Residential Energy Gateway – Phase II Report

Principle Investigator: Professor Dave Auslander
Graduate Student Researchers: Daniel Arnold, Michael Sankur, Kevin Ding

University of California, Berkeley
Dept. of Mechanical Engineering

Table of Contents

List of Figures

List of Tables

Introduction

This report describes the work done during Phase II of the Residential Energy Gateway (REG) project, spanning the timeframe from Jan. 1, 2010 – Sept. 30, 2010. Work was supported via funding from the California Energy Commission (CEC), administered through the California Institute for Energy and the Environment (CIEE). Research and design of the REG was conducted in the Mechanical Engineering Dept. at UC Berkeley.

This document assumes familiarity with the Phase I report, which provides background and motivation for the construction of the REG. Also included in the Phase I report is a description of the elements of the Home Energy Network (HEN), also known as the Home Area Network (HAN). This report equates these terms and treats them interchangeably. What follows now is a brief overview of important concepts from the Phase 1 report.

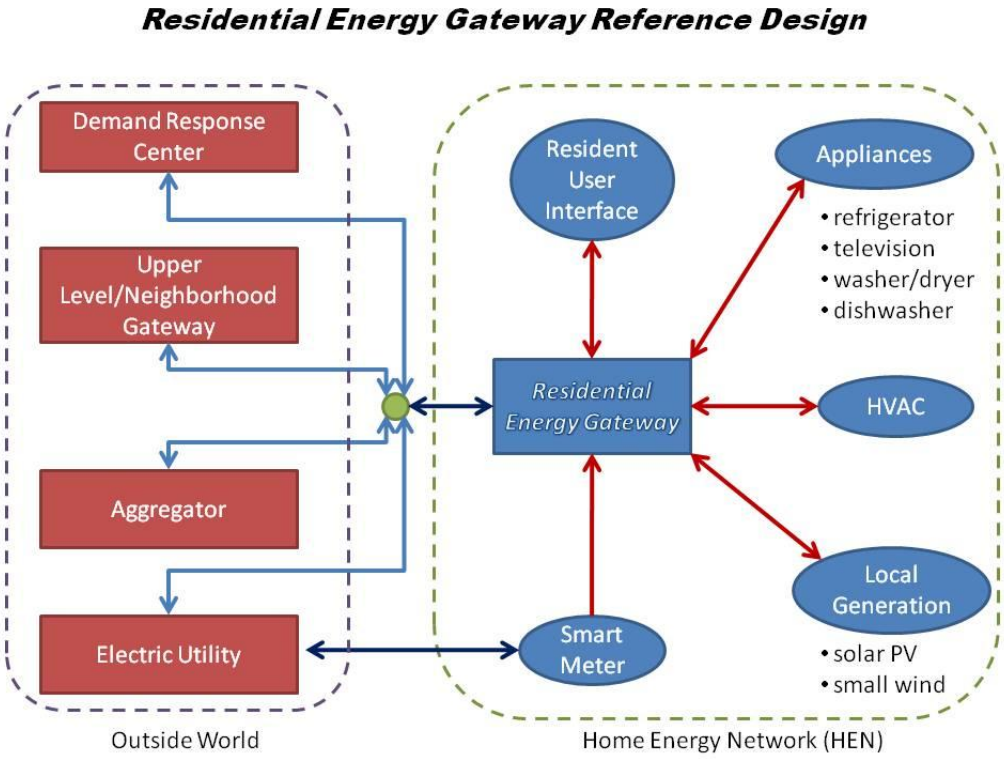

Figure 1 - Conceptual overview of Gateway/HEN interactions within the home and the outside world

Figure 1 provides a conceptual overview of the Gateway/HEN interactions within the home and with the outside world. All arrows in the figure denote energy usage data/information flow. As the figure shows, all energy information from all other entities within the HEN is passed to the Gateway. Most information flow is bi-directional, with the exception of the smart meter, which is not required to accept data from the Gateway. Also, with the exception of the utility/smart meter communication, the REG is modeled to provide the primary means of connectivity from the HEN to the outside world.

The REG software package is being written in JAVA and developed in a Windows OS. The code is being written in the most portable way possible, as to minimize overhead when moving the software to Linux or Unix. The REG utilizes the Open Services Gateway initiative (OSGi Alliance, 2010) software framework for JAVA which allows components of the Gateway to be compartmentalized in a modular fashion. OSGi will be discussed substantially in this report. It was decided that the prototype REG would support communications over Wi-Fi/Ethernet and ZigBee. The ZigBee stack over which communications takes place thus far is generic, and not S.E.P. 1.0 or 2.0, as no COTS hardware could be located to suit the project's needs. Further motivation behind these decisions can be found in the Phase 1 report.

As the REG was meant to allow universal Plug and Play with HEN elements communicating through Wi-Fi or ZigBee, a standardized data model was adopted to facilitate uniformity. Data is passed from the HEN elements to the Gateway in the form of formatted JAVA Strings in "pseudo" XML format:

<value>36.5</value>

Data formatted in this manner is easily parsed by the Gateway and can be passed between HEN elements with relative ease. JAVA Strings were adopted as the medium to encapsulate this data as it is inefficient to pass DOM objects (representing XML data) through network sockets and COM ports in JAVA. While this data structure is fixed within the Gateway, the ability exists to utilize full JAVA XML DOM objects as a data model for communication with outside entities.

OSGi

The Open Services Gateway Initiative, or OSGi, framework is, for lack of a better term, the heart of the Gateway. OSGi provides a framework running within the JAVA runtime, which allows for the dynamic use of OSGi bundles (very similar to JAVA Jar files) (OSGi Alliance, 2010). Bundles can be installed/uninstalled to the framework and started/stopped using a simple command window style interface to the OSGi framework. This feature lends itself well to modular programming, and to the Gateway project in particular as appliances wishing to connect to the Gateway would simply install a representative bundle into the OSGi framework to communicate with the Gateway.

Perhaps most importantly, the OSGi framework is a "service" oriented architecture, meaning individual bundles can export a "service" (JAVA interface, or JAVA interface object) to the OSGi framework, and other bundles can consume that service (instantiate an interface object, or access the provided interface object). More advanced operations involving ServiceFactory and ServiceListener can be implemented to automate the exporting and consumption of services (OSGi benefits, 2009) (OSGi Bundle Interface, 2010). These important concepts will be discussed later in this section.

OSGi – Bundle Framework Overview

An OSGi bundle is, for all intents and purposes, an encapsulated JAVA project which is installed and run inside the OSGi framework. In this sense it is similar to a JAR file, except for its restriction on the environment in which it can be run. At minimum, all bundles must contain a MANIFEST.MF file, which defines the lifecycle of the bundle and specifies any external dependencies. It is also common for bundles to contain an ACTIVATOR.java class which defines bundle behavior. Note that only the MANIFEST.MF file is required, while the ACTIVATOR.java file is optional. Within the OSGi framework, bundles are identified by a unique number, known as the bundleID. The bundleID is automatically assigned by the OSGi framework and can be specified if need be (OSGi Bundle Interface, 2010).

