ISBN 978-83-60810-22-4

Monitoring Building Indoors through Clustered Embedded Agents

Giancarlo Fortino, Antonio Guerrieri
DEIS – University of Calabria,
Via P. Bucci, cubo 41c,
Rende (CS), 87036, Italy
Email: g.fortino@unical.it, aguerrieri@deis.unical.it

Abstract—**Future buildings will be smart to support personalized people comfort and building energy efficiency as well as safety, emergency, and context-aware information exchange scenarios. In this work we propose a decentralized and embedded architecture based on agents and wireless sensor and actuator networks (WSANs) 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 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. 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.**

I. Introduction

NOWADAYS, due to advances in communication and computing technologies, the need to have high comfort levels together with an optimization of the energy consumption is becoming important for inhabitants of buildings. Moreover, buildings should also support their inhabitants with automatic emergency and safety procedures as well as context aware information services. To meet all these requirements, future buildings have to incorporate diversified forms of intelligence [1].

We believe that agent-based computing [2] can be exploited to implement the concept of intelligent buildings due to the agent features of autonomy, 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 [3][4][5]. 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) [6] can be adopted. WSANs represent 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 effort. This should enable monitoring of structure conditions, and space and energy (electricity, gas, water) usage while facilitating the design of techniques for intelligent device actuation.

In this paper we propose a decentralized and embedded management architecture for intelligent buildings that is based on WSANs and overcomes the limitations of the aforementioned solutions [3][4][5]. 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.

The rest of this paper is organized as follows. Section II describes approaches related to our work. In Section III the proposed agent-based architecture for building management is defined. Section IV presents the MAPS-based implementation of the architecture, specifically the sensor/actuator agents. Section V shows the system GUI and a system deployment for monitoring the workstation usage in computer laboratories. Finally, conclusions are drawn and directions of future work elucidated.

II. Related Work

In [3] the authors present the MASBO (Multi-Agent System for Building cOntrol) architecture that aims to provide a set of software agents to support both on-line and off-line applications for intelligent work environments. MASBO is used to develop a multi-agent system (MAS) able to tradeoff energy saving and inhabitants' preferences where preferences can be learnt and predicted through an unsupervised online real-time learning algorithm (analyzing inhabitants' behavior).

MASBO agents reside on a server and constantly monitor data from sensors and eventually actuate some commands. MASBO works as an enhancement to an existing building automation system by adding learning, reasoning and autonomous capabilities. The responsibility of controlling sensors and actuators, and keeping a requested environmental value constant is not addressed by MASBO.

In [4] the authors propose a working solution to the problem of thermal resource distribution in a building using a market-based MAS. Computational agents representing individual temperature controllers bid to buy or sell cool or warm air. The agents, running in a monolithic process on a workstation, are able to distribute the thermal resources so that all the building offices have an equitable temperature distribution. Temperature sensors and air flow actuators are all accessible directly through distributed hardware modules via a network connection.

In [5] the authors describe a MAS that monitors and controls an office building in order to provide added values like energy saving together with the delivery of energy. The developed system is distributed in the sense that some agents are located on PDAs and others run on the Bluetooth access points (workstations) that communicate with the PDAs. The system makes use of the existing power lines for communication between the agents and the sensing and actuation system controlling lights, heating, ventilation, etc.

Differently from the described approaches, our agent-based architecture embeds agents into the wireless sensor and actuator network used as infrastructure for building monitoring and control. This important feature would provide decentralized intelligence and improve system efficiency.

III. Agent-Based Architecture

The agent-based architecture (see Fig. 1) 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 Fig. 2 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) [7].

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 behaviors for implementing specific building monitoring and control strategies.

