# Enabling Portable Energy Applications with a Building Operating System

*Gabe Fierro, EECS Department, University of California, Berkeley*
*David E. Culler, EECS Department, University of California, Berkeley*
*Therese Peffer, California Institute for Energy and Environment, University of California*

## ABSTRACT

Modern buildings, which consist of hundreds or thousands of embedded, networked sensors and actuators in a building management system, are a rich target for applying advances in information technology to facilitating common building application tasks. Even small-medium commercial or residential buildings, which usually do not have incumbent control infrastructure, are now commonly retrofitted with smart devices such as thermostats and lighting and plug controllers. However, there is little integration between HVAC, lighting and electrical subsystems in a building. Understanding the energy accounting of each piece of this diverse collection of building subsystems and devices can enable improved optimization and energy efficiency, better load shifting and shedding, and improved building-to-grid dynamics.

In this paper, we demonstrate how XBOS, an extensible building operating system, enables the construction of portable, building-agnostic applications that use the control signals from smart devices. These applications include: modeling energy consumption of devices, attributing energy usage to individuals or faulty schedules, establishing candidates for peak shedding to alleviate demand charges, creating auto-generated HVAC and energy reports, and integrating external signals (such as demand response) for incorporating energy accounting into building control processes. The XBOS platform allows for novel control and analysis schemes to be evaluated across many building contexts.

## Introduction

Over 90% of buildings in the U.S. are less than 50,000sf, yet most are not equipped with a building automation system (BAS) for monitoring or control (Katipamula, et al 2012). The lack of control and monitoring infrastructure in buildings means there is often very little visibility (beyond a monthly power bill) into how a building performs, let alone how it might be improved. Though analytics and efficient control are the subject of both research and commercial ventures, most solutions are ad-hoc and non-portable, being programmed for specific buildings. In this paper, we discuss how XBOS, an eXtensible Building Operating System, promotes the rapid construction of portable applications. XBOS leverages previous work on sMAP to integrate a collection of smart devices (including thermostats, lights and plug controllers) with data archival and real-time communication.

In this paper, we discuss the principled design of the XBOS platform and how its components and interfaces enable a wide range of applications. Using the XBOS query mechanism, applications can discover the features of a building and adapt themselves to its structure. Complex applications can be written once by an expert and then ported to other buildings with a minimal amount of effort. We examine four such applications in this paper.

The data presented in the application evaluation section is taken from a real-world deployment of XBOS in a 7,000 sf office building in Berkeley, CA. The equipment consists of four networked thermostats (two WiFi-based CT80 and two Ethernet-based IMT550C) connected to four rooftop units (RTUs) and a full-building power meter (Rainforest Eagle). The XBOS platform (drivers, broker, timeseries and metadata databases — explained below) is completely on-premises, installed on a 2-core, 8 GB RAM commodity server running Ubuntu 15.10.

## Research Design

Rooftop Units (RTU), typically controlled by simple thermostats, condition the vast majority of small-medium commercial buildings. Recently, these thermostats are being replaced with “smart thermostats” (offered by companies such as Honeywell, Regen Energy, Ecobee, Nest and Radio Thermostat) that offer optimized control schemes and network connectivity. However, these technologies are often proprietary, and the control and monitoring infrastructure is restricted to operating with a single vendor’s equipment.

The goal of this paper is to demonstrate the ease with which the XBOS platform enables control and report applications that integrate previously isolated building subsystems such as HVAC and lighting. In contrast to prior work in which applications are hardcoded to the specific construction and equipment in a particular building, XBOS applications express at a high level which types of devices are needed and how those devices relate to each other. This method of abstraction enables potentially complex applications to be written without prior knowledge of a building and deployed with minimal changes. Below, we describe the design principles and primary features of the XBOS platform and then discuss the construction and execution of four representative applications — an extensible building scheduler, a RTU demand shedding application, an HVAC equipment energy disaggregator and an auto-generated energy report.

## XBOS Platform

This section briefly discusses the XBOS platform as it relates to the applications described in this paper. XBOS is open source and is freely available online; a more complete and technical description of XBOS is left for the XBOS technical report (Fierro, Culler 2015).

### Dimensions of BOS Design

Building automation systems are an active research topic, but existing solutions lack the principled approach for a truly extensible platform. In (Fierro, Culler 2015) the authors establish six dimensions of building operating system (BOS) design, which we summarize here.

**Hardware Presentation Layer** is a layer of abstraction that establishes uniform read/write access to the set of devices and data sources in a building. This is necessary for interoperability in multi-vendor deployments, a core aspect of the Simple Monitoring and Actuation Profile, sMAP, which informed much of the XBOS design and is in fact forwards-compatible with it (Dawson-Haggerty, et al 2010). XBOS extends the sMAP device driver ecosystem beyond a read/write interface to include a rich subscription mechanism for composing real-time and reactive services and applications above.