The ACTIVATOR.java file (or simply "activator") imports certain packages from the OSGi framework and implements the interface *BundleActivator*. The activator contains 2 methods: *start()* and *stop()*. These methods are called when the following commands are entered into the OSGi command line:

start <bundleID>

stop <bundleID>

The *start()* and *stop()* methods are left to be populated by the designer. As one might assume, these methods define the operations the bundle will execute when the respective method is called in the OSGi framework.

Both the ACTIVATOR.java and the MANIFEST.MF files are used when exporting or importing a service. When exporting a service from a bundle, within the ACTIVATOR.java file an interface or interface object is registered as a service by passing a ServiceReference object specific to the service to the framework. The package containing the interface implementation must also be exported to the framework via the MANIFEST.MF file.

When importing a service (into a separate bundle), a bundle developer would import the package containing the interface implementation via the new bundle's MANIFEST.MF file, and then register for the ServiceReference object (exported by the source bundle) in the second bundle's ACTIVATOR.java. A significant advantage to this "service" oriented architecture is knowledge of the source bundle is not required by the bundle consuming a particular service (a.k.a. the "sink" bundle). When attempting to consume a service, the sink bundle looks to the OSGi *framework* and not the source bundle to first locate and then consume the service in question. If he service is not found, the framework does not crash, it throws a catchable error. This allows for bundles to be dynamically added/removed from the framework in a stable manner.

OSGi – ServiceFactory

The OSGi ServiceFactory interface can be used to create a ServiceFactory object instead of a service object to allow the bundle developer to gain control of a specific service object, rather than the interface which created the service object (OSGi ServiceFactory, 2010). Simply put, the ServiceFactory allows one to independently manage many objects of the same service. A typical service object does not allow for this, as when creating multiple service objects, from the same source bundle, the latest service object overwrites the previous one. The REG design makes use of the ServiceFactory when creating a new Wi-Fi or ZigBee connection, as each connection creates a unique service object from the same service (or JAVA interface).

OSGi – ServiceListener

The ServiceListener is an OSGi interface which allows a bundle to automatically listen for changes to services in the OSGi framework. Specific actions can then be taken whether the service is being registered, unregistered, or modified. The ServiceListener also allows for services to be filtered according to different criteria (OSGi ServiceListener, 2010). For the purposes of our project, the ServiceListener interface is implemented in the Gateway Control bundle.

Gateway Services

The following section describes the various services the Gateway provides to the OSGi framework. Not all services are available for external bundles (representing HEN elements) wishing to connect to the Gateway. It should be noted that these services are subjected to change as the REG is constantly evolving.

NetService

The Gateway provides the NetService service for HEN elements wishing to connect via Wi-Fi or Ethernet connections. Both the HEN element and the Gateway must be on the same local area network for this connection to be established. The NetService service makes use of the OSGi ServiceFactory, as an undetermined number of bundles will be consuming the same service object. ServiceFactory allows the OSGi framework to distinguish these different implementations of the same service object. Bundles importing the NetService service will have access to the interface INetService, which is characterized by the following methods:

public void makeWiFiConnection(String name, int portNum, int timeOut, boolean writeEnable);

public void closeWiFiConnection();

The first method, *makeWiFiConnection()* is the method which will create a Wi-Fi connection to the appliance in question. This method creates a JAVA network socket and opens input and output streams to the external appliance. The arguments for the method are as follows:

- *String name* specifies the name of the appliance or the name with which the connection will be known in the OSGi framework

- *int portNum* is the specific port over which the Gateway will establish a network socket
- *int timeOut* specifies the timeout on the network socket connection protocols, or the amount of time (in millis) before the network socket will abort waiting for a socket connection. This timeout is implemented in a while loop, meaning if the timeout is reached, the network socket connection attempt will abort, but will be recalled in the next iteration of the while loop
- *Boolean writeEnable* informs the Gateway that the appliance in question can controlled (will accept data from the Gateway). For smart appliances, this boolean is "true" and for "dumb" appliances this field is "false".

The second method, *closeWiFiConnection()* safely closes the network socket connection and all input and output streams between the Gateway and the appliance.

ZigBeeService

The Gateway provides the ZigBeeService service for HEN elements wishing to connect to the Gateway over IEEE 802.15.4. Both the Gateway and the appliance must have ZigBee radio capability to establish this connection. Hardware used to enable this connection will be discussed later in this document. The ZigBeeService service makes use of the OSGi ServiceFactory, as an undetermined number of bundles will be consuming the same service object. ServiceFactory allows the OSGi framework to distinguish these different implementations of the same service object. Bundles consuming ZigBeeService will have access to the IZigBeeService interface, which contains the following methods:

public void makeZigBeeConnection(String name, String dongleIDno, boolean writeEnable);

public void closeZigBeeConnection();

The first method *makeZigBeeConnection* is the method which will create a connection over ZigBee between the appliance in question and the Gateway. The arguments for this method are described as follows:

- *String name* describes the name of the appliance wishing to connect to the Gateway, or the name with which the connection will be known within the OSGi framework.
- *String dongleIDno* describes a unique 16 digit Telegesis ZigBee dongle ID (similar to a fixed IP address), which identifies a specific ZigBee dongle. The ZigBee dongles are USB compatible and create a bridge between a ZigBee connection and a simulated COM port in Windows OS (more on this later).
- *boolean writeEnable* informs the Gateway as to whether the external appliance will accept control signals from the Gateway. For smart appliances, this field is "true", for dumb appliances, this field is "false".

The second method in the interface, *closeZigBeeConnection()*, safely closes the COM ports associated with the Telegesis ZigBee dongles, as well as all input and output streams.

ControlService

Each external appliance connection, Wi-Fi, or ZigBee (each implementation of NetService or ZigBeeService) will bind it's implementation of NetService or ZigBeeService to a new interface object of IControlService and then will export this unique service object to the OSGi framework. This is a very technical way of saying that each appliance is individually controllable. All ControlService service objects are then consumer by the Control bundle where control action takes place. All ControlService objects have access to the IControlService interface which is characterized by the following methods:

public void setControlValue(String l);

public double getCurrentEnergy();

public String getConnectionName();

public void setControlStatus(boolean status);

public Boolean getControlStatus();

The method *setControlValue(String l)* commands the external appliance to set it's new energy value to that described by the String l. A JAVA String object is used in place of a double as all internal communications between external appliances and the Gateway follow "pseudo" XML format as described earlier in this document, which consist of JAVA String objects.

