An Energy Information Gateway for use in Residential and Commercial Environments

Daniel Arnold, *Student Member, IEEE,*, Michael Sankur, and Dave Auslander

***Abstract*—Growing demands for products which enable consumers to manage their energy use more efficiently has led to the development of Energy Information Gateways, which are just beginning to gain traction in the marketplace. Such devices are envisioned to provide a communications and control infrastructure for appliances within their domain of influence, as well as communication with metering equipment. All relevant energy consumption information is expected to be relayed to the occupant in a practical manner. This paper outlines a software architecture for an EIG which is applicable to both residential and commercial environments. The software package utilizes the highly modular Open Services Gateway Initiative (OSGI) JAVA software framework to allow customization of individual EIG functionality and facilitates interoperability for existing smart grid products. In addition, the software in question provides a connection to external demand response resources and hosts a dynamic web based user interface to facilitate occupant interaction.**

***Index Terms*—Smart Grid, Communications, Demand Response, Demand Side Energy Management, Energy Information Gateway, Residential Energy Gateway**

I. Introduction

COMPUTATION platforms used to enable residential and commercial demand side energy management, often referred to as gateways, are devices gaining popularity with various Smart Grid entities. From the perspective of utilities, the growing presence of smart meters in homes across America could be coupled with such computing platforms to enable increased residential demand response during critical time periods. In the marketplace, gateway products fulfill a variety of roles, including enabling residential demand response, communicating energy usage information to the occupant, and interfacing with smart appliances.

More specific examples of the functionality of Gateways can be found in [1], where the authors outline a residential gateway architecture which enables communication between the residence and the utility to enable price and demand response signals to enter the home. A different set of functionality is demonstrated by [2], which supports a Plug and Play philosophy for integrating new appliances into the residential energy network. An architecture suggested by [3] features a residential energy gateway housed in a television set-top box.

While all of the architectures proposed thus far have merit, little attention has been paid in academia or industry to address interoperability issues. In particular, a lack of standardization in functionality, communications methods, and data structures has led to increased concerns regarding component/gateway interoperability when the home or commercial energy networks consists of elements of different nature. This issue can be addressed by an open design featuring a service oriented software architecture to integrate heterogeneous devices. Such an architecture is currently being developed in the Mechanical Engineering Dept. at the University of California, Berkeley.

This paper is organized as follows. Section II describes the general functionality an EIG is expected to contain and maps these into a set of requirements in both the residential and commercial spaces. Section III provides an overview of the service oriented architecture in which the EIG was created, the Open Service Gateway Initiative (OSGi). Section IV describes EIG functionality in the context of OSGi and Section V discusses results from the deployment of the EIG in managing a group of appliances during a demand response event.

II. EIG Supported Functionality

Arguably, the largest causes of interoperability issues amongst smart grid products stem from dissimilar communications media and data models. In the residential space, various energy information networks exist, including those based on Wi-Fi/Ethernet, ZigBee, ZigBee Smart Energy Profile, Z-Wave, etc.. In addition, there is no guarantee that devices which can communicate over the same protocol, Wi-Fi for example, will have compatible data models. For example, one device may exchange data in binary format, the other in Strings. The central thesis of this research project is to utilize an open software architecture to lessen the burden of integrating heterogeneous energy related elements.

As such, both the residential and commercial versions of the Energy Information Gateway created at UC Berkeley were developed to meet the following objectives:

1 Connect to a wide array of residential and commercial smart appliances
2 Connect to existing, or "Legacy", appliances
3 Connect to metering devices or building Energy Management and Control Systems (EMCSs)
4 Connect to the internet
5 Display relevant energy usage information to the resident
6 Accept input from resident to change (override) current operating conditions

This work was sponsored by the California Energy Commission (CEC) Public Interest Energy Research (PIER) program and the California Institute for Energy and the Environment (CIEE)

D. Arnold is a graduate student with the Department of Mechanical Engineering, University of California, Berkeley, CA, 94720 USA (e-mail: dbarnold@berkeley.edu.)

M. Sankur is a graduate student with the Department of Mechanical Engineering, University of California, Berkeley, CA, 94720 USA (e-mail: msankur@berkeley.edu).

D. Auslander is a Professor in the Graduate School in the Department of Mechanical Engineering, University of California, Berkeley, CA, 94720 USA (e-mail: dma@me.berkeley.edu).

7 Must be platform agnostic

The requirements of the EIG are purposely kept generic in order to capture as most of the functionality which currently exists in various EIG implementations. As an example, functionality such as relaying pricing or demand response signals into the home or office energy network fall under items 3 or 4.

