Towards Programmable Buildings: A Study of System Design for Application Portability in Buildings

Andrew Krioukov

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-241
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-241.html

December 20, 2013

Acknowledgement

Many thanks to my advisor, David Culler for his mentorship and encouragement to constantly address the biggest challenges and to build practical, real-world systems.

Thanks to Domenico Caramagno for his invaluable insight into building operation and patience as I experimented with building controls. Thanks to Albert Goto, Venzi Nikiforov and Paul Wright for making our deployments possible.

This work benefited tremendously from many fruitful discussions and collaboration with all of my colleagues and friends in the LoCal research

group. Special thanks to Stephen Dawson-Haggery and Jay Taneja who were instrumental in the design and development of the foundations on which this work builds.

Finally, I am grateful to my family for their ongoing support and encouragement.

Towards Programmable Buildings:
A Study of System Design for Application Portability in Buildings

by

Andrew Krioukov

A thesis submitted in partial satisfaction of the
requirements for the degree of
Masters of Science

in

Computer Science - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor David Culler, Chair
Professor Edward Arens

Fall 2013

Abstract

Towards Programmable Buildings:
A Study of System Design for Application Portability in Buildings

by

Andrew Krioukov

Masters of Science in Computer Science - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor David Culler, Chair

Buildings consume 72% of electricity in the U.S. [30] and are a prime opportunity for software to improve sustainability and unlock new capabilities. Many commercial buildings have digital controls and extensive sensor networks that can be used to develop novel applications for saving energy, detecting faults, improving comfort, etc. However, buildings are custom designed, leading to differences in functionality, connectivity, controls and operation. As a result today's building applications are hard to write and non-portable. We present BAS, an application programming interface and runtime that addresses these issues using a fuzzy query API, a graph representation of building metadata and a hierarchical driver model for building components.

We demonstrate and evaluate BAS by exploring three applications enabled by this architecture. Specifically, we focus on a class of applications that incorporate occupant feedback into building operation. We develop a personalized lighting control application that uses existing lighting hardware and saves over 50% of lighting energy. We also develop a ventilation optimization and a personal HVAC control application, demonstrating that BAS enables rapid development and scale for building applications.

Contents

Acknowledgments

Many thanks to my advisor, David Culler for his mentorship and encouragement to constantly address the biggest challenges and to build practical, real-world systems.

Thanks to Domenico Caramagno for his invaluable insight into building operation and patience as I experimented with building controls. Thanks to Albert Goto, Venzi Nikiforov and Paul Wright for making our deployments possible.

This work benefited tremendously from many fruitful discussions and collaboration with all of my colleagues and friends in the LoCal research group. Special thanks to Stephen Dawson-Haggery and Jay Taneja who were instrumental in the design and development of the foundations on which this work builds.

Thanks to Randy Katz, Ed Arens and my colleagues at the Center for the Built Environment for asking challenging questions and helping improve this work.

Finally, I am grateful to my family for their ongoing support, encouragement and patience during my academic pursuits.

Chapter 1

Introduction

Buildings, where we spend over 90% of our time [31] and 72% of our electricity in the U.S. [30], are a prime opportunity for information technology to improve sustainability. However, the building sector is slow to innovate, with design lifetimes counted in the decades and limited budgets for improvements. Though changes in building codes exert some pressure on new buildings to incorporate new technologies that improve comfort and energy efficiency, little is generally done to improve existing buildings and their control systems.

Investigation of building applications Achieving portability in building application

Nearly half of existing commercial buildings contain digital control systems that comprise some of the largest deployed sensor networks, often containing thousands of sensors and actuators per building. This existing infrastructure is a potential goldmine, enabling new analyses and applications that can be implemented in software alone. These applications have the potential to drastically improve building energy use, occupant comfort, reliability and maintenance (e.g. [13, 17, 27]).

However, today there are a myriad of challenges in developing building applications: the systems deployed in buildings are a cornucopia of aged technologies speaking a wide array of protocols; the control systems that govern building operation are vertically-integrated, barely programmable, and not extensible; and the custom design of buildings and building control systems by a range of different parties results in a potpourri of naming schemes. Today, this requires applications to be custom written for each building by an engineer with deep knowledge of the particular architecture, connectivity, and control operation – effectively, the building equivalent of programming in assembly.

We present a new system architecture that addresses these problems and enables writing portable code by providing methods to explicitly and implicitly handle differences in building designs. A key insight of the design is the use of fuzzy, relativistic queries to allow authors to express their high-level intent in a way that is inherently portable, as well as supporting programmatic exploration of a building's specific components, allowing applications to explicitly handle differences amongst buildings. Thus, programmers can alternate between macro and micro level views of the building to express both general intentions and specific actions. We also present a hierarchical driver model that efficiently abstracts differences in

building control systems and provides a consistent API to application developers.

Finally, we explore applications that are enabled by this new framework, specifically focusing on a class of personal control applications that incorporate occupant feedback into building operation. We show that by providing a simple user interface that can actuate the building, we both save energy and make occupants more comfortable. Our prototype personalized lighting controls saved over 50% of lighting energy over a period of 12 weeks.

Chapter 2

Background

2.1 HVAC Mechanical Systems

Most modern commercial buildings contain extensive systems to ensure occupant health, safety, and comfort. This includes providing heating, ventilation, and air conditioning (HVAC), as well as, lighting, security, and fire safety services. These systems are often networked and can be centrally managed through operator interfaces. However, each system is often provided by a different vendor and has little interoperability or extensibility beyond the scope of the original design.

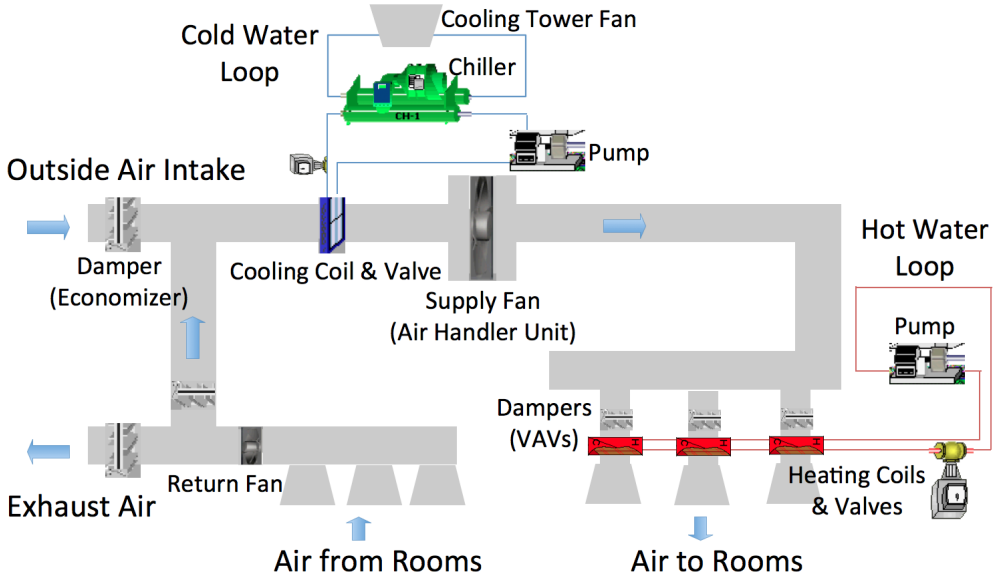


Figure 2.1: A typical HVAC system for a commercial building.

A typical commercial building HVAC system is shown in Figure 2.1. Fresh air is brought in from the outside to satisfy health requirements and is mixed with return air from the

building. This mixed air is cooled by passing over cold water coils and blown through ducts throughout the building. In each thermal zone, typically encompassing 1 to 3 private offices or 4 to 8 cubicles, the air passes through a variable air volume (VAV) box that dynamically controls airflow to meet temperature and ventilation requirements. From there, air enters the occupied space through diffusers. After circulating, air is sucked back through a return air plenum where a portion is exhausted and the remaining portion is recirculated.

A typical HVAC system for a large office buildings contains thousands of sensors and actuators measuring air and water temperatures, airflow, humidity, and duct pressures throughout the building. Actuators range from simple on/off relays to variable speed fans and pumps, water valves, and dampers. A modern HVAC system with digital controls contains embedded controllers, also known as programmable logic controllers (PLCs), throughout the building that are used to collect data from these sensors and to run the logic to actuate active components. These controllers are typically networked to allow communication between logic running on different PLCs and to allow a level of supervisory control, enabling an engineer or facility manager to view all sensor data and set configuration parameters (e.g. set points). The physical layer for building networks is most often RS-485 or Ethernet with a range of open and closed protocols running on top.

The logic running on embedded controllers is custom written for each building. Historically these devices were programmed with ladder logic [16]; today, a range of graphical and text-based programming languages are used. For example, Siemens Apogee systems use the Powers Process Control Language [29], a BASIC-like interpreted language, while Automated Logic WebCTRL systems use a graphical tool that consists of "microblocks," simple functions and logical blocks that can be wired together [3]. All of these systems lack meaningful high-level abstractions, easy communication with data sources outside the building, and an environment that allows rapid upgrades. Instead, today's building applications are hard-coded in low-level programming languages, requiring an engineer to visit the building for even minor changes.

Most HVAC vendors follow a stovepipe design with proprietary sensors, actuators, controllers, programming languages, and management software, making upgradability and interoperability a major challenge. Several standards have been established to address these problems. BACnet [1], standardized in 1995, is the most widely adopted controls standard. It establishes a common protocol for communicating with controllers, or in some cases with gateways that translate to internal proprietary protocols.

2.2 Existing Protocols

BACnet, or the Building Automation and Control Network protocol, is widely deployed and designed to allow control systems to interoperate. BACnet specifies physical, data link, network, and application layers. At the physical and link layers BACnet can run over RS-485, Ethernet or IP. At the application layer, BACnet exposes a set of devices, building controllers, that are discoverable through a broadcast message. Each device exposes a set of

objects (points) corresponding to either physical sensors and actuators or virtual inputs and outputs of the control logic. BACnet objects have a number of properties including name, description string, unit and present value. These properties can be read and in some cases written.