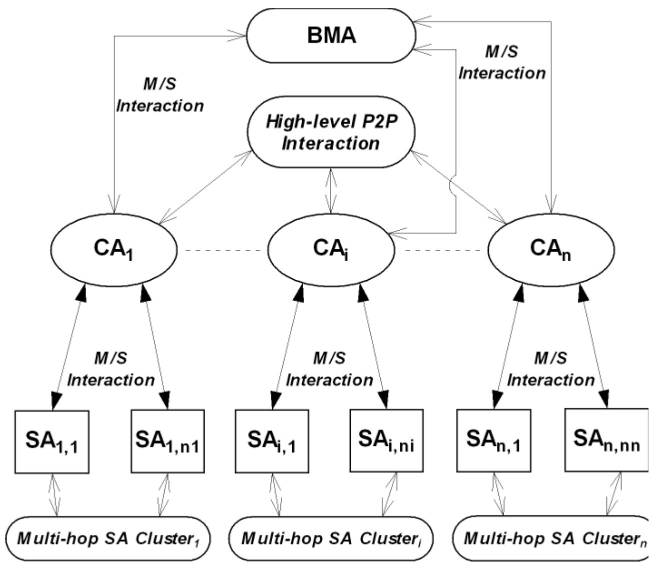


Fig. 1 Agent-based architecture for decentralized and embedded management of buildings based on wireless sensor and actuator networks.

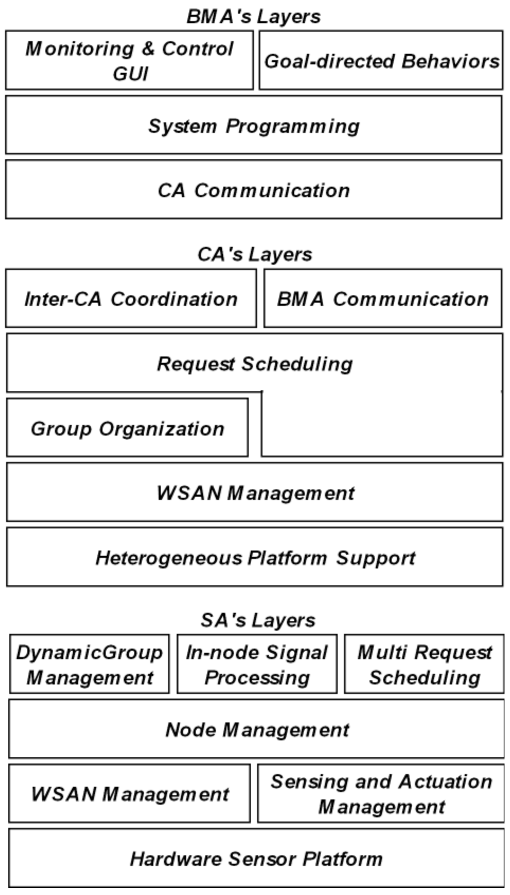


Fig. 2 The layered organization of BMA, CA and SA.

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 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 [8]. 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.

IV. MAPS-based Implementation

The agent-based building management architecture is currently implemented through MAPS [9], our agent-based framework for developing WSN applications on the Sun SPOT sensor platform. In this section we first provide a brief overview of MAPS (details can be found in [9, 10]) and, then, present the MAPS-based implementation of the proposed building management architecture at sensor-node side, specifically behavior and event-based interactions of the SA.

A. MAPS: a brief overview

MAPS [9, 10] is an innovative Java-based framework specifically developed on Sun SPOT technology for enabling agent-oriented programming of WSN applications. It has been defined according to the following requirements:

— *Component-based lightweight agent server architecture* to avoid heavy concurrency and agents cooperation models.

— *Lightweight agent architecture* to efficiently execute and migrate agents.

— *Minimal core services* involving agent migration, agent naming, agent communication, timing and sensor node resources access (sensors, actuators, flash memory, and radio).

— *Plug-in-based architecture* extensions through which any other service can be defined in terms of one or more dynamically installable components implemented as single or cooperating (mobile) agents.

— *Use of Java language* for defining the mobile agent behavior.