A. Residential

In order to meet the generic objectives outlined above, the EIG team adopted a communications hierarchy outlined in Fig. 1. As the figure shows, the Residential EIG, referred to as the Residential Energy Gateway (REG), facilitates communications with a wide array of Home Area Network (HAN) elements, including a user interface, the smart meter, and any on-site local generation. In addition, with the exception of the smart meters periodic connection to the utility, the REG is the means by which HAN energy usage information is relayed to entities in the outside world.

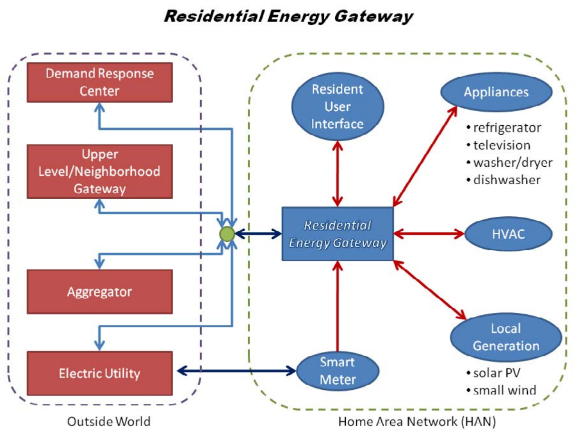

Fig. 1. Block diagram outlining communications capability of a residential energy gateway. As the figure shows, the REG bridges the gap between the Home Area Network (HAN) and the outside world.

Given the current lack of standardization for energy communication within the HAN, in order to meet the objectives outlined in the previous list, three critical issues needed to be addressed:

1 Allow smart appliances of different manufacture to interact via the Gateway. Such appliances are believed to have the ability to communicate their energy consumption and state of operation

2 Must allow ZigBee, Wi-Fi, Z-Wave HAN elements to interact with relative ease

3 Provide mechanism for legacy appliances to connect to the Gateway

Items 1 and 2 refer to the current fractured state of the smart appliance market, where appliances of different manufacture support dissimilar communications protocols and data structures. This places constraints on consumer choices in purchasing new smart appliances from different manufacturers. As such, in order to enable connections to as many smart appliances as possible, the predominant communications media must be supported in the REG, this being the ZigBee Smart Energy Profile 1.0-1.x. This standard is also the medium through which smart meter communications with the HAN are accomplished (though this functionality has yet to be activated in meters at large in California). A Wi-Fi connection easily allows communication with the outside world though the use of traditional web services and wireless networks are firmly embedded in American residences. Enabling legacy appliances to connect to the REG (item 3) is conveniently facilitated through the use of ZigBee or Wi-Fi load switches, which connect in series between the appliances power cord and the wall outlet. An example of a wall wart load switch is seen in Fig. 2. Such products are widely available.

Fig. 2. Typical wall wart design for series load switch to enable legacy appliances to connect to ZigBee or Wi-Fi networks, by courtesy of [4].

B. Commercial

EIG requirements for the commercial space are distinct in two sub-environments: 1) light commercial and 2) commercial office. In 1) a small office setting closely resembles a residence and may contain similar appliances. As such, the EIG team feels the requirements of the REG will satisfy the needs of this category. In 2) it is envisioned that a commercial version of the EIG, henceforth referred to as a Commercial Energy Gateway, or CEG, will operate over a domain consisting of one or several offices in a larger commercial building. The development of this CEG is considered here.

Fig. 3 outlines the communications architecture of the CEG in a commercial office setting. As the figure shows, the CEG communicates with local appliances and an occupant user interface. In settings where local control over HVAC and lighting systems is available, the CEG is expected to communicate with these entities as well. The CEG may also communicate with some sort of database where energy related information can be collected and analyzed. With regard to entities external to the local office environment, the CEG is expected to communicate with a building energy management control system (EMCS) and, possibly, the internet. These connections will allow demand response signals and other pertinent information to filter down to the CEG.

Communication protocols in a commercial office setting are largely expected to consist of those seen in a residential environment: ZigBee and Wi-Fi. Utilization of these communications protocols facilitates the incorporation of legacy appliances in the Office Energy Network (OEN) via wall wart load switches, as seen in Fig. 2.

