An Agent-based Open Source Platform for Building Energy Management

W. Khamphanchai, M. Pipattanasomporn, M. Kuzlu, and S. Rahman
Bradley Dept. of Electrical and Computer Engineering
Advanced Research Institute – Virginia Tech, Arlington, VA, USA 22203

Abstract—**The objective of this paper is to propose a building energy management (BEM) platform thatallows sensing and control of equipment in small- and medium-sized buildings. Theproposed platform aims to improve energy efficiency, reduce energy consumption, and foster demand response (DR) implementation by controlling three major loads in buildings, including HVAC, lighting and plug loads. In addition, it aims to offer scalability, robustness, plug and play, open protocol, interoperability, cost-effectiveness, and local and remote monitoring. The software architecture,including user interface, application and data management, operating system and framework, and connectivity are discussed in this paper with the special focus on the Multi-agent system (MAS) development which is the core of the platform. A laboratory test bed is employedto demonstrate the functionalities of the proposed software platform.**

Index Terms—**Building energy management, multi-agent systems, demand response**

I. Introduction

Multi-agent systems (MAS), featuring the implementation and utilization of multiple distributed intelligent agents, share many common characteristics such as being adaptive, self-aware and semi-autonomous or autonomous. The most outstanding advantages that MAS embraces are that they can respond to the external environment rapidly, and can provide timely solutions based on distributed control with or without human intervention. For this reason, MAS technology is recognized as a promising approach for the development of numerous real-world applications, ranging from e-commerce to power systems. One of interesting MAS applications is energy management. When designing an intelligent energy management system, one needs to take into consideration a few key features, including efficiency, scalability, robustness and flexibility, along with the ability to sense the environment and make decisions. These features make MAS technology a natural choice, as they are among the many benefits that MAS can offer.

This paper targets the application of MAS for developing an open-source Building Energy Management (BEM) solution to improve energy efficiency in small- and medium-sized commercial buildings. While in the United States, buildings consume over 40% of the total energy consumption [1][2], one study [3] shows that due to the lack of building monitoring and control, significant portion of the energy consumed in buildings is wasted. At present, most BEM solutions are proprietary, thereby cost-prohibitive and used mostly in large buildings. BEM are not popular in most small- and medium-sized buildings due to lack of awareness of benefits, lack of inexpensive packaged solutions, and sometimes due to an owner not being a tenant thus finding no incentive to invest in these systems [3]. However, small- and medium-sized buildingssignify a huge market for BEM deployment as they represent over 90% of all commercial buildings in the United States according to U.S. Energy Information Administration (EIA) [4].

These issues are the driving factors that inspire us to create an agent-based open source software platform for BEM system, to promote the rapid evolution and wide adoption of a BEM system. The remainder of this paper is organized as follows. Section II discusses the prior work. Section III presents the core concept of the proposed platform. The development of MAS in the platform is elaborated in Section IV. Section V demonstrates the lab setup and experiments to showcase the platform functionalities.

II. Prior Work

Agent-based technology, often associated with "intelligent" and "efficient", has shed light on traditional power system applications such as scalability and resilience issues inherent in the power grid [5][6][7], as well as novel applications such as home and building energy management [8]-[13]. Many of these projects are agent-based, but their demonstrations are limited to simulation or proof-of-concept implementations that would not operate well in the field [14].

Instead of simulations, our goal is to provide a tangible solution to BEM with a real and holistic software product that handles everything between users and physical hardware devices. At the initial stage, a number of open source agent development platforms were investigated. The target platform should be language-agnostic with features of security, mobility and scalability, and resource management.

One platform is JADE (Java Agent Development Framework), software framework fully implemented in Java [15]. The advantage is that developers can easily build a

This work was supported in part by the U.S. Department of Energy under Grant# DE-EE-0006352.0000.

W. Khamphanchai, M. Pipattanasomporn, M. Kuzlu and S. Rahman are with Virginia Tech – Advanced Research Institute, Arlington, VA 22203 USA (e-mails: kwarodom@vt.edu, mpipatta@vt.edu, mkuzlu@vt.edu,srahman@vt.edu).