The architecture of MAPS (see Fig. 3) is based on several components interacting through events and offering a set of services to mobile agents, including message transmission, agent creation, agent cloning, agent migration, timer handling, and an easy access to the sensor node resources. In particular, the main components are the following:

— *Mobile Agent (MA)*. MAs are the basic high-level component defined by user for constituting the agent-based applications.

— *Mobile Agent Execution Engine (MAEE)*. It manages the execution of MAs by means of an event-based scheduler enabling lightweight concurrency. MAEE also interacts with the other services-provider components to fulfill service requests (message transmission, sensor reading, timer setting, etc) issued by MAs.

— *Mobile Agent Migration Manager (MAMM)*. This component supports agents migration through the Isolate (de)hibernation feature provided by the Sun SPOT environment. The MAs hibernation and serialization involve data and execution state whereas the code must 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).

— *Mobile Agent Communication Channel (MACC)*. It enables inter-agent communications based on asynchronous messages (unicast or broadcast) supported by the Radiogram protocol.

— *Mobile Agent Naming (MAN)*. MAN provides agent naming based on proxies for supporting MAMM and MACC in their operations. It also manages the (dynamic) list of the neighbor sensor nodes which is updated through a beaconing mechanism based on broadcast messages.

— *Timer Manager (TM)*. It manages the timer service for supporting timing of MA operations.

— *Resource Manager (RM)*. RM allows access to the resources of the Sun SPOT node: sensors (3-axial accelerometer, temperature, light), switches, leds, battery, and flash memory.

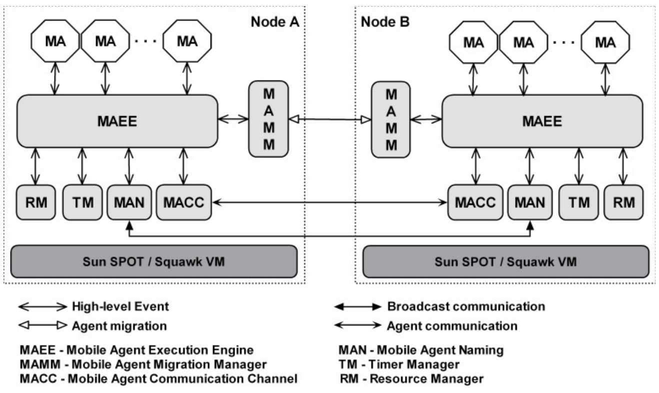


Fig. 3 The architecture of MAPS.

The dynamic behavior of a mobile agent (MA) is modeled through a multi-plane state machine (MPSM). Each plane may represent the behavior of the MA in a specific role so enabling role-based programming. In particular, a plane is composed of local variables, local functions, and an automaton whose transitions are labeled by Event-Condition-Action (ECA) rules E[C]/A, where E is the event name, [C] is a boolean expression evaluated on global and local variables, and A is the atomic action. Thus, agents interact through events, which are asynchronously delivered and managed by the MAEE component.

It is worth noting that the MPSM-based agent behavior programming allows exploiting the benefits deriving from three main paradigms for WSN programming: event-driven programming, state-based programming and mobile agent-based programming.

MAPS is also interoperable with the JADE framework [11]. Specifically, a JADE-MAPS gateway [12] has been developed for allowing JADE agents to interact with MAPS agents and vice versa. While both MAPS and JADE are Java-based, they use a different communication method. JADE sends messages according to the FIPA standards (using the ACL specifications), while MAPS creates its own messages based on events. Therefore, the JADE-MAPS Gateway facilitates message exchange between MAPS and JADE agents. This inter-platform communication infrastructure allows rapid prototyping of WSN-based distributed applications/systems that use JADE at the basestation/coordinator/host sides and MAPS at the sensor node side.