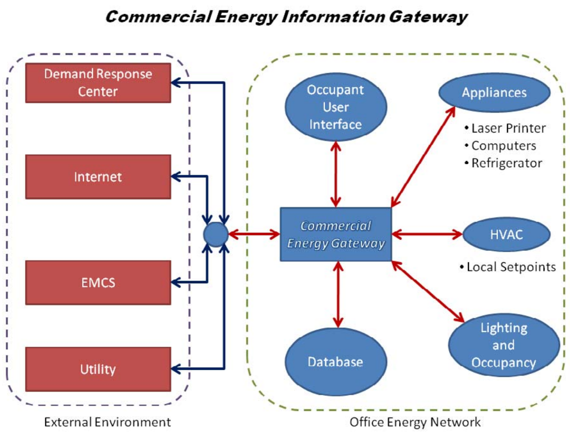

Fig. 3. The role of the CEG in enabling communications regarding energy related information in a small commercial office setting. The CEG also enables communication between the local office and the external environment.

III. Overview of the OSGi Software Framework

A. Overview

The Open Services Gateway initiative (OSGi) is a dynamic module system built on top of the JAVA programming language. The service oriented framework facilitates the development of compartmentalized pieces of code which are highly interoperable. Developing applications in this fashion reduces overall software complexity as the code modules, known as bundles in OSGi terminology, can be written, updated, versioned, and deployed independently. Bundle interaction is determined through the use of OSGi services, which facilitate the integration of third party applications into already existing programs. OSGi is a widely used software standard and is incorporated in applications such as the Eclipse IDE and Spring. Already four open source implementations of OSGi exist [5]. In addition, the developer community has created large bundle repositories, providing many useful services, including XML document handling and web hosting capabilities.

Fig. 4 shows a layered model of the OSGi framework. In the context of the figure, *Bundles* are modular pieces of code created by developers. These bundles are connected via *Services*, which create a publish-find-bind model for JAVA objects. The *Life-Cycle* layer defines an API to start, stop, update, install, and uninstall bundles. The *Module* layer defines how bundles are able to export and import pieces of code. Finally, the *Execution Environment* defines what methods and classes are available in a specific platform [5].

B. Modules and Bundles

For all intents and purposes, OSGi bundles are simply plain old JAVA Archive (JAR) files. In normal JAVA programming, the contents of JARs are visible to all other JARs. However, in the OSGi framework, by default, JAR (bundle) contents are hidden unless explicitly exported to the framework. In addition, bundles must explicitly import portions of code which they want to use. This functionality is enabled by the module layer.

Bundles can interact with each other through the use of services and the OSGi service registry. An illustration of two

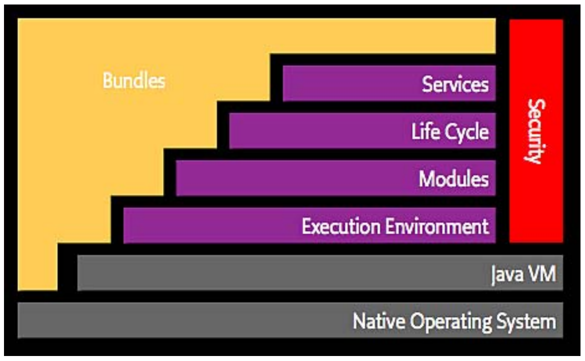

Fig. 4. A layered model of the OSGi framework, courtesy of [5]

bundles interacting via the use of services and the service registry is shown in Fig. 5. As shown in the figure, Bundle A can create an object which is then bound to a service and registered with the service registry, denoted by the triangular block S. Bundle B sees the service registry and can list all objects which are registered under a certain interface or class name. Bundle B can import, or get, the object provided by Bundle A if needed. In addition, bundle B can listen to the service registry and take specific action when the service provided by A is added, removed, or changed.

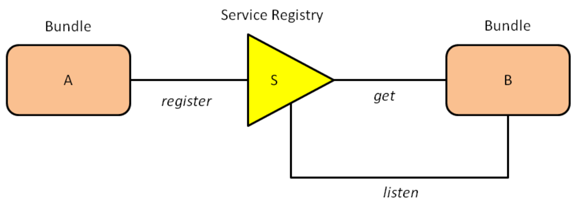

Fig. 5. Bundles and service interaction through the use of the service registry, denoted by *S*, courtesy of [5]

Services in the OSGi framework are dynamic, meaning that if bundle A withdraws its service from the registry, and bundle B was using that service, B must discontinue its use. Such a process allows bundles A and B to be operated on independently; one bundle can be running, while the other is being updated or taken off line.