The method *getCurrentEnergy()* returns the current energy usage of the appliance in question at the current instant in time as a double.

The method *getConnectionName*(); returns the name of the connection as a JAVA String specified when calling the *makeWiFiConnection* or *makeZigBeeConnection* methods (described above).

The method *setControlStatus(boolean status)* sets a flag in the NetService or ZigBeeService object indicating that the connection in question is, or is not, currently being controlled.

The method *getControlStatus()* returns the control status flag in the NetService or ZigBeeService object which indicates if the object is currently being controlled by the Gateway.

OpenADRService

Provided an active internet connection exists to the Gateway, this service allows the Gateway to connect to an Akuakom DRAS, from which demand response information can be read. An external user can login to the DRAS and schedule demand response events, which can then be interpreted by the Gateway. Bundles implementing the OpenADRService service will have access to the IOpenADRService interface which is characterized by the following methods:

public String getEventStatus();

public void closeOpenADRService();

The method *getEventStatus()* returns a JAVA String indicating the state of a DR event. This String is always one of the four following Strings: "NONE", "FAR", "NEAR", "ACTIVE". Please see the Akuakom OpenADR Client Development program user guide (Akuakom, 2009) for configuration of the DRAS for further explanation of these Strings. The guide and example software can be found here: http://www.openadr.org/pdf/OpenADR-2008-R2.zip

The method *closeOpenADRService()* safely closes the network socket connection to the DRAS and all relevant input/output streams.

Gateway Core Bundles

This section specifies the various bundles which comprise the REG. Each bundle is discussed individually. Please note that the contents/functions of these bundles is likely to change as the Gateway continues to evolve.

Gateway.OSGi.util – Utility Bundle

The utility bundle in the REG provides JAVA interface classes for all of the services being exported or consumed in the framework (ControlService, NetService, OpenADRService and ZigBeeService). Each interface is contained in a separate package, and each package is made visible to the framework via the MANIFEST.MF file. The utility bundle contains no ACTIVATOR.java class so it cannot be started or stopped in the framework. This bundle can only be installed or uninstalled in the framework. Table 1 shows the individual java interface files and package organization in the utility bundle.

Package	JAVA files	Description
.controlservice	IControlService.java	Interface for *ControlService*
.netservice	INetService.java	Interface for *NetService*
.OpenADRService	IOpenADRService.java	Interface for *IOpenADRService*
.zigbeeservice	IZigBeeService.java	Interface for *IZigBeeService*

Table 1 - Bundle description for Gateway.OSGi.util

Gateway.OSGi.WiFi - WiFi Bundle

The WiFi bundle implements the NetService service and creates an interface object, INetService, which is then exported to the framework. Since this bundle uses the interface INetService, provided in Gateway.OSGi.util, the package Gateway.OSGi.util.netservice, must be imported to the WiFi bundle via the MANIFEST.MF file.

NetService provides all of the code necessary to establish a connection with an external appliance via JAVA network sockets, but only exposes the methods in INetService (used for making a WiFi connection). In addition the WiFi bundle manages all input/output streams

between the Gateway and the appliance as well as data logging. Since the WiFi bundle has an activator, it can be manually started or stopped in the framework.

The WiFi bundle implements the OSGi ServiceFactory when registering INetService to the framework. The ServiceFactory allows multiple unique service objects created from the same interface to exist simultaneously in the framework. This powerful feature allows an undetermined number of WiFi connections to be made to the Gateway, and allows for each connection to be managed individually. The individual packages and their inherent java files which makeup the WiFi bundle can be seen in Table 2.

Package	JAVA files	Description
.wifi	WiFiActivator.java	Activator for WiFi bundle
.wifi.internal	DataLog.java, NetService.java, INetServiceFactory.java	Datalogger, NetService class, and interface for NetServiceFactory

Table 2 - Bundle description for Gateway.OSGi.WiFi

Gateway.OSGi.ZigBee - ZigBee Bundle

The ZigBee bundle implements the ZigBeeService service and creates an IZigBeeService interface object, which is then exported to the OSGi framework via the MANIFEST.MF file. The ZigBee bundle uses the IZigBeeService interface provided by the utility bundle, therefore the package Gateway.OSGi.util.zigbeeservice must be imported via MANIFEST.MF as well.

Similarly to the WiFi bundle, the ZigBee bundle contains all of the code necessary to establish or join a ZigBee mesh network, except without the use of JAVA network sockets. A USB dongle was used to allow for ZigBee connectivity on a laptop. The dongle in question, the Telegesis ETRX2USB (http://www.telegesis.com/products/test_page_2.htm) provides connectivity over IEEE 802.15.4, thereby providing ZigBee capability to the Gateway. Equipped with this device, the REG can host or join a mesh network. Firmware is provided by Telegesis which allows the dongle to operate as a virtual COM port in Windows, thereby accessible by any terminal software (Telegesis, 2009). The data logging feature of the ZigBee bundle is identical to that of the WiFi bundle.

The ZigBee bundle also implements the OSGi ServiceFactory to allow multiple ZigBeeService objects to be created independently and exist simultaneously. The packages and java files which comprise the ZigBee bundle are visible in Table 3.

Package	JAVA Files	Description
.zigbee	ZigBeeAcitvator.java	Activator for ZigBee bundle
.zigbee.internal	DataLog.java, ZigBeeService.java, IZigBeeServiceFactory.java	Datalogger, ZigBeeService class, and interface for IZigBeeServiceFactory
gnu.io	**see appendix for full list of files	Allows JAVA to access windows COM ports

Table 3 - Bundle description for Gateway.OSGi.ZigBee

Gateway.OSGi.OpenADR - OpenADR Bundle

The Gateway.OSGi.OpenADR bundle provides classes which establish a connection to an Akuakom server which allows the Gateway to receive ADR signals from an Akuakom DRAS. Example code provided by Akuakom facilitates the connection to their DRAS. The OpenADR information is downloaded to the Gateway as a JAVA DOM object. The OpenADR bundle then parses the DOM object and extracts the DR event status flag, which is always NONE, NEAR, FAR or ACTIVE. The OpenADR bundle also registers OpenADRService with the OSGi framework. The packages and subsequent JAVA classes which comprise the OpenADR bundle are shown in Table 4.

Package	JAVA files	Description
.openADR	OpenADRActivator.java	Activator for OpenADR bundle
.openADR.internal	OpenADRService.java	Provides methods for IOpenADRService
com.akuakom.pss2.clientws.rest.client	RestWSClient2.java	Akuakom source code to connect to their DRAS

Table 4 - Bundle description for Gateway.OSGi.OpenADR

Gateway.OSGi.Control - Control Bundle

The control bundle consumes all service objects of the ControlService type. ControlService service objects are exported by OSGi bundles representing external appliances within the framework. Such bundles are the subject of the next section. The control bundle implements the OSGi ServiceListener in its activator, allowing the control bundle to automatically consume an object of ControlService the instant it is registered with the framework. In addition, the ServiceListener allows the control bundle to unregister a service when the specific appliance bundle is stopped or uninstalled in the framework. The package description for the control bundle can be seen in Table 5.