FIPA-compliant multi-agent system with their set of Java classes. The disadvantage is limited support on resource management and security that are important requirements for our platform.

Another platform is Spade [16], which is a platform based on the XMPP/Jabber technology and written in the Python programming language. SPADE is the first to base its roots on the XMPP technology and is also FIPA-compliant platform. However, the support for security and resource management seems to be missing.

There is also AgentScape [17], a distributed agent middleware. Its design philosophy is "less is more", meaning AgentScape provides a minimal but sufficient support for agent applications. This will cost a lot of extra to develop the required platform features.

Finally, VOLTTRON™, a distributed agent platform developed by Pacific Northwest National Laboratory (PNNL) [14], [18], [19], is chosen to be an agent development platform for the proposed platform. VOLTTRON™ is designed to be able to run on small-form-factor computers, capable of interfacing with hardware devices, maintain security, manage platform resources, and service for applications. VOLTTRON™ platform enables the deployment of intelligent sensors and controllers in residential/commercial buildings and the smart grid. Distributed agents using peer-to-peer communications in VOLTTRON™ cooperate to bring computation closer to data to enable distributed control decisions and data analysis. Intelligent agents residing in VOLTTRON™ are designed to have most of these capabilities: reactive, pro-active, social, mobility, veracity, benevolence, rationality, and learning/adaptation.

Built on top of VOLTTRON™, the proposed platform is designed specifically for energy management in buildings. The platform greatly simplifies the efforts for seamless integration of hardware and software applications.

III. THE PROPOSED PLATFORM CONCEPT

The absense of low-cost and user-friendly BEM has prompted the development of an agent-based software platform for sensing and control of equipment in small- and medium- sized buildings.The platform aims to improve energy efficiency, optimize electricity usage to reduce energy consumption, and help implement demand response (DR) programs.

For small- and medium-sized buildings, HVAC heating and cooling consumption is the dominant end use, followed by lighting loads, and plug loads [2]. With these loads combined, they account for almost 75% of all consumption in buildings.

Apart from possessing the ability to monitor and control these three major load types, the platform presents a multitude of features listed as follows: 1) *Open source, open architecture*: built upon a robust open source platform, the platform is ready for manufacturers and engineers to seamlessly interface their devices and add functionalities. 2)*Plug & play*: the platform can automatically discover supported load controllers, then monitor and control them to perform the desired functions. 3) *Interoperability*: the platform can work with load control devices from different manufacturers that operate on different communication technologies (Wi-Fi, Zigbee, Ethernet, Serial), and data exchange protocols (HTTP/HTTPS, BACnet, Modbus, Zigbee-API, SEP). 4) *Cost effectiveness*: the platform is open source, and it can operate on a low-cost single-board computer (e.g., PandaBoard [20] or BeagleBone Black [21]). 5) *Mobility andScalability*: In a multi-floor and high occupancy building, multiple single-board computers hosting the platform can communicate among each other and a master controller to monitor and control a large number of load controllers. 6) *Local and remote monitoring/control*. This platform allows local and remote monitoring with role-based access control. 7) *Security*. Security features including agent authorization & authentication, encrypted multi-layer communication and agent validation.

Fig.1 shows the conceptual implementation of the proposed platform.

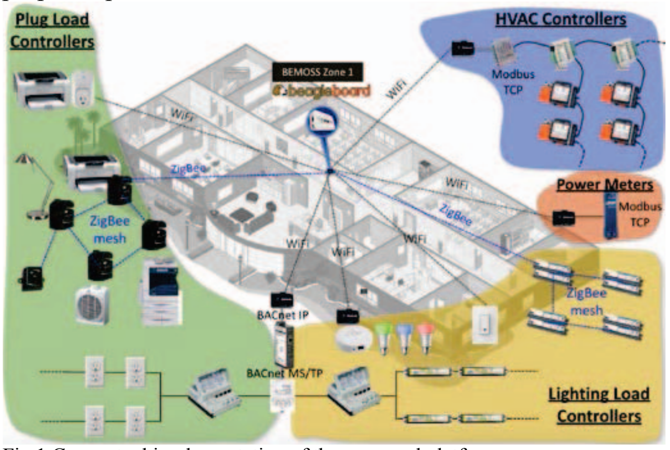


Fig.1.Conceptual implementation of the proposed platform.

