the amount of data sent from the SENDER_ID, the REQUEST_ID indicates what is the request the RESULT data refers to.
- The *Ack Packet (A-PKT)* is a Node-to-BS packet sent when nodes have to confirm the reception and elaboration of a packet from the BS. Besides the PKT_TYPE and the SENDER_ID fields, this packet contains a field, PKT_TYPE_TO_ACK, which reports the packet type that is acknowledged and an ACK_PARAM that reports a field of the acknowledged packet (e.g. in the case of acknowledging a SS-PKT, the REQUEST_ID is provided).

In order to understand the interactions between BS and sensor nodes through the BMFCP, Figure 3.7 shows a sequence diagram of example. Once a node is started, it periodically emits AD-PKTs until the BS sends a configuration packet (group management or request scheduling). Through the GM-PKT, the BS manages the membership of target nodes. After the node processes the received packet, it sends the A-PKT to acknowledge the packet received to the BS. The SS-PKT (or AS-PKT) allows to request a specific sensing (or actuation) operation to target nodes. The node transmits sensed

(processed) data to the BS through the D-PKT. The BS can unschedule previously scheduled requests through the U-PKT. Finally the BS sends out the R-PKT to reset target nodes that start again to send AD-PKTs.

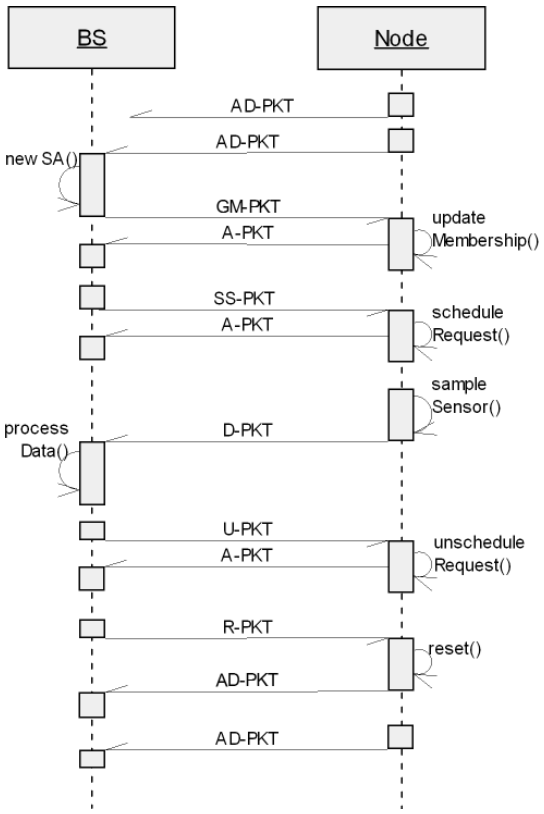


Fig. 3.7. Sequence Diagram of the interactions between BS and Nodes.

BMFCP relies on standard network protocols. Actually, in the TinyOS implementation of the framework, the downstream communication (from BS to nodes) is based on the *Dissemination protocol* [67]. Dissemination is a service for establishing eventual consistency on a shared variable. Every node in the network stores a copy of this variable. The dissemination service tells nodes when the value changes, and exchanges packets so it will reach eventual consistency across the network. At any given time, two nodes may disagree, but over time the number of disagreements will shrink and the network will converge on a single value. Eventual consistency is robust to temporary disconnections or high packet loss. Unlike flooding protocols, which are discrete efforts that terminate and not reach consistency, dissemination assures that the network will reach consensus on the value as long as it is not disconnected. So, using

the Dissemination protocol in the BMF, every node receives any packet sent from the BS and, by simply analyzing the packet destination field (in the application level protocol) representing target nodes or groups, the node decides if the packet is to be handled (the node belongs to the destination field) or discarded (the node does not belong to the destination field). The upstream communication relies on the *Collection Tree Protocol (CTP)* [67]. CTP is a tree-based collection protocol. In CTP, some number of nodes in a network advertise themselves as tree roots. Nodes form a set of routing trees to these roots. CTP is address-free in that a node does not send a packet to a particular root; instead, it implicitly chooses a root by choosing a next hop. Nodes generate routes to roots using a routing gradient. In the case of BMF, the only node set as root is the BS, so that all the packets are implicitly directed to the BS itself.

3.4.3 BMF-enabled Application Scenarios

The BMF was specifically designed for dense WSANs based on a wide range of heterogeneous platforms with numerous application scenarios. The BMF is mainly targeted to building indoors, however, its versatility would allow the framework to be effectively reused in different contexts (e.g. building outdoor monitoring, area monitoring, industrial monitoring, agriculture monitoring). In particular, the BMF has been used for:

1. *energy monitoring,* analysing data coming from the sensors to understand the energy spent in a building;
2. *behavioural monitoring,* understanding the behaviour of people in building;
3. *space monitoring,* understanding the use of the spaces in building;
4. *intelligent actuation,* using actuators to achieve specific aims (e.g. energy saving, maximization of the comfort in the building).

In literature, there are several cases of energy monitoring through WSNs which can be more effectively re-engineered through BMF. In particular, in [68] low power cameras are used to monitor the occupancy of several rooms through the capture of small images that are elaborated on the node through processing algorithms. To allow controlling the HVAC (Heating, Ventilation and Air Conditioning) system for those rooms by using BMF, such cameras could be dynamically grouped and controlled so to seamlessly manage the application deployment in several building without the need to reprogram the nodes individually; similarly, the system to monitor, schedule, and manage energy in conference rooms proposed in [69] could also be greatly enhanced through the BMF; in [70] authors show a system to monitor office occupancy detection, ambient light and the state of lighting system in order to understand the waste of energy of the lighting system. In this case waste of energy

is intended like the energy used to power on the lights if the ambient light from the windows was enough or if there are no people in the office. In the system all nodes were programmed to periodically send every few seconds the data of ambient lighting, lighting state and presence detection.

The realization with the BMF of the cases exposed above would be effective and straightforward due to the flexibility of the framework that allows to set over the air the proper nodes to produce the data requested at the target rate. The BMF autonomously collects data from the WSAN, stores it in files or database and allows to elaborate it in real time through a specifically created OSGi-Bundle which can be designed, for example, to process data and actuate actions on the network. The possibility to introduce new and dynamic OSGi-Bundles on the BS side is very important and represents a major improvement with respect to other existing frameworks which provide monolithic BSs that can be extended only stopping and restarting a network.

The applicability of BMF is elucidated in the next section through a complete and representative energy management case study, namely SmartEnLab.

3.5 A case study: the SmartEnLab

To show the functionalities and the effectiveness of the BMF to support the monitoring of the inhabitants' behaviour in a building and energy auditing, we deployed sensors in a so called smart energy laboratory use case, namely SmartEnLab. This consists of heterogeneous nodes scattered within computer laboratory / office environments used to collect information of how the energy is used in several parts of the structure. In the SmartEnLab the sensor nodes are distributed throughout the office space to collect the appropriate data while BMF is responsible for the management of the network and the configuration of the nodes. The SmartEnLab demonstrates how BMF facilitates the management of a heterogeneous WSAN and the collection of data from the network. SmartEnLab monitors all aspects of energy use and space usage within the office space. This includes the monitoring of energy spent in several workstations and utilities such as lighting and heating. These cover the majority of the energy used in an office space so allowing to attribute energy cost for each service down to a workstation level.

Providing energy usage patterns is useful for several reasons including:

- To raise awareness of energy costs, which can increase motivation for individuals to lower their own energy use [71];
- To allow decision making polices for dynamic energy management systems [72];
- To facilitate a more efficient/greener office space design [73];

- To allow intelligent space and energy apportionment, which yields further reductions in energy use [74].

The testbed consists of heterogeneous nodes scattered within the CLARITY office space (including 40 workstations) at UCD (University College Dublin). In the attempt to identify energy usage per workstation, lighting and heating can be considered as shared entities while electrical sockets can be attributed to individual workstations. In fact, wall-mounted radiators provide heating while the office space is provided with a standard lighting array covering the whole space. Each monitored workstation was provided with a PC or a laptop and was occupied by a full time employee. The office and sensor layout is shown in Figure 3.8. Each workstation is furnished with two types of sensor, an infra-red (IR) presence detector for the occupancy, and an electricity monitor to measure the electrical energy used by the workstation. Several light sensors placed above PVC ceiling panels determine the activity of the lighting array while avoiding exposure to ambient light in the laboratory. Each radiator is provided with two temperature sensors - one sensor on the inflow and one on the outflow pipe - used to monitor the radiated energy. In particular, the SmartEnLab included: (1) ACme Electricity monitors [56] used to measure the electricity at each workstation; (2) Wieye IR sensor board [75] on top of Crossbow TelosB motes [54] to measure occupancy and light utilization; (3) Tyndall 25mm motes [55] equipped with temperature sensor for the radiator activity.

In the context of SmartEnLab, the absence of a BMF would imply that each node should be programmed individually according to the sensors onto the node, the rate of sampling from sensors, and the computation to execute on the sampled data. In addition, the user has the cumbersome issue of keeping notes of node locations, monitoring period and task synchronization during deployment. The BMF greatly facilitates this by allowing to set separate or joint tasks, group nodes according to certain properties, and initiate synchronous data collection tasks for individual or multiple groups. SmartEnLab creates a separate group for each destination of use of the nodes and for each workstation. Nodes are joined to groups as soon as they are deployed in the laboratory and switched on. Here are the groups that are basically set:

- The *workstation* group consists of an electricity monitor and an occupancy monitor. A workstation group is set for each workstation.
- The *light array* group contains a single node to determine the state of the lighting array. To save on node transmissions, the BMF allows setting simple and yet effective aggregation mechanisms such as threshold-driven transmission used for the lighting array group to identify on/off lighting state.
- The *ambient lighting and temperature* group contains a number of nodes distributed throughout the office to measure the light level and the temper-

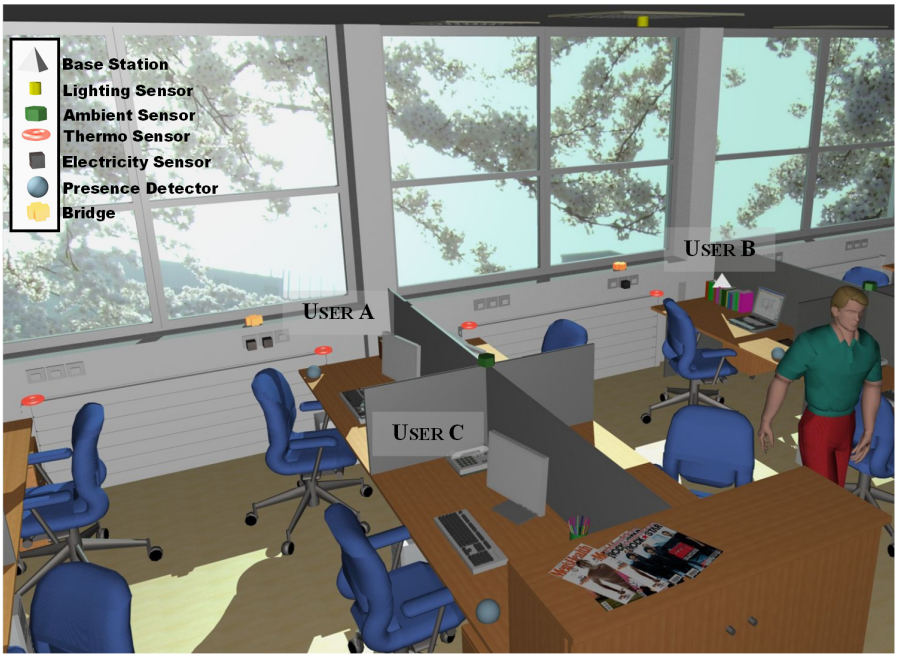


Fig. 3.8. A 3D-rendered snapshot of the SmartEnLab testbed.

ature. A task of average value on light and temperature sensors calculated over a minute was set for this group.

- The *electricity* group contains the ACme electricity monitors. A task of average value on electricity sensor calculated over a minute was set on the nodes of this group.
- The *presence* group contains the presence detectors for each workstation. SmartEnLab necessitates to set a workstation occupancy state for each minute. To facilitate this, a task to send the max sensor reading, over a minute is set to all the nodes belonging to this group.
- The *radiators* group contains nodes equipped with temperature sensor to sample radiators activity.

All the listed requests were set through the BMF Management GUI Application (provided by the BMF Management GUI Bundle) shown in Figure 3.4. The data coming from the WSAN are automatically stored by the Data Saving Bundle as soon as it is started (it is started by default). New Bundles can be dynamically implemented and added to the core set of Bundles to collect data from the network (as done by the Data Saving Bundle and the BMF Management GUI Bundle) and elaborate, correlate and visualize it in different ways.

Table 3.5 reports the complete configuration of the building sensor network set-up.

Table 3.5. Configuration of the building sensor network of SmartEnLab.

GROUP	# NODES	SENSING TYPE	SAMPLING (sample/s)	AGGREGATION (calculation/min)	SELECTION
LIGHT SYSTEM	1	Light [lx]	1	Threshold Transition	1/4
AMBIENTAL	2	Light [lx] & Temperature [°C]	1	Average	2/4
PRESENCE	40	IR	1	Max	1/4
ELECTRICITY	40	Power [W]	1	Average	1/4
RADIATORS	10	Temperature [°C]	60	NONE	1/4

Following, we describe an off-line analysis of the expended energy, the overall performance evaluation of the BMF, and an estimation of the duration of a BMF network.

3.5.1 Off-line Energy Analysis

The SmartEnLab approach allowed collecting a large amount of auditing information from the office space. This section presents the analysis results and discusses how these can be used to identify source of energy waste prior to implementing user-personalized energy reduction strategies.

Firstly, we analyse energy and occupancy profiles over a 24-hour period for each of the monitored workstations. Figure 3.9 reports data for 3 users within the office showing that each user displays a dissimilar usage/energy pattern, which requires personalised strategies of energy reductions. User A is the least efficient using 2000 W/h while spending 3.45 hours at the workstation. As shown in Figure 3.9, User A uses a high power desktop PC, which is energy inefficient as not required for the tasks performed by the employee. In addition, User A shows a tendency to leave the PC on at night, which has a major contribution to the energy cost, also identified in previous work such as [76]. User B shows the greatest efficiency of the 3 studied cases. This user uses 59 W/h of energy and spends 6.6 hours at the workstation with a total of 8.9 W/h used per hours occupied. User B uses a low-power laptop, which is switched off over long periods of inactivity. User C uses a laptop to perform general office tasks and activate a more powerful desktop PC only when higher computational needs are required. User C spends 550 W/h of energy

and spends 2.33 hours at the workstation. These preliminary indications suggest some energy reducing schemes such as informing users that laptops are preferred for all general purpose computing needs, while the employees can avail of a shared server to run more demanding tasks not suitable for laptops.

It is interesting to note that sampling rate, node positioning and grouping could not be identified prior to WSAN deployment. Therefore, BMF has been key to configure the WSAN at runtime so to allow the achievement of such results.

In a second instance, the SmartEnLab allowed understanding how productivity is correlated with occupancy. Naturally, this has a new purpose to demonstrate the BMF potential while the correlation between occupancy and productivity is a complex matter that involves many aspects and recommendations based on workstation occupancy that have to carefully be considered with several considerations in mind such as job typology and habits of the employees.

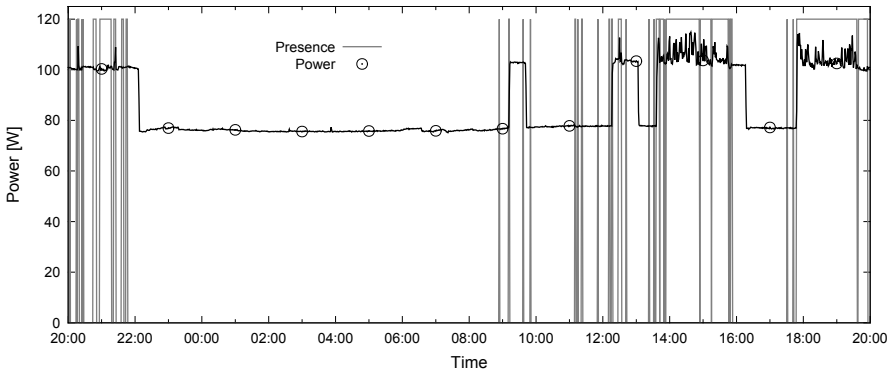
Figure 3.10 depicts the dynamics associated with the heating of the office. The graph shows the ambient temperature of the room as well as the in-flow and out-flow temperature on the pipes of one of the radiators in the office. These temperatures are important to understand the dynamics of the heating in the office and, as highlighted in [77], are fundamental for determining the power used by radiators.

Furthermore, by monitoring occupancy, light state and ambient lighting, the SmartEnLab has automated the development of LightWise-type energy-saving strategies - as demonstrated in [70] - to be applied to the lighting system.

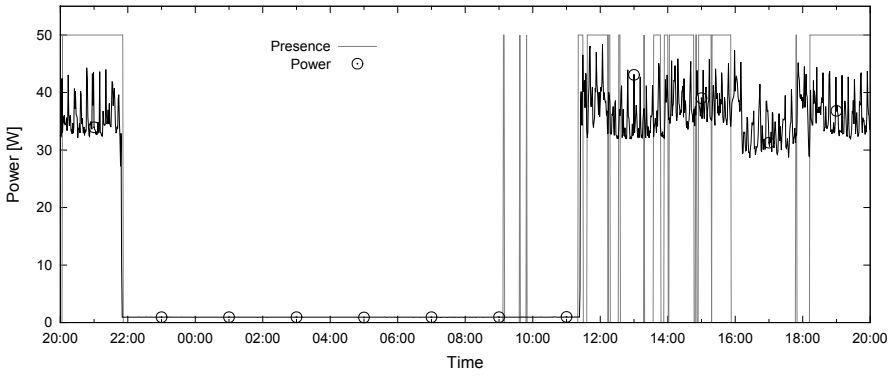
To conclude, the SmartEnLab case study demonstrated the potential of BMF is easily employed to enable a detailed profiling of the energy spent down to an individual level in an office space scenario. Thorough building monitoring is a stepping-stone towards achieving building carbon foot-printing [78] and personalised strategies to achieve high energy-efficiency taking into account individual needs and personal habits. The next section provides energy performance evaluation for the BMF in the context of the SmartEnLab, highlighting results from energy-saving techniques used to prolong the lifetime of the network and increase transmission reliability.

3.5.2 Performance Evaluation

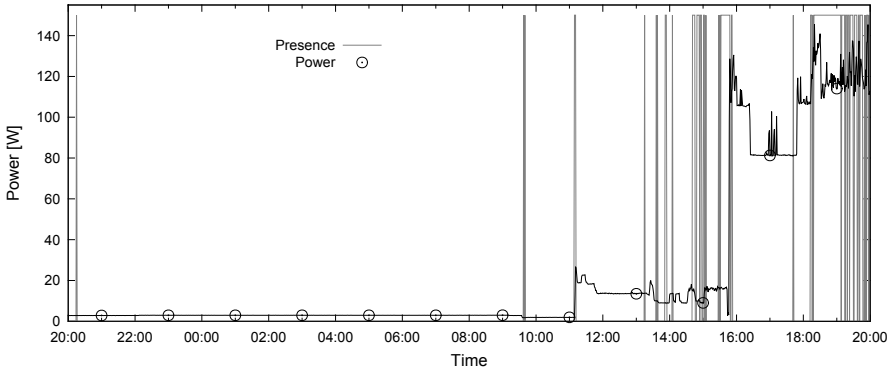
BMF sits on top of standard routing mechanisms and therefore its transmission reliability is inherently dependent on the performance of the underlying layers. However, BMF can be effectively evaluated by looking at the aggregation mechanisms, which help reducing the number of superfluous packets across the network, aiming to improve transmission reliability and node lifetime. Hence, the aim of this performance evaluation is to compare four different operating modes of the BMF used for the SmartEnLab:



(a)



(b)



(c)

Fig. 3.9. Energy and occupancy profiles over a 24 hour period for (a) User A, (b) User B and (c) User C.

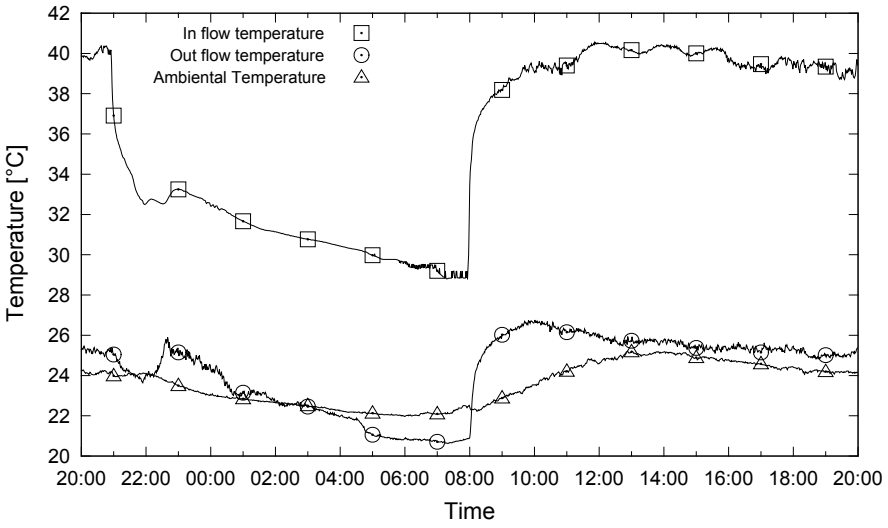


Fig. 3.10. Temperature in the office and at the radiators pipes.

(i) *SADL* (Selection and Aggregation with Dynamic Length), where selection, aggregation and the ad-hoc communication scheme of BMF based on dynamic packet length (DL) are activated;

(ii) *ADL* (Aggregation with Dynamic Length), where only aggregation and DL are activated so each sensor node sends data from all the sensors available;

(iii) *DL*, where only the DL mechanism is activated so raw sensed data are sent according to the sampling time;

(iv) *FL*, where a conventional protocol based on fixed-length packets (hereafter referred as Fixed Length - FL scheme) is used to enable raw sensed data transmission without any selection and aggregation.

In particular, for each experiment, the considered performance metrics - *packet loss* and *transmission energy* - have been experimentally evaluated over a monitoring period of 30 minutes. The experiments have been executed by varying the sensor node radio duty-cycle (DC) values in 2%, 1%, 0.5%, 0.3%. It is worth noting that a DC set to less than 1% can guarantee network longevity of more than a month. Moreover, in the FL operating mode, the high level packets are all set to 10 bytes whereas in the DL-based operating modes high level packet length ranges from 7 to 10 bytes. The building sensor network parameters (sampling, selection and aggregation) are set as in Table 3.5.

Figure 3.11 reports packet loss computed for the four different schemes by varying DC. Results demonstrate how the aggregation mechanism can

be effectively traded-off with an increase of duty-cycle to achieve a significant reduction of packet loss and a decrease of node energy consumption. In particular, the technique yields an average increase of more than 10% in transmission reliability with respect to the DL and FL, while reducing significantly the energy consumption in transmission. In node aggregation allows to appreciably reduce the radio channel usage, too. In sum, by leveraging on the in-node and dynamic length packet mechanisms, BMF can achieve lower duty-cycle therefore saving energy against classic FL transmission mechanisms. Figure 3.12 shows the transmission energy. This evaluation is carried out by using a software-based Energy component [79] which allows to know how long the radio is sending and listening to the radio channel by monitoring the right pins of the microcontroller. It is worth noting that the average energy spent can be approximated as constant and the SADL and ADL schemes outperform the DL and FL schemes, which are not based on in-node aggregation. In-node aggregation benefits are mainly network traffic reduction and transmission energy saving as also highlighted in [80].

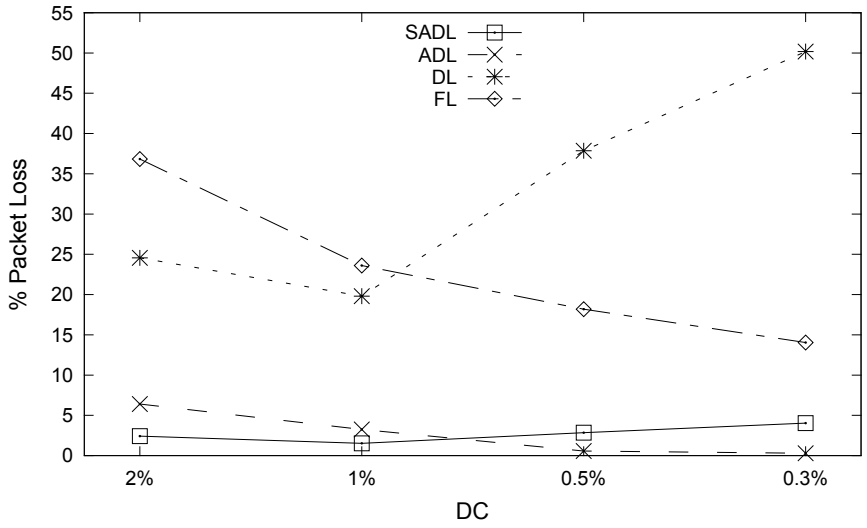


Fig. 3.11. Packet loss evaluation.

Next section provides an estimation of the duration of a network both running the BMF and applying only some of the energy-saving techniques used to prolong the lifetime of the network.

3.5.3 Lifetime estimation of the BMF-enabled Network

The same experiments explained in Section 3.5.2 were performed to estimate the WSAN life in terms of energy spent while running the BMF. For these

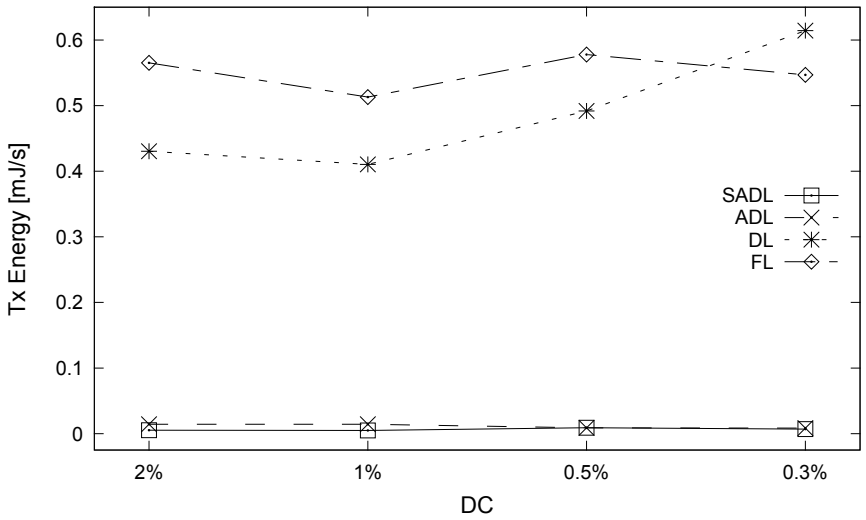
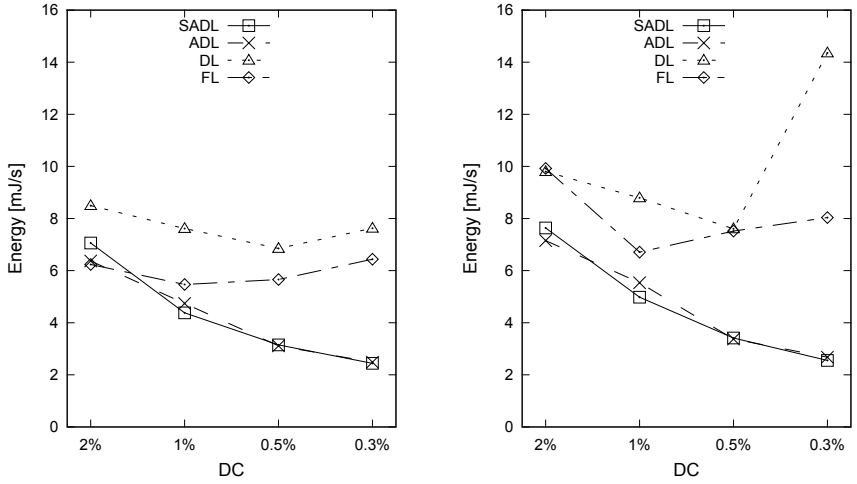


Fig. 3.12. Transmission energy evaluation.

experiments the Energy component introduced in the previous section was set to detect not only the period the radio was transmitting data but also the period in which the radio was receiving and in sleep mode. Once the duration for each radio mode was known, it was possible to calculate the mean and the max radio energy spent (by using data from the used radio chip datasheet [81]) by the nodes in our deployment for each experiment type described in Section 3.5.2. Figure 3.13(a) and Figure 3.13(b) show respectively the mean and the max radio energy spent in mW for each experiment.

