Final Report for DOE/EERE

Project Title:	**Software-Defined Solutions for Managing Energy Use in Small to Medium Sized Commercial Buildings**
Covering Period:	December 7, 2013 – October 31, 2014
Approved Project Period:	November 1, 2013 – October 31, 2014
Submission Date:	Initial submission January 30, 2015, revised June 22, 2015, revised August 28, 2015, final submission September 10, 2015
Recipient:	University of California
	University of California, Berkeley California Institute for Energy and Environment 2087 Addison Street, 2nd Floor Berkeley, CA 94704
Website (if available)	http://i4energy.org/
Award Number:	DE-EE0006351
Working Partners:	UC Davis
	Lawrence Berkeley National Laboratory
Cost-Sharing Partners:	Building Robotics
PI:	Dr. Carl Blumstein Director, California Institute for Energy and Environment Phone: 510-643-1440 Fax: 510-643-9324 Email: blumstei@berkeley.edu
Submitted by: (if other than PI)	Dr. Therese Peffer Title: Academic Coordinator Phone: 510-289-4278 Fax: 510-643-9324 Email: therese.peffer@uc-ciee.org

DOE Project Team:	DOE Contracting Officer	-	Michael Buck
	DOE Project Officer	-	Jim Payne
	Project Engineer	-	Michael Wofsey

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-EE0006351

Disclaimer

Acronyms and Terms

API	Application Program Interface is a set of routines, protocols, and tools for building software applications that is typically proprietary and specific to each type of device.
BAS	Building Automation System is the automatic centralized control of a commercial building's heating, ventilation and air conditioning system as well as lighting.
BOSS	Building Operating System Services is a UC Berkeley software platform for managing building services
BOSSWAVE	Building Operating System Services Wide Area Verification Exchange is a UC Berkeley developed software that provides decentralized Web of Trust authentication and authorization security for data exchange
HVAC	Heating, Ventilation, and Air-Conditioning equipment
REST	REpresentational State Transfer is the software architecture style consisting of guidelines and best practices for developing web services. RESTful systems communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) used by web browsers to retrieve web pages and send data to remote servers
RTU	Roof top unit, packaged HVAC unit typically installed on the roofs of buildings.

Table of Contents

Introduction

Much American business is conducted in buildings with less than 50,000 square feet of floor area—about 90% of the commercial buildings in the US which represent about half of the commercial floor area. These buildings—accommodating retail stores, banks, grocery stores, drugstores, restaurants, offices; often in the same structure—also account for much of the nation's energy use. Unlike their larger brothers, they are rarely equipped with a building automation system (BAS) to manage heating, cooling, lighting, and other energy-consuming operations. Why? This has not been an attractive market for mainstream building controls companies to enter. Uses of these smaller buildings are diverse and changing, as are the types of ownership, operation, and financing. In addition, these buildings are often provided with heating, ventilation, and air conditioning (HVAC) by small multiple packaged rooftop top units (RTUs), rather than large central systems. These systems vary in age, complexity, and controllability. Energy uses in these buildings are at the same time too simple, too dispersed and too diverse — a less-than-streamlined infrastructure that makes applications of traditional BAS not cost-effective and energy management a challenge. Also, these businesses rarely have the facilities manager that a mainstream BAS is designed around, to engage with the controls at a high technical level in keeping an eye on energy use and minimizing waste.

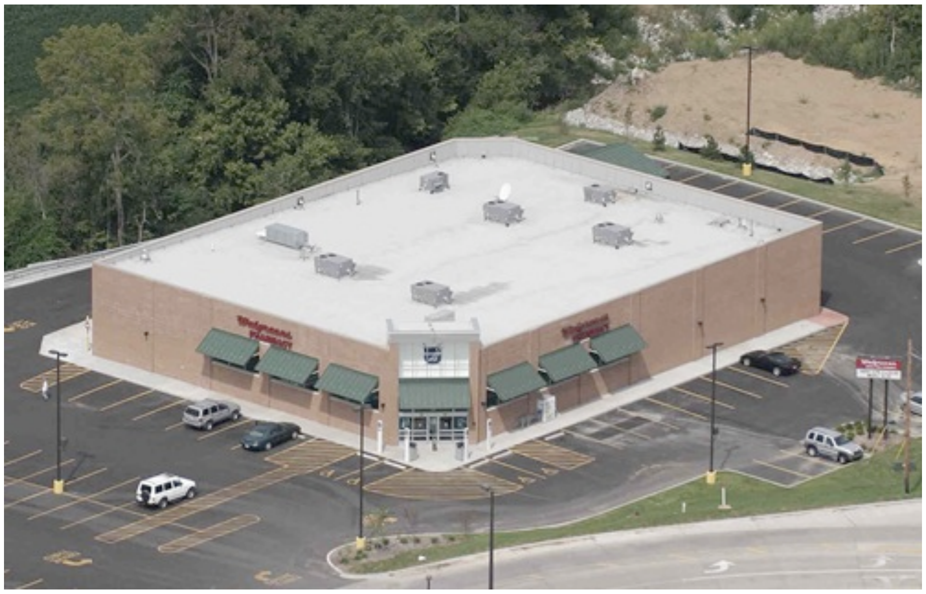


Figure 1: A typical medium-sized building with roof top unit (RTU) heating, ventilation, and air conditioning systems.

Project Objective

The Project uses state-of-the-art computer science to extend the benefits of Building Automation Systems (BAS) typically found in large buildings (>100,000 square foot) to medium-sized commercial buildings (<50,000 sq ft). The BAS developed in this project, termed OpenBAS, uses an open-source and open software architecture platform, user interface, and plug-and-play control devices to facilitate adoption of energy efficiency strategies in the commercial building sector throughout the United States.

At the heart of this “turn key” BAS is the platform with three types of controllers—thermostat, lighting controller, and general controller—that are easily “discovered” by the platform in a plug-and-play fashion. The user interface showcases the platform and provides the control system set-up, system status display and means of automatically mapping the control points in the system.

Team

University of California, Berkeley researchers led the team, with Carl Blumstein as the PI at the helm; the team included researchers from Lawrence Berkeley National Laboratory (LBNL) and UC Davis. The Software Defined Buildings group at UC Berkeley, led by Professor David Culler, led the software platform development. The Western Cooling Efficiency Center at UC Davis, led by Professor Mark Modera, drove the thermostat control development and algorithm development, aided by Professor Dave Auslander of UC Berkeley, Mechanical Engineering. Alan Meier and Rich Brown of LBNL led the lighting control development. Stephen Dawson-Haggerty of Building Robotics supported the platform development and initiated the auto discovery service development.

Background

OpenBAS is the “software-defined building” with a flexible multi-service open source software architecture developed at UC Berkeley that dramatically reduces the effort to add new functions/applications and supports sensor and actuator access, access management, metadata, archiving, and discovery, as well as multiple simultaneously executing programs. The primary innovation is the open software architecture compared to proprietary “closed” BAS. Typical services in buildings are vendor-specific; in large buildings, these systems might be networked, but typically communicate using different protocols. For example, an HVAC system has its own user interface, schedule, data management, sensors and actuators. In turn, the lighting system has a schedule, occupancy and lighting sensors, and controls. Likewise, a security system may have a separate set of occupancy sensors, schedule, and user interface. These are often termed stove-pipe, siloed, or vertically integrated.

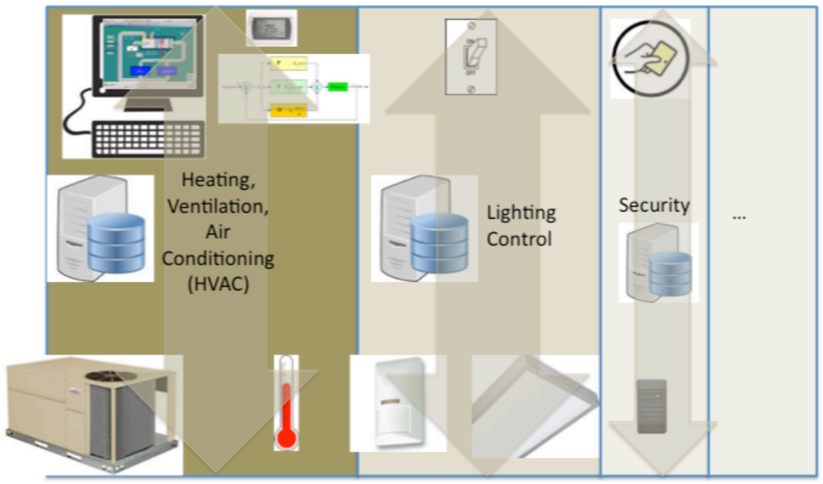


Figure 2: Proprietary building services typically cannot communicate with each other, showing a vertical integration.

By contrast, the OpenBAS platform is open-source (publicly available) through the BSD 2-Clause license and has an open software architecture, constructed from horizontal layers. This architecture allows various system components, no matter what protocol each uses, the ability to communicate, thus improving operations and reducing redundancy. For example, a single schedule can control HVAC, lighting, and plug-loads. An occupancy sensor for the lighting system may be used to control the HVAC system. Networked thermostats can allow optimization strategies to reduce energy consumption while still providing comfort and fresh air. Third party vendors of both hardware devices and software applications can connect their wares to the platform, regardless of the physical interface, maximizing its effectiveness and market value. For example, to connect new hardware requires both a physical (e.g., wired or wireless) and logical connection (e.g., translates and manages data to be useful to applications, such as an Application Programming Interface (API)). Each hardware device, whether a new sensor or actuator requires a sMAP driver to interface with the OpenBAS platform. The platform was deliberately designed to ensure that drivers would be simple and easy to write initially, and eventually become part of the hardware itself. Likewise, a software product vendor, such as a dashboard or diagnostics application, can easily view and query the database or write simple control scripts to actuate hardware. This creates an industrial ecosystem for innovation and rich application development. The layers provide modularity; the horizontal nature speaks to integration and interoperability of various devices and systems.

Figure 3 below shows the layered architecture, and highlights the required software platform, user interface, three software tools (system set-up, status display, and auto-mapping), and three hardware devices (thermostat, lighting controller, and general controller). The OpenBAS architecture is horizontal, layered, and open, providing a modularity that allows nimble adaptation to changes at all layers. The bottom layer contains interfaces to physical devices, typically sensors and actuators. The middle layer links the sensors and actuators with the top (application) layer through the sMAP (simple Monitoring and Actuation Profile) interface and services. sMAP makes diverse sources of physical data available for applications such as control, visualization, and fault detection, and is agnostic to physical interface (e.g., WiFi, ZWave, ZigBee, Ethernet). sMAP drivers allow sensors to “publish” data to services and applications via the database and allow applications to “publish” commands to actuators. The platform supports multiple simultaneously executing programs. Various new components can be automatically discovered and added to the network.

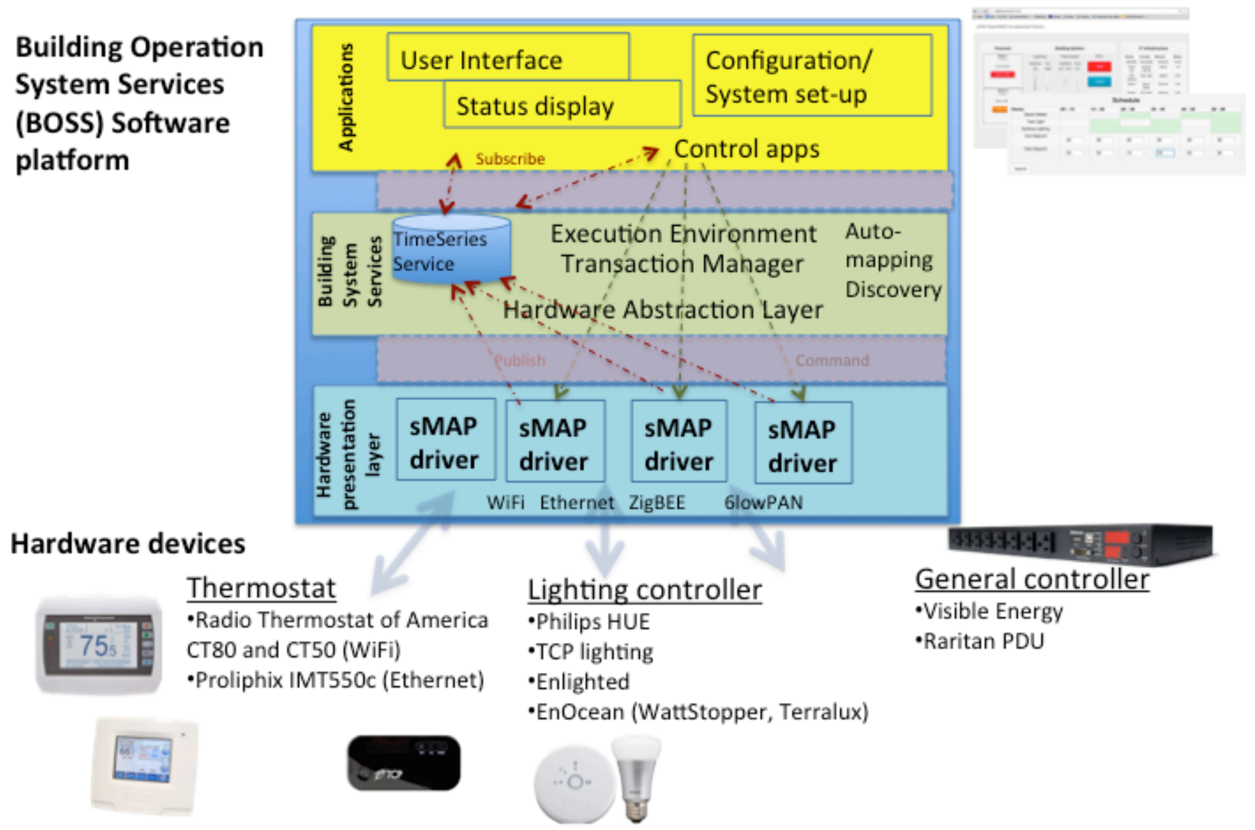


Figure 3: The OpenBAS layered architecture, consisting of Hardware devices, a Hardware Presentation Layer, Building System Services layer, and an Application layer.