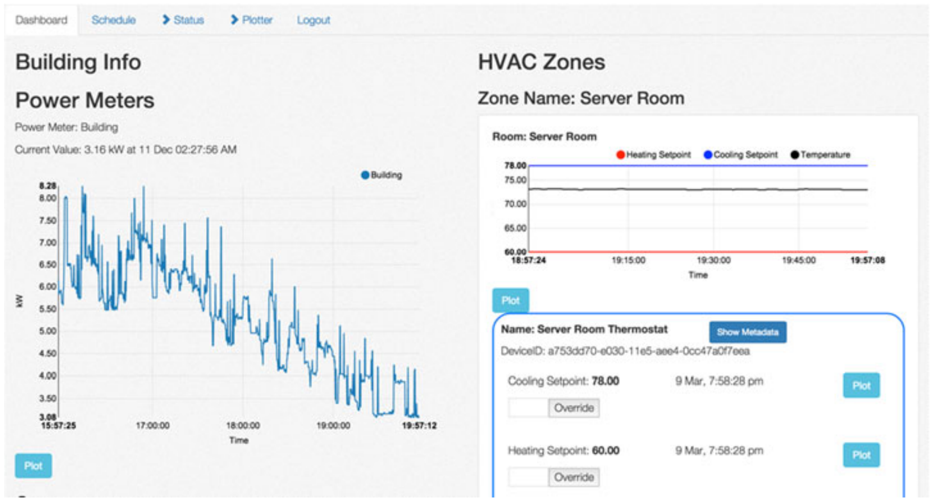
Drivers are lightweight processes that can be run directly on a capable device (such as an embedded sensor platform), on cheap hardware local to the building (such as a Raspberry Pi) or even in the cloud. Drivers expose each sensing or actuating feature of a device as a *stream*. A *stream* consists of an ordered sequence of (time, value) pairs called a timeseries identified by a unique key (UUID) and described with a set of key-value pairs drawn from the Building Profile that denote the function, location and other important context of the stream. Each driver receives new values (e.g. sensor readings) from an attached device and transmits these to the central XBOS Kernel. The driver also manages all incoming data, including direct actuations or schedule outputs.

**Canonical Metadata** is a standard representation of the structure of a building and the subsystems and devices within. Such a representation gives apps the means to discover what features exist in a building without being pre-programmed for a specific installation. Existing proposals and standards include Industry Foundation Classes (IFC), Open Building Information Exchange (OBIX), Green Building XML (GBXML) and Project Haystack. Despite the existence of powerful platforms operating on these representations (e.g. Tridium over OBIX), the resulting systems do not actually facilitate the task of writing building-agnostic applications. Academic BOS BuildingDepot2 (Went, et al 2013) and Building Application Stack (Krioukov, et al 2012) construct applications using *relationships* between building components. This ability to discover building components through functional relevance is key, and the XBOS Building Profile adheres to this principle.

The Building Profile consists of standardized metadata keys which describe the structure of a building at a high level. These keys are stored as metadata on the collection of streams produced by device drivers. Keys denote spatial structure of a building (floors and rooms), HVAC structure (mapping between RTUs and HVAC zones, and mapping zones to sets of rooms), lighting structure (mapping lighting controllers to rooms and individuals), electrical structure (organizing the set of power meters) and monitoring structure (occupancy, temperature, humidity, CO2 sensors and others). XBOS queries use knowledge of the Building Profile to describe desired information, for example the heating setpoint for the HVAC zone that includes the kitchen, or all rooms that have both illumination sensors and dimmable lights.

**Control Process Management** is the means of installing, monitoring and reasoning about the decision-making processes that govern the operation of a building, ranging from schedules over thermostats and lights to safety, security and alarm systems. There is little-to-no control infrastructure in small-medium commercial buildings; instead, both smart and un-networked devices (e.g. thermostats, lights, plug controllers) operate independently on limited schedules or, more often, at the behest of a human operator. An effective BOS must provide more than just attractive visualizations and static control processes hardcoded to the construction of a particular building — it must enable code reuse and simplify the reasoning about and evolution of building control. XBOS schedules and control processes are straightforward to define, as explored in the latter half of this paper.