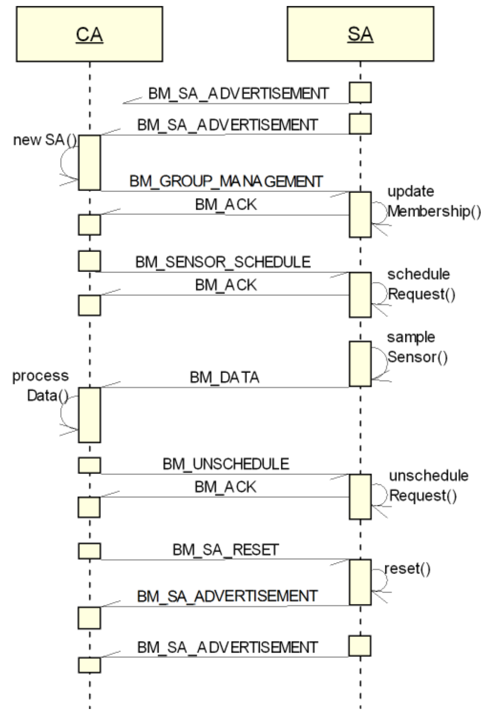


Fig. 4 Sequence Diagram of the interactions between CA and SA

B. MAPS-based sensor agents

The MAPS-based SA (hereafter simply named SA) interacts with its cluster CA through events as sketched in the sequence diagram of Fig. 4. 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 section III). 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.

Tables I and II reports 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 I. Moreover, each event type has its own *additional parameters*, which are described in Table II. It is worth noting that the ADDRESSEE value can be set through the fol-

TABLE I.
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>
		endIf

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>

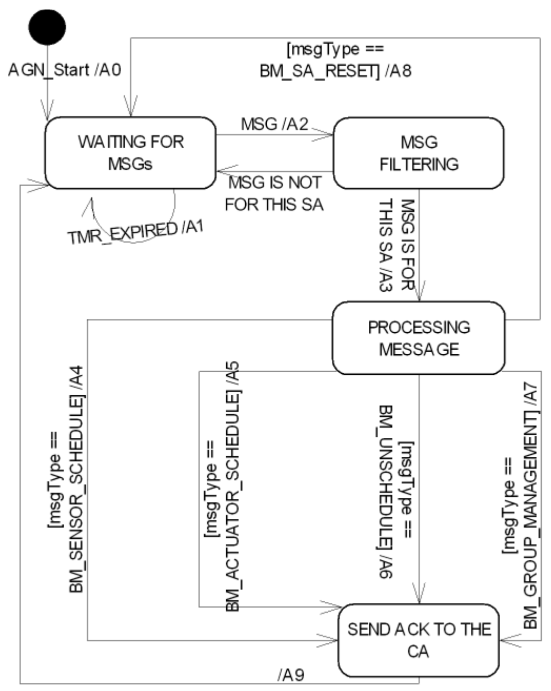
BM_ACK	ID_SA;
	ID_CA;
	Event.MSG_TO_
	BASESTATION;
	Event.NOW
		<MSG_TYPE, BM_ACK>
		<MSG_TYPE_TO_ACK, VALUE>
		<ACK_PARAM, VALUE>

TABLE II.
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, SOUND, ELECTRICITY, MAGNETIC_X, MAGNETIC_Y, , TEMPERATURE, 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	*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

lowing regular expression: *SA+ | ([NOT] G [TSO [NOT] G]*)*, 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.