IV. MAS DEVELOPMENT

The proposed platform software architecture, which comprises the following four layers: user interface (UI) layer, application and data management layer, operating system and framework layer, and connectivity layer. The more detail explanation of these layers can be found in [22].

MAS is a core fundamental of the proposed platform. With the current implementation, most agents reside in the operating system and framework layer. In addition, some agents are devised as applications residing in application and data management layer. This section devotes to the discussion of the agent development and its requirements in both layers.

BEMOSS agents developed inVOLTTRON™ [14] are of the following types: (1) device discovery agent, (2) control agent, (3) sensor agent, and (4) cloud agent. Each type of agents has different functionalities as described below.

1) Device discovery agent is responsible for detecting the presence of devices in a building, querying their model numbers, identifying their API interfaces, and launching control agents to monitor/control discovered devices.

2) Control agent includes thermostat agent, lighting load agent, plug load agent, VAV agent, and RTU agent. These agents are responsible for monitoring and control a thermostat, a lighting load controller, a plug load controller, a variable air volume (VAV) controller, and a rooftop packaged unit (RTU)respectively. Each of these agents will be automatically initiated and launched, if the device discovery agent discovers a corresponding device with the same type (e.g., thermostat, plug load, or lighting load etc.). It should be noted that one control agent is assigned particularly to one hardware device.

3) Sensor agent communicates with sensors (e.g., occupancy sensor, humidity sensor, ambient light sensor, etc.) and/or power meters to obtain their readings. Similar to control agents, sensor agents are automatically launched after the device discovery agent discovers the associated devices.

4) Cloud agent is an agent that communicates with cloud or web services such as an Open Automated Demand Response (OpenADR) agent. This agent receives demand response request from a utility or an aggregator (e.g., EnerNOC [23]) through a web service on the cloud. It then notifies selected agents of a DR event.

This section discusses agent development, including agent architecture, agent behavior design, agent knowledge representation, agent ontology, and agent communications. In addition, linkages between agents and the UI, agents and API interface, as well as application agents are also discussed as a guideline for developers wishing to develop agents or applications residing in the proposed platform.

A. Agent Architecture

Agent architecture is the fundamental mechanism underlying autonomous software components that support effective behavior in dynamic, real-world and open environments. Theoretically, agent architecture can range from a purely reactive (or behavioral) architecture that reacts to an environment in a simple stimulus-response fashion to a more deliberative architecture that reasons about its action based on Belief Desire Intention (BDI) model. Anyagent architecture can fall into four main categories: Logic based, Reactive, BDI, and Layered architectures depending on its required functionalities and capabilities.

Fig. 2 illustrates an example of a thread path of execution of a generic control agent modeled as a purely reactive agent that reacts to its environment such as its corresponding UI, applications, or other agents. A brief discussion of the thread path of execution is given as follows:

*Step 1:*Agent acquires its configuration including agent's parameter setting (e.g., agent id, agent message publish/subscribe addresses), device information (e.g., IP address, API interface), database interfaces.

*Step 2:*Agent instantiates device object from the loaded API interface to be able to communicate, monitor, and/or control a device.

*Step 3:*Agent initializes itself based on settings from the previous steps by declaring necessary variables and connecting with databases and other required services.

*Step 4:*With the DeviceMonitor behavior (discussed in Section IV.B), agent periodically gets a current status of a device by calling a method of anAPI interface. Then maps keyword and variables to agent knowledge (explained in Section IV.D) and update both metadata database and time-series database with the recent device status.

*Step 5:*Receivinga message sent by its corresponding UI or an application, an agent triggers one of its reactive behaviors (UpdateDeviceStatus, DeviceControl, and IdentifyDevice) according to a message's topic and content as described in the subsequent sections.