The platform was developed and tested in a laboratory setting, in a bench-top setting (Plexiglass two-zone structure), and in a 7500 sf floor of a building in Berkeley, California.

Task by Task Description

Task 1: Integrated platform with software tools, user interface, and hardware devices.

Subtask 1.1: Develop open architecture software platform prototype (BOSS) for small to medium commercial buildings.

The Software Defined Buildings research group at UC Berkeley expanded and refined the BOSS platform built on the sMAP open source information infrastructure for buildings and grids.

Initially developed by Dawson-Haggerty, sMAP (http://code.google.com/p/smap-data) is 1) a universal information representation for physical data (self-describing, compact JSON schema, transportable over UDP/TCP, with integrated metadata; 2) software architecture for physical data processing and actuation (Real-time and archival data, time-series DB and adapters/drivers for legacy and direct streams); 3) subscription, syndication, distillates and query processing, visualization interface; and 4) a resource-oriented web-service framework for embedded applications.

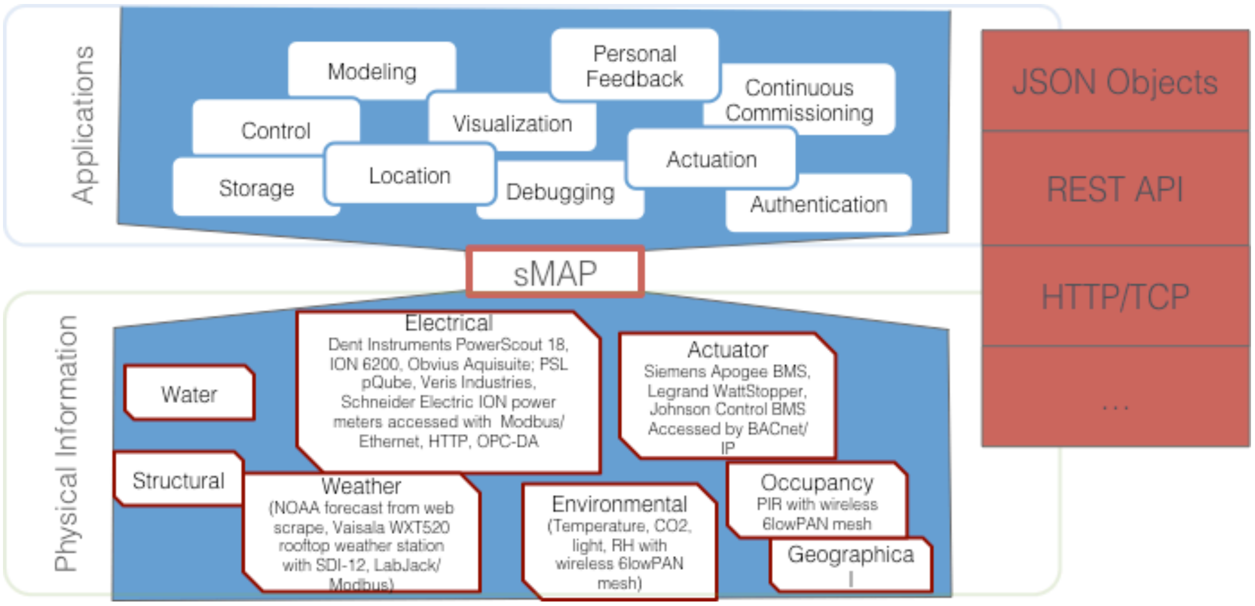

Applications
Modeling
Personal Feedback
Continuous Commissioning
Control
Visualization
Actuation
Storage
Location
Debugging
Authentication
sMAP
JSON Objects
REST API
HTTP/TCP
...
Physical Information
Water
Electrical
Dent Instruments PowerScout 18, ION 6200, Obvius Aquisuite; PSL pQube, Veris Industries, Schneider Electric ION power meters accessed with Modbus/ Ethernet, HTTP, OPC-DA
Actuator
Siemens Apogee BMS, Legrand WattStopper, Johnson Control BMS Accessed by BACnet/ IP
Structural
Weather
(NOAA forecast from web scrape, Vaisala WXT520 rooftop weather station with SDI-12, LabJack/ Modbus)
Environmental
(Temperature, CO2, light, RH with wireless 6lowPAN mesh)
Occupancy
PIR with wireless 6lowPAN mesh
Geographical

Figure 4: Simple Monitoring and Actuation Profile allows data aggregation from various sources.

The hardware presentation layer, shown in Figure 3, includes drivers for the OpenBAS demonstration devices (network thermostats, lighting control, general control), along with more than fifty other open source drivers (in https://github.com/SoftwareDefinedBuildings/smap/tree/master/python/smap/drivers) for energy metering, demand response notification, BACnet, Modbus, commercial BMS systems, weather metering, thermal monitoring, air quality monitoring, and so on. We have also developed a family of virtual sensor and actuator drivers to (1) allow for development and prototyping of higher level services and application in a stand-alone environment and (2) connect OpenBAS with simulators and other analytical tools.

Sample driver

To offer an example of how simple it is to create *drivers* for available commercial devices in our OpenBAS platform, below we provide a sample thermostat driver. A driver class contains three methods: `setup, start`, and `read`.

- Typically `setup` collects configuration parameters and initializes driver state. Here it provides a mapping of all the points offered by the device and creates a timeseries for each of these points. These become resources in the RESTful webservice associated with the config file declaring the driver interface to a particular physical device.
- The `start` method declares how this driver is to be scheduled; here, periodically.
- The `read` method accesses the hardware device and publishes the result to a stream, whereupon it can be utilized in a manner consistent with all other streams. For the CT80 the device access is fairly trivial because it is a networked device with an REST API, although even here the device presents its humidity sensor quite differently from its temperature sensor. In generaly, the driver can provide a systematic interface for a variety of particular products. For the numerous legacy devices of interest, the read

method is used to access the underlying hardware resource, whether it be a serial port, BACnet, Modbus, or one of the many other forms of connection in the market.

```
class CT80(SmapDriver):
    def setup(self, opts):
        self.tz = opts.get('Timezone', 'America/Los_Angeles')
        self.rate = float(opts.get('Rate', 1))
        self.ip = opts.get('ip', None)

        self.points0 = [
                         {"name": "temp", "unit": "F", "data_type": "double"},
                         {"name": "tmode", "unit": "Mode", "data_type": "long"},
                         {"name": "fmode", "unit": "Mode", "data_type": "long"},
                         {"name": "override", "unit": "Mode", "data_type": "long"},
                         {"name": "hold", "unit": "Mode", "data_type": "long"},
                         {"name": "t_heat", "unit": "F", "data_type": "double"},
                         {"name": "program_mode", "unit": "Mode", "data_type": "long"}
                       ]
        for p in self.points0:
            self.add_timeseries('/' + p["name"], p["unit"], data_type=p["data_type"],
timezone=self.tz)

        # points not in the root resource
        self.add_timeseries('/humidity', '%RH', data_type="double")

    def start(self):
        # call self.read every self.rate seconds
        periodicSequentialCall(self.read).start(self.rate)

    def read(self):
        r = requests.get('http://' + self.ip + "/tstat")
        vals = json.loads(r.text)

        for p in self.points0:
            self.add('/' + p["name"], vals[p["name"]])

        r = requests.get(url + '/humidity')
        val = json.loads(r.text)
        self.add('/humidity', val['humidity'])
```

Choice of Information Bus

A critical area of investigation in the OpenBAS project has been the requirements of the information bus connecting the Hardware Presentation Layer to the Building Systems Services tier. The sMAP infrastructure provides an intentionally "lean" information bus. Collections of devices (interfaced through drivers) present themselves as a REST web services resource hierarchy, called an sMAP source. That is, each resource presents itself as a Uniform Resource Identifier (URI) (i.e., `http(s)://<hostname>:<port>/<resource path>`) with properties and metadata presented at each resource in the tree, inherited down the path as defined by the configuration (.ini) file for the sMAP source.

Resources, services, and applications can access a resource directly through its URI. Sources also can report to one or more archivers through a subscription UUID. The sMAP archiver, as part of the service tier, provides syndication and publication through its *republish* API. This allows resources to subscribe to events on sets of streams identified declaratively through a fully general 'where clause' in the sMAP query language. Such restrictions provide the full generality of the resource metadata in constructing relationships among services and streams, but provide clear accountability and tracking of the flows of information for reliability.

In particular, in the embedded systems context, we found that heavy-weight pub/sub infrastructures developed for the Enterprise provided little utility and introduced a great deal of complexity. For example, XMPP is designed to carry payloads such as videos, audio streams, and video game avatars and interactions, whereas what is required in OpenBAS is temperature readings, control actions and other “tiny” notifications. Furthermore, message brokers, such as ZeroMQ, provide a low-level socket-like interface with the connections between endpoints being facilitated by a broker based on a complex match on an intermediate topics namespace. While extremely general, these permit arbitrary interactions, utilize none of the natural structure of the building automation setting, and have no built-in connections to data historians, real-time control, and visualization, which are so inherent to OpenBAS.

In earlier stages of the project we carried out an integration into sMAP of a recent, open source pub/sub infrastructure designed for embedded systems, MQTT, which is being utilized within the building systems industry. In this approach, the resources paths provided by the sMAP framework become natural topics for the broker to manage, while also providing a reporting connection to the historian. In this context, we worked on the Authorization and Authentication Service, using a distributed system for transferring trust (using dot = declaration of trust), or trusted subsystems. We have termed this the BOSS Wide Area Verifed Exchange (BOSSWAVE). Simple ubiquitous security such as SSL provides a means of encrypting traffic between a user and a server. However, SSL does not provide protection if the server is compromised, as for example in Target Corporation’s security breach. BOSSWAVE seeks to manage keys so that every transaction can be verified. It is a distributed key management scheme so that anyone can verify a requestor.

In developing the OpenBAS demonstration application, UI, and status, we have sought to maintain austere simplicity. Thus, we have set aside, as least for this stage of the project, general purpose pub/sub brokers and built the complete, integrated HVAC, lighting, and general control directly on the syndication capabilities on the sMAP archiver. This reduces overall code complexity and dependencies while providing the simple, accountable structure more familiar to the building systems industry.

Building System Services

The OpenBAS Building System Services tier provides a family of services (i.e., continually running, monitorable computational processes) that operate upon the driver services in support of applications. A prime example is the Discovery service. This includes a hardware-level driver to provide notification of connection events of new physical hardware devices, but its main function is to recognize the particular device that has been attached to the Building Area Network and configure that device for use in the particular site. Further detail on the discovery service is provided in Subtask 1.3, page 9. We have developed several other services to support the OpenBAS platform, including controller services, scheduler services and others. These services use the same execution container as drivers to provide continuous, monitored operation, but operate logically at a higher level.