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 an unschedule event. Agent planes receive events from the MAPS dispatcher component that is programmed to deliver the events fetched from the agent


AGN_Start /A0
[msgType == BM_SA_RESET] /A8
WAITING FOR MSGs
MSG /A2
MSG FILTERING
MSG IS NOT FOR THIS SA
TMR_EXPIRED /A1
MSG IS FOR THIS SA /A3
PROCESSING MESSAGE
[msgType == BM_SENSOR_SCHEDULE] /A4
[msgType == BM_ACTUATOR_SCHEDULE] /A5
[msgType == BM_UNSCHEDULE] /A6
[msgType == BM_GROUP_MANAGEMENT] /A7
SEND ACK TO THE CA
/A9

```
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
```

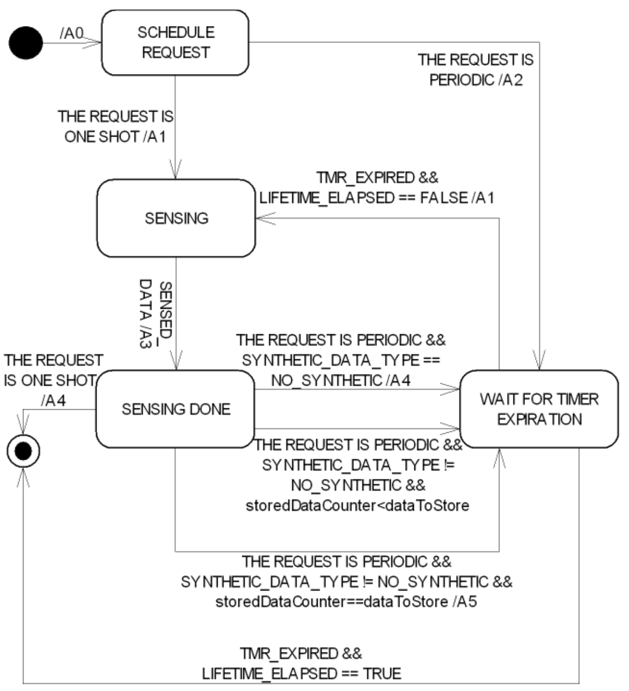
Fig. 5 The SA's Manager plane.

queue to the plane in charge to process them. The dispatcher rules are reported in Table III.

TABLE III.
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

The Manager plane is reported in Fig. 5. 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. 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


/A0
SCHEDULE REQUEST
THE REQUEST IS PERIODIC /A2
THE REQUEST IS ONE SHOT /A1
TMR_EXPIRED && LIFETIME_ELAPSED == FALSE /A1
SENSING
SENSED DATA /A3
THE REQUEST IS PERIODIC && SYNTHETIC_DATA_TYPE == NO_SYNTHETIC /A4
THE REQUEST IS ONE SHOT /A4
SENSING DONE
WAIT FOR TIMER EXPIRATION
THE REQUEST IS PERIODIC && SYNTHETIC_DATA_TYPE != NO_SYNTHETIC && storedDataCounter<dataToStore
THE REQUEST IS PERIODIC && SYNTHETIC_DATA_TYPE != NO_SYNTHETIC && storedDataCounter==dataToStore /A5
TMR_EXPIRED && LIFETIME_ELAPSED == TRUE

```
A0: Process Request
A1: Create and Submit a Sensing Event on the Sensor Requested
A2: Inizialize and Submit a TMR_EXPIRED Event with the Params PERIOD and LIFETIME
    Set dataToStore
A3: Store sensed data and increase the storedDataCounter
A4:  if DATA_TYPE.VALUE == "threshold notification"
       Send sensed data to CA if the threshold is verified and reset the storedDataCounter
     else Send sensed data to CA and reset the storedDataCounter
A5: Calculate the SYNTHETIC_DATA_TYPE requested
     if DATA_TYPE.VALUE == "threshold notification"
       Send synthetic data to CA if the threshold is verified and reset the storedDataCounter
     else Send synthetic data to CA and reset the storedDataCounter
```

Fig. 6 The SA's Sensing Request plane.

of each action of the Manager plane is provided using a self-explanatory pseudocode (see Fig. 5).

In Fig. 6 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 a self-explanatory pseudocode (see Fig. 6).