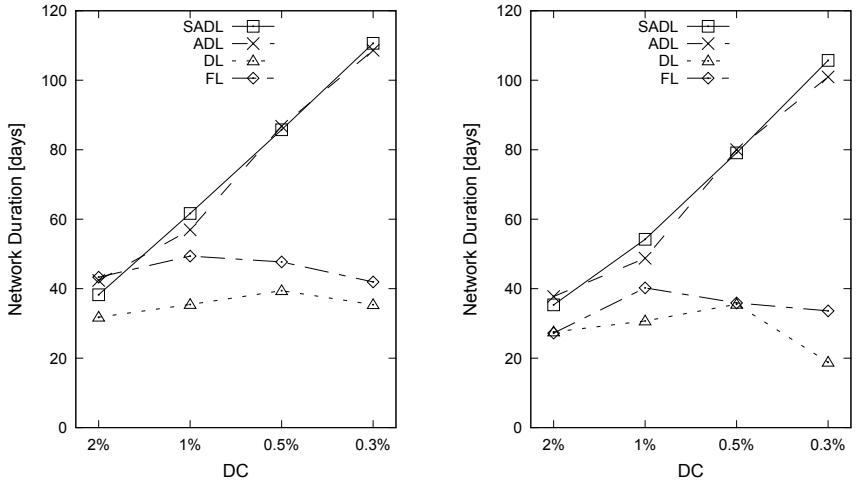
Considering each battery powered node equipped with two 1.2V rechargeable batteries of 2700 mAh and considering no supplemental energy consumptions in the nodes, the mean and the min network durations, represented in Figure 3.14(a) and Figure 3.14(b), would be respectively between 31 and 110 days and between 18 and 105 days depending on the operating modes and the duty cycle.

Since sensor nodes are working with very low duty cycles, consumptions derived from the microcontrollers and the sensors on the nodes have to be considered because they are comparable to the energy spent by the radio chip while it is in the sleep mode. So, considering the CPU active at 30% of the time, the estimated mean and min network durations decrease respectively to an amount between 47 and 22 days and between 46 and 15 days (see Figure 3.15(a) and Figure 3.15(b)) depending on the operating modes and the duty cycle.



(a) (b)

Fig. 3.13. The (a) Mean and the (b) Max radio energy spent.



(a) (b)

Fig. 3.14. The (a) Mean and the (b) Min network durations considering only the radio consumption.

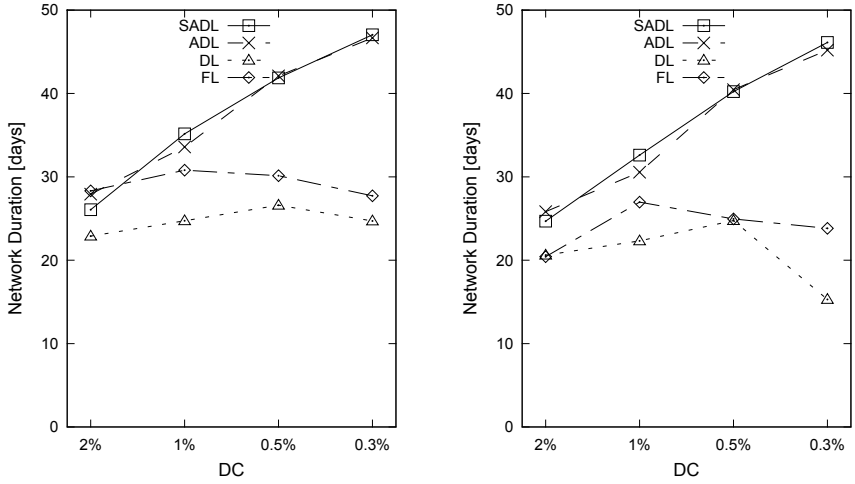


Fig. 3.15. The (a) Mean and the (b) Min network durations.

4

Mobile Agent Platform for Sun SPOT - MAPS

In this chapter the design, implementation and experimentation of MAPS (Mobile Agent Platform for Sun SPOT), an innovative Java-based framework for wireless sensor networks based on Sun SPOT technology which enables agent-oriented programming of WSN applications is presented. The MAPS architecture is based on components that interact through events. Each component offers a minimal set of services to mobile agents that are modeled as multi-plane state machines driven by ECA rules. In particular, the offered services include message transmission, agent creation, agent cloning, agent migration, timer handling and easy access to the sensor node resources (sensors, actuators, input switches, flash memory and battery).

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 realtime monitoring in smart buildings. Moreover, a performance evaluation of MAPS carried out by computing micro-benchmarks, related to agent communication, creation and migration, is illustrated and a detailed comparison between MAPS and Agent Factory Micro Edition (AFME) is shown.

4.1 Introduction

Among the programming paradigms proposed for the development of WSN applications [82] [83], the mobile agent based paradigm [84] [85], which has already demonstrated its effectiveness in conventional distributed systems as well as in highly dynamic distributed environments, can effectively deal with the programming issues that WSNs have posed. In particular, a mobile agent is a software entity encapsulating dynamic behavior and able to migrate from one computing node to another to fulfill distributed tasks. We believe that mobile agents can provide more benefits in the context of WSNs than in conventional distributed environments. In particular, mobile agents can support

the programming of WSNs at the application, middleware and network levels. At the application level, mobile agents can be used as design and programming abstractions through which WSN applications can be effectively designed and implemented. At the middleware level, mobile agents can be used for implementing WSN core services such as data aggregation/fusion/dissemination and query-based information retrieval, and for dynamically deploying new services through efficient code dissemination. At the network level, mobile agents can be used as the mobile capsules in active networks for smart multihop routing and other network services. A few trials have to date been devoted to the development of mobile agent systems (MASs) for wireless sensor networks (Agilla [35], actorNet [36], SensorWare [34]); however, none of them has been specifically developed for the Sun SPOT sensor platform [57], which is completely programmable in Java, supported by the SquawkVM [29] and compatible with J2ME. Indeed, a noteworthy agent-based framework for J2ME devices, agent factory micro edition (AFME) [86], has been recently ported on Sun SPOT; however, not being conceived specifically for Sun SPOT technology, AFME does not completely exploit its functionality.

In this chapter, we propose MAPS (Mobile Agent Platform for Sun SPOT) [87], an innovative Java-based framework for wireless sensor networks based on Sun SPOT technology which enables agent-oriented programming of WSN applications. The architecture of MAPS is component-based and offers a minimal set of services to mobile agents, including message transmission, agent creation, agent cloning, agent migration, timer handling and easy access to the sensor node resources (sensors, actuators, input switches, flash memory and battery). The dynamic behavior of mobile agents is modeled as multi-plane ECA-based state machines. MAPS therefore enables a highly effective application programming through an integration of three of the most important paradigms for WSN programming: agent-oriented, event-based and state-based programming. The effectiveness of MAPS is demonstrated by the development of simple example applications (e.g. mobile agent-based remote monitoring of sensors) as well as a more complex case study. While the former aims at testing MAPS and describing how to program an application with MAPS, the latter highlights the effectiveness and suitability of MAPS for developing a real-time monitoring in smart buildings. However, great effectiveness usually implies performance penalties; to address such issues a performance evaluation of the basic mechanisms of MAPS and Sun SPOT (e.g. communication and migration) has been carefully carried out to quantify the unavoidable overhead introduced by MAPS.

The contributions that this work offers to the WSN programming research area are the following:

- A novel Java-based agent platform for Sun SPOT that allows an effective Java-based development of agents and agent-based applications by integrating agent oriented, event-driven and state-based programming paradigms. Moreover, being based on a component based approach, which

clearly separates component interfaces from their implementations, MAPS is easily portable on other Java sensor platforms (e.g. Sentilla JCreate sensors [88]).

- The performance evaluation carried out allows evaluating not only MAPS per se but also the degree of maturity of the Sun SPOT technology for supporting (mobile) agent-based applications and systems. An interesting result is that migration is still an open issue since Sun SPOT mechanisms supporting migration still need to be improved (e.g. lack of dynamic class loading) and optimized (hibernation and serialization of Isolates are too time-consuming operations).
- The development of a real case study concerned with testing the effectiveness and the suitability of the agent based approach featured by MAPS for the development of a complex application for the real-time monitoring in smart buildings [89].

The rest of the chapter is organized as follows. Section 4.2 explains the deep correlation between mobile agents and WSNs [90]. Section 4.3 discusses related work and, in particular, currently available (mobile) agent systems/platforms for WSNs. Section 4.4 presents the requirements, architecture and the agent programming model of MAPS. Section 4.5 describes the implementation of MAPS based on the Java Sun SPOT library and shows a simple example for exemplifying the agent based application programming with MAPS. Section 4.6 shows the performance evaluation of MAPS carried out through micro-benchmarks, whereas Section 4.7 proposes a real case study developed through MAPS for the real-time monitoring in smart buildings. Finally, Section 4.8 illustrates a detailed comparison between MAPS and Agent Factory Micro Edition (AFME) [91].

4.2 Mobile agents in WSNs

Mobile agents are a distributed computing paradigm particularly suitable for supporting the development of distributed applications, services, and protocols in conventional distributed systems as well as in highly dynamic distributed environments [82] [92]. A mobile agent is a software process able to migrate from one computing node to another by retaining its execution state. As advertised by Lange and Oshima in their seminal paper [93], there are at least seven good reasons to use mobile agents; in the following we elucidate them by proposing examples in the WSN context:

1. *Network load reduction.* Mobile agents can access remote resources (e.g. databases, data repositories, sensors) or communicate with remote mobile/stationary agents or with other software components, by moving to their locations and interacting with them locally so saving network resources such as bandwidth during the interaction phase. As an example,

the following scenario is considered: *"A sensor node periodically transmits the sensed data to a sink node which, in turn, processes them to extract some aggregated information"*. The transmission of the sensed data from the sensor node to the sink node consumes bandwidth in the path from the sensor node to the sink node (this effect is much more remarkable in multi-hop paths than in single-hop paths). A mobile agent incorporating the data processing algorithm can migrate to the sensor node and locally apply this algorithm so reducing bandwidth consumption. Finally, the mobile agent should also transmit the aggregated data to the sink node but, in this case, the consumed bandwidth is much lower than the bandwidth needed for the transmission of raw sensed data.

2. *Network latency overcoming.* Mobile agents can move to remote nodes and locally apply real-time control logic for regulating physical/logical devices/components so avoiding remote control which is heavily affected by the network latency. An example scenario is as follows: *"A remote actuator/sensor should be calibrated for performing a given task and, then, controlled in real-time for successfully completing this task"*. A mobile agent equipped with the calibration and control logic can move to the actuator/sensor node location and locally calibrate and control this actuator/sensor. In this way, network latency will not affect the real-time control operations which can be continued to be performed also in case of lack of network connectivity with the base station.

3. *Protocol encapsulation.* Mobile agents can encapsulate protocols and dynamically deploy them so overcoming standardization or upgrading issues. Protocols can therefore evolve dynamically or be modified on-demand to be more efficient. In the following an example in the routing domain is provided: *"A specific routing protocol supporting multi-hop paths should be deployed in a given zone of a WSN"*. A task force of cooperating mobile agents implementing the routing protocol (e.g. rumor routing [94]) can be created and migrated into the sensor nodes of the target WSN zone. When a new protocol version is defined, a new task force implementing the new protocol is launched for replacing the previous task force at run-time.

4. *Asynchronous and autonomous execution.* Asynchrony and autonomy are distinctive properties of mobile agents. Once injected into the network, mobile agents can autonomously carry out the programmed tasks and support disconnected operations by operating asynchronously with respect to the process or device that has generated them. These properties are very important in dynamic environments where the network topology rapidly changes and connections are not stable so that they can be established only for short time periods. In the following an example of asynchronous and autonomous computing in WSN is given: *"Periodically the WSN manager queries all the nodes (or a node subset) of its WSN to have a snapshot of the energy consumption"*. To this purpose, a mobile agent, spawn by the manager, can autonomously travel across the

network to gather the requested information "node by node" and, finally, asynchronously report this information to its owner.

5. *Dynamic adaptation.* Mobile agents can sense their execution environment and react autonomously to changes. Moreover task forces of mobile agents can distribute themselves among the nodes in the network to solve particular problems in a cooperative and coordinated way. Dynamic adaptation can support autonomic behavior of a WSN; in fact, WSN are long-running systems in which device failures are overwhelmingly likely to occur. Adaptation around failed devices becomes yet another pressing feature. An example is provided in point 7.

6. *Orientation to heterogeneity.* Mobile agents can be used as integrators of heterogeneous systems. They can act as wrappers or mediators among systems based on different hardware and software. They can even be converted to cross the border among heterogeneous systems. An interesting scenario is *"the integration of a WSN with IP-based networks"*. A mobile agent can be sent to the gateway between the WSN and an IP-based network to act as wrapper of the WSN. Any information to be gathered from the WSN is requested to the mobile agent from components residing in the IP-based network. The mobile agent, in turn, will translate such requests into specific requests and submit them to the WSN and, once obtained the responses, will send them back to the requesting components.

7. *Robustness and fault-tolerance.* As mobile agents possess the ability to react dynamically to unfavorable situations and unexpected events, they can support the construction of robust and fault tolerant distributed systems. A frequent fault tolerance scenario in WSNs is the following: *"A sensor node is going to be shut down due to uncharged batteries so the sensing activity should be activated on a neighbor sensor node at runtime"*. All the mobile agents executing on the uncharged sensor node can (autonomously or passively) migrate to another equivalent node (already present or newly introduced) so continuing their activity after migration.

4.3 State-of-the-Art and Related Work

Mobile agents are supported by MASs, which basically provide an agent server, an Application Programming Interface (API) for mobile agent programming and, sometimes, supporting programming and administration tools. In particular, the agent server is able to execute agents by providing them with basic services such as migration, communication and resource access. In the last decade, a significant number of MASs for IP-based distributed computing systems have been developed [84]. The majority of them are Java-based (e.g. Aglets, Voyager, Ajanta, JADE etc.) and few others rely on other languages (D'Agents, ARA etc.).

In the context of WSNs it is challenging to develop MASs for supporting mobile agent-based programming [95]. Due to the currently available resource-constrained sensor nodes and related operating systems, building flexible and efficient MASs is a very complex task. Very few MASs for WSNs have to date been proposed and actually implemented. The most significant ones are: SensorWare [34], Agilla [35] and actorNet [36]. A general mobile-agent-oriented sensor node architecture to which such MASs adhere is shown in Figure 4.1. The MAS relies on the services offered by the OS and the mobile agents are executed within the MAS, which supports their inter-node migrations, sensing capabilities and resource access, and inter-agent communications.

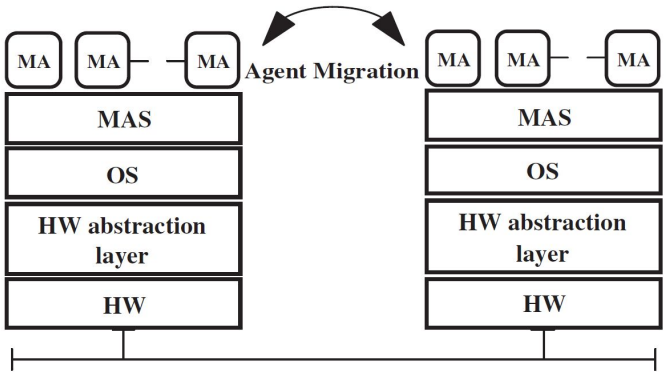


Fig. 4.1. A general mobile-agent-oriented sensor node architecture.

SensorWare [34] is a general middleware framework based on agent technology, where the mobile agent concept is exploited. Mobile control scripts in Tcl model network participants' functionalities and behaviors, and routing mechanisms to destination areas. Agents migrate to destination areas performing data aggregation reliably. The script can be very complex and diffusion gets slower when it reaches destination areas. The replication and migration of such scripts in several sensor nodes allows the dynamic deployment of distributed algorithms into the network. SensorWare defines, creates, dynamically deploys and supports such scripts. SensorWare is designed for iPAQ devices with megabytes of RAM. The verbose program representation and on-node Tcl interpreter can be acceptable overheads; however, they are not yet on a sensor node.

Agilla [35] is an agent-based middleware where each node supports multiple agents and maintains a tuple space and neighbor list. The tuple space is local and shared by the agents residing on the node. Special instructions allow agents to remotely access another node's tuple space. The neighbor list contains the address of all one-hop nodes. Agents can migrate carrying their

code and state, but do not carry their own tuple spaces. Agilla is currently implemented on MICA2, MICAZ and TelosB motes.

While both Agilla and SensorWare rely on mobile agents they employ a different communication model: Agilla's agent interaction is based on local tuple spaces, whereas SensorWare's agent interaction is based on direct communication based on network messages. In [96] another mobile agent framework is proposed. The framework is implemented on Crossbow MICA2DOT motes. In particular, it provides agent migration and agent interaction based both on local shared memory and network messages. In [97] the authors propose an extension of Agilla to support direct communication based on messages. In particular, to establish direct communications, agents are mediated by a middle component (named landmark) that interacts with agents through zone-based registration and discovery protocols.

In [36] actorNet, a mobile agent platform for WSNs based on the Actor model is proposed. In particular, it provides services such as virtual memory, context switching and multi-tasking to support a highly expressive yet efficient agent functional language. Currently, the sensor node actorNet platform is specifically designed for TinyOS on Mica2 sensors.

The above described MASs for WSNs [35] [36] [96] [97] are all implemented for TinyOS-based sensor platforms and use *ad hoc* languages for agent programming (Agilla uses a micro-programming language, whereas actorNet employs a functional-oriented language). Although some supported operations (e.g. migration) are very efficient, programming complex tasks is not so straightforward and, moreover, developers need to learn another very specific language.

Finally, the Java-based AFME [98], a lightweight version of the agent factory framework purposely designed for wireless pervasive systems and implemented in J2ME, has been recently ported onto Sun SPOT and purposely used for exemplifying agent communication and migration in WSNs [86]. However, AFME was not specifically designed for WSNs and, particularly, for Java Sun SPOT. MAPS, the Java-based agent platform proposed in this chapter, is conversely specifically designed for WSNs and fully uses the release 4.0 (blue) of the Sun SPOT library to provide advanced functionality of communication, migration, sensing/actuation, timing and flash memory storage. Moreover, it allows developers to program agent-based application in Java according to the rules of the MAPS framework, and thus no translator and/or interpreter need to be developed and no new language has to be learnt. In Table 4.1 a comparison between Agilla, actorNet, AFME and MAPS is shown.

A deeper comparison between AFME and MAPS is shown in Section 4.8.

Table 4.1. Comparison between Agilla, actorNet, AFME and MAPS.

	Agilla	*actorNet*	*MAPS*	*AFME*
Migration	Yes	Yes	Yes	Yes
Multitasking	Yes	Yes	Yes	Yes
Communication Model	tuple space	messages	messages	messages
Programming Language	proprietary ISA	Scheme-like	Java	Java
Agent Model	Assembler-like	Functional	Finite State Machine	BDI
Intentional Agents	No	No	No	Yes
Sensor Platforms	Mica2, MicaZ, TelosB	Mica2	Sun SPOT	Sun SPOT

4.4 MAPS Architecture and Programming Model

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

4.4.1 Requirements

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

1. *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.
2. *Lightweight agent architecture.* The agent architecture must also be lightweight so that agents can be efficiently executed and migrated.
3. *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.

4. *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.
5. *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.

4.4.2 Agent server architecture

The designed sensor node architecture is shown in Figure 4.2. 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 [65], the *de facto* standard for motes, relies on this kind of architecture.

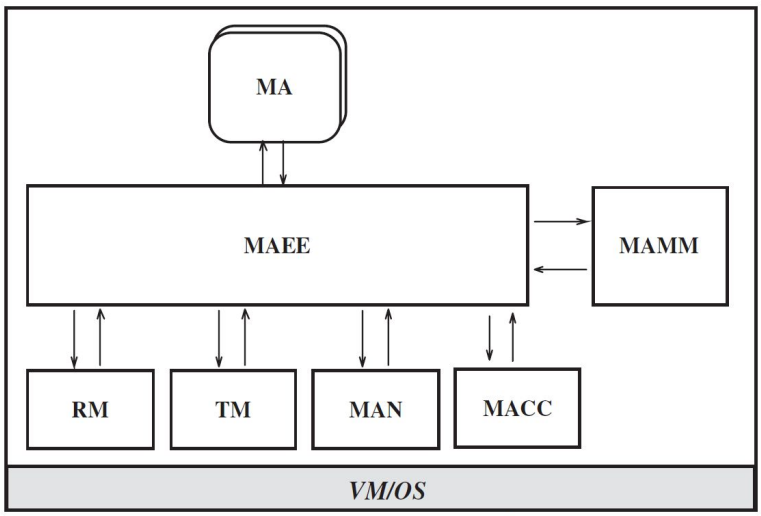


Fig. 4.2. The sensor node architecture.

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, deserialize 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 [47] 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.

4.4.3 Agent programming model

The architecture of an MA is modeled as a multi-plane state machine communicating through events (see Figure 4.3). This architecture allows exploiting the benefits derived from three paradigms for WSN programming: event-driven programming [65], state-based programming [99] and agent based programming [35]. Moreover, it enables role-based programming, an important

paradigm for agents, as agents behave differently according to the role they can assume during their lifecycle [100].

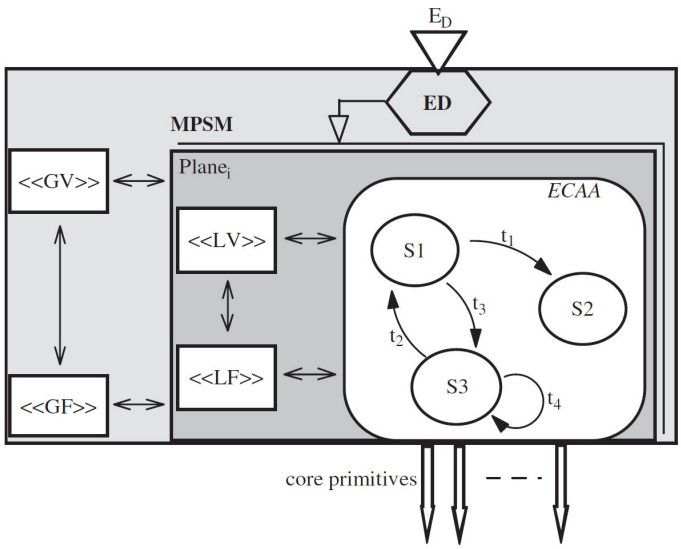


Fig. 4.3. The mobile agent architecture.

In particular the architecture consists of:

1. *Global variables (GV).* The GV component represents the data of the MA including the MA identity.
2. *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.
3. *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:
   - *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 (see Figure 4.4) 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.

4. *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.

4.5 The Software Framework

The implementation of MAPS is a real challenge due to the constrained resources of the current sensor nodes. Nevertheless, due to recent advances in operating systems and virtual machines as well as sensor technologies, an actual implementation could be done in nesC/TinyOS on TelosB motes or in Java on Sun SPOT nodes [57]. Although the implementations of the currently available mobile agent frameworks for WSN (see Section 4.3) have to date been carried out in nesC/TinyOS, by also using the Maté virtual machine [26], 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. The Sun SPOT sensor nodes are based on the SquawkVM [29] which is fully Java compliant and CLDC 1.1-compatible. In particular, the offered features are the following:

1. *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.

2. *NetBeans IDE for software development.* The IDE fully supports code editing, compilation, deployment and execution for Sun SPOTs. This enables a more rapid software prototyping.

3. *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

```
send(SourceMA, TargetMA, EventName, Params, Local)
SourceMA  = id of the transmitting MA
TargetMA  = id of the MA target |
            id of the Group target |
            ALL for event broadcast to neighbors
EventName = name of the event to be sent
Params    = set of event parameters encoded
            as pairs <attribute, value>
Local     = local (true) or remote (false) scoped event

create(SourceMA, MAId, MAType, Params, NodeLoc)
MAId    = id of the MA to be created
MAType  = type of the MA to be created
Params  = agent creation parameters
NodeLoc = node location of the created agent

clone(SourceMA, MAId, NodeLoc)
MAId    = id of the cloned MA
NodeLoc = node location of the cloned agent

migrate(SourceMA, NodeLoc)
NodeLoc = target location of the MA | ALL neighbors

sense(SourceMA, IdSensor, Params, BackEvent)
IdSensor  = id of the sensor
Params    = parameters for sensor readings
BackEvent = notifying event containing the readings

actuate(SourceMA, IdActuator, Params)
IdActuator = id of the actuator
Params     = parameters for actuator writings

input(SourceMA, BackEvent, Params)
Params     = parameters for switch selection
BackEvent  = event notifying the input captured from the
selected switch(es)

flash(SourceMA, Params, BackEvent)
Params     = flash memory access parameters
BackEvent  = event notifying the completion of the flash
memory operation (if it is a read operation, it contains
the read data)

setTimer(SourceMA, Params, BackEvent)
Params     = timer parameters
BackEvent  = event notifying the timer firing

resetTimer(SourceMA, IdTimer)
IdTimer    = id of the timer to reset
```

Fig. 4.4. The prototypal core primitives.

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.

4. *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.
5. *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.

MAPS is implemented on the basis of the aforementioned Java Sun SPOT features which fully provide support to the implementation of each component introduced in Section 4.4.2. In the following subsections the main MAPS classes (see Figure 4.5) and related functionalities are described (more implementation details as well as the MAPS framework code ver. 1.1 can be found in [101]).

The sensor node components are threads that can be instantiated through a Factory class based on the Singleton pattern [102]. Such components are actually created at the node bootstrap when the MobileAgentServer is instantiated by the main application MIDlet. The MobileAgentServer creates the MobileAgentExecutionEngine which, in turn, creates all the other components. As soon as the MobileAgentExecutionEngine starts, it activates an InterIsolateServer to communicate with mobile agent components and broadcasts a *discovery publish* event to announce itself to the neighbor agent-based sensor nodes. After the creation of the MobileAgentServer, mobile agent components can be added to it by the *addAgent* method.

The MobileAgentExecutionEngine is the core component which exposes the interface for supporting all the primitives defined in Section 4.4.3 (see Figure 4.4). The communication among agents, between agents and system components and, sometimes, among components are based on Event objects. An Event object is composed of:

- *sourceID,* which is the agent / component identifier of the event source;
- *targetID,* which is the agent / component identifier of the event target;
- *typeName,* which represents the name of the event types that are grouped according to their specific function/component (see Table 4.2 for the most important ones);

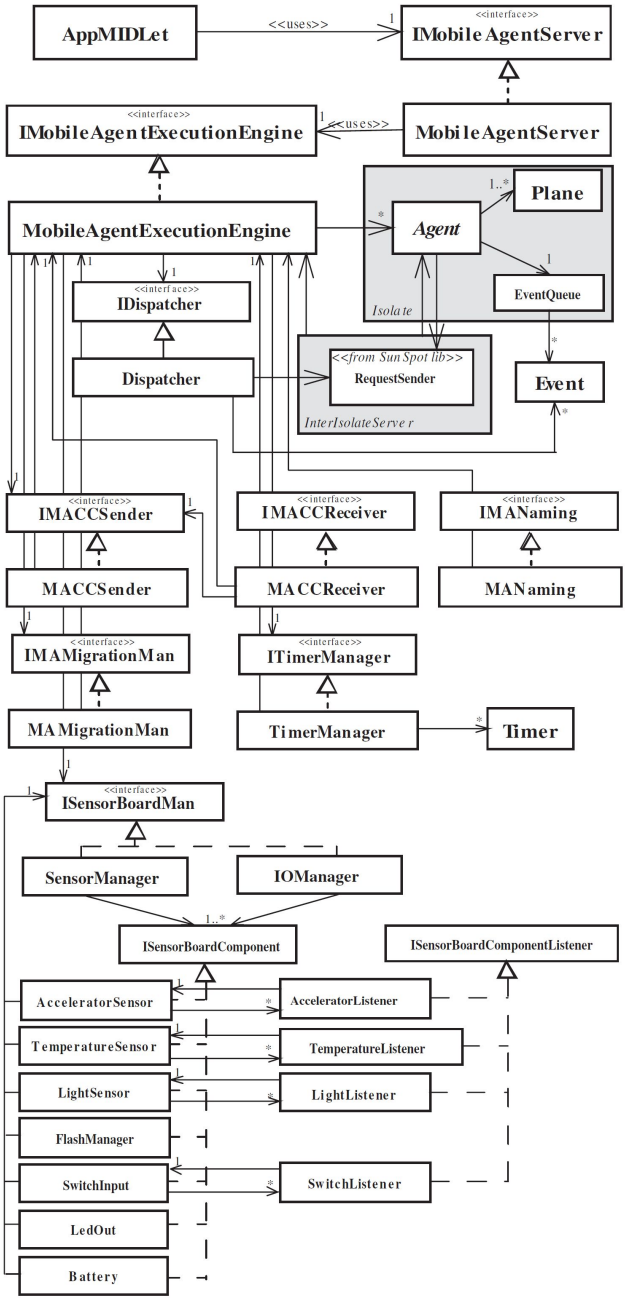


Fig. 4.5. A simplified class diagram of the MAPS framework.

- *params,* which include the event data organized as a chain of pairs <key, value>;
- *durationType,* which specifies the event duration. It can assume the following three values:
  - NOW, for instantaneous events;
  - FIRST_OCCURRENCE, for events that wait for the first occurrence of a specific value;
  - PERMANENT. In this case, the event is sent every time values set in the event parameters are met.

Table 4.2. Event types for functions and components.