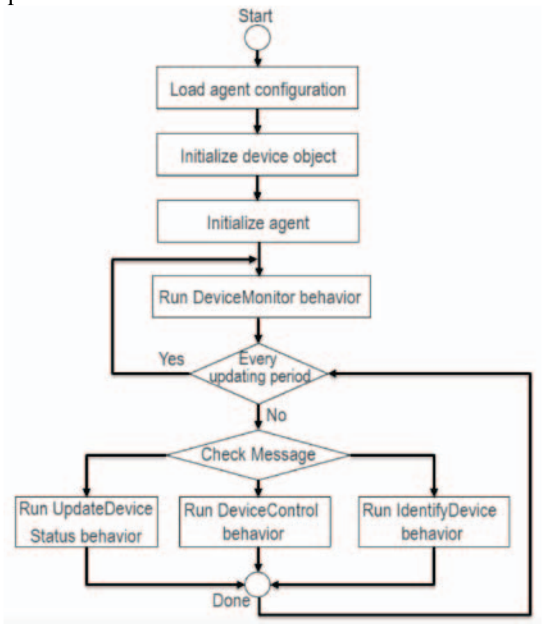


Fig.2. Control agent thread path of execution.

B. Agent Behavior Design

One of the most important steps of designing MAS is to design agent behaviors. Fundamentally, behaviors of an agent are its abilities to react to changes in its external environment as well as its neighboring entities in pursuit of a system goal(s) or its own goal(s). There are three common types of agent behaviors: one-shot, cyclic, and generic behaviors.

For example, regarding Fig.2,a generic control agent behaviors can be explained as follows:

- *DeviceMonitor* behavior is implemented as a cyclic behavior. A control agent will periodically update its knowledge on a device current status every a specified device monitoring time.
- *UpdateDeviceStatus* behavior is implemented as a generic behavior. It is called upon when other entities (e.g., UI, application, or another agent) would like to obtain a current status and setting of a devicesuch as current temperature, thermostat temperature set point orthermostat mode, etc.
- *DeviceControl* behavior is implemented as generic behavior. A control agent will update device control parameters (e.g.,thermostat temperature set point, change heat/cool mode and fan mode) by sending a control command using an API interface to change a current status/setting of a device.

- *IdentifyDevice* behavior is used in order to visually identify a device pertaining to a corresponding control agent. This behavior can be triggered by the UI sending identify device message to a control agent.

C. *Agent Knowledge Representation*

Meta data and time-series data are two types of knowledge that an agent needs to maintain to allow its interaction with other entities (e.g., the UI), as well as its reasoning processes. For each agent, there are two required tables to model an agent knowledge: metadata table and time-series data table.

•The metadata table is used to model agent knowledge with the data that have no timestamp, e.g., an agent identifier (AID) or an address of an agent.

•The time-series data table is used to model agent knowledge with the rest of the time-stamped data.

Table 1 gives example of common metadata of a control agent that is necessary for agents' knowledge modeling.

Table 1Metadata of a control agent

Attributes	Data type
- AID (Agent identifier)	AID object
- Address (e.g., IP, MAC)	string
- Zone	string
- Device type	string
- MAC address	macaddr

Table 2 gives an example of time-series data of a thermostat agent that is necessary for agents' knowledge modeling.

Table 2 Time-series data of a thermostat agent

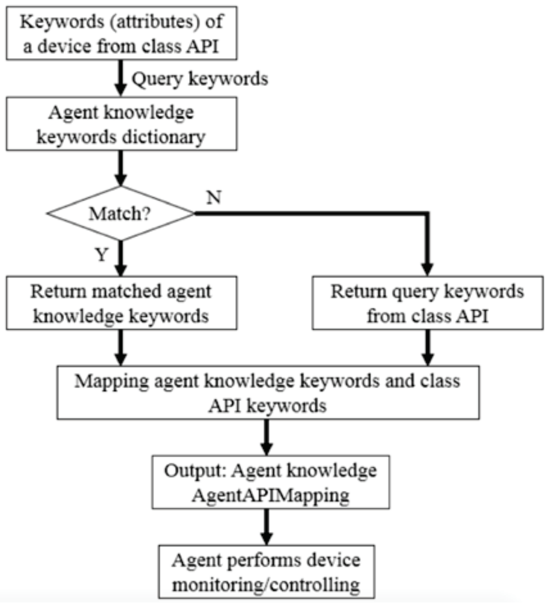
Attributes	Unit
- temperature	Farenheit
- thermostat mode	N/A
- fan mode	N/A
- heatsetpoint	Farenheit
- coolsetpoint	Farenheit
- thermostat state	N/A
- fan state	N/A