V. A SYSTEM DEPLOYMENT: MONITORING WORKSTATION USAGE IN COMPUTER LABORATORIES

To show the functionality and effectiveness of the proposed architecture for the management of building indoors, we present an example of system deployment for the monitoring of workstation usage in a computer laboratory or in offices. The wireless sensor network consists of heterogeneous sensor nodes based on Sun SPOTs that are used to collect information about the ambient light (through the standard Sun SPOT light sensor), the user presence (through a

Wieye IR sensorboard [13]) and the electricity consumed by the workstation (through a customization of the ACme electricity sensorboard [14]).

In Fig. 7, the main window of the Building Management GUI is shown. It is organized in five main sections supporting all the functionalities provided by the system:

— *Nodes* and *Groups Management* sections allows to visualize the nodes of the WSAN and configure groups, respectively. By right clicking on the sensors/groups the user can configure sensor/actuator requests to schedule on the nodes;

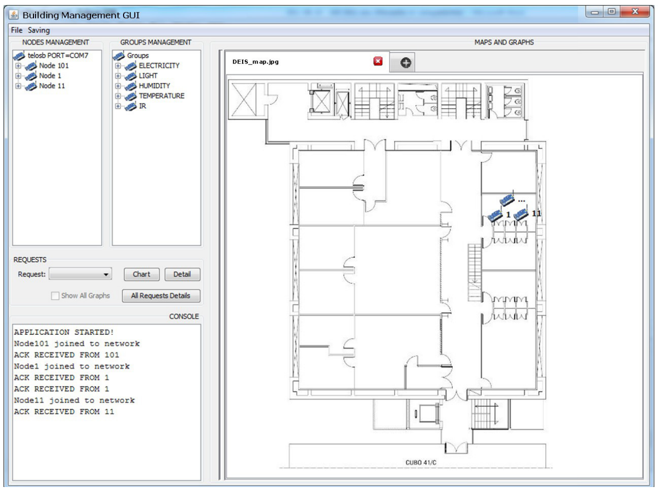
— *Request* section allows to list details of scheduled requests, display data charts related to the scheduled requests, unschedule and re-schedule requests;

— *Maps and Graphs* section allows visualizing WSAN deployment maps and displaying charts of the data coming from the sensors (examples of charts are shown in Fig. 9);

— *Console* section displays the real-time log of the activity of the system;

— *File and Saving* menu section enables to save data from the system in structured files and load stored files to display them in the GUI.

In Fig. 8, the graphical window for sensor/actuator request scheduling is shown. The window allows setting the parameter of a new request: name, destination (specific nodes or group composition), execution period, lifetime, one shot request or unlimited lifetime flags, action type and related device, possible actuator parameters, requested sensed data possibly filtered by thresholds and/or synthetic data is requested and its type (average/max/min) and eventual threshold parameters can be set.



```
APPLICATION STARTED!
Node101 joined to network
ACK RECEIVED FROM 101
Node1 joined to network
ACK RECEIVED FROM 1
ACK RECEIVED FROM 1
Node11 joined to network
ACK RECEIVED FROM 11
```

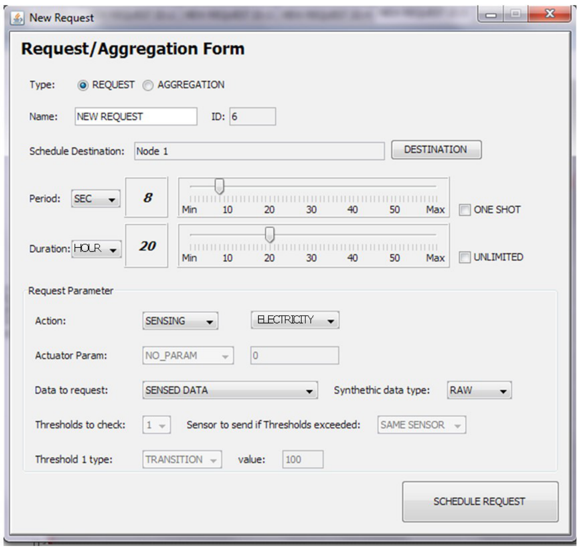
Fig. 7 The Building Management GUI

In the experimental system deployment the following requests were set:

— the raw electricity data (in watt) are gathered every second;

— the average of the ambient light value (in lux) is collected every 10 seconds;

— the max IR sensor value is sensed every minute.



