Poster Abstract: 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 Building Profile, a canonical, executable description of a building and its subsystems that changes with a building and evolves control processes and applications accordingly. The design and implementation of XBOS addresses the six dimensions of an effective BAS – hardware presentation layer, canonical metadata, control process management, building evolution management, security, scalable UX and API.

Categories and Subject Descriptors

C.3 [Special-Purpose and Application-Based Systems]: Process Control Systems

Keywords

Building Operating System; Service Composition

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, 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.

We present the principled design of a new, extensible building operating system, XBOS, based on six primary dimensions of the building operating system (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 [1]. Addressing this challenge impacts every aspect of BOS design.

2. SIX DIMENSIONS OF BOS DESIGN

In this section we briefly describe the six dimensions of BOS design and establish state of the art solutions in current BOS implementations. A high-level mapping of the state of the art relative to these dimensions is provided in Table 1.

	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: A blank cell means that the BOS does not address the feature, a -means problem is not sufficiently addressed, a ✓is a standard solution, and a +pushes the state of the art

Hardware Presentation Layer (HPL): Traditional building management system (BMS) solutions are vertically integrated despite using open protocols such as BACnet, often overlaying proprietary networks to connect to vendor-specific control code. Most BOS solutions establish uniform read/write access to data sources in a building, typically as RESTful web services or agents, via a standard library or framework installed on gateways and servers in the building.

Canonical Metadata: Traditional BMS typically use proprietary, opaque descriptors on individual points internally and SCADA tags with naming conventions to capture attributes of a point, such as type, location, and function. Recent academic efforts standardize the representation of buildings and their subsystems through attributes and links [2], but make non-generalizable assumptions about structure.

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.

Building Evolution Management: The classical lifecycle of buildings incorporates changes in physical structure, com-

Permission to make digital or hard copies of part or all 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. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s). Copyright is held by the owner/author(s).

BuildSys'15, November 4–5, 2015, Seoul, South Korea..

ACM 978-1-4503-3981-0/15/11.

DOI: <http://dx.doi.org/10.1145/2821650.2830311>.

ponents and usage; these present a challenge for traditional BMS whose composition is wedded to the particular set of physical components in a building. While these lifecycle changes to the building system are not frequent, they are essential, a BOS should be designed around handling these changes automatically.

Security: Most BMS use user accounts for authorization, with access control lists implemented in the equivalent of an overlay network.

3. 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*. Standards such as oBIX, Project Haystack and IFC focus on how to create a static graph of devices and building components, but are not designed to programmatically adapt to building evolution.

We have found that a strict relational model prevents the system from realizing emerging relationships required by controllers but that are not reflected in the schema. Enforcing a specific family of relationships limits the applicability of a schema across classes and instances of buildings. The XBOS Building Profile separates the descriptions of points from the binding of relationships between those points. Points are described by a set of metadata keys which are drawn from a well-defined namespace. Example keys include `Metadata/HVAC/Zone` for an HVAC Zone and `Metadata/Sensor/Measure` for identifying the target quantity for a sensor. The XBOS query language operates over these keys to capture ad-hoc relations between collections of points, either as one-time results or as continuously evaluated relational views.

Building evolution in XBOS is possible because XBOS operates on a family of relationships not captured in any table. *Each continuous view establishes its own foreign-key relationships on demand* rather than operating over a strict, static schema that prevents the representation of the building from evolving. These relationships offer a well-formed abstraction for implementing monitoring and control processes, and inform the design of the XBOS architecture.

Continuous views are rendered by the XBOS Kernel. When a continuous query is executed by a client, the query initially behaves like a traditional execute-once query, but then continues to update the client 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.

4. 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 1.

Devices: at the bottom of Figure 1 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 describe devices according to the XBOS Building Profile and represent them with established, canonical de-

vice interfaces. Drivers are not bound to a particular client implementation, only requiring a simple socket to publish timeseries data to the kernel and receive actuations and subscribed data over a family of lean application transports.

XBOS Kernel: the central component of XBOS. The Kernel is the data historian for both metadata and timeseries data and contains a high-performance, multiprotocol pub-sub broker. The Kernel query processor continuously renders foreign-key relationships between points without introducing substantial overhead. All connections between XBOS components take place via the XBOS Kernel.

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. Framework-agnostic services are written against continuous views and can naturally adapt to changes in the available set of sensors and actuators.

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.

Figure 1: XBOS system architecture

5. ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under grants CPS-1239552 and CPS-0931843 and the California Energy Commission under grant PIR-12-026 as well as industrial partners including Takenaka Corporation and Intel Corporation.

6. REFERENCES

- [1] 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.
- [2] 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.