Sample service

A redacted illustration of such a controller service is shown below to indicate how simple these are to create. This one could be instantiated with a configuration clause such as the following, which has the controller and the archiver running on the same host, which could be local to the premises or remote:

```
[/control]
type=coolControllerService.Controller
deadband=2
rate=1
archiver_url=http://localhost:8079
zonepath = /room/airtemp
zonesiteid =  2c8ed966-146a-11e4-9e46-000c29b778da
```

To connect the coolControllerService to the /room/airtemp resource at a particular site while configuring its initial state:

```
class Controller(driver.SmapDriver):
    def setup(self, opts):                         # configure the controller service
         < collect args from config and initialize controller state – removed >
        # subscribe to zone air temperature
        self.roomclient = RepublishClient(self.archiver_url, self.controlcb,
                   restrict=”Path={} and Metadata/Site/id = {}”.format(path,siteid)
        # create timeseries for contoller actions
        self.add_timeseries('/cool', 'On/Off', data_type='long')

    def start(self):                                # start the controller service
        self.roomclient.connect()
        periodicSequentialCall(self.read).start(self.rate)

    # Periodically schedule controller: take action, update state, publish event
    def read(self):
        if (self.cur_temp > self.sp + self.db) : self.state = 1  # cool on
        if (self.cur_temp < self.sp - self.db) : self.state = 0  # cool off
        # otherwise float in the current state, cooling or not until deadband is crossed
        self.add('/cool', self.state) # publish the state even when no change

    # Handle temperature reporting event, here just report state
    def controlcb(self, _, data):
        mostrecent = data[-1][-1]
        self.cur_temp = mostrecent[1]
```

The `setup` method within the Controller class of the coolControllerService module subscribes to the /room/airtemp resource (specifically, temperature resource declared the configuration). The `start` method initates the connection this subscription and schedules the `read` method periodically. In this example, `controlcb` (the event call back) will be invoked whenever a new temperature value is published to the /room/airtemp stream. Here, we simply update the controller state with the most recent temperature reading for use in the next control epoch. A periodic control strategy is used which publishes an event to the /control/cool stream based on whether the temperature in the room has moved outside the deadband around the setpoint.

The main point here is that the control algorithm is clearly described in a few lines of understandable code. The OpenBAS framework provides all the rest as described above—the configuration and the orchestration and management of data flow (e.g., initiation of connection, scheduling the reading of data, invoking controller).

Note that here we have essentially implemented a virtual thermostat for an air conditioner. It could be much more sophisticated, say controlling both heating and cooling, implementing a PID (Proportional-Integrative-Derivative) control loop or MPC (Model Predictive Control). Moreover, if this were a virtual thermostat controlling one or more networked thermostats, each providing an internal RTU (RoofTop Unit) controller, the event stream would be setpoints published to those themostats. Such a controller could adjust for differing requirements in different zones or relative offsets among zones or integrate multiple temperature sources into the higher level controller loop.

Other services in the middle Building System Services tier include schedulers, such as one that maintains a day-by-day weekly schedule to specify operations of multiple thermostats, lighting groups, and general control state for morning, daytime, evening, and night epoch on various classes of days, e.g., weekdays, weekends, and holidays. Sophisticated schedules can be created that integrate multiple systems. Other services provide analytics, model building, energy analysis and so forth.

We implemented version control of configuration files as a fail safe for future improvement upgrades.

The OpenBAS sMAP system has been fully integrated into the Ubuntu and Debian package manager. On a standard installation, the entire server infrastructure is downloaded and built through a simple administrative action:

```
sudo add-apt-repository ppa:cal-sdb/smap
```

followed by some straightforward setup.

Subtask 1.2: Procure Off-The-Shelf hardware devices; develop interface to BOSS
The LBNL, UC Davis, and UC Berkeley teams completed the lists of OpenBAS-appropriate hardware devices (lighting control, thermostats, and plug load controls respectively), functions and physical/logical interfaces. The criteria were: appropriate functionality, commercially available, reasonable cost, and open communication protocol (e.g., available API or other means of communicating with the device such as a web interface).

Two thermostats were procured: Radio Thermostat of America CT80 (WiFi interface) and Proliphix IMT550c (Ethernet).

Several lighting controllers were obtained as well: Phillips HUE and TCP lighting (controlled light bulbs) and the Enlighted System (uses an Energy Manager connected to a gateway that wirelessly transmits to a controller/occupancy sensor physically connected to lighting control for LED fixture or fluorescent ballast). We also obtained parts of the EnOcean Alliance system: WattStopper wireless light switches and occupancy sensors and Terralux LED lamp replacement for pin CFL hanging pendant fixtures. EnOcean uses a proprietary gateway, which is a simple USB stick that communicates to devices at 902 MHz. All lighting controls use proprietary gateways; however, both EnOcean and Enlighted provided APIs for their gateways, allowing OpenBAS easy access for sensing and control.

The three plug load controllers we worked with were the WiFi-enabled VisibleEnergy plugstrip and monostrip (single outlet), the Ethernet-enabled Raritan and Echola Power Distribution Units (monitor and plugstrip).

The team wrote sMAP drivers for the RTA CT80, the Proliphix IMT550c, Phillips Hue, TCP lighting, LabJack (controlled an LED light strip and received temperature/humidity data), a VisibleEnergy plugstrip and monostrip. In addition, we worked with Enlighted, who made the API available to us, and wrote drivers for Enlighted and Enocean WattStopper and Terralux.

Manufacturer	*Device*	*Description*	*Connection protocol*
Radio Thermostat of America	CT80	Thermostat	WiFi
Radio Thermostat of America	CT50	Thermostat	WiFi
Proliphix	IMT550c	Thermostat	Ethernet
Proliphix	NT160e	Thermostat	Ethernet
Philips	HUE	LED light bulb control, color and dimming	Zigbee (Proprietary gateway)
TCP	Connected	LED light bulb control, dimming	WiFi (Proprietary gateway)
Enlighted	Energy Manager	controls LED or fluorescent ballasts, dimming, uses occupancy and light level sensors	Zigbee light (proprietary gateway)
Wattstopper	EOSW-111	single pole light switch	EnOcean 902MHz (USB stick)
Wattstopper	EOSW-112	bi-pole light switch	EnOcean 902MHz (USB stick)
Terralux	DRV	LED retrofit for pin-CFLs, dimming	EnOcean 902MHz (USB stick)
Raritan	Power Distribution Unit	8 individually monitored and controlled receptacles	Ethernet
Echola	Power Distribution Unit	6 individually monitored and controlled receptacles	Ethernet
Visible Energy	monostrip	single monitored and controlled receptacle	WiFi
Visible Energy	UFO power center	4 individually monitored and controlled receptacles	WiFi
EnOcean		single receptacle relay	EnOcean 902MHz (USB stick)
Rainforest Automation	Eagle energy monitor	Home Area Network device to read interval utility meter data	Zigbee
Korea Electronic Technology Institute	Environmental sensors	Temperature, Light, Relative humidity, Carbon Dioxide, Passive InfraRed	6loWPAN mesh network (USB mote)

Table 1: List of hardware devices that interfaced with the platform through sMAP drivers.

We also explored how these devices will be controlled by the OpenBAS platform. In general the devices as interfaced with the OpenBAS platform met the general test specifications and capabilities of the FOA. For example, typical of thermostatic control, the commands to turn on HEAT, COOL for thermostats are NOT directly accessible from outside the thermostat itself. The setpoints are used to control the thermostat, whether on the device itself or through the OpenBAS platform.

Subtask 1.3: Develop software tools (system setup, status display, and auto-mapping/discovery)

Discovery

Building Robotics and UC Berkeley wrote the initial Discovery service, which automatically detects new devices on the network, finds and installs the appropriate sMAP driver, and configures it to that particular installation. UC Berkeley continued to develop this service further. This discovery service is similar to “plug and play” as it appears in various forms in consumer markets, such as plugging in a new hardware device into a PC or laptop. However, in recognition of the state of the industry relevant to OpenBAS and the product development lifecycles, it does not depend on vendor products implementing a particular discovery standard. The discovery service receives notification of an attachment of a new device, it probes the device using a collection of detection scripts to identify what it is; once identified, the service pulls in the appropriate driver for the device, creates the configuration file integrating the driver and the particular site, and creates an sMAP source.

The Discovery service has two repositories: detectors (small pieces of script that indicate a type of device, typical of plug and play) and sMAP drivers (continually running processes) that are cached on the local OpenBAS server. First, the specific hardware (e.g., an RTA CT80 thermostat) is connected (Step 1) to the OpenBAS Building Area Network (BAN), which may be dedicated or the existing building Local Area Network (LAN). It may include Ethernet, WiFi, or various other links. (If WiFi, vendor-provided methods are used to join, for eample, WiFi thermostats typically present themselves as a WiFi access point to allow an installer to connect over a smartphone or laptop and add it to a desired SSID.)

The Discovery service is constantly monitoring the BAN, and (Step 2) sees the device on the network (using DHCP logs, or Bonjour, Apple’s version of Zeroconf (http://en.wikipedia.org/wiki/Bonjour_software)), and runs through the nmap scripts in the detector repository to “discover” what the device is, locate its driver, and configure it. The matching script produces an sMAP “.ini” file (Step 3), a description of a web resource that identifies the driver and all the metadata associated with configuring it. The appropriate sMAP driver for the device is auto-loaded; that is, an instance of the driver is loaded from the cache onto the local OpenBAS server.

We conducted a functional test in the lab as well as in the field and successfully loaded appropriate drivers on several devices.

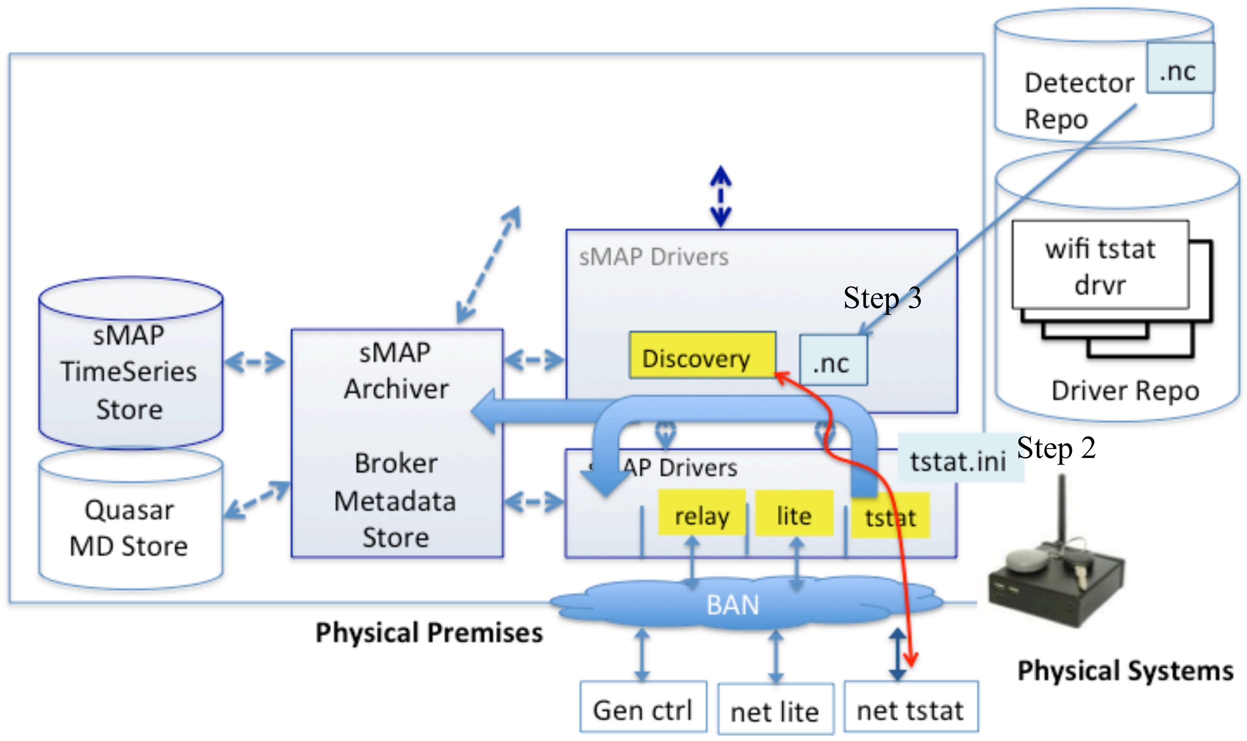


Figure 5: A schematic of the Discovery service.

Status Display

A simple user interface and application was created for the integrated control of HVAC, building lighting, and general control, such as a plug load, including task lighting and personal environments (e.g. space heaters, and independent monitoring networks), as well as zone control/display and whole building control/display. The user interface shown in the next subsection below in Figure 9 displays many types of information. The pane in the upper left portion shows the whole building energy from the interval meter. The middle left displays the schedule and current period. The bottom left shows personal devices controlled by a general plug load controller, such as task lamp, computer monitor or fan. The middle portion of the interface displays the status, current temperature and setpoints for the thermostats. The right pane displays lighting control, with indication of whether the light is on or off. Each device has a drill-down window which provides even more information.