D. *Agent with API Interface*

An API interface allows an agent to communicate, monitor and control a device regardless of its communication technology or data exchange format (protocol-agnostic). In order to deal with heterogeneous application programming interface (API) documents offered by different hardware vendors, the mapping mechanism between agent's knowledge and an API interface is provided. This mechanism ensures that agent's knowledge obtaining from an API interface follows the agent ontology used throughout the platform for interoperability among agents and the other services. The mapping mechanism of a class API to an agent's knowledge is shown in Fig.3.

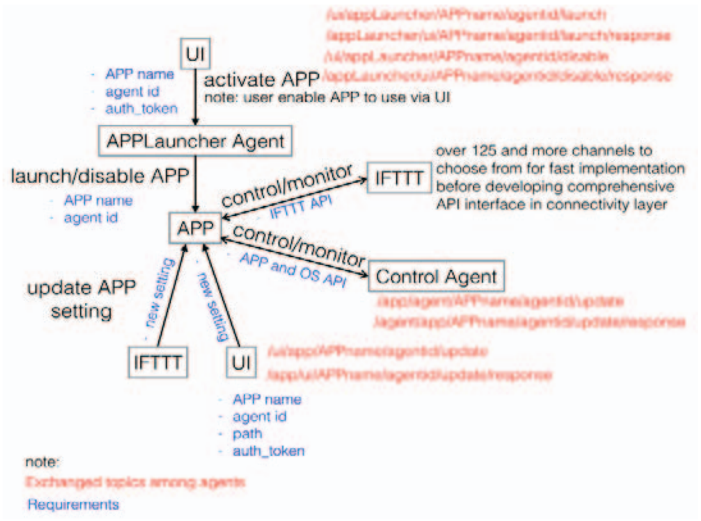
E. *Agent as an Application*

With its ability to communicate with other agents, web services, cloud services and database interfaces, an agent can also be developed as an application (App). Examples of possible applications include demand response, price-based management, operation monitoring, power and energy consumption analysis, load control based on local conditions, alarming notifications, planning and scheduling, data visualization and web services. Some of these applications and lab demonstrations are discussed in Section V. The application development is depicted in Fig.4.



Keywords (attributes) of a device from class API
Query keywords
Agent knowledge keywords dictionary
Match?
N
Y
Return matched agent knowledge keywords
Return query keywords from class API
Mapping agent knowledge keywords and class API keywords
Output: Agent knowledge AgentAPIMapping
Agent performs device monitoring/controlling

Fig.3.Mapping mechanism between agent knowledge and API interface.



/ui/appLauncher/APPname/agentid/launch
/appLauncher/ui/APPname/agentid/launch/response
/ui/appLauncher/APPname/agentid/disable
/appLauncher/ui/APPname/agentid/disable/response
UI
- APP name
- agent id
- auth_token
activate APP
note: user enable APP to use via UI
APPLauncher Agent
launch/disable APP
- APP name
- agent id
control/monitor
IFTTT API
IFTTT
over 125 and more channels to choose from for fast implementation before developing comprehensive API interface in connectivity layer
APP
control/monitor
APP and OS API
Control Agent
/app/agent/APPname/agentid/update
/agent/app/APPname/agentid/update/response
update APP setting
new setting
new setting
IFTTT
UI
/ui/app/APPname/agentid/update
/app/ui/APPname/agentid/update/response
- APP name
- agent id
- path
- auth_token
note:
Exchanged topics among agents
Requirements

Fig. 4.Development of BEMOSS Apps.

There are five essential elements of the application development architecture.

1) *User Interface (UI):*UI is deployed to carry out three activities: activate application, disable application and update new application setting.

