Engineering Applications of Artificial Intelligence 24 (2011) 1147–1161

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks

F. Aiello [a], F.L. Bellifemine [b], G. Fortino [a,*], S. Galzarano [a], R. Gravina [a,c]

[a] Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), Italy
[b] Telecom Italia, Turin, Italy
[c] Telecom WSN Lab, Berkeley, CA, USA

ARTICLE INFO

Available online 23 July 2011

Keywords:
Wireless body sensor networks
Software agents
In-node signal processing
Real-time human activity monitoring
Mobile Agents Platform for Sun SPOT

ABSTRACT

Nowadays wireless body sensor networks (WBSNs) have great potential to enable a broad variety of assisted living applications such as human biophysical/biochemical control and activity monitoring for health care, e-fitness, emergency detection, emotional recognition for social networking, security, and highly interactive games. It is therefore important to define design methodologies and programming frameworks which enable rapid prototyping of WBSN applications. Several effective application development frameworks have been already proposed for WBSNs designed for TinyOS-based sensor platforms, e.g. CodeBlue, SPINE, and Titan. In this paper we present an application of MAPS, an agent framework for wireless sensor networks based on the Java-programmable Sun SPOT sensor platform, for the development of a real-time WBSN-based system for human activity monitoring. The agent-oriented programming abstractions provided by MAPS allow effective and rapid prototyping of the sensor-side software. In particular, the architecture of the developed system is a typical star-based WBSN composed of a coordinator node and two sensor nodes located respectively on the waist and the thigh of the monitored assisted living. The coordinator relies on a JADE-based enhancement of the SPINE coordinator and allows configuring sensors, receiving their data, and recognizing pre-defined human activities. On the other hand, each sensor node runs a MAPS-based agent that performs sensing of the 3-axial accelerometer sensor, computation of significant features on the acquired data, feature aggregation and transmission to the coordinator. The experimentation phase of the prototype, which allows evaluating the obtainable monitoring performances and activity recognition accuracy, is described. Moreover, a comparison of the monitoring system based on MAPS, AFME and SPINE in terms of programming effectiveness and system performances is discussed.

1. Introduction

Wireless sensor networks (WSNs) are currently emerging as one of the most disruptive technologies enabling and supporting next generation ubiquitous and pervasive computing scenarios (Sohraby et al., 2007). WSNs are capable of supporting a broad array of high-impact applications in several domains such as disaster/crime prevention, military, environment, logistics, health care, and building/home automation. WSNs applied to the human body are usually called Wireless Body Sensor Networks (WBSNs) (Yang, 2006). WBSNs are conveying notable attention as their real-world applications aim at improving the quality of life of human beings by enabling continuous, real-time and non-invasive medical assistance at low cost. Health-care applications in which WBSNs could be greatly useful include early detection or prevention of diseases, elderly assistance at home, e-fitness, rehabilitation after surgeries, motion and gestures detection, cognitive and emotional recognition, medical assistance in disaster events, etc.

The design and programming of applications based on WBSNs are complex tasks. This is mainly due to the challenge of implementing signal processing intensive algorithms for data interpretation on wireless nodes that are very resource constrained and have to meet hard requirements in terms of wearability and battery duration as well as computational and storage resources. This is challenging because WBSN applications usually require high sensor data sampling rates which affects real-time data processing and transmission capabilities since computational power and available bandwidth of the WBSN infrastructure are generally scarce. Indeed, efficient implementation of WBSN applications requires appropriate allocation of the limited resources on the nodes in terms of energy,

* Corresponding author. Tel.: +39 0984 494063; fax: +39 0984 494713.
E-mail addresses: faiello@si.deis.unical.it (F. Aiello), fabioluigi.bellifemine@telecomitalia.it (F.L. Bellifemine), g.fortino@unical.it (G. Fortino), sgalzarano@deis.unical.it (S. Galzarano), rgravina@deis.unical.it (R. Gravina).

0952-1976/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2011.06.007

memory and processing. This is especially critical in signal processing systems, which usually have large amounts of data to process and transmit. Current embedded operating systems do not address such high level and complex requirements as they mainly focus on hardware abstraction, power management, routing, security and synchronization algorithms, and sometimes on general-purpose data structures, dynamic memory management, and multi-tasking.

To deal with the aforementioned issues, several software frameworks have been developed such as CodeBlue (Malan et al., 2004), Titan (Lombriser et al., 2009), and SPINE (Fortino et al., 2009). They aim at decreasing development time and improving interoperability among signal processing intensive applications based on WBSNs. In particular, they basically rely on a star-based network architecture, which is organized into a coordinator node and a set of sensor nodes. Moreover, they are developed in TinyOS at the sensor node side and in Java at the coordinator node side.

However, apart from the adoption of effective frameworks, we believe that the exploitation of the agent-oriented programming paradigm to develop WBSN applications could provide more effectiveness as demonstrated by the application of agent technology in several key application domains (Luck et al., 2004).

In this paper, we therefore propose an agent-oriented approach to develop WBSN applications based on the MAPS (Mobile Agent Platform for Sun SPOTs) framework (Aiello et al., 2008, 2011) that enables agent-oriented programming by offering powerful abstractions allowing rapid prototyping of WSN applications on the Sun SPOT sensor platform. The proposed approach is exemplified through the design, implementation and evaluation of an agent-based real-time human activity monitoring system. In particular, the system architecture is a typical star-based WBSN composed of a coordinator node and two sensor nodes which are located on the waist and thigh of the monitored assisted living, respectively. The coordinator relies on a Jade (2011)-based enhancement of the SPINE coordinator (Fortino et al., 2009; Signal Processing In-node Environment, 2011) and allows configuring the sensing process, receiving sensed data features, and recognizing pre-defined human activities through a KNN (K-nearest neighbor) classifier. Each sensor node executes a MAPS-based agent that performs sensing of the 3-axial accelerometer sensor, computation of significant features on the acquired data, features aggregation and transmission to the coordinator. Finally, the experimentation phase of the developed prototype allows evaluating the obtainable monitoring performances and activity recognition accuracy.

The main research contribution of this paper is twofold. On one hand, it proposes the design, implementation and evaluation of a novel agent-oriented system based on WBSNs for real-time monitoring of human activities by means of MAPS atop the Sun SPOT sensor platform. On the other hand, it provides an analysis of the effectiveness and efficiency of the application of agent frameworks, namely MAPS and AFME the only two frameworks available for Sun SPOTs so far, within the WBSN application domain; such analysis is carried out also with respect to the SPINE framework (Bellifemine et al., 2011), an open-source domain-specific framework for WBSN applications. Obtained results show that MAPS can be more effective (from the programming point of view) and efficient (from the system perspective) than AFME and SPINE in developing WBSN applications.

The rest of this paper is organized as follows. Section 2 first discusses related work for the development of WBSN applications, ranging from monolithic applications to domain-specific frameworks, and then introduces a reference architecture for WBSN applications from network and functional perspectives. In Section 3 the available agent platforms for WSNs (Agilla, ActorNet and AFME) are described and compared with MAPS. Section 4 describes the architecture and programming model of MAPS that is used to design and implement an agent-oriented signal processing in-node environment for real-time human activity monitoring that is presented in Section 5 along with the analysis of programming effectiveness and system performances of MAPS, AFME and SPINE. Finally conclusions are drawn and on-going research discussed.

2. WBSN application development

Programming WBSN applications is a complex task mainly due to the hard resource constraints of wearable devices and to the lack of proper and easy to use software abstractions. In this section we overview the available approaches for the development of applications based on WBSNs and describe a reference architecture for WBSNs that enables rapid prototyping of efficient signal processing in-node applications.

2.1. Programming frameworks

Most of previous research on WBSN applications has focused on proof-of-concept applications with the aim of demonstrating the feasibility of new context-aware algorithms and techniques, e.g. for the recognition of physical activity through accelerometer sensors or prompt detection of hearth diseases (Najafi et al., 2003; Bao and Intille, 2004; Yang, 2006). Moreover such research has considered issues related to power consumption and radio channel usage but has scarcely taken into account code reusability and modularity. One of the most relevant attempts to define a general platform able to support various WBSN applications is CodeBlue (Malan et al., 2004). CodeBlue is a framework based on TinyOS (2011) and specifically designed for integrating wireless medical sensor nodes and other devices that could be involved in a disaster response scenario. CodeBlue allows such devices to discover each other, report events, and establish communications. It relies on a publish/subscribe-based data routing framework in which sensors publish relevant data to a specific channel and end-user devices subscribe to channels of interest. CodeBlue provides end-user devices with a query interface to retrieve data from previously discovered sensor nodes. While it is possible to select sensor types or physical node identifiers as data sources, configure the data rate and define in-node threshold-based filters to avoid unnecessary data to be transmitted, more sophisticated in-node processing of the sensor data is not supported. A different approach is proposed by Titan (Lombriser et al., 2009), which is also implemented in TinyOS. Titan is a middleware for distributed signal processing in WSNs that supports implementation and execution of context recognition algorithms in dynamic WSN environments. Titan represents data processing by a data flow from sensors to recognition results. The data is processed by tasks which implement elementary computations. Tasks and their data flow interconnections define a task network, which runs on the sensor network as a whole. Tasks are mapped onto each sensor node according to the sensors and the processing resources it provides. Titan dynamically reprograms the WSN to exchange context recognition algorithms and handle defective nodes, variations in available processing power, or broken communication links. The architecture of Titan is composed of several software components, which enhance modularity. Although CodeBlue and Titan raise the programming abstraction level by offering general-purpose platforms and middlewares for effectively developing signal processing applications in WBSNs, they are sometimes too general for providing efficient solutions in specific application domains. Thus, domain-specific frameworks (Bao and Intille, 2004; Bellifemine et al., 2011) have been proposed which are positioned in the middle between application-specific code and middleware approaches. They specifically address and standardize the core challenges of WSN design within a particular application domain. While providing high efficiency, such frameworks allow for a more effective development of customized applications with little or no additional hardware configuration and with the provision of high-level programming abstractions tailored for the reference application domain. A notable example of such approach is represented by the SPINE framework (Fortino et al., 2009; Bellifemine et al., 2011; Signal Processing In-node Environment, 2011). SPINE provides libraries of protocols, utilities and processing functions, and a lightweight Java API that can be used by local and remote applications to manage the sensor nodes or submit service requests. By providing these abstractions and libraries, which are common to most signal processing algorithms used in WBSNs for sensor data analysis and classification, SPINE also provides flexibility in the allocation of tasks among the WBSN nodes and allows the exploitation of implementation tradeoffs. Currently SPINE is implemented for several sensor platforms based on TinyOS (2011) and Z-stack (2011) by using the programming paradigms offered by such platforms (event and component-based programming in TinyOS and C programming in Z-Stack) and is being effectively applied to the development of applications in the health care domain (Iyengar et al., 2008). In this paper we propose an agent-oriented approach which borrows the basic features characterizing the domain specific framework approach, particularly SPINE, with the aim of providing more programming effectiveness as demonstrated by the application of agent technology in several key application domains (Luck et al., 2004).