IV. The EIG in the OSGi Software Framework

Both the commercial and residential versions of the EIG consist of collections of OSGi bundles. The bundles, in turn, support a variety of services which enable previously discussed Gateway functionality. At a bare minimum, the EIG features several services to support connections over different mediums, various data management services, and a web user interface bundle which facilitates resident interaction. These, and several other important services and bundles are described in this section. It should be noted that the service oriented architecture of OSGi allows, in this context, any proprietary Gateway functionality to be supported in parallel with other functions provided the proprietary software is cast in terms of OSGi bundles and services. The services discussed in this section are meant to illustrate the modular interoperability potential of the EIG in the OSGi framework.

A. JSON

In order to facilitate communications between bundles and with the web based user interface, the JavaScript Object Notation (JSON) format was adopted. JSON is a human-readable, lightweight data interchange format which is completely language independent [6]. This structure consists of objects which are made up of comma separated name/value pairs (the name/value pairs themselves are colon separated), where names have the string data type, see Fig. 6.

Objects themselves are surrounded by left and right braces and individual objects can be placed into arrays which are surrounded with left and right brackets, see Fig. 7. A simple JSON object which contains the name of a connected appliance and the appliance state might look like: $\{ApplianceName : Refrigerator, State : OFF\}$

A typical JSON array in the context of the EIG might look like: $[\{ApplianceName : Refrigerator, State : OFF\}, \{ApplianceName : Microwave, State : ON, ID : 1200356\}]$

This particular array consists of two JSON objects with dissimilar name/value pairs (the last object has an additional pair). This property is exploited in the EIG architecture and will be discussed later in this section. The supported data types for values in JSON objects are illustrated in Fig. 8.

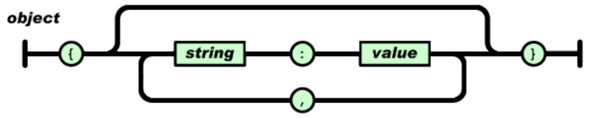

Fig. 6. JSON object architecture, courtesy of [6]

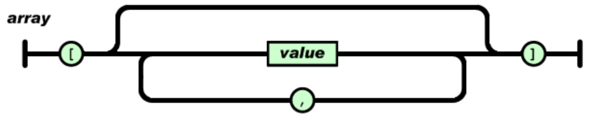

Fig. 7. JSON array architecture, courtesy of [6]

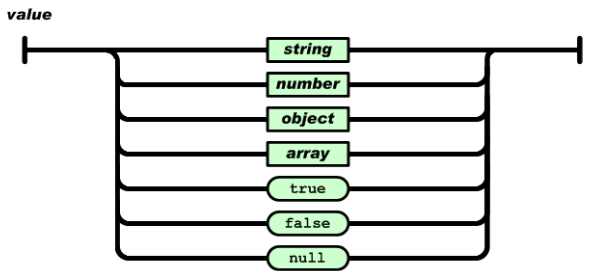

Fig. 8. JSON value supported data types, courtesy of [6]

B. Appliance Bundles

For all intents and purposes, an appliance is a generic term meaning any device which will exchange energy information with the EIG. In order to connect an appliance to the EIG, a developer must install a bundle representing that appliance in the OSGi software framework. This bundle will be the medium through which communication with the actual device is bridged with the EIG data structure.

The ability to index dissimilar data into JSON objects and arrays is quite advantageous in integrating different appliances, as we assume that the information passed from appliances into the EIG software framework is not the same. Simply put, appliances can communicate any information to the EIG they desire. The unique data from each appliance is mapped to JSON objects which are then made available to the rest of the EIG/OSGi framework through a JAVA interface. This process is illustrated in Fig. 9. As is shown, the appliances are of different nature as they communicate over different media (Wi-Fi and ZigBee in this case) and support different data structures (XML and Strings, respectively). However, the bundles corresponding to each appliance perform the same root function of mapping their unique data structures to the JSON format, and making interfaces to these JSON objects available to the rest of the framework. These JSON objects correspond to individual data points for each appliance.

Once the appliance data is in the JSON format, the EIG provides an aggregation service where JAVA interfaces to these individual JSON objects are indexed into a JAVA array list, see Fig. 10. As the figure shows, the Aggregation bundle can index an arbitrarily large number of JSON objects into a JSON array list, which is then made available to the Control and Web UI bundles.

