BRADLEY University

Laboratory Notebook

BEMOSS and Its Enhanced Applications

Brian Lauer

blauer@mail.bradley.edu

Beginning March 13, 2018

Contents

Contents

Contents

Monday, May 06, 2019

I emailed Mr. Mattus asking him whether he made any progress on finding a laptop that can be used to demonstrate the installation of BEMOSS.

Thursday, May 09, 2019

I picked up a department laptop from Mr. Mattus today. He cleared the partition completely, so no time had to be spent removing a previous operating system from the machine. Then, I installed Ubuntu 16.04.6 LTS on the system with a bootable USB flash from the link provided on the BEMOSS installation guide.

Wednesday, May 22, 2019

As recommended by Dr. Miah, I worked on running BEMOSS on the previous team's laptop. By running `./startBEMOSS_GUI.sh` inside the directory `/home/bemoss/BEMOSS3.5/GUI`, I was able to start up the BEMOSS Launcher Wizard. By selecting Run BEMOSS in the TKinter GUI the server was started, and I was able to connect to the local web server at `localhost:8082.` At this point, the software was able to detect the WeMo Insight Switch and thus control the Philips Hue bulb connected to it. I was not able to control the motor due to time constraints, but I will do so soon.

I did some more research on the features that BEMOSS offers at https://github.com/bemoss/BEMOSS3.5/wiki/BEMOSS-Features including the ability to provide local and remote monitoring and security.

Thursday, May 23, 2019

Friday, May 24, 2019

The hierarchy of BEMOSS was researched today with the motivation of understanding the software better in the Developer Resources. The first layer consists of the UI and User Management which reside in the central server. Here admins can manage different nodes on the network and either deny or accept requests from users. Layer 2 is the BEMOSS Application and Data Management Layer which allows developers to create custom applications for target devices that can be added to the UI. It may be interesting to explore this feature once I have gotten a better idea of what sort of original contributions I would like to make to the project. A motivation for this part of BEMOSS is to integrate web services like IFTTT (IF This Then That) which may be interesting to use. The third layer is the operating system and framework layer consisting of the agent platform VOLTTRON developed by the Pacific Northwest National Laboratory. Six different agents perform various different tasks such as detecting new devices (lighting/plug load controllers) on the network and monitoring them to ensure they are running properly. Layer 4 is the BEMOSS connectivity layer that handles the communication between the operating system layer and the physical hardware devices. This is where support is extended to different communication technologies like Wi-Fi, Ethernet, and Serial(RS-485). Each device supported by BEMOSS has an API translator needed to handle the differences in device attributes.

Monday, May 27, 2019

No work was done due to Memorial Day.

Tuesday, May 28, 2019

The outline for the presentation on May 31 follows:

- Introduction

- Applications of BEMOSS

- Hardware/software needed to install BEMOSS

- Immediate future work

The following questions are answered to determine what needs to be added to the presentation:

- What is BEMOSS? BEMOSS or Building Energy Management Open Source Software is an agent-based software platform engineered to allow small- and medium-sized commercial buildings to more seamlessly integrate equipment designed for sensing and control. This software can allow building owners and engineers to manage building energy use better by monitoring different load control devices such as lighting loads, plug loads, and HVAC controllers.

- How can BEMOSS be applied to the real world?

- What hardware and software is needed to install BEMOSS?

- What kind of future work is available to be implemented with this software?

After working on the presentation, I worked in the lab to document the toggling of the WeMo insight switch with BEMOSS. Most of this information was gathered from https://github.com/bemoss/BEMOSS3.5/wiki/User-Guide-for-BEMOSS-UI. Once the BEMOSS server has started, type localhost:8082 in the web browser to go to the BEMOSS Web UI. The username is 'admin' and the password is the one set during installation. To discover the switch, click the "Discover New Devices" tab in the left navigation bar. Under the "All Plug Load Controllers" menu, select either "All Plug Load Controllers" or "Belkin International Inc. Insight." Click "Discover Selected Devices" to complete the process. The number of discovered devices will appear on the Discover New Devices tab. If only one device has been discovered, a 1 will appear next to the name Discover/Manage. On the Discover/Manage page, approve the device by setting the approval status to "Approved" then select "Save Changes to Plugload Controllers." Navigate to the tab NODE1. Select "View All" under "Plugload", then select the WeMo smart plug icon to change the status of the plug and view the power consumption.

Wednesday, May 29, 2019

I spent the first few hours of the day reading through [4]. I worked on researching the applications of BEMOSS and the introduction for the May 31st presentation. The future work still needs to be researched and added. Also, I would like to add some pictures to the slides to help the audience members gain a better visual understanding. Ideally, the presentation should be finished tomorrow morning, so I can have more time to practice. I practiced the presentation at the end of the day today without everything completed which isolated my knowledge gaps and gave me a better idea of what I should work on. To provide better flow between slides I must find ways to transition well between them.

Thursday, May 30, 2019

More work was done on the presentation.

Friday, May 31, 2019

Work on presentation and meeting with other members of the Robotics and Mechatronics (RAM) group.

Monday, June 3, 2019

I read through [5] and [2] to obtain more ideas on original contributions I can make to the project. One thing I found in [2] is the use of an induction motor in McNeese State University's microgrid. With a variable frequency drive, this could be integrated with BEMOSS to control different types of industrial loads. One problem is the high price tag on both. In [5], a Particle Photon board was used to control the brightness of fluorescent lighting via step-dim ballasts. The Raspberry Pi is definitely a better option here than the Photon board as the previous senior project group used an RPi in their project.

Tuesday, June 4, 2019

Today, I attempted to install BEMOSS on my Ubuntu laptop. After running `./startBEMOSSGUI_.sh` in the GUI directory, I encountered some problems. The following errors were thrown:

```
Traceback (most recent call last):
    File "Web_Server/run/defaultDB.py", line 91, in <module>
        admin = User.objects.get(username='admin')
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/django/db/models/manager.py",
    line 85, in manager_method
        return getattr(self.get_queryset(), name)(*args, **kwargs)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/
    python2.7/site-packages/django/db/models/query.py",
    line 379, in get
        self.model._meta.object._name
django.contrib.auth.models.DoesNotExist: User matching query does not exist.
OS settings imported

Traceback (most recent call last):
    File "bemoss_lib/databases/cassandraAPI/initialize.py",line 186 in
<module>
        init()
    File "bemoss_lib/databases/cassandraAPI/initialize.py", line 99, in init
        casYamlFile = open(settings.PROJECT_DIR+"/cassandra/conf/cassandra.yaml",'r')
IOError: [Errno 2] No such file or directory:
'/home/ramgroup/BEMOSS3.5/cassandra/conf/cassandra.yaml'
```

I was able to eliminate the first error by using 'admin' for the Django superuser rather than the default 'ramgroup'. I tried to eliminate the second error by deleting BEMOSS3.5 from my home directory and recloning; however, the same problem persisted. I found a directory named `/BEMOSS3.5/cassandra` on the previous group's laptop which is not being created when I run BEMOSS on my machine. This directory contains the file "cassandra.yaml" which the file "initialize.py" is attempting to access. This leads me to believing that there is some issue with the creation of the directory.

Monday, June 10, 2019

More work was done to install BEMOSS. I emailed one of the members of the previous senior project named Bob about the error.

Tuesday, June 11, 2019

After looking through some of the files in `/BEMOSS3.5/GUI/GUI.py`, I found the line of code preventing the installation of BEMOSS which is 108:
`bemoss_is_installed = os.path.isdir(ui_path) and os.path.isdir(cassandra_path) and os.path.isdir(env_path)`
Since the cassandra directory is non-existent, the expression `os.path.isdir(cassandra_path)` evaluates as `False`.

After some further searching I found that the BEMOSS is failing to download and install the cassandra database due to a dead link in the shell script
`/BEMOSS3.5/GUI/bemoss_install_v3.5.sh`
When line 48:
`wget http://downloads.datastax.com/community/dsc-cassandra-3.0.9-bin.tar.gz`
is run, a "404 not found" error is generated by the server. The URL was entered into a web browser and it was found that the requested URL was not found on the server.

As a possible solution, I used a different URL to download the cassandra database in
`/BEMOSS3.5/GUI/bemoss_install_v3.5.sh`
After adding a comment on line 48, the previously mentioned URL on line 49 is changed to
`https://archive.apache.org/dist/cassandra/3.0.9/`
`apache-cassandra-3.0.9-bin.tar.gz`
Lines 49-53 were changed to
`wget https://archive.apache.org/dist/cassandra/3.0.9/`
`apache-cassandra-3.0.9-bin.tar.gz`
`tar -xzf apache-cassandra-3.0.9-bin.tar.gz`
`sudo rm apache-cassandra-3.0.9-bin.tar.gz`
`sudo rm -rf cassandra/`
`sudo mv apache-cassandra-3.0.9 cassandra`
After this, BEMOSS was successfully installed. The post-installation instructions were followed on the BEMOSS wiki, but, at the end, errors were still being thrown while attempting to get the web server up and running.