Package	JAVA files	Description
.control	ControlActivator.java	Activator for control bundle
.control.internal	GatewayControl.java	Provides all control logic for Gateway

Table 5 - Bundle description for Gateway.OSGi.Control

Currently, the Gateway control logic is very rudimentary, as the purpose of this phase of the project was to demonstrate communications capability. Once an OpenADR signal from an Akuakom DRAS is received, the Gateway will take specific action depending on the state of the event status flag. For example, if the event status is “ACTIVE”, the Gateway will command all connected appliances to shut down. While this logic is simplistic, it demonstrates that the Gateway has the communications capability to interpret an external demand response signal and

actuate appliances based on that signal. The development of more refined control logic will be pursued in the 3rd phase of this project.

Gateway.OSGi.WebUI – Web User Interface Bundle

The Gateway web UI bundle will manage the lifecycle of the web user interface, which is currently being developed. The importance of the user interface in the context of both the Gateway and greater home energy management cannot be understated. Currently, the rudimentary web based user interface will provide the following functionality:

Register/Unregister appliances: The user interface is the most logical place to include functionality to register or unregister appliances within an individual's HEN. Figure 2 shows a screenshot from the current Gateway web user interface which will be used to facilitate appliance addition into an individual's HEN. As the figure shows, the user would need to insert some relevant parameters (not necessarily those seen in the figure) specific to the appliance in question. Upon clicking the "create" button, the Gateway would, using the parameters entered on the website, create an external appliance bundle (discussed in a following section) specific to the appliance being connected, and install that bundle to the Gateway OSGi framework. A similar, but reversed, process would be employed when removing an appliance from the Gateway/HEN.

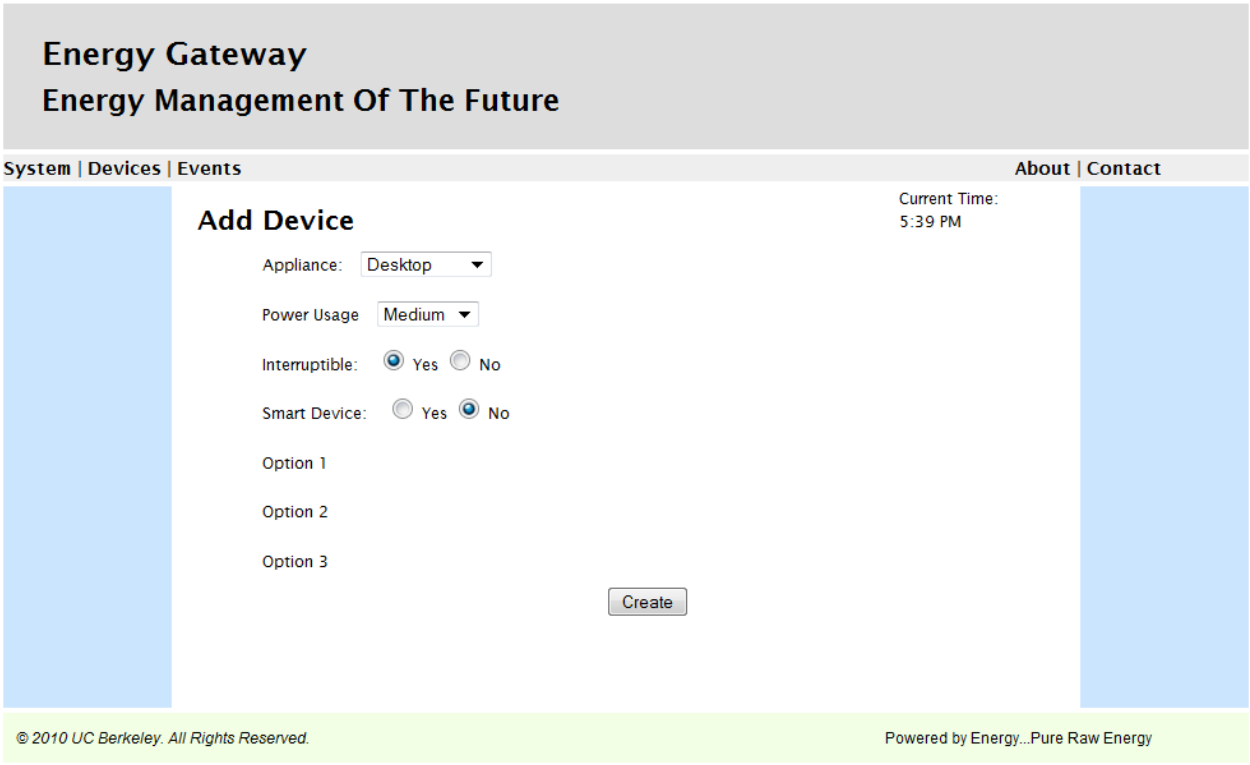

Figure 2 - Draft webpage for registering a HEN device with the REG

Individually control appliances: With the goal of automating home energy usage, the Gateway web user interface will facilitate the actuation of individual appliances at the resident's discretion. Figure 3 shows a draft webpage which would accomplish this functionality. As the page shows, the user would have the option to turn on or off individual appliances connected to the HEN/Gateway. This interaction is very simplistic and will be expanded to enable scheduling and reduced usage states (rather than just on or off).

