XBOS: An Extensible Building Operating System

Gabriel Fierro
David E. Culler

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-197
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-197.html

September 9, 2015

Copyright © 2015, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

XBOS: An Extensible Building Operating System

Gabe Fierro
UC Berkeley
fierro@eecs.berkeley.edu

David E. Culler
UC Berkeley
culler@cs.berkeley.edu

ABSTRACT

We present XBOS, an eXtensible Building Operating System for integrated management of previously isolated building subsystems. The key contribution of XBOS is the development of the Building Profile, a canonical, executable description of a building and its subsystems that changes with a building and evolves the control processes and applications accordingly. We discuss the design and implementation of XBOS in the context of 6 dimensions of an effective BAS – hardware presentation layer, canonical metadata, control process management, building evolution management, security, scalable UX and API – and evaluate against several recent building and sensor management systems. Lastly, we examine the evolution of a real, 10 month deployment of XBOS in a 7000 sq ft office building.

1. INTRODUCTION

Buildings have long been a focus of advanced automation techniques to improve energy efficiency and occupant comfort, as they are the dominant electricity consumer and where we spend most of our time. However, even in large commercial buildings with advanced building management systems, building subsystems remain largely siloed, with little integration of lighting control with HVAC or with appliance usage or with occupant calendars. Recent efforts have begun to integrate building subsystems into unified, cyber-physical solutions, but fail to capture comprehensive, evolvable and sufficiently abstract representations of those systems.

This paper develops the principled design of a new, extensible building operating system. XBOS, based on six primary dimensions of the BOS design space. In particular the design captures the much bemoaned, longstanding challenge of support for the inevitable evolution of a building throughout its lifecycle, which is only more acute with smart buildings [7]. Addressing this challenge impacts every aspect of BOS design. We present the XBOS design in detail along with a comparative analysis across the design space.

2. DESIGN SPACE

Considerable research has investigated the design and implementation of flexible, principled execution environments for building monitoring, analysis, and control as alternative to the rigid, ad hoc designs of traditional building management systems. Based on these investigations, it is now possible to articulate the primary dimensions of the design space of *building operating systems* (BOS). In our analysis, there are six such dimensions:

- Hardware Presentation Layer
- Canonical Metadata Definition, Storage, and Usage
- Control Process Management
- Building Evolution Management
- Security: Authorization and Authentication
- Scalable UX and API

A high level mapping of the state-of-the-art relative to these dimensions is provided in Table 1. The assessment yielding the markings in the figure is provided in Section10 as part of the evaluation of XBOS, after having developed its design. Here we identify the core issues associated with each of these dimensions to frame the design. While advancing in almost every dimension, its primary contribution is graceful handling of the evolution in operation over the building lifecyle, which can be rich and vibrant with the artificial constraints of traditional BMS removed. We seek to show how embracing change as a primary requirement impacts the design of essentially every aspect of the system. The six dimensions also serve to structure the presentation.

Hardware Presentation Layer (HPL): Traditional BMS solutions are vertically integrated despite using open protocols such as BACnet, often overlaying proprietary networks to connect to vendor-specific control code. Introducing new devices or control logic is a major effort requiring building technology professionals to write proprietary glue code or install new hardware for devices from external vendors. Most BOS solutions establish uniform read/write access to data sources in a building, typically as RESTful web services or agents, through a standard library or framework installed on gateways and servers in the building. sMAP [5] helped to introduce the idea of a uniform data plane for devices in buildings. The XBOS HPL removes the reliance on any particular heavyweight device client and expands the domain of devices that can natively interact with the building operating system (Section 5).

Canonical Metadata: Standardized building metadata and system representation is a subject of active research and standardization [15] [13] [12] [10]. Traditional BMS typically use proprietary, opaque descriptors on individual points internally and SCADA tags with naming conventions to capture the attributes of a point, such as its type, location, and function. Operating on an abstracted representation of a building simplifies and increases the portability of control and application code. Several efforts standardize the representation of building objects through attributes and links, but permit arbitrary structures with little semantics. Build-

	BeMOSS	BAS	BOSS	Sensor Andrew	HomeOS	SensorAct	Volttron	Building Depot v2	Tridium	**XBOS**
HPL	-	✓	✓	-	-	-	-	-	✓	+
Canonical Metadata		-	-		-			✓		+
Control Process Mgmt	-		✓		✓	-	-	-	-	+
Bldg Evolution Mgmt		-	-					-		+
Security	-	✓	✓	✓	✓	✓	-	-	✓	✓
UX	-	-		-	✓			✓	-	+
API	-	-	-	-	-	-	-	✓	-	+
Supports BMS		✓	✓			-		✓	✓	✓
No BMS, Non-res	✓	✓	✓			-	-			✓
Residential	✓	✓	✓		-	-	-			✓
Open Tech Doc	✓				+	✓	+	+		+
Deployed At Scale				+	+		+	+	+	+

Table 1: Comparison of BOS features. This Table will be discussed in detail in Section 10. A blank cell means that the BOS does not address the feature, a -means problem is addresed, but not sufficiently, a ✓means solution is not well-developed, and a +means that the BOS solution pushes the state-of-the-art [2]

ingDepot2 [16] stands out in its ability to describe the hierarchical subsystems typically present in a building. XBOS extends this with the Building Profile, providing multiple hierarchies (e.g. spatial, HVAC, lighting, electrical, monitoring) with strict structure and attributes to provide clear semantics, but cross-links through dynamically computed attribute relationships to provide flexibility (Section 3).