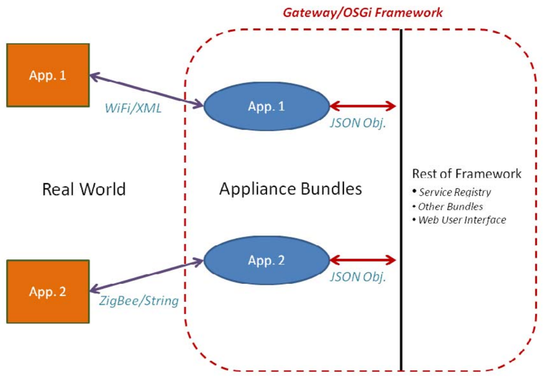

Fig. 9. An illustration of the role of appliance bundles in the EIG software framework. The bundles provide a mapping from their respective appliances communication and data model to the JSON format.

The upcoming sections describe the services provided in the current implementation of the EIG. Since the OSGi software framework allows services to be added and removed dynamically, new services can always be added to the framework without compromising the functionality provided by existing services.

C. Connection Services

In addition to mapping data structures, the appliance bundles written by developers also dictate the connection medium through which the appliance bundles/EIG communicate with the appliance itself. The core EIG software provides several services for facilitating connections over three different

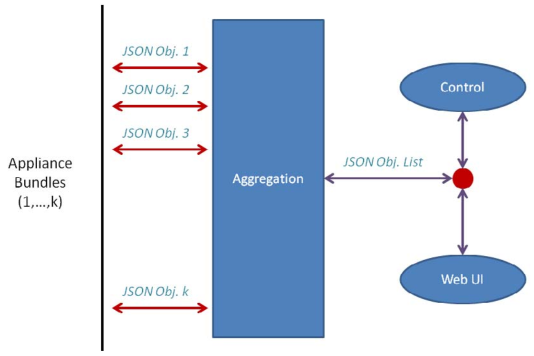

Fig. 10. The Aggregation bundle compiles all JSON objects from each appliance into a JSON array, which is then made available to other Gateway bundles.

Fig. 11. Telegesis ETRX2USB dongle. Provides a USB to serial bridge allowing for ZigBee wireless mesh networking, courtesy of [10].

mediums: 1) JAVA Sockets, 2) COM Ports, and 3) SSH Connections. Appliances bundles wishing to connect to their appliances over a particular standard can use these services to lessen the amount of customized code included in their bundles.

The JAVA Socket service provides an interface to allow connections over TCP/IP sockets in JAVA. Developers need only specify a port number over which the connection is to be made and the JAVA Socket service will create and manage a socket bound to that port and provides input and output streams to the appliance bundle for bi-directional communication. This service allows connections to appliances over Ethernet and Wi-Fi. It is assumed that the platform which houses the EIG contains the appropriate hardware and drivers to allow a connection to a Wi-Fi network. For example, this functionality was tested on a Netbook, which contains a built in Wi-Fi radio.

The COM Port service allows appliance bundles to read/write to a standard computer COM port. In order to utilize the service, appliance bundle developers must identify the COM port with which the EIG will communicate and the service will provide input and output streams to transmit information between the appliance bundle and the appliance itself. In addition to standard serial communications using a parallel, serial, or USB ports, many COTS ZigBee mesh network equipment provide drivers for their hardware to be recognized as a COM port by the EIG platform operating system. One such device is the Telegesis USB ZigBee dongle pictured in Fig. 11. This particular product provides ZigBee wireless mesh networking capabilities via a USB to serial bridge. In actual tests of the EIG, this product was used in conjunction with COM Port service to connect the EIG, running on a Netbook, to an appliance over a ZigBee wireless mesh network.

The SSH (Secure Shell) Service provides access to an open source library for SSH communications. Appliance bundles can consume this service to create an SSH connection using an IP address and a username and password (if required). The service provides input and output streams to the appliance bundle for bi-directional communications.

D. JSON Object List Service

JSON List Service provides an interface to access the JSON objects associated with individual appliances. This service is provided by the Aggregation bundle, shown in Fig. 10. Also included in this interface are methods to post data to individual appliance bundles, also in JSON format. The appliance bundle developers are meant to interpret these data objects as control signals, which should then be mapped back into the original appliance data structure and transmitted from the appliance bundle to the appliance itself. As Fig. 10 shows, this type of control is allowed in the Web User Interface bundle and in the Control bundle, as these are the two bundles which consume the JSON Object List Service.

E. OpenADR Service

Open ADR, or Open Automated Demand Response is an open communication standard for enabling demand response in commercial buildings [9]. In an investigation of demand response research potential in the residential sector, the EIG developers created an OpenADR bundle to allow the EIG to connect to an OpenADR server. This bundle registers OpenADR Service with the framework, which provides a connection to an Akuakom Demand Response Actuation Server via traditional web services (REST or SOAP). XML OpenADR signals are fetched, converted to a JSON format, and then exported via an interface bound to OpenADR Service for other bundles to consume. Currently, only the Web User Interface bundle and the Control bundle consume OpenADR Service.