2.2. A reference system for in-node signal processing

The network architecture of the reference WBSN system for signal processing, which is derived from the SPINE system (Fortino et al., 2009; Signal Processing In-node Environment, 2011; Bellifemine et al., 2011), is organized into multiple sensor nodes and one coordinator node (see Fig. 1). The coordinator manages the network, collects, stores and analyzes the data received from the sensor nodes, and also can act as a gateway to connect the WBSN with wide area networks (e.g. Internet) for remote data access. Sensor nodes measure local physical parameters and send raw or pre-processed data to the coordinator. In this system, sensor nodes only communicate with the coordinator according to the star network topology, which is the most used topology in WBSN (Yang, 2006). However, the system could be easily extended to support also direct and multi-hop communications among sensor nodes. In the reference architecture a sensor node is associated with a single coordinator; a possible extension is to allow sensor nodes to be associated and communicate with multiple coordinators. A scenario where such architecture could be used is when a human wearing sensor nodes moves across locations; in this case such sensors should connect to a different coordinator at each different location. The software architecture of the system consists of two main components, implemented, respectively, on the coordinator (e.g. a PC or a PDA/smartphone)

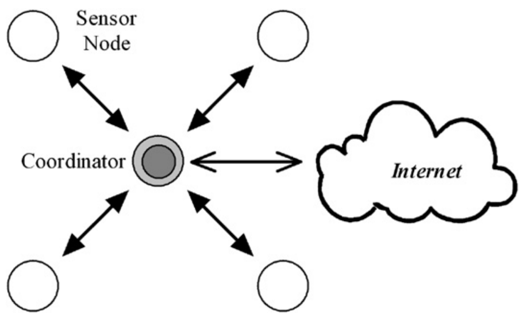


Fig. 1. Reference system network architecture.

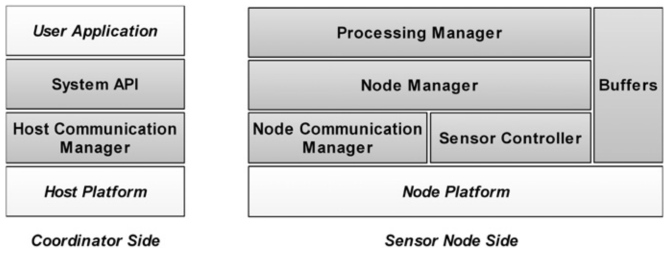


Fig. 2. Software architecture layers of the system from the functional perspective.

and on the WBSN sensor nodes. Fig. 2 shows a schema of the architecture from a functional point of view.

At the coordinator side, an interface to the WBSN which is placed between user applications and the hardware and software host platform is made available. User applications manage the WBSN through a system API. The top level of the software architecture at the coordinator side allows registered applications to be notified of the following events generated by the WBSN: discovery of new nodes, sensor data communication, node alarms, and system messages such as low battery warnings. Commands issued by the user application and network-generated events are both coded in lower-level messages and decoded in higher-level information by the Host Communication Manager according to a specific over-the-air protocol. This component handles packets generation and retrieval and is interfaced with specific software components of the host platform to access the physical radio module for transmitting/receiving packets to/from the WBSN. At the sensor node side, the software framework provides abstractions of hardware resources such as sensors and the radio, a default set of ready-to-use common signal processing functions and, most importantly, a flexible and modular architecture to be customized and extended to support new physical platforms and sensors and introduce new signal processing services. In particular, the Node Communication Manager acts as the counterpart of the Host Communication Manager. The Sensor Controller manages and abstracts the sensors on the node platform, providing a standard interface to the diverse sensor drivers. It is responsible of sampling the sensors and storing the sensed data in properly defined Buffers. The Node Manager is the central component, responsible for interpreting the remote requests and dispatching them to the proper components. Finally, the Processing Manager consists of a dispatcher for the actual processing services and a standard interface for user-defined services integration.

3. Agent-based platforms for wireless sensor networks

Agents are supported by agent platforms (APs) which basically provide an API for developing agent-based applications, and an agent server able to execute agents by providing them with basic services such as communication, migration and node resource access. Although many APs exist for conventional distributed systems (Luck et al., 2004), developing flexible and efficient APs for WSNs is a challenging and very complex task due to the currently available resource-constrained sensor nodes and related operating systems (Vinyals et al., 2011). Very few APs for WSNs have been to date proposed and actually implemented. In the following, we first introduce Agilla and ActorNet, the most significant available research prototypes based on TinyOS, and then describe in more details Agent Factory Micro Edition (AFME), which runs on Java Sun SPOTs, as AFME is the only available Java-based agent platform related to MAPS. Finally we provide a qualitative comparison among Agilla, ActorNet, AFME and MAPS.

3.1. TinyOS-based agent platforms

Agilla (Fok et al., 2005) is an agent-based middleware developed on TinyOS (Gay et al., 2003) and supporting multiple agents on each node. As shown by its software architecture (see Fig. 3), Agilla provides two fundamental resources on each node: a tuplespace and a neighbors list. The tuplespace represents a shared memory space where structured data (tuples) can be stored and retrieved, allowing agents to exchange information in a temporally decoupled way. A tuplespace can be also accessed remotely. The neighbors list contains the address of all one-hop-distant nodes, needed when an agent has to migrate. Agents can migrate carrying their code and state, but do not carry their tuples that are locally stored on a tuplespace. Packets used for communication between nodes (e.g. for agent migration/cloning, remote tuples accessing) are very small to minimize message losses, whereas retransmission techniques are also adopted. Although Agilla is developed for TinyOS platforms, agents are not programmed through nesC but a proprietary ISA (Instruction Set Architecture) was specifically defined for their programming.

ActorNet (Kwon et al., 2006) is an agent-based platform specifically designed for TinyOS/Mica2 sensor nodes. To overcome the difficulties in allowing code migration and interoperability due to the strict coupling between applications and sensor node architectures, ActorNet exposes services like virtual memory, context switching, and multi-tasking. Thanks to these features, it effectively supports agents programming by providing a uniform computing environment for all agents, regardless of hardware or operating system differences. The ActorNet architecture is depicted in Fig. 4. The ActorNet language used for high-level agent programming, has syntax and semantics similar to those of Scheme (Kent Dybvig, 1987) with proper instruction extension.

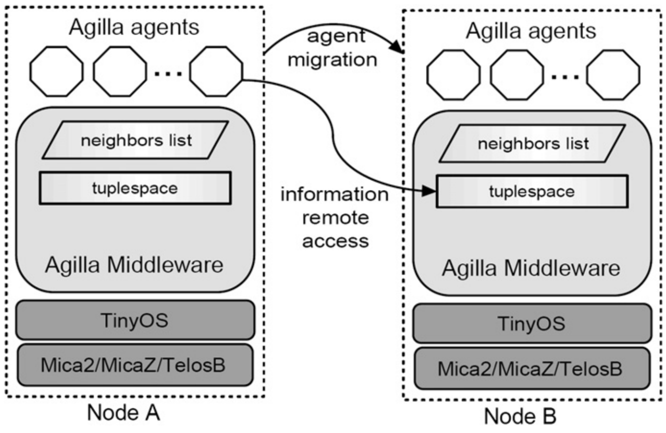


Fig. 3. Architecture of Agilla.

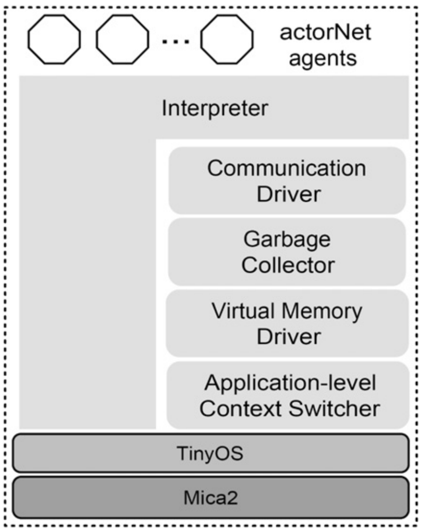


Fig. 4. Architecture of ActorNet.

3.2. Agent Factory Micro Edition

AFME (Muldoon et al., 2006, 2008; Agent Factory, 2011) is an open-source, lightweight, and J2ME Mobile Information Device Profile (MIDP) compliant agent platform based upon the Agent Factory Framework (2011) and intended for wireless pervasive systems. Thus, AFME has not been 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 agent-based WSN applications. AFME is based on the Believe-Desire-Intention (BDI) paradigm (Rao and Georgeff, 1995), in which agents follow a sense-deliberate-act cycle. To facilitate the creation of BDI agents the framework supports a number of system components that developers have to extend when building their applications: perceptors, actuators, modules, and services. Perceptors and actuators enable agents to sense and 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. First, the perceptors are fired and their sensing operations generate beliefs, which are added to the agent's belief set. A belief is a symbolic representation of information related to the agent's state or to the environment. Second, the agent's desires are identified using resolution-based reasoning, a goal-based querying mechanism commonly employed within Prolog interpreters. Third, the agent's 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 belief and commitment, is used to encode an agent's behavior by specifying rules defining the conditions under which commitments are adopted. The imperative Java code is instead used to encode perceptors and actuators. A declarative rule is expressed through the following form: $b1, b2, bn > doX$; where $b1,b2, \ldots ,bn$ 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 agent's belief set, the imperative code enclosed into the actuator associated to the symbolic string doX is executed.

The AFME platform architecture is shown in Fig. 5. It comprises a scheduler, a group of agents, and several platform services needed for supporting agent communication and migration.