Control Process Management: Closed loop control in traditional BMS is typically hardcoded to a specific set of opaquely described sensors and actuators and embedded in programmable logic controllers (PLCs). Aside from requiring a specialized technician to program, the ability of PLCs to adapt to new hardware or logic is limited by the requirement that these loops be hardcoded to a set of inputs and outputs described by internal network identifiers. Advances in control theory are often applied to individual buildings as “one-offs” for lack of sufficient abstraction to describe those processes. The BAS [11] and BOSS [6] approach to control processes gives loops access to an abstract representation of the building, which enables code reuse and makes control logic easier to reason about. Furthering this idea, XBOS describes the inputs and outputs of control processes using canonical metadata, which lets these processes evolve naturally with the building (Section 7).

Building Evolution Management: The classical lifecycle of buildings, with changes in physical structure, components, and usage, is a widely recognized challenge. When approached as cyberphysical systems, change becomes more natural, but the challenge is for the software operating on the building to recognize changes and adapt. BuildingDepot2 was one of the first to step away from hardcoded processes, leveraging canonical metadata to describe control loops and applications. However, the processes themselves are tied to specific endpoints bound by resolving the metadata to points statically. An instance of XBOS is constructed lazily using continuously-evaluated queries against the Building Profile rather than explicitly resolved by traversals of hierarchies. As the Building Profile evolves, so do the results of the queries, and the processes and application in XBOS dynamically adapt (Section 3). Observe that such adaptation is possible because control processes operate on canonical metadata that is determined dynamically and evolves with the building.

Security: Security is critical in all building management solutions. The “state of the art” for authorization in commercial building systems is user accounts in the BMS and BACnet priority arrays, with access control lists (ACLs) implemented in the equivalent of an overlay network. In most BOS designs, authentication is addressed by TLS and authorization is largely handled by user accounts and ACLs of varying granularity on the components of a building. SensorAct [3] advances a fine-grained, policy-oriented approach to managing the array of sensors and actuators in a building with user ACLs on a room and device basis. XBOS does not innovate in this space, but recognizes that the state of current solutions is far from a comprehensive approach to security and provides a framework for future work in distributed delegation and management of trust.

User Experience and API: Besides environmental conditions, the user interface for a building is the primary point of contact between stakeholders and the building systems. User interfaces for traditional BMS are typically animated mechanical diagrams of specific subsystems in buildings. While helpful for HVAC engineers and other technical specialists, these do not present views of holistic building state that are vital for determining the health of building at a glance, nor do they give the integrated, cross-subsystem view that is most helpful for occupants (Section 11).

Classes of Intended Deployments: The main classes of buildings which a BOS can address are:

- large-scale commercial buildings characterized by one or more incumbent BMS, typically with a full HVAC system and multiple floors and rooms and administra-

[2]Sensor Andrew was extended to work on building as Mortar.io, but there is no existing technical documentation for the project

tive domains

- small-medium commercial buildings, which have distinct, non-managed subsystems (thermostat wired directly to an RTU) but can potentially retrofit certain systems, e.g. by installing a smart thermostat
- residential bulidings, which are usually very simple (1 HVAC zone) and are much easier to retrofit with networked components

Most of the BOS solutions considered are written towards a particular classes of buildings. Issues of scale directly effect the design of the UX, e.g., a holistic view across HVAC, lighting, and plug loads may be a single screen in a small building, whereas large commercial buildings have hundreds of zones and require various hierarchical views. But scale also presents potential performance challenges for a BOS, especially as relationships are resolved dynamically. XBOS provides a framework for developing scalable interfaces and tracks entity relationships to minimize dynamic resolution.

3. THE XBOS BUILDING PROFILE

The outstanding BOS challenge is mitigating the complexity in describing and referencing the natural hierarchies inside a building *as they change over time*. A building profile must be able to manifest a representation of the building at a point in time, gracefully handling changes in the building's description without substantial manual intervention. Traditional lifecycle changes involve retrofits which may change floorplan and the HVAC layout, building system upgrades, and new components. In a BOS, they may also be enhancements to metadata as more is recorded or learned about the building, new control algorithms or application deployed, cross-system optimizations, and so on.

Standards, such as oBIX, Project Haystack, gbXML, IFC, focus on how to construct the graph that represents an instance of a building at a point in time. To effectively incorporate the lifecycle of the building, that graph must be constructed from some underlying database that represents the building as it evolves. In XBOS, the Building Profile captures the family of relationships and builds the graph automatically. These relationships offer a well-formed abstraction for implementing monitoring and control processes in buildings, and inform the design of the XBOS architecture.

As concrete example of building evolution, consider the expansion of a control loop around a thermostat. Upon installation, each thermostat is preprogrammed with a basic internal schedule. With several of these thermostats installed in a building, the natural BOS progression is to introduce a master schedule governing all thermostats in a space. The scheduler is an open-loop control process and the thermostats subscribe to its setpoint. After some time, it may be discovered that the poor placement of a thermostat causes a set of rooms to be overly cooled, prompting the installation of temperature sensors in the other rooms in that HVAC zone. A zone controller could be introduced that subscribes to both the master schedule setpoint and the temperatures of rooms in the HVAC zone to bias the scheduled setpoint to account for the room temperature differences. Occupancy sensors in the lighting control system might then be used to avoid conditioning empty rooms. This draws on relationships across distinct subsystem and evolves as physical and executible components are added. The physical building has not changed, but its components and their relationships evolve.

```
[/]
Metadata/Site = UC Berkeley
Metadata/Building = CIEE

[/spatial/floor/2]
Metadata/Floor = 2

[/spatial/floor/2/room/207]
Metadata/Room = 207
Metadata/Type = Room
Metadata/Name = Conference Room

[/hvac]
Metadata/System = HVAC

[/hvac/zones/zone2]
Metadata/HVAC/Zone = Zone2
Metadata/Name = South Offices
Metadata/Rooms = [200, 205, 206, 207, 208]

[/hvac/equipment/thermostats/tstat2]
Metadata/Name = Conference Room Thermostat
Metadata/HVAC/Zone = Zone2
Metadata/Equipment = RTU
Metadata/RTU = rtu2
Metadata/Location = Room
Metadata/Room = 207
type = smap.drivers.thermostats.CT80

[/hvac/equipment/thermostats/tstat2/temperature]
Properties/UnitofMeasure = C
Properties/UnitofTime = s
Metadata/Sensor/Measure = Temperature
Metadata/Sensor/Type = Sensor
```