F. Control

Currently, the Web User Interface allows for supervisory control over individually connected appliances through access of JSON Object List Service (see Fig. 10). However, the EIG contains a bundle, the Control bundle, which consumes JSON Object List Service and OpenADR Service. This bundle is provides an environment for developers to design automated control algorithms which operate on individual appliance data objects in JSON format (provided by the JSON Object List Service).

G. Web User Interface

The OSGi software framework provides bundles which support the hosting of Java Server Pages, or JSP [8]. These pages allow the interleaving of static HTML with actual JAVA or JavaScript code, thereby creating a more dynamic web page. In addition to JSP, OSGi allows for JAVA servlet hosting as well. A JAVA servlet can be thought of as a JAVA applet that runs server-side instead of client-side [7]. These servlets synergize well with dynamic client-side scripting techniques (made available through the use of JSPs), such as JavaScript/jQuery. In such client applications, JavaScript and jQuery (which is Javascrpt shorthand) can be used to execute calls to JAVA servlets independently. This process is illustrated in Fig. 12, which depicts the Web User Interface bundle hosting three servlets, each of which interacts with a different portion of JavaScript code running on a Java Server Page (JSP). An advantage to this modular approach brought about by the use of JAVA servlets is that portions of the client side JavaScript code can refresh the content of a servlet independently, without refreshing the entire webpage.

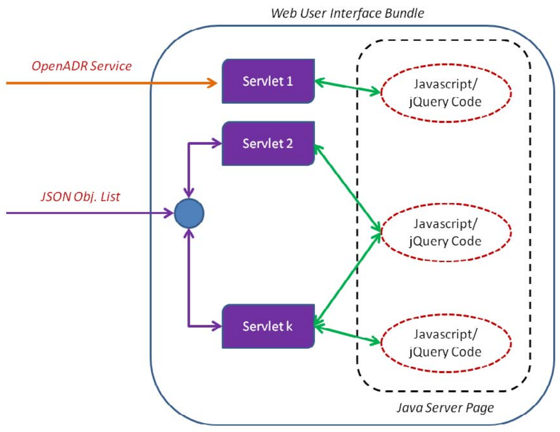

Fig. 12. Diagram of how JAVA servlets communicate with JavaScript and jQuery code in a JSP.

V. Results and Discussion

EIG functionality was successfully tested in a laboratory setting in a commercial office at the University of California, Berkeley. For this experiment, an EIG was deployed on a desktop computer, and was used to manage a group of simulated and physical appliances.

The simulated appliances were housed on laptops, one of which supported Wi-Fi connection capability with an XML data structure, the other supported a connection to a ZigBee Pro mesh network with a simple String based data structure. Connection to the ZigBee Pro mesh network was accomplished via the use of the Telegesis USB dongle discussed in the previous section. For each of these two ”appliances”, a unique appliance bundle was installed into the EIG OSGi framework. The appropriate connection service was consumed to lessen the amount of custom code written in each bundle. JAVA Socket service was consumed by the Wi-Fi Appliance bundle, and COM Port service was consumed by the ZigBee Appliance bundle. In each appliance bundle, code was written to support a mapping from the physical appliance’s data structure into the JSON format. This specific example was illustrated in the previous section when explaining the process of JSON data mapping (see Fig. 9).

For the experiment in question, connection to physical appliances was accomplished through the use of an intermediary device, the Raritan Power Distribution Unit (PDU) [11]. This device features eight independently metered and actionable 120V electrical outlets. A four outlet version of the device is shown in Fig. 13. Interaction with each outlet is accomplished via an SSH connection over IP. To connect the Raritan to the EIG, an appliance bundle was written which consumed the SSH connection service. Within the appliance bundle, eight individual JSON objects were constructed, one for each plug on the Raritan. Interfaces to all of these objects were exported to the OSGi service registry and subsequently aggregated into a JSON object array list in the *Aggregator* bundle.

Fig. 13. Photograph of the Raritan Power Distribution Unit (PDU), courtesy of [11].