Wednesday, June 12, 2019

After viewing the issue on the BEMOSS repo: `https://github.com/bemoss/BEMOSS3.5/issues/47`, I found that the IP address in `parent_ip.txt` did not match the IP of my system, so I changed this to the correct IP. This corrected the problem and the BEMOSS web server was able to boot successfully. Note this text file is only created after BEMOSS is run.

Thursday, June 13, 2019

In the lab, I worked on getting BEMOSS up and running. I mistakenly used the wired connection at first when attempting to run the BEMOSS server but decided to connect to the wireless network ECE-Robotics1 as the Raspberry Pi controlling the motor uses this network. I was able to login as admin into BEMOSS but experienced a problem when attempting to connect to the WeMo Insight switch. When I attempt to navigate to the plug load page to control the WeMo switch the page does not load. It is unclear whether this is an issue with the Insight switch or with BEMOSS itself. I also tried working with the WeMo plug on the previous group's Ubuntu laptop but ran into the same issue leading me to believe it is possibly an issue with the WeMo switch.

Friday, June 14, 2019

More work was done at the beginning of the day to help identify and fix the problem of the Plugload page not loading. After one attempt the page eventually loaded but took a great deal of time. It was finally discovered that the laptop must be connected to the wired network as well as ECE-Robotics1 in order to function properly. Without a wired connection, the PC is unable to connect to the Internet which causes errors. However, although the software was working properly errors were reported by the TSDagent. These are captured in the figure below.

I decided to try the same setup on the previous group's machine to see if I would receive the same problem. After running the software on the previous group's machine, I received the same errors with the TSDagent thus concluding that the proposed solution mentioned on page 31 is not a complete one. I will need to email Ashraf with the details on this.

In the meantime, I will attempt to get the DC motor running with the BEMOSS software. After modifying the file permissions of three of `shell_control.sh` using
`chmod u+x shell_control.sh`
I was able to identify and control the motor using `pyshell_control.py` which rotates the motor counter clockwise then clockwise. Soon I need to start creating the GUI that will show all devices on the network and enable the ability to control them.

Monday, June 17, 2019

Work on the presentation slides for June 21 was done. Further research on the Beamer class was conducted to add more detail to the presentation.

Tuesday, June 18, 2019

The presentation slides are almost complete at this point. A few more captions need to be added to the figures and sources must be added to the bibliography.

Wednesday, June 19, 2019

Work on the presentation was continued and uploaded to github. A few additions may need to be made as I was not able to get a full 10 minutes out of it. Tonight, the presentation will be practiced and polished, so that I am ready to go by Friday. This time I need to know exactly what I am saying before going in so I can avoid any pauses.

A page on wikipedia on computer networks was read:
`https://en.wikipedia.org/wiki/Computer_network`
to better understand what is going on with this project. Knowledge of the python Tk interface must be obtained to build the GUI due June 21.

Thursday, June 20, 2019

Had a short meeting with Dr. Miah in the lab. Here is what needs to be done:

- Add progress and plan to presentation.

- IoT discovery and control GUI and BEMOSS plugload icon must be implemented before June 28. Conference paper must be completed and submitted before June 28. However, this is not likely to be finished by then as little to no progress has been made on the GUI or BEMOSS motor integration.

- Need to start recording hours when working with the DC motor so I can get paid. I need to talk with Mrs. Polen to get an account setup with Bradley.

- I need to start thinking of a device to implement in BEMOSS. Otherwise, this project will not be successful without an original contribution. Thus, a lot of research must be done.

- Agent-based architecture will need to be researched by reading some research papers.

- Need to send Ashraf an email asking if he has made any progress on the project.

  All the scripts written by Reece and Bob to control the motor were understood except for `XBEETEST.py` on the Raspberry Pi as I still need to do research on the XBee modules if I am to use them in the project.

Friday, June 21, 2019

To better understand how to create the GUI to control the devices in the lab, I read up on the documentation for PyGTK at `https://python-gtk-3-tutorial.readthedocs.io/en/latest/layout.html#`.

Sunday, June 23, 2019

Work was done on researching a device to integrate within BEMOSS. Here are some possible ideas:

- Ultrasonic Range Finding module - hcsr04
  I have one of these and have been programming it some and could potentially have an interesting application for IoT. However, this does seem rather simple and would not likely take long to fully implement.

- Digital Multimeter

- Accelerometer

- Gyroscope

- Dust Sensor `https://www.waveshare.com/dust-sensor.htm`

- PM2.5 Particle Sensor `https://www.cytron.io/p-honeywell-pm2.5-particle-sensor-module`
  Looking at this module it uses a two wire UART output so it would likely be very easy to interface with the Raspberry Pi

Monday, June 24, 2019

While building the GUI, I came to realize that Gtk is simply too complex for me, so I decided to change tkinter which is a bit simpler. At this point I have finished the GUI and simply need to connect the callbacks to the events using tk.widget.bind(event,callback). Once this has been done I will be ready to move on to implementing the logic to control the wemo switch and motor within the tkinter application.

Tuesday, June 25, 2019

To better understand lower level networking concepts, I will use the python module `socket` to ping addresses on the network and resolve their hostnames. It may save time to use the `nmap` command used by the previous group; however, I would like to build a system from scratch completely in python. As the motor needs to be implemented within BEMOSS as soon as possible, I will work on this first and determine how to use the wemo switch later. I will base some of my work off [?]. However, after working for some time I found that the program I was attempting to write was rather inefficient and using the nmap command will be much more faster. Thus I have decided to use the scripts written by Bob and Reece. I was able to write a single python script using the socket module to parse through all hosts on the network 'ECE-Robotics1' and place them in a list along with their respective IP addresses.

Wednesday, June 26, 2019

I was able to succesfully add the Raspberry Pi and Wemo switch names to the listbox; however if the button 'Discover IoT Devices' is pressed continually, devices will continually be added to the list. Right now I need to determine how to initialize the Raspberry Pi by sshing into it. Then, after selecting the toogle button, I must figure out how to remotely send commands to the device to turn it on or off without having to reconnect. After some tinkering I found that I can simply just ssh into the pi and run a script that simply turns the moltor on when the button is toggled on and off when the button is toggled off. Thus no initializatin will be needed.

I need to write scripts to perform the following operations:

- Use nmap to the scan wifi credentials

- Place the credentials into text file

- Scan for the IP addresses and place them into a text file

- Assign the address read from the file to a variable and use this to remotely login to the device (for the RPi)

However, I have almost no knowledge of bash so this will take some researching. I found a way of storing the first IP address in `IPAddresses.txt` in a variable that can be used to call the python scripts running on the pi to control the motor. Each time the toggle button is pressed, the text file is read from which is inefficient. More ideally I would like to implement some feature where the IP addresses of the devices are stored in a place that can be accessible to any shell script within the directory. I would like to do this later on; however, I need to move on to working the Wemo switch's API into this application.

To perform the implementation of the WeMo switch, I will need to read through the documentation provided by the BEMOSS team and some of the python code to find exactly what code needs to be written to create a fully functional system. This code is very complicated thus it may take a great deal of time to work through.

I found a url on the bemoss website: `www.bemoss.org/api-interface-wemo-smart-plug/` that explains how some of the code works. The switch uses the upnp (Universal Plug and Play) protocol.

Thursday, June 27, 2019

A github gist was located that contains a script to control a wemo device: https://gist.
github.com/pruppert/af7d38cb7b7ca75584ef. I was able to successfully control the
wemo switch with this. This will be helpful in understanding the code provided in the
bemoss repo and on the bemoss website. The url to send commands to the switch is
`http://192.168.1.112:49153/upnp/control/basicevent1`
I attempted to add this url into the method `getDeviceStatus`; however when I run the
code I receive a 500 response (internal server error). After running an nmap scan with
the port scan enabled I found that the wemo switch uses both ports 53 and 49153 for
communication, so the cause of the problem here is unknown. Later, I determined the
problem is that no XML was being sent in the body of the POST request to the device. I
simply copied the XML from the python script I found online to control the switch and
modified the code from the bemoss website. This is the function used to turn on the
Insight switch.

```
def turnOnSwitch():
    header = {
        'Content-Type': 'text/xml; charset="utf-8"',
        'SOAPACTION': '"urn:Belkin:service:basicevent:1#SetBinaryState"'
    }
    body='<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body><u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState>
</u:SetBinaryState></s:Body></s:Envelope>'
    controlUrl='http://192.168.1.112:49153/upnp/control/basicevent1'
    response = requests.post(controlUrl, body, headers=header)
    del response
```

To understand how this code works more research will need to be done as I know very
little about XML.

Friday, June 28, 2019

Layer 1 (UI layer) of the BEMOSS hierarchy was researched for a few hours to try and understand how new devices are to be added to BEMOSS. Details such as how the Model-Message-View-Template works and the UI project structure are provided. However, this doesn't really provide any details on what happens when an element in the UI is selected such as pressing a button to toggle a device on and how the message flows down to the device. In other words, I'm not able to understand the chain between the UI layer and the device as this is not documented on the BEMOSS wiki. A single file I found that could be possibly helpful is `/BEMOSS3.5/Web_Server/webapps/device/templates/plugload/plugload.html`. This will require some digging to understand the html, css, and jQuery as I have little to no experience in any of these. In addition, I may need to research the agent based system to understand how these agents interact with the devices' APIs.