**Building Evolution Management** is the means by which the physical state and virtual representation of a building are kept consistent. An oft overlooked aspect of buildings is that they change. Static building configurations quickly become inconsistent with the inevitable repairs, equipment replacements, subsystem retrofits and control changes that are natural to a building's lifecycle (Claridge, et al 2004). A key feature of XBOS is its ability to keep applications informed of changes in the building structure. Applications identify resources in terms of functional relationships, for example "all dimmable lights in room 410" or "the heating setpoint for the zone containing the conference room." Because XBOS knows what an application or controller needs at a higher level of abstraction, it can inform that process of any relevant changes in the building, for example if a new device has been installed or if an old device has moved or been decommissioned. This is in contrast to current systems in which applications and controllers must explicitly name which devices they are using — e.g. “thermostat at IP address 10.4.10.50” or “light relay with name WS86007:RELAY01” — which can easily become invalid. Furthermore, in these schemes there is no linking between a device’s “name” and its context, e.g. which room it is in, who owns it, what features it has or how it is linked to a building subsystem.



Building Info

Power Meters

Power Meter: Building

Current Value: 3.16 kW at 11 Dec 02:27:56 AM

HVAC Zones

Zone Name: Server Room

Room: Server Room

Name: Server Room Thermostat

DeviceID: a753dd70-e030-11e5-aee4-0cc47a0f7eea

Cooling Setpoint: **78.00** 9 Mar, 7:58:28 pm

Heating Setpoint: **60.00** 9 Mar, 7:58:28 pm

Figure 1. The XBOS Building Dashboard is autogenerated from the Building Profile and offers schedule management, device setting override, device and building status and visualizations.

**Security** in a BOS deals with the authentication of users interacting with the system and the authorization of the actions they perform; this cannot be overlooked in any system that manages buildings. XBOS supports basic read-only and admin accounts, and a more sophisticated permission system is under development. More importantly, an XBOS deployment can be entirely local, without any need to send any data to untrusted cloud services. This severely reduces the attack surface for an XBOS deployment, which can operate successfully without any interaction with the outside world.

**User Experience** defines the primary point of interaction (beyond environmental conditions) between building occupants and building systems. “Smart” devices typically provide limited local interfaces (such as a touchscreen on a thermostat) or connect to a vendor-specific mobile or cloud application. These interfaces rarely provide features beyond direct control and displaying current device state. XBOS includes a building dashboard that integrates HVAC, lighting and monitoring systems in a building, combining historical visualizations with direct control and schedule management (Figure 1).

## XBOS Kernel

The XBOS Kernel (Figure 2) is the primary, central component of the XBOS platform, containing a timeseries database for historical data archival and access, a metadata database for maintaining the Building Profile, and a query processor and message broker that handle discovery and communication among the drivers, applications, controllers and other services in the platform. The kernel provides three interfaces exposed over a variety of transports and protocols including JSON/HTTP, JSON/WebSockets and MsgPack/TCP (see Fierro, Culler 2015).

Publishing data and control decisions is done by sending a message to the “Add Data” interface. Any scalar values (such as sensor readings) in the message are archived in the time series database and the message metadata is stored in the metadata database. Then, the message is handed to the message broker where it can be forwarded to any subscribers. This interface is identical to that of the sMAP archiver service to maintain compatibility with the large ecosystem of sMAP drivers.

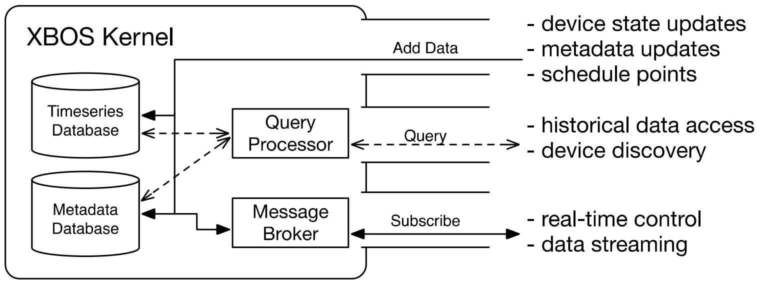


Figure 2. Architecture of the XBOS Kernel, which provides three interfaces to aid the construction of applications.

Discovery of building structure, available devices and retrieval of historical data is performed by sending a query to the “Query” interface. Queries use the same two-part structure as sMAP queries: a “select” clause and a “where” clause. The “select” clause denotes which data is to be returned, for example the most recent data point, all data points in a given range or a list of metadata tags. The “where” clause specifies using a relational predicate the set of constraints on which devices or streams should be “selected.” This clause combines boolean operators (AND, OR) with tests for existence, equality or partial matching on metadata keys and values.

When the “Query” interface receives a query from a client, the query processor evaluates it against the timeseries and metadata databases, returns the result to the client, and terminates the connection. If instead a query is sent to the “Subscribe” interface, the connection is kept open after the initial result is returned. As new sensor values, new devices, and changes in metadata arrive at the kernel (via the “Add Data” interface), the message broker forwards these to subscribed clients. Over a building’s lifecycle updates on device metadata express changes in equipment, structure and control, which may affect the set of streams or devices clients subscribe to. To account for these changes, the message broker dynamically reroutes subscriptions so that all clients (including applications, schedules and controllers) have a consistent view of the building’s state.