1.1) Activate application: in order to activate application a message with the following topic and content should be published on the IEB. This message is picked up by the APPLauncher agent to start the requested application.

Topic: /ui/appLauncher/AppName/agentid/launch
Message content: {"auth_token": "Token to grant access to App"}

Where, AppName is a name of an application, agentid is an agent identification, auth_token is a token to grant access for other agents or entities to use this application.

1.2) Disable application: in order to disable application, a message with the following topic and content should be published on the IEB. This message is picked up by the APPLauncher agent to disable the specific application.

Topic: /ui/appLauncher/AppName/agentid/disable
Message content: {"auth_token": "Token to grant access to App"}

1.3)Update application setting: in order to update application setting, for example update a schedule for brightness setting of a lighting controller, a message with the following topic and content should be published on the IEB. This message is picked up by the corresponding App to update its setting sent by the UI or other web/cloud services.

Topic: /ui/app/AppName/agentid/update
Message content: {"auth_token": "Token to grant access to App", "path": "path to the App setting file (JSON format)"}

2) *APPLauncher agent:* each application is required to register with the APPLauncher agent so that it can be launched once a user activates application to use. In addition, it also serves when a user would like to disable the running application.

2.1) Activate application: upon receiving a launch App message from the UI, the AppLaucher agent looks up the database whether the requested application is validated, registered, and installed. Then it looks whether the requested agent is available and running by checking with the Platform Agent. Finally, it checks whether the provided authorization token (auth_token) is valid to launch the requested application. If these conditions are satisfied, the AppLauncher agent launches the requested application providing application name (APPName) and agentid. The recently launched application will start to communicate, monitor, and/or control the control agent (e.g., thermostat agent, plugload agent, or lighting agent) using the API between the application and data management layer and the operating system and framework layer (App and OS API). Once the AppLauncher finishes launching the requested App, it replies to the UI by sending back the following message:

Topic: /appLauncher/ui/AppName/agentid/launch/response
Message content: {"result": "success/failure"}

2.2) Disable application: upon receiving a disable message from the UI, AppLauncher agent looks up whether the requested agent (agent_id) is available and running by checking with the Platform Agent. Finally, it checks whether the provided auth_token is valid to disable the requested App. If these conditions are satisfied, the AppLauncher agent disables the requested application. Once the AppLauncher finishes disabling the requested application, it replies to the UI by sending back the following message:

Topic: /appLauncher/ui/AppName/agentid/disable/response
Message content: {"result": "success/failure"}

3) *Application (App):*App is designed to communicate, monitor, and/or control agent(s). The steps and requirements for developing App in the platform are the same as developing an agent with additional capabilities providing APIs between layers. After an App is successfully launched by the AppLauncher agent, it starts to communicate with a control agent using App and OS API. In order to control a device, the App needs to publish a message with the following topic and content on the IEB.

Topic: /app/agent/AppName/agentid/update/
Message content: {"control parameter": "setting"}

For example, to change a mode and temperature set points of a thermostat according to a user-defined schedule, the thermostat scheduler App with an agent_id = '1TH571a4760189f' needs to publish the following message to IEB.

Topic: /app/agent/thermostat_scheduler/1TH571a4760189f/update/
Message content: {"mode": "COOL", "setpoint": "72"}

4)*Control Agent:* upon receiving control message from the App, a Control agent (e.g., thermostat agent) changes the setting of a corresponding device (e.g., a thermostat) accordingly by using the API between OS layer and Connectivity layer. Once the Control agent successfully changes device setting, it replies back to the App by publishing the message with the following topic and content to IEB.

Topic: /agent/app/AppName/agentid/update/response
Message content: {"result": "success/failure"}

V. APPLICATION DEMONSTRATION

A laboratory has been set up to demonstrate features and capabilities of the developed platform. It includes a computer to host the platform and selected hardware devices that use different communication technologies and data exchange protocols as shown in Fig. 5. The following list showcases the applications that have been developed and implemented in the proposed platform.

