BOSS: Building Operating System Services

Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja,
Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler

Computer Science Division, University of California, Berkeley

Abstract

Commercial buildings are attractive targets for introducing innovative cyber-physical control systems, because they are already highly instrumented distributed systems which consume large quantities of energy. However, they are not currently programmable in a meaningful sense because each building is constructed with vertically integrated, closed subsystems and without uniform abstractions to write applications against. We develop a set of operating system services called BOSS, which supports multiple portable, fault-tolerant applications on top of the distributed physical resources present in large commercial buildings. We evaluate our system based on lessons learned from deployments of many novel applications in our test building, a four-year-old, 140,000sf building with modern digital controls, as well as partial deployments at other sites.

1 Introduction

Researchers and futurists working on ubiquitous and pervasive computing have long argued that a future full of personalized interaction between people and their environment is near [50, 43]. But this future has been primarily held back by the lack of a path from concept demonstration to broad deployment: developers have prototyped hundreds of interesting sensors [12, 37, 22], bringing new information about the world into a digital form, and tied these sensors together with actuators to provide interesting new capabilities to users. But invariably, these are developed and deployed as standalone, vertical applications, making it hard to share infrastructure investment among a variety of applications.

What is needed is an operating system to knit together existing pieces of infrastructure, Internet data feeds, and human feedback into a cohesive, extendable, and programmable system; *i.e.*, provide convenient abstractions and controlled access to shared physical resources. Doing so is a significant challenge, since such a system must bring together legacy systems with their own quirks, provide a path forward for new, native devices, and provide improved and simplified interfaces at multiple levels of abstraction. Existing buildings are not "programmable" in a meaningful sense: there are no layers of abstraction between the program and the system; programs may only access sensors and actuators at the very lowest level. As a result, applications are not *portable*, and it is impossible to provide *protected access* to an application, due to semantic mismatches between the level of policy and the level of access. We propose a new architecture for building control systems which, in addition to operating the machinery, provides for robust, portable application development and support many simultaneously running applications on the common physical infrastructure of a building. While buildings provide a concrete context, many of the ideas could be applied to other complex, connected physical systems.

We develop a collection of services forming a distributed operating system that solves several key problems that prevented earlier systems from scaling across the building stock. First, as buildings and their contents are fundamentally complicated, distributed systems with complex interrelationships, we develop a flexible approximate query language allowing applications to specify the components they interact with in terms of their relationship to other components, rather than specific hardware devices. Second, coordinated distributed control over a federated set of resources raises questions about behavior in the presence of failure. To resolve this concern, we present a transactional system for updating the state of multiple physical devices and reasoning about what will happen during a failure. Finally, there has previously been a separation between analytics, which deal with historical data, and control systems, which deal with real-time data. We demonstrate how to treat these uniformly in this environment, and present a time series service which allows applications to make identical use of both historical and real-time data.

Commercial buildings are an excellent environment in which to investigate such new systems. Many buildings already contain thousands of sense and actuation points which can be manipulated to provide new and surprising experiences without hardware retrofits. They have large energy bills, consuming about 73% of all electricity in the United States [49], making energy efficiency a compelling incentive for new investment. Furthermore, they are large enough and contain enough people to drive issues of scale, partial failure, isolation, and privacy.

2 Existing Building Systems

A large modern commercial building represents the work of thousands of individuals and tens or hundreds of millions of dollars of investment. Most of these buildings contain extensive internal systems to manufacture an indoor environment: to provide thermal comfort (heating and cooling), good air quality (ventilation), and sufficient lighting; other systems provide for life safety (fire alarms, security) and connectivity (networking). These systems are frequently provided by different vendors and have little interoperability or extensibility beyond the scope of the original system design.

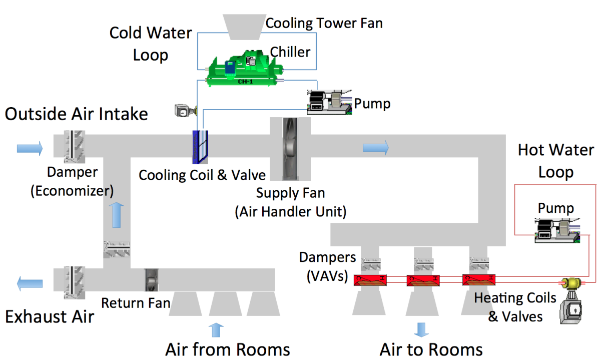


Figure 1: A typical process diagram of an HVAC system loop in a commercial building.

As an example of the complexity of many of these subsystems, Figure 1 shows one common design of a heating, ventilation, and air conditioning (HVAC) system for a large building. Air is blown through ducts, where it passes through variable air volume (VAV) boxes into internal rooms and other spaces. After circulating, it returns through a return air plenum where a portion is exhausted and the remaining portion is recirculated. The recirculated air is also mixed with fresh outside air, before being heated or cooled to a target supply temperature within an air handler unit (AHU), completing the loop. Other systems circulate hot and cold water for changing the air temperature. Many different control loops are present; the predominant control type is PID controllers used to meet setpoint targets for air pressure, temperature, and air volume.

This control in existing building systems operates on two levels. Direct control is performed in open and closed control loops between sensors and actuators: a piece of logic examines a set of input values and computes a control decision which commands an actuator. These direct control loops frequently have configuration parameters that govern their operation that are called setpoints; they are set by the building operator, installer, or engineer. Adjusting setpoints and schedules forms an outer logical loop, known as supervisory control. This logical distinction between types of control is typically reflected physically in the components and networking elements making up the system: direct control is performed by embedded devices called Programmable Logic Controllers (PLCs) that are hard-wired to sensors and actuators, while supervisory control and management of data for historical use is performed over a shared bus between the PLCs. This architecture is natural for implementing direct control loops since it minimizes the number of pieces of equipment and network links information must traverse to affect a particular control policy, making the system more robust, but it provides no coordinated control of distinct elements, and hard boundaries which are difficult to overcome. This collection of equipment is collectively known as a Building Management System (BMS).

The BMS is typically configured, managed, and programmed through a head-end node sitting on the shared bus. This node is responsible for providing an operator interface, storing historical "trend" data, and providing a point at which to reprogram the other controllers on the bus. It may also be a point of integration with other systems and therefore support some amount of reprogrammability or external access; for this reason, it can be a natural point of access to existing systems.

3 Design Development

To articulate the design of an operating system for buildings, we introduce three concrete, novel applications developed through research programs on our testbed building. The essential commonality is that all involve substantial interaction between components of building infrastructure, substantial computational elements, and building occupants, rather than simply providing a new interface to existing controls.

3.1 Motivating Applications

Ordinarily, the temperature within an HVAC zone is controlled to within a small range using a PID controller. The drive to reach an exact setpoint is actually quite inefficient, because it means that nearly every zone is heating or cooling at all times. A more relaxed strategy is one of *floating*: not attempting to affect the temperature of the room within a much wider band; however this is not one of the control policies available in typical commercial systems even though numerous studies indicate that occupants can tolerate far more than the typical $2°F$ variation allowed [5]. Furthermore, the minimum amount of ventilation air provided to each zone is also configured statically as a function of expected occupancy; however the actual requirement is stated in terms of fresh, outside air per occupant. The *HVAC optimization* application uses occupancy information derived from network activity, combined with information about the mix of fresh and return air currently in use to dynamically adjust the volume of ventilation air to each zone.