## Applications

For each application, we explain the methodology, and, where applicable, show the architecture and source code for the application, and evidence of the application running in an active XBOS deployment.

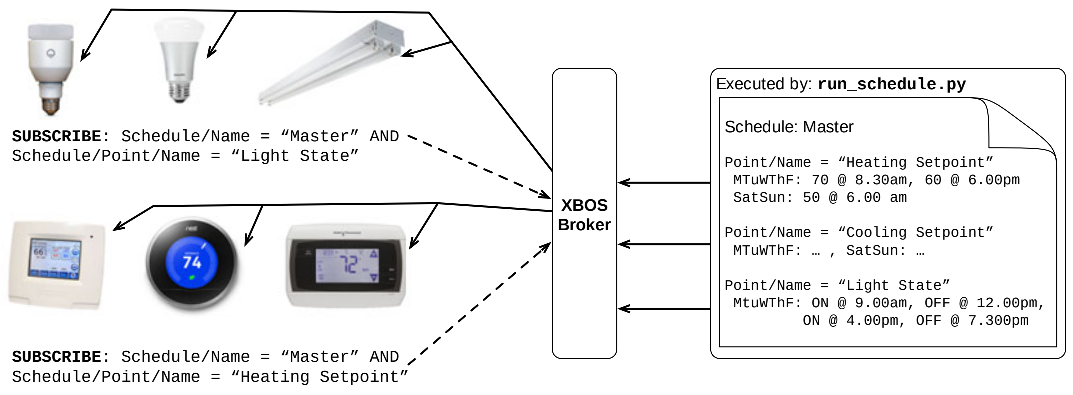

SUBSCRIBE: Schedule/Name = "Master" AND Schedule/Point/Name = "Light State"
SUBSCRIBE: Schedule/Name = "Master" AND Schedule/Point/Name = "Heating Setpoint"
XBOS Broker
Executed by: run_schedule.py
Schedule: Master
Point/Name = "Heating Setpoint"
MTuWThF: 70 @ 8.30am, 60 @ 6.00pm
SatSun: 50 @ 6.00 am
Point/Name = "Cooling Setpoint"
MTuWThF: … , SatSun: …
Point/Name = "Light State"
MtuWThF: ON @ 9.00am, OFF @ 12.00pm,
ON @ 4.00pm, OFF @ 7.300pm

Figure 3. The basic communication pattern for how schedules are defined in XBOS. Devices (via their drivers) express a subscription query to the broker (dashed line), identifying which schedule(s) it is interested in. Schedules push their control decisions to the XBOS broker, which routes the current schedule settings to the relevant devices (solid lines).

## Master Building Scheduler

Setting and maintaining schedules is one of the primary purposes of a building automation system. Most smart thermostats come with this ability, but place arbitrary constraints on how many schedules can be set or even how they can be set (e.g. weekdays and weekends, but no exceptions for holidays or Spare the Air alerts). Most other types of smart appliances lack the ability to schedule themselves. The ability to easily define full-building schedules across the set of multi-vendor devices in a building is a crucial feature to efficient building management.

A schedule service for a building should be able to: a) define schedules to control more than one device and more than one *type* of device (e.g. lights, thermostats, plug loads), and b) define schedules in terms of *time* (e.g. "set heating stepping to 60 deg F at 7pm") and *events* (e.g. "turn on overhead lights when someone enters the room") and apply conditionals to those triggers. An example of a conditional might be the schedule "turn on overhead lights when someone enters the room *unless* their desk lamp is already on." Individual devices that have scheduling capabilities usually limit their schedule definitions to a handful of periods. Current building automation offerings ranging from the small-commercial and residential (Nest Labs) to the large commercial building (Tridium Inc.) offer more flexible time-based scheduling, but event-driven schedules are limited to hardcoded connections between sensors and actuators.

XBOS is naturally suited to defining and enacting multi-device schedules. Device drivers query XBOS for the set of available schedules and then subscribe to the outputs of the schedules that interest them. We first illustrate the basic architecture of how schedules are produced and consumed in XBOS, then explain how traditional time-based schedules work before demonstrating how event-based and conditional schedules can be composed.