Unfortunately, there is no standardization of BACnet point names or values: a variable air volume box can be represented by tens of points with unrelated names and widely differing functionality from one vendor versus another. Crucial metadata about the location and functionality of each point is inconsistently encoded in names, description strings and units. BACnet also does not specify a standard way to reprogram building controllers; instead, writes to BACnet points may override the inputs or outputs of the programmed control logic.

SOAP, the Simple Object Access Protocol [32], is another commonly used interoperability protocol for control systems. SOAP is a generic RPC interface expressed in XML and typically run over HTTP. In building control systems, SOAP is often used to define basic discovery, read and write methods. For example, Automated Logic systems define `String getValue(String path)`, `String setValue(String path, String newValue)`, and `String[] getChildren(String path)` methods [2]. Hierarchical "path" strings are used to discover and address specific points. As with BACnet, there is no standardization of names or paths and no consistent way express metadata. A potpourri of less common protocols including WirelessHART [35] and Modbus [20] are also in use. These suffer from similar problems of inconsistent naming, lack of metadata and a lack of meaningful high-level abstractions.

Table 2.1 shows an example of BACnet point names form a large office building, Sutardja Dai Hall. Names are specified by a controls engineer during the initial configuration of the system. Most engineers follow a loose convention of acronyms and hierarchical naming. For example, in Sutardja Dai Hall periods, colons and underscores are used to separate parts of the name. All points are prefixed by SDH, an acronym for the building name. VAV points are named with a floor number and identifier – "S4-01" represents the first VAV on the 4th floor. Sensors and actuators pertaining to each VAV follow the name. The underlying functionality or control logic is not explicitly defined in BACnet and must be found in a separate manual [28]. In this control system, a single set point ("CTL STPT") is used combined with a heating or cooling mode ("HEAT.COOL"). When in heating mode the VAV will provide hot air while the room temperature ("ROOM TEMP") is below the set point, and will provide only minimum required ventilation air otherwise. In cooling mode, cold air is provided whenever the room temperature is above the set point. In the default configuration, this mode is switched automatically with a 1 degree hysteresis band.

Table 2.2 shows an analogous list of VAV sense and control points for Bancroft Library, a large office and library building with an Automated Logic control system and SOAP data access. Paths have a clear hierarchical structure, but the metadata encoded at each level is still inconsistent and follows only a loose convention. "doe_vav_c-2-14" indicates the building name, the Bancroft Library is adjacent to the Doe Library and was previously referred to as the Doe Library Annex, as well as the device type (VAV), floor (2) and identifier (14). Within each VAV, the control points use different acronyms for roughly the same set of

Point Name	Meaning
SDH.S4-01:ROOM TEMP	[Float] Floor 4, Zone 1, Room Temperature (F)
SDH.S4-01:CTL STPT	[Float] Temperature Setpoint (F)
SDH.S4-01:VLV POS	[Int] Measured Hot Water Valve Position (%)
SDH.S4-01:VLV CMND	[Int] Hot Water Valve Position Setpoint (%)
SDH.S4-01:HEAT.COOL	[Boolean] Heating or Cooling Mode
SDH.S4-01:AIR VOLUME	[Int] Airflow (CFM)
SDH.AH2A_RAT	[Float] Air Handler 2A, Return Air Temperature (F)
SDH.AH2A_MAT	[Float] Mixed Air Temperature (F)
SDH.AH2A_SAT	[Float] Supply Air Temperature (F)
SDH.AH2A_SAT.STP	[Float] Supply Air Temperature Setpoint (F)
SDH.AH2A_CCV	[Float] Measured Cooling Coil Valve Position (%)
SDH.AH2A.SF_VFD:POWER	[Float] Supply Fan VFD Measured Power (kW)

Table 2.1: Example BACnet point names and their meanings for Sutardja Dai Hall

sensors as in Sutardja Dai Hall. Finally, the control logic is subtly different. Instead of a set point and mode selector, the Automated Logic VAV uses two set points one for heating and one for cooling. When the room temperature falls below the heating set point, the heat is turned on and when the temperature raises above the cooling set point, cold air is supplied.

Path	Meaning
doe_vav_c-2-14/lstat/zone_temp	[Float] Floor 2, Zone 14, Room Temperature (F)
doe_vav_c-2-14/m024/input_value	[Float] Cooling Setpoint (F)
doe_vav_c-2-14/m023/input_value	[Float] Heating Setpoint (F)
doe_vav_c-2-14/rh_vlv_c	[Int] Hot Water Valve Position Setpoint (%)
doe_vav_c-2-14/air_flow/flow_input	[Int] Airflow (CFM)

Table 2.2: Example SOAP paths and their meanings for Bancroft Library

Overall, existing building control interfaces provide discovery and basic read/write access to a large number of points – sensors, actuators and virtual inputs/outputs. However, today's interfaces suffer from a lack of metadata, inconsistent naming and a lack of common representation of similar functionality. To enable rapid development of portable applications, we must raise the level of abstraction, providing a uniform interface to heterogenous control implementations.

Chapter 3

Towards Programmable Buildings

3.1 Architecture for Portability

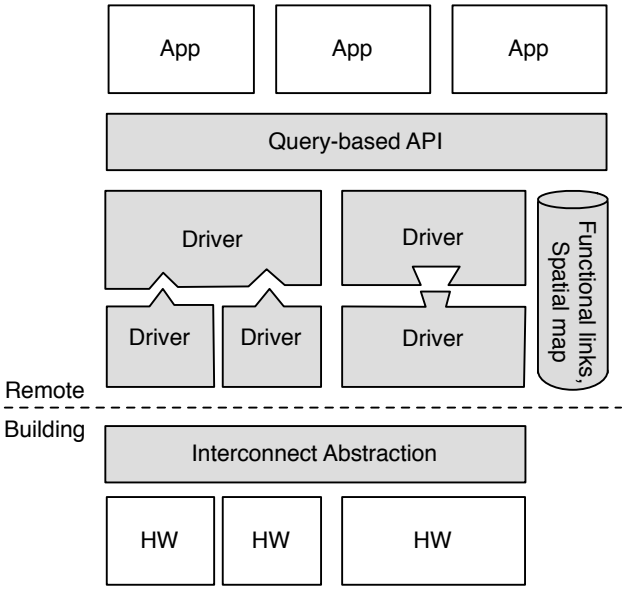


Figure 3.1: BAS layered architecture.

The Building Application Stack (BAS) is an API and runtime that allow applications to run on a wide range of buildings. At the same time, BAS aims to minimize the effort required to set up the system on a new building by factoring out common components and partially automating metadata collection.

The BAS architecture is shown in Figure 3.1. At the lowest level, BAS interfaces with the building hardware through existing control protocols. The BAS interconnect abstraction

layer acts as a hardware interface, exposing all points from different control networks with different protocols in a uniform way by using the sMAP architecture [7].

While having access to building sensor and actuator points is sufficient for implementing applications, point names and functionality are very building-specific, thus programming at this level leads to non-portable code. The BAS driver layer abstracts groups of points into functional objects with standardized methods. Each driver type must expose at least a minimum predefined interface. For example, fan drivers expose "get_speed" and "set_speed" methods. These methods apply to many different fan designs, e.g. variable speed, 3-speed, or on/off fans. Drivers can be built up hierarchically so that common functionality is implemented once.

The objects exposed by drivers correspond to physical components within the building, e.g. chillers, light banks, electrical circuits, etc. Drivers abstract the inner workings of these devices, but since buildings are all constructed differently the functional and spatial relationships between these objects are crucial. Being able to answer *which* air handler serves a particular room or *which* circuit powers a light bank is key to expressing control actions in a general way. BAS captures functional relationships in a directed graph of "supplies" relations, effectively capturing airflow loops, water loops and electrical trees. In addition, BAS incorporates spatial tags stored in a GIS database, allowing us to answer queries such as, "select the lights for all rooms near windows" or "select the air handler that supplies room 123."

At the highest level, BAS provides a query-based API for writing applications. The API consists of a selector query for choosing a set of driver objects corresponding to building components based on object name, type, attributes, functional relationships and spatial relationships. Each object type has a predefined minimal set of methods, i.e. sensor readings and actuation capabilities. The application can make use of these guaranteed methods or discover all available methods. Similarly, objects can be selected with a precise query, e.g. "the thermostat for room 123," or by exploring the graph of functional relationships, e.g. "all temperature sensors upstream of room 123." Applications access the BAS API through a web service and can be written in any language.

3.2 Interconnect Abstraction

The interconnect layer provides a RESTful interface that routes sequences of read and write requests to the underlying building control protocols. It addresses three main challenges in processing requests from the driver and application levels: interfacing with different building control protocols, handling network throughput, and maintaining a consistent building state. By using sMAP [8] to provide this interface, BAS is able to implicitly handle distributed setup as well as data archiving. Based on extensive deployment experience [6], sMAP provides robust support for representing and publishing generalized time series data: any timestamped sequence of scalar or vector values, as well as actuators.

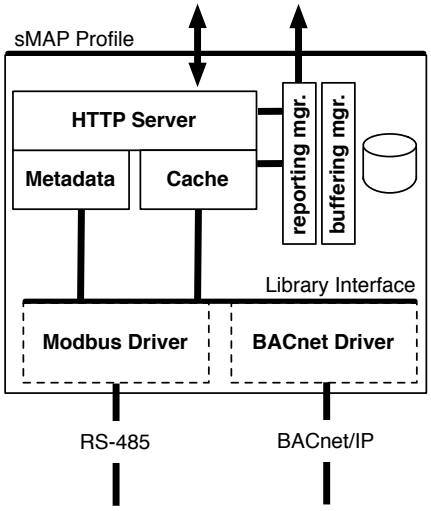
A building may contain any one of a number of building control protocols – such as BACnet [1], OPC [23], LONtalk [11], ALC SOAP [2], and ModBus [20] – each of which may define or provide their own communication protocol, software library, etc. for the purpose of reading and writing to a set of points corresponding to aspects of the building. It is the job of the interconnect layer to handle the implementation details to sufficiently abstract the reading and writing of points for the driver and application levels. Figure 3.2b shows the design of the sMAP library used in the interconnect layer. When interfacing with BACnet, the interconnect layer helpfully removes the need to know the unusual or specific numeric constants required to correctly formulate a BACnet packet; it exposes the command resource tree shown in Figure 3.2a.