A second application was developed to improve comfort by giving occupants direct control of their spaces, inspired by previous work [17]. Using a smart-phone interface, the *personalized control* application gives occupants direct control of the lighting and HVAC systems in their workspaces. The application requires the ability to command the lights and thermostats in the space.

A third application is an *energy audit* application for our highly-instrumented building. Researchers input information about the structure of the building and the relationship between sensors and devices. This requires access to a uniform naming structure and streaming sensor data coming from physically-placed sensors. The former captures relationships between locations within the building, sensors, and loads (energy consumers) and the latter provides up-to-date information about physical measurements taken in locations throughout the building. This combination of data and metadata allow dashboarding and occupant feedback to access fine-grained slices of data – for instance, displaying the total energy consumed by all plug-loads on a particular floor.

Many other building applications have been developed in prior work including demand response [39], peak-price minimization [35], and occupant feedback [38]. All of these would benefit from a robust, common framework for controlling the building and the ability to run alongside other applications.

3.2 Architectural Implications

Experience with the *ad hoc* development of these kinds of applications led us to conclude that better abstractions and shared services would admit faster, easier, and richer application development, as well as a more fault tolerant system. The *HVAC optimization* application highlights the need for real-time access, locating the appropriate actuator for each temperature control unit, and replacing the local control logic with something new. Access to historical data is vital for training models and evaluating control strategies.

The *personalized climate control* application highlights the need for the ability to outsource control, at least temporarily, to a mobile web interface in a way that reverts gracefully to local control. It also integrates control over multiple subsystems that are frequently physically and logically separate in a building: HVAC and lighting.

The *energy audit* application highlights the need to couple semantic information with streaming sensor data in a uniform fashion and a way to meaningfully combine it with raw sensor data. It also emphasizes the need for real-time data cleaning and aggregation. Sensor feeds can be quite dirty, often missing values or containing errant data.

4 Design

The BOSS architecture consists of six main subsystems shown in Figure 2: (1) hardware abstraction and access abstraction; (2) naming and semantic modeling; (3) real-time time series processing and archiving; (4) a control transaction system; (5) authorization; and finally (6) running applications. The hardware abstraction layer elevates the plethora of underlying sensors and actuators to a shared, RESTful level and places all data within a shared global namespace, while the semantic modeling system allows for the description of relationships between the underlying sensors, actuators, and equipment. The time series processing system provides both real-time access to all underlying sensor data as well as stored historical data, and common analytical operators for cleaning and processing the data. The control transaction layer defines a robust interface for external processes wishing to control the system that is tolerant of failure and applies security policies. Last, "user processes" comprise the application layer. We expand on the design of each of these services below.

The mapping of these components to fault domains determines key properties of the overall system. The HPL must be physically co-located with the machinery it monitors and controls; failure of these components will prevent undoing actions taken by applications, although built-in control strategies provide the most basic level of fallback control. The transaction manager coordinates control over a set of HPL points and so it, combined with the set of HPL services it manages, determines a second, wider fault domain: this component forms the boundary at which other OS components can fail and still provide guaranteed behavior. The placement of the other services is flexible but impacts the availability of the resulting service; any other service failing could cause an application to crash.

4.1 Hardware Presentation Layer

Building systems are made up of a huge number of specialized sensors, actuators, communications links, and controller architectures. A significant challenge is overcoming this heterogeneity by providing uniform access to these resources and mapping them into corresponding virtual representations of underlying physical

Architectural component	Functional requirements	Placement
Hardware presentation layer	Expose the primitive low-level operations of hardware using a common interface.	Distributed as close to the physical sensors as possible (ideally, co-located).
Control transaction manager	Provide "all or nothing" semantics when applying control inputs; provide rollback of actions on failure, cancellation, or expiration.	Within the same failure domain as the HPL used to affect the changes.
Hardware abstraction layer	Map the low-level functions of the physical hardware to higher-level abstractions.	Anywhere.
Time series service	Maintain a history of readings from the sensors and actuators; provide application interface to data.	Replicated; may be offsite.
Authorization service	Approve application requests for access to building resources.	Anywhere.
Control processes	"User processes" implementing custom control logic.	Anywhere.

Table 1: Architectural components of a Building Operating System

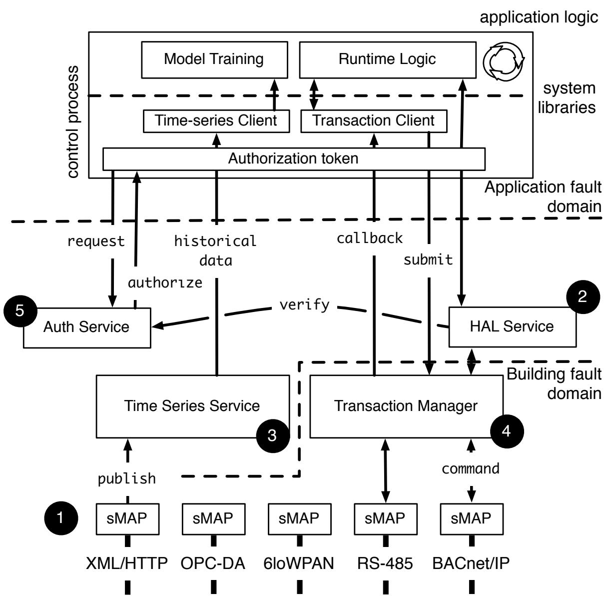


Figure 2: A schematic of important pieces in the system. BOSS consists of (1) the hardware presentation layer, the (2) hardware abstraction layer, the (3) time series service, and the (4) control transaction component. Finally, the (5) authorization service determines access controls. Control processes sit on top and consume these services.

hardware. At the lowest level is a Hardware Presentation Layer. The HPL hides the complexity and diversity of the underlying communications protocols and device interfaces, presenting hardware capabilities through a uniform, self-describing protocol. The HPL abstracts all sensing and actuation by mapping each individual sensor or actuator into a *point*: for instance, the temperature readings from a thermostat would be one sense point, while the damper position in a duct would be represented by an actuation point. These points produce *time series*, or streams, consisting of a timestamped sequence of readings of the current value of that point. The HPL provides a small set of common services for each sense and actuation point: the ability to read and write the point; the ability to subscribe to changes or receive periodic notifications about the point's value, and the ability to retrieve and append simple key-value structured metadata describing the point.

In order to provide the right building blocks for higher level functionality, this layer includes:

Naming: each sense or actuation point is named with a single, globally unique identifier. This provides canonical names for all data generated by that point for higher layers to use.

Metadata: most traditional protocols have limited or no metadata included about themselves, or their installation; however metadata, is incredibly important for the interpretation of data and for developing portable applications. The HPL allows us to include key-value metadata tags describing the data being collected to consumers.

Buffering and Leasing: many sources of data have the capability to buffer data for a period of time in case of the failure of the consumer; the HPL uses this to guard against missing data wherever possible. For actuators, safely commanding them in a fault tolerant way requires associating each write with a lease.

Discovery and Aggregation: sensors and the associated computing resources are often physically distributed with low-powered hardware. The HPL provides a mechanism to discover and aggregate many sensors into a single source on a platform with more resources, to support scalability.