To improve reuse and modularity within AFME, actuators, perceptors, and services are prevented from 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 modified without having to modify actuators and 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

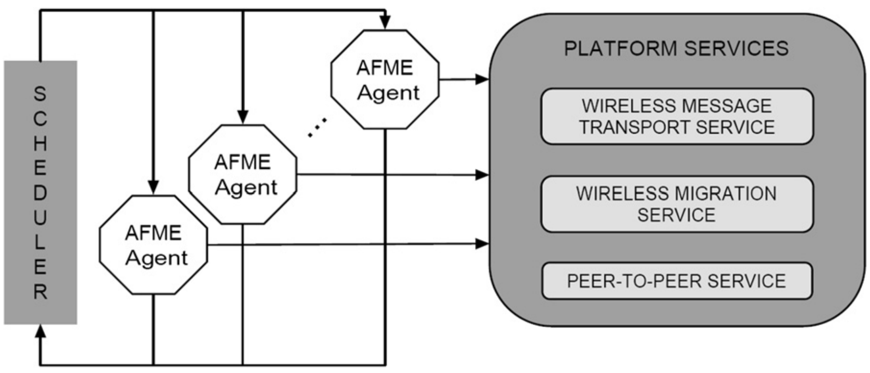


Fig. 5. Architecture of AFME.

Table 1
Feature comparison of the analysed agent platforms.

	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	AFAPL/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

are interchangeable because they interact without directly referencing one another.

3.3. A comparison

In Table 1, a comparison among the aforementioned agent platforms with respect to seven characteristics (migration, multitasking, communication model, programming language, agent model, intentional agents, sensor platforms) is reported. Agent migration and multitasking, which allows for the execution of multiple agents on the same node, is supported by all the systems. The agent communication model of Agilla is centered on local tuple space where agent can asynchronously insert tuples and take tuples left by other agents. Conversely the communication model of the other systems is based on (unicast and broadcast) message passing. The programming language and model is different among the systems. Agilla is based on a proprietary low-level language composed of an assembler-like instruction set which makes programming of complex agents very difficult. ActorNet is based on a functional Scheme-like language whereas MAPS and AFME on the Java language. Indeed, MAPS uses a finite state machine model to define agent behaviour whereas AFME employs a more complex BDI-like model based on the AFAPL language. Intentional agents are therefore only offered by AFME. Agilla and ActorNet run on motes; in particular Agilla on Mica2, MicaZ, and TelosB, whereas ActorNet currently only on Mica2. On the contrary, MAPS and AFME are based on Sun SPOTs.

All the four compared systems are effective solutions for agent-oriented programming of WSNs even though they are based on (very) different programming abstractions and architectures. However, being MAPS and AFME specifically conceived for Sun SPOT sensor technology, which is more capable than the sensor *mote* technology on which Agilla and ActorNet are based (see the last row of Table 1), they are less limited in terms of resources so more capable agents can be defined. Moreover, as MAPS offers FSM-based agents, such programming paradigm is very appealing for programmers and designers of embedded systems who usually exploit programming tools based on FSMs or their derivatives.

4. MAPS: a Java-based agent platform for sun spots

MAPS is a novel Java-based framework for wireless sensor networks based on Sun SPOT technology (Aiello et al., 2008, 2011; Mobile agent, 2011) which enables agent-oriented programming of WSN applications. MAPS has been appositely defined for resource-constrained sensor nodes according to the following requirements:

- Component-based lightweight agent server architecture to avoid heavy concurrency models and, therefore, exploit cooperative concurrency to execute agents.
- Lightweight agent architecture to efficiently execute and migrate agents.
- Minimal and plugable core services involving agent migration, agent naming, agent communication, activity timing, sensor resource capability access (actuators, input signalers, flash memory, and battery).
- Plug-in-based architecture extensions through which any other service should be defined in terms of one or more dynamically installable components implemented as single or cooperating (mobile) agent/s.
- Java as programming language for the agent system and (mobile) agents.

In the following subsections we focus on the system architecture and the agent programming model.

4.1. System architecture

The MAPS architecture, which is shown in Fig. 6, consists of the following basic components:

- The *mobile agent* (MA) is the high-level components of each agent-based application. Its implementation is formed of two

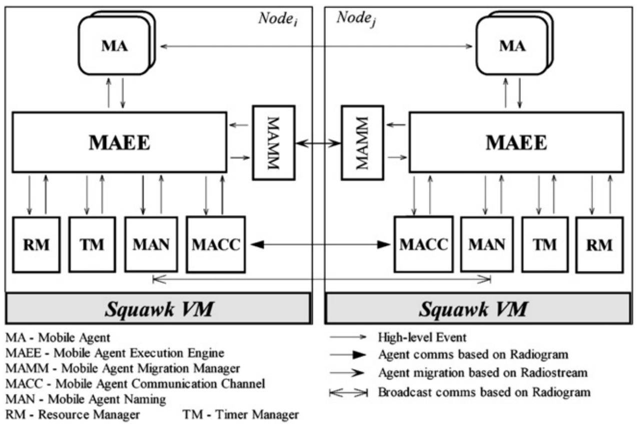

Node_i
Node_j
MA
MAEE
MAMM
RM
TM
MAN
MACC
Squawk VM
MA - Mobile Agent
MAEE - Mobile Agent Execution Engine
MAMM - Mobile Agent Migration Manager
MACC - Mobile Agent Communication Channel
MAN - Mobile Agent Naming
RM - Resource Manager
TM - Timer Manager
High-level Event
Agent comms based on Radiogram
Agent migration based on Radiostream
Broadcast comms based on Radiogram

Fig. 6. Architecture of MAPS.

parts: an object formalizing the agent behavior based on a multi-plane state machine (see Section 4.2) embedded into an architectural component directly connected to the MAEE (see below).

- The *mobile agent execution engine* (MAEE) supports the execution of agents by means of an event-based scheduler enabling lightweight concurrency. It handles each event emitted by or to be delivered at an MA through decoupling event queues. The MAEE interacts with the other core components (see below) to fulfill service requests issued by MAs.
- The *mobile agent migration manager* (MAMM) supports the migration of agents from one sensor node to another. In particular, the MAMM is based on the feature of Isolate (de)hibernation provided by the Sun SPOT (2011) environment and is therefore able to stop and hibernate an MA, serialize it into a byte array and transmit it to the target sensor node. On the migration target sensor node, the MAMM can receive a message containing a serialized MA, deserialize, dehibernate and resume it. The agent serialization format includes data and execution state whereas the code should already reside at the destination node (this is a current limitation of the Sun SPOTs which do not support dynamic class loading and code migration).
- The mobile agent communication channel (MACC) enables inter-agent communication based on asynchronous messages supported by the Radiogram protocol. Messages can be unicast or broadcast.
- The *mobile agent naming* (MAN) provides agent naming based on proxies to support the MAMM and MACC components in their operations. The MAN also manages the (dynamic) list of the neighbor sensor nodes which is updated through a beaconing mechanism based on broadcast messages.
- The *timer manager* (TM) provides the timer service which allows for the management of timers to be used for timing MA operations.
- The *resource manager* (RM) provides access to the resources of the Sun SPOT node: sensors (3-axial accelerometer, temperature, light), switches, leads, battery, and flash memory.

Indeed, the core components for agent migration and resource access can be plugged and unplugged to allow for optimization of computing and memory resources according to specific applications needs. As an example, when agent mobility is not a requirement of the application, the MAMM component can be unplugged so saving memory space in central memory and on the flash.

To allow for different application needs and resource requirements, three different versions of the MAEE are available (see Fig. 7) which support three different implementations of the agent architectural component. In particular, such component

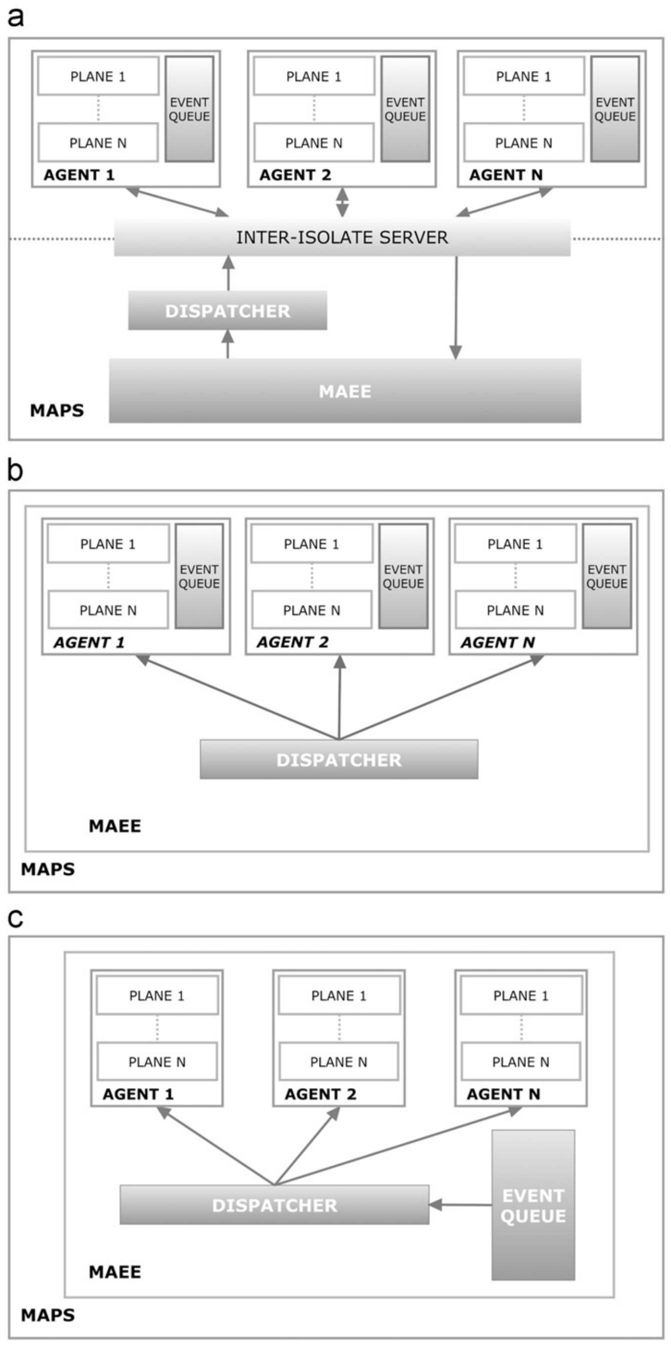


Fig. 7. Three different architectures of MAEE: (a) isolate-based, (b) thread-based, and (c) object-based.

can be: (a) a single-threaded Isolate according to the Java Sun SPOT libraries, (b) a Java thread, or (c) a Java object. Such three solutions can be used according to specific application contexts. The first solution is the only one supporting mobility as agent migration is only supported in terms of Isolate migration. The second and third solutions are exploited when migration is not necessary and more execution efficiency is required. In particular, in the first solution, agents are connected to the MAEE through a mediator component called inter-isolate server which introduces internal communication overhead as agents are in their own Isolate; in the second and third solutions, agents are in the same isolate of the MAEE so communication between agents and MAEE is based on direct object references. The last solution is the most lightweight in terms of memory requirements as an agent is only formed by a non-thread-based composite object.