The devices inside buildings often communicate over a specialized physical layer that may not be immediately compatible with TCP/IP. A building's control protocol will usually provide some mechanism for communicating with these devices, but the implied intricacies of such a mechanism are often too complex to be accounted for at a higher level in BAS. By handling the construction of packets and managing network throughput at the interconnect layer, the driver layer does not require extensive knowledge of various network protocols.

Furthermore, the interconnect layer handles all end-to-end reliability on behalf of the drivers. The utility of the higher abstraction layers is made with the guarantee of a consistent building state, so despite the multi-tiered indirection between an application and its points of actuation, the end-view of a building will be correct.

```
/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
```

(a) Resource tree exported by sMAP.



(b) The sMAP library exposes dataand metadata over HTTP.

3.3 Fuzzy Queries

`#FLOOR`	All floor areas
`#LIGHT > $Floor 1`	All lights on the first floor
`#AREA < $Lightbank 1`	Zones served by light bank 1
`#AH > $Room 123`	All air handlers that serve Room 123
`#AREA < #AH > $Room 234`	Areas served by the same air handler as Room 234
`#ELECMETER > #AREA < $Air Handler 1`	Power meters for zones served by AH1

Table 3.1: Example BAS queries showing fuzzy and relative lookups. $A > B$ means A that feed into or supplies B; $X < Y$ means the X that is fed by or supplied by Y. Operators are right associative.

The BAS query interface is designed to allow application authors to select objects based on type, attributes and functional or spatial relationships. This allows authors to *describe* the particular sensor or actuator that the application requires rather than hardcoding a name or tag that may not apply in differently designed buildings.

Queries are expressed in terms of names indicated with a $ prefix, tags starting with # 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. Query strings are evaluated right to left (right associativity) with the left-most operand indicating the object to return. Table 3.1 lists example queries.

Internally, BAS stores a directed graph of objects to answer queries. Figure 3.2 shows a partial rendering of functional and spatial graph for one of our two test buildings. 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 air, water, electricity, etc. Edges are purposefully unnamed to allow for buildings with many different designs to be represented without introducing new names. Tags are automatically applied based on the type of driver instantiated and the interface it implements. Tags describe the object type and functionality. They are intentionally low-level and are not meant to uniquely identify objects. The intent is to use relationships to select objects in a general way. For example, a supply air temperature sensor can be selected as `#TEMP < #COOL < $Air Handler 1`, that is, the temperature sensor down stream of the cooling element in Air Handler 1. This allows the application to run with both a standard temperature sensor built into the air handler as well as a less common building design with a temperature sensor installed in the duct work or at a VAV input. A partial list of tags is show in Table 3.2.

In addition to functional relationships, BAS also stores spatial data. Spatial areas are defined as polygons on floor maps of the building and stored in a GIS database [24]. Areas are defined for the regions served by a given duct or VAV box and for lighting zones. These

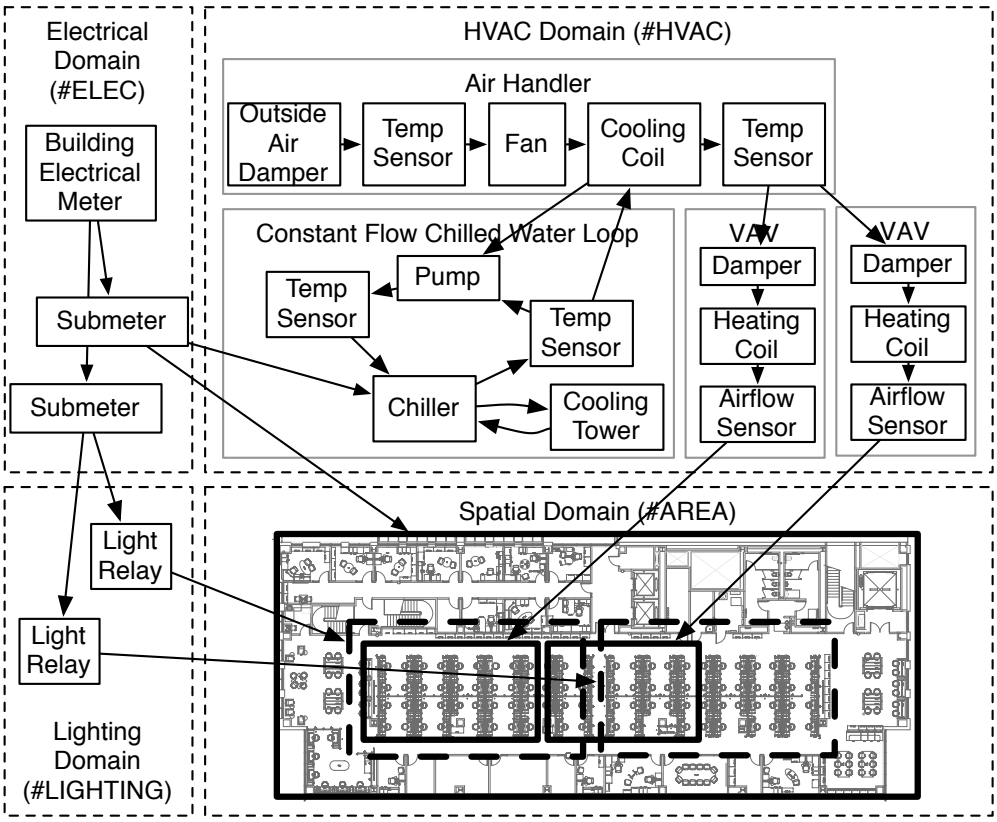


Figure 3.2: Partial functional and spatial representation of our test building. Directed edges indicate a supplies or feeds into relationship. BAS queries are executed by searching the graph.

functional areas do not always align with logical spaces such as hallways, offices or the cubicles assigned to a research group. BAS allows logical areas to be defined for use in the query system. By default the query system treats all intersecting areas as logically linked. If a region named `$BAS Cubicle` was defined, then the query `#LIGHT > $BAS Cubicle` would return all light zones that overlap with the BAS cubicle area. Spatial areas can be easily added or modified by updating the underlaying GIS database.

Queries are evaluated with a graph traversal algorithm. Starting at the right-most operand, all objects matching the tags or object names are added to the search list. Next, we search all incoming edges or all outgoing edges based on the operator ($<$ or $>$) for objects that match the left hand operand. Each search is preformed recursively in a breadth-first way. Visited objects are tracked to prevent infinite loops. Searches are limited to objects in the domains of the two operands, to search across a third domain it must be specified explicitly. For example, to find lights for zones served by Air Handler 1 the query is `#LIGHT > #AREA < $Air Handler 1`; `#AREA` must be specified explicitly to search through

the spatial domain. Intersecting spatial areas are treated as bidirectionally linked for the purposes of query execution.

#ELEC	All objects in electrical domain
#HVAC	All objects in the HVAC domain
#AREA	All areas in the spatial domain
#LIGHT	All objects in lighting domain
#FLOOR	All floors
#SEN	All sensors
#ACT	All actuators
#RELAY	Relays
#DMP	Dampers
#VLV	Valves
#ELECMETER	Electrical meters
#AH	Air handlers

Table 3.2: Partial list of object tags. Tags describe the types of primitive objects. Queries use tags and functional or spatial relationships to find objects.

3.4 Drivers

In order to provide a fuzzily-searchable query interface for device discovery, measurement and actuation, BAS must be able to provide a sufficiently concrete method of defining a building that does not drastically limit the portability or generalizability of any of its components. BAS exploits the high-level functional similarities between buildings to define a set of high and low-level driver interfaces that encompass the range of possible internal components in a given building. The goal of this structure is to facilitate the adherence of a building to a common, exposed API on top of which portable applications can be built.

BAS driver interfaces define common APIs for high-level objects whose functionality depends on and consists of a certain set of lower-level components, e.g. a certain type of air handler may contain an outside air damper, a set of temperature sensors, a cooling coil valve, etc, but it retains a level of functional congruity with different types of air handlers that may contain different components. The low-level BAS driver interfaces define these fundamental components in terms of their basic type: sensors, dampers, valves, fans and so forth.

Each of these interfaces is then implemented by one or more classes whose code handles the combination of device-specific functionality to provide the logic for the interface. The ability of the drivers to support multiple classes for a given interface means that BAS remains extensible enough to integrate any custom interface for any esoterically-designed building component – provided that the logic exposes the expected API – as well as promoting

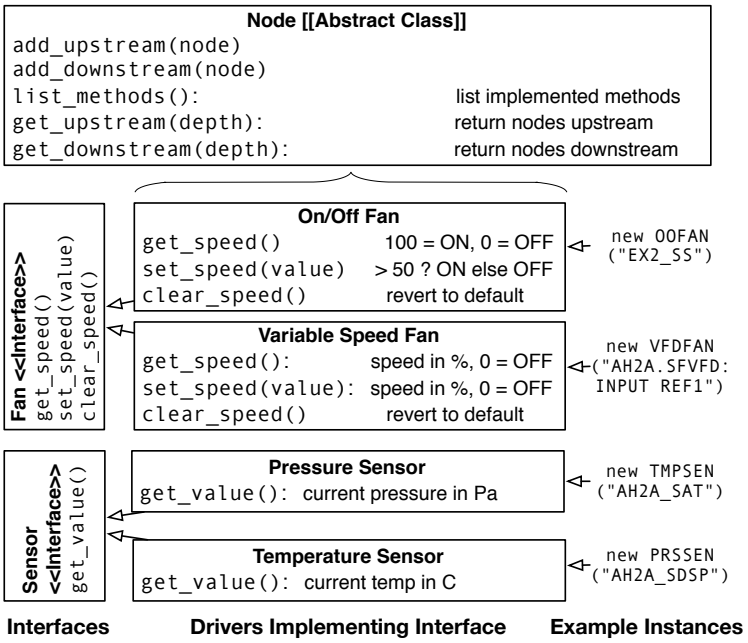


Figure 3.3: Interfaces for lower-level devices specify methods for general equipment types to be implemented in equipment-specific classes. Classes are instantiated with references to the actual hardware.

reusability and portability. As seen in Figure 3.3, each of the driver implementation classes contains custom logic that adapts the specifics of a type of equipment to the expected interface.