This functionality is distributed across the computing resources closest to each sensor and actuator; ideally it is implemented natively by each device, although for legacy devices use a gateway or proxy. The HPL provides a small set of common services for each sense and actuation point: the ability to read and write the point; the ability to subscribe to changes or receive periodic notifications about the point's value, and the ability to retrieve and append simple key-value structured metadata describing the point.

4.2 Hardware Abstraction Layer

Unlike computer systems, buildings are nearly always custom-designed with unique architecture, siting, layout, mechanical and electrical systems, and control logic adapted to occupancy and local weather conditions. The

HAL allows applications to inspect these differences at a high level of abstraction, crucial for application portability. To do this, the HAL provides an approximate query language [29] allowing authors to describe the particular sensor or actuator that the application requires based on the relationship of that component to other items in the building, rather than hardcoding a name or tag. Applications can be written in terms of high-level queries such as "lights in room 410," rather than needing the exact network address of that point. The query language allows authors to search through multiple views of underlying building systems, including *spacial*, where objects are located in three-dimensional space; *electrical*, describing the electrical distribution tree; *HVAC*, describing how the mechanical systems interact; and *lighting*.

The HAL also abstracts the logic used to control building components such as pumps, fans, dampers, chillers, using a set of drivers to provide standard interfaces. Drivers provide high-level methods such as `set_speed` and `set_temperature` that are implemented using command sequences and control loops over the relevant HPL points. These drivers provide a place to implement device-specific logic that is needed to present standardized abstractions on top of eclectic hardware systems.

Drivers and applications use this functionality to determine locking sets, necessary for coexisting with other applications. For instance, an application that is characterizing the air handler behavior might want to ensure that the default control policy is in use, while varying a single input. It could use the approximate query language to lock all points on the air handler, excluding other processes. Since different models of the same piece of equipment may have different points even though they perform the same function, it is essential that applications can control sharing at the level of functional component rather than raw point name.

4.3 Time series service

Most sensors and embedded devices have neither the ability to store large quantities of historical data nor the processing resources to make use of them; such data are extremely important for historical analyses, model training, fault detection, and visualization. The challenge is storing large quantities of these data efficiently, while allowing applications to make the best use of them in near real-time for incorporation into control strategies. In existing systems, most historical data are goes unused because they are difficult to access and make sense of. Applications typically access data either by performing range queries over timestamps and streams, or by subscribing to the latest values. For instance, a typical query might train a model based room light-level readings for a period of one month, touching hundreds of millions of values. Even a modest-sized installation will easily have tens of billions of readings stored, with new data from the HPL mostly appended to the end of the time series. Finally, the data are usually dirty, often having desynchronized timestamps requiring outlier detection and recalibration before use.

The time series service (TSS) provides a low-latency application interface for accessing the large repository of stored data at different granularities. It consists of two parts: a stream selection language and a data transformation language. Using the stream selection language, applications can inspect and retrieve metadata about time series. The data transformation language allows clients to apply a pipeline of operators to the retrieved data to perform common data cleaning operations. This moves common yet complex processing logic out of applications, allowing them to focus on making the best use of the data, and also enables the possibility of optimizing common access patterns.

4.4 Control transactions

BOSS applications typically take the form of either coordinating control among multiple resources, which would otherwise operate independently as in the HVAC optimization, or extending control beyond the building to other systems, as in the personalized control or electric grid responsive control. The challenge is doing so in a way that is expressive enough to implement innovative new control algorithms, yet is robust to failure of network elements and controllers. Control algorithms that involve users or Internet-based data feeds should survive the failure of the parts of the control loop that run outside of the building without leaving any building equipment in an uncertain state. Therefore, we use a *transaction* metaphor for affecting changes to control state. Transactions in database systems are a way of reasoning about the consistency guarantees made when modifying multiple pieces of underling state; within BOSS, we use transactions as a way of reasoning about what happens when collections of control actions are performed.

A control transaction consists of a set of actions to be taken at a particular time: for instance, a coordinated write to multiple actuators. Actions at this level operate at the level of "points" – individual actuator outputs; the HAL uses its system model and driver logic to translate high-level requests into point-level operations that may include reads, writes, and locks.

To ensure reliability, control transactions require a *lease time* during which actions are valid, a *revert sequence* specifying how to undo the action, and an *error policy* stating what to do in case of a partial failure. When the lease expires, the transaction manager executes the revert sequence, which restores control of the system to the next scheduled direct controller. The ability to revert transactions provides the fundamental building block for allowing control of the building to be turned over to more sophisticated and less-trusted applications, while providing baseline control. Actions may also be reverted on a partial failure, depending on the error policy; for instance, if the transaction manager cannot acquire a lock or the write to the underlying device fails. Revert sequences are provided for each action and can be thought of as the "inverse action" that undoes the control input. We require there to be a "lowest common denominator" control loop present that is able to run the building in its default (although potentially inefficient) operation. In this way, applications can always simply release control and the building will revert to its default control regime.

To support multiple applications, each point-level operation also is associated with a *priority level* and a *locking strategy*. These allow multiple higher-level processes or drivers to access the underlying points, while providing a mechanism for implicit coordination. Using a concept borrowed from BACnet, writes are performed into a "priority array" – a set of values that have been written to the point ordered by priority level. The actual output value is determined by taking the highest priority write. Although it provides for basic multiprocessing, the BACnet scheme has several problems. Without leases, a crashing application could leave the system locked in an uncertain state until its writes are manually cleared. Without notifications, it is difficult to determine if a particular write has been preempted by another process at a higher priority without periodically polling the array. The transaction manager adds notification, leasing and locking, allowing applications to be notified when their writes are preempted, or to prevent lower-priority processes from accessing the point.

4.5 Authorization service

In addition to providing high-level, expressive access to building systems, BOSS seeks to limit the ability of applications to manipulate physical resources. Most building operators will not turn over control to just anyone, and even for a trusted application developer, safeguards against runaway behavior are needed. The authorization service provides a means of authorizing principals to perform actions and is based on the approximate query language of the HAL; applications may be restricted by location (only lights on the fourth floor), value (cannot dim the lights below 50%), or schedule (access is only provided at night). This provides access control at the same semantic level as the operations to be performed.

BOSS checks access permissions on the level of individual method call and point name in the HAL and HPL using a two-stage approve/verify process. Applications first register their *intent* to access a point or method name with the service and what arguments they will call it with. The intents may either be automatically approved or presented to a building manager for approval. Security and safety checks are performed at time-of-use on each method call, providing the ability to revoke access. Verifying access permissions at time-of-use using an online server rather than at time-of-issue using signed capabilities has negative implications for availability and scalability, as it places the authorization service on the critical path of all application actions. However, we found the ability to provide definitive revocation a critical functionality necessary to convince building managers that the system is safe. This is one place where practical considerations of the domain won over our bias against adding more complexity to the command pathway.

