Open Software-Architecture for Building Monitoring and Control

Carl Blumstein*, David Culler‡, Gabe Fierro‡, Therese Peffer* and Marco Pritoni+

*Berkeley Energy and Climate Institute, University of California, Berkeley
‡Electrical Engineering and Computer Science, University of California, Berkeley
+Mechanical and Aerospace Engineering, University of California, Davis

Abstract

Information technology can increase energy efficiency by improving the control of energy-using devices and systems. Awareness of this potential is not new—ideas for applications of information technology for energy efficiency have been promoted for more than 20 years. But much of the potential gain from the application of information technology has not yet been realized. Today a combination of new requirements for the operation of the electricity system and the development of new technology has the potential to cause a rapid increase in the pace of adoption of improved controls. In this paper we discuss one promising avenue for technology advancement. First, we review some basic concepts with emphasis on open software-architecture. Then we describe the components of XBOS, a realization of this open software-architecture. XBOS has the ability to monitor and control many different sensors and devices using both wired and wireless communication and a variety of communication protocols. Finally, we illustrate the capabilities of XBOS with examples from an XBOS installation in a small commercial office building in Berkeley California.

Keywords

building controls, building automation, open software-architecture

Introduction

Information technology can increase energy efficiency by improving the control of energy-using devices and systems. Awareness of this potential is not new—ideas for applications of information technology for energy efficiency have been promoted for more than 20 years. But much of the potential gain from the application of information technology has not yet been realized. In an earlier paper one of the authors (Blumstein (2011)) discussed some reasons for the slow exploitation of information technology's potential to increase energy efficiency. The earlier paper also suggested that a combination of new requirements for the operation of the

electricity system and the development of new technology could cause a rapid increase in the pace of adoption.

This paper is about *open software-architecture*[1] for the control of energy use in buildings. Open software-architecture is a way of organizing the software that links together the physical elements of a building control system to allow the addition of other systems or components. The reason we are concerned about open software-architecture is that open software-architecture is the key to creating an environment that supports innovation. Proprietary and closed systems, which are prevalent today, typically create barriers to innovation.

To make this clear, consider a commercial building with a control system for its Heating Ventilating and Air Conditioning (HVAC) system. If you want to control the lighting in the building, the technology currently used for HVAC control cannot easily be modified for lighting control—in practice you need to add a completely separate control system for lighting. Further, if the control system for lighting includes occupancy sensors and you want to use occupancy to control HVAC, you cannot, as a practical matter, use the lighting system's occupancy sensors. Still further, if you develop new software for detecting faults in the HVAC system, you cannot easily install the new software in the existing building control software. These are all problems that can be solved with open software-architecture.

We will have more to say about how we addressed these problems in a small commercial office building in Berkeley California later in this paper. First we discuss in more detail the idea of open software-architecture, drawing on lessons from the Internet.

Lessons from the Internet

The most important lesson from the Internet is interoperability—the ability of the Internet to accommodate diverse devices and systems and enable them to work together. The practical effect of interoperability is that equipment suppliers and software developers can compete to supply established needs and can innovate to create new uses. This environment has fostered both cost reductions and rapid innovation. So, one may well ask, can we make building monitoring and control systems look like the Internet? The answer is, yes we can.

Doing this is facilitated by using the Internet's open architecture and protocol stack. The critical step is to move from a vertical to a horizontal architecture—an essential element of open architecture. Figure 1 provides a simplified representation of horizontal layered architecture[2] to help explain the concept. Each layer is independent, and thus creates modularity. The bottom layer in Figure 1, here called

[1] Readers should be careful to distinguish between open software-architecture and open-source software. Open software-architecture does not necessarily involve open-source software.