All of the components previously mentioned are depicted in a photograph of the test laboratory, shown in Fig. 14. As illustrated, the Raritan PDU is connected to two appliances, a desk lamp and a fan. For this experiment, the devices shown were all actuated in response to an external demand response event. To enable this capability, a test event was programmed into an Akuakom Demand Response Actuation Server (DRAS) which communicates over the web with the EIG via OpenADR service. Once programmed into the DRAS, the event information was fetched by the EIG, which stored the information until the onset of the DR event, when a pre-programmed actuation signal was sent to each appliance bundle in JSON format. Each JSON control signal (actually a JSON object) was customized for each appliance. The individual appliance bundles parsed their respective JSON control signal into their original data format and transmitted the appropriate signal to the appliances themselves. For the case of the Raritan, the routing of the appropriate control signal to the correct outlet was handled in the Raritan appliance bundle. At the onset of the DR event, which had a 5 minute duration, all appliances were commanded to shut down. Following the 5 minute event, another JSON control signal was sent to each appliance, instructing a return to normal operations. These signals were staggered by 15 seconds to avoid a large measured energy spike associated with cold-load pickup.

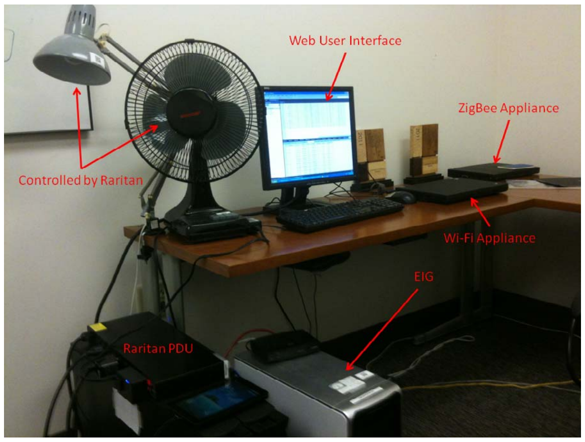

Fig. 14. Photograph of experimental test setup.

VI. Conclusion

The EIG has been shown to be able to connect to both simulated and physical appliances of dissimilar connection medium and data structure. This ability allows potentially any appliance to be integrated into the home or commercial energy networks. The modular nature of OSGi allows new connection services to be installed while the EIG is running. In addition the abstraction of unique appliance data into the JSON format facilitates information transfer from newly added appliances into the rest of the EIG, via the *Aggregation* bundle.

Quite unique about the architecture proposed in this paper is the fact that current proprietary Gateway functionality from any manufacturer can, potentially, be completely supported in the OSGi environment. If such proprietary functions are cast as OSGi services and registered in the service registry, the proposed EIG architecture would allow the functionality of different Gateways, which currently are separate physical entities, to run in parallel on the same platform. Such an option might be attractive to a consumer wishing to maximize his/her options in purchasing smart grid products. Simply put, a service oriented architecture has the potential to eliminate the need for different proprietary Gateway platforms.

The architecture presented in this paper also presents opportunities for the participation of third party vendors in new markets, namely, the development of appliance bundles and the development of control strategies to be housed in the *Control* bundle. The authors envision, eventually, the availability of a large repository of appliance bundles and control strategies on the cloud, from which the resident can select to be implemented into their EIG.

The authors plan to continue development of the EIG software to enable connections to ZigBee Smart Energy Profile 1.0/1.x networks, as well as a connection to the smart meter itself. In addition, integration of the Z-Wave communication protocol is to be investigated. In the commercial setting, the notion of inter-EIG communication could be leveraged to allow different commercial energy networks to ”auction” the ability to shed load during a demand response event.

It should be noted that the presence of EIGs in residences and in the commercial buildings presents several advantages from the perspective of the grid/utility in addition to the individual occupant who owns/operates the EIG. The EIG provides not only an energy information communications infrastructure, but constitutes a measurement and control platform, with the ability to bridge the gap between cloud resources and the commercial/residential energy network. As such, energy related quantities such as residential voltage waveforms and harmonics measurements could be transmitted to a distribution aggregation system for, perhaps, real-time analysis and control. In a companion paper, we outline several scenarios where using energy information obtained from a residence can be used to benefit the grid.

Acknowledgment

The authors would like to thank Ron Hofmann and Gaymond Yee for their excellent advice throughout the duration of this research project.

The authors also would like to thank the California Energy Commission (CEC), the California Institute for Energy and the Environment (CIEE), the United States Department of Energy (DOE), and the Siemens Corporation for providing funding for this research effort. In addition, the authors would like to acknowledge the fruitful results produced by collaboration with colleagues at the Lawrence Berkeley National Laboratory (LBNL), Environmental and Energy Technologies Division.