4.6 Control processes

Updates to building control state are made atomically using control transactions; however, these are often part of larger, more complex long-lived blocks of logic. These is known as a "control process" (CP) and are analogous to a user process; each of our motiving applications is implemented as a control process in BOSS. CPs connect to services they require, such as the time series service, HAL, and transaction managers, and manage the input of control actions. Because of the careful design of transactions and the TSS, there are few constraints on where control processes can be placed in the computing infrastructure; if they fail or become partitioned from the actuator they control, the transaction manager will simply "roll back" their changes and revert to a lower-priority CP which has not experienced partition or failure and ultimately to the hard-coded control strategy.

5 Implementation and Evaluation

To evaluate our architecture for building software systems, we have developed a prototype implementation of the Building Operating System Services. BOSS implements all system components, and is currently being used by researchers in a living lab context. The system is built mostly in `Python`, with `C` used for certain performance-critical parts; all together, it is about 10,000 lines of non-application source code. The system uses a service-oriented design, with canonical DNS names used for finding other services; most services communicate using RESTful interfaces exchanging JSON objects. In cases where authentication is required, two-sided SSL is used.

We have fully installed BOSS on our test building, Sutardja Dai Hall: a four-year-old, 140,000 square foot building containing mostly open "collaboratory" spaces alongside faculty offices on the UC Berkeley campus. Additionally, we have partially installed BOSS in many other buildings; we have performed full BMS integration in two other campus buildings, and have used the

HPL for data collection and analysis in around 100 other buildings. The system is running many applications, including our motivating examples; we study these actual applications to illustrate how BOSS achieves its goals.

In our implementation, the HPL and the transaction manager run physically co-located with the BMS computer. Because the HPL buffers data when the time series service is unavailable and the transaction manager will revert any actions taken by failing or disconnect processes, this provides the best fault tolerance to failures outside of the building. Other services run mostly in a server room, although some applications are hosted on cloud services such as EC2. The failure or partition of an application from any of the services generally causes the application to fail, with the transaction manager reverting any actions taken once its leases expire.

5.1 Hardware Presentation

The presentation layer allows higher layers to retrieve data and command actuators in a uniform way. Our HPL is a revised version the Simple Measurement and Actuation Profile [13, 14], which provides RESTful access to data sources and actuators, exposing the command resource tree shown in Figure 3.

```
/data/                    # all timeseries and collections
    {
        "Contents" : ["sensor0"],
        “Metadata” : { “SourceName” : “Example sMAP Source” },
    }
/data/sensor0
    { "Contents" :["channel0"] },
/data/sensor0/channel0
    {
        "uuid" : "a7f63910-ddc6-11e0-8ab9-13c4da852bbc",
        "Readings" : [ [1315890624000, 12.5 ] ]
    }
/reports/                 # data destinations
```

Figure 3: Resource tree exported by sMAP. *Sensors and actuators are mapped to time series resources identified by UUIDs. Metadata from underlying systems are attached as key-value tags associated with time series or collections of time series.*

Ease of integration is key when interfacing with existing systems. The sMAP library[1] takes care of the mechanics of providing the external interface, and allows driver writers to focus on implementing only sensor or actuator-specific logic. It uses abstracted drivers that separate the device-specific logic needed for talking with a device (native communications protocols, *etc*) from the site-specific configuration (network locations, sampling rates, *etc*). To demonstrate its flexibility and versatility, we have implemented around 25 driver modules which integrate with three major BMS vendors, various low-power wireless devices, several different three-phase electric meters, weather stations, and detailed operations data from all major US electric grids to enable control strategies taking account of time-of-use pricing and renewable energy availability; together, more than 28,000 streams are present in the HPL[2]. We interface with the building management system of our test building primarily over BACnet.

5.2 Hardware Abstraction Layer

The hardware abstraction enables application portability in two ways: it supports queries over the relationships between building components, and provides drivers with standardized methods for making control inputs.

5.2.1 Semantic Query Language

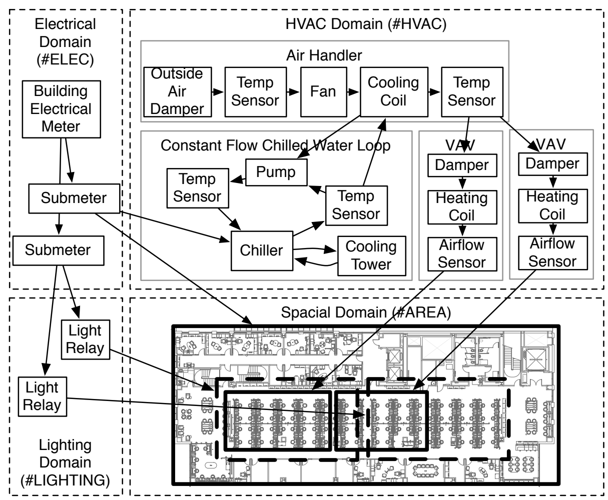


Figure 4: Partial functional and spacial representation of our test building. *Directed edges indicate a “supplies” or “feeds into” relationship. Queries are executed by searching the graph.*

The query interface allows applications to select objects based on type, attributes, and functional or spacial relationships, allowing programmers to describe the particular sensor or actuator that the application requires rather than hardcoding a name or tag. This allows applications to be portable across buildings with different designs[29].

Queries are expressed in terms of metadata tags from the HPL, and relationships indicated by $<$ and $>$ operators with $A > B$ meaning that A supplies or feeds into B. For example, an air handler might supply variable air volume (VAV) boxes that supply rooms; a whole building power meter may feed into multiple breaker panels that supply different floors. The execution engine evaluates queries by searching a directed graph of objects. Figure 4 shows a partial rendering of the functional and spacial relationship graphs. Objects are exposed by drivers and can be low-level (*e.g.*, damper, sensor, fan) or high-level (*e.g.*, air handler, chilled water loop). Directed edges indicate the flow of physical media, *i.e.*air, water, electricity. Tags describe the object type and functionality. We also represent spatial areas, defined as polygons on floor maps of the building and stored in a GIS database.

5.2.2 Drivers

BOSS drivers are implemented as persistent objects within a global namespace; CPs obtain references to drivers using the semantic query language from the HAL. To make driver code reusable in an environment where very little can be assumed about the underlying technology, drivers present both uniform top-level interfaces to the components they represent, as well as a template-based bottom interface allowing them to be automatically instantiated when the underlying HAL points have been tagged with appropriate metadata.

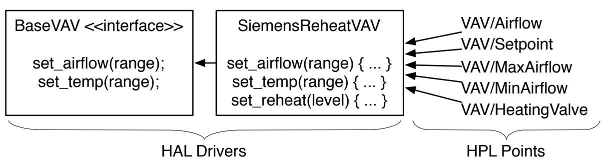


Figure 5: Drivers expose a set of methods on top, and are bound to HPL points based on metadata provided by the HPL. *Here, a Siemens VAV driver exposes the methods common to all VAVs as well as an additional* `set_reheat` *method.*

Figure 5 shows a schematic of a VAV driver: to illustrate these relationships, the `BaseVAV` interface which provides methods for controlling setpoint and temperature has been extended by the `SiemensReheatVAV` driver to provide an additional, non-standard `set_reheat` method. Drivers encapsulate device-specific logic needed to provide a standardized interface on top of the actual hardware present which may be more sophisticated than what is installed in a given building. For instance, the standard VAV control method allows applications to set a target temperature range (dead band). However, for many VAVs, including those in our test building, the width of this band is impossible to change once installed. Therefore, the driver emulates the desired behavior by manipulating the set point as the temperate moves in and out of the dead band.