Monday, July 1, 2019

To understand how the device discovery agent works, I studied each line of
`BEMOSS3.5/Agents/DeviceDiscoveryAgent/devicediscovery/agent.py`
carefully. In `BEMOSS3.5/BEMOSS_lib/db_helper.py` a class named db_connection is de-
fined with method database_connect that reads the system's ip address from `parent_ip.txt`
and passes it as an argument into the method psycopg2.connect in order to connect to
the PostgreSQL database named 'bemossdb'. Other keyword arguments that must be
passed include port number, database name, user name, and database password. The
full method call is

```
con = psycopg2.connect(host='136.176.122.127',port='5432',database='bemossdb', user='admin',pa
```

Tuesday, July 2, 2019

Today, I copied the file `API_WeMoPlugload.py` and renamed the copy to `API_rpi.py` in the directory `BEMOSS3.5/DeviceAPI`. However, this did not result in a new device named RPI appearing in the "Discover New Devices" page on the BEMOSS dashboard. Thus, I must do further research to determine where these names are added to the dashboard. After using the command `grep -r 'All Plug Load Controllers' .` in the directory ~/`BEMOSS3.5` I was able to determine which html file places the vendor name and device model on the "Discover New Devices" page. This file is
`/BEMOSS3.5/Web_Server/webapps/`
`discovery/templates/discovery/manual_discovery.html`
After changing the values of `'vendor_name'` and `'api_name'`in the dictionary returned by the method `API_info`, the rpi was still not appearing in the device discovery page. An error was encountered as the values corresponding to the key `vendor_name` in the copied python script `API_API_WeMoPlugloadtest`were the same as the values in the original API script `API_WeMoPlugload`. Thus it is impossible to have two devices with different vendor names and the same device model.

Wednesday, July 3, 2019

After receiving a message from Dr. Miah, the current goal is to find a way to add a new copy of the plugload interface into the same directory as the current one and change the variables `vendor_name` and `device_model`. Following the meeting with Dr. Miah, I realized that I must work to fully understand the BEMOSS interface and the wemo switch otherwise I will not be able implement a new device successfully. This must be done by understanding the source code. I started reading through the device discovery agent source code line by line to fully understand it.

Thursday, July 4, 2019

After making some comments in the `agent.py` file, indentation errors appeared while launching the server, the decision was made to reclone the BEMOSS repo. Progress must be made quickly today to understand the code and complete the clone of the plugload interface. I found a try ... except statement on lines 307 to 314 of `agent.py` that prevent the device from being detected. In a nutshell, the agent queries the PostgreSQL database with the query `SELECT * from supported_devices where vendor_name = 'Belkin International Inc.'  and device_model = 'Insight'` when querying the Insight. If no row in the table is found, then it is determined that the device is not supported by BEMOSS.

I attempted to place a copy of the file `API_WeMoPlugload.py` in a separate directory named `research`; however, when running the file `platfrom_initiator.py` the database prevents a device with a duplicate key from being added.

Friday, July 5, 2019

Several attempts to restart the server in the lab were made by recloning the repo as many errors were encountered causing the server to not startup.

Monday, July 8, 2019

I was able to detect a new device with the copy of the wemo plug api in the second folder I made. On the device discovery page, the device name and model were the ones I declared in the `API_WeMocopy.py` file. Once the device was discovered, the vendor name and model did not match what I provided in the previously mentioned file. Plug and play discovery is never completed due to this error reported by the device discovery agent:

```
IntegrityError: duplicate key
value violates unique constraint
"device_info_pkey"\nDETAIL:
Key (agent_id)=(Insi_231707K120123A) already exists.\n\n')
```

I restarted BEMOSS to see if this was an issue due to the fact that the device had not been deleted from the device info table. Since significant alterations to the source code will likely be necessary, it was decided that the focus will now be placed on the dc motor interface now which is due July 31, 2019.

The first step I took in understanding how to add the device to the software is hardcoding a dictionary with ip address, mac address, device model (raspberrypi's username), and vendor name (Raspberry Pi Foundation). I defined these variables in a method named discover in the class API inside the module `API_PittmanMotor`. In the near future, an automated plug and play discovery process will need to be implemented but for now I would like to simply hardcode the parameters. Since I am a little lost on how the web framework used to build the platform (Django) works, I decided to watch a tutorial on it and do further research.

Django is an open source web framework. It is based on the model, view, template architecture system as explained below.

1. Model: Interact with and validate python data

2. Template: Presentation layer (html files)

3. View: Decides which template to display by interacting with the model

Project uses multiple apps to define the functionality of the site.

Tuesday, July 9, 2019

I postponed increasing knowledge as the more important task at the moment is understanding the API translator methods such as `setDeviceStatus`, `getDataFromDevice`. After starting up BEMOSS without any code to the body of the previously mentioned functions the device was able to be succesfully discovered, but the navigation bar disappeared when setting the device status to approved in the UI. Thus more configuration will be necessary to create a system that perfectly emulates the plugload interface.

As a test I ran the file `DeviceAPI/BaseAPI_WeMo.py` in a virtual environment to toggle the wemo switch. After commenting out a couple of method calls in the main function I was able to succesfully toggle the device. This should be a good way of testing the raspberry pi without having to start BEMOSS everytime.
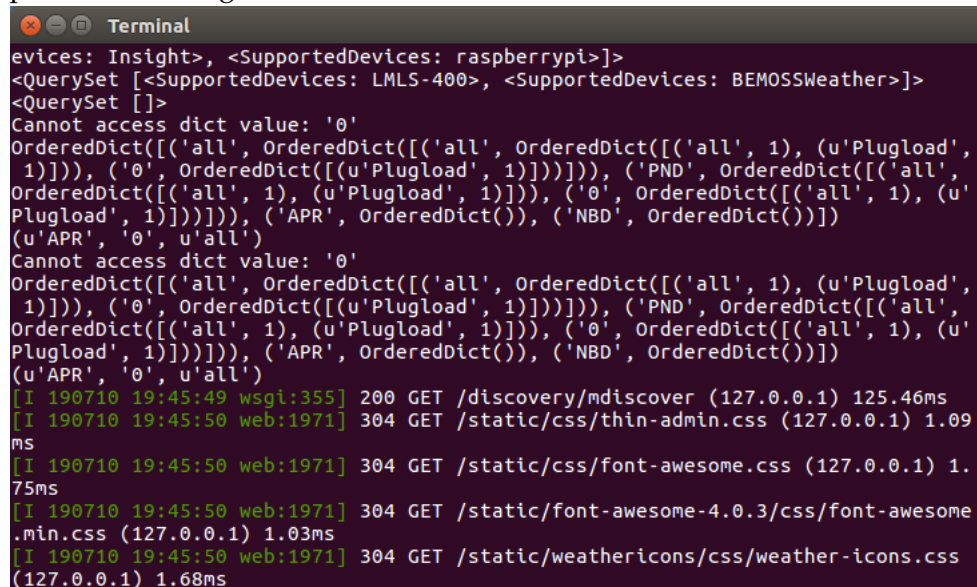
Something I found on the developer resources is that the Device Agent is responsible for handling most of the communication with the UI. It will likely beneficial for me to read over this source code. The method `agent_setup` contains a decorator that registers the method to the callback 'onsetup' which is an event triggered at one of the agents' stages of life. Inside this method, three method calls generate callbacks to methods defined inside the BasicAgent class which are `updateUIBehavior`, `deviceIdentifyBehavior`, and `deviceControlBehavior`.

I added code to `setDeviceStatus` in the file `API_RaspberryPi.py` to ssh into the raspberry pi and run python scripts depending if the device status is set to "ON" or "OFF". Once I started up BEMOSS though I ran into some problems as had not defined the method `getDataFromDevice`. This must return a dictionary otherwise errors will arise when displaying the data on the dashboard.

Wednesday, July 10, 2019

The problem involving the side navigation bar not being rendered was not a result of no dictionary being returned from getDataFromDevice. I found that each group of the mac address must not be separated by colons; rather no delimiter is to be used at all. Once I removed the colons from the MAC address in the API file, I was able to succesfully control the Raspberry Pi at my house. I wrote scripts that are able to turn on and off an led connected to one of the GPIO pins on my RPi. In the lab, it should be fairly simple to run the scripts on the RPi in the lab XBEEON.py and XBEEOFF.py to send commands to the motor in place of the scripts I use to control the LED.