References

[1] A. Cuevas, C. Lastres, J. Caffarel, R. Martínez, and A. Santamaría, ”Next Generation Energy Residential Gateways for Demand Response and Dynamic Pricing,” in *Proc. 2011 IEEE International Conference on Consumer Electronics (ICCE).*, pp.543-544.

[2] N. Kushiro, S. Suzuki, M. Nakata, H. Takahara, and M. Inoue, ”Integrated Home Gateway Controller for Home Energy Management System,” *IEEE trans. on Consumer Electronics,* vol. 49, no. 3, pp. 629-636, 2003.

[3] A. Pal, C. BhauMik, J. Shukla, and S. Kolay, ”Energy Information Gateway for Home,” in *Proc. 2011 IEEE Second International Conference on Intelligent Systems, Modeling and Simulation.*, pp. 235-240.

[4] (2011 November) The HAI website. [Online]. Available: http://www.homeauto.com/main.asp

[5] (2011 October) The OSGi website. [Online]. Available: http://www.osgi.org/About/WhatIsOSGi

[6] (2011 October) The JSON website. [Online]. Available: http://www.json.org/

[7] (2011 October) JAVA Servlet Wiki. [Online]. Available: http://en.wikipedia.org/wiki/Java_Servlet

[8] (2011 October) JSP Wiki. [Online]. Available: http://en.wikipedia.org/wiki/JavaServer_Pages

[9] (2011 March) OpenADR Wiki. [Online]. Available: http://en.wikipedia.org/wiki/Open_Automated_Demand_Response

[10] (2011 November) Telegesis ETRX2 webpage. [Online]. Available: http://www.telegesis.com/products/etrx2_usb.htm

[11] (April 2011) Raritan webpage. [Online]. Available: http://www.raritan.com/

[12] (April 2011) Akuakom webpage. [Online]. Available: http://www.akuakom.com/

Daniel Arnold graduated with a BS in mechanical engineering from the University of California San Diego in 2005. He received his Masters the following year in the same department. His research focused on autonomous, 2-dimensional vehicle navigation with minimal sensory input. Since receiving his masters in 2006, Daniel has been employed by the Space and Naval Warfare Center in San Diego, Ca. where he worked as a test and evaluation engineer developing unmanned underwater vehicles. He later worked for the Naval Facilities Engineering Service Center in Port Hueneme, Ca, where he researched ocean renewable energy technologies for application to navy facilities. In the fall of 2009, Daniel began his PhD studies at UC Berkeley in the mechanical engineering department. His research is focused on the implementation of residential demand side management via the introduction of communications infrastructure in the home. His other interests include power systems, controls and estimation theory, and robotics.

Michael Sankur is from Ventura, CA. He attended UC San Diego studying mechanical engineering. During his undergraduate studies, he worked in vairous internships, including for SPAWAR and Hughes Research Labs, and held a research assistant position for an environmental engineering professor. After graduating, he earned an MS in aerospace engineering at UCSD as part of a five-year program. He then was awarded the NDSEG fellowship and started PhD studies in the Mechanical Engineering Dept at Berkeley. He is currently working under Dr. Dave Auslander. His research interests include plug load control and communication, and commucation and negotiation schemes between local plug load controllers.

Dave Auslander David M. Auslander is Professor of the Graduate School, Mechanical Engineering Department, University of California at Berkeley. He has also served as Associate Dean and Acting Dean of the College of Engineering. He has interests in dynamic systems and control. His research and teaching interests include mechatronics and real time software, bioengineering, and mechanical control. Current projects in these areas are building energy control, design methodology for real time control software for mechanical systems, satellite attitude control, simulation methods for constrained mechanical systems, and engineering curriculum development. He consults in industrial servo control systems and other control and computer applications. He is co-founder and senior technical consultant to Berkeley Process Control, Inc. (now a part of Moog, Inc.), a company specializing in industrial machine control. His undergraduate studies were at the Cooper Union and his graduate studies were at MIT, both in Mechanical Engineering. He has been awarded the Levy Medal (best paper) from the Franklin institute (twice), the Education Award of the Dynamic Systems and Control Division of ASME, the Education Award of the American Automatic Control Council, the Control Practice Award of the Dynamic Systems and Control Division of ASME, the Donald P. Eckman Award of the Instrumentation, Systems, and Automation Society, IEEE/ASME Mechatronics and Embedded Systems Applications (MESA) Career Award, and is a Fellow of the ASME. He has a longstanding association with the Dynamic Systems and Control Division of ASME including past service as its chair and as the editor of the *Journal of Dynamic Systems, Measurement and Control*.