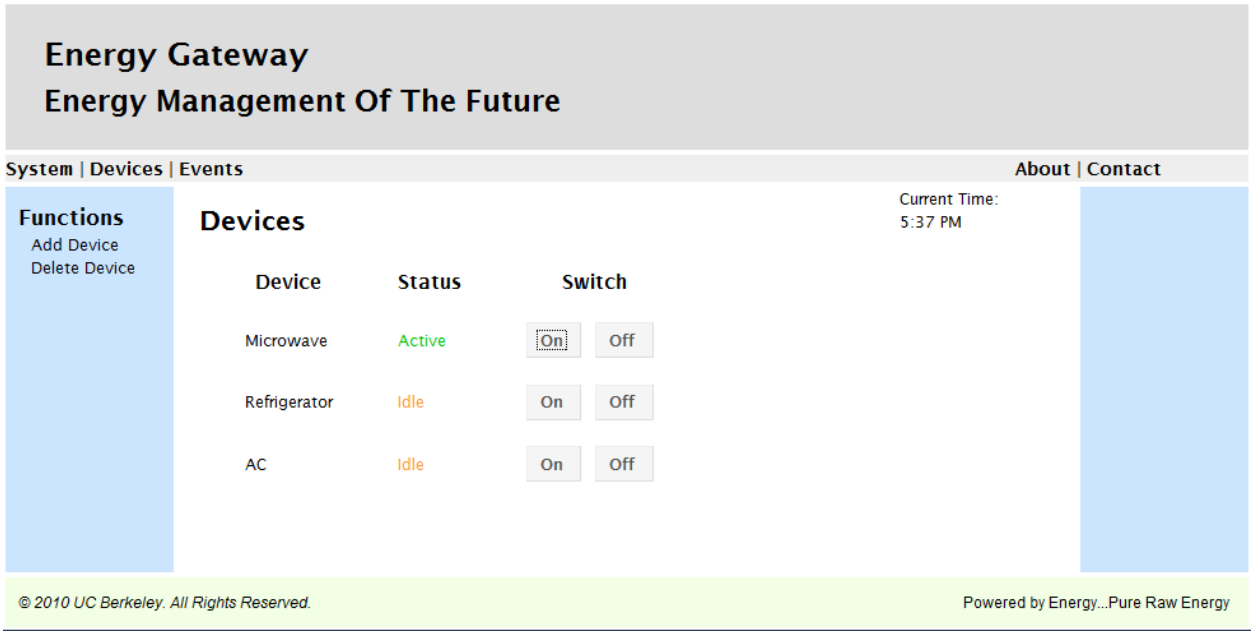

Figure 3 - Draft webpage for controlling individual appliances

Control demand response participation: The Gateway web user interface will also provide the resident with the option of whether or not to participate in a DR event. As shown in Figure 4, successive DR events will be read from an Akuakom DRAS (if so scheduled) and presented to the resident via the UI. The resident will have the option to Opt-In or Opt-Out of each individual event. We intend to include a default setting of either Opt-In or Opt-Out. The user can override the default when an event occurs, but if the user does nothing, the default will apply. Please note that that cost per kWh is not included in Akuakom DR event information received from their DRAS. It is included in the below figure for illustrative purposes only.

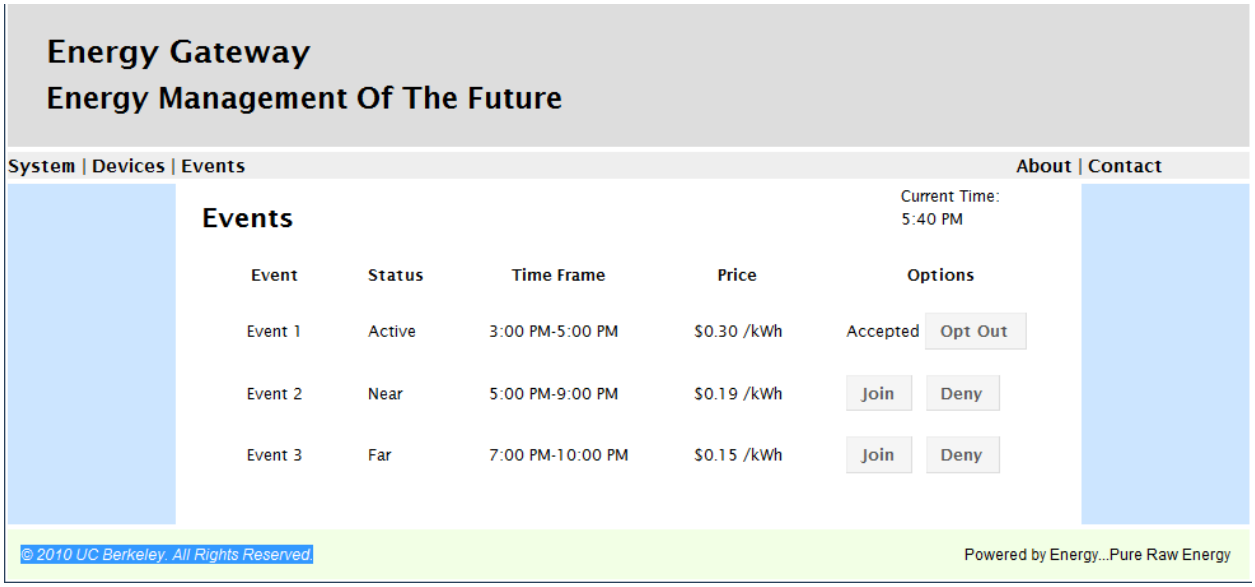

Figure 4 - Draft webpage for supervisory control of DR event participation

Provide energy usage data: The Gateway must provide functionality to communicate individual or aggregate appliance energy usage data to the resident in a clear and direct manner. This issue is largely tied to the way in which energy usage data is to be stored in the Gateway. This is a database problem, and will be addressed extensively in Phase III of the Gateway project.

Gateway External Bundles

Gateway external bundles represent external appliances within the HEN. In order for a home owner to connect an appliance with the Gateway, a bundle representing that appliance must be installed to the framework. Depending on the type of connection over which the appliance will communicate with the Gateway, the appliance bundle will need to specify different parameters specific to the appliance and the appliance's connection to the Gateway. For WiFi or Ethernet connections, which are made over JAVA network sockets, the appliance bundle will consume and implement the NetService service. It will therefore need to provide parameters which populate the *makeWiFiConnection* method (see NetService section). These parameters will include the IP address of the appliance on the LAN and a port number over which the connection will be made.

For a connection over ZigBee, the appliance bundle will need to specify parameters which populate the *makeZigBeeConnection* method of the ZigBeeService service. Among the parameters which will need to be specified, the most important of which is the ZigBee dongle ID which will identify the ZigBee dongle connected to the appliance in question over the ZigBee network. Further research is required to adapt this model to connect to ZigBee networks operating on different hardware.

The actual process of how the Gateway and appliance registration will take place will be addressed in Phase III of the Gateway project.

Gateway.OSGi.WiFi.Appliance – WiFi Appliance Bundle

As discussed in the WebUI bundle section, a user wishing to register an appliance over Wi-Fi will, upon completing the web template seen in Figure 2, automatically create a new WiFi Appliance Bundle and install it to the Gateway/OSGi framework. This WiFi Appliance bundle will be unique to the appliance which it represents and will contain all of the user entered parameters which describe the appliance. This bundle will automatically consume the NetService service to establish a WiFi connection to the Gateway. In addition, the WiFi appliance bundle will create an instance of ControlService and export that service to the OSGi framework. In order to accomplish these operations, the WiFi appliance bundle will need to import the following packages in its MANIFEST.MF file: *gateway.osgi.util.controlservice* and *gateway.osgi.util.netservice* (both packages are located in the utility bundle).