A second problem still exists stemming from a KeyError exception in the file Web_Server/webapps/dashboard/templatetags/device_count_in_zone.py. I get the error Cannot access the value, returning default: '0' which is printed in the function get_value_with_default when the dictionary does not contain one of the keys passed as an argument into the function. This is shown in the screen capture below.

```
evices: Insight>, <SupportedDevices: raspberrypi>]>
<QuerySet [<SupportedDevices: LMLS-400>, <SupportedDevices: BEMOSSWeather>]>
<QuerySet []>
Cannot access dict value: '0'
OrderedDict([('all', OrderedDict([('all', OrderedDict([('all', 1), (u'Plugload',
 1)])), ('0', OrderedDict([(u'Plugload', 1)]))])), ('PND', OrderedDict([('all',
OrderedDict([('all', 1), (u'Plugload', 1)])), ('0', OrderedDict([('all', 1), (u'
Plugload', 1)]))])), ('APR', OrderedDict()), ('NBD', OrderedDict())])
(u'APR', '0', u'all')
Cannot access dict value: '0'
OrderedDict([('all', OrderedDict([('all', OrderedDict([('all', 1), (u'Plugload',
 1)])), ('0', OrderedDict([(u'Plugload', 1)]))])), ('PND', OrderedDict([('all',
OrderedDict([('all', 1), (u'Plugload', 1)])), ('0', OrderedDict([('all', 1), (u'
Plugload', 1)]))])), ('APR', OrderedDict()), ('NBD', OrderedDict())])
(u'APR', '0', u'all')
[I 190710 19:45:49 wsgi:355] 200 GET /discovery/mdiscover (127.0.0.1) 125.46ms
[I 190710 19:45:50 web:1971] 304 GET /static/css/thin-admin.css (127.0.0.1) 1.09
ms
[I 190710 19:45:50 web:1971] 304 GET /static/css/font-awesome.css (127.0.0.1) 1.
75ms
[I 190710 19:45:50 web:1971] 304 GET /static/font-awesome-4.0.3/css/font-awesome
.min.css (127.0.0.1) 1.03ms
[I 190710 19:45:50 web:1971] 304 GET /static/weathericons/css/weather-icons.css
(127.0.0.1) 1.68ms
```

In the lab, I was able to successfully control the motor with the plugload interface by making changes to the file API_RaspberryPi.py. More work was done to correct the error.

Thursday, July 11, 2019

To test whether the new API script I wrote was the cause of the problem, I moved the file `API_RaspberryPi.py` to the home directory and restarted BEMOSS. The message `Cannot access the value, returning default:  0` was still printed to the server log at startup, so the only thing I can conclude is that this is not an issue with the new API file. As another test, I will install BEMOSS in another directory. The same messages were printed to the log when attempting to the get the number of devices on the network, so at this point the messages will be ignored.

As a first step in automating the discovery process, I added the line to the method discover in `API_RaspberryPi.py`
`nmap = subprocess.check_output('sudo nmap -sn 192.168.1.0/24',shell=True)`
When this line is run, a `CalledProcessError` is thrown which remove nmaps as an option that can be used to detect device information. The command was attempted to be run without superuser priviliges; however, raw IP packets are needed to resolve MAC addresses and manufacturer names with nmap.

Friday, July 12, 2019

I completed writing the discover method for the Raspberry Pi API today. A file named `ownerinfo.json` was added to the research directory where the mac address of the Raspberry Pi used to control the motor can be added. This makes identifying the correct Raspberry Pi easier as it is possible that more than one Pi could be on the network at the same time. The command nmap was used to ping all machines on subnet 255.255.255.0. After this has been done information is stored in the machine's cache so that the address resolution protocol (arp command) may be used to find the mac addresses corresponsding to the ip address on the network. After this, it is determined whether the pi's mac addresses is located in the output of the `arp -a` command.

I need to research DNS python packages to find the username of Raspberry Pi. Also, methods for storing the Raspberry Pi's password securely in the BEMOSS database must be researched.

Monday, July 16, 2019

After doing some research I decided that I will be attempting to discover the Raspberry Pi with SSDP (Simple Service Discovery Protocol) rather than using the `nmap` and `arp` commands. Research will need to be done on these two protolocols to be able to understand how to discover and control the RPi.

UPnP or Universal Plug and Play consists of three different protocols which are

- SSDP (Simple Service Discovery Protocol) for finding devices on the network

- SCPD (Service Control Point Definition) defines the actions provided

- SOAP (Simple Object Access Protocol) calls actions

I found after further research that UPnP is simply not going to work with raspberry pi as it is more designed for home media servers and not suited for my specific application which is running commands on the Pi.

Tuesday, July 16, 2019

I attempted to run the command `sudo nmap -sn 192.168.1.0/24` in the discover method; however, I was not able to login in the volttron log terminal although the prompt for the password did appear. I attempted to launch the terminal as superuser by running the command to launch the terminal in the `GUI.py` as superuser. However, I ran into several issues involving priviliges although I did remove the sudo prefix from the command. Now I will reinstall BEMOSS.

Rather than using both the nmap and arp commands together I made the decision to running the nmap command as superuser to resolve the manufacturer names and MAC addresses. I ran into a problem when using `subprocess.call('sudo nmap -sn 192.168.1.0/24')` as the terminal does not allow any text to be passed to stdin while the agents are running. In addition, execution of the function will continue even though system password is provided. In the end I decided to revert back to the original code as the motivation for running nmap as super user is to retrieve the mac and IP addresses.

Lastly, I found that the Raspberry Pi contains an option named Remote GPIO in the Raspi-Config which will enable the GPIO pins to be controlled over the network. No login is required to access the pins on this server only the IP address. This will hopefully streamlines tasks involving the gpio pins greatly.

Wednesday, July 17, 2019

Following work on my presentation, I discussed with Dr. Miah using the Remote GPIO which was agreed upon. However, for the next presentation I would like to have a fully functional interface using the current method. This will require me to encrypt both the username and password of the Raspberry Pi and store in the json file that the raspberry pi API file accesses. Thus some research will need to be done on how to perform this encryption. Later (next week) I would like to get the remote GPIO interface working and implemented. Some problems still exist with this method such as security as multiple different machine may be able to control the GPIO pins at once. Also, the XBee library that is used to control the xbee modules might not be compatible with the latter method. I know very little of how zigbee communication works or how the modules work, so this will require much research.

I encrypted both the username and password of the raspberry pi by using the function `encrypt_value` in the module `bemoss_lib.utils.encrypt` and placed both these encrypted strings in the json file. I found that I needed to decrypt the values when formatting the string used to ssh into the pi. After finding that only the password must be encrypted, I replaced the username in the json file with the plain text version.

Thursday, July 18, 2019

Today, I found that I have not implemented the method `identifyDevice` in the Raspberry Pi API. It is to be implemented as follows:

- At start, `identifyDeviceResult = False`

- Keep the current state of one of the changeable device variables (in this case the status)

- Change device variable that is visually noticeable

- Delay for 5 seconds

- Change device variable back to previous state

- if successful, `identifyDeviceResult = True`

After looking through some of the Philips Hue API code, I found that the method `get_variable` is used to check whether the device is on or off rather than directly communicating with the device through the method `getDataFromDevice` like in the WeMo plugload API. As a test I will print `get_variable('status')` in the method `identifyDevice` in the Raspberry Pi API to check whether this key value pair is in the dictionary defined in the BaseAPI's constructor. As of now, I do not know of a way of checking the status of the motor from the RPi using the GPIO pins and XBEE modules, so using the method from the Philips Hue API would be ideal.

After pressing the button `Identify Device` in the list of plugload controllers, the message logged to the console is always `'ON'`. The reason for this is likely the code I wrote in the method `getDataFromDevice` which is shown below:

```
def getDataFromDevice(self):
    # TODO
    devicedata = dict()
    devicedata['status'] = BEMOSS_ONTOLOGY.STATUS.POSSIBLE_VALUES.ON
    # setting the value to "ON"
    return devicedata
```

The statement `BEMOSS_ONTOLOGY.STATUS.POSSIBLE_VALUES.ON = 'ON'` is defined in the file `BEMOSS_ONTOLOGY`. From the BEMOSS developer resources for layer 4 of the BEMOSS

hierarchy I found:

"Out of all the above methods, the getDataFromDevice method can be periodic. This means it can query and receive device data periodically. This ensures that the data are updated."

In the Basic Agent source code, a method name `deviceMonitorBehavior` is called periodically which calls `getDeviceStatus`. A call to `getDataFromDevice` occurs here which returns a dictionary and calls `convertDeviceStatus` to update the `variables` dictionary which is declared by the `BaseAPI`. Thus I must find a way to directly query the Raspberry Pi for the motor status. To start the pin mappings are between the RPi and XBee s2c module are listed below:

Raspberry Pi	XBee S2C
6 (GND)	VSS
8 (TXD)	DIN
10 (RXD)	DOUT
17 (3V3)	VCC

Friday, July 19, 2019

In the afternoon session with the members of ram group, I experimented with the AT commands used to control the remote XBee connected to the L298N H-Bridge driver. The pin mappings are shown in the table below:

XBee	L298N
IO3	I2
IO4	I1

Also, from the L298N datasheet this table lists the DC motor control A pins and how the output of the h-bridge is effected:

EA	I1	I2	Motor A status
0	0	1	Clockwise rotation
0	1	0	Anticlockwise rotation

As a side note, the EA port is used for PWM.

However, the shaft rotations are swapped when using the motor in the lab.