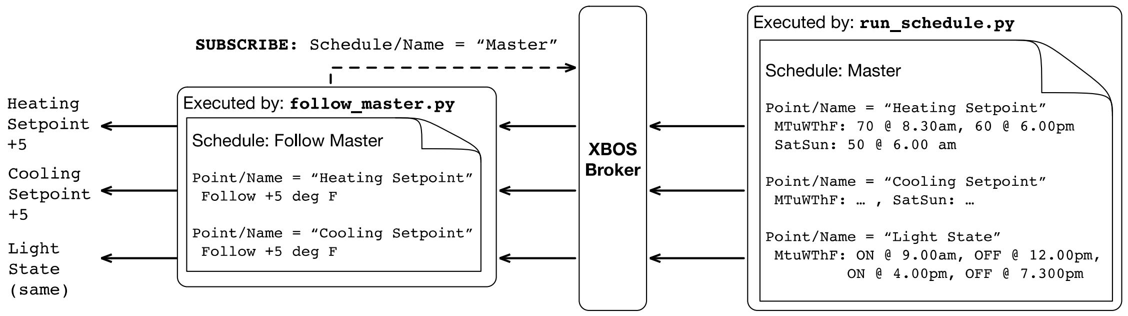

SUBSCRIBE: Schedule/Name = "Master"
Heating Setpoint +5
Cooling Setpoint +5
Light State (same)
Executed by: follow_master.py
Schedule: Follow Master
Point/Name = "Heating Setpoint" Follow +5 deg F
Point/Name = "Cooling Setpoint" Follow +5 deg F
XBOS Broker
Executed by: run_schedule.py
Schedule: Master
Point/Name = "Heating Setpoint" MTuWThF: 70 @ 8.30am, 60 @ 6.00pm SatSun: 50 @ 6.00 am
Point/Name = "Cooling Setpoint" MTuWThF: … , SatSun: …
Point/Name = "Light State" MtuWThF: ON @ 9.00am, OFF @ 12.00pm, ON @ 4.00pm, OFF @ 7.300pm

Figure 4. The "Follow Master" schedule subscribes to all outputs of the master schedule and selectively applies an offset. Here, the follow schedule adds 5 degrees to the heating and cooling set points of the master schedule before publishing them. The light state schedule point does not have an offset, so the follow schedule simply republishes it. The published points are identified with `Schedule/Name = "Follow Master"`.

Figure 3 illustrates a simple single-schedule setup for XBOS. Schedules are defined by textual documents stored in a database or local file, executed by a simple driver which registers each scheduled point as a unique stream. The driver produces a new timeseries value at each schedule epoch which is published to the XBOS broker for archival and forwarding to schedule subscribers. This structure has three primary benefits. Firstly, it is possible to define a full-building schedule using a single file that can control all devices regardless of their API or communication mechanism. Secondly, devices can "join" or "leave" a schedule at any time because schedules do not have to be informed of which devices they are controlling, which ameliorates the process of properly configuring a newly installed device. Lastly, because subscribers can be arbitrary processes as well as drivers, this mechanism enables the composition of more advanced schedules that combine device control signals and/or external configurations to augment a time-based schedule.

The most basic example of an augmented schedule is a "follow with offset" schedule which subscribes to the output of the master schedule, but has a zone-specific temperature offset to account for measurement error in the thermostat. The resulting schedule sends control signals at the same times as the master schedule, but with each value adjusted by some configured offset. Figure 4 illustrates an example of this configuration, and Figure 5 provides the real JavaScript code behind the example. We can easily adapt this structure to write a simple schedule whose epochs are triggered by control signal events: when an occupancy signal is triggered, turn on the overhead lighting. This is accomplished by subscribing to the latest value of any relevant occupancy sensors, which can be subscribed to using the query:

```
SELECT DATA BEFORE NOW WHERE Metadata/Point/Sensor = "Occupancy" AND
Metadata/Location/Room = "410";
```

In most commercial systems, even this simple integration of sensors and lights is not possible unless these have been explicitly programmed to talk to each other.

```
var WebSocket = require('ws');
// open connection to the XBOS broker
var w = new WebSocket('ws://<xbos broker url>:8078/subscribe');
w.on('open', function() {
    // initialize the subscription to the Master Schedule
    w.send('Metadata/Schedule/Name = "Master"');
});
// when a new schedule point is received,
// add 5 if it is Heating/Cooling, else forward it
w.on('message', function(point) {
    var pointname = point.Metadata.Schedule.Point.Name;
    if (pointname == "Heating Setpoint" ||
        pointname == "Cooling Setpoint") {
        point.value += 5; // add our offset
    }
    // if it isn't Heating or Cooling setpoint, skip the offset
    // change schedule name
    point.Metadata.Schedule.Name = "Follow Master";
    // send the (possibly adjusted) point to the broker
    sendToBroker(point);
});
```