- *Demand response (DR):* DR is an action to reduce electric power demand in order to reduce peak demand or avoid system emergencies, generally in response to signals received from a local utility (e.g., price signal or reliability signal). This application can be implemented in the platform using 1) an OpenADR agent receives a DR signal, 2) a planning/scheduling agent to implement DR algorithm and decide how much load to be shed for each load type, and 3) control agents to execute the decision (e.g., thermostat agent to change temperature set point, lighting agent to dim the lights).
- *Load management:* generally building residents have fixed schedules or certain behavior patterns. Thus it is wise to set up a load management application to optimize energy consumption on a daily basis. That requires 1) the user to set up their preference in UI (e.g., temperature control

schedule, lighting control schedule), 2) a planning/scheduling agent to process and convert these messages into control signals and 3) control agents to execute the schedule.

The proposed platform has also been tested on small-form-factor computers: PandaBoard [20] and BeagleBone Black [21]. This is to showcase that the platform can operate on hardware devices with limited resources in terms of computation power and memory. Additional devices are being explored for integration into the proposed platform.

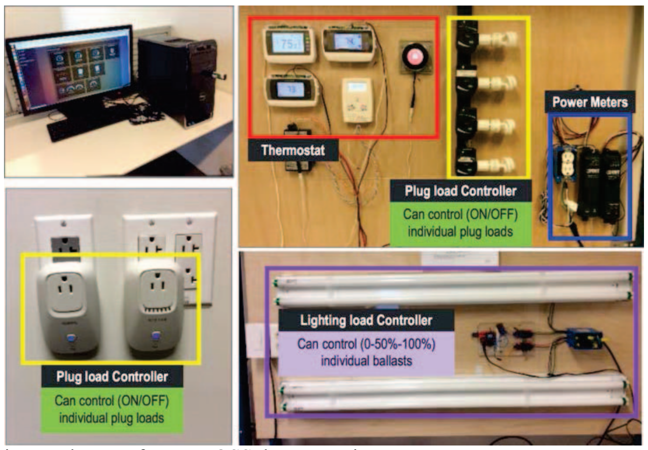

Thermostat
Plug load Controller
Can control (ON/OFF) individual plug loads
Power Meters
Plug load Controller
Can control (ON/OFF) individual plug loads
Lighting load Controller
Can control (0-50%-100%) individual ballasts

Fig. 5.Lab setup for BEMOSS demonstration.

VI. Conclusion

This paper presents an agent-based open-source software platform for building energy management (BEM), which fills a long-awaited gap in energy management in small to medium sized buildings. It is open-source, which allows developers with different skill sets to work on different layers; it is cross-standard, enabling vendors to build products suiting customer needs; it is user-friendly, providing customers a hassle-free experience with seamless integration and plug-n-play; it is cost-effective, which can promote its rapid deployment in the near future. In the long run, the proposed solution shows promise in opening up demand side ancillary services markets and creating opportunities for building owners. This in turn can help accelerate development of market-ready products like embedded BEM systems and device controllers for HVAC, lighting and plug loads. It also enables utilities and independent system operator (ISOs) to actively leverage DR as a partial substitute for generation reserve or transmission upgrade.

References

[1] U.S. DOE, *Buildings Energy Data Book* [Online]. Available: http://buildingsdatabook.eren.doe.gov.

[2] U.S. DOE, Office of Energy Eff.& Renewable Technology (EERE) [Online]. Available:http://energy.gov/eere/efficiency/buildings.

[3] PNNL, *Small- and Medium-Sized Commercial Building Monitoring and Controls Needs: A Scoping Study*, 2012[Online].Available:http://www.pnnl.gov/main/publications/external/technical_reports/PNNL-22169.pdf.

[4] U.S. EIA, *Commercial Building Energy Consumption Survey (CBECS)* [Online]. Available:http://www.eia.gov/consumption/commercial/reports/2012/preliminary/index.cfm.

[5] G.T. Heydt, C.C. Liu, A.G. Phadke, and V. Vittal, "Solution for the crisis in electric power supply," *IEEE Computer Applications in Power*, vol.14, no.3, pp.22-30, Jul. 2001.