Figure 1: Demonstrating a subset of Spatial and HVAC hierarchies with key-value pairs applied. Key-value pairs are implicitly inherited down the hierarchy, so `Metadata/System = HVAC` will be applied to everything under `/hvac/.*`. The `type` key references an instantiation of a sMAP *driver* that will define its own paths relative to its parent e.g. `/temperature`, seen here

Each subsystem in a building – HVAC, lighting, spatial, electrical, control, etc. – can be represented as a hierarchy of entities and their relationships: a floor *contains* some rooms, a VAV *has* a return air temperature sensor. Several methods can represent these hierarchies (BuildingDepot [3], oBIX [15], Project Haystack [13], gbXML [9], IFC [10]), but these fail to capture relationships across subsystems. For example, an HVAC control loop may need to find the temperature sensor for any room in a zone, a relationship spanning the HVAC and spatial hierarchies.

3.1 The Right Level of Abstraction

Building modeling solutions, such as Energy Plus [4] and Modelica [8], rely on detailed descriptions of the components within a building to construct an accurate simulation model and thus abstract very little – usually just communication to the component. Other detail-heavy schemes for describing buildings include Industry Foundation Classes (IFCs [10]) and (more generally) Building Information Modeling (BIM), which are intended for the physical construction of a building and can be used as input for Modelica or Energy Plus. These models contain detailed information such as the material and thickness of walls and the estimated BTU output of an RTU – information not pertinent to general monitor-

[3]Only captures limited, predetermined relationships across hierarchies

```
-- All temperature sensors on the 2nd floor
select * where Metadata/Sensor/Measure = Temperature and Metadata/Sensor/Type = "Sensor" and Metadata/Floor = 2;
-- The heating setpoint for Room 207
select data before now, Properties/UnitofMeasure where Metadata/Sensor/Measure = "Temperature" and
    Metadata/Sensor/Type = "Setpoint" and Metadata/Sensor/Setpoint = "Heating" and Metadata/Room = 207;
-- How much energy is Room 207 using right now?
select data before now where Metadata/Sensor/Measure = "Power" and Metadata/Room = "207";
-- Which rooms in Building ABC have occupancy sensors
select Metadata/Room where Metadata/Building = "ABC" and Metadata/Sensor/Measure = "Occupancy";
```

Figure 2: Example XBOS queries against the Building Profile.

ing and control applications. Furthermore, each constructed model is unique to its building and cannot be effectively reused, introducing a significant cost for incorporating new buildings.

Other solutions abstract to provide flexible, descriptive power, but do not contain enough structure to allow applications to make meaningful assumptions. The Web Ontology Language (OWL) for the Semantic Web is intended to represent knowledge about "things, groups of things and relations between things" [4], and is therefore able to describe a wide variety of systems. However, with no assumptions about the structure of the system, any usage of an OWL ontology requires re-establishing the structure of required relationships.

Project Haystack [13] seeks to provide semantic structure to the body of devices and components in buildings. Each entity contains a flat list of tags from an established vocabulary, and has a notion of a generic link between two entities. However, few restrictions are placed on the relationships that can be established between entitites, so similar relationships may be represented in different ways in different buildings. And with few complete building examples openly available, even idiomatic conventions are hard to establish.

3.2 XBOS Building Profile

The XBOS Building Profile models building subsystem hierarchies as *paths*. It has a collection of pre-defined path templates for the HVAC, lighting, spatial, electrical and administrative subsystems; new custom hierarchies are easily added. Descriptive key-value pairs are applied to a point in a path and are inherited down to all prefixed children. When the profile is instantiated, these key-value pairs propagate down the paths. The resulting sets of key-value pairs are associated with a globally unique identifier (UUID) – representing a unique timeseries of `<timestamp,value>` pairs called a *point* – and stored in a flat namespace. These *points* can be referenced and manipulated by SQL-like operations on the key-value pairs, the UUID, or prefixes of the path used to describe the point.

Key names are drawn from a well-defined namespace, though a profile may define its own keys, e.g. `Metadata/HVAC/Zone` for an HVAC Zone and `Metadata/Sensor/Measure` for identifying the target quantity for a sensor. More keys can be found in the XBOS Building Profile example in Figure 1. The use of keys, such as `Metadata/HVAC/Zone`, is not limited to a single hierarchy. Keys can be used in multiple hierarchies and are used by the query language as foreign keys to perform "joins" between points to establish ad-hoc relational views.

The partial example of a profile in Figure 1 expands parts of the Spatial and HVAC hierarchies for a deployment described in Section 11. Key-value pairs applied at a "path" are inherited to all children that share that prefix. For example, pairs applied at `[/hvac]` will be applied down to `/hvac/equipment/thermostats/tstat2`. Likewise, the Building and Site keys applied at the "root" of the hierarchy, `[/]`, are applied to all paths in the profile. This figure is not the internal representation, but is resolved statically at boot time by sMAP and sent to the XBOS Kernel.

The key-value pairs at `/spatial/floor/2/room/207` and `/hvac/equipment/thermostats/tstat2` both contain the key-value pair `Metadata/Room = 207`, which can be used by the query language to treat `Metadata/Room` as a foreign key relationship between the thermostat and room. Furthermore, because `Metadata/Room` is inherited to all children of the thermostat, a query could associate a channel of the thermostat with the room as a generic termperature sensor. This same query could perform the association whether the temperature sensor was from a thermostat, a VAV return air sensor, or a wireless embedded temperature sensor, since all of these would use the same metadata key.