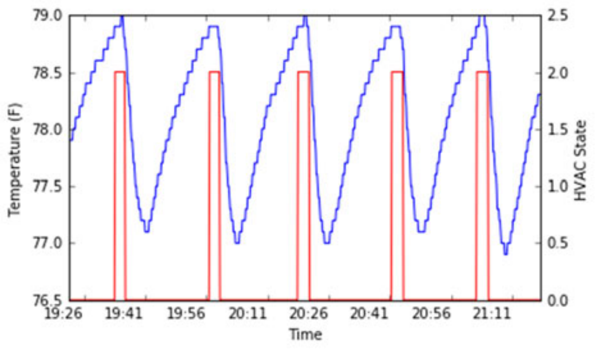
Figure 5. Real JavaScript code to subscribe to the output of the master building schedule using the XBOS WebSocket interface. The `sendToBroker()` method is a convenience method that properly formats the schedule point before sending it to the XBOS broker where it can be forwarded to subscribers. This code sample demonstrates how simple composing applications can be using XBOS.

## HVAC Equipment Disaggregation

Without any visibility into the loads in a building it is difficult to know where to look to cut costs. HVAC in particular is the single largest source of energy consumption in residential buildings and is significant in small-medium commercial buildings (Pérez-Lombard, Ortiz, and Pout 2008). A family of approaches with varying degrees of complexity have been developed to this end, but often require expensive training phases and are not sufficiently accurate (Yan, Wang, and Xiao 2012; Batra, Sing, and Whitehouse 2015). We show here a simple method that uses only the control signals from smart thermostats and a whole-building power meter to quickly calculate actionable estimates for HVAC equipment energy usage, which can then be folded into other advanced applications.

Most smart thermostats expose “HVAC state” as part of their API, which indicates whether the thermostat is heating, cooling, or off. Overlaying HVAC state with the trace from the full building power meter reveals corresponding rises and falls in demand whose magnitude should correspond to the demand placed by the RTU’s heating or cooling activation (Figure 6).

To estimate the demand incurred by calling for cooling for this thermostat/RTU combination given the two timeseries, a piece of code samples the power meter feed before and after each HVAC state transition from off to cooling (state = 0 to state = 2) and from cooling to off. The absolute value of the differences between each of these (before, after) pairs forms a distribution of guesses for the RTU’s demand.



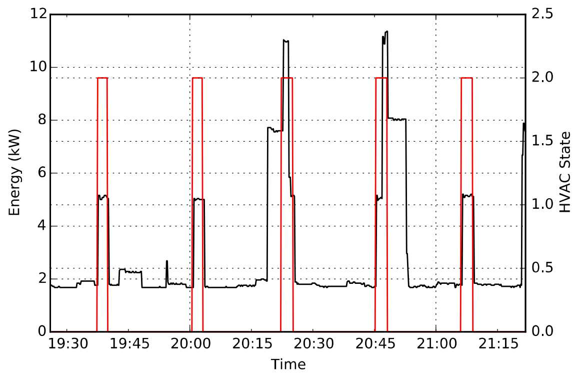


Figure 6. Left: Thermostat's reported HVAC state overlaid with reported temperature measurement. HVAC state of 2 indicates the thermostat is calling for cooling; 0 means the thermostat is "off." Right: Thermostat's reported HVAC state (cooling) overlaid with instantaneous building demand.

This application is implemented over the XBOS platform in less than 60 lines of Python code using the Pandas data analysis library, none of which is building-specific. To make the application portable to other buildings running XBOS, the program uses the XBOS query language to discover the structure of the building and retrieve the appropriate data. The following query retrieves the list of all HVAC zones in the given building:

```
SELECT DISTINCT Metadata/HVAC/Zone WHERE Metadata/Location/Building = "CIEE";
```

For each HVAC zone, the application uses the canonical building metadata to find the time series of HVAC state for the thermostat in that zone and pulls the data for some interval of time. This example uses a 24-hour period, but could easily be extended to use any amount of data. For each zone, the data query is as follows:

```
SELECT DATA IN ("8/30/2015", "8/31/2015") WHERE
Metadata/HVAC/Zone = <Zone> AND Metadata/Location/Building = "CIEE" AND
Metadata/Point/Type = "State" AND Metadata/Point/State = "HVAC";
```

Similarly, the application pulls the instantaneous demand for the full building meter. The application forms and executes these queries against the XBOS query interface to receive the requisite data. The application then uses the Pandas[1] data library to realign the time series data to a 10-second window and identifies the transitions in the HVAC state data. For each transition time, the application pulls from the building energy stream the three data points (covering roughly 30 seconds of samples) before the transition and the three after the transition (six in total) and computes the maximum difference between the two groups. The 30-second window accounts for delays between the building energy meter, the thermostat sending a signal to the RTU and the RTU responding to the signal. For the HVAC zone in Figure 6, over a 24 period, there are 37 transitions with a mean of 3.57 kW and a 3rd quartile of 3.40 kW.