Function/Component	Types	Usage description
Agent management	AGN_CREATION	Request an agent creation
	AGN_ID	Signal the ID of the created agent
	AGN_START	Start an agent
	AGN_TERMINATED	Terminate an agent
Migration	MGR_EXECUTED	Reactivate a migrated agent
	MGR_REQUEST	Request Migration an agent migration
	MGR_ACK	Signal an accomplished migration
SPOT discovery	DSC_PUBLISH	Publish a node discovery message
	DSC_ANSWER	Signal a discovered node
	DSC_REFRESH	Refresh the neighbor nodes
Message send/receive	MSG	Request a msg transmission
	MSG_TO_BASESTATION	Request a msg transmission to the BS
Timer	TMR_EXPIRED	Signal a timer expiration
Execution Engine	EXE_GET_LOCAL_AGENTS	Request the IDs of the local agents
	EXE_GET_NEIGHBORS	Request the IDs of the neighbor agents
Led	LED_ON	Request a led to be turned on
	LED_OFF	Request a led to be turned off
	LED_BLINK	Request a led to blink
Switch	SWT_PRESSED_RELEASED	Prepare a reading from a switch
Temperature sensor	TMP_CURRENT	Request the current temp value
Light sensor	LGH_CURRENT	Request the current light value
Accelerometer sensor	ACC_CURRENT	Request the current acceleration value
	ACC_TILT	Request the current tilt value
Flash	FLS_ADD	Request to write byte to the flash
	FLS_GET	Request to read byte from the flash
Battery	BTR_CURRENT_LEVEL	Request the current battery level

A mobile agent runs in a thread supported Isolate that is instantiated at agent creation time. It is composed of an event queue which contains the Event objects delivered to the agent by the Dispatcher but not yet processed, and the multi-plane state machine containing the dynamic agent behavior. Interaction between mobile agents and the MobileAgentExecutionEngine is made possible by the InterIsolate server and enabled by its RequestSender component (based on the Sun SPOT library).

Remote inter-agent communication is enabled by MACCSender and MACCReceiver component which, respectively, allows transmitting and receiving network messages according to the radiogram protocol.

The MANaming component allows managing the list of neighbor sensor nodes and agents by means of a lightweight beaconing-based announcement

protocol based on broadcast messages supported by the radiogram protocol. Moreover, agent proxy components [97] are used to route network messages to migrated mobile agents.

The MAMigrationMan component manages the migration process of a mobile agent from one sensor node to another. To this purpose, it uses the methods provided by the SquawkVM to hibernate/dehibernate and serialize/deserialize an isolate. However, as dynamic class loading is not yet supported by the current version (v4.0 blue) of the Sun SPOT libraries, the agent code should reside at the destination node. In particular, the migration process, which is single-hop and reliable, is implemented as follows: (i) the agent destination node is contacted through a specific message which causes the opening of a socket waiting for an incoming request based on the radiostream protocol; (ii) the agent destination node sends an ACK back to the agent source node; (iii) the source node therefore establishes a radiostream connection with the destination node; (iv) the mobile agent is paused, hibernated, serialized into a byte array and sent over the connection to the destination; and (v) at the destination node, the mobile agent is received, deserialized, dehibernated and reactivated.

The TimerManager component handles Timer objects which can be requested by mobile agents to time their operations. Timers can be one-shot or periodic.

Finally the SensorManager component manages available sensors (accelerometer, light and temperature) and actuators (e.g. LEDs), whereas the IOManager component manages input from switches and the flash memory.

4.5.1 A programming example: mobile agent-based remote sensor monitoring

The example agent-based application for monitoring remote sensors is structured in three agents:

1. DataCollectorAgent, which collects data sensed from the temperature, light and accelerometer sensors, and the battery of the Sun SPOT node;
2. DataMessengerAgent, which carries collected sensed data from the sensing node to the base station;
3. DataViewerAgent, which displays the received collected data.

After application deployment and execution, the DataViewerAgent sends a message to the DataCollectorAgent to start its activity as soon as the user presses a switch on the Sun SPOT on which the DataViewerAgent is running. The DataCollectorAgent therefore begins its collecting activity and as soon as the user pushes a switch on the Sun SPOT on which the DataCollectorAgent is running, it creates the DataMessengerAgent with the collected data that migrates to the DataViewerAgent node for data visualization. Finally, the

monitoring activity terminates when the user presses again a switch of the Sun SPOT on which the DataViewerAgent is running. The sequence of interactions among the three defined agents is shown in Figure 4.6 through an M-UML sequence diagram [103]. This simple yet effective application, deployed on just two sensor nodes, allows for testing the most important mechanisms provided by MAPS.

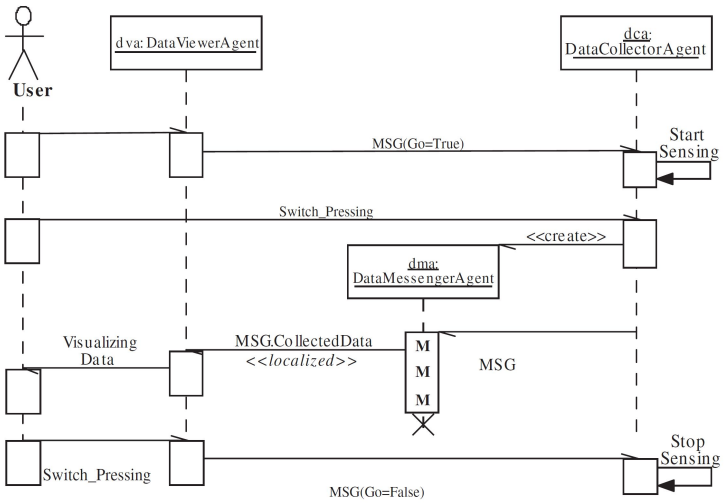
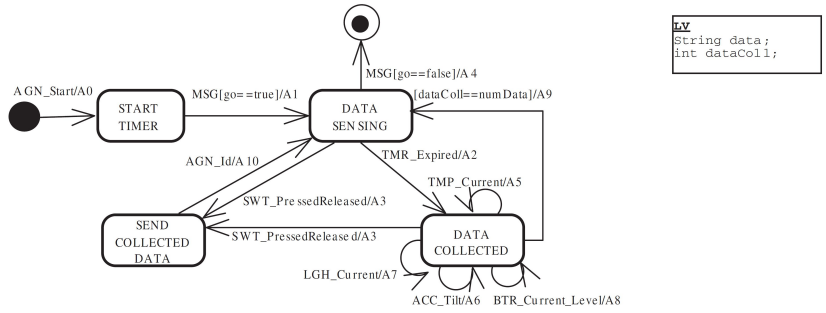


Fig. 4.6. The M-UML sequence diagram of the interactions among the agents.

For the sake of illustrating MAPS-based programming, the state machine of the DataCollectorAgent plane along with the action code, which uses the MAPS library, is shown in Figure 4.7 and briefly explained in the following. The AGN_Start event causes the transition from the creation state to the StartTimer state and the execution of an example operation on the flash memory (action A0), i.e. adding some data to the flash space of the agent. In the StartTimer state, when the network message (MSG) sent by the DataViewerAgent arrives and the guard [go == true] holds, a timer timing the sensing operations is set up to fire after 3 s, some actuation on the LEDs is requested and some input from the switches is ready to be read (action A1). When the timer fires (see TMR_Expired event), the sensing operations are requested (action A2). When such operations are completed (see actions A5A8), the guard [dataColl == numData] holds so that data are collected and, as a visual signal, an LED is actuated to blink blue (see action A9). When the switch is pressed by the user, a DataMessengerAgent is created (action A3) and the collected data are passed to it (action A10) when the AGN_Id event, containing the

agent ID of the created agent, is received. When the event MSG is received and the guard [go == false] holds, the agent is terminated (action A4).



Actions

```
A0: byte [] fls = new byte[]{12,13,14,15,16};
    Event l = new Event(agent.getId(), agent.getId(), Event.FLS_ADD, Event.NOW);
    agent.flash(l, fls);
A1: Event timer =new Event(agent.getId(), agent.getId(), Event.TMR_EXPIRED, Event.NOW );
    timerID = agent.setTimer(true, 3000, timer);
    Event blink = new Event(agent.getId(), agent.getId(), Event.LED_BLINK, Event.NOW);
    blink.setParam(ParamsLabel.LED_INDEX, "0");
    blink.setParam(ParamsLabel.LED_COLOR, "blue");
    agent.actuate(blink);
    Event swtPressed = new Event(agent.getId(),agent.getId(),Event.SWT_PRESSED_RELEASED,Event.PERMANENT);
    swtPressed.setParam(ParamsLabel.SWT_PRESSED, "false");
    swtPressed.setParam(ParamsLabel.SWT_RELEASED, "true");
    swtPressed.setParam(ParamsLabel.SWT_INDEX, "2");
    agent.input(swtPressed);
A2: Event temp = new Event(agent.getId(), agent.getId(), Event.TMP_CURRENT, Event.NOW);
    temp.setParam(ParamsLabel.TMP_CELSIUS, "true");
    agent.sense(temp);
    Event accel = new Event(agent.getId(), agent.getId(), Event.ACC_TILT, Event.NOW);
    agent.sense(accel);
    Event light = new Event(agent.getId(), agent.getId(), Event.LGH_CURRENT, Event.NOW);
    agent.sense(light);
    Event battery = new Event(agent.getId(), agent.getId(), Event.BTR_CURRENT_LEVEL, Event.NOW);
    agent.sense(battery);
A3: agent.create("test.Messenger", null, agent.getMyIEEEAddress().asDottedHex());
A4: this.terminateAgent();
A5: data+=event.getParam(ParamsLabel.TMP_TEMPERATURE_VALUE) + "-";
    dataColl++;
A6: data+=event.getParam(ParamsLabel.ACC_TILT_X_VALUE) + "-";
    dataColl++;
A7: data+=event.getParam(ParamsLabel.LGH_LIGHT_VALUE) + "-";
    dataColl++;
A8: data+=event.getParam(ParamsLabel.BTR_BATTERY_VALUE) + "-";
    dataColl++;
A9: data+="|";
    Event blink = new Event(agent.getId(), agent.getId(), Event.LED_BLINK, Event.NOW);
    blink.setParam(ParamsLabel.LED_INDEX, "0");
    blink.setParam(ParamsLabel.LED_COLOR, "blue");
    agent.actuate(blink);
    dataColl = 0;
A10:Event msg = new Event(agent.getId(), messengerAgentID, Event.MSG, Event.NOW);
    msg.setParam("collectedData", data);
    agent.send(agent.getId(), messengerAgentID, msg, true);
    data = "";
```

Fig. 4.7. The DataCollectorAgent behavior composed of one plane.

4.6 Performance Evaluation

The used testbed for testing and evaluation consists of a Sun SPOT kit (two sensor nodes and one base station) with the SDK 4.0 version (blue). The MAPS framework has a memory occupation (without any running agent) of about 70 kB in central memory, keeping free a space of 378 kB. Such space

can be exploited for agent execution. The agent developed for the agent migration benchmarking (see below) needs 22 kB of central memory. The space occupied by the jar of MAPS on the flash memory is 92 kB out of the 4 MB available [57]. To evaluate the performance of MAPS three micro-kernel benchmarks have been defined according to [104] for the following mechanisms:

1. *Agent communication.* The agent communication time is computed for two agents running onto different nodes and communicating in a client/server fashion (request/reply). Two different request/reply schemes are used: (i) *Data B&F*, in which both request and reply contain the same amount of data and (ii) *Data B*, in which only the reply contains data. Results are shown in Figure 4.8. By increasing the amount of data, communication times linearly increase. The noticed overhead of MAPS communication with respect to the cases without MAPS is mainly due to the message format of MAPS which contains event parameters (see Section 4.5).

2. *Agent creation.* The agent creation time is computed for agents having different number of planes ranging from 1 to 51. Figure 4.9 reports the results which show that the creation time is linear with respect to the number of planes.

3. *Agent migration.* The agent migration time is calculated for an agent ping-pong among two single-hop-distant sensor nodes. It has been computed in the following two cases: (i) *with MAPS*, which uses the complete functionality of MAPS and (ii) *without MAPS*, which does not use the MAPS engine and migration manager but just the Java SunSPOT library. This allows highlighting the overhead introduced by the framework for having complete migration reliability. Migration times are computed by varying the data cargo of the ping-pong agent. Although migration performances without MAPS are better, complete reliability of agent migration is not guaranteed. The obtained migration times (see Figure 4.10) are high due to the slowness of the SquawkVM operations supporting the migration process. In particular, serialization is a very costly operation: serialization of the ping-pong agent on an average takes 4.5 s. Moreover, radiostream connections are very slow to guarantee reliability.

4.7 A case study: Monitoring Smart Buildings through embedded agents

We believe that agent-based computing [85] can be exploited to implement the concept of intelligent buildings due to the agent features of autonomy,

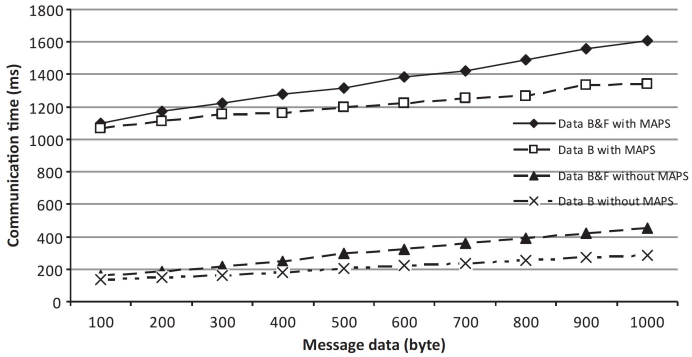


Fig. 4.8. Agent communication: request/reply time.

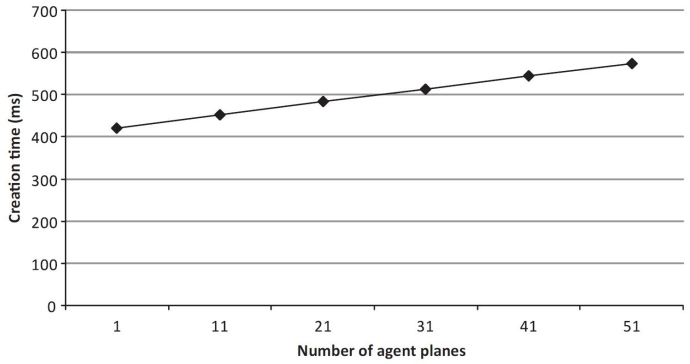


Fig. 4.9. Agent creation time.

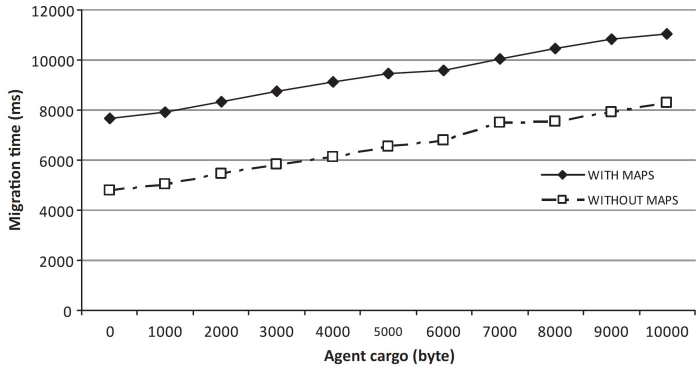


Fig. 4.10. Agent migration: ping-pong time.

proactiveness, reactiveness, learnability, mobility and social ability. Specifically agents can continuously monitor building indoors and their living inhabitants to gather useful data from people and environment and can cooperatively achieve even conflicting specific goals such as personalized people comfort and building energy efficiency. A few research efforts based on agents have been to date proposed to design and implement intelligent building systems [105] [106] [107]. However, none of them provide agents embedded in the sensor and actuator devices that would introduce intelligence decentralization and improve system efficiency. This is due to the exploitation of conventional sensing and actuation systems that do not offer distributed computing devices for sensing and actuation. To overcome this limitation, wireless sensor and actuator networks (WSAN) [46] can be adopted.

In this section we propose a decentralized and embedded management architecture for intelligent buildings that is based on WSANs [108] and overcomes the limitations of the aforementioned solutions [105] [106] [107]. In particular, the aim of our architecture is to optimize and fully decentralize the sensing and actuation operations through distributed cooperative agents both embedded in sensor/actuator devices and running on more capable coordinators (PC, plug computers, PDA/smartphones). The proposed architecture can be easily programmed to support a wide range of building management applications integrating comfort, energy efficiency, emergency, safety, and context-aware information exchange aspects.

4.7.1 Agent-based Architecture

The agent-based architecture (see Figure 4.11) for decentralized and embedded building management is composed of a building manager agent (BMA), which is installed in the control workstation, coordinator agents (CAs), which run in the basestations, and sensor agents (SAs), which are executed in the sensor/actuator nodes. Specifically, the architecture relies on a multi-basestation approach to allow for large buildings composed of multiple floors and diversified environments. Thus, the architecture is purposely hybrid: hierarchical and peer-to-peer. Interaction between CAs is peer-to-peer whereas interactions between CAs and their related SAs (or SA cluster) and between BMA and CAs are usually master/slave. Moreover, SAs of the same cluster coordinate to dynamically form up a multi-hop ad-hoc network rooted at the master CA.

In Figure 4.12 the main functionalities of BMA, CA and SA are shown according to a layered organization that is partially derived from the Building Management Framework (BMF) seen in the Chapter 3 [51].

The BMA makes it available the monitoring and control GUI through which the building manager can issue requests to configure/program the agent-based building network and visualize its status and the monitored data. Moreover, the BMA can be purposely extended to incorporate goal-directed

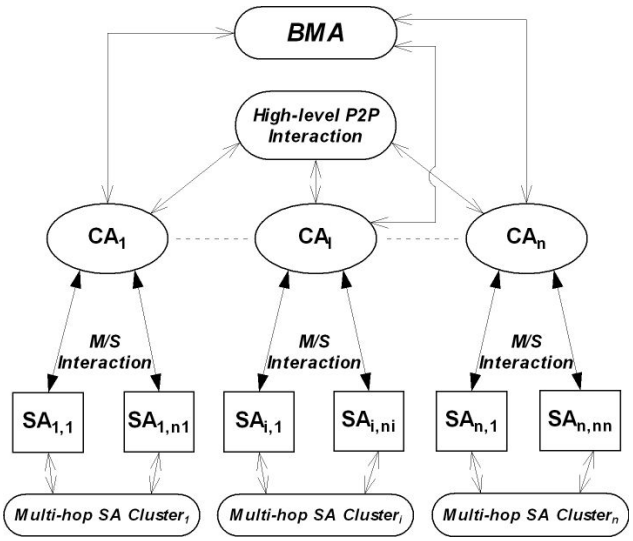


Fig. 4.11. Agent-based architecture for decentralized and embedded management of buildings based on wireless sensor and actuator networks.

behaviors for implementing specific building monitoring and control strategies. The CA includes the following layers:

- *Heterogeneous Platform Support* incorporates a set of adapters that allow interfacing the system with different type of sensor/actuator platforms. An adapter is linked to a specific hardware device able to communicate with a specific sensor platform in the network.
- *WSAN Management* allows to fully manage a WSAN cluster. This layer supports packet coding/decoding according to the BMF application-level protocol and packet transmission/reception to/from the WSAN cluster. Moreover, this layer supports device discovery within the cluster.
- *Group Organization* provides group-based programming of sensors and actuators, tracking of nodes and groups in the system, and management of node configurations and group compositions. Node organization in groups is specifically defined to capture the morphology of buildings. Nodes belong to groups depending on their physical (location) or logical (operation type) characteristics.
- *Request Scheduling* allows the support for higher-level application-specific requests. Through this layer, a CA can ask for the execution of specific tasks to single or multiple SAs or groups of SAs. Moreover, this layer keeps track of the requests submitted to the system, waits for data from the nodes and passes them to the requesting applications. A request is formalized through the following tuple: R = <Obj, Act, R, LT>, where

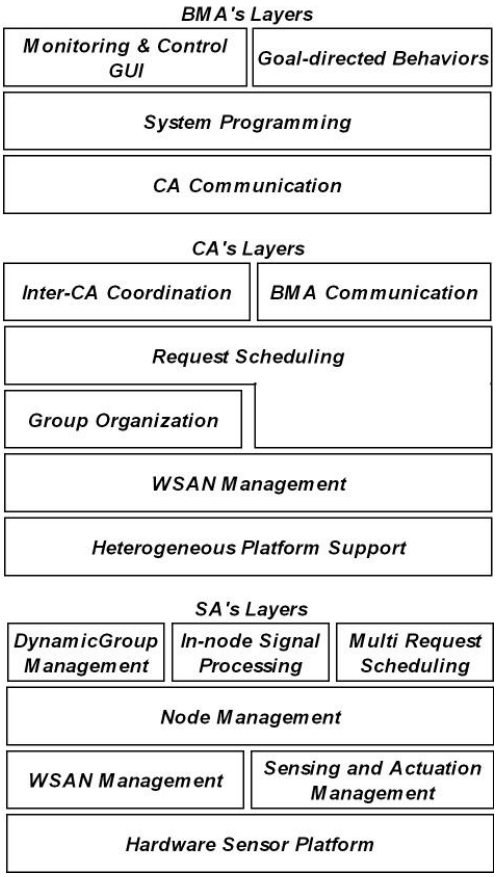


Fig. 4.12. The layered organization of BMA, CA and SA.

Obj is a specific sensor or actuator belonging to a node, Act is the action to be executed on Obj, R is the frequency of each executed Act, LT is the length of time over which these actions are to be reiterated. Moreover, a request can target a single node or a group of nodes having Obj.

- *Inter-CA Coordination* offers efficient mechanisms for coordination between CAs. Specifically, CAs cooperate for submitting queries and retrieving data spanning multiple SA clusters.

The SA is designed around the following layers:

- *Hardware Sensor Platform* allows to access the hardware sensor/actuator platform. In particular, the layer facilitates the configuration of the platform specific drivers and the use of the radio.

- *WSAN Management* manages the node communication with the reference CA according to the BMF application protocol and among the cluster nodes through the network protocol provided by the node sensor platform.
- *Sensing and Actuation Management* allows to acquire data from sensors and execute actions on actuators. In particular, this layer allows to address different types of sensors/actuators in a platform independent way.
- *Node Management* is the core of the SA and allows to coordinate all the layers for task execution. In particular, it handles events from the lower layers every time that a network packet arrives or data from sensor/actuator are available, and from the upper layers every time that data are processed or a stored request has to be executed.
- *Dynamic Group Management* provides group management functionalities to the SA. A node can belong to several groups at the same time and its membership can be dynamically updated on the basis of requests from CAs.
- *In-node Signal Processing* allows the SA to execute signal processing functions on data acquired from sensors [64]. It can compute simple aggregation functions (e.g. mean, min, max, variance, R.M.S.) and more complex user-defined functions on buffers of acquired data.
- *Multi Request Scheduling* allows the scheduling of sensing and actuation requests. In particular, it stores the requests from CAs and schedules them according to their execution rate.

4.7.2 MAPS-Based Implementation

The agent-based building management architecture is currently implemented through MAPS. In this section we present the MAPS-based implementation of the proposed building management architecture at sensor-node side, specifically behavior and event-based interactions of the SA.

The MAPS-based SA (hereafter simply named SA) interacts with its cluster CA through events as sketched in the sequence diagram of Figure 4.13. Once the SA is created, it periodically emits the BM_SA_ADVERTISEMENT event until the CA sends a configuring event (group management or request scheduling). Through the BM_GROUP_MANAGEMENT event, the CA manages the membership of target SAs (see 4.7.1). After the SA processes the received event, it sends the BM_ACK event to the CA. The BM_SENSOR_SCHEDULE (or BM_ACTUATOR_SCHEDULE) event allows to request a specific sensing (or actuation) operation to target SAs. The SA transmits sensed (processed) data to the CA through the BM_DATA event. The CA can unschedule previously scheduled requests through the BM_UNSCHEDULE event. Finally the CA sends out the BM_SA_RESET event to reset target SAs.

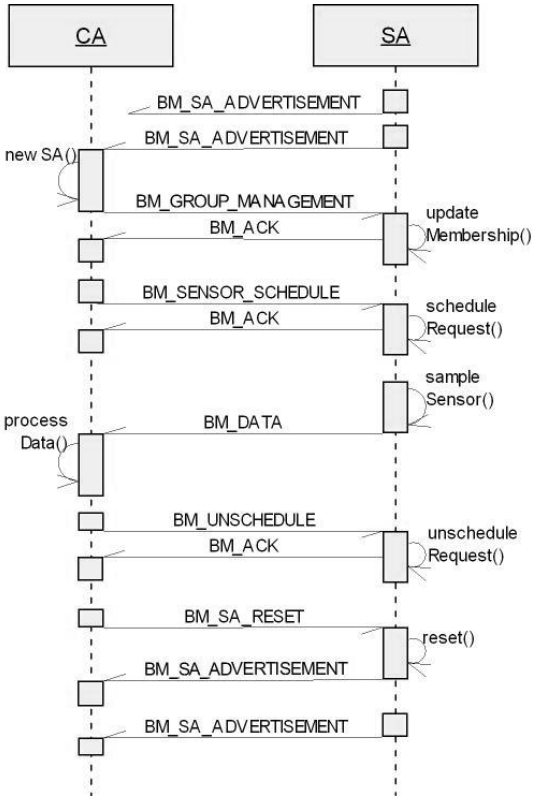


Fig. 4.13. Sequence Diagram of the interactions between CA and SA.

Tables 4.3 and 4.4 report the defined MAPS-based building management events and the predefined values of their parameters. In particular, an event is defined by its *standard parameters*: EventSender ID, EventTarget ID, Event Type, Event Occurrence. The defined events are of two possible super types: MSG (sent by CA to SA) and MSG_TO_BASESTATION (sent by SA to CA). Both types are further specialized in the defined BM events as reported in the pairs <MSG_TYPE, BM_event>of the 3rd column of Table 4.3. Moreover, each event type has its own additional parameters, which are described in Table 4.4. It is worth noting that the ADDRESSEE value can be set through the regular expression formalized in Expression 4.1 where SA is a sensor agent of the building management architecture, G is an element from the set of defined groups, STO is a set theory operator (e.g. union, intersection, difference) and NOT is the negation. Thus, the addressee of an event can be either one or more SAs, or SAs belonging to groups or complex compositions of groups.

$$SA^{+}|([NOT]G[STO[NOT]G]^{*}) \tag{4.1}$$

Table 4.3. Defined building management events.

Event Name	*Standard Parameters*	*Additional Parameters <KEY, VALUE>*
BM_SA_ ADVERTISEMENT	ID_SA; ID_CA; Event.MSG_TO_BASESTATION; Event.NOW	<MSG_TYPE, BM_SA_ADVERTISEMENT> <SENSOR_TYPE, VALUE>* <ACTUATOR_TYPE, VALUE>* if exists(<SENSOR_TYPE, VALUE>*) <FUNCTION, VALUE>*
BM_SENSOR_ SCHEDULE	ID_CA; ID_SA; Event.MSG; Event.NOW	<MSG_TYPE, BM_SENSOR_SCHEDULE> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE> <REQUEST_ID, VALUE> <PERIOD_TIMESCALE, VALUE> <PERIOD_VALUE, VALUE> <LIFETIME_TIMESCALE, VALUE> <LIFETIME_VALUE, VALUE> <SENSOR_TYPE, VALUE> <DATA_TYPE, VALUE> <SYNTHETIC_DATA_TYPE, VALUE> if DATA_TYPE.VALUE == THRESHOLD_NOTIFICATION <THRESHOLD_TYPE, VALUE> <THRESHOLD_VALUE, VALUE>
BM_ACTUATOR_ SCHEDULE	ID_CA; ID_SA; Event.MSG; Event.NOW	<MSG_TYPE, BM_ACTUATOR_SCHEDULE> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE> <REQUEST_ID, VALUE> <PERIOD_TIMESCALE, VALUE> <PERIOD_VALUE, VALUE> <LIFETIME_TIMESCALE, VALUE> <LIFETIME_VALUE, VALUE> < ACTUATOR_TYPE, VALUE> <ACTUATOR_PARAM, VALUE>*
BM_ UNSCHEDULE	ID_CA; ID_SA; Event.MSG; Event.NOW	<MSG_TYPE, BM_UNSCHEDULE> <ADDRESSEE_TYPE, VALUE> <ADDRESSEE, VALUE > <REQUEST_ID, VALUE>
BM_ GROUP_ MANAGEMENT	ID_CA; ID_SA; Event.MSG; Event.NOW	<MSG_TYPE, BM_GROUP_MANAGEMENT> <ADDRESSEE_TYPE, VALUE > <ADDRESSEE, VALUE> <MEMBERSHIP_TYPE, VALUE> <MEMBERSHIP_COUNT, VALUE> If MEMBERSHIP_TYPE.VALUE != RESET <MEMBERSHIP_GROUPS, VALUE>
BM_ SA_RESET	ID_CA; ID_SA; Event.MSG; Event.NOW	<MSG_TYPE, BM_SA_RESET> <ADDRESSEE_TYPE, VALUE > <ADDRESSEE, VALUE >
BM_DATA	ID_SA; ID_CA; Event.MSG_TO_BASESTATION; Event.NOW	<MSG_TYPE, BM_DATA> <TIMESTAMP, VALUE> <REQUEST_ID, VALUE> <RESULT, VALUE>
BM_ACK	ID_SA; ID_CA; Event.MSG_TO_BASESTATION; Event.NOW	<MSG_TYPE, BM_ACK> <MSG_TYPE_TO_ACK, VALUE> <ACK_PARAM, VALUE>

The SA agent behavior consists of two types of planes: Manager plane and Request plane. While the Manager plane is created at the SA creation time and handles all node targeting events, a Request plane is created by the Manager plane every time that a new request schedule is received. This type of plane is removed when it completes its task or due to the reception of

Table 4.4. Additional parameters of the building management events.