This collection of paths and key-value pairs is specified in a plain-text configuration file like the one in Figure 1 that can be easily kept under version control. Upon the execution of this file by an XBOS process, the key-value pair inheritance is performed and the resulting points are loaded via the XBOS Kernel into the metadata store (described in Section 6) where they can be utilized by the query processor.

3.3 Query Language

The XBOS query language captures ad-hoc relations between collections of points, either as one-time results or as continuously evaluated relational views. Because continuous views operate over the building profile, as the profile changes, so does the evaluation of a continuous query as a view. These queries are the basis for the XBOS system, and the architecture of XBOS is designed around leveraging continuous views so the system can gracefully react to changes in the building. The XBOS query language is a backwards-compatible complete re-implementation of the SQL-like sMAP query language that supports an expanded set of predicates on metadata and contains an explicit ab-

[4] `http://www.w3.org/2001/sw/wiki/OWL`

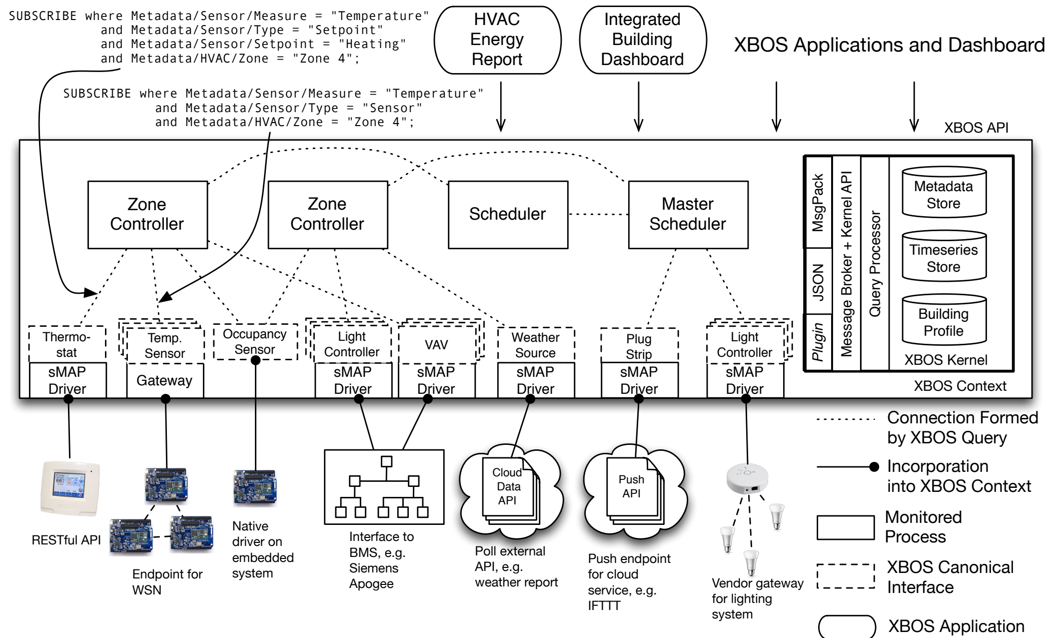

Figure 3: XBOS system architecture. Drivers can present more than 1 interface, and more than 1 instantiation of those interfaces, such as a lighting controller gateway or a smart plug strip

stract syntax tree used by the XBOS Kernel for intelligent reevaluations.

A query has two main components: a *select clause* and a *where clause*. The select-clause defines the desired output of a query or the contents of the materialized view [5]. A select-clause can extract key-value pairs, paths, UUIDs or timeseries data. The where-clause describes a predicate over key-value pairs, paths and/or UUIDs that conducts a join between hierarchies and points. Some example XBOS queries for the building from Figure 1 can be found in Figure 2.

Continuous views are handled by the XBOS Kernel pub-sub broker. When a continuous query is excecuted by a client, the query initially behaves like a traditional execute-once query, but then continues to give the client updates on changes in the set of metadata or points that match the view. These updates are delivered in real-time as changes in metadata are executed against the building profile. It is up to the client to maintain state and apply changes; after the initial delivery of the query evaluation, the XBOS Kernel delivers "diffs" on the set of matching points. The mechanism behind efficient reevaluation of continuous queries and implementation details of the query language is discussed in Section 6.

4. XBOS ARCHITECTURE

The architecture of XBOS centers on the Building Profile, which defines the interface to the underlying building and building systems and the means by which controllers and applications discover each other and describe their inputs and outputs, as illustrated in Figure 3. The XBOS architecture is a variation on the typical layered architectures seen in academic BOS (BEMOSS and BOSS) as well as industrial solutions (Tridium [14]):

- *Devices*: at the bottom of Figure 3 are the physical devices in the building, such as sensors, VAVS, thermostats and lighting controllers. "devices" can also be external data sources such as a weather API or physical resources in a building such as rooms or cubicles.
- *Drivers*: the HPL consists of continuously running processes that conform the representations of the underlying "devices" to the XBOS Building Profile (Section 5).
- *XBOS Kernel*: the central component of XBOS. The Kernel is the data historian for both metadata and timeseries data and contains the query processor and the pub-sub broker. All connections between XBOS components take place via the XBOS Kernel (Section 6).
- *XBOS Services*: Building operation logic is handled by continuously running processes that manifest higher-level logic and interconnections between drivers, the XBOS Kernel and the Building Profile (Section 7).
- *Applications + Dashboard*: XBOS applications including the building dashboard are written over the XBOS API, which is automatically generated from the Building Profile and facilitates application portability. The building dashboard provides integrated monitoring and control of all building subsystems. Implementation of these applications is discussed in Section 8.

[5]the language also supports *set* and *delete* queries that also use where clauses. These are evaluated once, not continuously

Drivers, services, applications and the kernel operate within the "XBOS Context", in which all descriptions are in terms of the XBOS Building Profile and all communication is done via the XBOS Kernel. A sample organization of a real-world XBOS deployment is in Figure 4. The XBOS architecture places no strict requirements on the placement of its components, though recommendations for network topology are mentioned in Section 9.