[1] http://pandas.pydata.org/ offers Matlab-like functionality in the Python programming language, including time series resampling and realignment used for this application

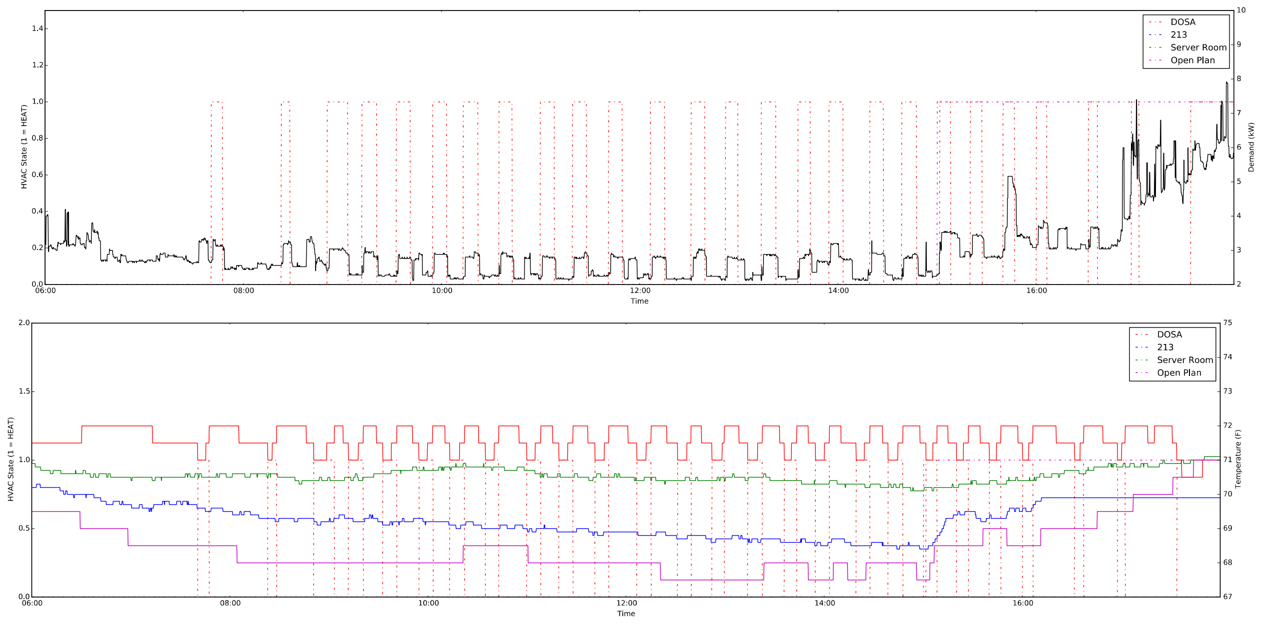


Figure 7. Dashed lines represent thermostat HVAC state. In the top plot, this is overlaid with the full building demand (solid line); in the lower plot, HVAC state is overlaid with the measured temperature for each thermostat (solid lines). We can visually identify that the “DOSA” thermostat dominates energy use during the plotted period.

## Energy Report Application

Many software solutions (e.g. BuildingIQ, Tridium) provide detailed HVAC reports that offer a high degree of visibility into a building’s operations. However, these are non-portable and often hardcoded to a building’s construction. XBOS enables adaptive building reports that do not require prior knowledge of a building’s construction because they can discover the relevant components using the XBOS query language. Furthermore, because XBOS can perform both monitoring and control, a report application can auto-generate and even deploy solutions to wasteful schedules and other phenomena identified by the report.

An example of a building-agnostic report application might identify over a week which day had the highest instantaneous demand and which day had the highest total energy use. For each of these days, the application queries the control signals for all monitored devices — this includes lights and plug loads in addition to RTU load — and using the energy disaggregation technique above can determine which lighting or HVAC zones, or even which rooms, contributed most to the demand over the given period. Once this information is known, the application analyzes the control signals to determine if a control scheme can lower energy usage. A full survey of these techniques is beyond the scope of this paper, but Figure 7 illustrates a scenario in which a single thermostat/RTU constantly duty cycles over a nighttime period with a heating setpoint of just 71 degrees Fahrenheit. A report application would verify that the building was unoccupied during this period, and experiment with lowering the heating setpoint to reduce energy consumption.