Additional Parameter	*Description*	*PREDEFINED VALUEs*
ADDRESSEE_TYPE	The type of event target	SA, List of SAs, GROUP, GROUP_COMPOSITION
ADDRESSEE	The event target	SA+ | ([NOT] G [STO [NOT] G]*)
REQUEST_ID	The unique identifier of a request	*no predefined int value*
PERIOD_VALUE	The period of the request execution	*no predefined int value*
PERIOD_TIMESCALE	The timescale of the period	MSEC, SEC, MIN, HOUR, DAY
LIFETIME_TIMESCALE	The lifetime of the request	MSEC, SEC, MIN, HOUR, DAY
LIFETIME_VALUE	The timescale of the request	*no predefined int value*
SENSOR_TYPE	The specific sensor type	ACC_X, ACC_Y, ACC_Z, HUMIDITY, IR, LIGHT, MAGNETIC_X, MAGNETIC_Y, SOUND, TEMPERATURE, ELECTRICITY, INTERNAL_VOLTAGE
ACTUATOR_TYPE	The specific actuator type	LED
ACTUATOR_PARAM	An actuator parameter	If ACTUATOR_TYPE == LED LED_0_TOGGLE, LED_1_TOGGLE, LED_2_TOGGLE
DATA_TYPE	The data type of sensor readings	SENSED_DATA, THRESHOLD_NOTIFICATION
SYNTHETIC_DATA_ TYPE	The synthetic data type of sensor readings. Data aggregation can be set.	NO_SYNTHETIC (RAW DATA), AVERAGE, MIN, MAX
THRESHOLD_TYPE	The threshold type applied on sensor reading	LOWER, BIGGER, TRANSITION
MEMBERSHIP_TYPE	The type of membership operation	UPDATE, ADD, DELETE, RESET
MEMBERSHIP_COUNT	The counter of the membership configuration sent	*no predefined int value*
FUNCTION	The type of in-node function computed on the sampled data	ELABORATION_AND_THRESHOLD_STANDARD, ELABORATION_STANDARD, THRESHOLD_STANDARD, AVERAGE, MIN, MAX, THRESHOLD_TYPE_LOWER, THRESHOLD_TYPE_BIGGER, THRESHOLD_TYPE_TRANSITION
TIMESTAMP	Timestamp of the transmitted data	*no predefined int value*
RESULT	Transmitted data	*no predefined int value*
MSG_TYPE_TO_ACK	The message type to ack	BM_SENSOR_SCHEDULE, BM_ACTUATOR_SCHEDULE, BM_UNSCHEDULE, BM_GROUP_MANAGEMENT
ACK_PARAM	Type of ack	if MSG_TYPE_TO_ACK == BM_SENSOR_SCHEDULE || BM_ACTUATOR_SCHEDULE || BM_UNSCHEDULE REQUEST_ID.VALUE if MSG_TYPE_TO_ACK == BM_GROUP_MANAGEMENT MEMBERSHIP_COUNT.VALUE

an unschedule event. Agent planes receive events from the MAPS dispatcher component that is programmed to deliver the events fetched from the agent queue to the plane in charge to process them. The dispatcher rules are reported in Table 4.5.

The Manager plane is reported in Figure 4.14. In particular, after agent creation, the Manager plane starts a periodic timer to advertise the agent presence along with its sensor/actuator available functions and waits for an incoming event from the CA. When it receives the first event, the timer is reset.

Table 4.5. Dispatcher rules.

Event	*Plane*
BM_SENSOR_SCHEDULE	MANAGER
BM_ACTUATOR_SCHEDULE	MANAGER
BM_UNSCHEDULE	MANAGER
BM_GROUP_MANAGEMENT	MANAGER
BM_SA_RESET	MANAGER
Event.TMR_EXPIRED <ID, ID_MANAGER_PLANE>	MANAGER
Event.TMR_EXPIRED, <ID, REQUEST_PLANE_ID>	REQUEST
Event.SENSOR_CURRENT_READING, <ID, REQUEST_PLANE_ID>	REQUEST

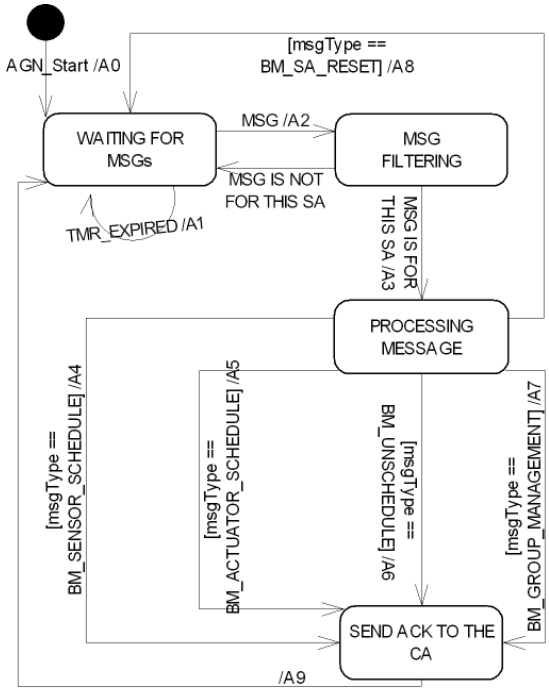
Each received event is filtered against the current SA's group membership. If the filtered event is for the current SA, it is processed according to its type. A more detailed description of each action of the Manager plane is provided using both a self-explanatory pseudocode (see Figure 4.14) and the MAPS code (intended for MAPS programmers; see Figure 4.15).

In Figure 4.16 the Sensing Request plane is portrayed. This plane is created every time that the agent receives a BM_SENSOR_SCHEDULE event. In particular, after the Sensing Request plane creation, the plane creates and submits the MAPS sensing event formalizing the sensing request. A sensing request can be either one-shot or periodic with a given lifetime. The request is scheduled until LIFETIME_ELAPSED==true after the expiration of the periodic timer driving the submission of the sensing event. A more detailed description of each action of the Sensing Request plane is provided using both a self-explanatory pseudocode (see Figure 4.16) and the MAPS code (intended for MAPS programmers; see Figure 4.17).

4.8 AFME/MAPS comparison

This section proposes an in-depth analysis of the only two available Java-based mobile agent platforms for WSNs: Mobile Agent Platform for Sun SPOT (MAPS) and Agent Factory Micro Edition (AFME) [98]. 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. This case study is developed both in MAPS and AFME on Sun SPOTs so as to allow both an analysis of efficacy of their programming models and an evaluation of their performances.



A0: firstProcessedEvent=FALSE
Start a periodic Event.TMR_EXPIRED to send the BM_SA_ADVERTISEMENT
A1: Send BM_SA_ADVERTISEMENT to CA
A2: if the MSG is for this SA
firstProcessedEvent=TRUE && resetTimer (ID_TIMER)
A3: msgType = msgEvent.getParam(ParamsLabel.MSG_TYPE)
A4: Create a new Sensor Plane:
PlaneID = ID_REQUEST, the Request as parameter and start it.
A5: Create a new Actuator Plane:
PlaneID = ID_REQUEST, the Request as parameter and start it
A6: Unschedule the Request deallocating the Plane with ID = ID_REQUEST
A7: Update current SA Group Membership
A8: Reset the SA and deallocate all the Request Planes
firstProcessedEvent=FALSE
A9: Send BM_ACK to CA

Fig. 4.14. The SA's Manager plane.

4.8.1 Agent Factory Micro Edition (AFME)

AFME [109] [86] [98] is an open-source lightweight J2ME MIDP compliant agent platform based upon the pre-existing Agent Factory framework [110] and intended for wireless pervasive systems. Thus, AFME was not specifically designed for sensor networks but, thanks to a recent support of J2ME onto the Sun SPOT sensor platform, it can be adopted for developing agentbased WSN applications.

AFME is strongly based on the Belief-Desire-Intention (BDI) paradigm [111], in which agents follow a sense-deliberate-act cycle. To facilitate the creation of BDI agents the framework supports a number of system components

```
A0: addDispatcherRule(msgTypeList());
    firstProcessedEvent=FALSE;
    Event timer = new Event(agent.getId(), agent.getId(), Event.TMR_EXPIRED, Event.NOW );
    timerID = agent.setTimer(true, advertisementTime(), timer);
    addDispatcherRule(timer);
A1: Event msg = new Event(agent.getId(), agent.getCAId(), Event.MSG_TO_BASESTATION, Event.NOW);
    msg.setParam(ParamsLabel.MSG_TYPE, BM_SA_ADVERTISEMENT);
    setAdvertisementParams(msg);
    agent.send(agent.getId(), agent.getCAId(), msg, true);
A2: if(isMsgForCurrSA(msgEvent.getParam(ParamsLabel.ADDRESSEE),
    msgEvent.getParam(ParamsLabel.ADDRESSEE_TYPE)){
        firstProcessedEvent=TRUE;
        removeDispatcherRule(timer);
        agent.resetTimer(agent.getId(), timerID);
    }
A3: msgType = msgEvent.getParam(ParamsLabel.MSG_TYPE);
A4: plane = createSensorPlane(msgEvent.getParam(ParamsLabel. REQUEST_ID), msgEvent);
A5: plane = createActuatorPlane(msgEvent.getParam(ParamsLabel. REQUEST_ID), msgEvent);
A6: agent.removePlane(msgEvent.getParam(ParamsLabel.REQUEST_ID));
A7: updateMembership(msgEvent);
A8: Iterator i = agent.getPlaneList();
    while(i.hasNext()){
        plane = (Plane)i.next();
        if(plane.getID() != this.getID())
            agent.removePlane(plane.getID());
    }
    firstProcessedEvent=FALSE;
    timer = new Event(agent.getId(), agent.getId(), Event.TMR_EXPIRED, Event.NOW );
    timerID = agent.setTimer(true, advertisementTime(), timer);
    addDispatcherRule(timer);
A9: Event msg = new Event(agent.getId(), agent.getCAId(), Event.MSG_TO_BASESTATION, Event.NOW);
    msg.setParam(ParamsLabel.MSG_TYPE, BM_ACK);
    setAckParams(msg);
    agent.send(agent.getId(), agent.getCAId(), msg, true);
```

Fig. 4.15. The MAPS actions of the SA's Manager plane.

which developers have to extend when building their applications: perceptors, actuators, modules, and services. Perceptors and actuators enable agents to sense and to act upon their environment respectively. Modules represent a shared information space between actuators and perceptors of the same agent, and are used, for example, when a perceptor may perceive the resultant effect of an actuator affecting the state of an object instance internal to the agent. Services are shared information space between agents used for agent data exchange.

The agents are periodically executed using a scheduler, and four functions are performed when an agent is executed (BDI-cycle). First, the perceptors are fired and their sensing operations generate beliefs, which are added to the agents belief set. A belief is a symbolic representation of information related to the agents state or to the environment. Second, the agents desires are identified using resolution-based reasoning, a goal-based querying mechanism commonly employed within Prolog interpreters. Third, the agents commitments (a subset of desires) are identified using a knapsack procedure. Fourth, depending on the nature of the commitments adopted, various actuators are fired.

In AFME, agents are defined through a mixed declarative/imperative programming model. The declarative Agent Factory Agent Programming Language (AFAPL), based on a logical formalism of beliefs and commitments, is used to encode an agents behavior by specifying rules that define the con-

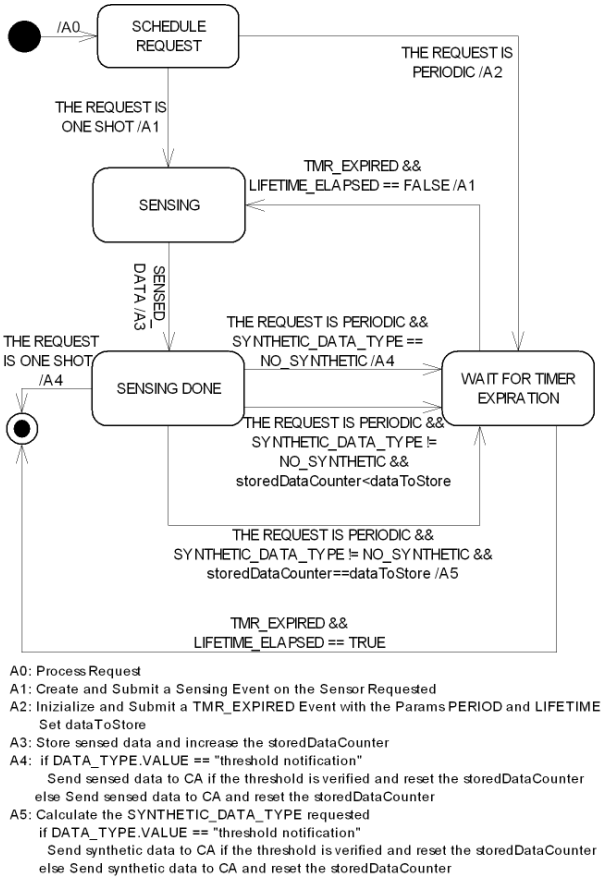


Fig. 4.16. The SA's Sensing Request plane.

ditions under which commitments are adopted. The imperative Java code is instead used to encode perceptors and actuators. A declarative rule is expressed in the following form:

$$b1, b2, \ldots, bn > doX;$$

where $b1, \ldots, bn$ represent beliefs, whereas doX is an action. The rule is evaluated during the agent execution, and if all the specified beliefs are currently included into the agents beliefs set, the imperative code enclosed within the actuator associated to the symbolic string doX is executed.

The AFME platform architecture is shown in Figure 4.16. It comprises a scheduler, a group of agents, and several platform services needed for supporting, among the others, agents communication and migration. To improve reuse and modularity, actuators, perceptors, and services are prevented from

```
A0: storedDataCounter = 0;
    isOneShotRequest = isOneShot(request);
A1: Event sensing = new Event(agent.getId(), agent.getId(), request.getParam(ParamsLabel.SENSOR_TYPE),
    Event.NOW);
    agent.sense(sensing);
    addDispatcherRule(sensing);
A2: Event timer = new Event(agent.getId(), agent.getId(), Event.TMR_EXPIRED, Event.NOW );
    period = getPeriodTimer(request);
    lifetime = getLifetimeTimer(request);
    timer.setParam(ParamsLabel.LIFETIME_ELAPSED, "false");
    timerID = agent.setTimer(true, period, lifetime, timer);
    addDispatcherRule(timer);
    dataToStore = getDataToStore(request);
A3: storeData(event.getParam(SENSED_DATA));
    storedDataCounter++;
A4: if(request.getParam(ParamsLabel.DATA_TYPE) != "THRESHOLD_NOTIFICATION" ||
    isThresholdChecked(request, getStoredData())){
        Event msg = new Event(agent.getId(), agent.getCAId(), Event.MSG_TO_BASESTATION, Event.NOW);
        msg.setParam(ParamsLabel.MSG_TYPE, BM_DATA);
        setDataParams(msg, getStoredData());
        agent.send(agent.getId(), agent.getCAId(), msg, true);
    }
    storedDataCounter = 0;
A5: syntheticData = calculateSyntheticData(getStoredData(), request.getParam(ParamsLabel.SYNTHETIC_DATA_TYPE));
    if(request.getParam(ParamsLabel.DATA_TYPE) != "THRESHOLD_NOTIFICATION" ||
    isThresholdChecked (request.getParams(), syntheticData)){
        Event msg = new Event(agent.getId(), agent.getCAId(), Event.MSG_TO_BASESTATION, Event.NOW);
        msg.setParam(ParamsLabel.MSG_TYPE, BM_DATA);
        setDataParams(msg, syntheticData);
        agent.send(agent.getId(), agent.getCAId(), msg, true);
    }
    storedDataCounter = 0;
```

Fig. 4.17. The MAPS action of the SA's Sensing Request plane.

containing direct object references to each other. Actuators and perceptors developed for interacting with a platform service in one application can be used, without any changes to their imperative code, to interact with a different service in a different application. In the other way round, the implementation of platform services can be completely changed without having to modify the actuators and the perceptors. Additionally, the same platform service may be used within two different applications to interact with a different set of actuators and perceptors. So, all system components of the AFME platform are interchangeable because they interact without directly referencing one another.

4.8.2 Programming and architectural features comparison between MAPS and AFME

In this subsection, a comparison of the features provided by the two platforms is presented. In particular, in Table 4.6, MAPS and AFME are compared with respect to eight characteristics: agent behavior model, intentional agent support, agent behavior definition language, basic programming language, migration type, migration mechanism, agent communication model, and dynamic agent creation.

The most important difference between the two platforms is represented by the way through which agent-based applications are designed. Both platforms exploit the same basic programming language, but the agent model is quite different. In fact, MAPS uses a finite state machine model to define agent behavior whereas AFME employs a more complex BDI-like model.

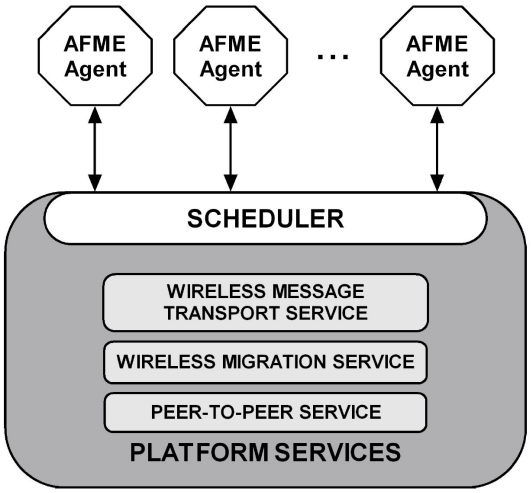


Fig. 4.18. The architecture of AFME.

Intentional agents are therefore only offered by AFME. As a consequence, the development of MAPS and AFME applications is based on different approaches. MAPS uses state machines to model the agent behavior and directly the Java language to define events, guards and actions. AFME uses a more complex model centered on perceptors, actuators, modules, and services which are developed in Java but have to be strictly correlated to the declarative rules provided for modeling the agent behavior. Both approaches are effective for developing agent-based applications even though MAPS is more straightforward as it relies on a programming style based on state machines widely known by programmers of embedded systems.

Agent migration, in terms of data and state of agents, is supported by both systems, but neither MAPS nor AFME allow for the migration of code due to the fact that CLDC-compliant devices (like Sun SPOT) cannot support dynamic class loading. In particular, MAPS uses a migration mechanism based on hibernation and serialization of the isolate within which an agent lives whereas AFME uses a proprietary agent descriptor to capture and transmit agent data and state.

The agent communication model adopted by both architectures for exchanging information is based on message passing (unicast and broadcast) which is the communication paradigm mostly used in agent-oriented frameworks.

The ability to create an agent at runtime could be an important feature for application in which the number of necessary agents cannot be determined a priori and simply fixed at compile-time. MAPS allows for such capability

Table 4.6. Comparison between MAPS and AFME features.

	MAPS	*AFME*
Agent Behavior Model	Finite State Machine	Belief/Desire/Intension
Intentional Agent Support	No	Yes
Agent Behavior Definition Language	Java	declarative rules (AFAPL)
Basic Programming Language	Java	Java
Migration Type	Weak	Weak
Migration Mechanism	Sun SPOT Isolate	Agent descriptor transmission
Agent Communication Model	Message passing	Message passing
Dynamic Agent Creation	Yes	No

whereas AFME needs agents to be created only in a static way so providing more flexibility for the creation of dynamic distributed applications.

Finally, differently from AFME, MAPS is specifically designed for WSNs and fully exploits the release 5.0 red of the Sun SPOT library to provide advanced functionality of communication, migration, sensing/actuation, timing, and flash memory storage.

4.8.3 Performance comparison between MAPS and AFME

This subsection is devoted to show the basic performance differences of the two agent platforms. In particular, the typical communication capabilities of agent architectures (data exchange among agents and agent migration) have been evaluated. Furthermore, the usage of the memory resource (both RAM and flash) has been measured.

To evaluate and compare the communication/migration performance of MAPS and AFME two benchmarks have been defined according to [104] for the following mechanisms:

- *Agent communication.* The agent communication time is computed for two agents running onto different nodes and communicating in a client/server fashion (request/reply). Two different request/reply schemes are used: (i) data Back and Forward (data BF), in which both request and reply contain the same amount of data; (ii) data Back (data B), in which only the reply contains data. Comparison results are shown in 4.19. For agents with light data payload, AFME performs better than MAPS; however, when the agent data payload overtakes 700 bytes MAPS starts performing better in the case data BF.

- *Agent migration.* The agent migration time is calculated for an agent ping-pong between two single-hop-distant sensor nodes. Migration times are computed by varying the data cargo of the ping-pong agent. The obtained migration times are high due to the slowness of the Squawk VM operations supporting the migration process. Comparison results are shown in 4.20. AFME retains a higher performance migration mechanism, as it is not based on the heavy isolate hibernation/serialization mechanisms of the Squawk VM.

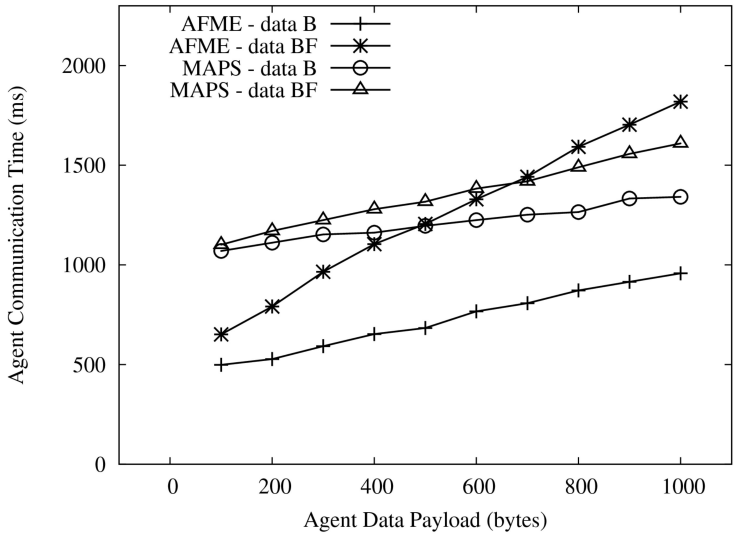


Fig. 4.19. Agent communication time comparison.

In Table 4.7 a comparison among the sensor-node-side agent-based runtime platform of MAPS and AFME with respect to RAM and Flash usage is reported. As can be seen, MAPS requires a bit less memory resources than AFME but both platforms show a relatively low usage of them. In particular, the used RAM is one of the most important parameter for an agent-based architecture atop a WSN node, so that the remaining memory can be exploited for the instantiation of the user-defined agents.

4.8.4 A case study: mobile agent-based remote monitoring

To analyze the effectiveness and performance of MAPS and AFME to support the programming of WSN applications, an agent-based remote monitoring application has been developed both in MAPS and AFME. The developed application involves two sensor nodes and consists of the following three interacting agents:

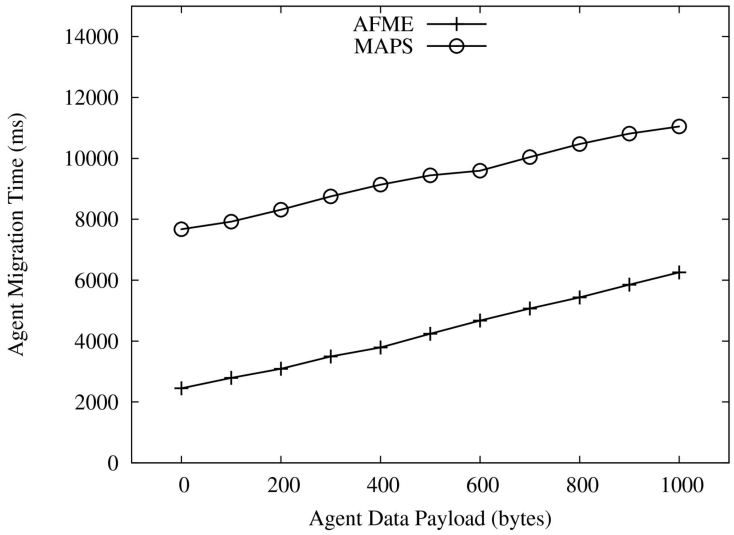


Fig. 4.20. Agent migration time comparison.

Table 4.7. RAM and Flash usage in MAPS and AFME.

	MAPS	*AFME*
RAM (platform memory occupation) used (KB) / available (KB) memory usage	85.8 / 512 16.76%	96.5 / 512 18.85%
Flash (platform code dimension) used (KB) / available (KB) memory usage	76.8 / 4096 1.87%	81.8 / 4096 2%

- DataCollectorAgent, which collects data related to the Sun SPOT node sensors (accelerometer, temperature, and light data);
- DataMessengerAgent, which carries collected sensed data from the sensing node to the basestation;
- DataViewerAgent, which displays the received collected data.

The sequence of interactions among the three defined agents is shown in Figure 4.21 through an M-UML sequence diagram [103].

The application execution is driven by the User that presses a switch on the Sun SPOT on which the DataViewerAgent is running. Upon the User event, the DataViewerAgent sends a remote message to the DataCollectorAgent (running on the sensing node) for starting its sensing operations. The

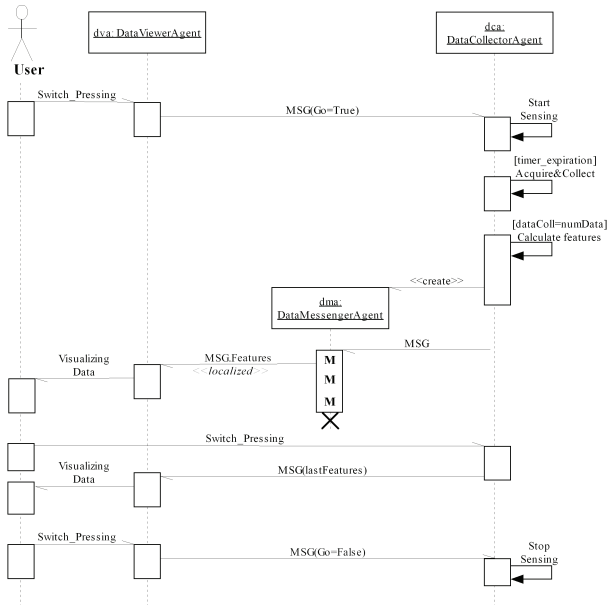


Fig. 4.21. M-UML sequence diagram for agents interactions.

agent therefore starts an internal timer to a particular value to begin its collecting activity: on timer expiration the agent acquires data from the onboard node sensors and collects them. As soon as the agent has acquired numData samples, it calculates one or more features (e.g. max, min and mean) for each of the sensor data types. Afterwards, the DataCollectorAgent creates the DataMessengerAgent that, with the computed features, migrates to the DataViewerAgent node (or basestation node) for data visualization. In case the User presses the switch on the node where the DataCollectorAgent is running, the agent sends an instantaneous message having the values of the last computed features. Finally, the remote monitoring activity terminates when the User presses again a switch of the Sun SPOT on which the DataViewerAgent is running.

In the following subsections we first describe the implementation of the DataCollectorAgent in MAPS and AFME and, then, discuss the results obtained from the performance evaluation of the MAPS and AFME implementations.

4.8.4.1 Agent definition in MAPS

As described in Section 4.4, MAPS agents are modeled through a multi-plane state machine. In Figure 4.22 the plane of the DataCollectorAgent is shown whereas the code of the actions (A0-A10) is reported in Figure 4.23. The

AGN_START event causes the transition from the agent creation state to the IDLE state with the execution of an initializing code represented by the action A0 (e.g. data structure initialization). In the IDLE state, when the network message sent by the DataViewerAgent arrives (see MSG event) and the guard [go==true] holds (i.e. the parameter go of the event is true), action A1 is executed and the agent transits to the WAITSENSING state. The action A1 is in charge of configuring and starting the timer for timing the sensor readings, and for configuring the agent plane so that it can be signaled by the SWT_PRESSED_RELEASED event when the user presses and releases (see the parameters setting) the switch identified by index "2". When the timer fires, the TMR_EXPIRED event is received and the sensing operations are requested (action A2) to the three onboard sensors. When each of the three sensor data is available (through the ACC_TILT, TMP_CURRENT and LGH_CURRENT events), their corresponding actions (A5, A6, A7) store the values on appropriate buffers. If numData samples for each sensor type have not been collected yet, the guard [dataColl!=numData] holds so that the agent returns to the WAITSENSING state waiting for the next sensing operations on timer expiration. On the contrary, if the necessary sample number is reached, sensor data are ready for being manipulated and results transmitted to the basestation node. So, the set of the features are computed and the DataMessengerAgent is created (action A9). When the AGN_ID event, containing the agent id of the created agent, is received the set of the features values are passed to it (action A10). Regarding the WAITSENSING and the DATACOLLECTING states, if the user presses and releases the switch (notified by the SWT_PRESSED_RELEASED event), the last set of computed features are immediately sent to the DataViewerAgent through a remote message (action A3). Finally, when the event MSG is received and the guard [go==false] holds, the agent is terminated (action A4).

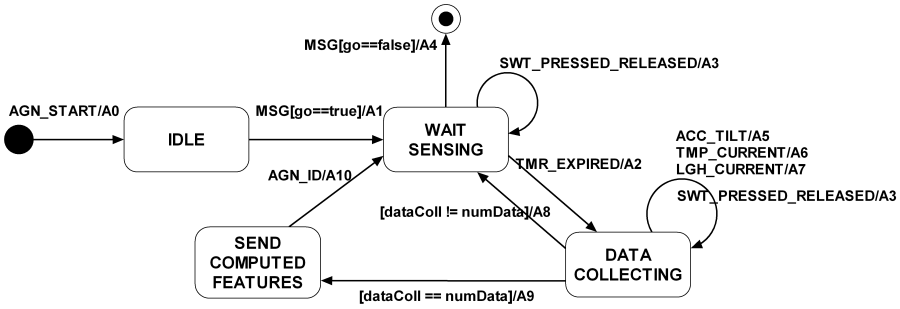


Fig. 4.22. MAPS-based DataCollectorAgent behavioral model.