Our driver system includes representations of many common building infrastructure elements such as VAVs, dampers, water pumps, air handlers, economizers, lights, light switches and transformers – and maps them into canonical representations that are placed into the metadata graph. Using drivers' bottom layer templates, appropriate drivers are loaded in places where HPL points have matching metadata, subject to hand checking.

5.3 Time Series Service

The time series service is responsible for **storing**, **selecting**, and **cleaning** both real-time and historical data. Figure 6 shows an example query exercising all three of these functions. The TSS contains three main components: the `readingdb` historian[3] provides compressed, low-latency, and high-throughput access to raw time series data. A selection engine performs SQL-to-SQL compilation on user queries, allowing them to flexibly select data streams on the basis of the tags applied by the HPL. A data transformation component applies domain-specific operations to the data.

5.3.1 readingdb

Many building energy products build on start SQL databases which are disappointing at scale. Figure 7 compares `readingdb` performance to MySQL (using both InnoDB and MyISAM storage engines) and PostgreSQL tuned for time series data on a representative usage pattern. Here, the database is loaded with synthetic data (simulating a trickle load), and periodically stopped to query a fixed-size set of data as well as to measure how large the stored data are on disk. Because MyISAM appends all records to the end of the volume, inserts are uniformly very cheap; however, query performance is poor and furthermore scales with the size of the database rather than the size of the results set. Postgres and InnoDB keep data ordered by time on disk resulting in more predictable query performance, but have more expensive insert paths; furthermore the B+-tree indexes scale poorly when presented with a large number of leaf keys. `readingdb`'s bucketing algorithm mitigates these issues (while still using an index for fast random access) by packing neighboring records together using only a single key, achieving an order of magnitude better compression than the other tree-based schemes.

5.3.2 Data Selection

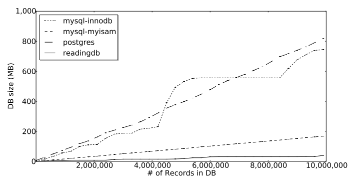
With tens of thousands of data streams present, finding the right one can be a challenge. The HPL provides the basis for this naming by identifying each data stream with a unique identifier (a `UUID`), and attaching key-value metadata to it. The time series service provides the mechanism to apply complex queries to these streams to locate them on the basis of the metadata, *i.e.*, an "entity-attribute-value" schema. Our system uses an SQL-to-SQL complier to transform logical queries in key-space to queries on the underlying database schema, allowing users to specify any attribute in the HPL. Line 3 in Figure 6 is an example of easily locating all datacenter power feeds.

5.3.3 Data Transformation

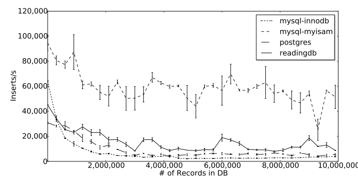
The processing pipeline allows operators to inspect both data (time, value vectors) as well as metadata: operators are first bound to the actual streams to be pro-

```
1: apply sum(axis=1) < missing < paste < window(mean, field="minute", width=15)
2: to data in ("4/20/2012", "4/21/2012")
3: where Metadata/Extra/System = 'datacenter' and Properties/UnitofMeasure = 'kW'
```

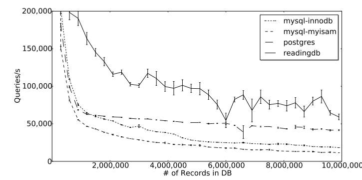
Figure 6: Example query executed by the time series service. *Line 1 uses a pipeline of four data cleaning operators to aggregate by resampling data in 15-minute intervals and then filtering missing data, Line 2 selects a range of time from the* `readingdb` *storage manager, and Line 3 queries metadata to locate datacenter power feeds.*



(a) Size of on-disk files



(b) Insert performance

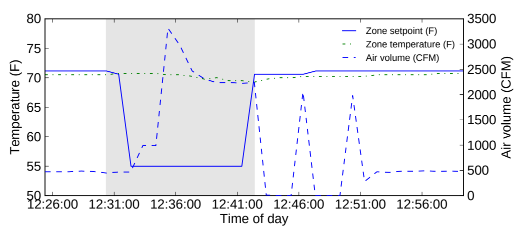


(c) Query performance

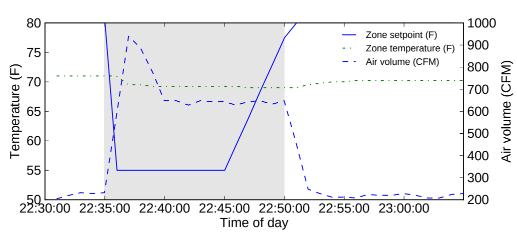
Figure 7: `readingdb` time series performance compared to two relational databases. *Compression keeps disk I/O to a minimum, while bucketing prevents updating the B+-tree indexes on the time dimension from becoming a bottleneck. Keeping data sorted by stream ID and timestamp preserves locality for range queries.*

cessed and since each operator can inspect the metadata of the input streams, it is possible to implement operators that transform data based on the metadata such as a unit or timezone conversion. Using simple combinations of these operators, queries can interpolate time-stamps, remove sections with missing data, and compute algebraic formulas over input data. Extending the set of operators is simple since we provide support for wrapping arbitrary `Python` functions which operate on vector data; in particular, we have imported most of the `numpy` numerical library automatically.

5.4 Transaction Manager



(a) Unstable behavior around a transition



(b) Specialized reversion sequences can deal with this problem

Figure 8: Drivers may implement specialized reversion sequences to preserve system stability when changing control regimes.

Submitted transactions become runnable once the start time has passed. The **scheduler** chooses the next action from among the runnable actions, taking into account considerations of both upper and lower layers. It considers the priorities of the various runnable actions as well as the concurrency requirements of the underlying hardware. For instance, some devices are present on a shared 9600-baud RS-485 bus; Internet-style applications can easily overwhelm such a limited resource. Priorities are implemented as separate FIFO queues at each priority level. Once actions are scheduled, actual execution is passed off to controller components that perform the action by communicating with the appropriate sMAP devices and proxies. When the lifetime of a transaction expires, it is canceled, or it encounters an error, the revert method is used to enqueue new commands to undo the previous control inputs.

The naïve reversion policy would simply clear any writes made; however, Figure 8(a) illustrates one problem with this method. Here, the setpoint is reduced at around 12:31, causing air volume to increase and room temperature to fall. However, when this change is reverted at 12:41, the default commercial controller which takes over becomes confused by the unexpected deviation from setpoint, causing the damper position (and thus air volume) to oscillate several times before finally stabilizing. Understanding and dealing with this issue is properly the concern of a higher-level component such as a VAV driver; to allow this, some drivers provide a custom revert action along with their inputs. These actions consist of restricted control sequences requiring no communication, replacing the default reversion policy. In Figure 8(b), the VAV driver uses a custom revert sequence to gradually release control.

6 Applications

We further evaluate BOSS in two ways: first, we examine how the system architecture makes implementing our three motivating applications simpler and more concise, while showing how it helps applications to coexist. Second, we provide a survey of other applications which have been implemented using BOSS, providing evidence of the system's generality.