Figure 3 shows the relationships between components in an instance of XBOS at a stage in the scenario of evolution discussed in Section 3. Within the XBOS context – that is, through continuous views executed against the XBOS Kernel – a zone controller subscribes to the outputs of a mesh of temperature sensors and an occupancy sensor as well as a scheduler process. This zone controller computes the temperature bias introduced by the measured temperatures of occupied rooms and then pushes the calculated setpoint to the thermostat in the space.

5. HARDWARE PRESENTATION LAYER

There are two tiers of interoperability addressed by the HPL: "logical" or syntactic interoperability – the ability to speak the same protocol – and semantic interoperability – the ability to discover and understand the interface to another device. Large commercial BMS typically have multiple subsystems (such as HVAC and lighting) controlled by different vendors each with their own protocol and description semantics. In small-medium commercial buildings and residential, there is typically a collection of vertically-integrated, vendor-specific gateways controlling separate subsystems; a single building may contain Nest (HVAC), LIFX (lighting) and PG&E SmartMeters (electrical).

A HPL solution also needs to support building evolution. When a device is added to or removed from the HPL, the system needs to recognize that change at the higher levels of the system. The XBOS HPL captures both metadata and timeseries data of each possible entity in a building management system, and pushes this unified description into the Building Profile.

XBOS leverages earlier work by sMAP to define a profile that mandates what information should be contained in a description (semantic) rather than the structure of that description (syntactic). This means that the XBOS HPL can function over arbitrary protocols, provided that an adapter exists on the XBOS Kernel (detail in Section 6).

5.1 XBOS HPL

The XBOS HPL is a set of drivers that expose canonical interfaces that adhere to the XBOS Building Profile. Drivers are continuously running processes that represent the underlying features of API of a device as a set of canonical interfaces. An XBOS driver publishes the state of its devices to the archiver by writing to a specific port on the kernel (depending on which protocol the client is using). Actuations, metadata updates and other subscription-based information are pushed to the client through the socket's connection to the kernel pub-sub broker.

The XBOS profile (Table 2) mirrors the sMAP profile and dictates what information should be included in all messages sent from a driver to the kernel. The simple key-value structure is trivial to construct in most protocols, which simplifies client code. `Metadata`, `Properties` and the `Path` follow an "upsert" policy on the archiver – this simplifies update semantics, but is also beneficial for clients with network constraints. The `Readings` field is delivered to the timeseries database.

XBOS supports actuation through NATs: upon instantiation, drivers initiate a subscription to a specific actuation stream. The kernel pushes published actuation "commands" (actually readings on that stream) to the subscribed driver, which can then take action. The properties of each stream help determine if any translation must take place, such as unit conversion.

Two examples of drivers can be found online for the CT-80 RTA WiFi Thermostat [6] and the IMT550C Ethernet Thermostat [7]. The IMT550c thermostat exposes an API very similar to the canonical interface expected by XBOS, so the driver performs a simple passthrough. However, the CT-80 API does not allow a user to alter the deadband and keep the thermostat in AUTO mode simultaneously, so the driver manages that state internally to expose the expected behavior.

Key (Prefix)	*Description*
Path	the hierarchical path that determined the inheritance of key-value pairs for this point
UUID	globally unique identifier for this point
Readings	list of `<timestamp, value>` pairs to be archived
Metadata	set of key-value pairs that adhere to XBOS Building Profile
Properties	set of key-value pairs for describing the point, e.g. units, timezone, time resolution

Table 2: Contents of XBOS profile messages, which follow the same structure as sMAP messages.

6. XBOS KERNEL

The adaptibility of the XBOS building profile to changes in the building is made possible because it operates on a family of relationships that are not captured in any table. *Each continuous view establishes its own foreign-key relationships on demand* rather than operating over a strict, predetermined schema that prevents the representation of the building from evolving. The benefit of using a predefined schema is that a BOS can make use of existing performant tools for executing queries – this includes Object Relational Models, which are a convenient abstaraction for forming relationships between entities. However, we have found that the strict relational model prevents the system from realizing emerging relationships required by controllers but not reflected in the schema. While these lifecycle changes to the building system are not frequent, they are essential, and designing the system around automatically handling such

[6] https://github.com/SoftwareDefinedBuildings/smap/blob/unitoftime/python/smap/drivers/thermostats/ct80.py

[7] https://github.com/SoftwareDefinedBuildings/smap/blob/unitoftime/python/smap/drivers/thermostats/imt550c.py

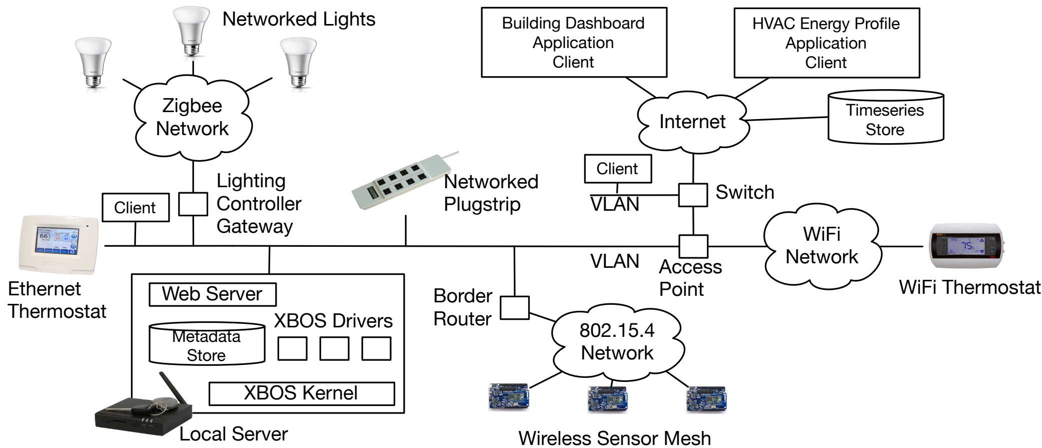

Figure 4: XBOS distributed system organization. XBOS is designed to scale up and down, and places no strict requirements on the location of its components. Illustrated here is a partial view of the deployment described in Section 11

"migrations" greatly increases the practicality of handling building evolution.