Monday, July 22, 2019

One of the AT commands use to read the digital IO of the XBee modules is the IC (Digital Change Detection) command. The command can "set or read the digital I/O pins to monitor for changes in the I/O state." When the DIO state changes, a sample can be sent immediately to the coordinator XBee from the remote XBee. However, this does not send any information on the state of the pin. The command IS "forces a read of all enabled digital and analog input lines." Of course, here a wire will be need to be connected between the output pin (IO3 or IO4) and another pin configured as a digital input. When the device needs to be queried for the status of the digital IO pins, the command IS will be sent to the remote XBee. Then using the method `XBee.wait_read_frame` I can read the data received on the UART connection from the remote XBee. Tomorrow, in the lab this will need to be experimented with.

Tuesday, July 23, 2019

I tried using the commands mentioned in yesterday's lab notebook to no avail. This is the code I wrote to try to receive data sent from the remote XBee:

```
from xbee import XBee
import serial
import time

ser = serial.Serial('/dev/ttyS0',9600)
xbee = XBee(ser)

xbee.remote_at(
        dest_addr=b'\xff\xff'
        command='D3'
        parameter=b'\x04')
xbee.remote_at(
        dest_addr=b'\xff\xff'
        command='IS'
        parameter=b'\x03')
while True:
    try:
        print(xbee.wait_read_frame())
    except KeyboardInterrupt:
        break
```

The XCTU software was downloaded to work on configuring the remote MotorDriver XBee. IR, sample rate, was set to 1000 ms or a hex value of 0x3e8 and D1, dio1, was set as a digital input. Running `ser.read()` results in an output of '\x00'. Thus, no data is being received even though data should be sent to coordinator at a rate of 1 second.

Wednesday, July 24, 2019

I decided to try experimenting with the digi-xbee python library. After following the instructions to configure the devices at
https://xbplib.readthedocs.io/en/latest/getting_started_with_xbee_python_library.html#gsginstall, I was unable to send AT commands to the end point XBee to toggle the motor. This was later resolved after setting the output values of IO3 and IO4 to opposite values using XCTU. Later I was able to read the digital values of these IO lines (whether they are high or low). This means that only two IO lines will be needed to control the remote XBee module rather than 4 as the status of the IO line can be read directly even if it is a digial output. A script was written named XBEEcontrol.py that can toggle the motor and determine the status of the enabled pins on the remote XBEE. At the end of the day I was able to successfully identify the motor by toggling it twice with a 5 second pause inbetween. The last part of the implementation is adding the motor images to the frontend.

Thursday, July 25, 2019

I found an picture of the GM8000 series motor and added images denoting when the motor is on and off to the `Web_Server/static/images` directory after removing the default background with GIMP.

The next task is working on adding a new device type to BEMOSS called 'Motor'. In the file `Web_Server/run/defaultDB.py,` I added the lines

```
dt5 = DeviceType(id=5, device_type='Motor')
dt5.save()
```

Friday, July 26, 2019

Starting BEMOSS with changing the lines from yesterday result in an error when attempting to add the Raspberry Pi API to the list of the supported devices. After running the file `BEMOSS3.5/Web_Server/run/defaultDB.py` by typing `python Web_Server/run/defaultDB.py` in the `BEMOSS3.5` directory the problem was fixed. BEMOSS was started successfully; however, the widget listing Motors was not available on the 'Discover New Devices' page. In the file `Web_Server/webapps/discovery/views.py`, on line 58, `power_meters` was changed to motors and on line 71, the key value pair for the power_meters in the `devices` dictionary was removed and a key value pair for motor was added. After making further changes to the javascript and html manual discovery files, the issue was not corrected. It was later found that a closing div tag was missing from the file to close the div with class "row". Another exception was thrown when trying to access the pass usage statistics.

Monday, July 29, 2019

The following error was thrown when running the `platform_initiator.py` file when attempting to add the motor API to the database.
```
insert or update on table "supported_devices"
violates foreign key constraint
"supported_devices_device_type_id_1f8832cb_fk_device_type_id"
DETAIL:  Key (device_type_id)=(5) is not present
in table "device_type".
```
This error is fixed by running the file `Web_Server/run/defaultDB.py` which will update the BEMOSS database. Next to help clear up the VOLTTRON log file I attempted to dissect the error message reported by the Time Series Database Agent. This error is presented below

```
Traceback (most recent call last):
    File "/usr/lib/python2.7/logging/__init__.py", line 861, in emit
        msg = self.format(record)
    File "/usr/lib/python2.7/logging/__init__.py", line 734, in format
        return fmt.format(record)
    File "/home/ramgroup/BEMOSS3.5/volttron/platform/agent/utils.py", line 242, in format
        return jsonapi.dumps(dct)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/zmq/utils/jsonapi.py", line 40, in dumps
        s = jsonmod.dumps(o, **kwargs)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/simplejson/__init__.py", line 399, in dumps
        **kw).encode(obj)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/simplejson/encoder.py", line 296, in encode
        chunk = self.iterencode(o, _one_shot=True)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/simplejson/encoder.py", line 378, in iterencode
        return _iterencode(o, 0)
    File "/home/ramgroup/BEMOSS3.5/env/local/lib/python2.7/
    site-packages/simplejson/encoder.py", line 273, in default
        o.__class__.__name__)
TypeError: Object of type DefaultEndPoint is not JSON serializable
```

```
Logged from file connection.py, line 1100
```

A logging issue is obviously present here and these stack traces are printed via logging message. Logging messages can have different levels of severity. In increasing order they are:

- DEBUG: Detailed information, typically of interest only when diagnosing problems.

- INFO: Confirmation that things are working as expected.

- WARNING: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.

- ERROR: Due to a more serious problem, the software has not been able to perform some function.

- CRITICAL: A serious error, indicating that the program itself may be unable to continue running.

The error above has level ERROR so the TSD Agent is not able to perform some function.

It appears as if some objects cannot serialized by the python json module after some investigation. A possible fix to this problem is removing all the debug logging statements from the files in the cassandra python driver. However, this would take a great deal of time as there are many of them. Also, these logging statements are likely needed in case another problem arises. As mentioned from the log statements, on line 1100 of cassandra.connection.py, the following debug log statement exists:

```
log.debug("Sending options message heartbeat on idle connection (%s) %s",
                id(connection), connection.endpoint)
```

Tuesday, July 30, 2019

In the lab, I found that the verision of the Datastax python driver used by the previous senior project group is different and only logs host names to the file rather than objects of classes defined inside the connection file. Today, I will meet with Dr. Miah to work on resolving this issue. While working on testing the motor interface I found that the power supply is not working properly. It fails to turn on immediately when the `XBEEcontrol.py` script is run with the argument `turnOnMotor`. A potential cause is the lack of a flyback diode across the input terminals to the motor which could be damaging the power supply due to voltage spikes.

In the meantime, before the meeting, I will work on trying to add plugload chart onto the motor page. Even though I am able to control the motor with the switch and identify device button, none of the data is added to the database. Something to note is that each table in BEMOSS is named according to the `agent_id`. Somewhere in the repository these tables are added to the database; however, it is unclear where that occurs. According to the BEMOSS-Data-Management section of the developer resources: "During operation, the device agents connect to the Cassandra node on the same machine the agent is running, using Cassandra python driver, and writes time series data to a unique table named after the device_id it is controlling."

After the meeting with Dr. Miah, it was determined that the current bug involving the cassandra python driver is a top priority. No paper will be submitted if this current issue still exists. A print statement was added on lines 133 and 139 of the TSDAgent source code: `print(message)`. One of the messages logged to the `volttron.log` file is

```
{u'tablename': u'platform_event', u'all_vars':
{u'event_id': u'155df44e-e45b-4271-bc44-ce053509d22a', u'start_time':
u'2019-07-31 04:37:42:936113*None', u'date_id': u'2019-07-30', u'agent_id':
u'platformmonitoragent_Node1', u'end_time': u'2019-07-31
04:38:44:552121*None'}, u'log_vars':
{u'event_id': u'UUID', u'start_time': u'TIMESTAMP', u'date_id': u'text',
u'agent_id': u'text', u'end_time': u'TIMESTAMP'}}
```