4.2. Agent programming model

The main programming abstractions of MAPS are Agents and Events (see Fig. 8).

Agents are active entities, uniquely identified by an identifier, whose behavior is modeled as a multi-plane state machine (MPSM) (Bölöni and Marinescu, 2000). The MPSM consists of a set of planes, global variables and global functions. Each plane may represent the behavior of the MA in a specific role so also enabling role-based programming. In particular a plane is composed of local variables, local functions, and an Event-Condition-Action (ECA) ruled automaton that represents the dynamic behavior of the MA in that plane. The automaton 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 global and local 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 ECA automaton is), the event with the event name E occurs and $[C]$ holds. If the transition fires, A is executed and the state transition finally takes place. In particular, the atomic action can contain global/local variable and functions to carry out computation, and, particularly, the core primitives (see Fig. 9) to request specific services. As agents interact through events, the delivery of an event at agents is asynchronous and carried out by the event dispatcher (a component of the MAEE, see Fig. 7) which inserts the event in the agent queue. Once the ECA automaton is idle (i.e. the handling of the last delivered event is completed), a new event is fetched out from the queue and handled by one or more planes. It is worth noting that the MPSM-based agent behavior programming allows exploiting the benefits deriving from three main paradigms for WSN programming (Yoneki and Bacon, 2005): event-driven programming, state-based programming and mobile agent-based programming.

$Agent = < Id, MPSM, EQ >$
where:
Id is the unique agent identifier,
$MPSM$ is the agent multiplane state machine,
EQ is the agent event queue;
$MPSM = < GV, GF, \{P_1, ..., P_n\} >$
where:
GV is the set of global variables,
GF is the set of global functions,
$\{P_1, ..., P_n\}$, with $n \geq 1$, is the set of one or more planes;
$P_i = < LV_i, LF_i, FSM_i >$
where,
LV_i is the set of local variables of the plane i,
LF_i is the set of local functions of the plane i,
FSM_i is the finite state machine of the plane i

$Event = < sourceID, targetID, type, params, duration >$
where:
$sourceID$ is the event source agent/component identifier
$targetID$ is the event target agent/component identifier
$type$ is the event type
$params$ are the event data
$duration$ specifies the event duration type

Fig. 8. Formalization of agents and events.

```
public void send(String sourceMA, String targetMA, Event
            message, boolean local);
```

sourceMA = message sender agent
targetMA = message receiver agent
message = event encapsulating message content and transmission mode (unicast or broadcast)
local = true for local messages, false otherwise

```
public boolean create(String agent, String [] params,
              String nodeID);
```

agent = agent class name
params = agent creation parameters
nodeID = id of node on which the agent should be created

```
public boolean create(String agentID, String agent,
              String [] params, String nodeID);
```

agentID = the ID of the agent to be created, in the create method above the agentID is automatically created by the system and returned to the creator by an event

```
public boolean clone(String agentID, String cloneId, String loc)
```

agentID = identifier of the agent to be cloned
cloneId = identifier of the cloned agent
loc = node location of the cloned agent

```
public boolean migrate(String agentID, String loc)
```

agentID = identifier of the agent to be migrated
NodeLoc = target location of the MA | ALL neighbors

```
public String setTimer(boolean periodic, long timeout,
              Event backEvent);
```

periodic = true if the timer is periodic, false if one-shot
timeout = expiration time in msec
backEvent= the event notifying the timer expiration
The String return value represents the timer ID

```
public void resetTimer(String timerID);
```

timerID = the ID of the timer to be reset

```
public void sense(Event backEvent);

public void actuate(Event backEvent);

public void flash(Event backEvent);

public void input(Event backEvent);
```

backEvent = event which includes the setting of the sense, actuate, flash and input operations when they are invoked. After their invocation, the operation results are inserted in the backEvent which is then asynchronously delivered to the invoking agent

Fig. 9. MAPS core primitives.

Events formalize interaction among components and between components and agents. In particular, Agents emit Events through the primitives reported in Fig. 9 for requesting the following services: (i) message transmission through the *send* primitive; (ii) agent creation, cloning and migration through the primitives *create*, *clone* and *migrate*, respectively; (iii) timer setting through the *setTimer* and *resetTimer* primitives; (iv) sensor resource handling through the primitives *sense* for sensing, *actuate* for led manipulation, *flash* for load/save data from/to the flash memory, *input* for reading switches. Emitted events are handled by the associated components which, after handling them, reply with Events that drive the agent behavior. Agents can also emit internal Events to proactively drive their behaviors; this is the basic mechanism to program goal-directed behaviors of event-driven agents as described in Fortino et al. (2010).

5. An agent-based real-time system for monitoring human activity

In this section we present an agent-oriented signal processing in-node environment specialized for real-time human activity monitoring based on WBSNs. In particular, it is able to recognize postures (e.g. lying down, sitting and standing still) and movements (e.g. walking) of assisted livings. The system is designed and implemented with MAPS at the sensor node side and through Java and JADE at the coordinator side. In Sections 5.1 and 5.2, system design, implementation and evaluation are detailed. Moreover, as the sensor-side system is also implemented with AFME and SPINE, a performance comparison between the MAPS-, AFME-, and SPINE-based versions is described. Finally, in Section 5.3, a discussion of the programming effectiveness of MAPS for the development of WBSN applications is provided.

5.1. Design and implementation

The architecture of the system, shown in Fig. 10, is organized into a coordinator and two sensor nodes according to the reference WBSN system described in Section 2.2.

The coordinator side (see Fig. 10) is based on a JADE agent that incorporates two modules of the Java-based SPINE coordinator (Fortino et al., 2009), developed in the context of the SPINE project (Signal Processing In-node Environment, 2011), which are the SPINE Manager and the SPINE Listener. In particular, the SPINE Manager is used by end-user applications (e.g. real-time activity monitoring application) for sending commands to the sensor nodes. Moreover, the SPINE Manager is responsible of capturing low-level messages and events sent from the nodes through the SPINE Listener, which integrates several sensor platform-specific SPINE communication modules (e.g. TinyOS, Z-Stack, etc), to notify registered applications with higher-level events and message content. A SPINE communication module is composed of a send/receive interface and some components that implement such interface according to the specific sensor platform and that formalize the high-level SPINE messages in sensor platform-specific messages. In this work, the SPINE Listener has been enhanced with a new MAPS/Sun SPOT communication module to configure and communicate with MAPS-based sensor nodes. Such module translates high-level SPINE messages formatted according to the SPINE OTA (Over-The-Air) protocol (Signal Processing In-node Environment, 2011) into lower-level MAPS/Sun SPOT messages through its transmitter component and vice versa through its receiver component. The JADE agent coordinator also integrates an application-specific logic for the synchronization of the two sensors (see below and Section 5.2). The SPINE-based real-time activity monitoring application was thus completely reused as well as the SPINE Manager, only the SPINE Listener was modified to account for such enhancement.

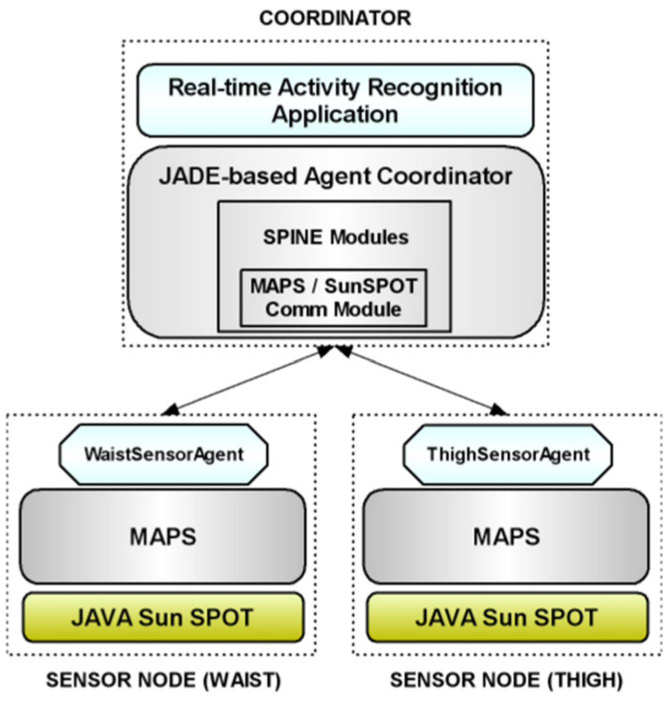


Fig. 10. Architecture of the real-time activity monitoring system.

The sensor node side (see Fig. 10) is based on two Java Sun SPOTs sensors respectively positioned on the waist and the thigh of the monitored person. In particular, MAPS is resident on the sensor nodes and supports the execution of the WaistSensorAgent and the ThighSensorAgent. Moreover, as the mobility feature of agents is not needed and in order to have higher execution performances, the thread-based version of the MAEE (see Section 4.1) is used. WaistSensorAgent and the ThighSensorAgent have the following similar step-wise cyclic behavior:

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

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

The interaction diagram depicted in Fig. 11 shows the interaction among the three agents constituting the real-time system: CoordinatorAgent, WaistSensorAgent and ThighSensorAgent. In particular, the CoordinatorAgent first sends one AGN_START event for each sensor agent to configure them with the sensing parameters (*W*, *S* and *ST*); then, it broadcasts the START event to start the sensing activity of the sensor agents. Sensor agents sends the DATA event to the CoordinatorAgent as soon as features are computed. If the CoordinatorAgent detects that the agents are not synchronized anymore, it sends the RESYNCH event to resynchronize them.