The XBOS Kernel contains a high-performance, broker-integrated query processor that takes a family of foreign key relations from the building profile and constructs new tables dynamically. We demonstrate in Figure 5 that the performance cost of dynamically forming these relationships at runtime does not introduce substantial overhead. The cost is mitigated by caching the results of queries and intelligently choosing which views to reevaluate by traversing the abstract syntax tree of pending metadata changes. The XBOS Kernel implements these mechanisms that allow the rest of the system to handle changes in building data, and presents the API and HPL to users of the system. The XBOS Kernel is written in the Go programming language [1] and is backwards-compatible with the sMAP ecosystem, but introduces several new features including scalability, ease of deployment, multiprotocol plugin support, an expanded query language and support for continuous relational views.

Multiprotocol support: The XBOS Kernel exposes interfaces over multiple encodings and transports to lower the implementation barrier for constrained clients and to obviate middleware for common but specialized tasks such as real-time web applications or an embedded program subscribing to a query. Transports and encodings currently supported by the kernel are MsgPack/UDP, JSON/HTTP(S), ProtoBuf/TCP, CapnProto/TCP and JSON/WebSockets, but this can be easily extended with the kernel's simple plugin model.

Continuous View: Continuous views are ultimately resolved as queries against an underlying metadata store (indicated in Figure 3). The current XBOS Kernel is backed by a MongoDB instance, but other DBMS can be used with a small kernel plugin. In contrast to the relational model of other BOS, XBOS does not store the trees of hierarchically structured metadata, nor does it store the relationships between trees. All key-value pairs are applied down hierarchies and each point is stored in the metadata database in a flat name space, with a `Path` field designating the point's position in a hierarchy.

To realize continuous relational views, the XBOS Kernel maintains a mapping of where-clauses (which define the contents of a view) to the list of clients subscribed to that clause. The kernel further decomposes each where-clause into the set of keys in that clause. Processes only access the metadata database through the kernel, so whenever a new piece of metadata arrives at the kernel – as part of a new source, changed metadata, or otherwise – the kernel extracts the set of affected keys and selectively re-evaluates the queries that are concerned with those keys. The re-evaluation is cached in kernel memory as a list of publishers for each subscription. The XBOS Kernel delivers metadata events as well as timeseries events, so the evolution of the building profile and the streams of data within is *completely transparent to the client.*

7. XBOS SERVICES

Services in XBOS can be used to implement both closed- and open-loop control over an evolvable representation of a building. These services use the same structure as the driver execution containers in the HPL and communicate using the same profile. XBOS controllers can exist anywhere in a deployment, using continuous views executed against the XBOS Kernel to operate on the most recent version of a building.

The outputs for a controller are analogous to "channels" for a sensor or timeseries endpoints of a sMAP driver: they are a stream of data identified by a UUID and have associated properties (units, etc) and metadata. In XBOS, services are first-class citizens of the Building Profile, meaning they are described in the same way as devices or other entities. Other systems such as BuildingDepot2 have a separate notion of a "virtual sensor" which is an aggregate of multiple data sources, but this is generalized in XBOS as a subscription (continuous view).

For example, an XBOS controller that wishes to bind the state of the lights in a specific room to an occupancy sensor would execute a view with the clause `Metadata/Sensor/Measure = "Occupancy" and Metadata/Sensor/Type = "Sensor" and Metadata/Room = 410` which would deliver in real-time the values of all occupancy sensors in Room 410 to the controller. This subscription allows the control process to remain agnostic both to the number of occupancy sensors that match the query but also the semantics of those sensors, thanks to XBOS's notion of canonical interfaces.

8. UX AND API

XBOS has a dynamic, implicitly constructed API generated from the set of devices and services in an installation of XBOS, tied together by the Building Profile and query language. The query language enables discovery of metadata and devices, so that users of the API can adapt to the deployment target.

A solid example of the expressive power of the XBOS API is the integrated building dashboard app (not shown here). The XBOS building dashboard presents a modern alternative to the standard building management system. The dashboard makes full-building energy management and configuration of the building's schedules immediately available from the main page. Each of the building subsystems is also represented in the dashboard; the HVAC, lighting, electrical and spatial subsystems are all traversable from the main page of the dashboard, allowing a user to "drill-down" from a high-level system view to the status of individual devices.

The XBOS building dashboard is effectively a self-contained building management system that is dynamically created *entirely from the Building Profile*. It does not make any assumptions about the construction of the building or the underlying hardware, but rather uses the relationships defined by the building profile to handle building administration from commissioning to maintenance.

9. SECURITY

Many commercial BMS systems make the (sometimes valid) assumption that a building management system will be deployed within some trusted domain on a private network, isolated from the public internet and from unauthorized users. XBOS can operate on the same premise, but also provides transport security. The simple plugin infrastructure for the XBOS kernel means the system can leverage HTTPS (e.g. for sMAP drivers) or AES encryption (e.g. for embedded systems).

Each stream of data corresponding to a point (UUID) in XBOS is allocated a unique API key, and any process attempting to publish data to that stream or change the metadata for that stream must be in posession of that API key. For actuatable points, a special subscription is initiated through the XBOS kernel to a specific stream that also has an API key. API keys can mark streams as public or private for reading. XBOS manages these API keys and enforces the security policies over them, leaving the creation and management of user accounts and ACLs over the points to the application layer.