Wednesday, July 31, 2019

This morning I checked the version of the cassandra-driver for python on the current machine. It is 3.18.0 and the version on Bob's machine is 3.16.0. Also, after commenting out lines 135 and 141 in `Agents/TSDAgent/TSD/agent.py`, the same logging error was reported to the volttron log file. Thus, the function `do_insertion_jobs` is not the only problem rather also the `insert` and `customInsert` functions. In all, the cause of this problem will be very difficult to find as there are likely many calls to methods in the cassandra-driver package. In addition, another error is reported involving the VOLT-TRON bacnet proxy agent when my machine is connected to both Wi-Fi and Ethernet. This problem is resolved if the host computer is only connected to the ECE-Robotics 1 network when running BEMOSS. A screencap is posted below. A attempt to correct

```
2019-07-31 09:50:55,553 (bacnet_proxyagent-0.1 3177) <stderr> ERROR: Traceback (most recent call last):
2019-07-31 09:50:55,553 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "<string>", line 1, in <module>
2019-07-31 09:50:55,554 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/.volttron/agents/
fb884523-9da2-4d54-8bbf-92bb41a07522/bacnet_proxyagent-0.1/bacnet_proxy/agent.py", line 607, in main
2019-07-31 09:50:55,555 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     utils.vip_main(bacnet_proxy_agent)
2019-07-31 09:50:55,555 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/BEMOSS3.5/volttron/platform/agent/
utils.py", line 210, in vip_main
2019-07-31 09:50:55,555 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     agent = agent_class(config_path=config,
identity=agent_uuid, **kwargs)
2019-07-31 09:50:55,556 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/.volttron/agents/
fb884523-9da2-4d54-8bbf-92bb41a07522/bacnet_proxyagent-0.1/bacnet_proxy/agent.py", line 349, in bacnet_proxy_agent
2019-07-31 09:50:55,556 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     heartbeat_autostart=True, identity=vip_identity, **kwargs)
2019-07-31 09:50:55,557 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/.volttron/agents/
fb884523-9da2-4d54-8bbf-92bb41a07522/bacnet_proxyagent-0.1/bacnet_proxy/agent.py", line 366, in __init__
2019-07-31 09:50:55,557 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     obj_id, obj_name, ven_id)
2019-07-31 09:50:55,558 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/.volttron/agents/
fb884523-9da2-4d54-8bbf-92bb41a07522/bacnet_proxyagent-0.1/bacnet_proxy/agent.py", line 395, in setup_device
2019-07-31 09:50:55,558 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     self.this_application = BACnet_application(this_device,
address)
2019-07-31 09:50:55,559 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/.volttron/agents/
fb884523-9da2-4d54-8bbf-92bb41a07522/bacnet_proxyagent-0.1/bacnet_proxy/agent.py", line 133, in __init__
2019-07-31 09:50:55,559 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     BIPSimpleApplication.__init__(self, *args)
2019-07-31 09:50:55,559 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/BEMOSS3.5/env/local/lib/
python2.7/site-packages/bacpypes/app.py", line 589, in __init__
2019-07-31 09:50:55,560 (bacnet_proxyagent-0.1 3177) <stdout> INFO: <module 'json' from '/usr/lib/python2.7/json/__init__.pyc'>
2019-07-31 09:50:55,560 (bacnet_proxyagent-0.1 3177) <stdout> INFO: OS settings imported
2019-07-31 09:50:55,560 () volttron.platform.auth INFO: authentication success: domain='vip', address='localhost:1000:1000:3172',
mechanism='NULL', credentials=[], user_id='BEMOSSAGENT'
2019-07-31 09:50:55,568 () volttron.platform.aip INFO: starting agent /home/ramgroup/.volttron/
agents/725d3913-3e13-4b1f-8a20-8459c00555b4/TSDagent-3.0
2019-07-31 09:50:55,574 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     self.mux = UDPMultiplexer(self.localAddress)
2019-07-31 09:50:55,575 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/BEMOSS3.5/env/local/lib/
python2.7/site-packages/bacpypes/bvllservice.py", line 95, in __init__
2019-07-31 09:50:55,576 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     self.broadcastPort = UDPDirector(self.addrBroadcastTuple,
reuse=True)
2019-07-31 09:50:55,577 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/home/ramgroup/BEMOSS3.5/env/local/lib/
python2.7/site-packages/bacpypes/udp.py", line 147, in __init__
2019-07-31 09:50:55,577 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     self.bind(address)
2019-07-31 09:50:55,578 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/usr/lib/python2.7/asyncore.py", line 342, in bind
2019-07-31 09:50:55,594 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     return self.socket.bind(addr)
2019-07-31 09:50:55,594 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:   File "/usr/lib/python2.7/socket.py", line 228, in meth
2019-07-31 09:50:55,595 (bacnet_proxyagent-0.1 3177) <stderr> ERROR:     return getattr(self._sock,name)(*args)
2019-07-31 09:50:55,595 (bacnet_proxyagent-0.1 3177) <stderr> ERROR: socket.error: [Errno 99] Cannot assign requested address
```

the error was made by cloning BEMOSS into a new repository and ensuring the version 3.16.0 was installed rather than the latest 3.18.0. Next time, however, I need to dig deeper into the issue and understand it rather than use the "hack it until it works method". Due to time constraints, I used this method instead. The TSD agent errors were no longer present; however, another possible unrelated error was thrown by the network agent:

```
Traceback (most recent call last):
  File "/home/ramgroup/tests/BEMOSS3.5/volttron/platform/
  vip/agent/subsystems/rpc.py", line 168, in method
    return method(*args, **kwargs)
```

```
File "/home/ramgroup/tests/BEMOSS3.5/volttron/platform/vip/
agent/subsystems/pubsub.py", line 262, in _peer_push
  callback(peer, sender, bus, topic, headers, message)
File "/home/ramgroup/.volttron/agents/
90ab11e3-bdb4-45e7-95c7-c3cbf682b92b/networkagent-0.1/network/agent.py",
line 222, in on_match_change
  self.curcon.commit()
File "/home/ramgroup/tests/BEMOSS3.5/
bemoss_lib/utils/db_helper.py", line 82, in reconnect
  return func(*args,**kwargs)
File "/home/ramgroup/tests/BEMOSS3.5/
bemoss_lib/utils/db_helper.py", line 109, in commit
  return self.con.commit()
IntegrityError: insert or update on table "node_device"
violates foreign key constraint
"node_device_assigned_node_id_cba9785e_fk_node_info_node_id"
DETAIL:  Key (assigned_node_id)=(0) is not present
in table "node_info"
```

Unfortunately, as time went on the errors involving the TSD agent appeared again. At this point I have very little idea what is causing the issue. I do have a theory that the replaced URL is causing problems. So I will run a few tests:

1. Run BEMOSS with resource from old link (`dsc-cassandra-3.0.9-bin.tar.gz`) with cassandra-driver 3.16.0 on ramgroup laptop

2. Run BEMOSS with resource from old link (`dsc-cassandra-3.0.9-bin.tar.gz`) with cassandra-driver 3.18.0 on ramgroup laptop

Number 1 could not be done as the cassandra directory cannot simply be transferred from one BEMOSS directory to another when installing. After performing 2, the same TSD agent errors occurred leading me to strongly believe that this has something to do with the python cassandra driver. I will run BEMOSS again with the new link and cassandra-driver 3.16.0. The version of the cassandra-driver packaged is changed with the command `pip install cassandra-driver==version` after running the command `. env/bin/activate` to run the previous command in the BEMOSS virtual environment. After further testing I found that errors will be thrown by the time-series database agent with the cassandra-driver 3.16.0 only when BEMOSS run is run for the first time.

Thursday, August 1, 2019

On line 2852 of cluster.py:

`log.debug("[control connection] Opening new connection to %s", host)`

After emailing Ashraf, the problem found yesterday is not an error as BEMOSS initializes some parameters when run the first time. So, once running the problems disappear. No more work needs to be done on this issues. The July 31 deadline was not successfully for the fully functional motor interface as the statistics chart for the motor did not appear aftre identifying and controlling the motor although the chart did appear one time.

I spent some time learning about VOLTTRON on `https://volttron.readthedocs.io`

Friday, August 2, 2019

The presentation was practiced today and a few additions were made. After reading a section on Agent mobility on the VOLTTRON documentation, I decided that I would like to implement ssh key host authentication to remove the password inconvenience of controlling the raspberry pi. Running the command

`ls .ssh`

showed only the file `known_hosts` so no private or public keys had been generated previously. Then the command

`ssh-keygen`

was run and the keys were placed into the `~/.ssh` directory. Lastly, the command `ssh-copy-id pi@192.168.1.41` was executed to send the public key generated on the BEMOSS host machine to the RPi. A login was prompted for and the system password of the ramgroup machine was entered. After this the process was succesfully completed. Now by simply running the command `ssh pi@<IP-ADDRESS>` commands can be executed. This was followed from the tutorial at

https://www.raspberrypi.org/documentation/remote-access/ssh/passwordless.md

Monday, August 5, 2019

Research was done on the publish subscribe model that volttron uses to communicate between agents. Publishers (analogous to server entities) publish topics to a message queue which are then subscribed to by subscribers (analagous to clients). A subscriber can subscribe to multiple publishers and a publisher can publish to multiple subscribers. The intention was to have a Skype meeting with Ashraf at 7:00 pm cst today, but the call was never made; thus, another day much be chosen for working on completing the interface.

Tuesday, August 6, 2019

Wednesday, August 7, 2019

Ubuntu 16.04 was reinstalled on the ramgroup laptop due to some package issues after attempting to install octave via the command line. The section 3 Example 2: Smart Appliance Scheduling was read in [3]. The optimal time to run a six-cycle dishwasher between $17:00$ and $24:00$ was analyzed with an electricity price that changes over time. The goal is to minimize

$$\sum_{k=0}^{N-1} \frac{1}{4} c_k p(x_k + u_k) u_k$$

where c_k is the electricity cost in USD/kWh, k refers to each 15 minute time period, N is the final period, $p(x_k + u_k)$ is the power required by cycle $x_{k+1} = x_k + u_k$, and $u_k \in \{0, 1\}$ is the control variable determing whether the dishwasher should maintain its cycle or move to the next cycle.

Next Monday, I will have a meeting with Ashraf about adding the new device API.