For the high-level drivers, the interface specifies the expected components in terms of descriptive tags; any instance of that driver must provide instances of the requisite low-level driver classes to populate that instance's functionality. These descriptive tags distinguish between the functional roles of the objects and classify them in a manner that enables the query interface to discover and traverse them in terms of their functional relationships to other objects. Tags are predefined and documented by BAS to ensure portability. Objects are described by lists of these tags, joined by underscores (see example below). Basic functionality for these objects is provided by the Node abstract class which endows objects with an awareness of the network graph, granting them the ability to establish themselves in relation to objects around them.

Once the objects for a building are fully populated and instantiated, BAS creates the network graph for the purpose of the query interface. The high-level objects use their sets of expected lower-level objects in order to pre-construct internal network graphs, which are incorporated into the comprehensive graph by linking lower-level objects to each other by calling `object.add_upstream(target_object)` and `object.add_downstream(target_object)`.

```
#instantiate air handler object
ahu1 = ahu('Air Handler 1', {
    'OUT_AIR_DMP': Damper('SDH.PXCM-01 SDH.AH1A_OAD'),
    'MIX_AIR_TMP_SEN': TmpSen('SDH.PXCM-08 SDH.AH1A_MAT'),
    'COOL_VLV': Valve('SDH.PXCM-01 SDH.AH1A_CCV'),
    'SUP_FAN': VfdFan('SDH.PXCM-01 SDH.AH1A.SF_VFD'),
    'SUP_AIR_TMP_SEN': TmpSen('SDH.PXCM-01 SDH.AH1A_SAT')
    } )
#link air handler object into the network graph
ahu1['COOL_VLV'].add_upstream(cold_water_loop['PUMP'])
ahu1['SUP_FAN'].add_downstream(vav1['DMP'])
ahu1['SUP_FAN'].add_downstream(vav2['DMP'])
...
```

Figure 3.4: Instantiating a simplified air handler

Example: Simplified Air Handler

In this simplified case, the type of air handlers found in the BACnet-based building contain an outside air damper, a mixed air temperature sensor, a cooling coil valve, a supply fan and a supply air temperature sensor. In order to create an instance of this type of air handler, it is first necessary to create instances of these expected components. Once these are provided in the instantiation of the air handler, the particular BAS air handler class knows how to incorporate the API for each component into the expected logic for the general air handler API.

BAS defines low-level driver interfaces for dampers, sensors, valves and fans, and the BAS implementation for the building contains interface-compliant classes that provide the necessary BACnet-specific logic for reading from and writing to the necessary points for each of these drivers. The exact point names for each object are provided upon instantiation of that object.

3.5 Autopopulation

In order to port applications between buildings, BAS must be correctly configured for each building, requiring some amount of work to be done to catalog or discover exactly what devices are in the building, how they are logically and functionally connected, and how these logical pieces affect the spatial aspects of the building. Automating this process is an area of ongoing work. Our initial approach is to use existing interfaces for object discovery combined with drivers we wrote for different building components and an image recognition technique for importing functional and spatial relationships from existing documents.

Most building control protocols provide some method for device discovery. This usually takes the form of providing a list of all devices that can be read from and/or written to over the internal network. Unfortunately, for some protocols such as BACnet, device discovery takes the form of a `whois` packet broadcast over the local subnet, meaning that in the absence

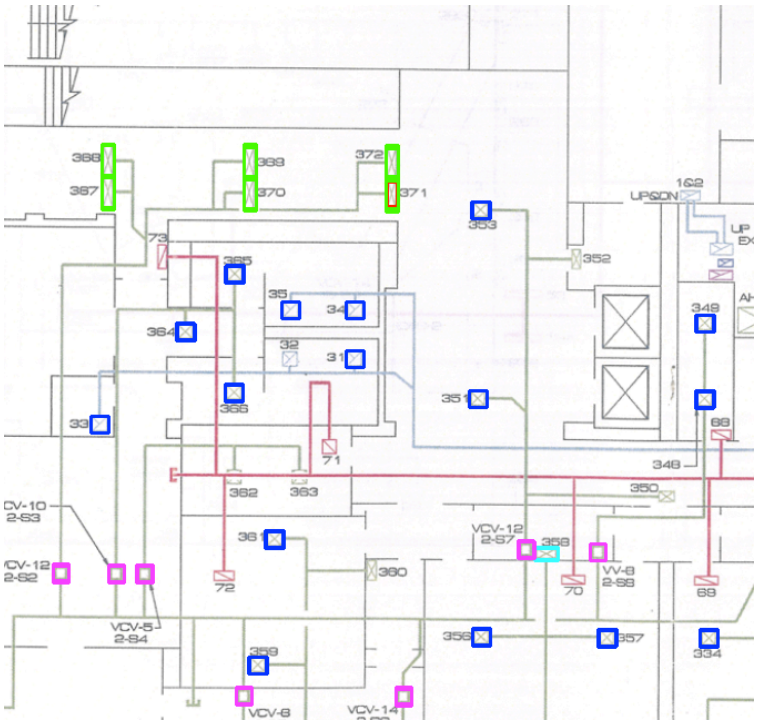


Figure 3.5: Example of parsing an image to identify air handlers, VAV boxes, and dampers

of having a machine on the building subnet, the discovery method is unable to return any results. It is possible, however, to construct packets that take into account the internal structure of the building network, and thus conduct a device discovery query remotely. We scan a given IP for valid gateway devices and then search for internal sub-networks by examining the error codes returned by BACnet.

Once we have a list of devices on the internal building network, we can use the building's naming schema to identify objects by type, name, and location. The reality is that most building control protocols do not enforce the inclusion of much of this metadata, leading to incomplete and sometimes inconsistent catalogs of building devices. Despite the wide variability in the quality of existing building metadata, BAS can automatically construct at least a partial representation of the building, drastically reducing the overhead for initializing a building, depending on the structure of the internal network and the amount of metadata provided.

In some cases it is possible to extract data from the existing building management system to construct the functional and spatial relationships. For example, the ALC WebCTRL [3] automation system provides a SOAP interface that contains an internal geographically organized tree of all the building's devices, helpfully combining the processes of discovery and geo-tagging.

Other representations of the building such as IFCs [15] or EnergyPlus models [12] could

be used to automatically load building metadata. However, these are not often available. As a fallback method we use computer vision techniques on architectural diagrams to recognize various types of objects on a floor plan, noting their location and relationship.

The example in Figure 3.5 uses only a few lines of the OpenCV Python library to extract VAV objects from a ductwork diagram by nature of their shape using the built-in template matching functionality. Combining this with OCR, it is possible to extract the geospatial coordinates, names and functional relationship of VAVs.

3.6 Applications and Evaluation

The power of BAS lies in its ability to be easily adapted from one building to another, bridging the common functionality between different hardware implementations, communication protocols, and building control systems. Here, we will examine the deployment of two applications on two buildings with different control systems, implemented with and without BAS.

The first building is approximately 100k sq. feet and has 7 floors. It contains large open cubicle areas for graduate students and private offices for faculty and administrators. It also contains a cafe, auditorium and several classrooms. It runs a Siemens control system with over 8000 available sensor and actuator points exposed over BACnet. The second building is approximately 75k sq. feet in size and has 6 floors with office areas and a library area. The control system is ALC, which supports both low-level BACnet access and a higher-level SOAP interface, exposing over 5000 points.

Occupant HVAC Control App

The occupant HVAC control application allows occupants to temporarily blow hot or cold air into a room. The logic is simple. For a given room, the corresponding damper and, in the case of a request for heating, heating element are opened for a limited time. Then, the damper and heating defaults are restored.

Figure 3.6 shows the implementation in Python using a BACnet library. BACnet commands require cumbersome low-level arguments: device name, object instance number, property and data type; this information is not necessarily easy to access. The application must contain an explicit building-specific mapping of which dampers and heating valves control each room, along with which internal BACnet object and value type those objects are. BACnet points have cryptic names and do not contain any spatial awareness of the building itself. This code could be additionally augmented with consistency checking that verifies that the writes were received correctly by the appropriate devices, as BACnet uses UDP and thus is inherently unreliable.

If this application were to be deployed to any other building, the devices and objects would have to be rewritten to point to the new building's BACnet IP router. The instance

```
#Using direct BACnet
import bacnet
import time
dampers = {
            'Room 444': ('SDH.PXCM-11','SDH.S4-03:DMPR POS', 'SDH.S4-03:VLV POS'),
            'Room 446': ('SDH.PXCM-11','SDH.S4-05:DMPR POS', 'SDH.S4-05:VLV POS'),
            'Room 448': ('SDH.PXCM-11','SDH.S4-09:DMPR POS', 'SDH.S4-09:VLV POS'),
            ...,
            }
def cool_room(room_number):
  damper = dampers[room_number]
  device = bacnet.find(name=damper[0]) #bacnet.find('SDH.PXCM-11')
  object = bacnet.find(name=damper[1]) #bacnet.find('SDH.S4-05:DMPR POS')
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=object.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=100, value_type=bacnet.BACNET_APPLICATION_TAG_REAL)
  time.sleep(1000)
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=object.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=1, value_type=bacnet.BACNET_APPLICATION_TAG_NULL)

def warm_room(room_number):
  damper = dampers[room_number]
  device = bacnet.find(name=damper[0]) #bacnet.find('SDH.PXCM-11')
  object = bacnet.find(name=damper[1]) #bacnet.find('SDH.S4-05:DMPR POS')
  heating = bacnet.find(name=damper[2]) #bacnet.find('SDH.S4-05:VLV POS')
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=heating.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=100, value_type=bacnet.BACNET_APPLICATION_TAG_REAL)
  time.sleep(60)
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=object.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=100, value_type=bacnet.BACNET_APPLICATION_TAG_REAL)
  time.sleep(1000)
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=object.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=1, value_type=bacnet.BACNET_APPLICATION_TAG_NULL)
  bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
          instance_number=heating.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
          value=1, value_type=bacnet.BACNET_APPLICATION_TAG_NULL)
```

Figure 3.6: BACnet-specific occupant controlled HVAC application code.

numbers as well as the property, value and object types would have to be double checked to ensure correctness, and the point names would have to be changed.