The behavior of the WaistSensorAgent is specified through 1-plane reported in Fig. 12 (the behavior of the ThighSensorAgent has the same structure but the computed features are different as discussed above). In particular, after an initialization action (A0)

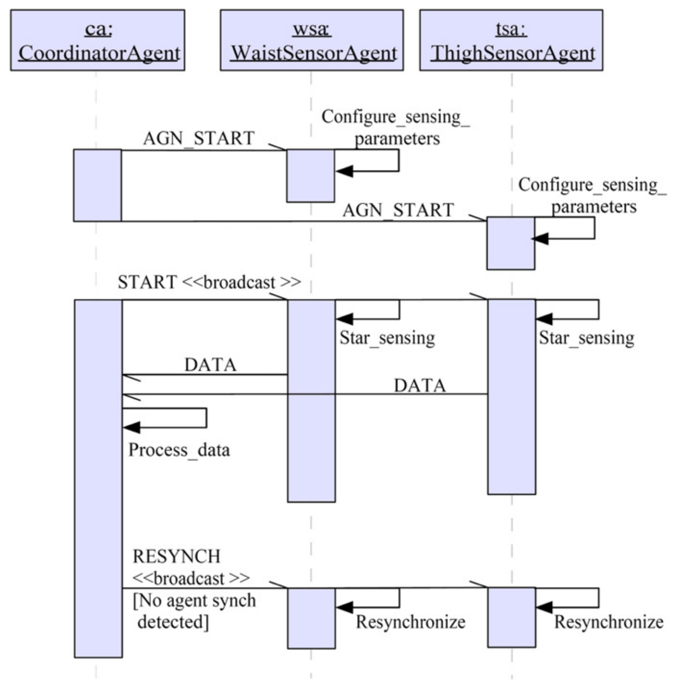


Fig. 11. Agents interaction of the real-time activity monitoring system.

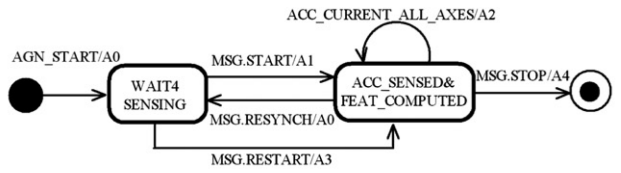
driven by the occurrence of the AGN_START event, the sensing plane goes into the WAIT4SENSING state. The MSG.START event allows starting the sensing process by the execution of action A1, which in particular performs the following steps:

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

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

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

In the ACC_SENSED&FEAT_COMPUTED state the MSG.RESYNCH might be received for resynchronization purposes (see Section 5.2); it brings the sensing plane into the WAIT4SENSING state. The MSG.RESTART brings the sensing plane back into the



```
GV
byte timestamp;
double [] windowX4FE, windowY4FE, windowZ4FE;
String basestationAddress;
LV
int W, S, ST;
byte sampleCounter;
int nextSampleIndex;
IAT91_TC timer;
double [] windowX, windowY, windowZ;
double [] resultsX, resultsY, resultsZ;
Actions
A0: initVars();
A1: initSensingParamsAndBuffers(event);
    timerSetForSensing();
    doSensing();
A2: bufferFilling(event);
    sampleCounter++;
    nextSampleIndex=(nextSampleIndex+1)%W;
    if (sampleCounter==S){
     sampleCounter==0;
     copySensingBuffersIntoBuffersForComputingFeatures();
     computeFeatures();
     transmitFeaturesComputed();
    }
    timerReset();
    doSensing();
A3: timerDisabling();
    initVars();   A1;
A4: timerDisabling();
LF
initVars():
  sampleCounter=0; nextSampleIndex=0; agent.timestamp=0;
  initSensingParamsAndBuffers(Event event):
  (WaistSensorAgent)agent.basestationAddress=event.getParam(
                             "BASESTATION_ADDRESS");
  W=Integer.parseInt(event.getParam("WINDOW_SIZE"));
  S=Integer.parseInt(event.getParam("SHIFT_SIZE"));
  ST=Integer.parseInt(event.getParam("SAMPLE_RATE_MS"));
  windowX = new double[W]; windowY = new double[W]; windowZ= new double[W];
  (WaistSensorAgent)agent.windowX4FE = new double[W];
  (WaistSensorAgent)agent.windowY4FE = new double[W];
  (WaistSensorAgent)agent.windowZ4FE = new double[W];
timerSetForSensing():
  timer = Spot.getInstance().getAT91_TC(0);
  int cnt = (int)(ST * 1000 / 2.1368);
  timer.configure(TimerCounterBits.TC_CAPT | TimerCounterBits.TC_CPCTRG |
                  TimerCounterBits.TC_CLKS_MCK128);
  timer.setRegC(cnt);
  timer.enableAndReset(); timerReset();
doSensing():
  Event accel = new Event(agent.getId(),agent.getId(),
                                Event.ACC_CURRENT_ALL_AXES,Event.NOW);
  agent.sense(accel);
bufferFilling(Event event):
  windowX[nextSampleIndex]=Double.parseDouble(
                        event.getParam(ParamsLabel.ACC_ACCEL_X_VALUE));
  windowY[nextSampleIndex]=Double.parseDouble(
                        event.getParam(ParamsLabel.ACC_ACCEL_Y_VALUE));
  windowZ[nextSampleIndex]=Double.parseDouble(
                        event.getParam(ParamsLabel.ACC_ACCEL_Z_VALUE));
timerReset():
  timer.enableIrq(TimerCounterBits.TC_CPCS);
  timer.waitForIrq(); timer.status();
timerDisabling():
  timer.disable(); timer.shutDown();
computeFeatures():
  resultsX = meanMaxMin((WaistSensorAgent)agent.windowX4FE);
  resultsY = meanMaxMin((WaistSensorAgent)agent.windowY4FE);
  resultsZ = meanMaxMin((WaistSensorAgent)agent.windowZ4FE);
trasmitFeaturesComputed():
  Event msgToServer = new Event(this.agent.getId(),
      Constants.MSG_TO_BASESTATION, Event.MSG_TO_BASESTATION, Event.NOW);
  msgToServer.setParam(ParamsLabel.AGT_BASESTATION_ADDRESS,
                     (WaistSensorAgent)agent.basestationAddress);
  msgToServer.setParam("MeanX","" + resultsX[0]);
  msgToServer.setParam("MeanY","" + resultsY[0]);
  msgToServer.setParam("MeanZ","" + resultsZ[0]);
  msgToServer.setParam("MaxY", "" + resultsY[1]);
  msgToServer.setParam("MinY", "" + resultsX[2]);
  (WaistSensorAgent)agent.timestamp=(
                        (WaistSensorAgent)agent.timestamp+1)%128;
  msgToServer.setParam("Timestamp", "" +
                        (WaistSensorAgent)agent.timestamp);
  agent.send(agent.getId(), Constants.MSG_TO_BASESTATION,
                                          msgToServer, false);
double [] meanMaxMin(double []): //omissis
```

Fig. 12. 1-Plane behavior of the WaistSensorAgent.

ACC_SENSED&FEAT_COMPUTED state for (reconfiguring and) continuing the sensing process. The MSG.STOP eventually terminates the sensing process.

5.2. System analysis

The analysis of the developed prototype involves the following two aspects (which are respectively detailed in the next two subsections):

- The performance evaluation of the timing granularity degree of the sensing activity at the sensor node and the synchronization degree or skew of the activities of the two sensor agents.
- The recognition accuracy which shows how well the human postures/movements are recognized by the system.

5.2.1. Performance evaluation

Two important issues to deal with are the timing of the sensing process in terms of admissible sampling rate and the synchronization between the operations of the two agents which is to be maintained within a maximum skew for not affecting the real-time monitoring. If such skew is overtaken, the two agents are to be re-synchronized. Indeed such two aspects are strictly correlated. In particular, as the sensor agents compute a different number of features, when the sampling rate is high, the agent computing more features (i.e. the WaistSensorAgent) takes more time to complete its operations for each S sample acquisition than the ThighSensorAgent. Re-synchronization is driven by the synchronization logic included in the developed MAPS/Sun SPOT comm module, which sends a resynchronization message (see the MSG.RESYNCH event in Fig. 12) as soon as it detects that the synchronization skew is greater than a given threshold. Detection is based on the skew time between the receptions of two messages sent by the agents that contain features referring to the same interval of S sample acquisition: if $skew >= P*S*ST$ then synchronize, where P is a percentage, $S=0.5$ W, and ST is the sampling time. Thus, the evaluation aimed at analyzing the synchronization of the sensor agents and their monitoring continuity. The defined measurements are:

- The *Packet Pair Average Time* (PPAT), which is the average reception time between two consecutive pairs of synchronized packets (same logical timestamp, see timestamp variable in Fig. 12) containing the computed features (see the MSG_TO_-BASESTATION event in Fig. 12) sent by the sensor agents. PPAT should be ideally equals to $ST*S$, i.e. the packet pair arrives each monitoring period and so there is no de-synchronization in the average.
- The *Synchronization Packet Percentage* (SPP), which is the percentage of resynchronization packets (see RESYNCH event in Fig. 12), which are sent by the coordinator for re-synchronizing the sensor agents, calculated with respect to the total number of received feature packets. SPP should be as much as possible close to 0, i.e. a few or no resynchronizations are carried out and so the monitoring can be continuous as a resynch operation usually takes 600 ms.

In particular, the experiments were carried out by fixing ST (ms)=[25, 50, 100], W (samples)=[100, 80, 40, 20, 10], and P (%)=[5, 10, 25]. Each experiment took 15 min and 50 tests per experiment were carried out. The obtained values were averaged over the 50 tests performed (also the standard deviation is reported). The values of ST and W were chosen to evaluate the system under different operating conditions: from high ($ST=25$ ms, $W=10$, $S=50\%->$ response time = 125 ms) to slow ($ST=100$ ms, $W=100$, $S=50\%->$ response time = 5 s) system response times. The system response time can directly affect the accuracy of the human activity recognition (see Section 5.5.2) as higher is the frequency of refreshing the human activity status, quicker is the capability of the system to capture human activity changes. Moreover the variation range of $P\%$ accommodates for small to medium skews.

Fig. 13 shows the obtained results for $P=25$ and 5% by varying ST and W in the ranges defined above. As can be noticed, the system cannot support an $ST=25$ ms because PPAT is always greater than the ideal value and SPP is too high. This leads to a non continuous monitoring due to very frequent resynchronizations (SPP $\geq 15\%$). An $ST=50$ ms can be supported for $P=25\%$ and $W\geq 40$ as SPP is maximum 8% so slightly impacting the monitoring continuity. The best results are obtained with $ST=100$ ms, $P=25\%$ and $W\geq 20$; they guarantee monitoring continuity due to an SPP $\approx 0\%$ and regularity as experimented PPAT $\approx$ ideal PPAT for $W\geq 20$. If $P=5\%$ and $W=[10, 20]$ or $P=25\%$ and $W=10$, an $ST=100$ ms is not a good value either because an out-of-limits skew frequently occurs.

It is worth noting that even though a lower ST would allow a more frequent monitoring, the considered human activities can be well captured by an $ST=100$ ms and $W=20$ (which implies a response time = 1 s) as demonstrated by the experimental results obtained from the carried-out real-time human activity monitoring (see Section 5.2.2).