Gateway.OSGi.ZigBee.Appliance – ZigBee Appliance Bundle

For an external appliance wishing to connect over ZigBee, the same web based template of Figure 2 can be used to register an appliance over ZigBee, except that a connection over ZigBee will need to be specified (whether this is to be done by the user or the appliance/Gateway is still unclear). Once completed by the resident, a ZigBee Appliance bundle representing the appliance will be installed to the OSGi framework. This bundle will automatically consume the ZigBeeService service and create an instance of ControlService, which will be exported to the OSGi framework. To accomplish this, the MANIFEST.MF file will need to import the following packages from the utility bundle: *gateway.osgi.util.zigbeeservice* and *gateway.osgi.util.controlservice*.

Gateway Support Software

Data Logger and Gateway Database

Both the NetService and ZigBeeService services include rudimentary data logging software which records individual appliance energy usage data and the time at which that data was read. Currently, this data is written to a simple .xml file for web display and developmental purposes. It is recognized that data storage in this manner is volatile and cumbersome, but is still useful in this stage of development. Software for data logging is easily removed for both WiFi and ZigBee connections. Conducting data management in this way presents an interesting problem, as since new data is simply appended to the end of the XML file, these files representing appliances will grow in size the longer the Gateway is operating. Currently, since this project is being developed on a computer with an overabundance of memory, so this issue is less important. However, since the final Gateway is intended to be contained on a modest operating platform such as a router, the issue of data management will become increasingly important.

In addition, as discussed earlier, efficient data storage will have implications affecting the web user interface as well. The manner in which data is stored and the amount of data available

will most definitely influence any visualization options used to communicate HEN energy usage to the resident. Phase III of the Gateway project will address this issue in detail.

Further details regarding the Gateway database can be found in Appendix B of this report.

Appliance Registration

Already partially discussed in several sections, the issue of appliance registration with the Gateway is extremely important and unanswered question in the context of the Gateway/HEN. One can imagine neighborhoods in which each house has a separate Gateway; the software and registration process must ensure that appliances are not paired with non-resident Gateways. This issue is currently not addressed with the Gateway in this phase of the project. However, several possible solutions exist. A process similar to Bluetooth headset/phone pairing might be employed to match the appliance in question with the Gateway. In this manner, the appliance would have a specific code associated with it which could be entered via the Gateway web UI. Thus unique code could be used in place of the appliance IP address (for a WiFi or Ethernet connection) or the ZigBee dongleID no. (for a ZigBee connection). Further research into this process will be conducted in Phase III.

Gateway/HEN Time Synchronization

Since some elements of the HEN will undoubtedly be "dumb" appliances, these elements will have little to no concept of time. In addition, smart appliances with internet connections might have access to time, but it may not be synchronized. Rather than manage individual times for each appliance, it was decided that the Gateway should manage time for all elements within the HEN. Energy usage data received from each appliance would be time stamped on the Gateway side of the connection. Time lost due to transmission time will be ignored as, for the purposes of this project, these losses are negligible.

In order to accomplish this time synchronization, a software bundle will need to be written to allow the Gateway to fetch the correct time at a predetermined interval. This will correct for drift in the Gateway's processor.

Further details regarding time synchronization can be found in Appendix A.

RxTxcomm jar

The Telegesis ZigBee dongle with the appropriate firmware is essentially an RS-232 to USB bridge, which allows any terminal software to access USB dongle using a simple "AT" command style interface. In the context of a Windows based OS using developmental software written in JAVA, one would need to have read/write capability with Windows COM ports to establish communications over the ZigBee network. However, for JAVA in Windows there is no software suite which allows read/write capability to COM ports. The JAVA software package Javax.COM is only suitable for Linux and Unix environments. Fortunately, the package RxTxcomm was available which, with some modification to the Windows Java runtime, allows

for read/write access to COM ports in Windows. This section briefly details the configuration process for RxTxcomm jar in Windows.

The RxTxcomm package can be found at the following website (Jarvi, 2006):

http://www.rxtx.org/

Care must be taken to ensure the proper package is downloaded, as binary builds for x86 and x64 versions exist. Once downloaded and unzipped, the package contains 2 files which are required: *RxTxcomm.jar*, and *rxtxSerial.dll*

The following instructions are copied from the RxTx website verbatim:

Choose your binary build - x64 or x86 (based on which version of
the JVM you are installing to)

NOTE: You MUST match your architecture. You can't install the i386
version on a 64-bit version of the JDK and vice-versa.

For a JDK installation:

Copy RXTXcomm.jar ---> <JAVA_HOME>\jre\lib\ext
Copy rxtxSerial.dll ---> <JAVA_HOME>\jre\bin

In addition to following the instructions above, one must also edit the build properties in your IDE of choice in which the ZigBee bundle is being developed. In Eclipse, it was necessary to alter the build path of the bundle thereby placing RxTxcomm.jar *ahead* of the JRE system library. Similar configuration is most likely required for alternate IDEs.

If development of the Gateway is to be used in Linux, then RxTx.comm is not required. The Javax.comm package will suffice. All methods and classes will remain the same when switching between software packages, requiring changing the imported package in the JAVA file only (from RxTx.comm to Javax.comm).

Future Work

As previously discussed, much of the work for Phase III will be focused on addressing the issues of the Gateway database, the Web UI, and appliance registration. However, in addition, the Gateway team seeks to address the following issues. During Phase III the overall design philosophy will remain dedicated to substituting external simplicity for internal complexity, or increasing the functionality of the REG while providing a simple and transparent user interface to the consumer.

Extension of OpenADR into the residential sector

Open-ADR, a communications data model focused on demand response implementation, has already been designed and implemented in the commercial sector. However, with the

development of the Gateway in the residential space, it would be beneficial to understand how the current standard can be applied to energy management processes within the home. On first glance, there are two potential models for the extension of OpenADR into the residential space:

i. **OpenADR stops at the Gateway -** The Gateway interprets/parses OpenADR information from a DRAS and communicates with then individual elements of the HEN based on an internal data model. This model implies that the Gateway would provide centralized control action for HEN elements. It would be performing control calculations and relaying actuation signals to the HEN. This model has the advantage of allowing for centralized entry of user preferences and actions based on those preferences.

ii. **OpenADR flows through the Gateway -** The Gateway acts as a conduit or bridge between OpenADR and individual HEN elements. In this model, each individual HEN element would respond independently to the OpenADR signals. This implies that each node of the HEN would have to include enough intelligence to execute a local control strategy based on the OpenADR signal.

Both strategies necessitate an evaluation of bandwidth and processing power required to accomplish each strategy.

Refinement of control strategies for load actuation within the home

The REG's potential to actuate electrical loads within the home provides a new avenue to investigate control strategies taking into account demand response criteria, consumer preferences and both local and large grid reliability. These concepts are not even clearly defined from the individual home perspective! Research into the modeling and interaction of these constraints would lead to the formulation of different optimization/control strategies (one could consider them as use-cases) which have large potential to increase energy savings, provide utility for the consumer, and positively affect grid reliability.