Figure 3.7 shows the functionally equivalent application written using BAS. The increased readability is immediately obvious. BAS encodes spatial relationships, so finding VAVs corresponding to a space is trivial. BAS drivers expose convenient methods for setting airflow and heating in a standard way that applies to all VAVs. The application makes no reference to the specific communication protocol nor to the specific point names. This app could be ported to a completely different building and run without further configuration of the application itself, provided that BAS is correctly installed on the given building. Figure 3.8 shows this application running on our building with Siemens controls. The app warms an enclosed office area in response to an occupant request.

```
#Using BAS
import appstack
import time
api = appstack.Appstack()

def cool_room(room_number):
  vav = api('#VAV > %s ' % room_number)
  vav.set_airflow(100)
  time.sleep(1000)
  vav.clear_airflow()

def warm_room(room_number):
  vav = api('#VAV > %s ' % room_number)
  if 'set_heat' in vav.list_methods():
    vav.set_heat(100)
    time.sleep(60)
    vav.set_airflow(100)
    time.sleep(1000)
    vav.clear_heat()
    vav.clear_airflow()
```

Figure 3.7: BAS implementation of occupant HVAC controls.

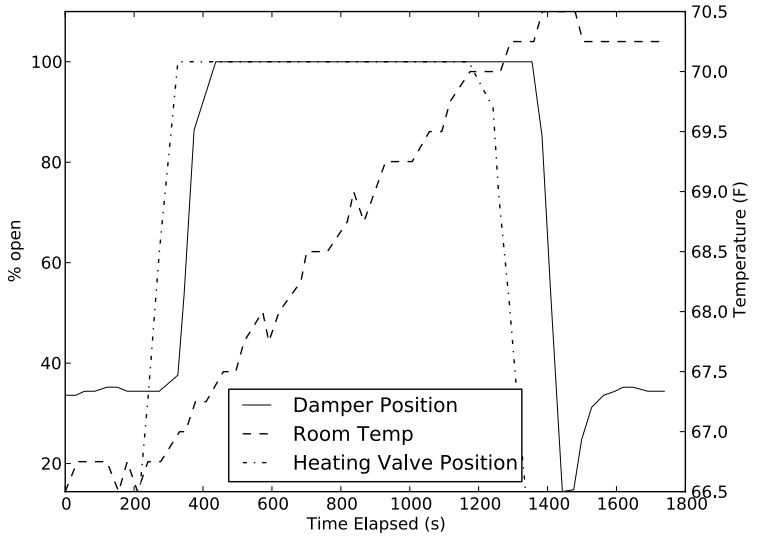


Figure 3.8: Trace of the occupant HVAC control app running on one of our buildings. The heating coil and airflow increase to maximum, rapidly warming the room.

Ventilation Optimization App

The ventilation optimization application is based on the observation that many buildings are over-ventilated; California Title 24 [5] requires 15 CFM of *fresh air* per person in the building, but the fraction of fresh air taken into the building varies from 30% to 100% of the supply air depending on the economizer (outside air damper) position. As a result, airflow rates are often set assuming minimum fresh air intake. The ventilation optimization app

```
#Using direct BACnet
import bacnet
#damper setpoints for each outside air damper
oad_to_dmp_stpts = {
  'SDH.PXCM-01 SDH.AH1A_OAD': [
    'SDH.PXCM-04 SDH.S1-20:CTL FLOW MIN',
    'SDH.PXCM-04 SDH.S1-19:CTL FLOW MIN',
    ...],
  'SDH.PXCM-01 SDH.AH1B_OAD': [
    'SDH.PXCM-11 SDH S2-04:CTL FLOW MIN',
    ...],
  'SDH.PXCM-08 SDH.AH2A_OAD':[
    'SDH.PXCM-11 SDH.S4-03:CTL FLOW MIN',
    ...]
  }
for oad in oad_to_dmp_stpts.keys():
  device = bacnet.find(name=oad)
  oad_airflow = bacnet.read_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
    instance_number=device.instance_number, property=bacnet.PROP_PRESENT_VALUE)
  for dmp in oad_to_dmp_stpts[oad]:
    damper = bacnet.find(name=dmp)
    old_setpoint = bacnet.read_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT, \
      instance_number=damper.instance_number, property=bacnet.PROP_PRESENT_VALUE)
    new_setpoint = old_setpoint / oad_airflow
    bacnet.write_prop(device, object_type=bacnet.OBJECT_ANALOG_OUTPUT,\
      instance_number=damper.instance_number, property=bacnet.PROP_PRESENT_VALUE, \
      value=new_setpoint, value_type=bacnet.BACNET_APPLICATION_TAG_REAL)
```

Figure 3.9: Ventilation optimization app implemented on our first building using BACnet.

adjusts the minimum airflow rates for all VAV boxes in the building based on the fraction of fresh air their corresponding air handler brings in.

It is immediately evident in the writing of this application that some knowledge of the layout of the HVAC system is required in order to adjust each VAV by the correct amount. Figures 3.9 and 3.10 show this application implemented on our Siemens and ALC buildings respectively. Note the hardcoding of associations between VAVs and air handlers, the cryptic point names and the control protocol specific actuation code. Porting the application from one building to another required a complete rewrite of the read/write logic, not just the point names.

The same application implemented using BAS is shown in 3.11. Using relative queries simplifies the code by dynamically finding the corresponding economizers, VAV boxes and dampers; thus, porting the application from our first building to the second required no change in the actual code.

Porting BAS

In configuring BAS to run on two buildings as different as the BACnet-based and ALC-based buildings mentioned above, there is a surprisingly small amount of work to be done considering the differences between them. The most accessible programmatic interface to

```
#Using ALC SOAP
import suds
url_to_wsdl = 'http://......'
client = suds.client.Client(url)
air_handlers = {
  '#doe_basement/#doe_base_equipment/#doe_ah-c': [
    '#doe_vav_c-2-13/air_flow/flow_tab',
    '#doe_vav_c-2-14/air_flow/flow_tab',
    ...]
  '#doe_penthouse/#doe_ah-b1_ah-b2_rf-1': [
    '#doe_vav_b-5-01/air_flow_b1/flow_tab',
    ...]
  }
for ahu in air_handlers.keys():
  oad_airflow = client.getValue(ahu+'/oa_damper')
  for vav in air_handlers[ahu]:
    old_damper_airflow = client.getValue(vav+'/m269')
    new_damper_airflow = old_damper_airflow / oad_airflow
    client.setValue(vav+'/m269', new_damper_airflow)
```

Figure 3.10: Ventilation optimization app implemented on our second building using SOAP.

the first is through BACnet; the particular implementation allows for direct access to many individual components within the building, but lacks much of the location-aware metadata that is useful to BAS. Conversely, the ALC-based building provides a SOAP (Simple Object Access Protocol) interface over BACnet, which allows BAS to glean a considerable degree of location-aware metadata, but does not allow as fine-grained control over individual devices.

```
#Using BAS
import appstack
api = appstack.Appstack()
ah_dampers = api('#OUT_AIR_DMP > #AH')
for dmp in ah_dampers:
  for vav in api('#VAV < £%s' % dmp.name):
    vav.set_min_airflow(vav.min_fresh_air() / dmp.get_percent_open())
```

Figure 3.11: Ventilation optimization app implemented in BAS and executable on both buildings.

In practice, configuring BAS to run the application in Figure 3.11 above required only some adjustment of the damper and heating valve drivers to communicate with the specific control protocol of the building in question, which was greatly simplified by the abstractions already provided by the interconnect layer (about 5 LOC per driver). While it could be argued that the BACnet specific code in Figure 3.9 could be configured to run on a different (yet still BACnet controlled) building with changes on the order of 5 LOC, this configuration would have to be done on an application-by-application basis, whereas BAS only requires configuration once per building. The hierarchical nature of BAS drivers means that only low-level drivers (e.g. damper, valve, fan, etc.) need to be ported, while high-level drivers (e.g. air handler) generally remain unchanged.

3.7 Related Work

Several approaches have been proposed for organizing building metadata. Tree structures are most commonly used today [3, 9]. In this approach, related sensor and actuator points are grouped hierarchically. For example, in one of our buildings HVAC equipment is grouped by location and function: `/basement/hot_water_plant/boiler/steam_flow` and power meters are grouped by panel and breaker: `/main_breaker/panel_41/floor4_lighting/real_power`. Hierarchies provide a single, "precomputed" way to explore the data. This works well for answering some queries, e.g. "what loads are on panel 41," but cannot handle more complex queries, e.g. "which circuits power room 123's outlets and lights."

The next approach is to use predefined classes or entity-relationship models for common building components. Industry Foundation Classes [15] 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 [4]. BAS models the active sensor and actuator components, while IFC focuses on the design specification without linking this to the running controls. BAS uses a similar connectivity model, but eliminates the need for rigid predefined classes by using tags and a driver model that only requires implementing basic read/actuate methods for each driver type.

Project Haystack [25] uses a list of tags and rules about their use to describe building components. The tagging approach is very flexible and overcomes the rigid structure of fixed object classes or hierarchies. However, tagging schemes like Haystack cannot encode the full range of functional and spatial relationships. Queries are limited to the relationships that are manually tagged. In BAS, relationships are queried by traversing a graph structure.

All of these efforts focus on describing building data, while BAS crucially provides methods for actively controlling the building and raises the level of abstraction to allow control to be implemented portably. We share the same vision as [19] of enabling building applications. [19] focuses on integrating existing metadata from multiple sources and devising a common data representation for use by applications. BAS is complementary and is focused on how applications can conveniently make use of available building controls portably and at a higher level of abstraction (driver API), how to make applications automatically adjust to different building designs (fuzzy queries and exploration) and how to execute applications on a new building (interconnect and runtime platform).

Chapter 4

Gaining Personal Control

4.1 Putting People in the Loop

A class of powerful building applications enabled by the BAS architecture give people increased control over the built environment. Buildings are fundamentally designed for people and yet today, occupant feedback rarely factors into building operation. The BAS architecture makes it easy to write portable user-facing building applications and to incorporate external data into building controls.

We present a personalized lighting application built upon a traditional commercial building automated lighting control system. It embodies three important design principles: individual empowerment with localized human-centered resolution, token effort for consumption, and return to a low-power state. These principles are actualized in the design and implementation of a simple "shared virtual light switch" application for smartphones and browsers that provides lighting control on each individual lighting zone in a large open-office academic research environment. Occupants began using this facility on their own (with enthusiasm) before the second slide of the presentation intended to introduce it. They continued to use it twelve weeks later and have cut their energy consumption for lighting in half by easily exploiting the part time, part space, partial power nature of individual needs.