The user interface also shows the status of each device: whether it has been automapped or configured (e.g., has had added information by the user) or not. One can see the actual driver code, and see the "health" of each device (e.g., when the system last detected the device on the network).

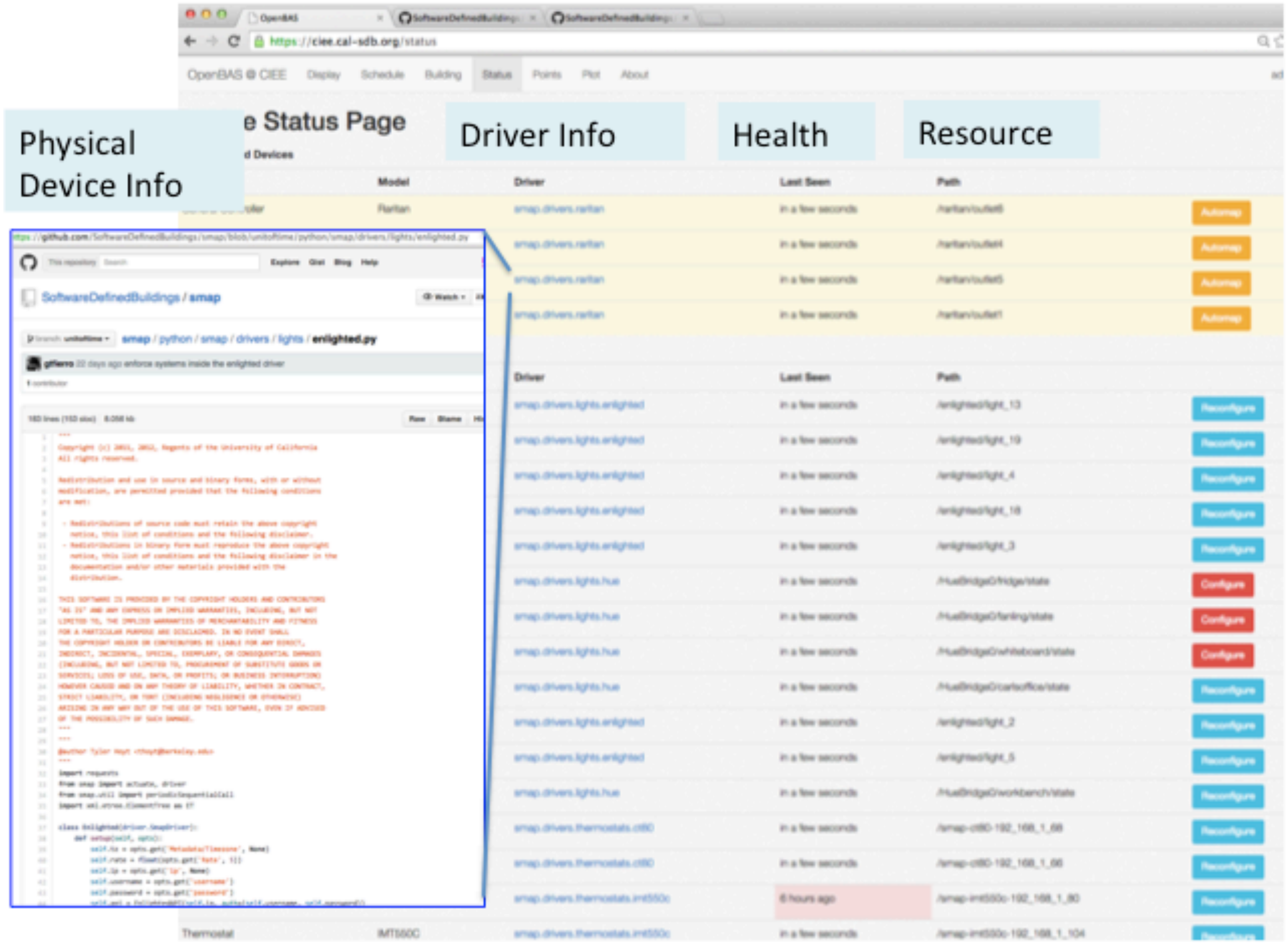
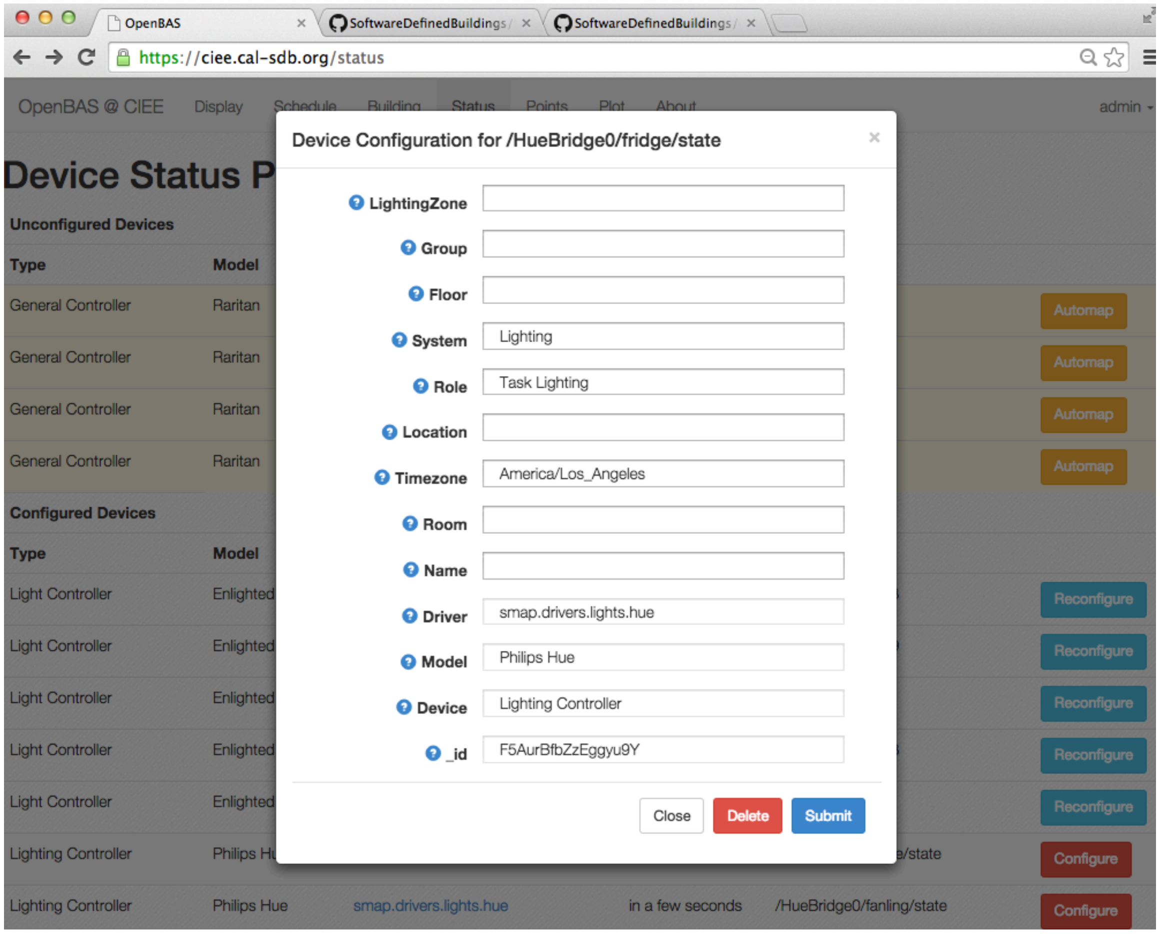


Figure 6: Status of each device.

Configuration and setup

The first step of the initial configuration and system setup of the OpenBAS platform is installing the devices, whether thermostats, lighting controls, or general controls. Once the device is installed, it is discovered by the system. The discovery automatically loads the appropriate driver and populates the user interface with information about the device (see figure below). The user is thus prompted to add information about the device.



Device Configuration for /HueBridge0/fridge/state

LightingZone	
Group	
Floor	
System	Lighting
Role	Task Lighting
Location	
Timezone	America/Los_Angeles
Room	
Name	
Driver	smap.drivers.lights.hue
Model	Philips Hue
Device	Lighting Controller
_id	F5AurBfbZzEggyu9Y

Unconfigured Devices

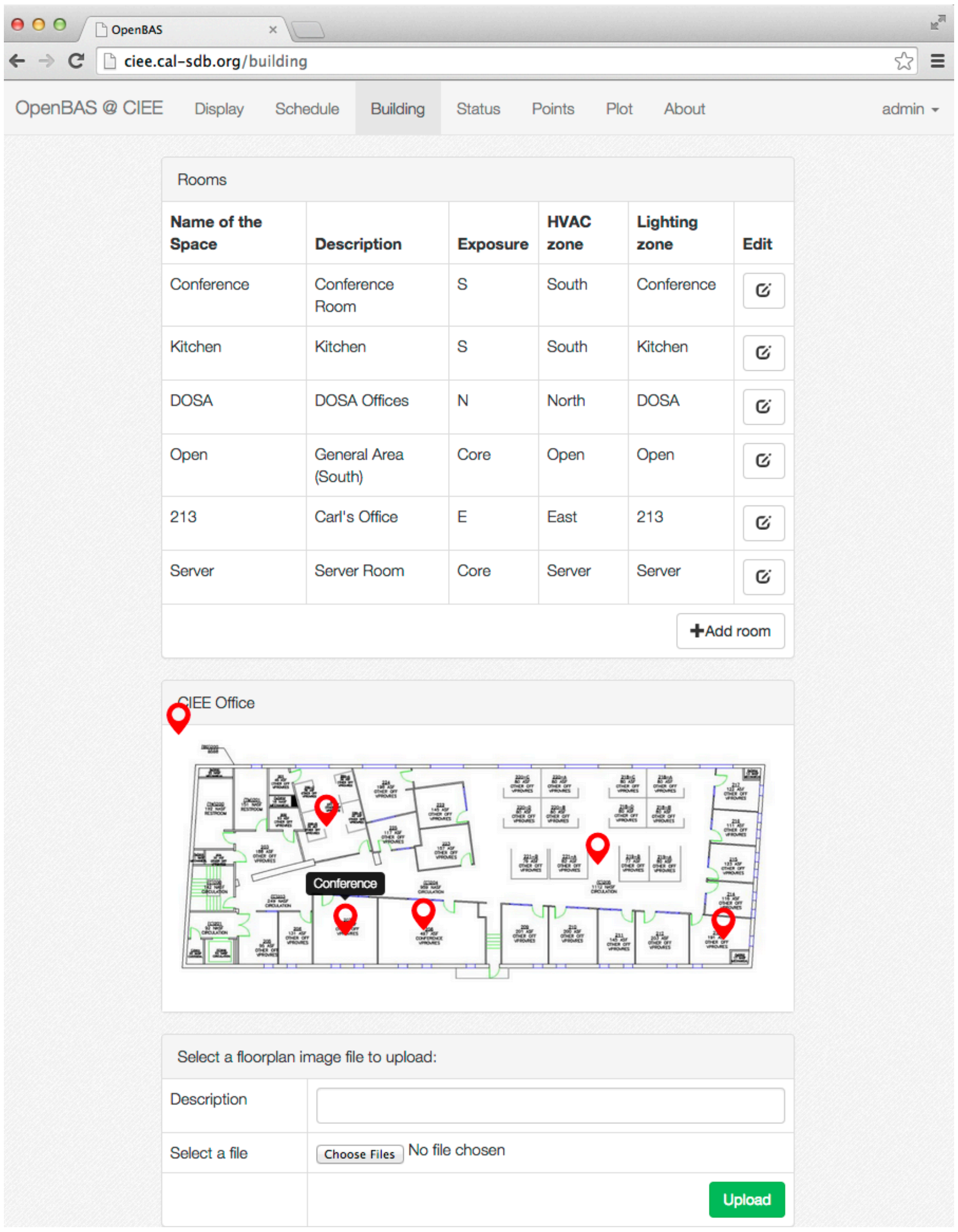
Type	Model
General Controller	Raritan
General Controller	Raritan
General Controller	Raritan
General Controller	Raritan

Configured Devices

Type	Model			
Light Controller	Enlighted			
Light Controller	Enlighted			
Light Controller	Enlighted			
Light Controller	Enlighted			
Light Controller	Enlighted			
Lighting Controller	Philips H			e/state
Lighting Controller	Philips Hue	smap.drivers.lights.hue	in a few seconds	/HueBridge0/fanling/state

Figure 7: Automapping each device provides a prepopulated field for the user to add additional information.

The user can also locate the device on an imported plan of the building as shown below.