[6] S. McArthur, E. Davidson, V. Catterson, A. Dimeas, N. Hatziargyriou, F. Ponci, and T. Funabashi, "Multi-agent systems for power engineering applications—Part I: Concepts, approaches, and technical challenges," *IEEE Transactions on Power Systems*, vol.22, no.4, pp.1743-1752, Nov. 2007.

[7] S. McArthur, E. Davidson, J. Hossack, and J. McDonald, "Automating power system fault diagnosis through multi-agent system technology," in *Proc. 37th Annual Hawaii International Conf. on System Sciences*, vol., no., pp.5-8, Jan. 2004.

[8] Z. Peng, S. Suryanarayanan, M.G. Simoes, "An Energy Management System for Building Structures Using a Multi-Agent Decision-Making Control Methodology," *IEEE Industry Applications Society Annual Meeting (IAS),* vol., no., pp.1-8, 3-7 Oct 2010.

[9] B. Asare-Bediako, W.L. Kling, P.F. Ribeiro, "Multi-agent system architecture for smart home energy management and optimization," *IEEE Innovative Smart Grid Technologies Europe (ISGT EUROPE)*, vol., no., pp.1-5, 6-9 Oct 2013.

[10] K. Mets, M. Strobbe, T. Verschueren, T. Roelens, F. De Turck, and C. Develder, "Distributed multi-agent algorithm for residential energy management in smart grids," *IEEE Network Operations and Management Symposium (NOMS)*, vol., no., pp.435-443, 16-20 Apr. 2012.

[11] Y. Rui, and W. Lingfeng, "Multi-agent based energy and comfort management in a building environment considering behaviors of occupants," *IEEE Power and Energy Society General Meeting*, vol., no., pp.1-7, 22-26 July 2012.

[12] S.D. Smitha, and F.M. Chacko, "Intelligent energy management in smart and sustainable buildings with multi-agent control system," *International Multi-Conference onAutomation, Computing, Communication, Control and Compressed Sensing (iMac4s)*, vol., no., pp.190-195, 22-23 March 2013.

[13] M.G. Simoes, S. Bhattarai, "Multi agent based energy management control for commercial buildings," *IEEE Industry Applications Society Annual Meeting (IAS)*, vol., no., pp.1-6, 9-13 Oct. 2011.

[14] B. Akyol, J. Haack, S. Ciraci, B. Carpenter, M. Vlachopoulou and C. Tews "VOLTTRON: an agent execution platform for the electric power system," in *Proc.3rd International Workshop on Agent Technologies for Energy Systems*, Valencia, Spain, June 2012.

[15] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE: A FIPA2000 compliant agent development environment," *in Proc.Fifth International Joint Conference on Autonomous Agents and Multigent Systems (AAMAS)*, Hakodate, Japan, 8-12 May 2006.

[16] Spade2 - Smart Python Agent Development Environment [Online]. Available: https://code.google.com/p/spade2/.

[17] AgentScape - Distributed Agent Middleware [Online]. Available: http://www.agentscape.org.

[18] B. Akyol, J. Haack, C. Tews, B. Carpenter, A. Kulkarni, and P. Craig "An Intelligent Sensor Framework for the Power Grid."*ASME 2011 5th International Conference on Energy Sustainability*, Washington, DC, USA, August 7–10 2011.

[19] J. Haack, B. Akyol, B. Carpenter, C. Tews, and L. Foglesong "VOLTTRON: An agent platform for smart grid." in*Proc.4th International Workshop on Agent Technologies for Energy Systems*, Minnesota, USA, May 10, 2013.

[20] PandaBoard [Online]. Available: http://pandaboard.org.

[21] BeagleBone Black [Online]. Available: http://beagleboard.org/black.

[22] W. Khamphanchai, A. Saha, K. Rathinavel, M. Kuzlu, M. Pipattanasomporn, S. Rahman, B. Akyol, and J. Haack, "Conceptual architecture of building energy management open source software (BEMOSS)," *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, vol., no., pp.1-6, 12-15 Oct. 2014.

[23] EnerNOC Open Source for an Open Grid [Online]. Available: http://open.enernoc.com/.