In Figure 4.24, the architecture of the DataCollectorAgent is depicted. The simplicity of this architecture reflects the simplicity in developing a MAPS agent. The user-defined components, represented in grey color, are the Data-

```
A0: dataColl= 0;
    data= new double[3][numData];

A1: Event timer = new Event(this.agent.getId(),this.agent.getId(), Event.TMR_EXPIRED, Event.NOW);
    timerID = this.agent.setTimer(true, SAMPLING_TIME, timer);
    Event switchPressed= new Event(this.agent.getId(),this.agent.getId(),
                        Event.SWT_PRESSED_RELEASED, Event.PERMANENT);
    switchPressed.setParam(ParamsLabel.SWT_PRESSED, "false");
    switchPressed.setParam(ParamsLabel.SWT_RELEASED, "true");
    switchPressed.setParam(ParamsLabel.SWT_INDEX, "2");
    this.agent.input(switchPressed);

A2: Event temperature = new Event(this.agent.getId(), this.agent.getId(), Event.TMP_CURRENT,
                        Event.NOW);
    temperature.setParam(ParamsLabel.TMP_CELSIUS,"true");
    this.agent.sense(temperature);
    Event accel = new Event(this.agent.getId(), this.agent.getId(), Event.ACC_TILT, Event.NOW);
    this.agent.sense(accel);
    Event light = new Event(this.agent.getId(), this.agent.getId(), Event.LGH_CURRENT, Event.NOW);
    this.agent.sense(light);

A3: Event msg = newEvent(this.agent.getId(), dataViewerAgentID,Event.MSG, Event.NOW);
    msg.setParam("lastFeatures", this.computedFeatures);
    this.agent.send(this.agent.getId(), dataViewerAgentID, msg, true);

A4: this.terminateAgent();

A5: data[0][dataColl]= Double.parseDouble(event.getParam(ParamsLabel.ACC_TILT_X_VALUE));

A6: data[1][dataColl]= Double.parseDouble(event.getParam(ParamsLabel.TMP_TEMPERATURE_VALUE));

A7: data[2][dataColl]= Double.parseDouble(event.getParam(ParamsLabel.LGH_LIGHT_VALUE));

A8: dataColl++;

A9: computeFeatures(data);
    this.agent.create("applications.demo.Messenger",null,
                      this.agent.getMyIEEEAddress().asDottedHex());
    dataColl= 0;

A10: Event msg = new Event(this.agent.getId(), messengerAgentID, Event.MSG, Event.NOW);
     msg.setParam("features", this.features);
     this.agent.send(this.agent.getId(), messengerAgentID, msg, true);
```

Fig. 4.23. Java code of the actions of the DataCollectorAgent plane.

CollectorAgent, which inherits from the basic Agent class of MAPS, and its only associated DataCollectorPlane, derived from the basic MAPS Plane class and modeling the agent behavior driven by (1 or more) Event objects.

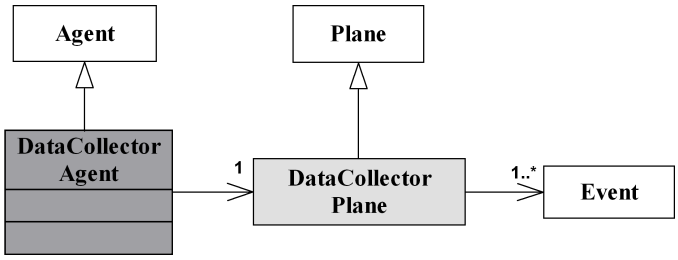


Fig. 4.24. Architecture of the MAPS-based DataCollectorAgent model.

4.8.4.2 Agent definition in AFME

As described in Section 4.8.1, AFME agents have to be defined by providing both a declarative code (i.e. a set of rules declared into a script file), representing the agent behavior, and a set of Java classes strictly correlated to these rules and constituting the preceptor, actuator and module components.

The set of rules defining the DataCollectorAgent behavior are defined as follows:

```
1.   message(inform, sender(dataViewer, addresses(?addr), ?content)),
          !goTrue > par(timerActivatorAct, adoptBelief(always(goTrue)));
2.   timerExpired, goTrue > par(timerActivatorAct, activateSensorsAct);
3.   numDataSampled(?nSamples), #?nSamples>9 > computeFeaturesAct;
4.   featuresComputed(?computedFeatures) > inform(agentID(messenger,
          addresses("radiogram://"+ DataMessengerAgentNodeAddr)),
          ?computedFeatures);
5.  switchPressed(?computedFeatures) > inform(agentID(dataViewer,
          addresses("radiogram://"+ DataViewerAgentNodeAddr)),
          ?computedFeatures);
6.   message(inform, sender(dataViewer, addresses(?addr), ?content)),
          goTrue > retractBelief(always(goTrue));
```

Rule 1 enables the timer for sensing (through the `timerActivatorAct` actuator) and creates the `goTrue` belief upon the reception of the message coming from the DataViewerAgent (see the message belief) and if `goTrue` is not yet an agent belief. The *par* instruction is an AFME construct to be adopted when more actuators have to be executed in parallel, whereas the *always* construct defines a belief that has to be maintained into the beliefs set of the agent throughout its execution, until it will be retracted (see rule 6).

Rule 2 states that the sensor reading operations are activated (`activateSensorsAct` actuator) and also the timer is reactivated (`timerActivatorAct` actuator) after timer expiration (`timerExpired` belief).

Rule 3 checks if 10 samples have been acquired for each sensor type. When the `numDataSampled` belief is adopted, it returns the number of samples sensed into the `?nSamples` variable value of which is tested for confirming if the computation of features has to be started through the execution of the computeFeaturesAct actuator.

Rule 4 verifies that features have been correctly computed (i.e. the `featuresComputed` belief is adopted) and if this holds, the content of the `?computedFeatures` variable returned along with the belief is sent to the DataMessengerAgent through a radio message generated by the AFME inform actuator.

Rule 5 states that if the `switchPressed` belief is adopted (i.e. the user presses the switch) a message with the last computed features (the `?computedFeatures` variable returned by the belief) is sent to the DataViewerAgent.

Rule 6 is for checking the reception of a message from the DataViewerAgent and, since the `goTrue` belief is still adopted by the agent (see rule 1), the timer, which paces the sensing operations, is disabled by simply retracting the `goTrue` belief.

After the definition of the agent behavior by means of the declarative code, several Java components have to be specified with the purpose of giving support to the execution of these rules at run-time. In Figure 4.25 the architecture of the AFME-based DataCollectorAgent is depicted, which contains the supporting perceptor, actuator and module components.

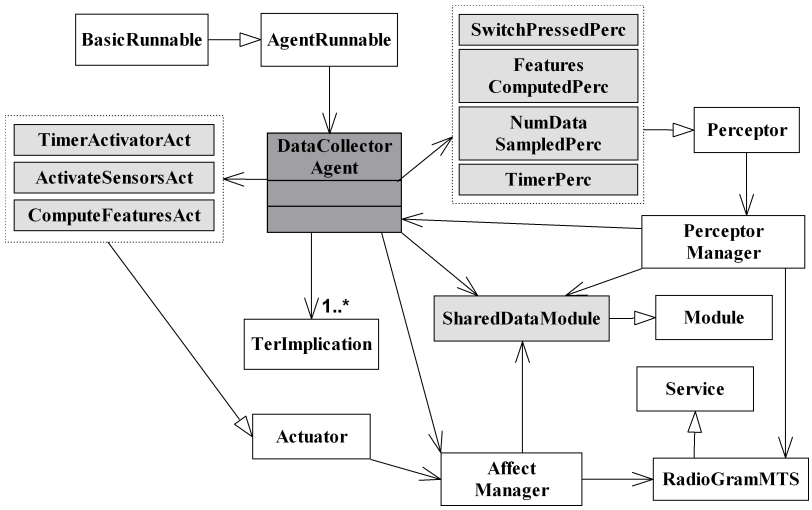


Fig. 4.25. Architecture of the AFME-based DataCollectorAgent model.

The white-colored components are basic classes of the AFME framework, whereas the grey ones represent the Java-based supporting components that have been specifically developed in this application. In particular, such components are:

- *Four perceptors.* Perceptors are responsible for generating the proper beliefs by monitoring the agent status and its environment. NumDataSampledPerc is needed for retrieving the number of sensor data samples that have been already collected whereas FeaturesComputedPerc checks that all features have been computed on the collected sensor samples. Finally, TimerPerc checks when the timer expires and SwitchPressedPerc recognizes the switch pressing by the user.
- *Three Actuators.* Actuators are in charge of generating the proper actions in response of the generation of beliefs. TimerActivatorAct is used to activate a timer of the sensor node for timing the sensor acquisition

operations. The ActivateSensorsAct is needed to enable sensors reading after timer expiration. ComputeFeaturesAct is responsible for the actual feature computation on the collected sensor data and for reinitializing all necessary data structure after a completed processing operation.

- *One Module.* SharedDataModule has been defined as a shared memory space which not only stores the data samples sensed from the node sensors (accelerometer, temperature, light) but also all other necessary data to be shared among perceptors and actuators.

Finally, it is worth noting that the TerImplication component represents the auto-generated Java code provided as output of the AFME pre-compiler that is executed on the script files containing the agent behavior rules and specifying the agent platform (see below).

In the following, we describe the SharedDataModule (Figure 4.26) module, the NumDataSampledPerc perceptor (Figure 4.27) and the ActivateSensorsAct actuator (Figure 4.28) of the DataCollectorAgent to provide some examples of module, perceptor and actuator programming.

The SharedDataModule (see Figure 4.26) is developed with the purpose of offering a common memory space for storing information shared among preceptors and actuators. In particular, the data structure needed for collecting the sensor data, for counting the acquired samples and for storing the feature extraction results are managed through this component.

In its constructor method, the module registers itself with the "shared-data" identifier so that it can be indirectly referenced by the other components (actuators and perceptors). Moreover, it provides two different methods: processPer and processAction. The former processes requests coming from the preceptors whereas the latter processes requests coming from the actuators. In both cases, the proper operation to accomplish is determined by an *id* value identifying the requesting operations. For the sake of space, not the whole module code is shown but only those parts devoted to managing requests coming from the NumDataSampledPerc perceptor and the ActivateSensors-Act actuator, which are discussed later. The excerpt of the processPer method shown in Figure 4.26 is configured to return the current sample counts related to the accelerometer, light or temperature sensors, depending on which operation *id* (5, 8 or 9) is passed as actual parameter. The code excerpt of the processAction is, conversely, devoted to sensor data acquisition, storing and counter update related to the accelerometer (id=6), light (id=9) or temperature (id=10) sensors. It is worth noting that, the FOS objects are used in AFME for representing a generic kind of information, be it a simple value or a more complex object like a belief.

The AFME perceptors are in charge of generating the appropriate beliefs on the basis of the agent status information through the perceive method, which is called every BDI-cycle execution. In particular, the NumDataSampledPerc perceptor (see Figure 4.27) first requires the SharedDataModule

```
public class SharedDataModule extends Module{

    private long activationTimerInstant, timerSampling;
    private int temperatureSensorActivated, lightSensorActivated,
accSensorActivated,
     accSensorValuesXCounter, temperatureSensorValuesCounter,
lightSensorValuesCounter;
    private double [] temperatureSensorValues, lightSensorValues, accSensorValuesX;
    private double temperatureSensorExtractedValues,
                   lightSensorExtractedValues, accSensorExtractedValuesX;
    private int dataReady, outputCount;

    public SharedDataModule(AgentName name){
        super("shareddata");
        //code for data structure initialization has been omitted
        //for shortness
    }

    public FOS processPer(int id)throws MalformedLogicException{
           ..................
        }else if(id == 5){
            return FOS.createFOS(""+accSensorValuesXCounter);
        }else if(id == 8){
            return FOS.createFOS(""+lightSensorValuesCounter);
        }else if(id == 9){
            return FOS.createFOS(""+temperatureSensorValuesCounter);
        }
           ..................
        return null;
    }

    public FOS processAction(int id, FOS data) throws MalformedLogicException{
           ..................
        }else if(id==6){
            accSensorValuesX[this.accSensorValuesXCounter] =
                                Double.parseDouble(data.toString());
            this.accSensorValuesXCounter++;
            return null;
        }else if(id==9){
            lightSensorValues[this.lightSensorValuesCounter] =
                                Double.parseDouble(data.toString());
            this.lightSensorValuesCounter++;
            return null;
        }else if(id==10){
            temperatureSensorValues[this.temperatureSensorValuesCounter] =
                                Double.parseDouble(data.toString());
            this.temperatureSensorValuesCounter++;
            return null;
        }
           ..................
        return null;
    }

}
```

Fig. 4.26. Code excerpt of the SharedDataModule component.

(identified by the shareddata identifier used as first parameter of the perManage method) to return the samples counter value related to each of the three different sensor types, through method calls having as differences the operation id passed as second parameter. The return values are represented as FOS object. Afterwards, if all the three counters hold a same value (meaning that a sensing operation for each sensor is completed), the numDataSampled belief is adopted and the counter value is assigned to the ?*nSamples* variable (see Rule(3)).

```
public class NumDataSampledPerc extends Perceptor{
    private PerceptionManager manager;
    public NumDataSampledPerc(PerceptionManager manager){
        super(manager);
        this.manager=manager;
    }
    public void perceive(){
        FOS contAccXFOS = manager.perManage("shareddata", 5);
        FOS contLightFOS = manager.perManage("shareddata", 8);
        FOS contTempFOS = manager.perManage("shareddata", 9);

        int contAccX = Integer.parseInt(contAccXFOS.toString());
        int contLight = Integer.parseInt(contLightFOS.toString());
        int contTemp = Integer.parseInt(contTempFOS.toString());

        if( contAccX!=0 && (contAccX==contLight)
                        && (contLight==contTemp) ) {
            this.adoptBelief("numDataSampled("+ contAccX +")");
        }
    }
}
```

Fig. 4.27. Code of the NumDataSampledPerc perceptor.

```
public class ActivateSensorsAct extends Actuator {
    private AffectManager m;
    public ActivateSensorsAct(AffectManager manager) {
        super(manager,"activateSensorsAct");
        m=manager;
    }
    public boolean act(FOS arg0) {
        String valueX= ""; String valueLight= ""; String valueTemp= "";
        try {
            valueX = Double.toString(
                  EDemoBoard.getInstance().getAccelerometer().getTiltX());
            valueLight = Double.toString(
                  EDemoBoard.getInstance().getLightSensor().getValue());
            valueTemp= Double.toString(
                  EDemoBoard.getInstance().getADCTemperature().getCelsius());
        } catch (IOException e) {
            e.printStackTrace();
        }
        FOS fX= FOS.createFOS(
                  valueX.substring(0, Math.min(5,valueX.length())));
        FOS fL= FOS.createFOS(
                  valueLight.substring(0, Math.min(5,valueLight.length())));
        FOS fT= FOS.createFOS(
                  valueTemp.substring(0, Math.min(5,valueTemp.length())));
        m.actOn("shareddata",6,fX);
        m.actOn("shareddata",9,fL);
        m.actOn("shareddata",10,fT);
        return true;
    }
}
```

Fig. 4.28. Code of the ActivateSensorsAct actuator.

The ActivateSensorsAct component (see Figure 4.28) is defined to perform the sensing operations and store acquired data into the SharedDataModule. In particular, the Sun SPOT EdemoBoard object is employed to acquire a sample data from each of the three different onboard sensors. Each value is then converted to string, trimmed to max 5 characters and stored into the SharedDataModule through the actOn method having the following parameters: the identifier of the module component (shareddata), the *id* operation that has to be performed by the module and the FOS object containing the data string to be stored.

4.8.4.3 Performance evaluation

The aim of the performance evaluation is to analyze memory usage and time parameters related to the developed MAPS- and AFME-based applications and to compare the results obtained from the analysis of the two different implementations. Memory and time are, of course, the two most critical computational resources that should be carefully analyzed to evaluate the efficiency of the agent frameworks to manage the agent lifecycle through their own run-time system based on the Sun SPOT platform.

The evaluation involves: (i) the analysis of the memory usage, both RAM (the occupation of memory at run-time in both static and dynamic conditions) and Flash (the agent code dimension); (ii) the timing analysis of the operation sequence (sensing, feature computation and mobile agent migration) of the agent-based application.

The RAM occupation and the code dimension are first evaluated with respect to the agents (DataViewer, DataCollector and DataMessenger) defined in the application and then added to those of their supporting framework to obtain the RAM and Flash usage in both sensors (basestation and sensing sensor) under static conditions (i.e. at application initialization). In particular, the sensing node is the one executing the DataCollector agent and on which the Messenger agent is initially resident, whereas the basestation node is the one executing the DataViewer agent. For both sensors, the amount of memory occupation, RAM and Flash, has been evaluated at application startup, by excluding one at a time the definition / creation code of each of agents constituting the application. The evaluation results are reported in Table 4.8 and Table 4.9. Results show that MAPS performs slightly better than AFME for both RAM and Flash resources.

An evaluation of the RAM usage under specific dynamic conditions can provide important information on the efficiency of the agent frameworks. To this purpose, the RAM usage at the sensing sensor node is evaluated (see Figure 4.29) by varying the sampling time for data acquisition from low (25ms) to high values (5s). The variation of the sampling time not only affects the rate of the sensing operation but also the rate of feature computation and

Table 4.8. RAM occupation and code dimension of the application agents.

		DataViewer	*DataCollector*	*Messenger*
Code dimension (KB)	MAPS	1.76	3.09	1.3
	AFME	1	2.1	1.6
RAM occupation (KB)	MAPS	2.8	2.76	25.4
	AFME	10.6	25.19	23.6

Table 4.9. RAM and Flash usage of the whole application (platform+application agents).

		MAPS	*AFME*
Flash usage used (KB) / available (KB) %	Sensing node	81.19 / 4096 1.98%	85.5 / 4096 2.08%
	Basestation node	78.56 / 4096 1.92 %	82.8 / 4096 2.02 %
RAM usage used (KB) / available (KB) %	Sensing node	88.6 / 512 17.3 %	107.1 / 512 20.92 %
	Basestation node	113.66 / 512 22.2 %	145.29 / 512 28.38 %

transmission so increasing (at high rates) or decreasing (at low rates) the application computational workload

The obtained results show that both MAPS and AFME provide a quite constant RAM usage; this demonstrate that memory management is not affected by heavy workloads. Moreover, MAPS performs better than AFME; in fact, RAM usage in MAPS is about 20% lower.

The purpose of the timing analysis is to identify parameters leading to bottlenecks possibly affecting the execution of the agent-based application. As the application consists in a continuous cycle of the following sequence of operations (sensor data collection, feature computation and migration of the agent carrying the computed features), the time of each cycle (hereafter called application iteration) can be calculated and its trend over the iterations analyzed. In particular, three different execution profiles have been considered: (i) high sampling rate (25ms and 50ms) to analyze systems in which very frequent sensor data acquisition and computation are needed; (ii) medium sampling rate (100m and 300ms) to analyze systems with moderate dynamics; (iii) low sampling rate (1s and 2s) to take into account systems that need slow sensing operations. For each of the defined execution profiles, the evaluation has been carried out both considering and not considering the time needed for agent migration. Moreover, each test is carried out by executing

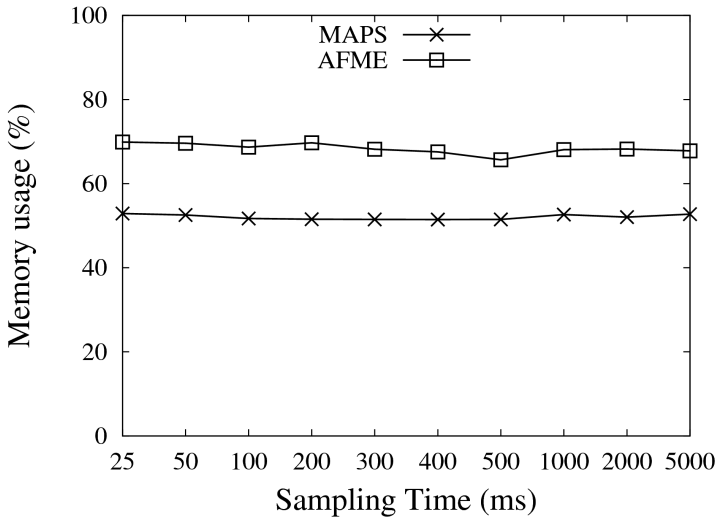


Fig. 4.29. Memory usage comparison by varying the sampling time.

the application for 100 iterations and considering *numData* samples (acquired from sensors before feature computation) = 10. Finally, results are obtained by averaging over 15 runs for each test.

Figure 4.30 shows the performance of the application running on MAPS and AFME for high sampling rates (25ms and 50ms) without considering agent migration. The results are quite interesting because it is evident that both platforms experience high times (from 4 to 10 times the estimated expected time) for completing an application cycle. The expected time could be estimated by considering the expression: $expected\_time = sampling\_time * samples + feature\_computation\_time\_estimation$.

Moreover, considering the pair of curves related to a same platform, only a slight time difference can be appreciated. Indeed, this experimental result highlights that the Sun SPOT device suffers from an intensive computational workload due to too frequent requests of the sampling operation. Thus, the poor performance results do not mainly derive from the run-time architectures of the agent frameworks but rather from the Sun SPOTs. However, the decreasing time performance over time is more visible in MAPS, which is more affected by the problems that the sensor device shows on high sampling requests, i.e. a relevant time delay that cumulatively impacts the next application iteration.

If agent migration is considered (see Figure 4.31), the migration operation implies a progressive worsening of the application iteration time during the application execution for both platforms, independently from the actual migration mechanism adopted. On one hand, the absolute difference in time

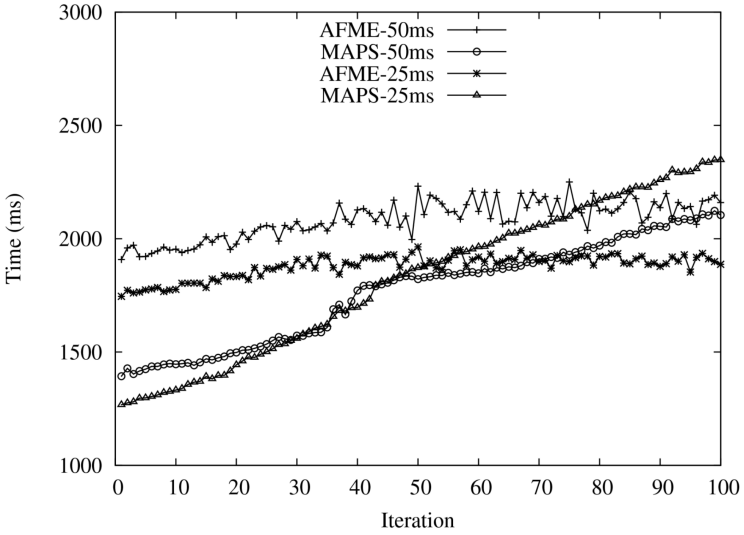


Fig. 4.30. Timing performance for high sampling rate without considering agent migration.

between MAPS and AFME is imputable to the low performance of the isolate-based agent migration; on the other hand, the time drift only depends on the high sample rates.

In Figure 4.32 higher sampling times (100ms and 300ms) have been considered. Also in these cases, both platforms show a percentage of time overhead in completing an application iteration. This is much less evident in MAPS than in AFME, especially at a sampling time of 100ms (with more than 50% of average overhead above the expected time for MAPS, and more than 100% for AFME). Consequently, the agent-based application developed for MAPS performs better than the one based on AFME.

As a confirmation of the discussed motivation related to Figure 4.31, in Figure 4.33 a significant better situation for both platforms is reported as a consequence of higher sampling times when agent migration is considered. In particular, the performance degradation over cycle iterations is much less evident in MAPS than in AFME, despite the fact that AFME shows lower absolute values than MAPS. Moreover, AFME shows a worse drift than MAPS, so that on long application execution it will have higher absolute application cycle completion times than MAPS.

The timing overhead becomes less and less evident with the increasing of the sampling time, as it is shown in Figure 4.34, where the performance results are demonstrated to be very similar to the expected values.

The decreasing of overhead, not in percentage but in absolute value, demonstrates that this is not due to a fixed performance cost, but that it

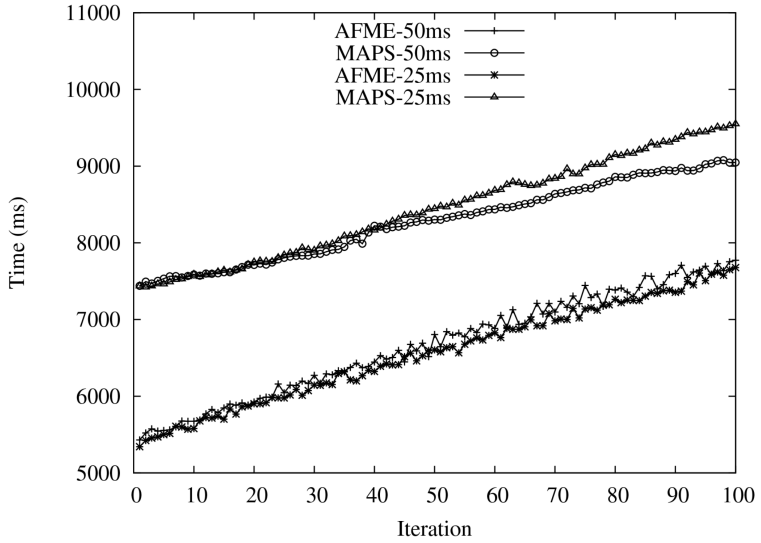


Fig. 4.31. Timing performance for high sampling rate considering agent migration.

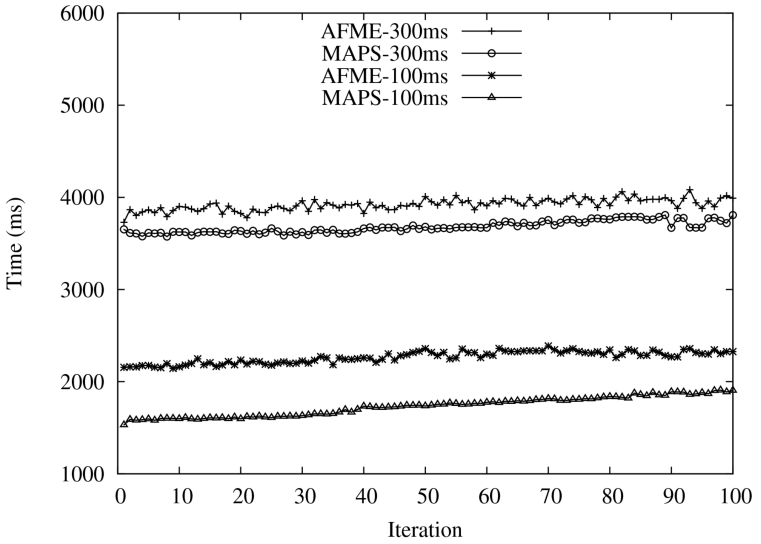


Fig. 4.32. Timing performance for sampling rates of 100ms and 300ms without considering agent migration.

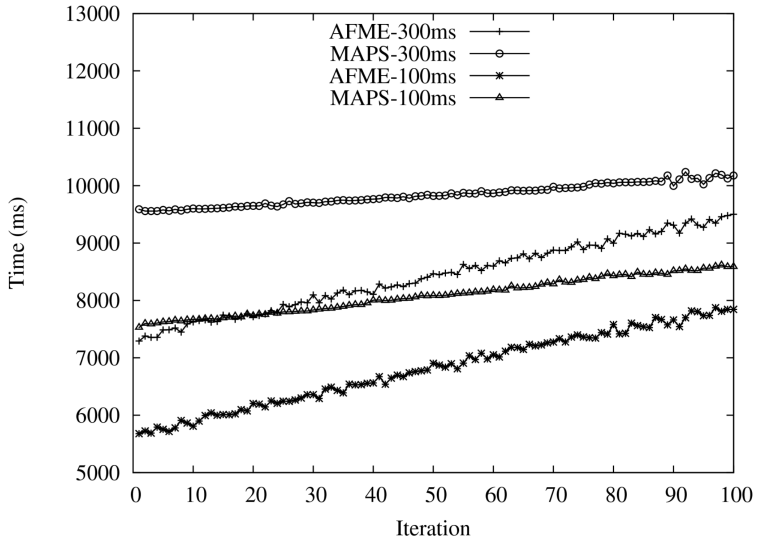


Fig. 4.33. Timing performance for sampling rates of 100ms and 300ms considering agent migration.

actually depends on a periodic little overhead cumulated for each sensor sampling operation.

Finally, when considering agent migration, for higher sampling times (see Figure 4.35), MAPS maintains a practically constant cycle completion time during application running, meaning that migration operations do not affect execution performance. On the contrary, AFME is still affected by an increasingly performance degradation, showing that a little cumulating delay continues to be present over time.