Thursday, August 8, 2019

The principle of optimality can be expressed as the equation $V_k(x_k)$ or the cost-to-go from time index k to N. In this problem example, the equation is

$$V_k(x_k) = min \left\{ V_{k+1}(x_k), \frac{1}{4} c_k p(x_k + 1) + V_{k+1}(x_k + 1) \right\}$$

In other words, a decision must be made whether to continue with the last state or continue to the next state depending on which option is cheaper. Overall, conclusions were made about when the cycles of the dishwasher should be started according to when the electricity cost c_k is lowest in the given time frame. This concept can be extended to really any stateful device that uses energy in a building. Using real power cost statistics from the web, this algorithm could be implemented within BEMOSS.

Friday, August 9, 2019

While reading the article [1], some interesting facts were discovered about BEMOSS. One is that lighting loads, plug load, and hvac controllers account for almost 75% of loads in small and medium sized buildings. In the research going forward, it would be ideal to focus efforts on implementing one of these types of devices as they have the most relevance. Section IV A. discusses the agent architecture. In a path of control agent execution, the following steps are followed:

1. Obtain the configuration for the agent which contains the agent id, agent message pub/sub address

2. Instantiate a new device API object for communicating, monitoring and controlling the device

3. Setup variables and connect with databases (time-series and metadata)

4. Monitor the status of the device with device monitor agent behavior

5. After receiving a message from the UI, one of the behaviors (UpdateDeviceStatus, DeviceControl, IdentifyDevice) is triggered

Periodically, the control agent will update its knowledge on what is knows about a device based on the setup monitoring period. The identify device behavior is for identification of a device according to its corresponding control agent. Section IV E. discusses application development. The process involves many different steps. Different examples of applications are demand response, price-based management, and load control based on local conditions. It may be interesting to implement the dynamic programming algorithm that was mentioned in the previous lab notebook entry by using demand response signals (OpenADR) from the cloud. As mentioned before, I would like to implement another either lighting, plugload or hvac controller. At first, I thought of adding a wifi enabled dimmer switch but this has actually been already added (WeMo dimmer). This was realized after looking in the script `BaseAPI_WeMo.py`. Official support has not been added yet for the device as seen on the supported device table on www.bemoss.org. Other options will need to be researched then.

A test was made to isolate the issue of the chart not appearing by configuring the motor as a plugload and running BEMOSS. The chart appeared normally.

Monday, August 12, 2019

After the meeting with Ashraf, it was concluded that I must find how to add the new device type to BEMOSS alone as this is an open-source project and the development team only supports lighting, HVAC, and plugload controllers. In `bemoss_install_v3.5.sh`, on line 62, the `defaultDB.py` program is run which is where the device types are added to the PostgreSQL database:

`python Web_Server/run/defaultDB.py`

A copy of `charts_plugload.html` was added to the directory
`PROJECT_DIR/Web_Server/webapps/charts/templates/charts` and renamed to `charts_motor.html`.
In addition, a directory named motor was added in the directory
`PROJECT_DIR/Web_Server/webapps/device/templates`. Inside the motor directory, a copy of the `plugload.html` file was added and renamed to `motor.html`. The webapp html template `manual_discovery.html` was modified by copying lines 192 to 213 and pasting them below the lines for the plug load controllers. Then, line 57 of
`PROJECT_DIR/Web_Server/webapps/discovery/views.py` was changed to

`motors = SupportedDevices.objects.filter(device_type_id=5)`

and line 71 was changed to

```
devices = {'hvac': hvac, 'lt_loads':lt_loads,
  'plugloads':plugloads, 'sensors':sensors, 'motors':motors}
```

With these changes, BEMOSS was restarted and tested. Then in the `API_info` method in the `API_motor.py` script, the device type id of the motor was set to 5.

Another issue appeared when no device appeared on the list supported devices on the discovery page. This was resolved by updating the html in the `manual_discovery.html` file for the motor widget.

I found another error when discovering the pittman motor. The device discovery agent receives an empty message when it subscribes to the message sent by the manual discovery views function `discover_new_devices`. This was fixed after making further changes to the django template code in `manual_discovery.html`.

Now the goal is to fix the chart issue. Some research was done in the file
`PROJECT_DIR/Web_Server/webapps/charts/views.py`. Print statements were added in the `returnChartsPage` to check if some sort of exception is being thrown. In the file `charts_plugload.js` on line 80, a TypeError is thrown in the firefox developer console on the statistics page for the motor:

```
TypeError: _status[0] is undefined
```
This is the problematic javascript object:
```
axes: {
      xaxis: {
        label: "Time",
        renderer: $.jqplot.DateAxisRenderer,
        tickOptions:{formatString:'%m/%d, %H:%M'},

            min : _status[0][0],
            max: _status[_status.length-1][0]
      }
```
To test whether this is the cause of the problem I will change the `device_type_id` of the motor back to 3 corresponding to plugload as the chart appears when the device is set as a plugload. If the error is not shown in the console, then this must be the problem. The `_status` variable is defined in `charts_plugload.js` as `_status = {{ status|safe|list_if_none }};` A `console.log` statement was added to determine the value of `_status`. The `status` variable is a key in the dictionary `data_dict` rendered to the page by the `charts/view.py` file by the function `returnChartsPage`. The value corresponding to `'status'` is an empty list which is the source of the issue. Tomorrow in the lab, I will check this dictionary when controlling the wemo insight switch.

Tuesday, August 13, 2019

The result set printed to the terminal for the insight switch was similar to that printed for the motor other than the power variable. This is the variable list and result set used in the `returnChartsPage` function for the motor:

```
variable list:
[u'agent_id', u'date_id', u'status', u'time', u'user']
result set:
[[u'GM82_b827ebbfbade' u'2019-08-13' u'None' 1565704113458
  u'device_monitor']
 [u'GM82_b827ebbfbade' u'2019-08-13' 0 1565704188459 u'Admin_']]
```

Also, this is the data dictionary rendered to the charts page by the `returnChartsPage` function:

```
{u'status': [], 'mac': u'b827ebbfbade', 'nickname': u'Plugload2',
'node_name': u'Node1'}
```

For the Insight switch, the variable list and result set are:

```
variable list:
[u'agent_id', u'date_id', u'power', u'status', u'time', u'user']
result set:
[[u'Insi_231707K120123A' u'2019-08-13' 0.0 0 1565703938458
  u'device_monitor']
 [u'Insi_231707K120123A' u'2019-08-13' None 1 1565703963458 u'Admin_']
 [u'Insi_231707K120123A' u'2019-08-13' None 0 1565703968466 u'Admin_']]
```

The data dictionaries are:

```
{u'status': [[1565703938458.0, 0.0], [1565703963457.0, 0.0],
[1565703963458.0, 1.0], [1565703968465.0, 1.0], [1565703968466.0, 0.0],
[1565704485071.0, 0.0]], 'mac': u'231707K120123A',
'nickname': u'Plugload1', 'node_name': u'Node1'}
{u'status': [[1565703938458.0, 0.0], [1565703963457.0, 0.0], [1565703963458.0, 1.0],
[1565703968465.0, 1.0], [1565703968466.0, 0.0], [1565704485071.0, 0.0]],
'mac': u'231707K120123A', 'nickname': u'Plugload1', u'power':
[[1565703938458.0, 0.0], [1565704485071.0, 0.0]],
'node_name': u'Node1'}
```

One difference noticed is that for all lists in the result set the status is set to `None` when the user is `Admin_` for the motor. In the `parse_resultset` function, the line

```
x = [[lst[variables.index('time')],
```

```
lst[variables.index(data_point)]+0.0]s
                for lst in result_set
                if lst[variables.index(data_point)] is not None]
```
will not return a list for each list in `result_set` if the value of `status` is None. However, some values should still be returned as the value of `status` is not None when the user is `Admin_`.

The reason why `parse_resultset` is returning an empty list is the line
```
if len(result_set)>0 and type(result_set[0][variables.index(data_point)])
in [str,unicode]:
return []
```
From the `result_set` shown above,
`result_set[0][variables.index('status')] = u'None'` which is a unicode string thus the `parse_resultset` function is returning an empty list. The next step I took in solving this issue is by looking in the motor api code. Since the WeMo insight switch is working perfectly, there is likely a problem with the API code I wrote for the motor. I may need to compare the two device API's to see if there are any differences. As a test I commented out the conditional statement mentioned above. This did not appear to fix the problem. From the looking at the `result_set` data, once BEMOSS is started the new `status` values are not u'Null'. The u'Null' values were likely saved in the table when there was a mistake in the motor api code. The data in this table is still present upon restart, so a way must be found to delete the data in the table. First, I will I try to change the device type to motor rather than plugload.