[2] The phrase "layered architecture" does not refer to spatial relationships among the system's components; rather, it refers to logical relationships. The "layers" are an abstraction. Here we are using the word "layers" as a heuristic; it has more specialized meanings in other contexts.

the hardware presentation layer, is where the control system connects to the physical environment. This layer interconnects sensors and actuators to the other layers through software adaptors called drivers. The middle layer—system services—organizes, stores, and transmits data from the hardware presentaion layer

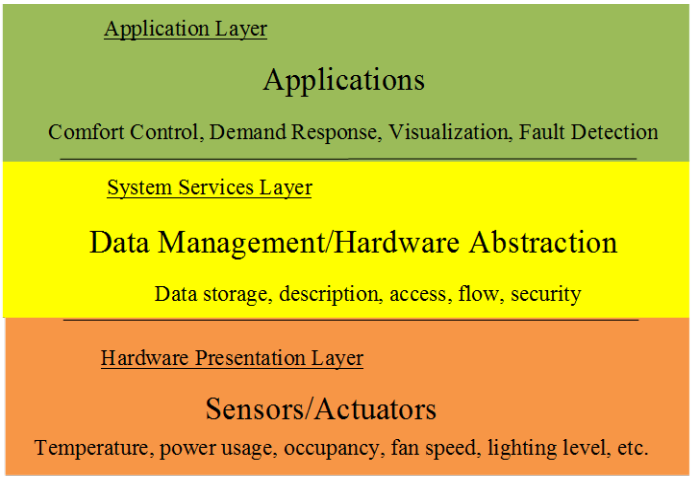

Figure 1 A simplified representation of layered architecture for building monitoring and control

and instructions from the application layer. The top layer, here called the application layer, has software applications that operate on data provided by the system services to produce outputs in the form of information (e.g., a dashboard) on the state of the building and instructions for the control of building systems.

Not all control is initiated on the application layer; some happens autonomously on the sensor/actuator layer—for example, lights might be directly controlled by an occupancy sensor. And not all instructions from the application layer are accepted. For example, a smoke alarm may override an instruction to open a damper. To make this more concrete, consider a building appropriately equipped with sensors, actuators, and applications. Suppose that the operator of the building wishes to minimize energy use during the peak time on a hot day by precooling the building so it can ride through the peak time. An application in the application layer contains a model of the building that can predict the best time to turn on the air conditioning based on the outdoor temperature, the indoor temperature, the weather forecast, and other variables all of which are resident in a database in the system services layer. The application gets the data from the database and predicts the best time to turn on the chillers, say, 7:00AM. If sensors and controllers in the hardware presentation layer determine that operation is safe, the chillers will be turned on at 7:00AM.

The difference between horizontal and vertical architecture is not in the functions that need to be performed. Sensing and actuating, data management and applications need to happen in monitoring and control systems regardless of the architecture. The difference is in the separation of these functions. In a vertical

system a "black box" might, for example, have hard-wired connections to sensors and actuators and have applications with built-in data structures that were inaccessible to other applications. Horizontal layered architecture can keep the functions from becoming entangled and allow devices and software from different suppliers to interoperate.

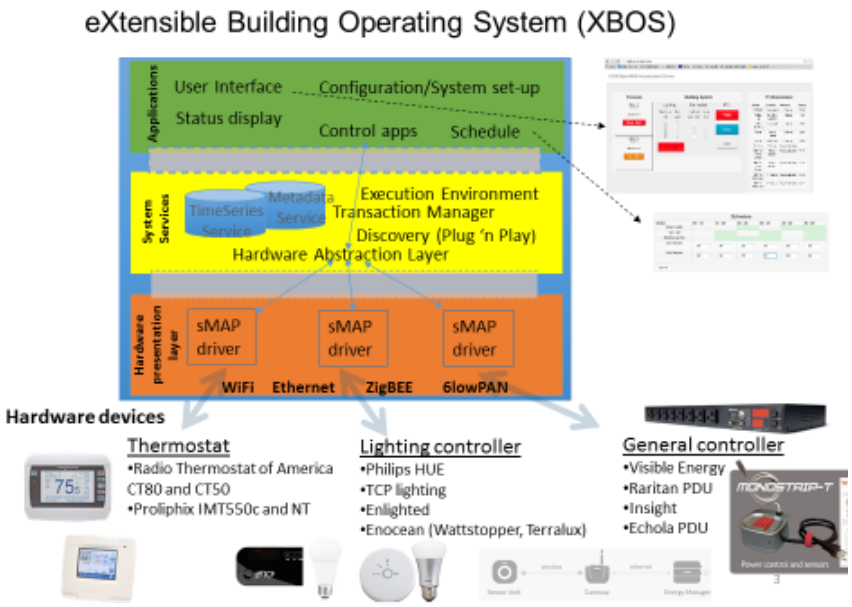

Figure 2: The XBOS layered architecture, consisting of Hardware devices, a Hardware Presentation Layer, System Services layer, and an Application layer. Drivers in the Hardware Preentation Layer present vendor supplied hardware devices (sensors and actuators) to the Hardware Abstraction Layer as canonical devices (for example, as a generic thermostat.