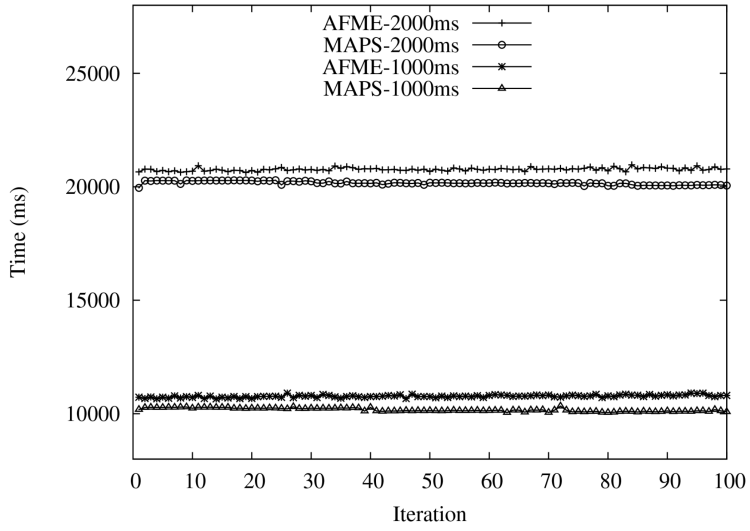


Fig. 4.34. Timing performance for sampling rates of 1000ms and 2000ms without considering agent migration.

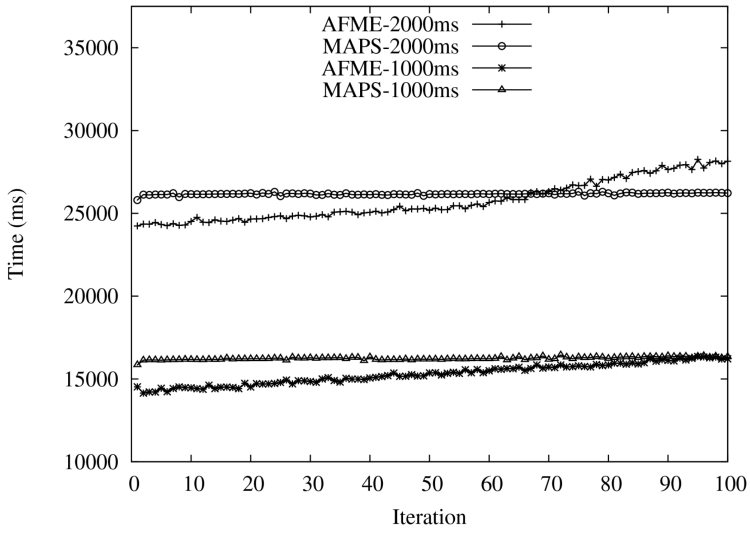


Fig. 4.35. Timing performance for sampling rates of 1000ms and 2000ms considering agent migration.

5

A Timer-Driven Framework for WBSN

Wireless sensor networks (WSNs) are a novel technology enabling new classes of applications and systems for ubiquitous and pervasive computing. In particular, WSNs for the human body, also known as Wireless Body Sensor Networks (WBSNs), will enable not only continuous, multi-purpose monitoring of people but also will support social interaction among people coming into physical contact. In these contexts, applications demand a wide range of functionalities, in terms of sensor types, processing performance, communication capabilities. Moreover the development of such applications has to deal with the issue of handling heterogeneous WBSNs since different kinds of sensor node architectures could be necessary to fulfill all the application requirements. This chapter proposes an approach based on the SPINE framework for the programming of signal processing applications on heterogeneous wireless sensor platforms.

5.1 Introduction

Wireless body sensor networks (WBSNs) have great potential to enable a broad variety of assisted living applications such as health and activity monitoring, and emergency detection. It is therefore important to provide design methodologies and programming frameworks which enable rapid prototyping of collaborative WBSN applications [112]. Although several effective application development frameworks already exist for WBSNs based on specific sensor platforms (e.g. CodeBlue [113], SPINE [80] [64], Titan [114]), effective methods for platform-independent development of WBSN applications which would enable rapid development of multi-platform applications and fast application porting from one platform to another, are still missing or in their infancy. In fact, the aforementioned frameworks can be only used to effectively develop WBSN applications for TinyOS-based sensor platforms. Thus, to develop applications for new sensor platforms, such frameworks should be

implemented for each new sensor platform to be exploited. This not only increases development efforts but also enforces developers to become skilled on the low-level programming abstractions provided by a new employed sensor platform.

This chapter is organized as follows: (Section 5.2) introduces some WBSN related work; Section 5.3 analyzes the evolution of SPINE; Section 5.4 categorizes and discusses interesting approaches which can effectively support platform-independent development of WSN applications [115]; Section 5.5 shows the current efforts (design and current implementation status) towards the definition of SPINE2, an evolution of the SPINE (Signal Processing In-Node Environment) framework [80] [64] based on the C-language, which aims at supporting the development of platform independent assisted living applications. The goal is to reach a very high platform independency for C-like programmable sensor platforms (e.g. TinyOS [19], Ember [116], ZStack [117]) and raise the level of the provided programming abstractions from platform-specific to platform-independent. Finally, Section 5.6 presents a case study that tests the effectiveness of the SPINE2 framework.

SPINE2 [118] offers a task-oriented model for programming the sensor nodes of a collaborative WBSN. In particular tasks (e.g. sensing, feature extraction, aggregation, data transmission) can be dynamically discovered, created, activated, scheduled and controlled by the coordinator on each sensor node in order to fulfill a goal-directed overall task of the distributed system implemented by the network of sensor nodes. Dynamic distribution of tasks allows among the others preprocessing of sensed data directly on the node, a significant reduction of data transmission and battery consumption, and an overall increase of the network lifetime. Different tasks can be assigned to each node and tasks can be controlled at execution time via proper message exchange; in this way the network can overall adapt to changes in context, overall goals, state of each single node, and it can better balance load and task types between each element of the network. Such a task-oriented model enables a holistic approach where the WBSN capability becomes higher than the sum of the capabilities of each element.

Aim of the chapter is the discussion of how such a model was implemented in resource-constrained environments, and what architecture and approach was selected in order to achieve platform independency and provide high level programming abstractions that reduce time-to-market for tiny environments.

5.2 State-of-the-art and Related Work

Several research projects in the academia and the industry have recently focused on the use of WBSNs including sensors such as pulse monitors, accelerometers, gyroscopes, and pulse oxymeters. Common to most WBSN applications is a network architecture composed of a coordinator node and sensor

nodes connected in a star topology. Sensor nodes transmit raw or interpreted data to the coordinator node, which may include algorithms for data analysis and interpretation. In remote monitoring applications the WBSN is connected through a gateway to a wide area network to allow doctors and caregivers access the patients' data. Requirements vary greatly depending on the scenario being considered, and may include: 1) *reliability*, especially for applications monitoring vital parameters; 2) *privacy and security* to ensure that only authorized people, e.g. relatives and caregivers, can access information on personal health or activities; 3) *latency*, especially in life emergency scenarios; 4) *low power consumption* to maximize battery lifetime; 5) *wearability* to allow patients carry sensor nodes in their daily life; 6) *extensibility* to other sensors and services, e.g. when new health care needs arise; 7) provisioning of service across locations to support *continuous monitoring*.

Health care and assisted living systems often require sophisticated algorithms to analyze the sensor data and extract higher-level information relevant to the user. For example, WBSNs are commonly used for the recognition of physical activities or health conditions of a patient. Figure 5.1 shows the block diagram of a typical recognition system that consists of offline training, runtime sensor data collection, feature extraction and classification steps [119] [120].

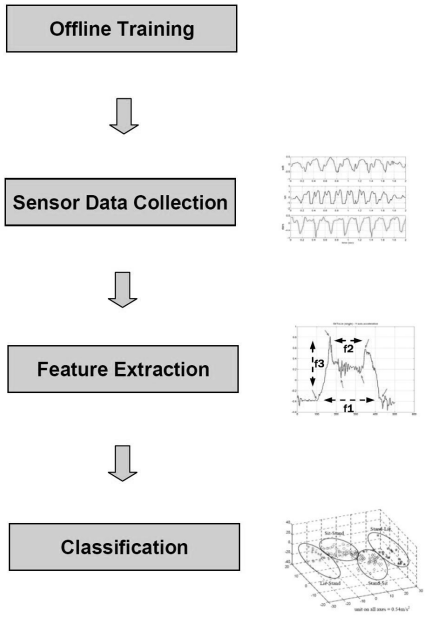


Fig. 5.1. Activity Recognition Systems.

Classification and pattern recognition techniques [119] have been proposed and used in several application domains. However, their real-time implementation on WBSNs adds additional challenges because WBSNs are very resource-limited in terms of battery power, memory, and computing power. To meet the requirements, designers must carefully evaluate implementation tradeoffs for example before allocating tasks across nodes. This requires a flexible design environment based on proper abstractions that hide low-level details to application developers and at the same time allow to easily quantify the performance of the system.

Hence, an important issue to be addressed is how to develop applications for WBSNs by minimizing application development efforts. An effective solution, which is being currently proposed by several research efforts, is the exploitation of domain-specific frameworks which facilitate application development in specific application domains [121]. It is in fact difficult incorporating application knowledge into the design of general-purpose middleware or into general purpose application frameworks. By focusing on a particular application domain (e.g. activity monitoring), these challenges are somehow mitigated by limiting the scope of applications to be built using the framework. In addition, this usually leads to the creation of frameworks that are more lightweight and therefore better fit the constrained resources of WBSNs.

In [122] a signal-processing-oriented framework, called Titan (Tiny Task Network), is proposed. The goal of Titan is to provide a mechanism to define and dynamically configure a network of data processing tasks on a WSN. In particular, Titan, which is developed atop TinyOS, allows to define a set of tasks, each of which implements some signal processing function, linked through connections, which transport the data from one task to another, to form a task network, which describes the application to be run on the WSN. Titan has been applied for activity recognition based on SensorButton devices equipped with accelerometers [114].

In [113] the CodeBlue framework for developing WBSN based health care monitoring applications is described. CodeBlue, which is implemented in nesC/TinyOS at the sensor-side and in Java at the base station-side, provides protocols for device discovery, publish/subscribe multihop routing, and a simple query interface allowing caregivers to request data from groups of patients.

Several other research efforts on developing WBSN applications for human activity monitoring have been to date proposed [120] [123] [124]; however most of them are not based on flexible frameworks providing in-node processing functions but mainly rely on raw data acquisition from sensors and on the application of feature extractors and classifiers for activity recognition at the base station-side.

5.3 From SPINE to SPINE2

SPINE (Signal Processing in Node Environment) [80] [125] is a software framework for the design of collaborative Wireless Body Sensor Network (WBSN) applications. It provides programming abstractions, APIs and libraries of protocols, utilities and data processing functions which simplify development of distributed signal processing algorithms for the analysis and the classification of sensor data. SPINE is distributed in Open Source under the LGPL license to facilitate establishing a broad community of users and developers that contribute to the scientific evolution of the framework with new capabilities and applications.

SPINE framework is constituted by two distinctive parts: a node side runtime system residing on the sensor nodes and a Java application, the coordinator, residing on a PC and having functionalities such as nodes configuration, data gathering and data analysis.

To date, two releases of SPINE are available:

- The version 1.2 which supports different kinds of sensor platforms running the TinyOS [19] operating system (supported platforms are TelosB, MicaZ, Shimmer).
- The version 1.3 which provides support for Z-Stack [117] so allowing the development of WBSN applications according to the ZigBee standard [11]. In particular, ZStack is the implementation of the ZigBee stack carried out by Texas Instruments.

In the following subsections the characteristics of each version of SPINE and the feasibility of integrating both of them into a single heterogeneous WBSN are explained in details.

5.3.1 SPINE 1.2

The software architecture of the node side part of the framework is reported in Figure 5.2. It is composed of a set of nesC components forming the runtime system which relies on the components provided by TinyOS for accessing the hardware resources, such as radio, sensors and timers. More specifically, the *SPINEApplication* is the core of the framework and is responsible for managing the overall system. The *PacketManager* allows the reception/parsing and the formatting/sending of application-level messages over the network through the *RadioController* that, in turn, relies on the communication interfaces of TinyOS. The *SensorsBoardController* provides accessing to the integrated sensors of the node, whereas the *FunctionManager* provides processing capabilities for data pre-elaborations useful for avoiding battery consumption due to the excessive raw data transmission. This component manages a set of functions already implemented in the release [125], but it is possible to easily extend the framework with other ones, on the basis of the user application requirements.

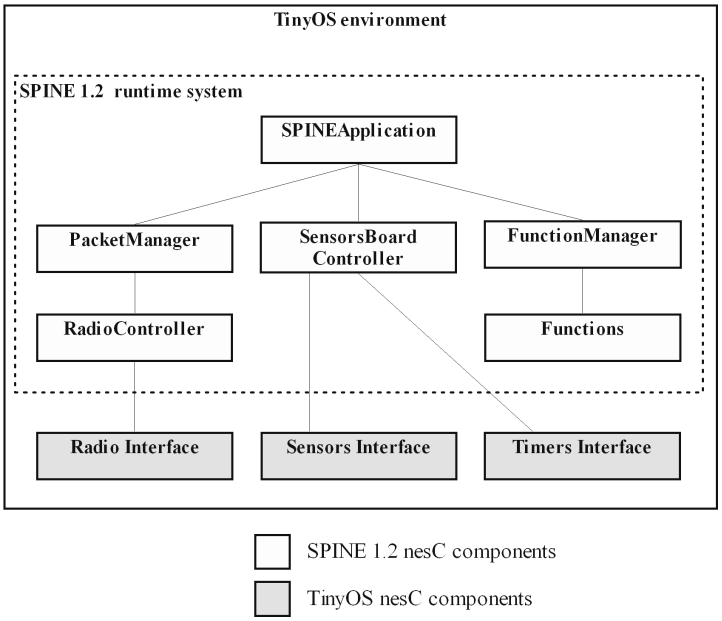


Fig. 5.2. The SPINE 1.2 node side framework.

5.3.2 SPINE 1.3

In the SPINE 1.3 architecture, shown in Figure 5.3, most of the components have similar functionalities as the ones in the version 1.2. However, the implementation of these components, in C language, is dependent on the Z-Stack. One of the principal differences from the previous version is related to the communication management. In fact, in order to be executed on the Z-Stack environment, the framework has to be ZigBee compliant. So, all the application-level messages have to be encapsulated in a message packet formatted according to the protocol stack specifications. This is performed through the use of APIs provided by the *ZigBee Device Object* of the Z-Stack system.

Another difference is that the *SensorManager* does not access sensors through some system interface, but by using the *Driver* module which provides direct access to the hardware sensors. Every application developed for the Z-Stack architecture has to be implemented as a set of tasks, so the SensorManger relies on the *Task Creation System* to create the necessary sensing tasks. The *FunctionManager* manages the processing functions on the node and makes use of the *Task Synch System* to request result transmission to an appropriate communication task. The *BufferPool* takes charge of providing a set of buffers in which sensed data and function results are stored.

It should be specified that *Task Creation System*, *Task Synch System*, *Timers* and *Memory Management* are not real Z-Stack components but func-

tionalities provided by the OSAL (Operating System Abstraction Layer) APIs of the Z-Stack system [117].

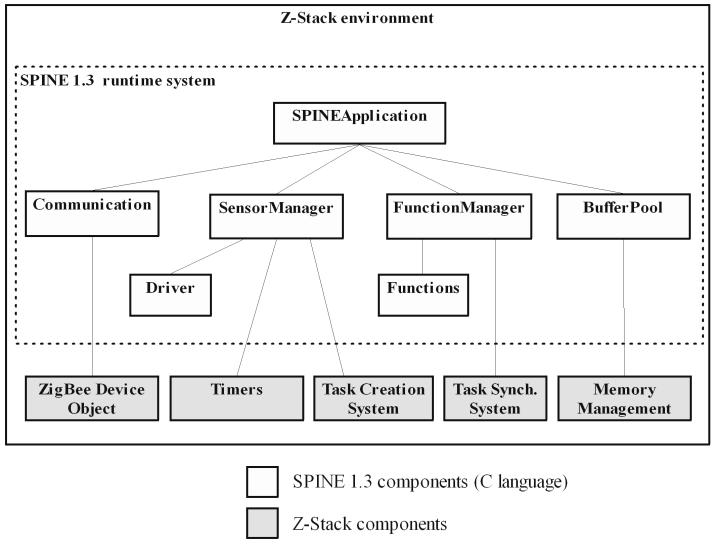


Fig. 5.3. The SPINE 1.3 node side framework.

5.3.3 Heterogeneous Programming

The SPINE coordinator, which runs at the base station side, is able to interact both with TinyOS sensor nodes and Z-Stack sensor nodes (as shown in Figure 5.4). This allows to build a WBSN composed of heterogeneous nodes which however should be programmed by using the node side SPINE implementation for each specific node. So, while this approach allows to use heterogeneous sensors in the same WBSN, different types of sensors must be differently programmed [126].

In order to perform its functionalities, the coordinator has to be interfaced, via USB cable, to one of each sensor type (TinyOS and Z-Stack) getting the appropriate radio communication capabilities for communicating with two different parts of the WBSN (it is worth noting that TinyOS sensor platforms supported by SPINE 1.2 use the IEEE 802.15.4 standard while Z-Stack uses the ZigBee standard). Nevertheless the high-level communication service is the same as it uses a unique format for application-level messages used for nodes configuration and information exchange.

Furthermore, the coordinator can be either local or remote; in fact, the SPINE 1.3 implementation of the coordinator supports multi-user remote application control through RMI technology.

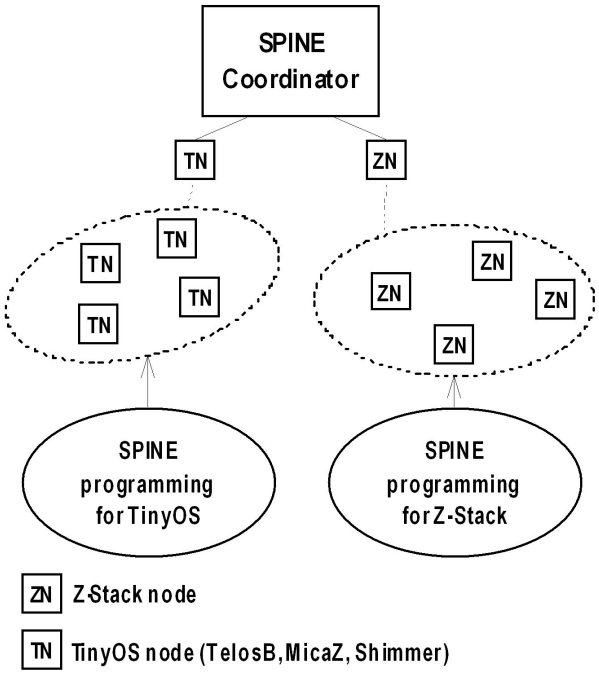


Fig. 5.4. A SPINE heterogeneous network.

When it is needed using different types of sensor within a WBSN application, it is important to know how much the framework performs differently on them. A performance comparison between SPINE 1.2 on TelosB nodes and SPINE 1.3 on Z-Stack nodes, concerning the processing of specific features computed on different sizes (50, 100) of data samples, is reported in Table 5.1.

As can be noted, a Z-Stack node provides better performance than TelosB (obviously, because of its different and higher performance hardware), so it is more desirable to use Z-Stack nodes to accomplish complex and time-consuming processing task.

The SPINE2 framework [127] is an evolution of SPINE based on the C-language for reaching a very high platform independency for C-like programmable sensor platforms (e.g. TinyOS, Ember [116], Z-Stack) and so raising the level of the provided programming abstractions from platform-specific to platform-independent.

In the next section several approaches to develop platform-independent WSN applications are described.

Table 5.1. Performance comparison between TelosB (SPINE 1.2) and Z-Stack (SPINE 1.3) sensor nodes.

Feature	Time (ms)			
	100 samples		*50 samples*	
	TelosB	*Z-Stack Node*	*TelosB*	*Z-Stack Node*
Max	0.88	0.27	0.49	0.15
Mean	1.63	0.53	1.07	0.32
Vector Magnitude	3.13	1.29	2.14	0.76
Pitch Roll	18.40	1.13	17.9	0.85
Standard Deviation	16.7	0.96	10.07	0.55

5.4 Platform-Independent Development of WSN Applications

To develop platform-independent WSN applications several approaches, defined for platform-independent software development in conventional distributed platforms, can be effectively adopted, as described in the followings.

Model-driven Development (MDD). MDD is an approach which provides a set of guidelines for structuring specifications expressed as models and, then, translating such models into platform-dependent code [128]. In particular, MDD defines system functionality using a platform-independent model (PIM) through an appropriate domain-specific language (DSL); then, given a platform definition model (PDM) corresponding to CORBA, .NET, the Web, etc., the PIM is transformed into one or more platform-specific models (PSMs) that computers can run. The PSM may use different DSLs, or a general purpose language like Java, C#, PHP, Python, etc. Moreover, automated tools generally perform this transformation. In [129], an MDD framework to manage the complexity of application development for WSNs is proposed. This framework consists of a UML profile for WSN applications and a UML virtual machine, named Matilda. The proposed UML profile abstracts the low-level details of WSNs and provides higher abstractions for application developers to graphically design and maintain their applications. Matilda is a runtime engine used to design, validate, deploy and execute WSN applications consistently at the modeling layer. In [130], it is argued that current software development of wireless sensor networks not only imposes much work on low-level programmers but also prevents domain experts from directly contributing parts of the software. The proposed solution is based on the exploitation of domain specific languages which are inexpensive to define, have a syntax that domain experts understand, and creating simulations for them is easy. An application modeled through a domain-specific language could be translated into (low-level) platform-dependent code or into bytecode for a virtual machine (see VM approach below). Finally in [131], a framework based on Simulink, State-

flow and Embedded Coder which follows the MDD approach is presented. By using such framework, an engineer can create sensor network components (both at the application and at the protocol level) that can be used as building blocks to model, simulate and automatically generate code for different underlying platforms and operating systems (TinyOS and MantisOS).

Virtual Machine (VM). A VM runs as a normal application inside an OS. Its purpose is to provide a platform-independent programming environment that abstracts away details of the underlying hardware or operating system, and allows a program to execute in the same way on any platform. Several efforts have been devoted to the definition of VMs for the programming of WSN application (Mat, Deluge, SOS, Agilla, etc). In particular, Mat [26] is a byte code interpreter (VM) running on TinyOS, that provides safe program execution environments, runtime re-programming, and an event-driven stack-based architecture. Applications running in Mat use instructions that are interpreted by virtual processors programmed onto network nodes. The performance penalty of the interpretation of the instructions can be alleviated by adding application-specific instructions to the virtual machine. Mat also supports dynamic reconfigurability of nodes through code diffusion.

Software Layering (SL). Software layering has been largely used for the development of communication protocol suites to hide network heterogeneity; TCP/IP is the most notable example. Therefore to hide heterogeneity of different sensor platforms a basic software layer (or core framework), which provides basic functionality, is defined for a set of sensor platforms based on a similar programming language and adapted to each different sensor platform through platform specific modules. Code development is carried out through such common programming language according to the defined core framework. In the context of WSNs, this approach is still scarcely used; however, due to its specific characteristics, it is adopted for the definition of SPINE2 (see next section).

5.5 SPINE2

The subject of this section is the new on-going developments to release version 2 of SPINE which aims to become independent on the low level details and operating systems of the used sensor platform. In order to fulfill this goal, each of the approaches introduced in Section 5.4 might be used.

According to the MDD approach (see Figure 5.5) platform independent models developed through the SPINE language, defined as a DSL, are translated into platform specific code through platform specific translators. Although this approach is very flexible and effective for platform-independent software development, a major problem is that automatic translation may introduce overhead in terms of generated code size and execution speed.

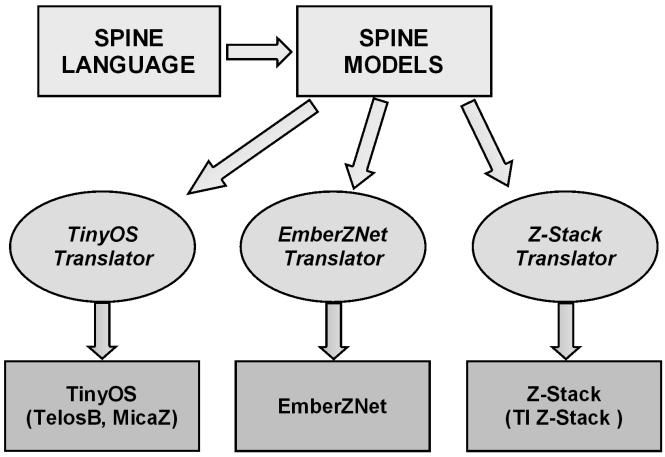


Fig. 5.5. SPINE2 based on the MDD approach.

According to the VM approach (see Figure 5.6) a SPINE VM programmable through a SPINE programming language should be defined and implemented for each sensor platform to be used. Although this approach is effective for providing platform independence, the deployment of a VM on node able to execute the SPINE language can be very expensive in terms of execution speed and used resources (e.g. memory).

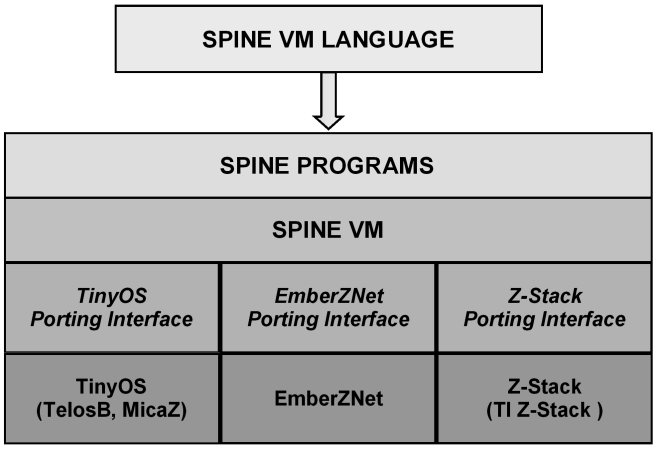


Fig. 5.6. SPINE2 based on the VM approach.

According to the SL approach (see Figure 5.7) a SPINE core framework is defined through a language used by the majority of sensor platform and, then,

adapted to such different platforms through platform specific software modules. With this approach the core framework can be accurately defined and implemented and kept highly efficient. However it fits only sensor platforms programmed through compatible languages.

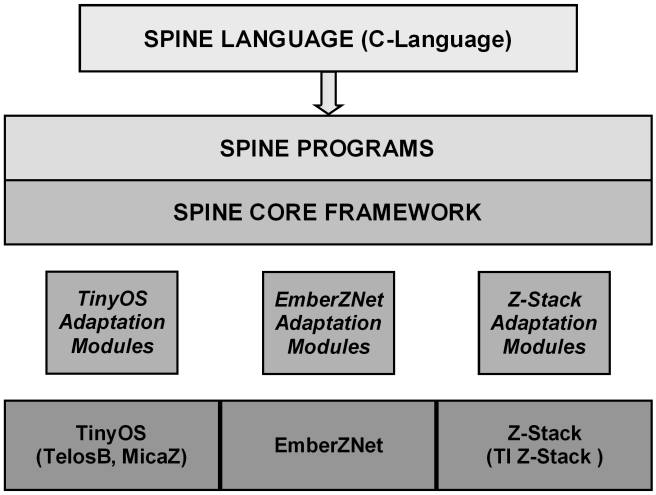


Fig. 5.7. SPINE2 based on the SL approach.

Considered that our requirements were the followings:

- execution on commercial resource-constrained sensor platforms, such as TinyOS [19], Ember [116] or Texas [117], each one having a different operating system;
- minimization of the amount of code that should be replicated for each specific implementation;
- enabling C-developers (eventually C++) to extend the SPINE framework without having to learn low-level details of specific sensor platforms or without having to learn new programming languages, such as nesC [19];
- enabling compiling and simulating the code by using normal ANSI C tools;

we selected the SL approach founded on the C language, which is the language used by the sensor platforms we selected and, as a matter of fact, by the majority of resource-constrained environments. In the next three subsections the programming model, the architecture and the some implementation details of SPINE2 are described.

5.5.1 SPINE2 Programming Model

In SPINE2 a task-oriented programming model was implemented on the nodes in order to best fit the requirements of collaborative distributed applications in a resource constrained environment: an agent is executed on each sensor node that via proper message exchange can discover, create, activate, schedule and control tasks. Distributed and collaborative applications can then be programmed as a dynamically schedulable and reconfigurable set of tasks. Different tasks can be assigned to each node of the network and tasks can be controlled at execution time via proper message exchange; in this way the network can overall adapt to changes in context, in overall goals, in the state of each single node, and it can better balance load and task types between each element of the network. Dynamic distribution of tasks allows among the others preprocessing of sensed data directly on the node, a significant reduction of data transmission and battery consumption, and an overall increase of the network lifetime.

Application developers do not need to program in tiny environments. A SPINE agent is installed on each node that allows interacting with the base station for the task assignment and control. An application is simply a recipe listing the set of tasks to be assigned to each node; the recipe can be executed via the Java API of the base station and, via the same API, the recipe can also be changed at execution time in order to implement different states and intentions of the system.