We all understand how to turn on and off a light by "flipping the switch" or "pushing the button" and have been well trained to avoid waste through countless repetitions of "turn off the light when you leave the room." It is a natural interface, typically associated with the act of entering and leaving a modest sized space. And yet, in commercial buildings we typically lose this simple form of personal control; instead, lighting is controlled automatically through schedules and overrides, sometimes augmented with motion detectors or daylight sensors. Often, a single lighting zone is shared by several occupants and is situated within a large space without a natural association between the particular region over which we want to exert control and the means for doing so. And, as a result lights are often on when they are not needed.

Baseline Usage Model

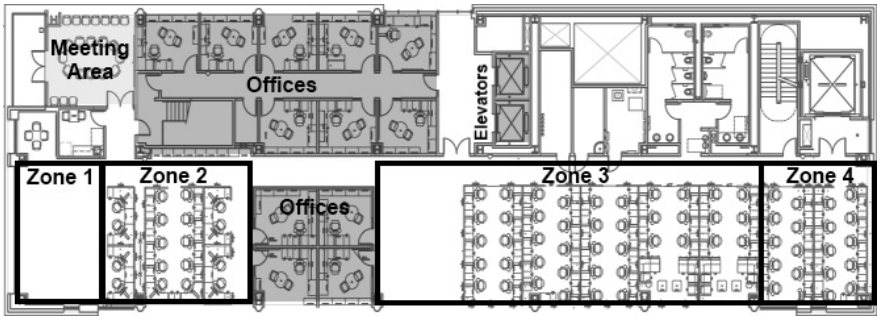


Figure 4.1: Floormap of the 7th floor showing a large open collaboratory and a smaller area of individual offices. The collaboratory is broken into 4 independently controllable lighting zones. Side zones 1 and 4 have large windows facing West and East respectively. Zone 1 is a common space used for meetings; all other zones consist of individual desks.

This situation was addressed explicitly in the design of our target building – Sutardja Dai Hall, the headquarters of the Center for Information Technology in the Interest of Society, a large, seven floor building with large open "collaboratories," in addition to traditional offices, classroom, labs, and an experimental semiconductor manufacturing facility. Figure 4.1 shows a typical floor plan (floor 7) with its large collaboratory spanning the east-west extent of the building, elevator access in the center of the north side, and individual office space along the north side. It was a design goal to have a green building consistent with the mission of the center. Collaboratories were divided into multiple sub-zones, typically five per floor. A typical working zone has three lighting power settings, low, medium, and high. A BACnet-based WattStopper automated lighting control system [33] was deployed throughout the building to control these zones and several switches (a pair for each zone and a pair for all zones on the floor) were placed on the north wall of the collaboratory. Through the automated lighting control system, the facility manager programmed the lights to a schedule in which all zones of a floor are on high, the maximum brightness setting, during weekdays from 10 am till 7 pm. The lights were off during nights and weekends, unless someone pushed one of the wall's override switches, which causes the whole floor or the associated zone to be on for a period of three hours, depending on which switch was pressed. It is also possible to specify the lighting level by pushing an appropriate combination of (unlabeled) switch pairs, but this is rarely used.

This program defines our baseline usage model, in place prior to this study for a year and a half since initial building occupation. Generally, occupants disliked this facility-controlled lighting schedule. The lights were on all day long, regardless of occupancy, but when the lights shut off after working hours, someone would have to run over to the north wall and push the physical override button every hour. Generally, that turned the entire collaboratory on high, since few could figure out the per-zone switches or the proper incantation for

intermediate lighting levels. Various efforts were made to incorporate motion sensors, as well as daylighting sensors in the end zones, but these were disabled after occupants grew frustrated with improper control actions, such as flapping their arms to keep the lights on while trying to concentrate.

A Personalized Automated Lighting Control Alternative

We learned of this less-than-optimal situation while engaged in a variety of deep energy efficiency efforts with this building, which included development of a rich infrastructure for energy usage monitoring and simple web-services based control (described below). The lighting control frustration provided an opportunity to test the ease of application development on our infrastructure, while benefitting the people in the building and hopefully saving some energy. Since prior studies have shown occupants are happier with direct personal controls [14, 22], we gave them a virtual light switch, accessible by their smartphone or browser. As illustrated in Figure 4.2, one click to locate the floor in the building and one to identify the zone, provides the "action page." There, a click on the "Reset Timer" button turns on the lights for a period of time. This page can be bookmarked or placed on the wall paper for easy access. In addition, QR-codes in the zone contain the URL and so take the smartphone directly to the action page with a camera snap.

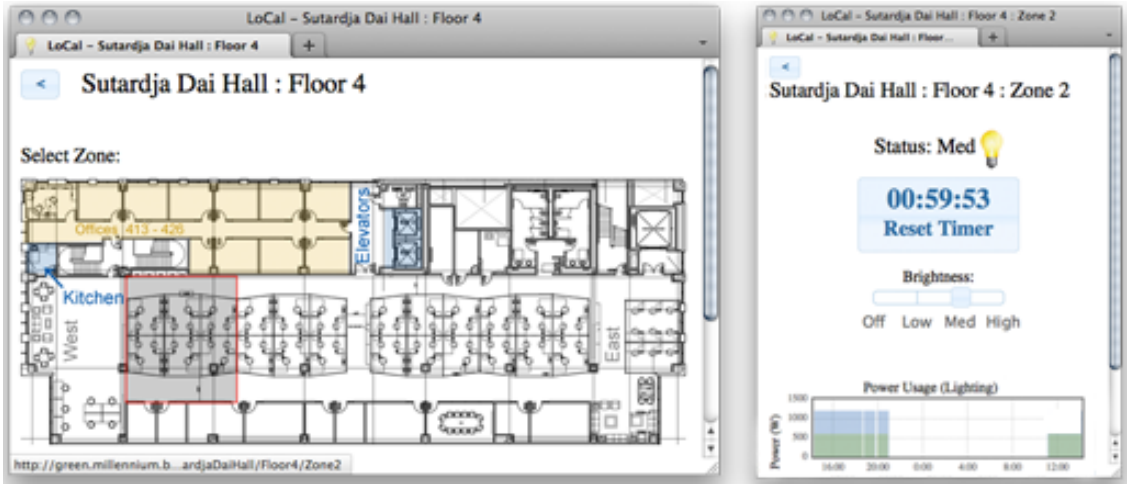


Figure 4.2: Screenshot of the personalized lighting control interface. Users select a zone by clicking on the floormap and then press the reset timer button to activate the lights and select a desired brightness level.

Other prior studies indicate that people react poorly to extremely bright or excessively dim environments [21]. Embracing the principle of personal empowerment in a shared setting as an opportunity for energy savings, the action page also makes it easy to specify the light level or turn it off. The action is experienced by all in the lighting zone and whoever is viewing

the action page. Although these stakeholders may have different preferences, we do not automate the resolution in contrast to more complex personal lighting control systems [34, 26]. The occupants are all present in a small physical area and can resolve lighting preference discrepancies through human-to-human interaction. The webpage reflects only the resultant action. It also provides a simple history of power usage for lighting in the zone and a sense of the cost associated with the available options going forward.

The user does have to expend a token amount of personal energy (clicking the reset button) to keep the lights on, thereby continuing to consume electric power. We introduced the personalized lighting controls in an informal meeting (on a Friday afternoon) with occupants of the fourth floor and discussed the trade-offs of a bit more irritation for more energy savings, and vice versa. The collective compromise was to use a three hour timer during working hours and a one hour timer all other times. This achieves the principle of quiescence at low power at a relatively fine grain, because each zone, if left alone for a couple of hours will go off.

At that introduction, lighting controls were switched from facility manager control to personal control–and were left that way since. A community exchange, replete with Facebook pages and tweets, has formed around lighting usage in the collaboratory and there is a sense of pride in the achieved improvement. The early adopters have become advocates in spreading the facility to other floors. A new plateau of lighting energy usage has been obtained, in part because there is no extra effort to be efficient. Somebody pushing a button somewhere every so often is mandatory to keep consuming. However, we do see that people make the extra effort to turn lights off when they are the last to leave the zone, even with a timer. We note that this was not intended to be a "human factors" study nor a deep examination of human-computer-building interface design. We have not performed a latitudinal study and can not claim that our findings are representative of open office space in general, nor have we performed a longitudinal study. We built a simple, principled tool quickly to gain experience with an infrastructure intended to support innovation in energy applications and asked an interested community to give it a try. Here we report on how it was built and how it has worked so far.

4.2 Physical Infrastructure

The study was conducted in a new, seven-floor, 140,000 sq. ft. building located at a latitude of 38 degrees north. The lower three floors contain building system infrastructure, classrooms, administrative offices, instructional labs, restaurant facilities, server and communications facilities, and an auditorium. Attached to the building on the fifth floor is a large semiconductor manufacturing facility, which shares chilled water and power with the rest of the building, but is otherwise self-contained. Floors four through seven have large "collaboratories" of open office spaces stretching the east-west extent of the building with glass walls at the east and west ends providing the only day lighting. As illustrated by Figure 4.1 for the fourth floor, this bay is divided into five primary zones along the east-west

extent. Additional zones cover kitchen, entry, and various floor-specific areas. The end zones are utilized largely as meeting areas. Workspaces fill the middle three zones, with multiple research groups forming contiguous blocks. The number of lighting fixtures varies with zone size. A fixture typically contains three T-8 fluorescent tubes and two ballasts; one illuminates a single tube for low lighting, the other two tubes for medium, and both together for high.

The building as a whole consumes 800-850 kW, with the fabrication facility accounting for approximately 600 kW. The HVAC system is controlled by a proprietary Siemens Apogee Insight BMS that contains over 6,000 points spanning two cooling towers, two centrifugal chillers, one evaporative chiller, two office air-handling units, 16 fab air-handling units, and 130 variable air valves. A portion of these points are accessible over BACNet [1] through a BMS add-on. Floor-by-floor power meters were installed to measure lighting and receptacle load as part of larger energy management effort. Lighting is controlled by a separate WattStopper lighting control system that provides a facilities management console and a BACNet interface. Overall, the power consumption of the office portion of the building is roughly 143 kW, with 20 kW on average for lighting. Power consumption per collaboratory in its typical "all zones on high" mode is shown in Table 4.1. Measured power consumption is 25 watts per tube.