Rooms

Name of the Space	Description	Exposure	HVAC zone	Lighting zone	Edit
Conference	Conference Room	S	South	Conference	
Kitchen	Kitchen	S	South	Kitchen	
DOSA	DOSA Offices	N	North	DOSA	
Open	General Area (South)	Core	Open	Open	
213	Carl's Office	E	East	213	
Server	Server Room	Core	Server	Server	

Select a floorplan image file to upload:

Description	
Select a file	Choose Files No file chosen

Figure 8: Configuration of devices shown on an imported plan of the building.

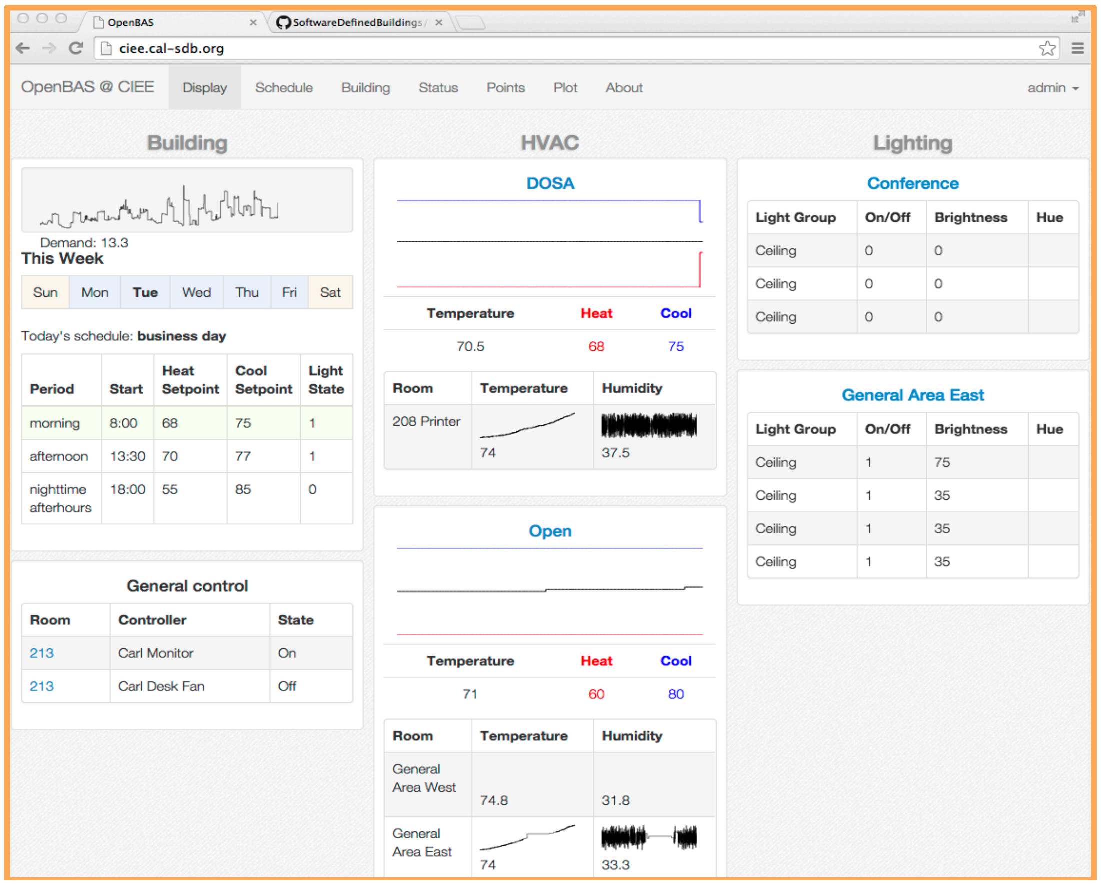
Part of the initial configuration and system setup of the OpenBAS platform typically involves defining a programmable schedule that dictates the behavior (on/off) of the subsystem components: thermostats, lighting controllers, and general load controllers. The Schedule part of the user interface shown below provides a graphical interface for constructing and monitoring schedules. This illustrates the transactional nature of our OpenBAS: initiation of a schedule supercedes the configured state of the integrated systems, which are restored upon termination of the schedule. While the schedule is running, the control pane provides visualization and can provide manual overrides of the current epoch in the schedule. The Master Schedule can provide a schedule for all devices and zones; however, each device can have a separate schedule that is not overruled by the Master. For example, in the test building, the server room is not occupied, and so its schedule does not change with occupancy the way the rest of the building schedule changes. In addition, occupants may override the Master schedule on a temporary basis.

Another configuration tool developed is a two-dimensional graphic tool that allows one to create “blocks” or zones that can represent spaces in a building. These blocks can be associated with various hardware controllers or sensors, and may be used for the initial installation of the platform in a building.

Subtask 1.4: Develop simple user interface

We developed a simple user interface for integrated control through a web browser. This demonstrated end-to-end and integrated functionality of our OpenBAS. We explored using Unity, a gaming development software tool, as a user interface to this platform. This tool allowed the creation of a 3-D representation of a building and provided both a means of controlling hardware and also a simulation tool to visualize thermal and lighting properties in accelerated time.

In the final user interface, we prototyped and evaluated open source web frameworks for providing dynamic, interactive user interfaces. The interface described below is created using the Flask open source micro-framework, as this is better suited to embedded settings than more traditional frameworks such as Rails and Django, AJAX and javascript. Based on this experience and the requirements of consistent integration of actuation, visualization, and UIs we have prototyped a redesign of key portions of this in the open source Meteor framework (https://www.meteor.com/) with the node.js event-based tools for server and client side integrated scripting.



Building

Demand: 13.3

This Week

Sun	Mon	Tue	Wed	Thu	Fri	Sat

Today's schedule: **business day**

Period	Start	Heat Setpoint	Cool Setpoint	Light State
morning	8:00	68	75	1
afternoon	13:30	70	77	1
nighttime afterhours	18:00	55	85	0

General control

Room	Controller	State
213	Carl Monitor	On
213	Carl Desk Fan	Off

HVAC

DOSA

Temperature	Heat	Cool
70.5	68	75

Room	Temperature	Humidity
208 Printer	74	37.5

Open

Temperature	Heat	Cool
71	60	80

Room	Temperature	Humidity
General Area West	74.8	31.8
General Area East	74	33.3

Lighting

Conference

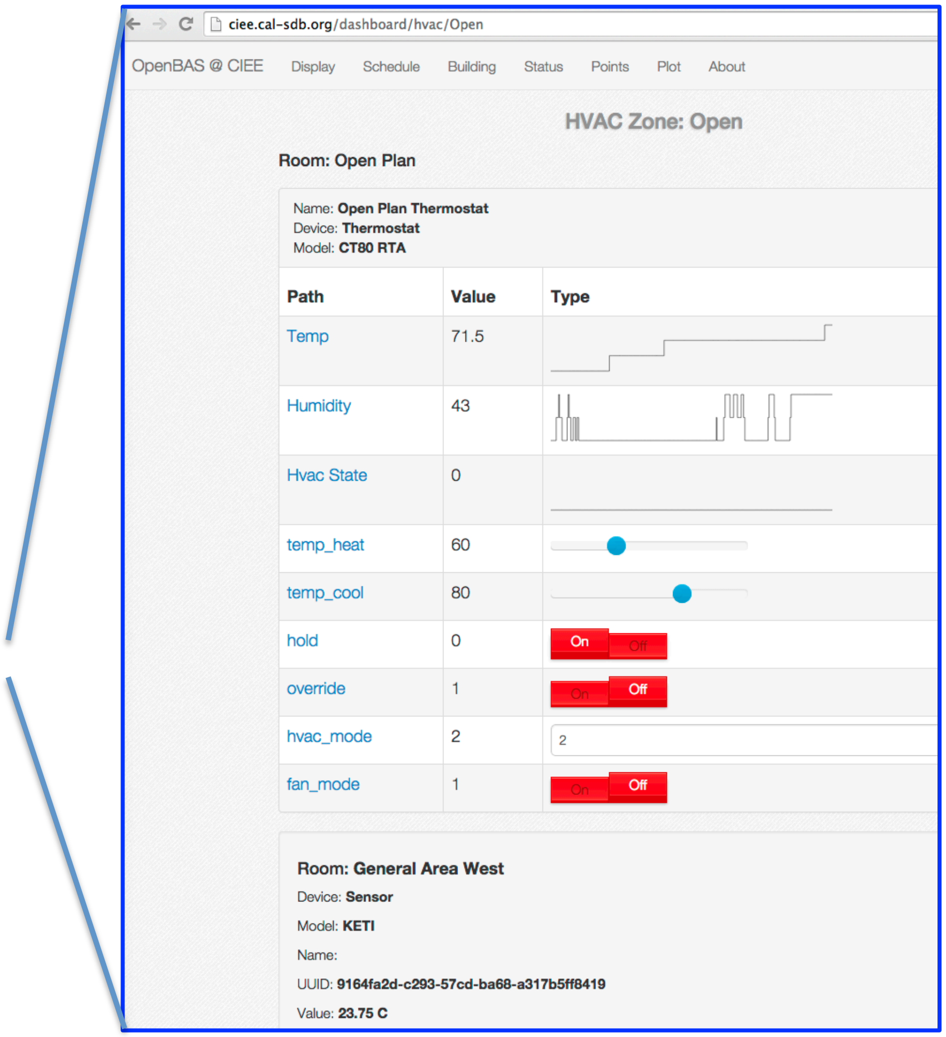
Light Group	On/Off	Brightness	Hue
Ceiling	0	0	
Ceiling	0	0	
Ceiling	0	0	

General Area East

Light Group	On/Off	Brightness	Hue
Ceiling	1	75	
Ceiling	1	35	
Ceiling	1	35	
Ceiling	1	35	

Figure 9: User interface for the building in Berkeley, showing live data from whole building meter, temperature, setpoint, lighting control and general control.

The following figure shows the additional display and control by selecting one of the devices, in this case the thermostat in the open plan portion of the building.



HVAC Zone: Open

Room: Open Plan

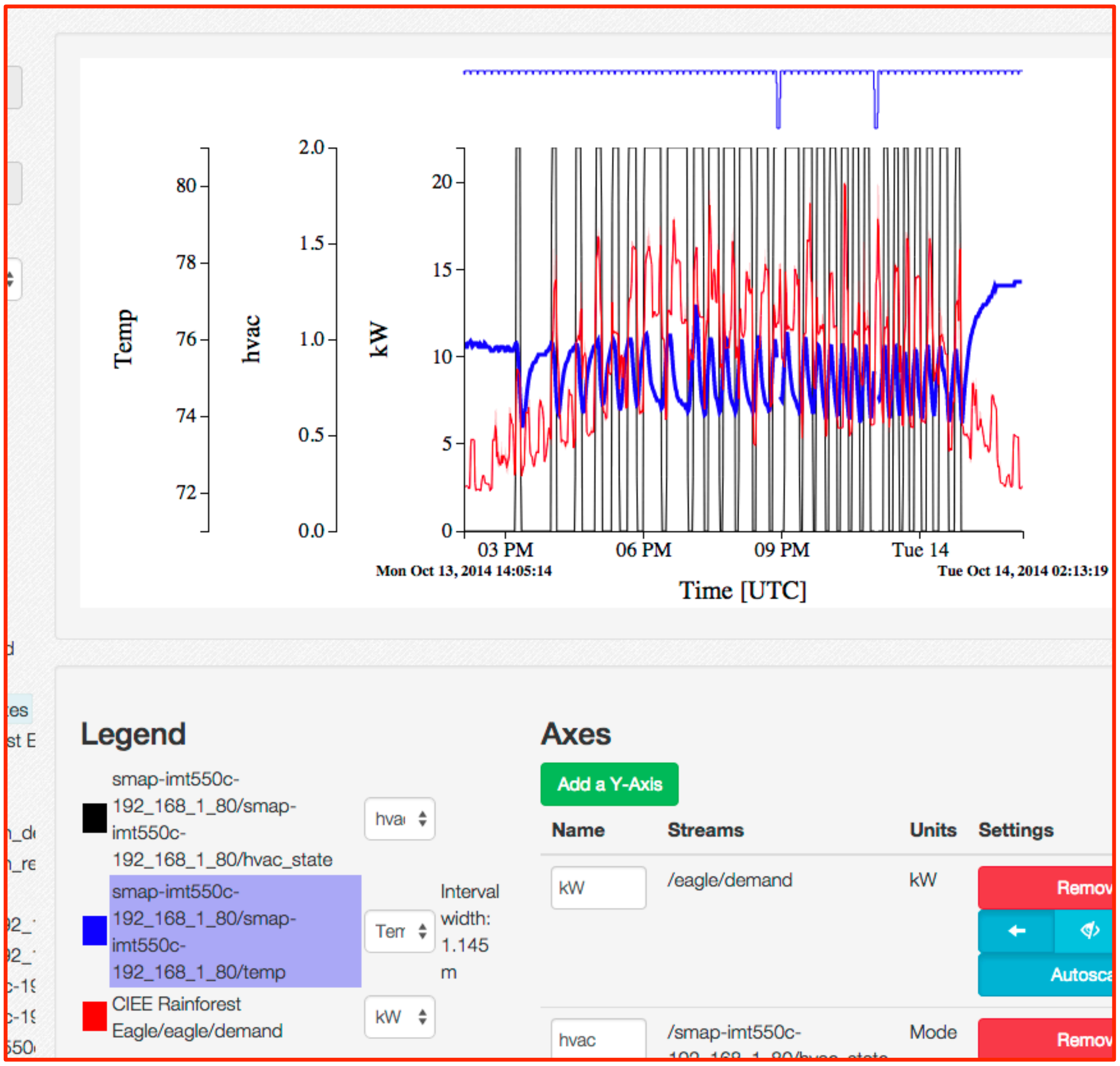
Name: **Open Plan Thermostat**
Device: **Thermostat**
Model: **CT80 RTA**

Path	Value	Type
Temp	71.5	
Humidity	43	
Hvac State	0	
temp_heat	60	
temp_cool	80	
hold	0	On Off
override	1	On Off
hvac_mode	2	2
fan_mode	1	On Off

Room: General Area West
Device: **Sensor**
Model: **KETI**
Name:
UUID: **9164fa2d-c293-57cd-ba68-a317b5ff8419**
Value: **23.75 C**

Figure 10: Drilling down into more detail of the open plan thermostat.