This effort could begin with updating Bill Burke's JRonSim home energy simulation, to include loads such as PHEVs and local generation. This model would then more accurately capture loads which are expected to increasingly become present in homes throughout California.

Analysis of the effect of Gateways from the utility/ISO perspective (Systemic Gateways)

The availability of information on the level of individual homes could give the utility/ISO extremely granular information about grid reliability and geography vs. energy consumption. From a bidding perspective, neighborhoods equipped with REGs in their homes could bid resources horizontally between other neighborhoods or aggregators, rather than vertically (up to the utility level). From a grid reliability perspective, the systemic availability of Gateways could have interesting robust control applications aimed at maintaining grid reliability.

Structuring of a multi-corporate environment

The potential for the Gateway to allow consumers to bid their resource to utilities, aggregators and other consumers creates a multi-corporate environment the interactions of which are not clearly defined. For example, the consumer could buy power from the utility, but would deal with an aggregator to sell their services. In order to accomplish such interactions, the utility/ISO would need to define a common point of coupling (CPC) about which a zero sum gain can be calculated. This point would provide a baseline to both the utility and the consumer to directly understand their energy savings/consumption in the presence of multi-Gateway interactions (prevents double counting, etc.).

Were the utility to define such a CPC for Gateway interactions, neighborhoods equipped with Gateways and some local generation could be, in effect, a microgrid from the perspective of the utility. The CPC would then be the connection point for the utility/microgrid. This concept would require further analysis of microgrids, as important factors such as microgrid frequency and power factor would need to be re-synchronized with the macrogrid prior to microgrid reconnection at the CPC.

How to structure grid constraints

The availability of Gateways within the home will empower the consumer to alter their energy consumption habits. This raises interesting issues of fairness on load at multiple levels within the grid. On the local level, consumers wanting to charge their PHEVs at the same time at night could overload the transformer serving their neighborhood. Some investigation into how the local grid capacity is managed between Gateways should be performed.

Management of local generation (PV, diesel, UPS), within home and from the systems level

It would only take a modest breakthrough in solar PV technology for the widespread adoption of this technology on the rooftops of California homes. Such large amounts of distributed generation would definitely raise problems for the grid, as large amounts of local power generation would be synchronized across the state, but also, to some extent, random (based on the weather, shading, etc.). If large scale local PV is adopted, can the grid accommodate many homes trying to sell their resource to the utility? Would this overload transformers or other grid components? The Gateway, which would act as the manager for residential rooftop PV, would be uniquely placed as an actuator to regulate this potential problem. This also raises issues discussed in item 5, as the Gateway must ensure that everyone with a rooftop PV is given a fair opportunity to sell their resource back to the grid.

Applications of home energy management for implicit and explicit thermal storage

As the manager of residential energy, the REG could allow for the storage of energy for implicit or explicit residential thermal storage. For implicit thermal storage, the Gateway could store energy in the form of building pre-cool, using energy purchased at off-peak times. Following a similar logic for explicit thermal storage, the Gateway could employ a different storage media

(such as the freezing of water into ice) using energy delivered from PV or purchased from the grid at incentivized times.

This effort should begin with a literature study into the different types of thermal storage available, followed by their characterization/simulation in differential equation form. Following this, control strategies can be developed to minimize energy usage using the characteristics of the type of thermal storage available.

Consumption vs. Net Metering

For consumers interested in buying and selling energy through their Gateway, the concept of net metering vs. consumption metering must be addressed. Net metering refers to zero sum metering, where the price of electricity at a certain time is not taken into account. However, consumption metering refers to metering in the context of variable electricity prices. This difference can have enormous implications on the consumer cost of electricity. For example, in a residence with local generation, generating electricity at peak times would be worth more than if done so at off peak times. The gateway must account for the value of electricity consumed/generated in addition to amount of consumption/generation. Perhaps it would be more beneficial for the gateway to record historical consumption metering for each residence and re-compute a consumer's energy cost/profit? Another potential solution is for the gateway keep a running value of cost/profit based on cumulative instantaneous consumption metering. The main tradeoff between these two cases is the recording and availability of data for control purposes. Recording consumption metering could prove difficult given the desired low cost of the gateway, however, not having this historical data available for control actions could make the design of such controllers more difficult.

Conclusions

This report provided an overview of the work performed on the Residential Energy Gateway Reference Design over Phase II (Jan. 1, 2010 – Sept. 30, 2010). This report was meant to illustrate the Gateway software structure in the context of the OSGi bundle system for JAVA. A brief introduction and explanation of OSGi was provided, followed by Gateway specific bundles and services and their interaction. The functions of the Gateway were described in terms of OSGi services exported and consumed by bundles in the OSGi framework. In addition, the Gateway user interface concept was extensively discussed, as well as important software issues which need to be addressed in Phase III.

References

Akuakom. (2009). *OpenADR Client Development Program Users Guide.* San Rafael: Akuakom.

Jarvi, K. (2006). *RXTX Home Page*. Retrieved October 1, 2010, from RXTX: http://www.rxtx.org/

OSGi Alliance. (2010). *OSGi Home Page*. Retrieved October 01, 2010, from Open Services Gateway initiative (OSGi): http://www.osgi.org/Main/HomePage

OSGi benefits. (2009). *Benefits of Using OSGi*. Retrieved August 24, 2009, from OSGi: http://www.osgi.org/About/WhyOSGi

OSGi Bundle Interface. (2010). *Bundle Interface*. Retrieved October 1, 2010, from Bundle (OSGi Service Platform Version 4 Release 4.2): http://www.osgi.org/javadoc/r4v42/org/osgi/framework/Bundle.html

OSGi ServiceFactory. (2010). *OSGi ServiceFactory Interface*. Retrieved October 1, 2010, from ServiceFactory (OSGi Service Platform Release 4 Version 4.2): http://www.osgi.org/javadoc/r4v42/org/osgi/framework/ServiceFactory.html

OSGi ServiceListener. (2010). *ServiceListener Interface*. Retrieved 10 1, 2010, from ServiceListener (OSGi Service Platform Release 4 Version 4.2): http://www.osgi.org/javadoc/r4v42/org/osgi/framework/ServiceListener.html

Telegesis. (2009). *ETRX2USB USB Stick Product Manual.* Retrieved October 1, 2010, from Telegesis Product Page: http://www.telegesis.com/downloads/general/TG-ETRX2USB-PM-004-105.pdf

Appendix A – Time Synchronization by Michael Sankur

Time Synchronization

Time synchronization is an important to distributed networks with many different devices communicating, such as the residential gateway or commercial building. Correct time information coming from both wired and wireless devices and smart meters is vital to ensuring that energy will be used wisely. Currently the allowable difference in time between various devices and the controller is unknown. However a good estimate is a minute since the operation time scale is a daily cycle.