6.1 HVAC Optimization

The HVAC optimization control process consists of two strategies: temperature floating and ventilation adjustment. Building codes often require a rate of fresh air ventilation per room based on occupancy and room size [10, 5]. Keeping ventilation rates at the required minimum is highly desirable for energy savings since it reduces fan power and the need for air conditioning; however, this is difficult to do in traditional building control systems because separate control loops are in charge of varying the fresh air intake into the building, controlling the per-room airflow, and detecting occupants. Occupancy detection is a well-researched subject that is best performed by fusing data from many sensors [2, 1, 32] not normally available.

Figure 9 shows pseudocode implementing the airflow reductions. The code uses the HAL semantic query interface to find all dampers controlling fresh air intake and adjusts the ventilation rates for their downstream rooms – the more fresh air being brought into the building from the outside, the less airflow is required per room to maintain the required freshness. In the example, line 3 returns dampers servicing the two air handlers (`AH1A` and `AH2A` in our building), each of which services around 70 zones, which are found on line 4. We use a simple occupancy model based on time of day and class schedule obtained from a Google Calendar feed, and scale the ventilation as a function of the number of people. This demonstrates coordinated control across traditionally independent building components: on line 6, the actual fresh air intake setting is used to control the room ventilation requirements. Furthermore, a separate building with completely different ventilation layout would be able to run virtually the same control application.

```
1 proc = BossProcess(timeout=15min, auth_token=ABC)
2 while True:
3   for dmp in hal.find('#OUT_AIR_DMP > #AH'):
4     for vav in hal.find('#VAV < $%s' % dmp.name):
5       occ = model.estimate_occupancy(vav)
6       vav.set_min_airflow((vav.min_fresh_air() /
7             dmp.get_percent_open()) * occ)
8   time.sleep(15*60)
```

Figure 9: Ventilation component of the HVAC optimization application.

6.2 Personalized Control

A second application, a personalized control system, takes direct occupant input to adjust room temperatures and ventilation. One of its key features is the ability to temporarily blast warm or cold air into the space in response to a user request. Fault tolerance is crucial in this application; blasts must be reverted even if the control process crashes to ensure occupant comfort and avoid wasting energy. Figure 10 shows the execution flow of the personalized control application and the error handling in response to an emulated crash.

The application writes to a room setpoint in response to a user request but shortly thereafter crashes. The transaction manager reverts the blast action by undoing the submitted transaction. A subplot of room temperature taken while executing this control flow is also shown in Figure 10. Temperature drops while the cold blast is running and reverts to normal after the application crashes. Unlike traditional computer systems, reverting the room temperature takes time as the space slowly warms back up to steady state.

We run the personalized control application concurrently with the HVAC optimization application. Since both apps access the same VAV point, some coordination is required to ensure correct behavior. In this case, the HVAC optimization application can coexist with the personal control application: if its commands are overridden at a higher priority, it simply regains control whenever the higher priority application is finished. However, the inverse situation is not acceptable: since users expect an immediate response when initiating a blast, the application locks the VAV points with an exclusive lock, so that if it is itself overridden, it will immediately cause the transaction to abort and display an error message.

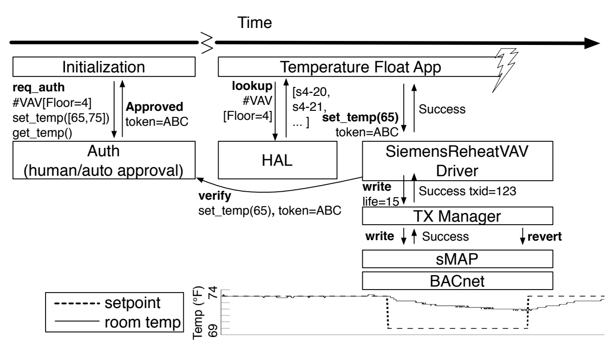


Figure 10: Execution flow of the personalized control application responding to a user request for cooling. *After the control process crashes, the transaction manager automatically reverts past actions.*

6.3 Auditing and Baselining

We use the auditing application to compute energy savings from the HVAC optimization and personal controls. The HPL and time series service allow easy access to historical data from all meters within the building. We use these data to train a baseline model of power consumption of the building in its standard operating regime, regressing against outside air temperature, time of day, and class schedules. This model is used to produce a new real-time baseline stream which appears as a new virtual feed in the HPL. Using this baseline, we can compare building performance before and after enabling our optimization and control algorithms. Figure 11 shows measured power consumption and the modeled power baseline. Power consumption drops by 28kW, about 17%, after launching our optimization apps.

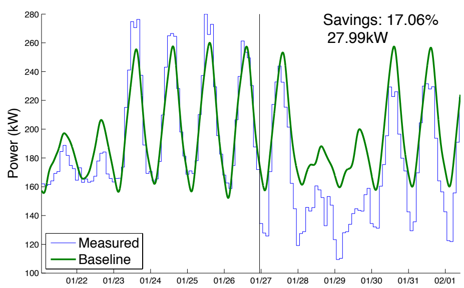


Figure 11: Energy use before and after starting our HVAC control applications in our test building showing energy savings of approximately 17%.

6.4 Deployment and Application Survey

To begin quantifying the generality of BOSS is, we surveyed a number of users inside and outside our group who have written applications to produce Table 2. These include the motivating applications above, as well as applications which perform model-predictive control of various system components and conduct comfort analyses of building data. Overall, application writers felt that their ability to spend time on their actual problems such as system modeling or producing visualizations of the data was much improved by operating at a higher level of abstraction; furthermore many appreciated the ability to write application code that might have bugs and yet be assured that the system would fail gracefully.

7 Related Work

7.1 Ubiquitous Computing

There have been many attempts to provide programmable abstractions to make it easier to run tasks on devices, predominantly in homes. For instance, ICrafter [40] integrates devices in an intelligent workspace into a service-oriented framework, used to generate user interfaces, while ubiHome [18] applies semantic web techniques to describe the services provided by a variety of consumer devices. Several "living laboratories" such as Sensor Andrew and HOBNET [42, 20] also work to make experimentation with ubiquitous computing environments simple and scalable, performing complimentary research on communications protocols.

Microsoft HomeOS [15] is a related attempt to provide high-level abstractions to make programming physical devices simpler. All intelligent devices in a home are presented as PC peripherals, and applications are implemented as modules loaded into a single system image; these applications are strongly isolated using run-time containers and `Datalog` descriptions of access rules. Our work takes a fundamentally different approach: we allow applications to be distributed and enforce safety at the transaction manager and driver services, at the cost of limiting control over the behavior of applications. By allowing control to be decentralized, we allow the system to be configured so as to trade off partition-tolerance with cost. Unlike HomeOS, we allow applications to be written in terms of components' relationship with other components, and provide efficient access to historical data; these functions are essential for scalability.

7.2 Metadata

Industry Foundation Classes [23] specify models for structural, mechanical, and electrical aspects of buildings. IFCs are intended to describe building design and facilitate sharing of information among design and construction teams. IFC includes classes for HVAC equipment and a connectivity model for building a directed graph of objects [9]. Project Haystack [41] uses a list of tags and rules about their use to describe building components. Liu, *et al.* [31] focus on integrating existing metadata from multiple sources and devising a common data representation for use by applications. Our work is complementary and focuses on how applications can conveniently make use of available building controls portably and at a higher level of abstraction.