Floor	Area (sq. ft.)	Zones	Peak Power (kW)
4	10,654	5	7.3
5	10,923	5	6.5
6	5,599	2	2.5
7	7,102	4	4.9

Table 4.1: Lighting power in collaboratories.

4.3 Implementation

```
1 import appstack
2 api = appstack.Appstack()
3
4 def set_lights(room_number, level):
5   lights = api('#LIGHT > %s ' % room_number)
6   for light in lights:
7     light.set_brightness(level)
```

Figure 4.3: BAS implementation of personalized lighting controls.

The personalized lighting control application is implemented on top of BAS. When a user selects a lighting zone, e.g. Zone 1, a BAS query for `#LIGHT > $Zone1` is executed to select the appropriate building object. Control actions are enacted by calling

`set_brightness(percent)` on the object. Figure 4.3 show a partial implementation of the lighting control app.

At the application layer, the personalized lighting controls consist of a web app written in Django [10] that provides the user-facing interface with a control process that enacts the desired light settings. The web application provides screens for selecting the appropriate building, floor, and zone. It then shows the action screen with the current timer countdown value and brightness setting. When a user hits "Reset Timer" or modifies the brightness settings an asynchronous JavaScript (AJAX) call is made to the server which records the change in a database. Since multiple users may access the web application simultaneously, a consistent view of the current settings is essential. This is done through periodically polling the server settings in JavaScript and reflecting any changes on the page. Finally, an independent control process on the server periodically reads the light settings recorded in the database and turns on and off the appropriate bulbs by interacting with the WattStopper controller through the sMAP interface.

For security, the web application ties the existing campus Central Authentication Service (CAS) for access. Users login with the same account used to access the wireless Internet or register for classes. If users are already logged in to a different campus service, they will not be prompted to login again. We currently do not restrict access to exclusively the occupants of a specific floor or even building. Students and visitors are constantly coming and going so maintaining an access list would be prohibitively difficult. Instead, we track users accessing the system by their CAS accounts and have the ability to block any abusers of the system. In the past month and a half of operation there has been no reported abuse. Tracking accesses also allows us to gather statistics about how occupants are using the controls and the number of unique users.

4.4 Energy Savings

We first deployed the personalized automated lighting application on one floor, the fourth. This was to gain confidence in the infrastructure and experience with the approach, as well as to provide an empirical control group within a living laboratory. Half the research groups situated on that floor focus on energy and climate issues, contributing to user enthusiasm. We enabled the system on May 13, 2011 with a local member of the floor serving as the point of contact. With changes in work pattern and season, we expected to use the other floors as a reference in assessing the impact on electrical energy consumption. The system was actively used with 40 unique users in the first two weeks.

We use two methods of obtaining the lighting power consumption. Our building has per-floor lighting circuit sub-metering which provides us with total floor lighting power consumption every 5 seconds. This does include lights outside the collaboratories that we are investigating. To estimate collaboratory power we obtain relay states (i.e. on/off) for all lights in the collaboratory and multiply by the number of bulbs controlled by each relay and the typical power consumption of a light bulb. From this we obtain the collaboratory

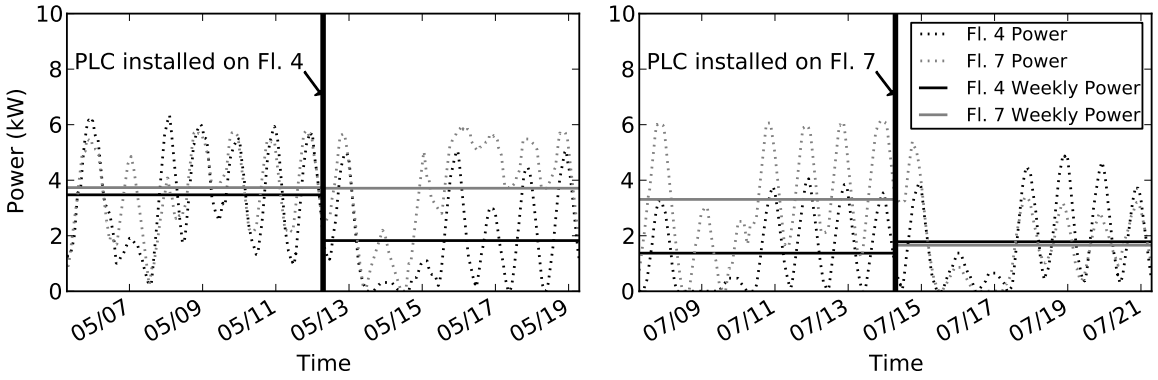
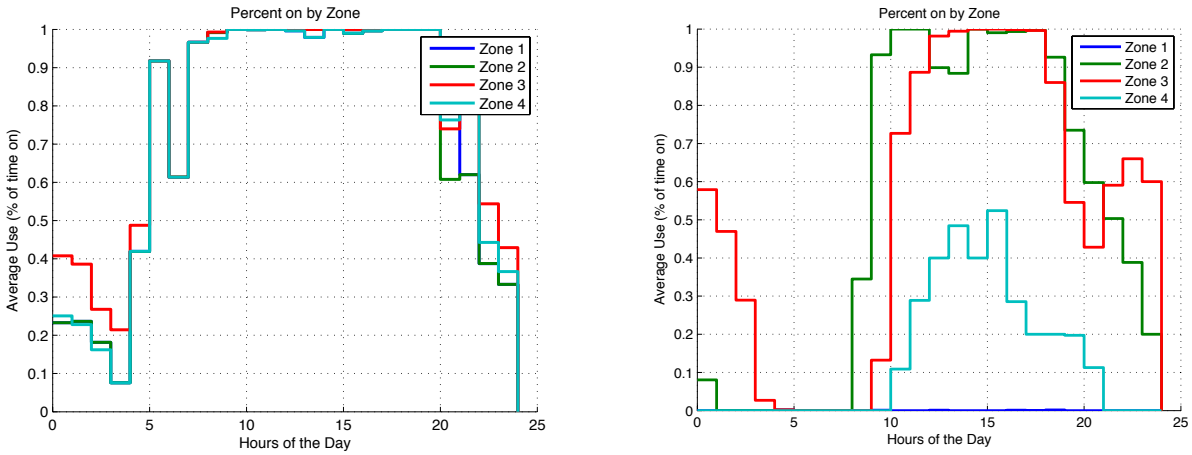


Figure 4.4: Lighting power of 4th floor and 7th floors showing a 47% reduction in energy use due to PLC.



(a) Average lighting use before personalized lighting controls.

(b) Average lighting use with personalized lighting controls.

Figure 4.5: 7th floor lighting use for each hour of the day averaged over three weeks, before and after installing PLC.

lighting power. Combining these two, relatively fine grained monitoring streams allows us to isolate the factors contributing to savings.

Figure 4.4 shows the measured total floor lighting power usage for the 4th and 7th floors before and after deploying personalized lighting controls (PLC). The 4th floor personal controls were enabled on May 13th and are followed by a 47% reduction in the average weekly power consumption. Over the same time period, power consumption for unchanged floors (e.g. floor 7 shown in figure) remained constant, suggesting that PLC was the cause of the 4th floor power drop. Some weeks later, we also deployed personal controls on floor 7,

resulting in a 50% drop in power consumption.

Week	Avg. Floor Power (kW)	Avg. Collab. Power (kW)	Collab. Savings (%)
Before	3.47	2.84	-
5/13	1.83	1.19	-58%
5/20	1.47	0.90	-68%
5/27	1.44	0.87	-69%
6/3	1.86	1.04	-63%
6/10	1.97	1.34	-53%
6/17	1.64	1.00	-65%
6/24	1.65	0.90	-68%
7/1	1.20	0.71	-75%
7/8	1.37	0.84	-71%
7/15	1.78	1.09	-61%

Table 4.2: Fourth floor energy savings over time. The energy savings do not drop off with time.

Week	Avg. Floor Power (kW)	Avg. Collab. Power (kW)	Collab. Savings (%)
Before	3.31	2.62	-
7/15	1.66	0.99	-62%
7/22	1.78	1.27	-52%

Table 4.3: Seventh floor energy savings over time.

Table 4.2 shows the total and only collaboratory weekly lighting power consumption of the 4th floor over the duration of the study. Savings of the collaboratory lighting power range from 53% to 75% compared to the week before installation. These savings are retained over the 10 weeks shown and vary primarily with holidays and occupant activity. Table 4.3 shows similar energy savings achieved on the 7th floor, demonstrating that PLC can be applied in other settings and works equally well with occupants that are not focusing on energy and climate issues.

Note that providing a relatively tight monitoring envelope around the subsystem under test is quite important. While the power savings of 1.6 kW per floor is significant it would be lost in the background of the other 825±25 kW of usage in the entire building. At the same time, finer grain monitoring allows us to isolate the factors contributing to the savings.

Source of Savings

To understand the source of energy savings we first explore the typical lighting use before the installation of personalized lighting controls. Figure 4.5a shows the fraction of time lights

were on, on average, for each zone for each hour of the day taken over a three week period before the installation of the new personalized controls on the 7th floor. An automated schedule was used to keep all lighting zones on high brightness from 10am to 7pm. At other hours occupants used the override buttons to keep the lights on. Most zones are on at the same time suggesting that occupants used the whole floor switches rather than the per-zone switches. The 6am spike in lighting use is due to janitors regularly cleaning the floor at that time. When on, the light brightness settings were set to maximum 98% of the time.