There are several free and widely available sources of time information

Source	Media	Pro	Con
RDS	Radio Waves		
NTP	Dedicated Time Servers		
IEEE 802.11	2.4 GHz wireless		

Time Synchronization for Simulation

The residential and commercial gateway simulations will also need a method of time synchronization. This is due to the fact that the simulation will be run in accelerated time and that several laptop computers will be simulating appliances and devices. Each laptop has its own computer clock and its own run speed, so a synchronization algorithm is needed.

To encounter the problem of differences in clock times and simulation run speeds, one proposed solution is to have a central computer act as a source of time information much like a virtual or synthetic NPT/RDS, similar to a Primary Reference Clock (PRC). This computer can have its own time updated by NPT, and run a program to broadcast time information on various communication channels such as Wi-Fi, Zigbee, LAN and other wireless signals. The time keeping and broadcast program should also use feedback to ensure it is broadcasting the appropriate number of accelerated time signals in a given real time span as well as ensure these are regularly spaced if needed. This broadcast time information will then be received by the laptops that are simulating appliances and used to update their simulation "times".

There are two main methods to deal with time discrepancies over a network. One is asynchronous, called "hand-shaking", in which one device tells the other to send a signal at a certain time in its clock. The other is synchronous in which one device is a master's time is used by other slave computers. Both methods are being investigated for the simulation.

To have a synchronous system, a time synchronization algorithm is needed. There are two main algorithms; Christian's Algorithmn and the Berkeley Algorithm. The Christian Algorithm is described by the following steps:

1. P requests the time from S
2. After receiving the request from P, S prepares a response and appends the time T from its own clock.
3. P then sets its time to be T + RTT/2

The Berkeley Algorithm is more complex and is described by the following steps:

1. A *master* is chosen via an election process such as Chang and Roberts algorithm.
2. The *master* polls the *slaves* who reply with their time in a similar way to Cristian's algorithm.
3. The *master* observes the round-trip time (RTT) of the messages and estimates the time of each *slave* and its own.
4. The *master* then averages the clock times, ignoring any values it receives far outside the values of the others.
5. Instead of sending the updated current time back to the other process, the *master* then sends out the amount (positive or negative) that each *slave* must adjust its clock. This avoids further uncertainty due to RTT at the *slave* processes.

For the purposes of the simulation, the Christian Algorithm may be better suited as it is less complex, and extreme accuracy is not needed. On this note, an even simpler method is proposed in that the broadcasting reference clock computer does not receive requests. The computer whose clocks are to be synced receive the time signals and reset their clocks using an estimated computation and transmission time.

Time Synchronization References

http://en.wikipedia.org/wiki/Time_synchronization
http://en.wikipedia.org/wiki/Network_Time_Protocol
http://en.wikipedia.org/wiki/Clock_synchronization
http://en.wikipedia.org/wiki/Synchronization_in_telecommunications
http://en.wikipedia.org/wiki/Distributed_computing
http://en.wikipedia.org/wiki/Cristian%27s_algorithm
http://en.wikipedia.org/wiki/Berkeley_Algorithm
http://en.wikipedia.org/wiki/Lamport_timestamps
http://en.wikipedia.org/wiki/Handshaking

Appendix B – Web User Interface and Database Issues by Kevin Ding

Gateway Web Interface

To host the gateway website, a dedicated Apache web server must be installed within the gateway along with PHP compatibility and a light-weight database for storage. Usually, each of these web server components would have to be installed separately but for simplicity and ease of testing, all these critical components can be obtained through the XAMPP application (http://www.apachefriends.org/en/xampp-windows.html) which will be implemented for our preliminary web hosting. XAMPP is an easy Apache installer that contains PHP along with other miscellaneous web tools. Once installed, all website components should be contained within the "xampp/htdocs/" directory and specifically within the "Gateway" file.

Connecting to the web server is fairly simple. First, the server as well as all clients should be linked on a LAN server (preferably wireless for gateway simulation), with all clients having access privileges to the server. Once connected, it becomes a matter of finding the IP address of the machine that the server is on, opening up the webpage and typing in "http://IPADDRESS/Gateway/setup.php" (where IPADDRESS = IP address of the server).

Traversing the webpage is fairly simple and self-explanatory with the top menu bar containing most of the important information about the gateway system such as a list of devices on the network and a schedule of DR events. Within these self-sustaining tabs are other functions that will be of great use such as adding and deleting devices, the option to opt in and opt out of DR events and examining energy data values for each appliance. All of this data will be contained within the host database for storage reliability and ease of access.

Database Selection

In the course of choosing a database, there were a few criteria that were examined among the countless database options that were available. Simplicity, ease of installation, and access were some of the core defining characteristics that stood out when considering compatibility with the gateway. The following chart notes the positive and negative attributes of some of the most popular and well-documented databases on the market as well as those that are open source. A clear distinction should be made between enterprise databases such as Oracle versus embedded databases such as SQLite in that embedded databases offer much less services in the form of user-friendly GUI, APIs and professional support.

Database	Pros	Cons
MySQL	1. Server/client database (Multi-user) 2. Large set of functions and support 3. Compatible with C, C++, Java 4. Available front end GUIs 5. Good for handling large data sets 6. Can users and permissions 7. Open source	1. Difficult installation process than embedded databases (specifically SQLite) 2. Does not contain triggers 3. Can't write to views 4. No stored procedures

SQLite	1. Ease of installation 2. Zero-configuration 3. Single-file 4. Small memory footprint 5. Concurrent read access 6. Embedded 7. Open source 8. Good for rapid development (temporary testing)	1. Partial support for triggers 2. Can't write to views 3. Single-write access 4. Loss of performance features compared to MySQL
Empress Embedded Database	1. Acts as local server for single threaded client 2. Zero-configuration 3. Triggers supported 4. Stored procedures available 5. Embedded	1. Does not support multi-thread access 2. Not open source (owned by Empress Software Inc.)
Berkeley DB	1. Multi process, multi thread 2. Multi-version concurrency control (MVCC) 3. Scalable to terabytes of data 4. Embedded	1. Not open source (owned by Oracle)

For the purposes of this secondary phase of the gateway project, it was deemed necessary to choose a simple database that was dedicated and solely private to the gateway software. Among the ones that were analyzed, it came down to a close match up between MySQL and its much lighter but restricted version SQLite. Though SQLite is very big on simplicity in providing rapid development and ease of installation and use, many of the performance enhancements available in MySQL are not present in SQLite. As the project increases in scope, data storage becomes increasingly complex and the need for performance becomes apparent in the gateway project the need to transition from SQLite to a much more comprehensive database might be cumbersome. To prevent such an unnecessary transition it would be much better to start with a complete database. Therefore, from the analysis presented above the database of choice for the gateway software would be MYSQL.