10. EVALUATION

Table 1 provides a framework for a comparative analysis of XBOS relative to prior designs. Our metrics of evaluation are drawn directly from the six established dimensions of the BOS design space.

HPL: XBOS substantially reduces the demands placed on devices to support the HPL. Tridium requires the purchase and installation of a Java-based JACE controller for connectivity. BAS and BOSS require the client to implement JSON over HTTP or to have a proxy do so on its behalf. The BuildingDepot2 HPL is the Data Connector layer, which requires devices to submit HTTP POST requests to the system if they do not expose their own API, with the additional requirement of handling proper authentication. Both present serious problems in receiving notifications through a NAT or firewall, severely complicating deployment and recovery. Agent-based peer-to-peer solutions, such as BEMOSS, overcome NAT restrictions, but require both a Python and ZeroMQ installation on every device. These place unnecessary restrictions on computational resources for the emerging class of embedded IoT devices. In constrast, the XBOS HPL only requires a process with network connectivity to the XBOS Kernel that has the ability to send key-value pairs using one of many lean application transports.

Canonical Metadata: Sensor Andrew only defines sufficient metadata to describe data types on sensor channels, similar to sMAP timeseries properties or Project Haystack sensor tags. BEMOSS does not provide contextual metadata above a basic HPL. Tridium uses the oBIX [15] standard, which is a semantically complete, but cannot cleanly evolve in real-time. BuildingDepot2 is able to model and link between building subsystems but these links are predetermined by building templates, which make certain non-scalable assumptions such as "assuming that [an] HVAC zone contains only one end-space" [2]. The XBOS metadata encapsulates canonical descriptions of all building subsystems – spatial, eletrical, lighting, HVAC, administrative, etc – and defines a methodology for including new subsystems. This gives XBOS the ability to support applications that operate entirely on the canonical metadata and building profile, rather than depending on out-of-band information.

Control Process Management: SensorAct deploys lightweight nonblocking Lua "tasklets" for periodic or event-driven actions; these processes are deployed over a resource-intensive Java Runtime Environment and require explicit lists of input and output streams, and thus cannot adapt to changes in the building. BAS and BOSS describe controller inputs using a fuzzy query language that adapts to changes in building infrastructure, but still require a Python environment and specific client library, which limits where control processes can be deployed. BEMOSS does not provide a mechanism for control beyond actuation of explicitly designated points, and even this requires a full VOLTTRON agent for each process. Tridium's Niagra AX framework is heavily architected and non-portable, allowing purportedly rapid but ultimately limited development. Conversely, framework-agnostic XBOS control processes are written against continuous views of the building and can naturally adapt to changes in the available set of sensors and transducers.

Building Evolution Management: BeMOSS has no model of the building so it cannot evolve, meaning all apps must be hardwired to the set of devices in a building. BAS and BOSS query on functional relationships, which are mostly independent of changes to the composition of a building, but are implemented over the sMAP archiver which does not offer continuously evaluated subscriptions, requiring manual intervention on behalf of the control process to check for building changes. BuildingDepot2 uses building templates to designate hierarchies, and traverses these hierarchies explicitly in applications, making it likely that apps will require some rewriting if the nature of those hierarchies change. The templates are versioned, but because applications must target specific versions, they are not fully agnostic to the building. In XBOS, all interactions with the building operate in a continuously updated context of the building, and can adapt naturally to such changes. Additionally, this mechanism can be used to "revert" a building to an earlier version, so applications can operate over the series of configuration changes.

Security: Tridium, BeMOSS, BuildingDepot and Sensor Andrew all combine ACLs with user accounts. BAS and BuildingDepot2 take the approach that each application must be approved by the building manager in order to run (although BuildingDepot2 also supports authorization of individual actuation points), and thus do not scale for larger buildings. In BOSS, applications receive permissions instead of users, and applications are restricted by location, value and schedule. SensorAct manifests fine-grained, guard-rule permissions in middleware, but at significant performance cost. XBOS supports fine-grained read/write control to individual actuators and sensors through a collection of API keys, and defers user accounts and ACLs to applications implemented over the XBOS API.

User Experience (UX): BEMOSS simply provides a flat list of devices and does not identify the context of those devices in relation to the building. Other solutions such as Sensor Andrew's Sense View and BuildingDepot2's BuildingViz require manual construction of flashy graphical representations of buildings that also fail to capture immediately useful information across multiple building subsystems. The XBOS building dashboard offers a integrated, full-building view that is completely autogenerated from the building profile and gives users the option of exploring subsystems in detail. Because the UI is constructed from the building profile, the dashboard dynamically reflects any building evolution.

Classes of Intended Deployments: As seen in Table 1, only a handful of proposed building operating systems are appropriate for multiple classes of buildings. There are many other solutions that touch on the space of building management, e.g. BuildingIQ for analytics or SensorAct for device connectivity, but do not address the larger problem of holistic building management. For example, BEMOSS claims to be a "platform for optimizing electricity usage and implementing demand response" [2], which are concerns that require a building management system to already be in place. Commercial BMS systems, including the recent Tridium system, are not appropriate for small-scale installations in small-medium commercial buildings or residential homes because of the amount of required physical infrastructure and financial cost.