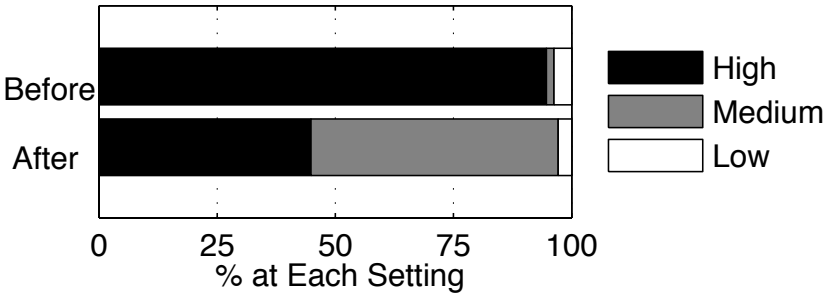


Figure 4.6: Brightness levels before and after PLC

We compare lighting use after installation of the personalized controls to this baseline to understand where the energy savings are coming from. Fundamentally, there are three sources of potential savings: part space (not illuminating all the zones when only some are needed), part time (allowing a zone to go off if unneeded), and part power (utilizing medium or even low brightness settings).

Figure 4.5b shows the lighting use after installation of PLC. Lights in zone 1, a meeting area near large windows, are now nearly always off, yielding large part space savings. In zones, 2 through 4, the lights are turned on later as occupants arrive at different times and turned off earlier in some cases. Zone 4 has only a few occupants so lights are often off even during working hours. Finally, brightness levels are set significantly lower than before as shown in Figure 4.6. The average brightness setting is medium yielding significant part power savings.

Figure 4.7 shows the breakdown of energy use before and after PLC. Energy is broken down by time: working hours (9am to 6pm), non-working hours (labeled evenings), weekends and by zone. For each zone and time, the full bar represent average weekly energy use before PLC implementation and the white bar represents the energy use after implementation. The difference, energy savings, is shaded indicating part power and part time savings. Overall, the majority of energy is saved from keeping lights on part time rather than on lower brightness levels. However, during working hours in the cubicle zones brightness settings make up most of the savings. The meeting area in zone 1 is kept off most of the time, however, those energy savings are dwarfed by turning off the large cubicle zone 3 slightly earlier in the evenings and on slightly later in mornings.

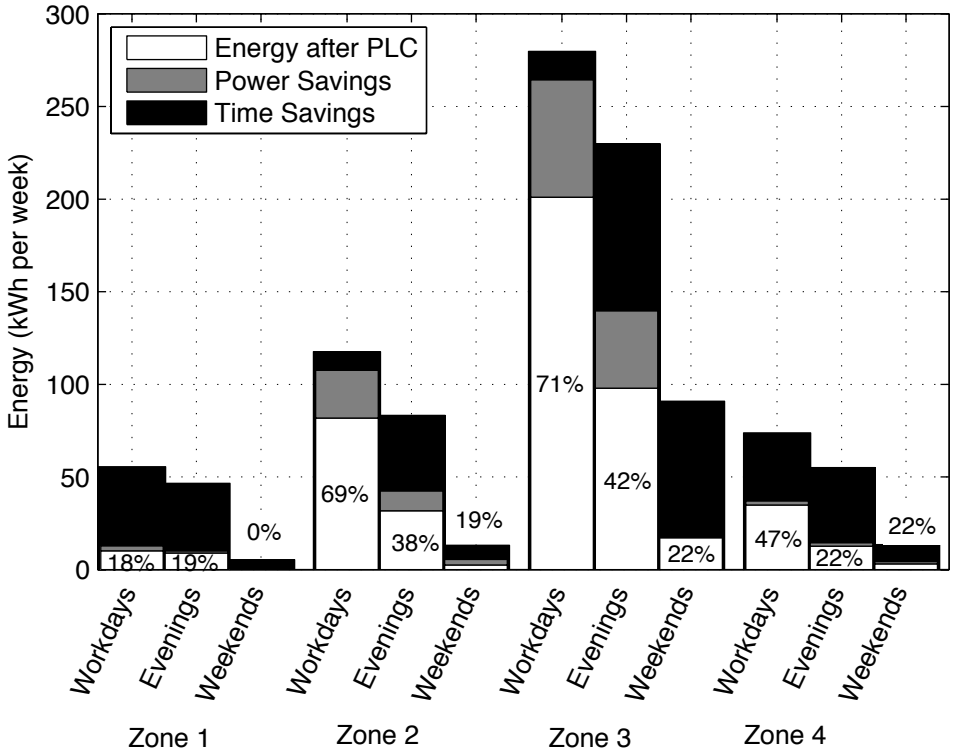


Figure 4.7: Breakdown of energy use and savings. Full bars represent the baseline weekly energy use before PLC. Shaded regions show sources of energy savings. The white bars show weekly energy consumption after PLC, labeled with the percent of baseline energy.

4.5 Enabling Extensions

Within one week of enabling PLC we observed that building occupants were independently creating applications on top of it. One student wrote a short Linux script to automatically extend the lighting control timer only when he was actively using his laptop and connected to one of the wireless access points on his floor. We encouraged this type of rapid development by providing an easy to use web service API.

We subsequently extended this idea by developing a cross-platform application with a graphical user interface. Occupants can now optionally download this alternative interface to the PLC on their laptops. The program asks users to input their typical seating zone and brightness preference. It then uses a combination of activity detection and coarse-grained WiFi localization, based on per-floor BSSID fingerprints, to determine when the lights should be kept on.

Alternative occupancy detection techniques could just as easily be programmed to control the lights using this simple web interface, enabling support for buildings with or without occupancy sensing hardware.

Chapter 5

Conclusion

Modern commercial buildings contain some of largest deployed sensor networks, often consisting of thousands of sensors and actuators. This infrastructure has the potential to enable a wealth of applications that change the way we interact with buildings, reduce energy consumption, support grid operation, improve reliability and maintenance.

The main challenge is to make applications portable and easy to develop. All buildings are designed differently and building controls are inconsistent in naming, function and available features. Today this requires developers to have detailed knowledge of each building's architecture, HVAC design, control network and hardware functionality.

BAS abstracts these details, allowing application authors to select building components with fuzzy queries and use standardized methods built up through a hierarchy of drivers to read or actuate the building. BAS queries are key to portability; they allow authors to select components in terms of functional or spatial relationships that are implicitly portable.

We demonstrate three control applications implemented using BAS, a ventilation optimization app and an occupant temperature control app and a personalized lighting control application. We show that BAS applications are much shorter, easier to understand and do not change form building to building. We also demonstrate how these applications can have a real energy impact, saving over half of lighting power.

Bibliography

[1] ASHRAE. *ANSI/ASHRAE Standard 135-1995, BACnet*. 1995.

[2] Automated Logic Corporation. "WebCTRL v5 User Manual". In: (2011).

[3] AutomatedLogic. *ALC WebCTRL*. 2012.

[4] Vladimir Bazjanac et al. "HVAC Component Data Modeling Using Industry Foundation Classes". In: *System Simulation in Buildings*. 2002.

[5] CA Energy Commission. *California's Energy Efficiency Standards for Residential and Nonresidential Buildings*. 2008.

[6] Stephen Dawson-Haggerty, Andrew Krioukov, and David E. Culler. *Experiences Integrating Building Data with sMAP*. Tech. rep. UCB/EECS-2012-21. EECS Department, University of California, Berkeley, Feb. 2012. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-21.html`.

[7] Stephen Dawson-Haggerty et al. "sMAP — a Simple Measurement and Actuation Profile for Physical Information". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 2011.

[8] Stephen Dawson-Haggerty et al. "sMAP: a simple measurement and actuation profile for physical information". In: *SenSys '10*. 2010.

[9] Robert Dickerson et al. "Stream feeds: an abstraction for the world wide sensor web". In: *IOT'08*. Zurich, Switzerland, 2008.

[10] Django Software Foundation. *Django*. https://www.djangoproject.com/. 2012.

[11] Echelon Corporation. *LonTalk Protocol Specification*. 1994.

[12] EERE. *EnergyPlus*. http://energyplus.gov. 1998.

[13] Varick L. Erickson, Miguel Á. Carreira-Perpiñán, and Alberto E.Cerpa. "OBSERVE: Occupancy-Based System for Efficient Reduction of HVAC Energy". In: *IPSN'11*. 2011.

[14] Anca D. Galasiu and Guy R. Newsham. "Energy Savings Due to Occupancy Sensors and Personal Controls: A Pilot Field Study". In: *Proceedings of Lux Europa 2009*. 2009.

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

[16] Edward W. Kamen. *Industrial Controls and Manufacturing*. Academic Press, 1999.

[17] Andrew Krioukov et al. "A Living Laboratory Study in Personalized Automated Lighting Controls". In: *BuildSys'11*. 2011.

[18] Andrew Krioukov et al. "Building Application Stack (BAS)". In: *Proc. of 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*. 2012.

[19] Xuesong Liu et al. "Requirements for A Formal Approach to Represent Information Exchange Requirements of a Self-managing Framework for HVAC Systems". In: *ICCCBE*. July 2012.

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

[21] T Moore, DJ Carter, and Al Slater. *A Qualitative Study of Occupant Controlled Office Lighting*. 2003.

[22] G. Newsham et al. "Effect of Dimming Control on Office Worker Satisfaction and Performance". In: *Proceedings of: IESNA Conference 2004*. 2004.

[23] OPC Task Force. *OPC Common Definitions and Interfaces*. 1998.

[24] *PostGIS*. http://www.postgis.org/.

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

[26] Jaspal S. Sandhu, Alice M. Agogino, and Adrian K. Agogino. "Wireless Sensor Networks for Commercial Lighting Control: Decision Making with Multi-agent Systems". In: *In AAAI Workshop on Sensor Networks*. 2004, pp. 131–140.

[27] Jeffrey Schein et al. "A rule-based fault detection method for air handling units". In: *Energy and Buildings*. 2006.

[28] Siemens. "APOGEE Actuating Terminal Equipment Controller—Electronic Output Owner's Manual". In: (2004).

[29] Siemens. "APOGEE Powers Process Control Language (PPCL) User's Manual". In: (2000).

[30] U.S. Department of Energy. *2011 Buildings Energy Data Book*. 2012.

[31] U.S. EPA/Office of Air and Radiation and Consumer Product Safety Commission. *The Inside Story: A Guide to Indoor Air Quality*. 1988.

[32] W3C. "SOAP Version 1.2". In: (2007).

[33] WattStopper Lighting Integrator. `www.wattstopper.com`.

[34] Yao-jung Wen, James Bonnell, and Alice M. Agogino. "Energy Conservation Utilizing Wireless Dimmable Lighting Control in a Shared-Space Office". In: 2008.

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