## Demand Shedding

Under some pricing schemes, a portion of a monthly energy bill is derived from the peak energy usage over the month. Orchestrating the on-off cycles of multiple RTUs can reduce costly demand charges, often the bane of small commercial businesses. Because all thermostats act independently, there is no mechanism to prevent two or more thermostats from calling for heating (or cooling )at the same time. In a building such as CIEE, all 5 RTUs calling for cooling at the same time can cause a spike of roughly 28 kW, seven times higher than the base nighttime load. Using XBOS, it is possible to write a control application that can enact a demand shedding policy in a thermostat-agnostic manner.

A possible implementation of the demand shedding controller uses a simple form of admission control on which thermostats are allowed to call for heating or cooling. The controller subscribes to the temperature reading and heating and cooling setpoints of each thermostat in the building and takes as a configuration option the admission threshold, likely the permitted number of active RTUs or the maximum permitted demand. As the controller receives temperature updates from each thermostat, when a thermostat is close to its setpoint, the controller evaluates whether or not that thermostat could cause the threshold to be exceeded. If so, the controller can temporarily change the setpoint to delay RTU activation.

## Discussion

These examples demonstrate compelling points in the feature space of possible applications. With the reasonable and minimal instrumentation of a full building power meter and a collection of off-the-shelf networked thermostats, XBOS enables the construction of control and report applications that are beyond the capabilities of any single vendor solution available for the small-medium commercial building market.

Future work will use networked lights and plug load controllers to increase the degree of control in the Berkeley XBOS deployment. Additionally, cheap and pervasive environmental monitoring using a sensor mesh will allow applications to make more sophisticated control decisions. For example, a thermostat may use passive infrared and temperature sensors distribute its temperature sensing to all *occupied* rooms in its HVAC zone, or a lighting controller may leverage illumination sensors to compensate for day lighting. Furthermore, we plan to explore simple model-predictive control applications that automate the construction of models constructed from easily accessible historical data.

## Conclusion

XBOS hides the inherent complexity of building subsystems, providing a set of effective abstractions that compose nicely into a family of applications that are more than the sum of their parts. The driver-based hardware presentation layer simplifies the integration of devices from multiple vendors with simply defined control processes and schedules, and the data archival capabilities of XBOS enables applications to take a building’s performance history into account when making control decisions. XBOS hopes to promote a new class of portable and extensible applications for increasing the visibility into and efficient operation of buildings.

## References

Batra, N., A. Singh, and K. Whitehouse. "If You Measure It, Can You Improve It?: Exploring The Value of Energy Disaggregation." In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments, pp. 191-200. ACM, 2015.

Bhattacharya, A., J. Ploennigs, and D. E. Culler. "Short Paper: Analyzing Metadata Schemas for Buildings: The Good, the Bad, and the Ugly." In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments, pp. 33-34. ACM, 2015.

Claridge, D., D. Choinière, and N. Milesi-Ferretti. Annex 47 Report 3: Commissioning Cost-Benefit and Persistence of Savings. US Department of Commerce, National Institute of Standards and Technology, 2011.

Claridge, David E., W. D. Turner, M. Liu, S. Deng, G. Wei, Charles Culp, Hui Chen, and SoolYeon Cho. "Is Commissioning Once Enough?." Energy Engineering 101, no. 4 (2004): 7-19.

Dawson-Haggerty, Stephen, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. "sMAP: a simple measurement and actuation profile for physical information." In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, pp. 197-210. ACM, 2010.

Fierro, Gabriel, and D. E. Culler. "XBOS: An Extensible Building Operating System." In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments, pp. 119-120. ACM, 2015.

Katipamula, Srinivas, Ronald M. Underhill, James K. Goddard, D. Taasevigen, M. A. Piette, J. Granderson, Rich E. Brown, Steven M. Lanzisera, and T. Kuruganti. "Small-and medium-sized commercial building monitoring and controls needs: A scoping study." Pacific Northwest National Laboratory (PNNL), Richland, WA (US), Tech. Rep (2012).

Krioukov, Andrew, Gabe Fierro, Nikita Kitaev, and David Culler. "Building application stack (BAS)." In Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, pp. 72-79. ACM, 2012.

Nest Labs. March 2016. nest.com/works-with-nest.

Pérez-Lombard, Luis, José Ortiz, and Christine Pout. "A review on buildings energy consumption information." Energy and buildings 40, no. 3 (2008): 394-398.

Weng, Thomas, Anthony Nwokafor, and Yuvraj Agarwal. "Buildingdepot 2.0: An integrated management system for building analysis and control." In Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings, pp. 1-8. ACM, 2013.

Yan, C., S. Wang, and F. Xiao. "A simplified energy performance assessment method for existing buildings based on energy bill disaggregation." Energy and buildings 55 (2012): 563-574.