To compare the efficiency of MAPS, AFME and SPINE, the node-side implementation of the system was also carried out with AFME whereas the implementation with SPINE was already documented in Bellifemine et al. (2011). The experiments were carried out by fixing ST (ms)=[25, 50, 100], W (samples)=[40, 20], and P (%)=[5, 25]. Each experiment took 15 min and 50 tests per experiment were carried out. Figs. 14 and 15 show the obtained results, which are the average values of the 50 tests (also the standard deviation is reported). As can be noticed, all the systems cannot support an $ST=25$ ms because PPAT is always greater than the ideal value and SPP is too high. This leads to a non continuous monitoring due to the very frequent resynchronization (SPP ≥ 20 for $W=20$ and $S=10$). The best results are obtained with $ST=100$ ms, $P=25\%$ and $W=20$; they guarantee monitoring continuity due to an SPP $\approx 0\%$ and regularity as experimented PPAT $\approx$ ideal PPAT for $W=20$. If $W=20$ and $P=5\%$, $ST=100$ ms is not a good value either because an out-of-limits skew frequently occurs. Although the AFME implementation performs better than the MAPS implementation, the AFME implementation collapses in the case $W=20$, $S=10$ and $P=5\%$. SPINE performs better for the parameters $ST=100$, $W=40$, and $P=25\%$ whereas it has lower performance in the other cases. On the basis of the obtained results we can state that MAPS on Sun SPOT shows comparable performances with SPINE on TelosB sensors, which is a domain-specific framework for WBSNs, so confirming its suitability for supporting efficient WBSN applications. In addition in Table 2 a comparison among the sensor-node-side applications based on MAPS, AFME and SPINE with respect to RAM usage and code dimension out of the available memory resources is reported. Both MAPS and AFME requires more memory than SPINE; however, the Sun SPOTs are wireless sensors more capable than the TelosB Sky-motes so the percentages of used memory by MAPS and AFME are much less than SPINE. It is finally worth noting that MAPS performs slightly better than AFME.

5.2.2. Recognition accuracy

The activity monitoring system integrates a classifier based on the K-Nearest Neighbor algorithm (Cover and Hart, 1967) that is capable of recognizing postures and movements defined in a training phase. The classifier was setup through a training phase and tested considering the following parameter setting: $ST=100$ ms, $W=20$ ($S=10$), $P=25\%$. Accordingly, the features (Min, Max and Mean) are computed on 20 sampled data every new 10 samples acquired by the sensors. The training phase used a KNN-based classifier parameterized with $K=1$ and the Manhattan distance which performs quite well as classes (lying down, sitting, standing

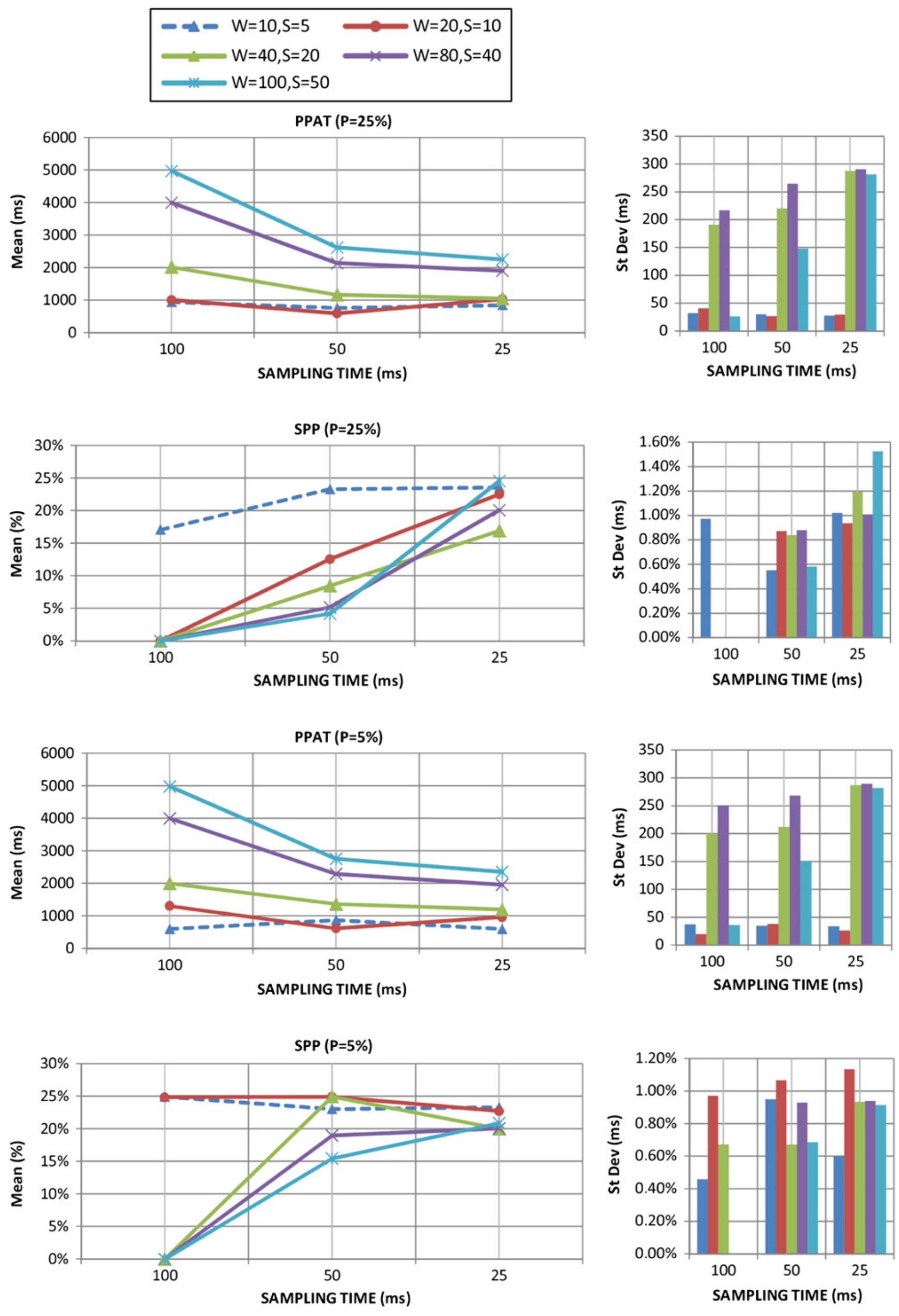


Fig. 13. Analysis of the synchronization of the MAPS sensor agents: PPAT and SPP for $P=25\%$ and $P=5\%$ by varying W.

still and walking) are rather separate and scarcely affected by noise. The test phase is carried out by considering the pre-defined sequence of postures/movements represented by the state machine reported in Fig. 16. Accordingly, the obtained classification accuracy results are reported in Fig. 17. As can be noted after a transitory period of 5 s from one state to another, all the postures/movements are recognized with an accuracy of 100%. The state transitions more difficult to recognize are STA→SIT, WLK→STA, and SIT→LYG, whereas the transition STA→WLK is recognized as soon as it occurs. The obtained results are good and encouraging if compared with other works in the literature which use more than two sensors on the human body to recognize activities (Maurer et al., 2006).

5.3. *On the programming effectiveness of MAPS*

While the previous section has shown that MAPS provides enough efficiency to support the requirements of real-time recognition of human activities, in this section, we discuss the programming effectiveness of the agent-oriented approach based on MAPS for the development of WBSN applications according to the experience gained in the development of the presented agent-based system and of a wide range of SPINE-based WBSN applications (Gravina et al., 2010). Programming effectiveness refers to two main aspects: (i) suitable programming abstractions for an effective modeling and prototyping of the system

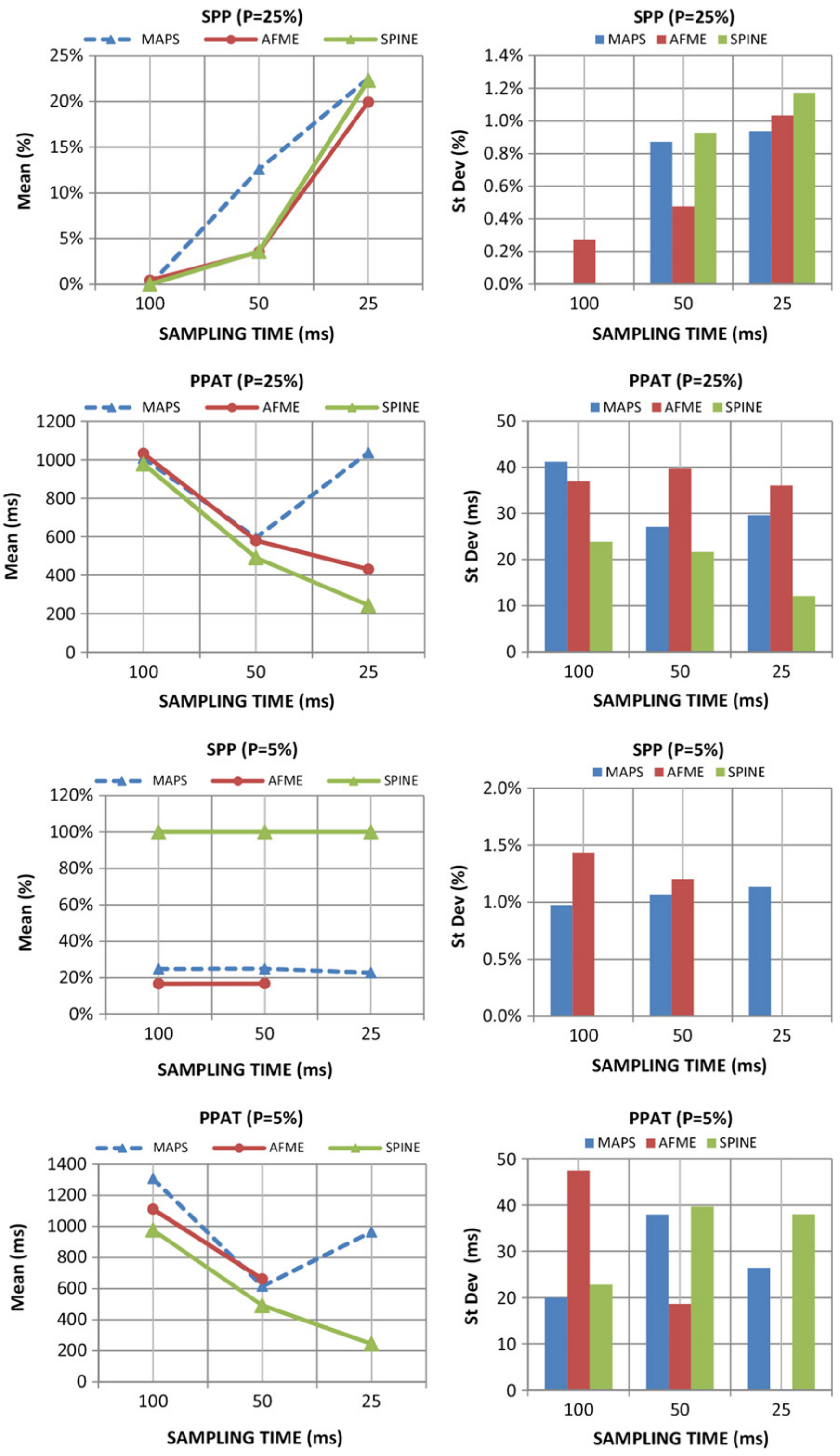


Fig. 14. Comparison of the synchronization between MAPS, AFME and SPINE sensor agents: PPAT and SPP for $P=25\%$ and $P=5\%$ and $W=20$, $S=10$.