7.3 Protocols

OLE for Process Control (OPC) is commonly used for controls interoperability. Based on DCOM, OPC accomplishes some of the same goals as the HPL [36], and contains a component for accessing historical data: OPC-HDA; however it does not provide the ability to apply analytical operators to the stored data and also can only locate data by point name. BACnet, or the Building Automation and Control Network protocol, also provides a standardized interface for accessing devices in a building [4]. It provides for the discovery of BACnet objects on a local subnet, and the ability for these objects to export standardized services, such as ReadValue and WriteValue. Other industrial controls protocols like WirelessHART [52], Modbus [34], and many others provide low-level access to field devices similar to the level of the HPL; however these form only the lowest-level building-blocks of a complete system. Protocols in the

Name	Description	Sensors Used	Actuators Used	Type of Control	HPL Adaptors	External Examples
Demand Ventilation	Ventilation rates are modulated to follow occupancy, and room temps can float at night.	VAV temps, airflow, & CO2	VAV Damper Positions	Supervisory	BACnet, 6loWPAN	
Supply Air Temp Control	Air Handling Unit supply air temp (AHU SAT) is optimized using model-predictive control (MPC) [6].	AHU SAT, VAV temps & airflow	AHU SAT	Supervisory	BACnet	[51], [30], [8], [3]
VAV Control	Individual variable air-volume boxes are exercised to create detailed models of their response.	VAV temps & airflow	VAV damper positions	Direct	BACnet	
Building Audit	Loads throughout the building are surveyed to enable DR and energy efficiency efforts.	Plug-load meters, submeters	N/A	N/A	ACme [26], BACnet	[19], [21], [33], [11], [46]
Personal Building Control	Users are presented with web-based lighting and HVAC control of building systems.	Light power, VAV temps	Light level, VAV airflow	Direct	BACnet	[28], [33], [24], [16]
Personal Comfort Toolkit	Enables analysis of air stratification in building system operation.	Zone temperatures, stratification	N/A	N/A	BACnet, CSV, 6loWPAN	
Demand Response	Responds to electric grid-initiated control signals requesting setback.	N/A	Airflow & temperature setpoints	Supervisory	BACnet	[27], [47]

Table 2: Deployed applications making use of BOSS.

residential space like uPNP [25] and Zigbee [45] tend to have higher-level interoperability as a goal, typically defining device profiles similar to our driver logic; this work is valuable for defining common device interfaces but does not address failures when these interfaces are implemented by coordinating multiple devices.

7.4 Building Controls

A number of Building Management Systems enable building engineers to create applications. The Siemens APOGEE [44] system provides the Powers Process Control Language (PPCL) for design of control applications; PPCL allows for custom logic, PID loops, and alarms. ALC's LogicBuilder [7] allows for graphical construction of control processes by sequencing "microblocks," or control functions, from a library; however, the library of control functions is not extensible. Further, both of these systems can only interact with equipment physically connected to the panel on which the code is installed, limiting the use of external information in control decisions. Tridium provides the Niagara AX framework [48] for designing Internet-connected applications using its HPL-like interfaces to building equipment and external data sources. However, Tridium provides no semantic information about its abstracted components, limiting application portability.

8 Conclusion

Our work is a re-imagining building control systems of the future: secure, modular, extensible, and networked. Many of the problems with SCADA systems which appear in the popular media with increasing frequency can be linked to system architecture designed for another world: one without ubiquitous connectivity where systems are designed and used in isolation from other systems. This view is simply not realistic, but fixing the problems requires a fundamental re-architecting of the control system, drawing on distributed systems and Internet design discipline to design a system which is robust even while subject to Internet-scale attacks. Although challenging, we do not see an alternative because it seems inevitable that control systems will become ever more networked to provide increased efficiency and flexibility; the only question is whether the result will be robust and conducive to application development.

BOSS begins to provide these properties by introducing flexibility in several new places where it did not previously exist: the HPL provides equal access to all of the underlying data, while the transaction and control process metaphors allow true applications to exist, and fail safely. Looking forward, many challenges remain. Inferring the HAL with minimum manual effort is an important step to enabling this architecture in existing buildings. Much better tools are needed for incorporating existing BIM models, and tools for checking the derived model against real sensor data will be crucial since drawings rarely reflect the true state of the world. The emergence of software-defined networks also presents an interesting avenue for future exploration: if control intent is expressed abstractly, SDNs might be used to enforce access control and quality-of-service guarantees.

Acknowledgements

Thanks to our shepherd, Richard Mortier, and the anonymous reviewers for helpful comments. Domenico Caramagno, Scott McNally, and Venzi Nikiforov's support was instrumental in allowing us gain access to real commercial building control systems. This project was supported in part by the National Science Foundation under grants CPS-0932209 (LoCal), CPS-0931843 (ActionWebs), and CPS-1239552 (SDB). The sMAP project is generously supported by the UC Berkeley Energy and Climate Research Innovation Seed Fund.

Notes

[1] http://code.google.com/p/smap-data
[2] http://www.openbms.org
[3] http://github.com/stevedh/readingdb

References

[1] AGARWAL, Y., BALAJI, B., DUTTA, S., GUPTA, R. K., AND WENG, T. Duty-cycling buildings aggressively: The next frontier in hvac control. In *IPSN/SPOTS 2011* (2011).

[2] AGARWAL, Y., BALAJI, B., GUPTA, R., LYLES, J., WEI, M., AND WENG, T. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings* (2010).

[3] ALVAREZ, J., REDONDO, J., CAMPONOGARA, E., NORMEY-RICO, J., BERENGUEL, M., AND ORTIGOSA, P. Optimizing building comfort temperature regulation via model predictive control. *Energy and Buildings* (2013).

[4] AMERICAN SOCIETY OF HEATING, REFRIGERATING AND AIR-CONDITIONING ENGINEERS. *ASHRAE Standard 135-1995: BACnet*. ASHRAE, Inc., 1995.

[5] AMERICAN SOCIETY OF HEATING, REFRIGERATING AND AIR-CONDITIONING ENGINEERS. *ASHRAE Standard 55-2010: Thermal Environmental Conditions for Human Occupancy*. ASHRAE, Inc., 2010.

[6] ASWANI, A., MASTER, N., TANEJA, J., KRIOUKOV, A., CULLER, D., AND TOMLIN, C. Energy-efficient building hvac control using hybrid system lbmpc. In *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control* (2012).

[7] AUTOMATED LOGIC CORPORATION. Eikon LogicBuilder for WebCTRL. http://www.automatedlogic.com/product/eikon-logicbuilder-for-webctrl/.

[8] AVCI, M., ERKOC, M., RAHMANI, A., AND ASFOUR, S. Model predictive HVAC load control in buildings using real-time electricity pricing. *Energy and Buildings* (2013).

[9] BAZJANAC, V., FORESTER, J., HAVES, P., SUCIC, D., AND XU, P. HVAC component data modeling using industry foundation classes. In *System Simulation in Buildings* (2002).

[10] CA ENERGY COMMISSION. California's energy efficiency standards for residential and nonresidential buildings, 2008.