Type:	[✓ REQUEST] AGGREGATION
Name:	NEW REQUEST
ID:	6
Schedule Destination:	Node 1
Period:	SEC 8
ONE SHOT	☐
Duration:	HOUR 20
UNLIMITED	☐
Request Parameter	
Action:	SENSING ELECTRICITY
Actuator Param:	NO_PARAM 0
Data to request:	SENSED DATA
Synthethic data type:	RAW
Thresholds to check:	1
Sensor to send if Thresholds exceeded:	SAME SENSOR
Threshold 1 type:	TRANSITION
value:	100

Fig. 8 The graphical window for sensor/actuator request scheduling.

The aim of the experiment was the monitoring of a workstation in a computer laboratory of the Dept. of Electronics, Informatics and Systems to understand its user's behavior. A snapshot of a significant monitoring activity of the duration of 45 min is shown in Fig. 9. In particular, in Fig. 9 two important time instants (*t1* and *t2*) are marked. Before *t1* the user was working at his workstation by using a word processor application and the ambient light is low as artificial light is off and window curtains were partially closed. Between *t1* and *t2* the user was out of the office, his workstation automatically switched the monitor off after a period of inactivity and the light was decreasing as late evening was approaching. At *t2*, the user came back, started a video streaming application, turned the ceiling lamp on, and after five minutes came out again.

VI. Conclusions and Future Work

In this paper we have proposed an agent-based architecture for flexible, efficient and embedded sensing and actuation in buildings. Specifically, the distributed software architecture is embedded into both WSANs and more capable computing devices (e.g. PCs, smartphones, plug computers). The proposed architecture can be seen as basic middleware for developing intelligent building management systems to achieve the Smart Building concept. Currently the proposed architecture is exploited to monitor the space occupation and energy expenditure in computer laboratories for students to analyze energy consumption patterns with respect to users' behavior so as to semi-automatically implement behavior policies. In the current implementation, BMA and CA are merged into a component-based application implemented through OSGi [15]. Moreover, only one cluster can be deployed. On-going work is aimed at completing the JADE-based implementation of the multi-cluster architecture founded on the BMA and on multiple coordinated CAs. Future work will be devoted to the design of a higher-level agent-based architecture for Smart Buildings atop the proposed ar-