The `cassandraDB.py` file contains a function named `delete` that is able to drop a table in the cassandra database. It takes a single argument, the agent id corresponding to the table to be dropped. When I called this function within a python interpreter in the `PROJECT_DIR/bemoss_lib/databases/cassandraAPI/` directory, errors were thrown but the operation seemed to complete sucessfully. There seems to be some problem with this function as `startDate` and `endDate` were not specified but errors were thrown by the pandas module although they are not required. However, the table was still deleted even when these parameters were not specified. After restarting BEMOSS with tables not present in the cassandra database, the chart appeared successfully on the statistics page. Lastly, another issue still exists on the dashboard discovery page. When I press the "Set all to 'Approved'" button, the device `Approval status` does not update. In `PROJECT_DIR/Web_Server/static/app_js/discovery.js`, the following code was added on lines 321 to 324:
```
$("#set_approve_Motor").click(function(e)) {
    e.preventDefault();
    $(".app_stat_Motor").text("Approved");
}
```

The `event.preventDefault` method will prevent the default action from being triggered and will not reload the page. This problem was not solved by performing the above action. After pressing the `Save Change to Motor Controllers` button, a popup is generated by firefox:

To display this page, Firefox must send information that will repeat any action (such as a search or order confirmation) that was performed earlier.

This issue was resolved by changing parenthesis and brackets of the above code as follows:

```
$("#set_approve_Motor").click(function(e) {
    e.preventDefault();
    $(".app_stat_Motor").text("Approved");
});
```

Wednesday, August 14, 2019

In a short meeting with Dr. Miah, we discussed issues with the motor. He found that something is shorting out causing the power supply to overload. This was quickly fixed by reinserting the twisted pair of wires into the screw terminals of the h-bridge driver.

Another issue with the software is when a plugload controller is discovered after discovering the motor, the nickname of the plugload controller (for example insight switch) is `plugload2` rather than `plugload1`. A likely reason is no changes were made to the html template after copying the plugload template and renaming to `motor.html`. After looking through the device discovery agent code, in the `DiscoveryAgent` constructor `self.device_num` is set to the number of devices discovered by the device discovery agent. Once a new device is discovered, in the `findDevices` method, the `self.device_num` variable is incremented, so the number after the device type refers to the total number of devices that have been discovered rather than the number of devices of the specific type that have been discovered. Later work was done on the presentation here are the topics:

- API translator discussion

- Adding a device API translator

- Adding a new device type

- Problems fixed

- Additions

- Priorities

Thursday, August 15, 2019

Friday, August 16, 2019

Monday, August 19, 2019

In the lab, when demonstrating the motor interface to Dr. Miah, the scheduling for the motor type has not been implemented. For now, this is fine as I do not have time at the moment. Later this could be useful in scheduling the motor to turn on based on responses from the grid. During the filming of a video demonstration of the progress I have made so far, the number displaying the number of discovered devices in the discover/manage tab appears to be incorrect when both the insight switch and motor are discovered (i.e. 1 is displayed when both devices are discovered rather than 2). One of the Python files likely does not contain code for the motor controller. This is the django template code used to display the number of devices next to the label of the tab:

`{% get_value_with_default device_count 'PND' 'all' 'all' 0 %}`

Arguments `device_count`, 'PND', 'all', 'all', and 0 are passed as arguments into the template tag `get_value_with_default`. However, I do not know where `device_count` is defined. `device_count` is a dictionary returning the number of devices under each approval status such as pending, approved, or non-bemoss device.

Tuesday, August 20, 2019

Today, I will document the full procedure for adding the motor to BEMOSS. Note `PROJECT_DIR` refers to the directory containing the installation of BEMOSS. For example, `~/BEMOSS3.5`.

1. Add the files `BaseAPI_RPi.py` and `API_Motor.py` to the directory `research` in the BEMOSS project directory. These are available on the Google Drive BEMOSS-Brian2019

2. In the file `PROJECT_DIR/bemoss_lib/utils/platform_initiator.py`, replace the line `APImodule = importlib.import_module("DeviceAPI."+file)` on line 247 with

   ```
   try:
       APImodule = importlib.import_module("DeviceAPI."+file)
   except Exception as er:
       APImodule = importlib.import_module("research."+file)
   ```

   This will allow the platform initiator to search both the `PROJECT_DIR/DeviceAPI` and `PROJECT_DIR/research` directories for device API files.

3. In the discovery agent code
   `PROJECT_DIR/Agents/DeviceDiscoveryAgent/devicediscovery/agent.py`,
   on line 167, replace `apiLib = importlib.import_module("DeviceAPI."+self.api)` with

   ```
   try:
       APImodule = importlib.import_module("DeviceAPI."+file)
   except Exception as er:
       APImodule = importlib.import_module("research."+file)
   ```

   to allow the basic agent to search both directories `PROJECT_DIR/DeviceAPI` and `PROJECT_DIR/research` for device AIP files.

4. Update the python script
   `PROJECT_DIR/Web_Server/run/defaultDB.py` with the line

   ```
   dt5 = DeviceType(id=5, device_type='Motor')
   dt5.save()
   ```

   to register the device type `Motor` in the BEMOSS database.

5. Update the html template
   `PROJECT_DIR/BEMOSS3.5/Web_Server/webapps/discovery/templates/discovery/manual_discovery.html,`
   by copying the html for the plugload widget and pasting below to create a widget for motors on the manual discovery page. Then replace the variables `plugloads` with `motors` and `eachpl` with `eachmot` in the copied html.

6. Update the django view
   `PROJECT_DIR/Web_Server/webapps/discovery/views.py` by replacing `power_meters` with `motors` on line 58 as such

   `motors = SupportedDevices.objects.filter(device_type_id=5)`

   On line 70 of the file, add the key value pair 'motors': motors to the `devices` dictionary that is rendered to the `manual_discovery.html` file.

7. In the directory, `PROJECT_DIR/Web_Server/webapps/device/templates` copy the directory `motor` and rename to `motor` and rename the html template `plugload.html` to `motor.html`.

8. In the directory, `PROJECT_DIR/Web_Server/webapps/charts /templates/charts/` copy the file `charts_plugload.html` into the directory and rename the file to `motor.html`.

Note: other steps will likely be required here due to the bug regarding the number of devices.

Wednesday, August 21, 2019

As a continuation of yesterday, further code must be added to the file
`PROJECT_DIR/Web_Server/static/app_js/manual_disc.js`
I have almost no knowledge of javascript and thus jQuery, so I do not understand this
code.
The lines
`$(".eq_height_mot").height(ht);`
and

```
    $("#mot_all").change(function() {
        if($(this).is(":checked")) {
            $("input[id^=mot_]").prop('checked', true);
        } else {
            $("input[id^=mot_]").prop('checked', false);
        }
    });
```
must be added. Also, the expression
`!(current_id == "mot_all")`
must be added to line 189 of `manual_disc.js`.

I will likely need to make a pdf file with these steps and place them in the Google Drive
for better accessibility.

Since the RPi was automatically connecting to BUGuest, I found a tutorial online to forget
the saved wifi network. By using the command,
`sudo nano /etc/wpa_supplicant/wpa_supplicant.conf`
the wifi network corresponding to BUGuest was deleted including the name and network
block. Hopefully, this will prevent the Pi from connecting to BUGuest and only connect
to ECE-Robotics1.

I found another error in the software. When clicking on the "Delete Data" button in the
BEMOSS advanced settings in the advanced settings GUI, this exception is thrown:

```
Traceback (most recent call last):
  File "/usr/lib/python2.7/lib-tk/Tkinter.py", line 1540, in __call__
    return self.func(*args)
  File "GUI.py", line 385, in del_dat
    shutil.rmtree(cas_data_dir)
```

```
  File "/usr/lib/python2.7/shutil.py", line 239, in rmtree
    onerror(os.listdir, path, sys.exc_info())
  File "/usr/lib/python2.7/shutil.py", line 237, in rmtree
    names = os.listdir(path)
```
OSError: [Errno 2] No such file or directory:
'/home/ramgroup/BEMOSS3.5/cassandra/data'

The data directory is not present in the cassandra directory. For now, to delete the data from the database, I will simply just remove the table by using the function call:

`cassandraDB.delete(agent_id)`

where `agent_id` is the `agent_id` corresponding to the cassandra database table.

Thursday, August 22, 2019

While researching sources for the paper, I found this source [1] to help me understand a little more terminology used in this project:

- Peak demand - highest average demand during a 15 period in the utility cycle

- Time-of-use pricing - electricity rate is charged based on the time of day

[1] https://lucidconnects.com/library/blog/electricity-demand-vs-consumption

Bibliography

[1] W. Khamphanchai, M. Pipattanasomporn, M. Kuzlu, and S. Rahman. An agent-based open source platform for building energy management. In *2015 IEEE PES*. 123

[2] Kaisar R. Khan, Atiqur Rahman, Adnan Nadeem, Muhammad Shoaib Siidiqui, and Rafi Ahmad Khan. Remote monitoring and control of microgrid using smart sensor network and internet of thing. In *2018 1st ICCAIS*, 2018. 29

[3] Professor Scott Moura. Cee systems analysis, 2012. 119

[4] M. Pippatanasomporn, M. Kuzlu, W. Khampahanchi, A. Saha, K. Rathinavel, and S. Rahman. Bemoss: An agent platform to facilitategrid- interactive building operation with iot devices. In *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. 23

[5] R. Adhikari X. Zhan and, M. Pipattanasomporn, M. Kuzlu, and S. Rahman. Deploying iot devices to make buildings smart: Performance evaluation and deployment experience. In *Proceedings on IEEE World Forum on Internet of Things*, December 2016. 29