A plotting tool was added to show trend data from various devices. For example, the following figure shows the indoor temperature, the HVAC status (e.g., on or off) and the whole building power consumption.



Legend

⬛	smap-imt550c-192_168_1_80/smap-imt550c-192_168_1_80/hvac_state	hva
🟦	smap-imt550c-192_168_1_80/smap-imt550c-192_168_1_80/temp	Ten
🟥	CIEE Rainforest Eagle/eagle/demand	kW

Interval width: 1.145 m

Axes

Name	Streams	Units	Settings
kW	/eagle/demand	kW	Remov
hvac	/smap-imt550c-	Mode	Remov

Figure 11: Plotting tool showing trended data, such as current temperature, HVAC state, and whole building power consumption.

Task 2: Market delivery strategy

Subtask 2.1 Initial market delivery strategy plan

The ultimate success of this technology is not merely low-cost functionality, but also eventually success in the market: market penetration (commercial building owners/tenants using the platform and saving energy; building occupants and managers alike able to easily use the platform), hardware device manufacturers providing an easy interface to their devices (such as providing access to their APIs), and software developers easily able to add applications (whether visualization, analysis, advanced controls) to the platform.

We initiated conversations with likely Primary Vendors (New BAS Vendors):

- Building Energy Engineering (retro-commissioning) Company
  - Quantum Energy Services & Technologies, Inc. (QuEST) currently using networked thermostats; interested in open data, ease of data mgmt
- Energy Utility (Catalyst) / Demand Response Controls Vendor

  - Sacramento Municipal Utility District is interested in combining OpenBAS and GridLinks for commercial DR
- Lighting Controls Vendor
  - EnOcean very interested in OpenBAS as platform for products
- Energy-Intensive Process (refrigeration) Control Vendor
  - CloudFridge currently using sMAP, interested in using openBAS for secure cloud-based applications, such as comm refrig optimization

We also identified other likely primary vendors:

- Physical Security (alarm) Company
- Contemporary Information Technology (IT) Business Ventures
- Communications (Cable, DSL, Phone) Companies
- Building Maintenance Companies (e.g., Takenaka)
- Other Energy Service Companies
- HVAC Service Companies
- Existing BAS Vendors

Finally, we identified other desirable partnerships and discussions:

- Other Energy Management App Developers (e.g., Visible Energy)
- Rooftop HVAC Package Unit Manufacturers (e.g., Daikon)
- Rooftop HVAC Package Unit Retrofit Vendors (e.g., Catalyst)

These were outlined in the Market delivery strategy report.

Task 3: Testing and Demonstration

Subtask 3.1 Testing and Demonstration of working prototype
Initial testing in the lab included setting up BOSS/OpenBAS on a Linux device, a miniature computer called a FitPC and a Wireless Access point (router) to make a Building Area Network in 410 Soda Hall at UC Berkeley (February 2014). The equipment tested were a RTA thermostat, Prophilix thermostat, HUE lamp, and LED strip light. All devices connected to the network, and had an sMAP source (instance of driver). We were able to show device interaction through a simple web interface, e.g., Change temperature setpoint on one thermostat automatically changes the setpoint on the other, and graphic display of room temperature using LED strips.

Team members set up several thermostats connected to LED lights to be used as a testbed at UC Davis; also they set up sMAP on a computer at UC Davis. The LBNL team developed a similar testbed at LBNL for testing Enlighted and other lighting controls.

In April, the Berkeley team developed a lab-scale testbed in 410 Soda Hall, consisting of a Plexiglass “building” with two zones and several hardware devices, shown below.

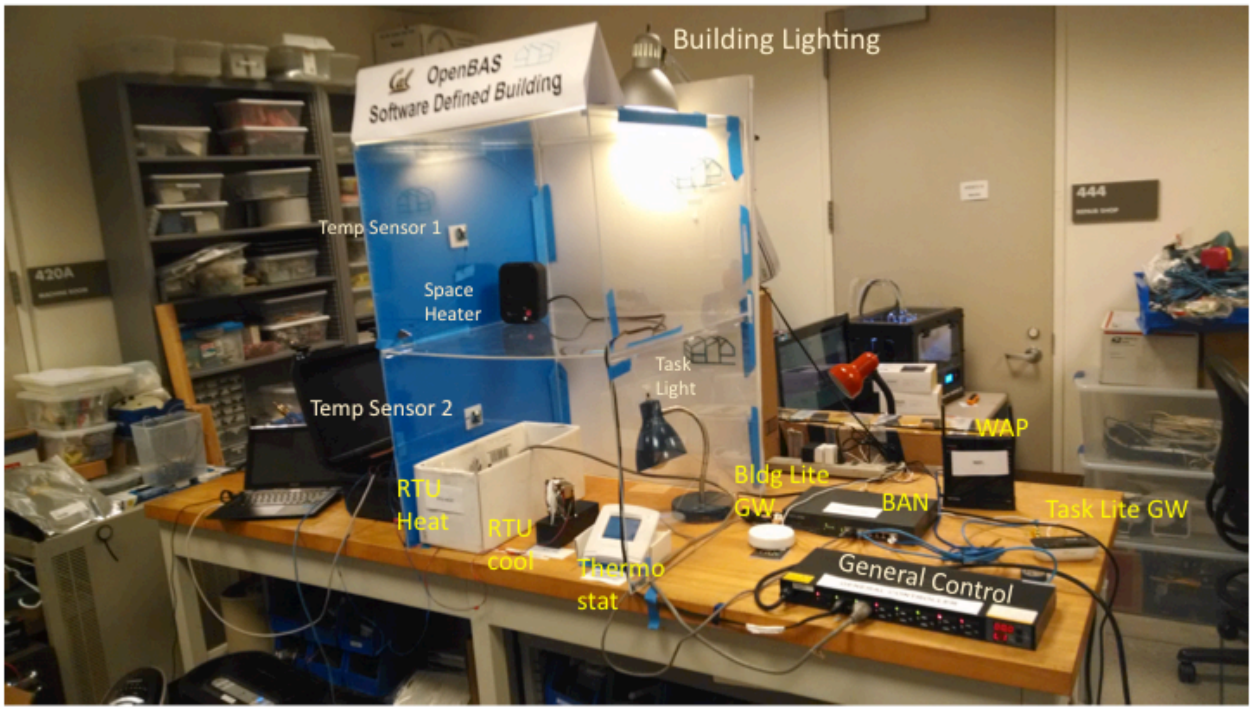


Figure 12: Lab-scale test bed for OpenBAS demonstration of integrated HVAC, lighting, and general control.

The schematic scheme is shown in the figure below. We demonstrated the interaction of a thermostat, two lighting systems (an overall building light (HUE) and a task light (TCP lighting)), two temperature sensors, and general controls (fan, heat lamp, space heater) with the platform, including a web-based user interface for control and status display. This testbed allowed us to show integrated thermal, lighting, and general control with a preliminary user interface, that indicates status display, and control schedule. Another key element is the managed relationship of manual, schedule, and override control.

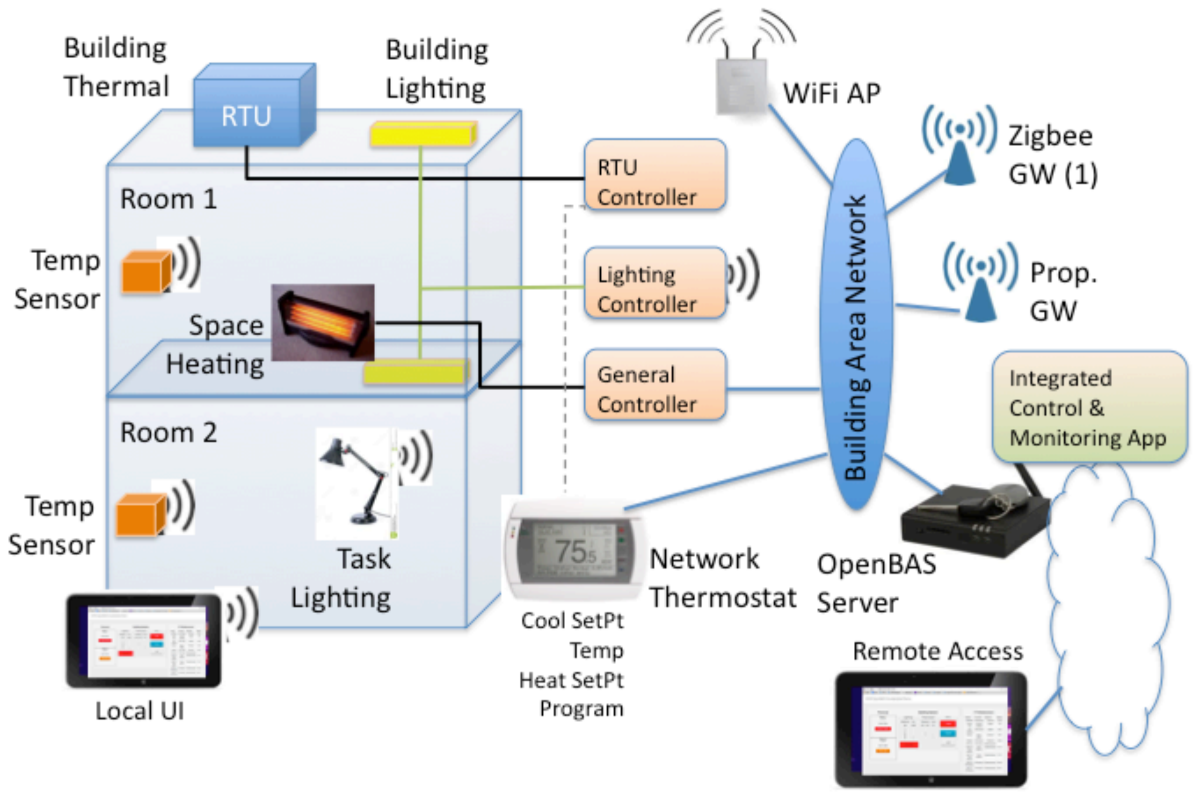


Figure 13: A schematic of the lab-scale test bed.

We also tested the scaling of OpenBAS down to "motes", such as wireless embedded microcontrollers. OpenBAS drivers were ported to a Raspberry Pi and successfully tested.

Deployment

The real test of the ultimate success of this project is how the platform performs under real world circumstances with real equipment (e.g., simultaneous heating and cooling, overlapping zones), real people (e.g., working different schedules, having different preferences for temperature and lighting conditions) and situations (e.g., power outages, network drops).

One potential deployment scenario is for OpenBAS to run on the premises within an existing private WLAN, as shown in Figure 14 below. For example, a small business that has an existing private wireless network would simply add the devices (such as the OpenBAS platform on a miniature computer and various thermostats, gateways to lighting controllers) to this existing network.