So far, the following types of tasks have been implemented:

- *SensingTask,* which allows defining a sensing operation on a given sensor. The sensing operation can be one-shot or periodic.
- *TimingTask,* which allows defining timers for timing other tasks. Timers can be one shot or periodic.
- *FunctionalTask,* which refers to the functional tasks defined through programming:
  - *ProcessingTask,* which allows to elaborate data. A specific type of processing is the feature extraction (or simply feature), a data processing algorithm which is carried out on a set of values that can be taken from a data buffer of the BufferPool.
  - *AggregationTask,* which allows aggregating data calculated by different functions.
  - *TransmissionTask,* which allows transmitting data produced by sensing, processing and/or aggregation tasks.

An example of task-oriented programming is shown in Figure 5.8 by means of a data-flow-based model. In this example, the sensed data generated by the *Sensing* task (by acquiring data from a 3-axis accelerometer) are fed to

the *Split* task that, in turn, splits the data for the computation of three features. Each feature is implemented as a Task and fed with different data: the *Mean* task uses data from all three axes (XYZ), the *Min* and *Max* tasks uses data from the X axis. Each triple of computed features (<Mean(AccXYZ), Min(AccX), Max(AccX)>) are aggregated by the *Aggregation* task (Aggr) and sent to the destination node by the data transmission task (*Sender*). The reader can easily guess the variety of complex tasks which can be created by using such a task composition formalism.

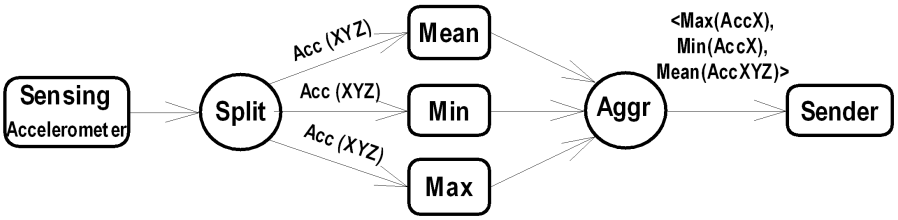


Fig. 5.8. Data-flow-based model.

The SPINE task-oriented programming model is not dataflow driven but event-driven. Thus, the data-flow model of Figure 5.8, which can be seen as a model at a higher-level of abstraction, can be defined according to the event-driven model of SPINE2 as in Figure 5.9. In particular, data, which are sensed by the Sensing task driven by a Timer1 set to a given sampling rate, are stored into decoupling buffers for the three axes channels (AccX, AccY, AccZ). Buffers are managed by the *BufferPoolManager* component which is based on the event-based publish subscribe paradigm. It accepts subscriptions coming from the *FunctionTasks* and as soon as events related to such subscriptions occur, it notifies, through the *AcqNotify*, the FunctionTask subscriber which, in turn, can fetch the data which it is interested in. In the proposed example, the ProcessingTasks are notified by the BufferPoolManager when S sensed data samples have been acquired; where S is the shift parameter of the ProcessingTasks which compute their function on a sample window (W) equals to $n * S$ samples. The processed data are passed to the AggregrationTask (Aggr) which, after aggregation, passes them to the TransmissionTask (Sender). In Figure 5.10 a timer-driven model of the same example is shown. It simplifies the architecture supporting SPINE2 models by avoiding the introduction of the BufferPoolManager active component. In particular, the FunctionTasks are not driven by events sent by the BufferManagerPool but by timers appositely set upon the timer driving the SensingTask. Timer2, Timer3, and Timer4 are therefore set to $S * Timer1$. Moreover, also the Aggr& SendTask (a task which jointly aggregates and transmits) is timer driven. $Timer5 = Timer2 + 0.1 * Timer2$.

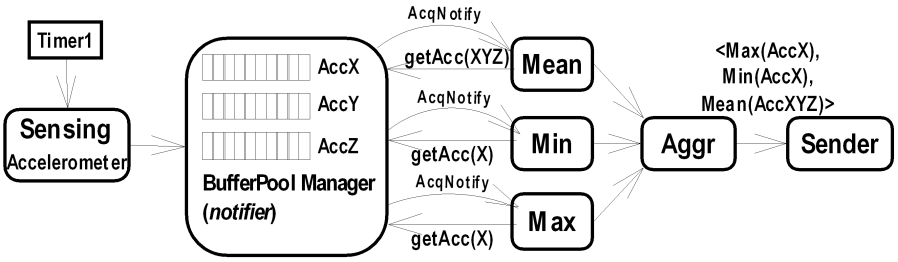


Fig. 5.9. Event-driven SPINE2-based model.

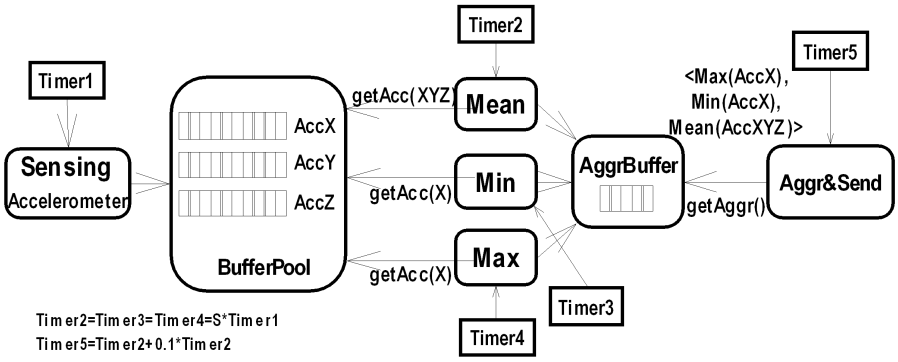


Fig. 5.10. Timer-driven SPINE2-based model.

5.5.2 A Timer Driven Architecture

The timer-driven architecture of SPINE2 (Figure 5.11) consists of a SPINE core framework which is to be adapted to platform specific components (sensor drivers, application lifecycle, timers, communication). The SPINE core framework currently implements the task execution logics according to the timer driven programming model.

The platform-independent components of the SPINE2 framework are:

- The *SPINEApplication* is the core component of a SPINE2 application. It reacts to external events, like messages, and to internal events. It provides three functions:
  - *init* for application initialization;
  - *handleIncomingMessage* for handling an incoming network message;
  - *handleFiredTask* for handling fired tasks.
- The *TaskDescriptionPool* component is a dynamic list of the tasks created in the node.

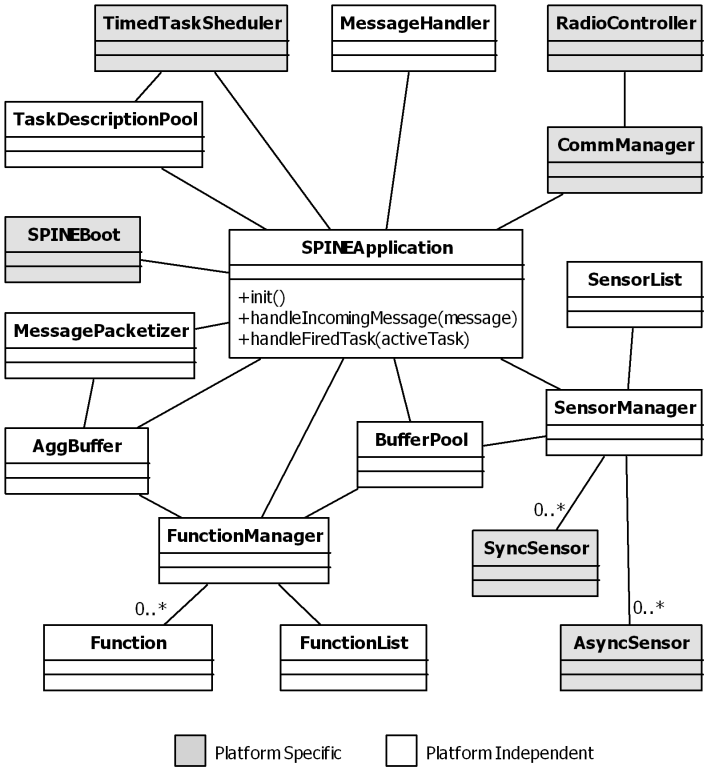


Fig. 5.11. The SPINE2 component diagram.

- The *BufferPool* component consists of a set of data buffer needed to store data produced by sensing tasks and aggregate data computed by functions.
- The *FunctionManager* component, which acts as a Dispatcher managing the available list of functions (FunctionList), is called by the SPINEApplication to execute ProcessingTasks, AggregationTasks and TransmissionTasks.
- The *AggrBuffer* component, which allows to temporary store computed features for aggregating them.
- The *SensorManager* component is connected to platform specific components, which are specific drivers for the Sensors components. The SensorManager manages a list of sensors (SensorList) which can be of synchronous and asynchronous type. While synchronous sensors are read through a synchronous primitive, asynchronous sensors are based on (i) an asynchronous primitive for requesting a sensor read and (ii) the related notification of the read value, which is done by the sensor driver, when data is available, so that the sensor manager can fetch it.

- The *MessageHandler* component contains the handling code of the SPINE2 protocol packets.
- The *MessagePacketizer* component allows building packets according to the SPINE2 protocol.

The aforementioned platform-independent components are to be appositely adapted to the following platform-dependent components which drive their execution:

- The *SPINEBoot* component is the application entry-point which provides the application lifecycle. It contains platform specific initializations and wirings, and drives the SPINEApplication by calling its *init()* method.
- The *TimedTaskScheduler* component manages timed tasks by using platform-specific timers.
- The *AsyncSensor/SyncSensor* components are the sensor drivers, which can be synchronous or asynchronous, through which real sensors can be accessed.
- The *CommManager* component contains the platform-specific radio communication logic. CommManager can receive messages from the RadioController and pass them to the SPINEApplication through the *handleIncomingMessage* method which uses the MessageHandler component. In particular, the RadioController allows to handle the sensor radio for receiving and transmitting packets and for putting the radio in stand-by for energy saving.

The SPINE 1.2 communication protocol which enables communication between the base station and the sensor node is function-oriented. As the programming model of SPINE 2.0 is task oriented, the SPINE2 communication protocol was redesigned. Nevertheless, to maintain backward compatibility, a SPINE1.2/SPINE2 software communication bridge was also implemented. The SPINE2 communication protocol is task-oriented and, in particular, provides the following packet types:

- *createTask,* which allows to create a task with the associated parameters;
- *startTask,* which starts a created task or restart a paused task;
- *pauseTask,* which pauses a started task;
- *updateTask,* which reconfigures a paused task;
- *deleteTask,* which stops and/or cancel a task;
- *getTasksDescription,* which returns the list of the created tasks of the node and their status;
- *startNode,* which starts all the tasks created on the node;

- *getNodeConfiguration,* which returns the configuration of the node in terms of available sensors, functions and tasks;
- *data,* which contains data sent from the node to the coordinator.

5.5.3 Implementation

SPINE2.0 architecture as described in Figure 5.11 easily allows supporting different platforms since the platform specific components are well defined. The parts to be implemented include the application lifecycle, the communication part, the timer related stuff and the low level access, that are all the OS specific parts as previously described. SPINE2.0 currently supports two software sensor platforms, TinyOS 2.1 and Texas Instruments Z-Stack 1.2.

Although these platforms are all based on a C-like programming language, they differ not only in terms of operating systems but also in terms of programming abstractions and communication protocols.

While TinyOS was designed as a general purpose open source operating system for wireless sensor networks, Texas Instrument Z-Stack is a software platform certified to be ZigBee compliant. On the other hand, while being more flexible in terms of applications and modules to be inserted, TinyOS brings another complexity due to the programming paradigm and language. Therefore such two platforms are very good candidates for evaluating the flexibility of the SPINE2.0 architecture.

Z-Stack environment defines not only an operating system but also standard related logics that must be preserved to maintain the standard compliance. ZigBee standard [11] defines not only the low level communication layers (IEEE 802.15.4 as MAC protocol, ZigBee network, management and security layers) and standard application profiles (such as the Home Automation one) but also basic rules to be followed when building a proprietary profile.

As an application level framework, SPINE2.0 on TinyOS has been implemented as an application module whereas on the Z-Stack platform has been designed to be a proprietary ZigBee profile with respect to all the ZigBee related rules.

In particular, the parts to be added into the different platforms are the followings:

- *SPINEBoot* takes care of the system initialization:
  - in the Z-Stack it defines all the ZigBee standard descriptors as well as sets up the ZigBee communication part;
  - in TinyOS this is simply the application entry point from where the TinyOS application is compiled and started and does not include any logic rather than the SPINE related one.

- *CommManager/RadioController* takes care of all the communication related operations:
  - in the Z-Stack this part takes care of the communication through primitives defined by the standard for the application protocol data units transport between peer application entities (ZigBee APS Data Service).
  - in TinyOS it uses send/receive low level APIs for communicating with other devices in the network.
- *TimedTaskScheduler* schedules the timed tasks created in the sensor node:
  - in the Z-Stack implementation this part takes care of all the timed event through an extensive usage of the utilities provided by the Z-Stack OS (OSAL, Operating System Abstraction Layer) such as task allocation, timer settings and so on. It is important to notice that SPINE2.0 task oriented architecture run without any modification even into a task oriented OS as the OSAL is;
  - the timer interface already defined in the OS is used in the TinyOS version to manage timer related events
- *Sensor drivers* which are platform specific since they have to access to low level functionalities.

Moreover each platform will need specific configuration files setting all the tunable parameters.

The Timer-driven SPINE2.0 release has been successfully implemented and tested on both mentioned platforms. In the following we elucidate the structure and programming of the available tasks.

A timed task is defined as a C-struct as follows:

```
typedef struct timedTaskDescriptor{
   unsigned char taskID;
   unsigned char taskType;
   unsigned char status;
   unsigned long timer;
   unsigned char timerScale;
   unsigned char isPeriodic;
   unsigned char parameters[TASK_PARAMETER_LENGTH];
} timedTaskDescriptor;
```

where, *taskID* is the unique task identifier, *taskType* is the type of task, *status* holds information about the task status (created, active, paused), *timer* contains the task firing time, *timerScale* contains the measurement unit of the timer, *isPeriodic* signals if the timed task is periodic or one-shot, and *parameters* contains parameters specific to the *taskType*.

The currently available taskTypes are *sensing*, *feature extraction*, and *aggregation and sending*:

```
enum taskTypes{
   TASKTYPE_FEATURE_EXTRACTION = 0x01,
   TASKTYPE_SENSING = 0x02,
   TASKTYPE_AGGR_AND_SEND = 0x03
};
```

In particular the parameters of task types are defined as follows:

```
enum sensing_TaskType{
   SENS_SENSOR_ID = 0,         //id of the sensor
   SENS_CHANNEL_BITMASK = 1,  //bitmask for Ch selection
   SENS_BUFFER_ID_1 = 2,       //buffer associated to Ch1
   SENS_BUFFER_ID_2 = 3,       //buffer associated to Ch2
   SENS_BUFFER_ID_3 = 4,       //buffer associated to Ch3
   SENS_BUFFER_ID_4 = 5        //buffer associated to Ch4
};


enum feature_extraction_TaskType{
   FEX_FEATURE = 0,          //id of the feature
   FEX_CHANNEL_BITMASK = 1, //bitmask for Ch selection
   FEX_WINDOW = 2,           //data window
   FEX_BUFFER_ID_1 = 3,      //buffer associated to Ch1
   FEX_BUFFER_ID_2 = 4,      //buffer associated to Ch2
   FEX_BUFFER_ID_3 = 5,      //buffer associated to Ch3
   FEX_BUFFER_ID_4 = 6,      //buffer associated to Ch4
   FEX_SENSOR_ID = 7,        //id of the sensed sensor
   FEX_AGGR_ID = 8           //id of the aggr&send task
};


enum aggregation_and_sending_TaskType{
   AGG_ID = 0,                    //id of the aggr&send task
   AGG_FEATURES_TO_WAIT_FOR = 1, // aggr feature number
   AGG_TIMER = 2,                 //reference aggr timer
   AGG_DEF_COUNTER = 6            //number of aggr trials
}
```

The example of Figure 5.10 is implemented and successfully tested on TelosB motes [132] and TI ZStack sensor nodes both equipped with specific 3-axial accelerometer sensor boards. In particular the defined timed tasks (sensing from accelerometer, calculation of mean, and aggregation and sending) are the following:

- `TASKTYPE_SENSING`
  ```
  (sensTask)->taskID = 1;
  (sensTask)->taskType = TASKTYPE_SENSING;
  ```

```
(sensTask)->timer = 25;
(sensTask)->timerScale = 1; //ms
(sensTask)->isPeriodic = 1; //true
(sensTask)->parameters[ACQ_SENSOR_ID]=1; //accelerometer
(sensTask)->parameters[ACQ_CHANNEL_BITMASK]=e; //Ch XYZ
(sensTask)->parameters[ACQ_BUFFER_ID_1] = 0;
(sensTask)->parameters[ACQ_BUFFER_ID_2] = 1;
(sensTask)->parameters[ACQ_BUFFER_ID_3] = 2;
```

- `TASKTYPE_FEATURE_EXTRACTION_1`

```
(featExtTask)->taskID = 2;
(featExtTask)->taskType = TASKTYPE_FEATURE_EXTRACTION;
(featExtTask)->timer = 1000; //40 samples
(featExtTask)->timerScale = 1;
(featExtTask)->isPeriodic = 1;
(featExtTask)->parameters[FEX_FEATURE] = 5; //MEAN
(featExtTask)->parameters[FEX_CHANNEL_BITMASK] = e;
(featExtTask)->parameters[FEX_WINDOW] = 80;
(featExtTask)->parameters[FEX_BUFFER_ID_1] = 0;
(featExtTask)->parameters[FEX_BUFFER_ID_2] = 1;
(featExtTask)->parameters[FEX_BUFFER_ID_3] = 2;
(featExtTask)->parameters[FEX_SENSOR_ID] = 1;
(featExtTask)->parameters[FEX_AGGR_ID]=1;
```

- `TASKTYPE_AGGR_AND_SEND`

```
(aggrSendTask)->taskID = 5;
(aggrSendTask)->taskType = TASKTYPE_AGGR_AND_SEND;
(aggrSendTask)->timer = 1100;
(aggrSendTask)->timerScale = TIMER_SCALE_MSEC;
(aggrSendTask)->isPeriodic = FALSE;
(aggrSendTask)->parameters[AGG_ID] = 1;
(aggrSendTask)->parameters[AGG_FEATURES_TO_WAIT_FOR]=3;
(aggrSendTask)->parameters[AGG_TIMER] = 1000;
(aggrSendTask)->parameters[AGG_DEF_COUNTER] = 0;
```

Tasks can be created and started either by explicit node programming or by the base station through the SPINE2 protocol.

To experiment with SPINE2, the sensor-node side application presented in [80], which allows the activity monitoring of individuals (standing, lying, walking and sitting), is now based on sensor nodes supported by SPINE2.0 (See Section 5.6).

Experimentation with SPINE2 was also carried out to demonstrate that the platform independency of SPINE2 does not introduce performance penalties with respect to SPINE1.2. To this purpose, an evaluation of the data processing performances of the TinyOS versions of SPINE1.2 and SPINE2 on

the TMote SKY TelosB sensor platform [132] has been carried out. The performance evaluation results are reported in Table 5.2. In particular, selected features (max, mean, standard deviation, vector magnitude, pitch & roll, and entropy) were computed on different sample sizes (50, 100, 200) acquired from a 3-axial accelerometer sensor board. As can be noted, SPINE2 feature processing performances are higher than those computed with SPINE1.2. This performance improvement is mainly due to the different methods of calling processing functions in SPINE1.2 and SPINE2: in SPINE1.2 a processing function is executed by means of a call to a nesC command whereas in SPINE2 a simple call to a C function of an included file is done.

Table 5.2. Comparison of feature processing times (ms) through SPINE1.2 and SPINE2 in TinyOS on TelosB sensor nodes.

	50 samples		100 samples		200 samples	
	SPINE1.2	SPINE2	SPINE1.2	SPINE2	SPINE1.2	SPINE2
MAX	0,488	0,488	0,883	0,886	1,696	1,679
MEAN	1,069	1,038	1,627	1,556	2,714	2,625
STANDARD DEVIATION	10,070	7,450	16,703	13,823	28,617	26,054
VECTOR MAGNITUDE	2,136	1,741	3,126	2,501	4,564	3,997
PITCH & ROLL	17,894	16,967	18,401	17,428	19,461	18,552
ENTROPY	239,291	235,848	488,185	481,167	1016,392	1001,735

5.6 A Case Study: Activity Monitoring on Heterogeneous WBSNs

To test the effectiveness of the SPINE2 framework and its capability on managing an heterogeneous network, a physical Activity Monitoring System (AMS [80]) has been reverse engineered and made heterogeneous.

AMS is able to recognize postures (e.g. lying, sitting or standing still) and a few movements (e.g. walking and jumping) of a person; furthermore it can detect if the monitored person has fallen and unable to stand up.

The system consists of a user application and two node applications. The former is implemented in Java and runs on top of the SPINE Manager (included in the coordinator part of the SPINE framework) which can access distinctive communication modules for interacting with different types of sensor. The latter consists of two different applications designed following the

SPINE2 task oriented approach and running on top of the SPINE2 node runtime system (the node part of the SPINE2 framework). The entire software design related to the activity monitoring system is depicted in Figure 5.12.

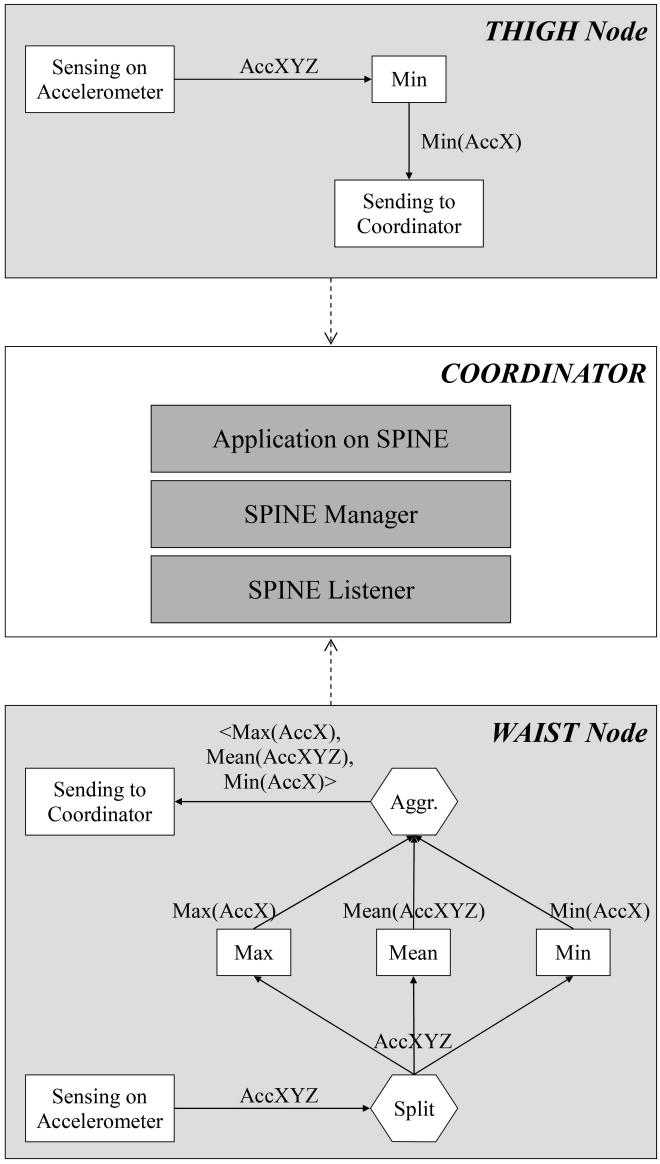


Fig. 5.12. The Activity Monitoring application.

In Figure 5.13 the complete Activity Monitoring system is shown. The network architecture is composed of a TelosB node placed on a thigh of a person and a Z-Stack node placed on the waist of the same person. The different types of nodes run the same SPINE2 core runtime system plus the particular adaptation code related to the particular node platforms. We already discussed on the possibility of having different types of sensor in the same SPINE net and this real application demonstrates the feasibility in managing a heterogeneous WBSN.

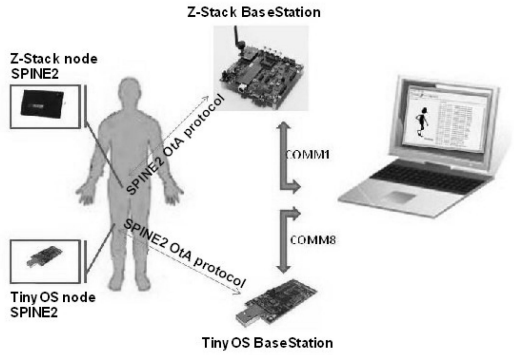


Fig. 5.13. The SPINE 2 Activity Monitoring system.

The coordinator (running on a notebook) has been interfaced with the wireless sensors network through other two nodes connected via USB cable and providing the necessary radio communication capability.

The user application on the coordinator is responsible for gathering preelaborated data taken from the accelerometer sensors of the nodes and relies on a classifier that recognizes postures and movements defined in a training phase.

The application integrates two different classifiers. One based on the K-Nearest Neighbor algorithm [133] and the other based on J48 Decision Tree [134]. They were setup through a training phase and tested considering the following settings for the data acquisition from sensors: the sample rate was set to 50ms, the window to 20, whereas the shift to 10. This means that the features in Figure 5.13 (Min, Max and Mean) are evaluated on 20 sampled data and computed every new 10 samples acquired by the sensors. See Table 5.3 for the obtained classification accuracy results.

Table 5.3. Classification accuracy for classifiers based on K-Nearest Neighbor and J48 Decision Tree.

	Walking	**Sitting**	**Standing**	**Lying**
K-NN	94,0%	96,0%	92,0%	98,0%
J48 D Tree	92,0%	98,0%	94,0%	94,0%

6

Conclusions, Publications and Future Directions

6.1 Conclusions

Throughout this thesis, several contributions have been made to the Wireless Sensor Networks research community.

In particular, this thesis has introduced two domain-specific frameworks and a general-purpose mobile agent system that provide an effective support to the development of WSN.

The first contribution of the thesis consists of the Building Management Framework (BMF), a domain-specific framework for effective management of Wireless Sensor and Actuator Networks (WSAN) which enables proactive monitoring of spaces and control of devices / equipments. BMF is specifically designed to provide flexible and efficient management of networked sensors and actuators and abstractions for logical and physical node grouping to expressively capture the morphology of buildings. The overall goal is to provide a versatile platform to allow intelligent sensing and actuation techniques, integration of heterogeneous WSANs, flexible system programming at low- and high-level, fast deployment of different applications through message-based programming. BMF is the only framework specifically conceived to address Building Management that fulfils all the requirements identified by building applications. The effectiveness of BMF has been demonstrated through SmartEnLab, a BMF-based application for energy monitoring in computer laboratory / office environments. SmartEnLab has shown not only flexibility in terms of monitoring (request schedule and data reception) and simplicity in the extraction of information, but also an interesting demonstrator of energy profiling application in building environments. SmartEnLab has been useful for: (i) the profiling of the workstation usage through the data gathered from the WSAN. In particular, energy and occupancy profiles over a 24 hour period were shown for some workstations; (ii) a performance evaluation of the BMF-based building wireless sensor network that shows some advantages in the use of the BMF. In particular, the use of aggregation and selection mech-

anisms yields an average increase of more than 10% in transmission reliability with respect to raw data transmission schemes, while reducing significantly the energy consumption in transmission. In node aggregation allows to appreciably reduce the radio channel usage, too; (iii) a WSAN lifetime estimation of a network running the BMF that shows the increased lifetime allowed by BMF with respect to WSAN with no support framework. In particular, BMF doubles the duration of the expected life of a WSAN still keeping low the packet loss rate.

As second contribution, the thesis has proposed mobile agents as an effective paradigm to program WSN applications and, in particular, presented MAPS, a Java-based framework for the development of agent-based applications for Sun SPOT sensor platforms. By using MAPS, a WSN application can be structured as a set of stationary and mobile agents distributed on sensor nodes supported by a component-based agent execution engine that provides basic services such as message transmission, agent creation, agent cloning, agent migration, timer handling and easy access to the sensor node resources. MAPS programming has been exemplified through a simple yet effective example that shows how to program the dynamic behavior of agents in terms of state machines on the basis of the MAPS library. An evaluation of MAPS has been presented according to micro-kernel benchmarks (agent communication, migration and creation) usually employed for mobile agent systems. Evaluation shows some performance penalties mainly due to very time-consuming operations (Isolate hibernation/serialization and radiostream based communications) provided by the Sun SPOT libraries and SquawkVM on which MAPS relies. Moreover, a case study concerning a decentralized and embedded management architecture for intelligent buildings that is based on WSANs has been described. It is emblematic of the effectiveness and suitability of MAPS to deal with the programming of complex applications. Finally, MAPS has been compared in depth with the AFME (Agent Factory Micro Edition) agent platform.

As third contribution, SPINE2 has been presented. SPINE2 is a domain-specific framework for the platform-independent development of collaborative WBSN applications. SPINE2 consists of two parts: (i) the core which is written in C and is independent from any C-like sensor platform on which it can be ported on; (ii) a set of platform-dependent components (sensors, communications, timers, application lifecycle) through which the core can be easily adapted. The task-oriented programming model of SPINE2 enables a flexible development of WBSN applications in terms of a star-based network of collaborative and dynamically reconfigurable tasks which concur to carry out an overall distributed task. SPINE2 (specifically the timerdriven architecture) is currently implemented for TinyOS sensor platforms (in particular for TelosB motes) and ZStack Zigbee sensor nodes, and was successfully applied to the realization of a multi-platform WBSN application for activity monitoring of

individuals. Moreover, SPINE2 increases performances of feature calculation with respect to SPINE1.2.