behavior (notably including resource constrained operations) and (ii) usability of such abstractions to create new applications. Such two aspects should be carefully considered in the specific application domains in which MAPS is currently applied such as real-time human monitoring systems based on WBSNs. MAPS provides agent-oriented programming abstractions which are suitable to model not only WBSN systems but also general-purpose WSN systems in terms of multi-agent systems: (i) FSM-based behavior that can model both reactive and proactive agents; (ii) event-driven interface that allows for an easy access to the agent system and the sensor-node resources; and (iii) message-based interaction that enables direct and broadcast communications among agents. The development of the activity monitoring system in AFME also provided useful insights for the comparison of the programming effectiveness between MAPS and AFME. As described in Section 3, the programming abstractions of AFME to specify the agent behavior, which is based on a BDI-like model, are very different from those of MAPS, even though the

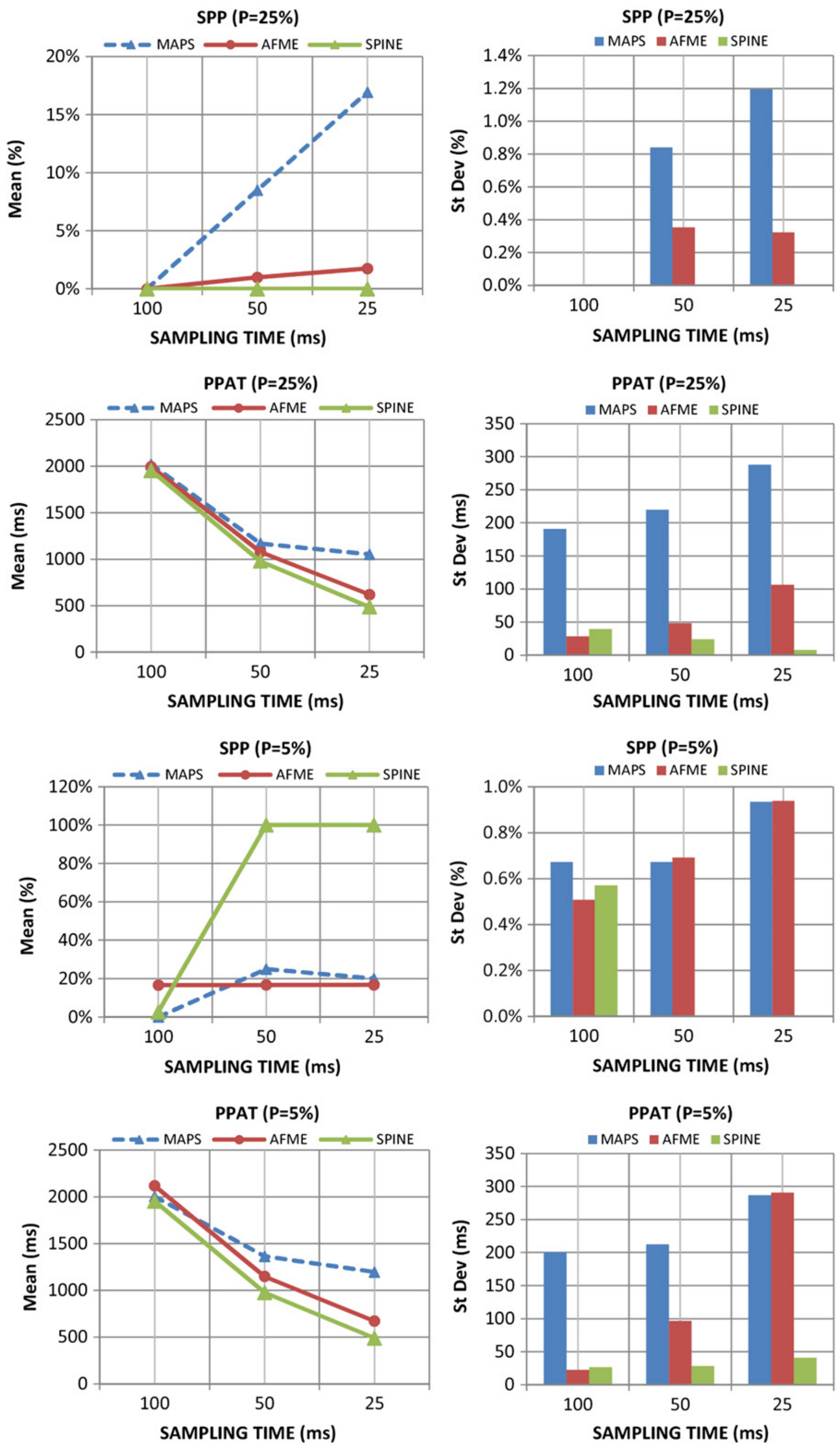


Fig. 15. Comparison of the synchronization between MAPS, AFME and SPINE sensor agents: PPAT and SPP for $P=25\%$ and $P=5\%$ and $W=40$, $S=20$.

interaction among AFME agents is also based on messages. In Figs. 18 and 19, the MAPS-based architecture and the AFME-based architecture of the sensor agents of the activity monitoring system are respectively reported. The AFME-based architecture is more complex than the MAPS-based architecture. In particular, in the former, according to the AFME model, specific components for Perceptors, Actuators, SharedDataModules and TERImplication formulas are to be defined; conversely, in the latter according to the MAPS architecture only the FSM plane is to be programmed.

From a usability point of view we can state that if agents to be modeled should not have a dynamic goal-oriented behavior which usually necessitates a complex architecture more suitable to capture complex requirements, the MAPS approach is more effective as the programming of the agent architecture is more rapid and straightforward. Indeed, it is worth pointing out that the programming of complex proactive agents in wireless sensor nodes is a very difficult task due to the limited computational resources of such nodes. In fact, AFME was originally conceived

Table 2
RAM usage and code dimension of sensor-node-side applications based on MAPS, AFME and SPINE.

	SPINE	*MAPS*	*AFME*
RAM used (KB)/available (KB) memory usage	4.8/10 48%	85.8/512 16.76%	96.5/512 18.85%
Code used (KB)/available (KB) memory usage	30/48 62.5%	76.8/4096 1.87%	81.8/4096 2%

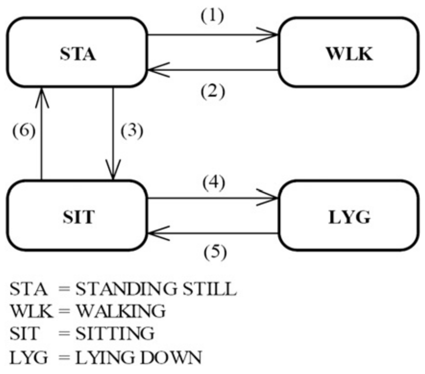


Fig. 16. State machine of the pre-defined sequence of postures/movements.

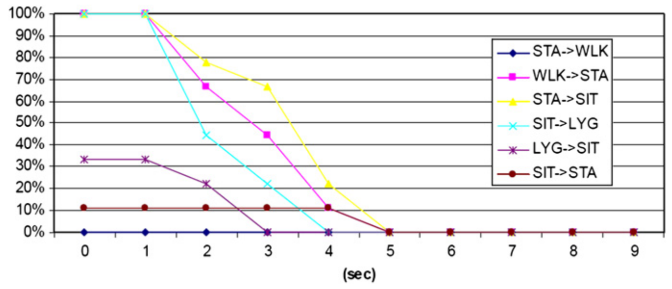


Fig. 17. Percentage of mismatches vs. transitory time computed with $ST=100$ ms, $W=20$, $P=25\%$.

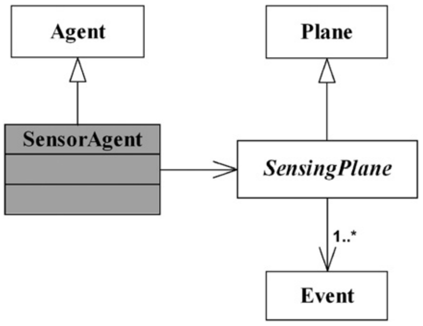


Fig. 18. MAPS-based architecture of the sensor-side agents of the activity monitoring system.

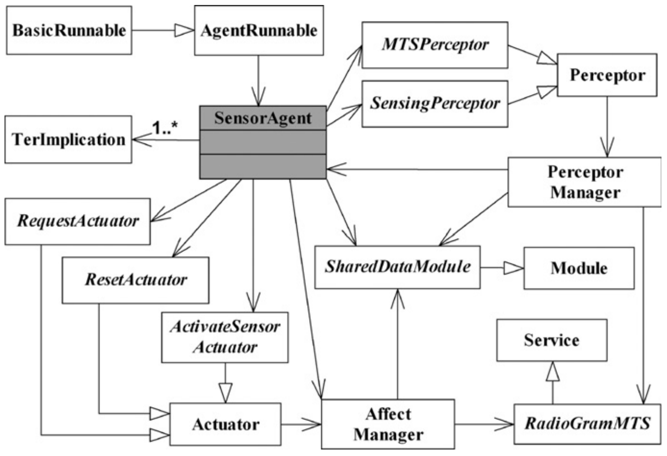


Fig. 19. AFME-based architecture of the sensor-side agents of the activity monitoring system.

for PDAs/smartphones that are devices much more resource-capable than wireless sensor nodes. In the context of WBSN applications which specifically requires intensive in-node signal processing, sensor-side computing components are more reactive than proactive. Moreover, as already stated in Section 3.3, the FSM model is one of the most used programming model in embedded computing so making MAPS appealing for programmers in the embedded computing research and development area. Finally, as the proposed activity monitoring system was also completely developed with SPINE and, previously, without SPINE by means of the nesC programming language (Gay et al., 2003), some insights deriving from the exploitation of different programming approaches (two agent-oriented approaches based on MAPS and AFME, a WBSN domain-specific approach based on SPINE and a lower-level programming approach based on nesC) for WBSN application development are discussed in the following. The effectiveness of the agent-oriented approach is evident if compared with the nesC approach as it provides higher-level programming methods to abstract away specific low-level operations, such as sensor driver access and low-level radio protocols, and ease the modeling of cooperative behavior through agent interactions. Finally, as SPINE is a WBSN domain-specific framework, it provides programming abstractions specific to the development of WBSN applications so being even more effective that MAPS and AFME in defining operations such as buffered sampling processes and data feature extractions. However, proactive behaviors and peer-to-peer sensor node interactions are not provided by SPINE whereas both MAPS and AFME agents support them in a straightforward way.