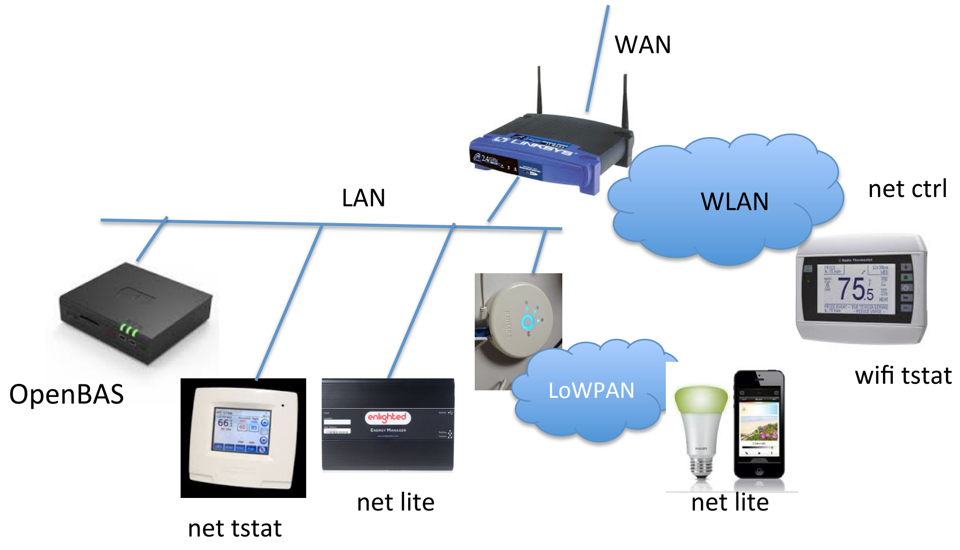


Figure 14: OpenBAS within an existing WLAN.

Another scenario is for OpenBAS to form a private Building Area Network, as shown below.

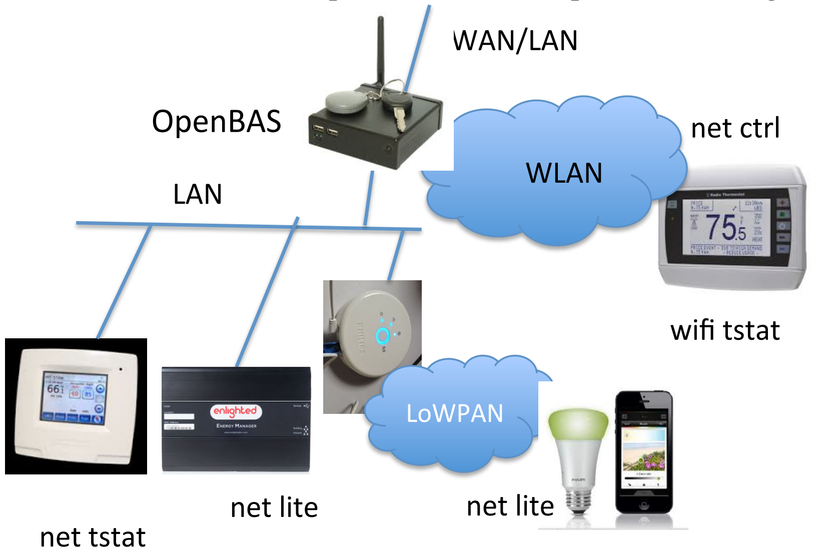


Figure 15: OpenBAS forming a private BAN.

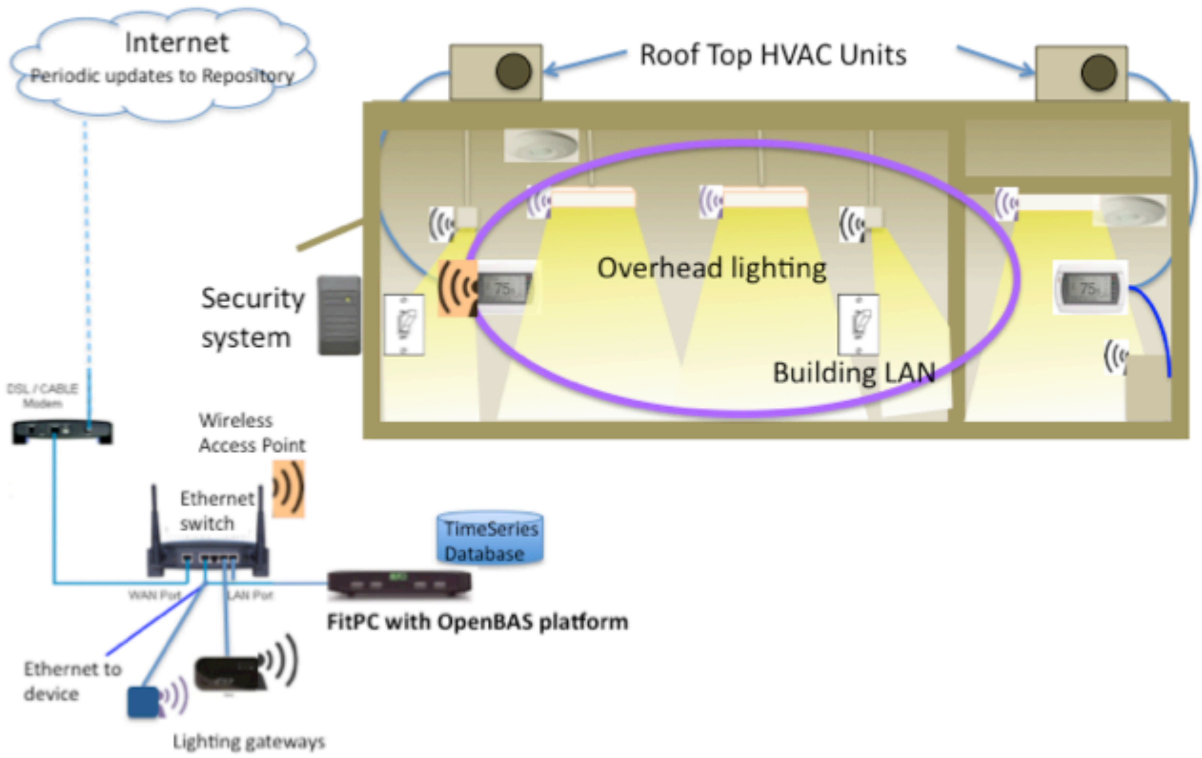


Figure 16: Schematic of OpenBAS installed in a building, and creating the Building Area Network of hardware devices.

We developed a pilot test of the OpenBAS platform in a historic 25,000 sf commercial building at the CIEE office on the second floor of 2087 Addison Street in Berkeley, CA.

Figure 17: The CIEE offices are located in the upper floor of the historic Kress building in downtown Berkeley, CA.

By the end of the project, the OpenBAS platform has been installed at the CIEE office with the following equipment:

Five thermostats

two RTAs (WiFi), and three Proliphixs (on virtual Local Area Network using Ethernet)

Three lighting controllers

Enlighted system on LED fixtures in the conference room

Enlighted system on dimming fluorescent fixtures in the copy room

Wattstopper/EnOcean switch and occupancy sensor in private office and in kitchen

Terralux/EnOcean dimmable LED retrofit in corridor

Two general controllers

Raritan PDU controlling fan, task lamp, and computer monitor in private office

Echola PDU in kitchen

14 indoor environmental sensors throughout the floor (6lowPAN mesh network)

temperature, relative humidity, light, carbon dioxide, motion

Rainforest Eagle interface with PG&E utility electric interval meter (ZigBee)

Data collected at 5 second intervals from the utility meter

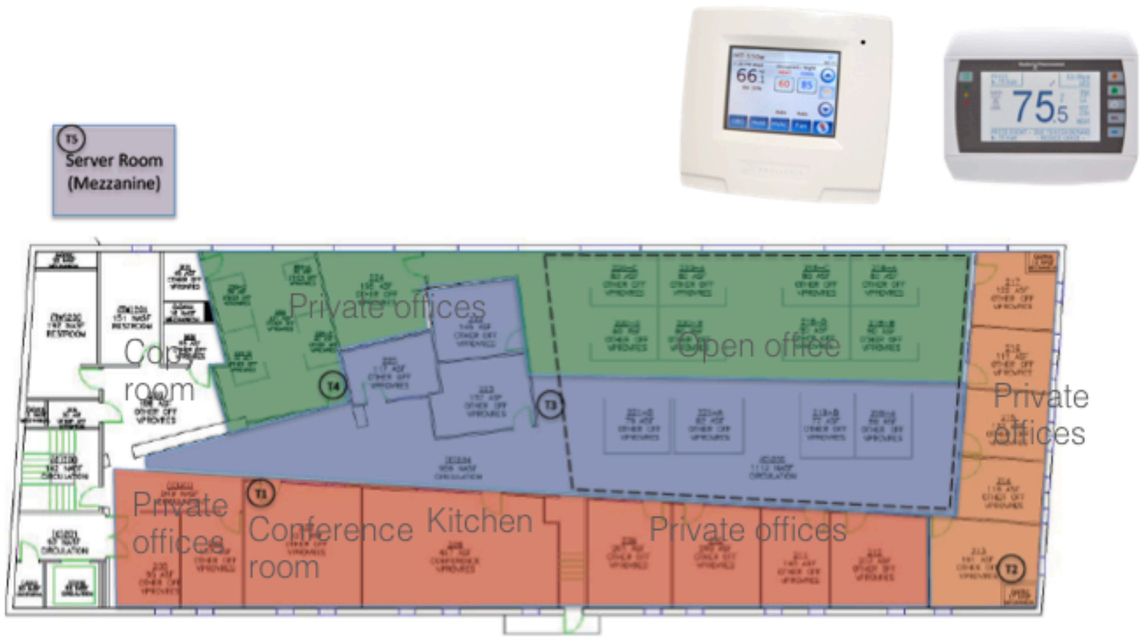
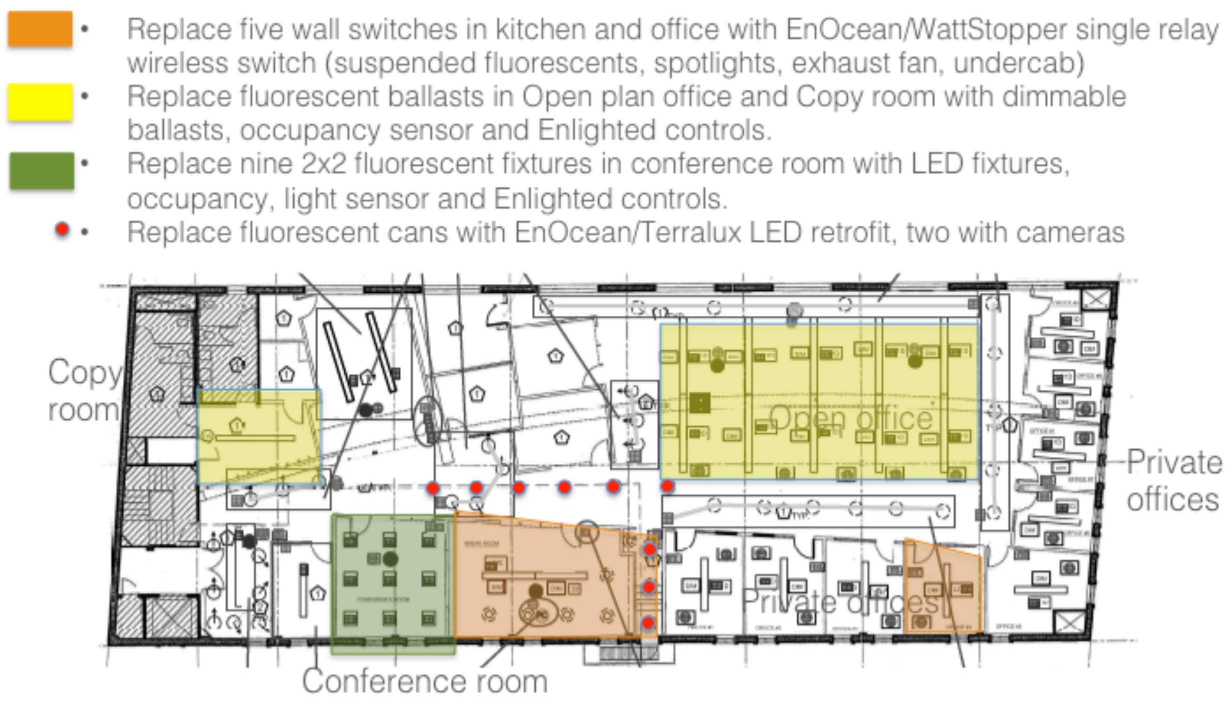


Figure 18: Five thermostats placed in the CIEE offices: two RTAs in the open area and the northern private offices, and three Prophilix thermostats.