[11] CHENG, Y., CHEN, K., ZHANG, B., LIANG, C.-J. M., JIANG, X., AND ZHAO, F. Accurate Real-Time Occupant Energy-Footprinting in Commercial Buildings. In *Proc. of 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2012).

[12] COHN, G., STUNTEBECK, E., PANDEY, J., OTIS, B., ABOWD, G. D., AND PATEL, S. N. SNUPI: sensor nodes utilizing powerline infrastructure. In *Proceedings of the 12th ACM international conference on Ubiquitous computing* (2010), pp. 159–168.

[13] DAWSON-HAGGERTY, S., JIANG, X., TOLLE, G., ORTIZ, J., AND CULLER, D. sMAP: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010).

[14] DAWSON-HAGGERTY, S., KRIOUKOV, A., AND CULLER, D. E. Experiences integrating building data with smap. Tech. Rep. UCB/EECS-2012-21, EECS Department, University of California, Berkeley, Feb 2012.

[15] DIXON, C., MAHAJAN, R., AGARWAL, S., BRUSH, A. J., LEE, B., SAROIU, S., AND BAHL, P. An operating system for the home. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012).

[16] ERICKSON, V., AND CERPA, A. E. ThermoVote: Participatory Sensing for Efficient Building HVAC Conditioning. In *Proc. of 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2012).

[17] FOUNTAIN, M., BRAGER, G., ARENS, E., BAUMAN, F., AND BENTON, C. Comport control for short-term occupancy. *Energy and Buildings 21*, 1 (1994), 1 – 13.

[18] HA, Y.-G., SOHN, J.-C., AND CHO, Y.-J. ubihome: An infrastructure for ubiquitous home network services. In *Proceedings of the IEEE International Symposium on Consumer Electronics* (2007).

[19] HAY, S., AND RICE, A. C. The Case for Apportionment. In *Proc. of 1st ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2009).

[20] Holistic platform design for smart buildings of the future internet.

[21] Hsu, J., Mohan, P., Jiang, X., Ortiz, J., Shankar, S., Dawson-Haggerty, S., and Culler, D. HBCI: Human-Building-Computer Interaction. In *Proc. of 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2010).

[22] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (2006).

[23] ISO. Industry Foundation Classes, Release 2x, 2005.

[24] Jazizadeh, F., and Becerik-Gerber, B. Toward Adaptive Comfort Management in Office Buildings Using Participatory Sensing for End User Driven Control. In *Proc. of 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2012).

[25] Jeronimo, M., and Weast, J. *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003.

[26] Jiang, X., Ly, M. V., Taneja, J., Dutta, P., and Culler, D. Experiences with a High-Fidelity Wireless Building Energy Auditing Network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems* (2009).

[27] Krioukov, A., Alspaugh, S., Mohan, P., Dawson-Haggerty, S., Culler, D. E., and Katz, R. H. Design and Evaluation of an Energy Agile Computing Cluster. Tech. Rep. UCB/EECS-2012-13, EECS Department, University of California, Berkeley, 2012.

[28] Krioukov, A., Dawson-Haggerty, S., Lee, L., Rehmane, O., and Culler, D. A Living Laboratory Study in Personalized Automated Lighting Controls. In *Proc. of 3rd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2011).

[29] Krioukov, A., Fierro, G., Kitaev, N., and Culler, D. Building application stack (BAS). In *Proceedings of the 4th ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings* (2012).

[30] Liang, C.-J. M., Liu, J., Luo, L., Terzis, A., and Zhao, F. RACNet: A High-Fidelity Data Center Sensing Network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2009).

[31] Liu, X., Akinci, B., Garrett, J. H., and Berges, M. Requirements for a formal approach to represent information exchange requirements of a self-managing framework for HVAC systems. In *ICCCBE* (2012).

[32] Lu, J., Sookoor, T., Srinivasan, V., Gao, G., Holben, B., Stankovic, J., Field, E., and Whitehouse, K. The smart thermostat: using occupancy sensors to save energy in homes. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010).

[33] Marchiori, A., Han, Q., Navidi, W., and Earle, L. Building the Case For Automated Building Energy Management. In *Proc. of 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2012).

[34] MODICON, Inc., Industrial Automation Systems. Modicon MODBUS Protocol Reference Guide, 1996.

[35] Nghiem, T., Behl, M., Pappas, G., and Mangharam, R. Green scheduling: Scheduling of control systems for peak power reduction. In *Proceedings of the International Green Computing Conference and Workshops* (2011).

[36] OPC Task Force. OPC common definitions and interfaces, 1998.

[37] Park, T., Lee, J., Hwang, I., Yoo, C., Nachman, L., and Song, J. E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems* (2011), ACM.

[38] Piette, M. A., Kiliccote, S., and Ghatikar., G. Design of an energy and maintenance system user interface for building occupants. In *ASHRAE Transactions*, vol. 109, pp. 665–676.

[39] Piette, M. A., Kiliccote, S., and Ghatikar., G. Design and implementation of an open, interoperable automated demand response infrastructure. In *Grid Interop Forum* (2007).

[40] PONNEKANTI, S. R., LEE, B., FOX, A., FOX, O., WINOGRAD, T., AND HANRAHAN, P. ICrafter : A service framework for ubiquitous computing environments. In *Proceedings of the 3rd International Conference on Ubiquitous Computing* (2001).

[41] Project Haystack. http://project-haystack.org/.

[42] ROWE, A., BERGES, M., BHATIA, G., GOLDMAN, E., RAJKUMAR, R., GARRETT, J. H., MOURA, J. M. F., AND SOIBELMAN, L. Sensor Andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development 55*, 1.2 (Jan. 2011), 6:1 – 6:14.

[43] SATYANARAYANAN, M. Pervasive computing: vision and challenges. *IEEE Personal Communications 8*, 4 (August 2001), 10 –17.

[44] SIEMENS. APOGEE Building Automation Software. `http://w3.usa.siemens.com/buildingtechnologies/us/en/building-automation-and-energy-management/apogee/pages/apogee.aspx`.

[45] *Smart Energy Profile 2.0*. Zigbee Alliance, 2010.

[46] TAHERIAN, S., PIAS, M., COULOURIS, G., AND CROWCROFT, J. Profiling Energy Use in Households and Office Spaces. In *Proceedings of the 3rd Int'l Conference on Future Energy Systems (ACM e-Energy)* (2012).

[47] TANEJA, J., CULLER, D., AND DUTTA, P. Towards Cooperative Grids: Sensor/Actuator Networks for Renewables Integration. In *Proceedings of the 1st IEEE Int'l Conference on Smart Grid Communications* (2010).

[48] TRIDIUM. NiagaraAX. `http://www.tridium.com/cs/products_/_services/niagaraax`.

[49] U.S. DEPARTMENT OF ENERGY. 2011 buildings energy data book, 2012.

[50] WEISER, M. The computer for the 21st century. *SIGMOBILE Mobile. Computer Communication Review 3*, 3 (July 1999), 3–11.

[51] WEN, Y.-J., DIBARTOLOMEO, D., AND RUBINSTEIN, F. Co-simulation Based Building Controls Implementation with Networked Sensors and Actuators. In *Proc. of 3rd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)* (2011).

[52] *WirelessHART*. HART Communication Foundation, 2009.