6.2 Publications Related with this Thesis

The research work related to this thesis has resulted in 18 publications. Among them, there are 4 journal articles, 2 book chapters, 11 conference papers, and 1 conference poster.

In the following, brief description of each publication is provided.

6.2.1 Journal Articles

- **A Java-based agent platform for programming Wireless Sensor Networks** [87]:

  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.

- **SPINE-based application development on Heterogeneous Wireless Body Sensor Networks** [115]:

  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.

- **SPINE: A domain-specific framework for rapid prototyping of WBSN applications** [64]:

  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 [80]. 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.

- **An analysis of Java-based mobile agent platforms for Wireless Sensor Networks** [91]:

  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.

6.2.2 Book Chapters

- **Signal processing in-node frameworks for Wireless Body Sensor Networks: from low-level to high-level approaches** [135]:

  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.

- **A WSN-based Building Management Framework to Support Energy-Saving Applications in Buildings** [51]:

  A. Guerrieri, G. Fortino, A. Ruzzelli and G. O'Hare. *Advancements in Distributed Computing and Internet Technologies: Trends and Issues*, chapter 12 - A WSN-based Building Management Framework to Support Energy-Saving Applications in Buildings, pages 161-174. Hershey, PA, USA: IGI Global, 2011.

  This book chapter defines the specific requirements for applications of energy management in the building context and proposes a novel framework for building management (BMF) to support heterogeneous platforms. To allow flexible node activity grouping, BMF defines roles and operations derived from the mathematical set theory, while it optimizes transmissions

through a mechanism of adaptive packet size. BMF has been implemented and tested in TinyOS. Results show an increase in reliability with respect to existing transmission schemes that can be traded off to reduce energy consumption.

6.2.3 Conference Papers

- **MAPS: A Mobile Agent Platform for WSNs based on Java Sun Spots** [90]:

  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. Budapest, Hungary, 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.

- **SPINE2: developing BSN applications on heterogeneous sensor nodes** [118]:

  G. Fortino, A. Guerrieri, F. Bellifemine and R. Giannantonio.SPINE2: developing BSN applications on heterogeneous sensor nodes. In *Proceedings of IEEE Symposium on Industrial Embedded Systems*, SIES'09. Losanna, Svizzera, July 2009.

  This paper presents SPINE2, an evolution of SPINE, which aims at reaching a very high platform independency and raising the level of the used programming abstractions by providing a task-oriented programming model. Furthermore, SPINE2 is exemplified through a case study related to human activity monitoring.

- **Programming signal processing applications on heterogeneous wireless sensor platforms** [126]:

  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, Rende, Cosenza, Italy, 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.

- **Platform-independent development of collaborative Wireless Body Sensor Network applications: SPINE2** [127]:

  G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio. Platform-independent development of collaborative Wireless Body Sensor Network applications: SPINE2. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, SMC 2009, pages 3144-3150. IEEE, San Antonio, TX, USA, October 2009.

  This paper discusses issues related to platform-independent development of collaborative WBSN applications and, specifically, describes the requirements, architecture and first implementation experiences of SPINE2 which aims at reaching a very high platform independency and raising the level of the used programming abstractions by providing a task-oriented programming model. The paper also discusses how such a task-oriented model enables dynamic task assignment and holistic collaborative task execution also for resource-constrained environments such as tiny sensor nodes.

- **An agent-based signal processing in-node environment for real-time human activity monitoring based on Wireless Body Sensor Networks** [136]:

  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, Toronto, Canada, 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.

- **ANNOT: Automated Electricity Data Annotation Using Wireless Sensor Networks** [137]:

  A. Schoofs, A. Guerrieri, D. T. Delaney, G. M. P. O'Hare, and A. Ruzzelli. ANNOT: Automated Electricity Data Annotation Using Wireless Sensor Networks. In *Proceedings of the 7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, SECON 2010, pages 244-252. Boston, MA, USA, June 2010.

  This paper proposes ANNOT, a system to automate energy data annotation leveraging cheap wireless sensor nodes. Characteristic sensory stimuli captured by sensor nodes using the Building Management Framework (BMF) are translated into appliance operating state and correlated to the energy data, autonomously generating the annotation of energy data with appliance activity. The system is able to generate appliance signatures, training data and validate the monitoring output.

- **Decentralized and Embedded Management of Smart Buildings** [108]:

  G. Fortino, and A. Guerrieri. Decentralized and Embedded Management of Smart Buildings.

In *Proceedings of the Workshop on Applications of Software Agents*, WASA 2011, pages 3-7. Novi sad, Serbia, July 2011.

This paper proposes a decentralized and embedded architecture based on agents and wireless sensor and actuator networks for enabling efficient and effective management of buildings. The main purpose of the agent-based architecture is to efficiently support distributed and coordinated sensing and actuation operations. The high modularity of the proposed architecture allows for easy adaptation of higher-level application-specific agents that can therefore exploit the architecture to implement intelligent building management policies.

- **Agent-based development of Wireless Sensor Network applications** [92]:

  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, Rende, Cosenza, Italia, 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.

- **Continuous, Real-time Monitoring of Assisted Livings through Wireless Body Sensor Networks** [138]:

  D.L. Carní, 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, Prague, Czech Republic, 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.

- **Monitoring Building Indoors through Clustered Embedded Agents** [89]:

  G. Fortino, and A. Guerrieri. Monitoring Building Indoors through Clustered Embedded Agents. In *Proceedings of Workshop on Agent Based Computing: from Model to Implementation jointly held with FedCSIS Conference*, ABC 2011. Szczecin, Poland, September 2011.

  This paper proposes the real implementation of an architecture based on agents and wireless sensor and actuator networks (WSANs) for enabling efficient and effective management of buildings. The building management architecture is implemented in MAPS (Mobile Agent Platform for Sun SPOTs), an agent-based framework for programming WSN applications

based on the Sun SPOT sensor platform. The proposed architecture is demonstrated in a simple yet effective operating scenario related to monitoring workstation usage in computer laboratories.

- **Pervasive Monitoring of Building Indoors through Heterogeneous Wireless Sensor Networks** [52]:
  G. Fortino, and A. Guerrieri. Pervasive Monitoring of Building Indoors through Heterogeneous Wireless Sensor Networks. In *Proceedings of Networking and Electronic Commerce Research Conference*, NAEC 2011. Riva Del Garda, Italy, October 2011.
  This paper proposes a multi-platform domain specific framework based on Wireless Sensor and Actuator Networks (WSAN) for enabling efficient and effective management of buildings. The proposed Building Management Framework provides powerful abstractions that capture the morphology of buildings to allow for the development of pervasive building monitoring applications. Moreover, the functionalities of the framework are shown in a simple yet effective operating scenario related to monitoring workstation usage in university offices and laboratories.

6.2.4 Conference Poster

- **An OSGi Dynamic Framework To Support Sensor Network Applications In Buildings** [66]:
  A. Guerrieri, G. Fortino, A. Ruzzelli, and G. O'Hare. An OSGi Dynamic Framework To Support Sensor Network Applications In Buildings. In *Proceedings of the 6th ACM Workshop on Hot Topics in Embedded Networked Sensors*, HotEMNETS 2010. Killarney, Ireland, June 2010.
  This poster resents the design of the Building Management Framework, a domain specific framework based on Wireless Sensor and Actuator Networks, and provides an overview on its features.

6.3 Future Directions

In the development of this thesis several issues emerged which deserve further examination in the future.

With regards to the first contribution of the thesis, future work will investigate the coordination of multiple BS aiming to manage a group of buildings. In fact, a limitation of the current implementation of the BMF, is the scalability to hundreds of nodes, which causes overloading with the loss of too many packets and significant delays of a tens of seconds, which may prevent the implementation of real-time actuation of critical appliances. In this case, a multi-BS system can significantly reduce the packet transmission delay and

would allow nodes to work on a lower duty cycle to reduce the energy spent in the network extending its lifetime. Future work will also design a high-level OSGi based architecture for Smart Buildings atop the proposed architecture to trade off inhabitants' personal comfort and building energy expenditure. In fact, while the proposed framework is effective to manage WSAN in buildings, it requires data mining techniques to infer information about habits and preferences of inhabitants of the building and merge them with energy saving techniques.

Regarding the second contribution, future research efforts are being devoted to: further optimizing the communication and migration mechanisms of MAPS; porting MAPS onto the Sentilla JCreate pervasive computers which are compliant to Java ME CLDC 1.1; developing real applications through MAPS in the context of human activity monitoring for e-health (re-implementing through agents the SPINE2 framework). Future research will also focus on the definition of high-level (based on application gateways) and low-level (based on the IEEE 802.15.4 layer) solutions for agent communication interoperability between MAPS and AFME that would enable the development of heterogeneous agent-based WSN applications.

With respect to the third contribution, future work will be devoted to complete the implementation of SPINE2 for the Ember sensor platform and design a version for ContikiOS; develop a SPINE2 coordinator based on a task-oriented protocol to program and control SPINE2 sensor nodes; design a flexible event-based architecture for SPINE2 to increase programming effectiveness and avoid an excessive use of timers; and extend SPINE2 for more general collaborative WSN applications (not only centered on star-based networks).

References

1. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 38:393–422, 2002.
2. K. J. Gabriel. Microelectromechanical systems (MEMS) tutorial. In *Proceedings of the 1998 IEEE International Test Conference*, ITC '98, pages 432–441, Washington, DC, USA, 1998. IEEE Computer Society.
3. G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 1–12. ACM, 2004.
4. H. J. Yoo and C. van Hoof. *Bio-Medical CMOS ICs*. Springer, 2011.
5. G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing*, 10:18–25, 3 2006.
6. Atmel Corporation. http://www.atmel.com, 2011.
7. Cypress Semiconductor Corporation. http://www.cypress.com, 2011.
8. Texas Instruments. http://www.ti.com/, 2011.
9. W. K. G. Seah, Z. A. Eu, and H. Tan. Wireless Sensor Networks Powered by Ambient Energy Harvesting (WSN-HEAP) Survey and Challenges. In *Proceedings of the 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE 2009)*, pages 1–5, 2009.
10. I. Howitt and J. A. Gutierrez. IEEE 802.15.4 low rate - wireless personal area network coexistence issues. In *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, pages 1481–1486. IEEE, 2003.
11. ZigBee: wireless control that simply works ZigBee Alliance. http://www.zigbee.org/, 2011.
12. HART The Logical Wireless Solution. http://www.hartcomm.org/protocol/wihart/wireless_technology.html, 2011.
13. ISA100.11a. http://www.isa.org/mstemplate.cfm?micrositeid=1134&committeeid=6891, 2011.
14. 6LoWPAN. http://6lowpan.net/, 2011.

15. IEEE 802.15 WPAN Task Group 3 (TG3). http://www.ieee802.org/15/pub/tg3.html, 2011.
16. M. O. Farooq and T. Kunz. Operating Systems for Wireless Sensor Networks: A Survey. *Sensors*, 11(6):5900–5930, 2011.
17. A.M. Reddy, P. Kumar, G.A. Kumar, and D. Janakiram. Operating Systems for Wireless Sensor Networks: A Survey Technical Report. *International Journal of Sensor Networks (IJSNet)*, 5(4):236–255, 2009.
18. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *TinyOS: An Operating System for Sensor Networks Ambient Intelligence*. Springer Berlin Heidelberg, Berlin/Heidelberg, 2005.
19. TinyOS Web Site. http://www.tinyos.net, 2011.
20. 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, Washington, DC, USA, 2004. IEEE Computer Society.
21. A. K. Dwivedi, M. K. Tiwari, and O. P. Vyas. Operating systems for tiny networked sensors: A survey. *Int. Journal of Recent Trends in Engineering*, 1:152–157, 2009.
22. 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*, 10:563–579, 8 2005.
23. 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*, pages 256–265. IEEE Computer Society, 2005.
24. A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar. Rate-Harmonized Scheduling for Saving Energy. In *Proceedings of the 2008 Real-Time Systems Symposium*, pages 113–122. IEEE Computer Society, 2008.
25. Q. Cao, T. Abdelzaher, J. Stankovic, and T. He. The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 233–244. IEEE Computer Society, 2008.
26. P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 85–95. ACM, 10 2002.
27. R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS Operating Systems Review*, 36:1–5, 4 2002.
28. D. Simon and C. Cifuentes. The Squawk Java Virtual Machine: Java on the Bare Metal. In *Proceedings of the 20th Object-Oriented Programming, Systems, Languages and Applications*, OOPSLA 2005, pages 150–151. ACM, 2005.
29. 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 '06, pages 78–88. ACM, 2006.

30. P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proceedings of the Second International Conference on Mobile Data Management*, MDM '01, pages 3–14, London, UK, 2001. Springer-Verlag.
31. C. Srisathapornphat, C. Jaikaeo, and C. Shen. Sensor Information Networking Architecture. In *Proceedings of the 2000 International Workshop on Parallel Processing*, ICPP '00, pages 23–, Washington, DC, USA, 2000. IEEE Computer Society.
32. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS) - Special Issue: SIGMOD/PODS 2003*, 30:122–173, 3 2005.
33. R. Mueller, G. Alonso, and D. Kossmann. SwissQM: Next Generation Data Processing in Sensor Networks. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, pages 1–9, 2007.
34. A. Boulis, C. Han, and M. B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 187–200. ACM, 2003.
35. C. Fok, G. Roman, and C. Lu. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, pages 653–662, Washington, DC, USA, 2005. IEEE Computer Society.
36. Y. Kwon, S. Sundresh, K. Mechitov, and G. Agha. ActorNet: an actor platform for wireless sensor networks. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1297–1300. ACM, 2006.
37. E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: a publish/subscribe middleware for sensor networks. *Journal Personal and Ubiquitous Computing*, 10:37–44, 12 2005.
38. J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 52:2292–2330, 8 2008.
39. G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63. ACM, 2005.
40. I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 154–165. ACM, 2005.
41. M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 192–204. ACM, 2005.
42. C. R. Baker, K. Armijo, S. Belka, M. Benhabib, V. Bhargava, N. Burkhart, A. D. Minassians, G. Dervisoglu, L. Gutnik, M. B. Haick, C. Ho, M. Koplow, J. Mangold, S. Robinson, M. Rosa, M. Schwartz, C. Sims, H. Stoffregen, A. Waterbury, E. S. Leland, T. Pering, and P. K. Wright. Wireless Sensor Networks for Home Health Care. In *Proceedings of the 21st International Conference on*

Advanced Information Networking and Applications Workshops - Volume 02, AINAW '07, pages 832–837. IEEE Computer Society, 2007.

43. L. Zhang, L. Lao, K. Wu, Q. Liu, and X. Wu. Research in Development on Wireless Health Care of Infants. In Yi Peng, Xiaohong Weng, and Ratko Magjarevic, editors, *Proceedings of the 7th Asian-Pacific Conference on Medical and Biological Engineering*, volume 19 of *IFMBE Proceedings*, pages 580–583. Springer Berlin Heidelberg, 2008.
44. P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in ZebraNet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 227–238. ACM, 2004.
45. D. Snoonian. Control systems: smart buildings. *IEEE Spectrum*, 40:18–23, 8 2003.
46. J. Stankovic. When sensor and actuator cover the world. *ETRI Journal*, 30(5):627–633, 2008.
47. M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*. USENIX Association, 2004.
48. L. Mottola, G. P. Picco, and A. A. Sheikh. FiGaRo: fine-grained software reconfiguration for wireless sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks*, EWSN'08, pages 286–304, Berlin, Heidelberg, 2008. Springer-Verlag.
49. Crossbow Ecowizard System. http://www.xbow.com/pdf/ecowizard_introduction_pressrelease.pdf, 2011.
50. EpiSensor SiCA for Building Management. http://episensor.com/solutions/building-management/, 2011.
51. A. Guerrieri, G. Fortino, A. Ruzzelli, and G. O'Hare. *A WSN-based Building Management Framework to Support Energy-Saving Applications in Buildings*. Hershey, PA, USA: IGI Global, 2011.
52. G. Fortino and A. Guerrieri. Pervasive Monitoring of Building Indoors through Heterogeneous Wireless Sensor Networks. In *Proceedings of Networking and Electronic Commerce Research Conference*, NAEC 2011, Riva Del Garda, Italy, 10 2011.
53. documents Open System Gateway Initiative (OSGi) and software. http://www.osgi.org, 2011.
54. J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.
55. J. Barton, G. Hynes, B. O'Flynn, K. Aherne, A. Normana, and A. Morrissey. 25mm sensor-actuator layer: A miniature, highly adaptable interface layer. *Sensors & Actuators: A. Physical*, 132:362–369, 11 2006.
56. X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and implementation of a high-fidelity AC metering network. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, IPSN '09, pages 253–264, Washington, DC, USA, 2009. IEEE Computer Society.
57. Sun Small programmable object technology (Sun SPOT). http://www.sunspotworld.com/, 2011.
58. K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3:325–349, 5 2005.

59. S. O'Connell, J. Barton, E. O'Connell, B. O'Flynn, E. M. Popovici, S. C. O'Mathuna, A. Schoofs, A. G. Ruzzelli, and G. M. P. O'Hare. Remote Electricity Actuation and Monitoring mote. In *Proceedings of International Conference on Distributed Computing in Sensor Systems and Workshops*, DCOSS, pages 1–6. IEEE, 2011.
60. Arch Rock Energy Optimizer. http://www.archrock.com/products/areo.php, 2011.
61. DELTA DORE Building Management Systems solutions (BMS). http://www.deltadore.com/accueil/gestion-technique-des-batiments/var/lang/en/rub/9141.html, 2011.
62. TREND IQ ASSURED. https://www.trendcontrols.com/en-gb/support/pages/default.aspx, 2011.
63. SENTILLA Energy Manager 3.0. http://www.sentilla.com/products/datacenter/why-sentilla, 2011.
64. 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*, 41:237–265, 03 2011.
65. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, PLDI '03, pages 1–11. ACM, 2003.
66. A. Guerrieri, G. Fortino, A. Ruzzelli, and G. O'Hare. An OSGi Dynamic Framework To Support Sensor Network Applications In Buildings. In *Proceedings of the 6th ACM Workshop on Hot Topics in Embedded Networked Sensors*, HotEMNETS 2010, 6 2010.
67. P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI '04, Berkeley, CA, USA, 2004. USENIX Association.
68. V. L. Erickson, Y. Lin, A. Kamthe, Brah R., A. Surana, A. E. Cerpa, M. D. Sohn, and S. Narayanan. Energy efficient building environment control strategies using real-time occupancy measurements. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 19–24, New York, NY, USA, 2009. ACM.
69. K. Padmanabh, V. A. Malikarjuna, S. Sen, S. P. Katru, A. Kumar, S. Pawankumar, S. K. Vuppala, and S. Paul. iSense: a wireless sensor network based conference room management system. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 37–42, New York, NY, USA, 2009. ACM.
70. D. T. Delaney, G. M. P. O'Hare, and A. G. Ruzzelli. Evaluation of energy-efficiency in lighting systems using sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 61–66, New York, NY, USA, 2009. ACM.
71. J. E. Petersen, V. Shunturov, K. Janda, G. Platt, and K. Weinberger. Dormitory residents reduce electricity consumption when exposed to real-time visual feedback and incentives. *International Journal of Sustainability in Higher Education*, 8:16–33, 2007.

72. H. Doukas, K.D. Patlitzianas, K. Iatropoulos, and J. Psarras. Intelligent building energy management system using rule sets. *Building and Environment*, 42(10):3562–3569, 2007.
73. P. Eichholtz, N. Kok, and J. M. Quigley. Doing Well by Doing Good: Green Office Buildings. *American Economic Review*, 100(6):2494–2511, 2010.
74. U. Gneezy, E. Haruvy, and H. Yafe. The inefficiency of splitting the bill. *The Economic Journal*, 114:265–280, 2004.
75. WiEye Sensor board for wireless surveillance and security applications. http://www.easysen.com/wieye.htm, 2011.
76. S. Karayi. Pc energy report 2009, www.climatesaverscomputing.org/docs/1e_pc_energy_report_2009_us.pdf, 2009.
77. CIBSE. *Heating, Ventilating, Air Conditioning and Refrigeration: CIBSE Guide B*. CIBSE, 2005.
78. B. K. Sovacool and M. A. Brown. Twelve metropolitan carbon footprints: A preliminary comparative global assessment. *The International Journal of the Political, Economic, Planning, Environmental and Social Aspects of Energy*, 38(9):4856–4869, 2010.
79. A.G. Ruzzelli, M. Dragone, R. Jurdak, C. Muldoon, A. Barbirato, and G. M. P. O'Hare. Poster Abstract: An Extensible Dashboard for Sensor Networks Control and Visualisation. In *Proceedings of 6th European Workshop on Sensor Networks*, EWSN 2009, 2 2009.
80. R. Gravina, A. Guerrieri, G. Fortino, F. Bellifemine, R. Giannantonio, 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)*, 10 2008.
81. CC2420: Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee Ready RF Transceiver. http://www.ti.com/product/cc2420, 2011.
82. E. Yoneki and J. Bacon. A survey of Wireless Sensor Network technologies: research trends and middleware's role. Technical Report UCAM-CL-TR-646, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue - Cambridge CB3 0FD - United Kingdom, 09 2005.
83. Min Chen, S. Gonzalez, and V. C. M. Leung. Applications and design issues for mobile agents in wireless sensor networks. *Wireless Communications, IEEE*, 14(6):20–26, 2007.
84. A. R. Silva, A. Romão, D. Deugo, and M. M. Da Silva. Towards a Reference Model for Surveying Mobile Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 4:187–231, 09 2001.
85. M. Luck, P. McBurney, and C. Preist. A Manifesto for Agent Technology: Towards Next Generation Computing. *Autonomous Agents and Multi-Agent Systems*, 9:203–252, 11 2004.
86. C. Muldoon, G. M. P. O'Hare, M. J. O'Grady, and R. Tynan. Agent Migration and Communication in WSNs. In *Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 425–430, Washington, DC, USA, 2008. IEEE Computer Society.
87. 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, 2010.
88. The Sentilla labs. http://labs.sentilla.com/, 2010.

89. G. Fortino and A. Guerrieri. Monitoring Building Indoors through Clustered Embedded Agents. In *Proceedings of Workshop on Agent Based Computing: from Model to Implementation jointly held with FedCSIS Conference*, ABC 2011, 9 2011.
90. F. Aiello, G. Fortino, A. Guerrieri, and R. Gravina. MAPS: A Mobile Agent Platform for WSNs based on 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, 5 2009.
91. 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.
92. 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, Rende, Cosenza, Italia, 7 2011.
93. D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Magazine Communications of the ACM*, 42:88–89, 3 1999.
94. D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 22–31. ACM, 2002.
95. F. Aiello, G. Fortino, and A. Guerrieri. Using Mobile Agents as Enabling Technology for Wireless Sensor Networks. In *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, pages 549–554, Washington, DC, USA, 2008. IEEE Computer Society.
96. L. Szumel, J. LeBrun, and J. D. Owens. Towards a mobile agent framework for sensor networks. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pages 79–87, Washington, DC, USA, 2005. IEEE Computer Society.
97. S. Suenaga and S. Honiden. Enabling Direct Communication Between Mobile Agents in Wireless Sensor Networks. In *Proceedings of the 1st Int.Workshop on Agent Technology for Sensor Networks (ATSN-07), jointly held with the 6th Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS-07)*, Honolulu, HI, USA, 2007.
98. Agent Factory Micro Edition (AFME). http://sourceforge.net/projects/agentfactory/files/, 2011.
99. O. Kasten and K. Römer. Beyond event handlers: programming wireless sensors with attributed state machines. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05. IEEE Press, 2005.
100. H. Zhu and R. Alkins. Towards Role-Based Programming. In *Proceedings of CSCW '06*, pages 4–8. ACM, 11 2006.
101. Mobile Agent Platform for Sun SPOT (MAPS). http://maps.deis.unical.it, 2011.
102. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
103. K. Saleh and C. El-Morr. M-UML: an extension to UML for the modeling of mobile agent-based software systems. *Information & Software Technology*, 46(4):219–227, 2004.

104. M. D. Dikaiakos, M. Kyriakou, and G. Samaras. Performance Evaluation of Mobile-Agent Middleware: A Hierarchical Approach. In *Proceedings of the 5th International Conference on Mobile Agents*, MA '01, pages 244–259. Springer-Verlag, 2002.
105. B. Qiao, K. Liu, and C. Guy. A Multi-Agent System for Building Control. In *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, IAT '06, pages 653–659. IEEE Computer Society, 2006.
106. B. A. Huberman and S. H. Clearwater. A Multi-Agent System for Controlling Building Environments. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the International Conference on Multiagent Systems (ICMAS-95)*, pages 171–176. The MIT Press, 1995.
107. P. Davidsson and M. Boman. Distributed monitoring and control of office buildings by embedded agents. *Information Sciences-Informatics and Computer Science: An International Journal - Special issue: Intelligent embedded agents*, 171:293–307, 05 2005.
108. G. Fortino and A. Guerrieri. Decentralized and Embedded Management of Smart Buildings. In *Proceedings of the Workshop on Applications of Software Agents*, WASA 2011, pages 3–7, Novi sad, Serbia, 7 2011.
109. C. Muldoon, G. M. P. O'Hare, R. Collier, and M. J. O'Grady. Agent Factory Micro Edition: A Framework for Ambient Applications. In *Proceedings of Intelligent Agents in Computing Systems Workshop (held in Conjunction with International Conference on Computational Science (ICCS)) Reading, UK. Lecture Notes in Computer Science (LNCS)*, pages 727–734. Springer-Verlag Publishers, 2006.
110. Agent Factory. http://www.agentfactory.com, 2011.
111. A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. In *In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95)*, pages 312–319, 1995.
112. O. Gama, C. Figueiredo, P. Carvalho, and P. M. Mendes. Towards a Reconfigurable Wireless Sensor Network for Biomedical Applications. In *Proceedings of the 2007 International Conference on Sensor Technologies and Applications*, SENSORCOMM '07, pages 490–495. IEEE Computer Society, 2007.
113. V. Shnayder, B. Chen, K. Lorincz, T.R.F. Fulford-Jones, and M. Welsh. Sensor networks for medical care - Technical Report TR-08-05. Technical Report TR-08-05, Division of Engineering and Applied Sciences, Harvard University, 2005.
114. C. Lombriser, N. B. Bharatula, D. Roggen, and G. Tröster. On-body activity recognition in a dynamic sensor network. In *Proceedings of the ICST 2nd international conference on Body area networks*, BodyNets '07, pages 17:1–17:6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 06 2007.
115. 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.
116. Ember Web Site. http://www.ember.com, 2011.
117. ZStack website. http://www.ti.com/tool/z-stack, 2011.
118. G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio. SPINE2: developing BSN applications on heterogeneous sensor nodes. In *Proceedings of IEEE Symposium on Industrial Embedded Systems*, SIES'09, 7 2009.
119. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley Interscience, New York, 2. edition, 2001.

120. N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman. Activity recognition from accelerometer data. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence (IAAI 2005) - Volume 3*, pages 1541–1546. AAAI Press, 2005.
121. Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE Network Magazine*, 18(1):15–21, 2004.
122. C. Lombriser, D. Roggen, M. Stäger, and G. Tröster. Titan: A Tiny Task Network for Dynamically Reconfigurable Heterogeneous Sensor Networks. In *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS)*, pages 127–138, 02 2007.
123. B. Najafi, K. Aminian, A. Paraschiv-Ionescu, F. Loew, C. J. Bula, and P. Robert. Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. *IEEE transactions on bio-medical engineering*, 50(6):711–723, 6 2003.
124. B. P. L. Lo and G. Z. Yang. Key Technical Challenges and Current Implementations of Body Sensor Networks. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, 04 2005.
125. SPINE documents and software. http://spine.tilab.com, 2011.
126. 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 *5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, IDAACS 2009, pages 682–687. IEEE Press, 2009.
127. G. Fortino, A. Guerrieri, F. Bellifemine, and R. Giannantonio. Platform-independent development of collaborative Wireless Body Sensor Network applications: SPINE2. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, SMC 2009, pages 3144–3150. IEEE, 10 2009.
128. B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20:19–25, 09 2003.
129. H. Wada, P. Boonma, J. Suzuki, and K. Oba. Modeling and executing adaptive sensor network applications with the Matilda UML virtual machine. In *Proceedings of the 11th IASTED International Conference on Software Engineering and Applications (SEA)*, pages 216–225. ACTA Press, 11 2007.
130. D. A. Sadilek. Prototyping Domain-Specific Languages for Wireless Sensor Networks. In *Proceedings of the 4th International Workshop on Software Language Engineering*, 2007.
131. M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri. A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application. In *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, pages 515–522. Ieee, 2008.
132. Tmote SKY TelosB. http://www.sentilla.com/moteiv-transition.html, 2011.
133. T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1 1967.
134. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
135. Aiello F., Fortino G., Galzarano S., Gravina R., and Guerrieri A. *Signal processing in-node frameworks for Wireless Body Sensor Networks: from low-level to high-level approaches*. Pan Stanford publishing, 2011.

136. 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, 5 2010.
137. A. Schoofs, A. Guerrieri, D. T. Delaney, G. M. P. O'Hare, and A. G. Ruzzelli. ANNOT: Automated Electricity Data Annotation Using Wireless Sensor Networks. In *Proceedings of the 7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks*, SECON 2010, pages 244–252. IEEE, 6 2010.
138. D. L. Carní, 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, 9 2011.