🟧 • Replace five wall switches in kitchen and office with EnOcean/WattStopper single relay wireless switch (suspended fluorescents, spotlights, exhaust fan, undercab)

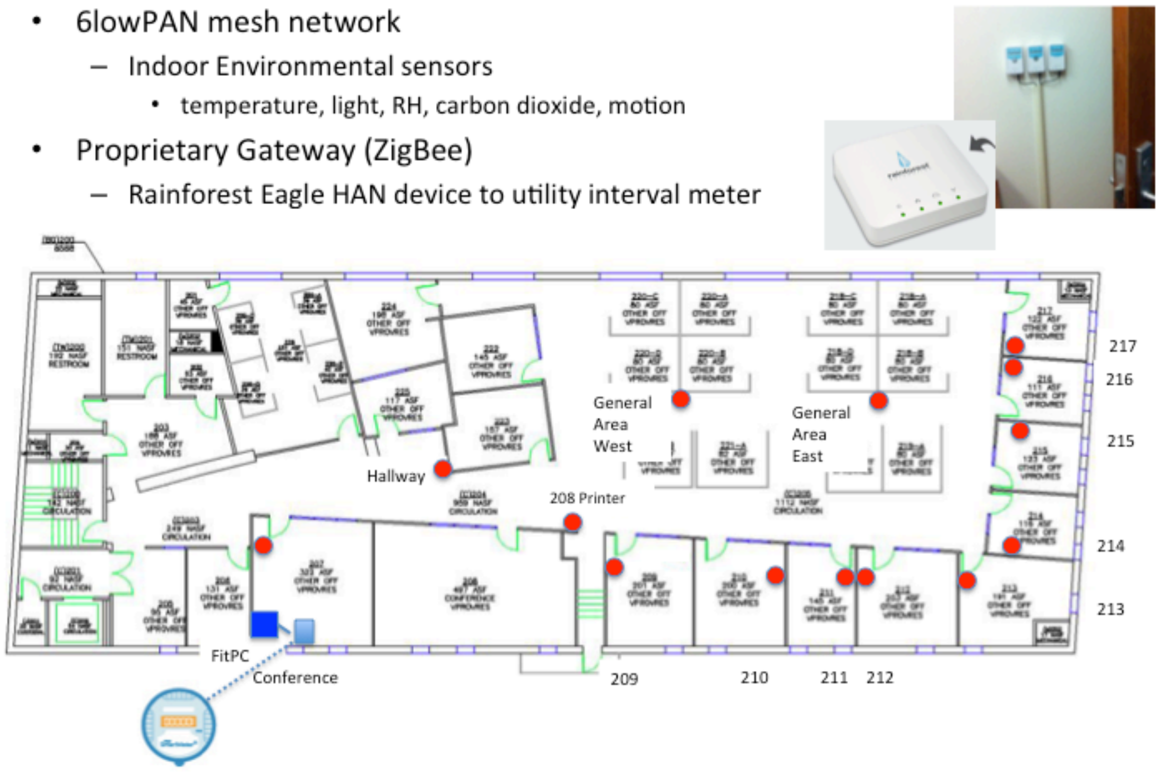
🟨 • Replace fluorescent ballasts in Open plan office and Copy room with dimmable ballasts, occupancy sensor and Enlighted controls.

🟩 • Replace nine 2x2 fluorescent fixtures in conference room with LED fixtures, occupancy, light sensor and Enlighted controls.

● • Replace fluorescent cans with EnOcean/Terralux LED retrofit, two with cameras

Figure 19: Lighting control installed in CIEE office.

In addition, 14 sets of indoor environmental sensors were added in the building, interfaced to the OpenBAS platform with a 6loWPAN mesh network; these reliably provided detailed sensor data throughout the building. The types of sensors are shown below in Figure 20. Also, a gateway was installed for the electrical utility interval meter for this floor of the building. The gateway interfaced with the OpenBAS platform and provided real-time (less than 30 second interval) power data for the floor of the building. The research team found this data quite valuable in understanding the overall contribution of HVAC and lighting to the whole building load.



- 6lowPAN mesh network
  - Indoor Environmental sensors
    - temperature, light, RH, carbon dioxide, motion
- Proprietary Gateway (ZigBee)
  - Rainforest Eagle HAN device to utility interval meter

Figure 20: Auxiliary sensors added to the OpenBAS platform.

Publications/Presentations/Travel

Project was presented at EPRI's Power Delivery & Utilization Program, Software Defined Buildings Winter retreats (UC Berkeley/industry), Green Tech Center/ITU/SDU (Denmark), Saga University (Japan), and Daikin Konwakai (St. Michael's, MD). The working platform in a plexiglass "building" was demonstrated at the DOE BTO annual review, at the Carnegie Mellon University OpenBAS workshop, to Ethan Rogers, ACEEE Chairman of the Intelligent Efficiency conference, and at the IT University in Copenhagen Denmark. A paper was presented at ACEEE's Summer Study on Energy Efficiency in Building.

Conclusion

In developing both a building-scale and lab-scale deployment of an open software architecture building automation system, we satisfied and exceeded Milestone1.1 for this project. We developed and tested a working open source (Access instructions in Appendix A) open software architecture building automation system targeted to small-medium commercial buildings. OpenBAS provides integrated building services, including HVAC, Lighting, General Control, Energy, Environment. The hardware devices represented multiple vendors in each of multiple trades. We developed plug-n-play capability from device to application, *executable* specification of semantic relationships, and interactions within and between system zones.
Key developments included:

- Broad driver suite of modern devices
- Canonical driver resource architecture

- Clean handling of actuation and subscription
- Well-defined Service Tier
  - scheduler, discovery, zone ctrl, x-zone ctrl
  - interconnections through *select* clause
- Canonical metadata => entity-relationships
- Discovery protocol
  - detection, driver support, profile linkage
- Profile / Document Store – versioned
- Clear infrastructure / Application Separation
  - publish to multiple stores => meteor syndication
- Independent Application Tier

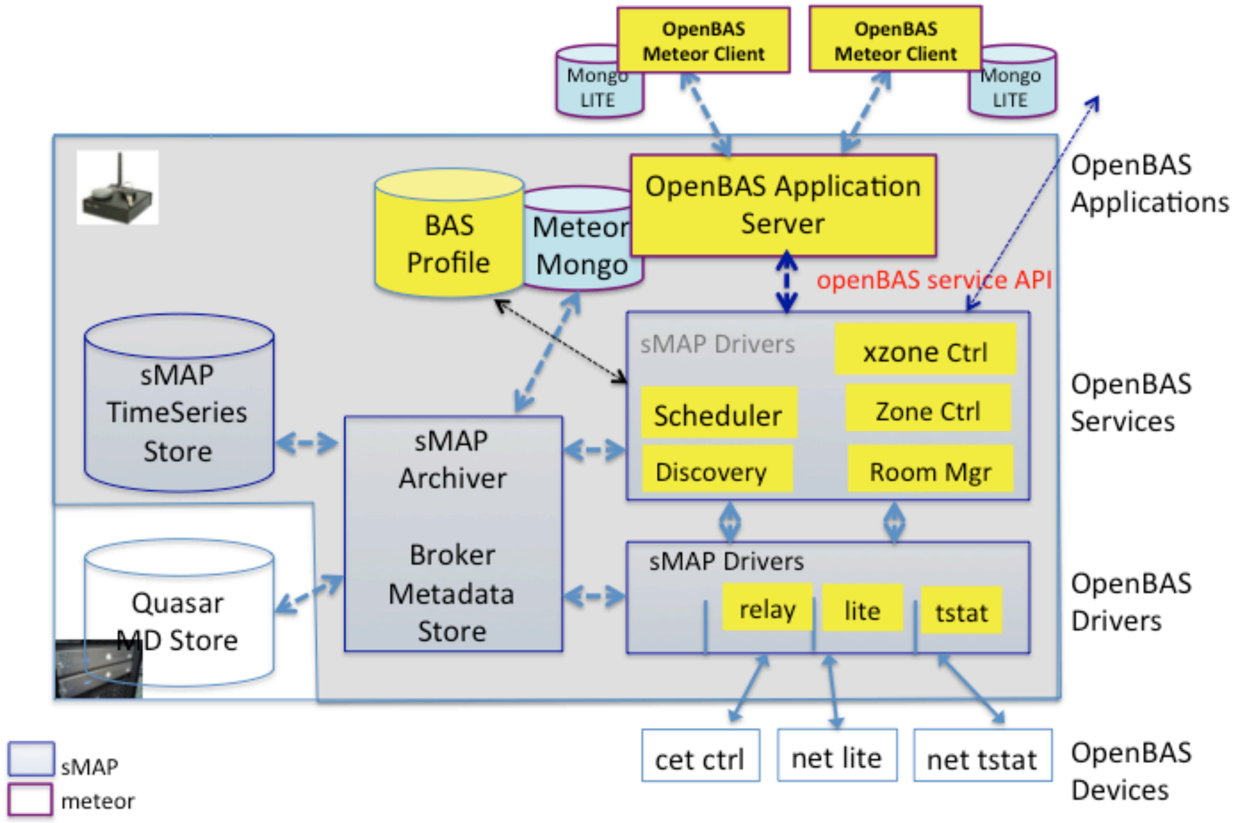
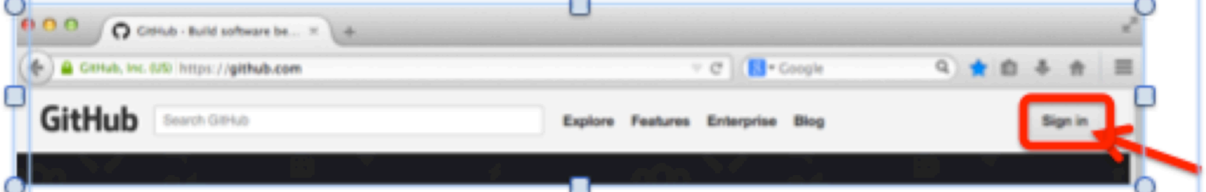


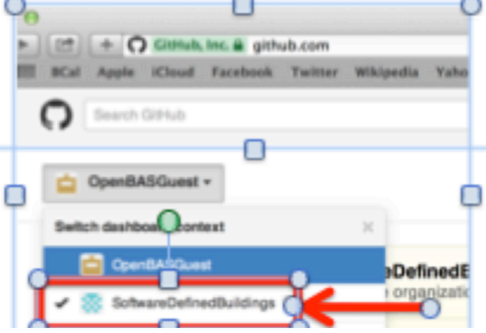
Figure 21: OpenBASArchitecture

Appendix A: Access to OpenBAS code

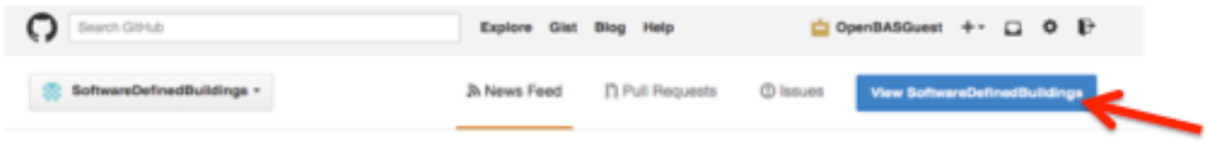
1. Go to https://github.com/
2. Click on “Sign in” (as shown below)


GitHub - Build software be...
GitHub, Inc. (US) https://github.com
GitHub
Search GitHub
Explore
Features
Enterprise
Blog
Sign in

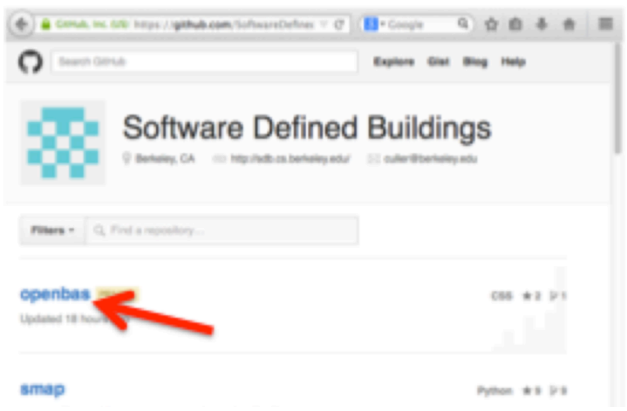
3. Enter for User name: OpenBASGuest, and Password: openbas822project
4. In the repository selection on the left (as shown below), select Software Defined Buildings.



5. Click “View SoftwareDefinedBuildings” on the right, as shown below.



6. You will see the complete repository for sMAP as well as OpenBAS (see below). Click on openbas.



We have provided a complete install procedure for an OpenBAS deployment on appropriate hardware, which is described at the beginning of the README. To see the full install process, click on INSTALL.md.