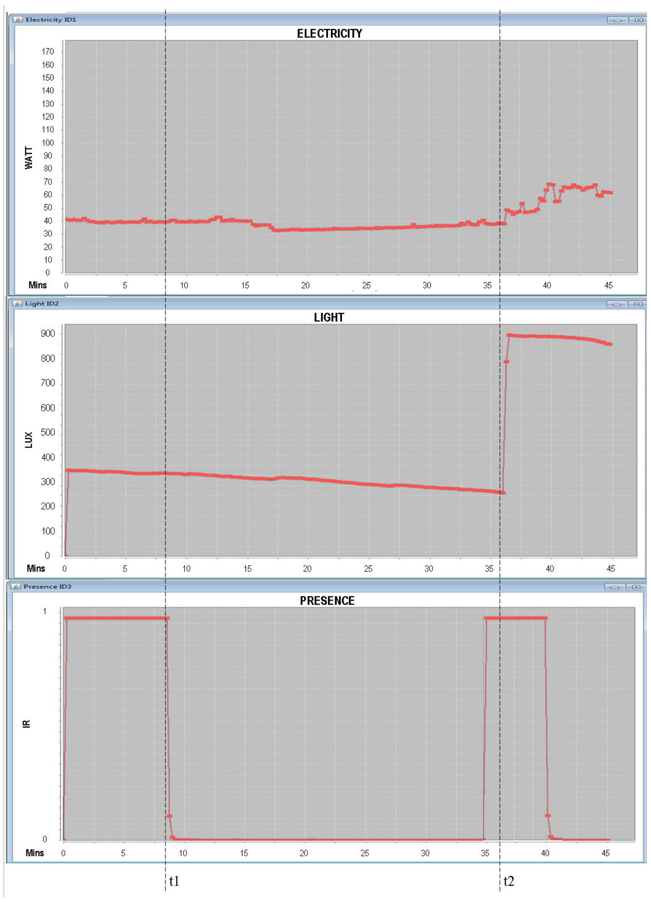


Fig. 9 Real-time data of the workstation usage (workstation consumed power, ambient light and user presence)

chitecture to trade off inhabitants' personal comfort and building energy expenditure.

Acknowledgment

This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053, and by TETRis – TETRA Innovative Open Source Services, funded by the Italian Government (PON 01-00451).

References

[1] Davidsson, P., Boman, M.: A multi-agent system for controlling intelligent buildings. In the Fourth International Conference on MultiAgent Systems, pp. 377-378, Boston (2000)

[2] Luck, M., McBurney, P., Preist, C.: A manifesto for agent technology: towards next generation computing. Journal of Autonomous Agents and Multi-Agent Systems, vol. 9, n. 3, pp. 203-252 (2004)

[3] Qiao, B., Liu, K., Guy, C.: A Multi-Agent System for Building Control. In the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT '06), pp.653-659, Hong Kong (2006).

[4] Huberman, B. A., Clearwater, S. H.: A Multi-Agent System for Controlling Building Environments. In the International Conference on Multiagent Systems (ICMAS-95), pp. 171-176, San Francisco (1995)

[5] Davidsson, P., Boman, M.: Distributed monitoring and control of office buildings by embedded agents. In Information Sciences—Informatics and Computer Science: An International Journal - Special issue: Intelligent embedded agents, vol. 171, issue 4, pp. 293-307 (2005)

[6] Stankovic J.: When sensor and actuator cover the world. ETRI Journal; vol. 30, n. 5, pp. 627–633 (2008)

[7] Guerrieri, A., Ruzzelli, A., Fortino, G., O'Hare, G.: A WSN-based Building Management Framework to Support Energy-Saving Applications in Buildings. In Advancements in Distributed Computing and Internet Technologies: Trends and Issues, Al-Sakib Khan Pathan, Mukaddim Pathan, Hae Young Lee, eds, chapter 12, pp. 161-174, IGI Global (2011)

[8] Bellifemine, F., Fortino, G., Giannantonio, R., Gravina, R., Guerrieri, A., Sgroi, M.: SPINE: A domain-specific framework for rapid prototyping of WBSN applications. Software Practice and Experience, Wiley, vol. 41, issue 3, pp. 237-265 (2011)

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

[10] Mobile Agent Platform for Sun SPOT (MAPS), documentation and software at: http://maps.deis.unical.it/.

[11] Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Softw., Pract. Exper. vol. 31, issue 2: pp. 103-128 (2001)

[12] Domanski, J.J., Dziadkiewicz, R., Ganzha, M., Gab, A., Mesjasz M.M.: Implementing GliderAgent – an agent-based decision support system for glider pilots. In NATO ASI Book, IOS press, 2011, to appear.

[13] http://www.easysen.com/WiEye.htm

[14] Jiang, X., Dawson-Haggerty, S., Dutta, P., and Culler, D. Design and Implementation of a High-Fidelity AC Metering Network. In Proc. of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN09) Track on Sensor Platforms, Tools, and Design Methods (SPOTS 09). 2009.

[15] Open System Gateway Initiative (OSGi), documents and software at: http://www.osgi.org