6. Conclusions

Programming WBSN applications is a complex task which requires suitable programming paradigms and frameworks coping with the WBSN specific characteristics. Several kinds of frameworks and approaches have been to date proposed. Among them, domain specific frameworks have the potential to provide both rapid development of WBSN applications and also good performances. In this paper we have proposed an agent-oriented approach, which relies on the basic features characterizing the domain specific frameworks and on the agent-oriented MAPS framework, aiming to offer more programming effectiveness while providing the required efficiency. In fact, MAPS has been purposely defined for resource-constrained environments and is based on (i) lightweight agents so avoiding conventional heavy-weight agent architectures and (ii) run-time architecture formed of components efficiently handling the low-level sensor node functions and providing higher-level services to agents. In particular, by using MAPS, a WBSN application can be structured as a set of agents distributed on sensor nodes supported by a component-based agent execution engine which provides basic services such as message transmission, agent creation, timer handling, easy access to the sensor node resources, and agent migration (if needed). The development and testing of a full-fledged real-time human activity monitoring system based on wireless body sensor networks has been described. It is emblematic of the effectiveness and suitability of MAPS to deal with the programming of WBSN applications. The carried out performance evaluation of the developed prototype shows fine synchronization of the sensor nodes, continuous real-time monitoring, and good recognition accuracy, once parameters are carefully set. However, the MAPS-based development of new applications having stringent requirements (sensing rate, computing speed, message transmission latency) must be carefully analyzed case by case as WSNs are application-specific systems.

Finally, the comparison of MAPS with the AFME framework based on Sun SPOTs and the WBSN-specific framework SPINE based on TinyOS in the development of the monitoring system, has produced important considerations about the provided system efficiency and programming effectiveness. From the system performance perspective, MAPS shows performances similar to those obtainable with AFME and SPINE that are suitable for fulfilling the real-time requirements of the monitoring system. From the programming effectiveness perspective, MAPS is more effective than AFME for WBSN applications as it is based on an FSM-based agent model that is more suitable than the AFME agent model for the development of lightweight WBSN-based components that are mostly reactive components. Moreover, with respect to SPINE, MAPS (and also AFME) is able to support peer-to-peer interactions among WBSN sensor nodes and proactive components.

On-going research efforts are devoted to: (i) porting MAPS onto the Sentilla JCreate pervasive computers which are compliant to J2ME CLDC 1.1 but based on TelosB-like sensors that are less powerful than Sun SPOTs; in particular, the on-going porting has arisen the need to define a TinyMAPS, a compressed version of MAPS, to drastically reduce the memory footprint; (ii) developing a full-fledged agent-based version of SPINE (named ASpine) through MAPS and the JADE framework to enable agent-oriented development of pervasive applications for assisted livings (such as emergency medical care) based on heterogeneous computing platforms: PC/workstations (JADE), PDA/smartphones (JADE Leap), and sensor nodes (MAPS); and (iii) defining an agent-oriented methodology for heterogeneous W(B)SN applications which uses MAPS as main target agent platform for wireless sensors and JADE for sensor coordinators.

Acknowledgments

Authors wish to thank Roberta Giannantonio, Marco Sgroi and Antonio Guerrieri for their precious contributions in terms of ideas and discussions, and Alessio Carbone for his on-going implementation efforts on ASpine. This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with Contract number FP7-2007-2-224053.

References

Agent Factory, 2011. Documentation and software at: ⟨http://www.agentfactory.com⟩.

Agent Factory Micro Edition (AFME), 2011. Documentation and software at: ⟨http://sourceforge.net/projects/agentfactory/files/⟩.

Aiello, F., Fortino, G., Guerrieri, A., 2008. Using mobile agents as enabling technology for wireless sensor networks. In: SENSORCOMM '08: Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications. IEEE Computer Society, Washington, DC, USA, pp. 549–554.

Aiello, F., Fortino, G., Gravina, R., Guerrieri, A., 2011. A Java-based agent platform for programming wireless sensor networks. The Computer Journal 54 (3), 439–454.

Bao, L., Intille, S.S., 2004. Activity recognition from user-annotated acceleration data. In: Pervasive Computing. Lecture Notes in Computer Science, pp. 1–17.

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

Bölöni, L., Marinescu, D.C., 2000. A multi-plane state machine agent model. In: AGENTS'00: Proceedings of the Fourth International Conference on Autonomous Agents. ACM, New York, NY, USA, pp. 80–81.

Cover, T., Hart, P., 1967. Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13 (1), 21–27.

Fok, C.-L., Roman, G.-C., Lu, C., 2005. Rapid development and flexible deployment of adaptive wireless sensor network applications. In: ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems. IEEE Computer Society, Washington, DC, USA, pp. 653–662.

Fortino, G., Guerrieri, A., Bellifemine, F., Giannantonio, R., 2009. Platform-independent development of collaborative wireless body sensor network applications. In: SMC'09: Proceedings of the 2009 IEEE International Conference on Systems Man and Cybernetics. IEEE Press, Piscataway, NJ, USA, pp. 3144–3150.

Fortino, G., Garro, A., Mascillaro, S., Russo, W., 2010. Using event-driven lightweight DSC-based agents for MAS modelling. International Journal of Agent-Oriented Software Engineering (IJAOSE) 4 (2), 113–140.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D., 2003. The NESC language: a holistic approach to networked embedded systems. In: PLDI '03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. ACM, New York, NY, USA, pp. 1–11.

Gravina, R., Alessandro, A., Salmeri, A., Buondonno, L., Raveendranathan, N., Loseu, V., Giannantonio, R., Seto, E., Fortino, G., 2010. Enabling multiple BSN applications using the SPINE framework. In: International Conference on Body Sensor Networks (BSN) 2010, pp. 228–233.

Iyengar, S., Bonda, F.T., Gravina, R., Guerrieri, A., Fortino, G., Sangiovanni-Vincentelli, A., 2008. A framework for creating healthcare monitoring applications using wireless body sensor networks. In: BodyNets '08: Proceedings of the ICST Third International Conference on Body Area Networks. ICST (Institute for Computer Sciences Social Informatics and Telecommunications Engineering), Brussels, Belgium, pp. 1–2.

Jade, 2011. Documentation and software at: ⟨http://jade.tilab.com/⟩.

Kent Dybvig, R., 1987. The SCHEME Programming Language. Prentice Hall Inc., Upper Saddle River, NJ, USA.

Kwon, Y.M., Sundresh, S., Mechitov, K., Agha, G., 2006. Actornet: an actor platform for wireless sensor networks. In: AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. ACM, New York, NY, USA, pp. 1297–1300.

Lombriser, C., Marin-Perianu, R., Roggen, D., Havinga, P., Tröster, G., 2009. Modeling service-oriented context processing in dynamic body area networks. IEEE Journal on Selected Areas in Communications 27 (1), 49–57.

Luck, M., McBurney, P., Preist, C., 2004. A manifesto for agent technology: towards next generation computing. Autonomous Agents and Multi-Agent Systems 9 (3), 203–252.

Malan, D., Fulford-Jones, T., Welsh, M., Moulton, S., 2004. Codeblue: an ad hoc sensor network infrastructure for emergency medical care. In: International Workshop on Wearable and Implantable Body Sensor Networks.

Maurer, U., Smailagic, A., Siewiorek, D.P., Deisher, M., 2006. Activity recognition and monitoring using multiple sensors on different body positions. In: BSN '06: Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks. IEEE Computer Society, Washington, DC, USA, pp. 113–116.

Mobile Agent Platform for Sun SPOT (MAPS), 2011. Documentation and software release 1.1: ⟨http://mapsproject.sourceforge.net/⟩.

Muldoon, C., O'hare, G.M.P., Collier, R., O'grady, M.J., 2006. Agent factory micro edition: a framework for ambient applications. In: Proceedings of the Intelligent Agents in Computing Systems—the Workshop in Frames of the ICCS 2006 International Conference on Computational Science, pp. 28–31.

Muldoon, C., O'Hare, G.M.P., O'Grady, M.J., Tynan, R., 2008. Agent migration and communication in WSNs. In: PDCAT '08: Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing Applications and Technologies. IEEE Computer Society, Washington, DC, USA, pp. 425–430.

Najafi, B., Aminian, K., Paraschiv-Ionescu, A., Loew, F., Bula, C., Robert, Ph., 2003. Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. IEEE Transactions on Biomedical Engineering 50 (6), 711–723.

Rao, A.S., Georgeff, M.P., 1995. BDI-agents: from theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, San Francisco.

Signal Processing In-node Environment (SPINE), 2011. Documentation and software: ⟨http://spine.tilab.com⟩.

Sohraby, K., Minoli, D., Znati, T., 2007. Wireless Sensor Networks: Technology, Protocols, and Applications. Wiley-Interscience.

Sun Small Programmable Object Technology (Sun SPOT), 2011. Documentation and software: ⟨http://www.sunspotworld.com⟩.

TinyOS, 2011. Documentation and software: ⟨http://www.tinyos.net⟩.

Vinyals, M., Rodríguez-Aguilar, J.A., Cerquides, J., 2011. A survey on sensor networks from a multiagent perspective. The Computer Journal 54 (3), 455–470.

Yang, G.-Z., 2006. Body Sensor Networks. Springer.

Yoneki, E., Bacon, J., 2005. A survey of wireless sensor network technologies: research trends and middleware's role. Technical Report UCAM-CL-TR646, University of Cambridge, October.

Z-stack (Zigbee Protocol Stack), 2011. Texas instruments, documentation and software: ⟨http://focus.ti.com/docs/toolsw/folders/print/z-stack.html⟩.