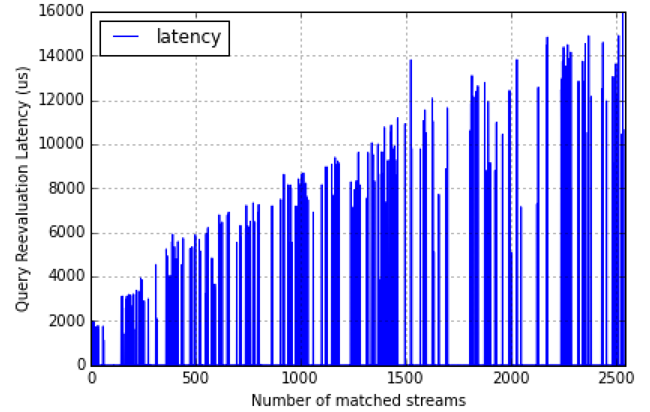

Figure 5: Continuous view re-evaluation latency during accelerated building evolution. Over 75% of building evolutions did not require reevaluation, and at peak, the system averaged 16ms per change (amortized 88μs per query over 180 zone controllers)

Continuous Views: Figure 5 illustrates the performance of the XBOS Kernel when re-evaluating continuous views for the accelerated evolution of a hypothetical 10 floor, 200 zone, 500 room building. Following the sample evolution from Section 3, one thermostat is introduced for each zone. A zone controller is introduced for each floor that subscribes to all temperature sensors before adding a temperature sensor to each room. Then, a new controller is installed for each HVAC zone that subscribes to all occupancy sensors, followed by the installation of an occupancy sensor in each room. The system scales linearly with the number of matched streams. During the accelerated evolution, the XBOS Kernel averaged 1% CPU usage, and 284 MB of memory (Unique Resident Set Size).

11. DEPLOYMENT: CIEE

XBOS has been fully deployed in a 7000 square foot office building and has been used by the occupants to control the building for 10 months. The building was retrofitted with 2 different models of networked thermostat, 4 models of networked light, 2 models of smart plug strips components and a 28-node sensor network.

As an anecdote, despite the lack of an individual dedicated to monitoring the health of the building, an occupant was able to identify, replicate and report anomalous behavior in the building by using the building dashboard to verify in real-time that although a thermostat was calling for heat, the temperature was still dropping. She attached a screenshot of the dashboard demonstrating behavior (Figure 6) in an email, drastically shortening the administrative overhead of verifying the problem, and leading to the discovery of a broken RTU damper.

An earlier version of XBOS was originally deployed at the site, but the progressive nature of the building retrofit identified problems with how the system handled removing or

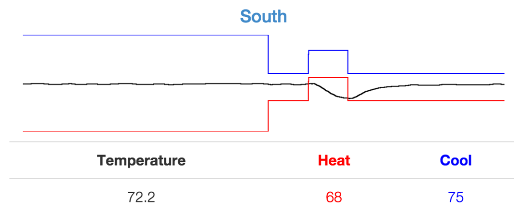

Figure 6: When the heating setpoint was raised above the current temperature, the thermostat called for heating, but the temperature decreased.

even moving devices. Hardcoded control loops needed to be changed as new sensors were added, and replacing the old thermostats with new meant that the building metadata needed to progressively adapt to new HVAC zones, which complicated the development of the building dashboard. The building was later re-retrofitted with XBOS; the use of continuous views to render the dashboard and write the control logic removed must of the manual effort previously associated with bringing new devices online.

The deployment, partially illustrated in Figure 4, ran on a small, on-premises server with a cloud-hosted timeseries database.

12. CONCLUSION

This paper presents the design of XBOS, an extensible building operating system, as influenced across the 6 dimensions of BOS design by the need to gracefully handle the natural lifecycle of a building and its subsystems.

The XBOS Building Profile captures a family of relationships across building subsystem hierarchies that functions as a canonical metadata for multiple classes of buildings and provides a descriptive basis for the system. All components of the XBOS design are tied together by a query processor embedded in a robust kernel that materializes continuous views of the building constructed in an ad-hoc manner by applications and control processes.

XBOS provides an extensible platform for studying further aspects of building management systems including appropriate security models and advanced controls leveraging ubiquitous sensing capabilities.

13. ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under grant CPS-1239552 and the California Energy Commission under grant PIR-12-026 as well as industrial partners including Takenaka Corporation and Intel Corporation.

14. REFERENCES

[1] Go programming language. http://golang.org/.

[2] B. Akyol and J. Haack. Conceptual architecture of building energy management open source software (bemoss).

[3] P. Arjunan, N. Batra, H. Choi, A. Singh, P. Singh, and M. B. Srivastava. Sensoract: a privacy and security aware federated middleware for building management. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 80–87. ACM, 2012.

[4] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte, et al. Energyplus: creating a new-generation building energy simulation program. *Energy and buildings*, 33(4):319–331, 2001.

[5] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 197–210. ACM, 2010.

[6] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. E. Culler. BOSS: Building Operating System Services. In *NSDI*, volume 13, pages 443–458, 2013.

[7] H. Friedman, D. Claridge, C. Toole, M. Frank, K. Heinemeier, K. Crossman, E. Crowe, and D. Choinière. *Annex 47 Report 3: Commissioning Cost-Benefit and Persistence of Savings*. Energy Conservation in Buildings and Community (ECBCS) Program, 2010.

[8] P. Fritzson and V. Engelson. Modelica - a unified object-oriented language for system modeling and simulation. In *ECOOP '98 Object-Oriented Programming*, pages 67–90. Springer, 1998.

[9] Green building xml. http://gbxml.org/.

[10] ISO. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries. ISO 16739, International Organization for Standardization, Geneva, Switzerland, 2013.

[11] A. Krioukov, G. Fierro, N. Kitaev, and D. Culler. Building application stack (BAS). In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 72–79. ACM, 2012.

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

[13] Project Haystack Corporation. Project Haystack. http://project-haystack.org/, 2015.

[14] T. Samad and B. Frank. Leveraging the web: A universal framework for building automation. In *American Control Conference, 2007. ACC'07*, pages 4382–4387. IEEE, 2007.

[15] O. Standard. Open building information exchange (obix) 1.0, 2006.

[16] T. Weng, A. Nwokafor, and Y. Agarwal. Buildingdepot 2.0: An integrated management system for building analysis and control. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 1–8. ACM, 2013.