Open Building Control Architecture—the XBOS Example

Figure 2 illustrates the eXtensible Building Operating System (XBOS) a realization of an open software-architecture control system. This program has evolved from several years of work at UC Berkeley (see Dawson-Haggerty *et al.* (2013)). A more detailed description of XBOS can be found in Fierro (2015).

XBOS is built on UC Berkeley's simple Measurement and Actuaction Profile (sMAP), an open-source information infrastructure for buildings and grids. The Hardware Presentation Layer, shown in Figure 2, includes drivers for network thermostats, lighting control, and general control, along with more than fifty other open source drivers[3] for energy metering, demand response notification, BACnet,

[3] in https://github.com/SoftwareDefinedBuildings/smap/tree/master/python/smap/drivers

Modbus, commercial BMS systems, weather metering, thermal monitoring, air quality monitoring, and so on.

While control systems based on open software-architecture are much more versatile than vertically architected systems, open software-architecture systems do present some challenges. An example is the need of recognizing new devices connected to the system. XBOS handles this problem with a discovery service, which automatically detects new devices on the network, finds and installs the appropriate sMAP driver, and configures it to that particular installation. This discovery service is similar to "plug and play" as it appears in various forms in consumer markets, such as plugging in a new hardware device into a PC or laptop. However, it does not depend on vendor products implementing a particular discovery standard. The discovery service receives notification of the presence of a new device, it probes the device using a collection of detection scripts to identify what it is; once identified, the service pulls in the appropriate driver for the device, creates the configuration file integrating the driver and the particular site, and connects the device to the system.

Vertically architected systems do not require a discovery service because, by design, they do not interact with initially unknown devices. Other challenges for open software-architecture are discussed in Fierro (2015)

Installation of XBOS in the Offices in the Berkeley Kress Building

The research team developed a pilot test of the XBOS platform in a commercial building in Berkeley, California that was built in 1935. The 700 square meter top floor of the building holds private offices, open plan office space, a kitchen, and a conference room; a server room for the office is on the mezzanine level. The heating, cooling and ventilation system is provided by five packaged roof-top units each controlled by programmable thermostats. The overhead fluorescent lighting system is generally controlled by wall switches, with occupancy sensors in the private offices; some corridor lighting remains on permanently.

The XBOS platform installation consisted of a miniature computer (FitPC), Ethernet switch, and Wireless Access Point; this provided the means for all control systems and sensors—no matter what network protocol—to communicate with the computer. For example, some equipment required proprietary gateways (such as for ZigBee devices such as the Enlighted lighting controller and the Rainforest power meter); other equipment used simple USB dongles (EnOcean lighting controllers communicating at 902 MHz and the environmental sensors using 802.15.4). The computer held the sMAP drivers for communicating with all sensors and controllers, and the database, archiver, and services (such as discovery described above). A total of five smart thermostats, three different lighting controllers, two general controllers, 14 sets of indoor environmental sensors and one power meter were controlled by XBOS.

XBOS's integrated control system provides several advantages over vertical solutions. The unified interface greatly improves user interaction as the building

manager can access all the devices from a single dashboard. It is possible to develop such an interface, because data from heterogeneous devices is made uniform by the hardware presentation layer. Devices can be organized and grouped in different ways. For instance one can easily access all the lighting in the building or all the systems (HVAC, lighting, plug loads and additional sensors) in a room. This simple feature is not currently available in vertically integrated and isolated systems, since each vendor uses separate intefaces that require distinct logins. It is almost impossible to exchange data between commercial interfaces. In addition, XBOS develops a building-level scheduler to enforce policies on the whole building. For instance, when a calendar event occurs, all the devices are notified, synchronized and their setpoints adjusted. The ability of accessing external sensors allows creating customized control schemes. For example one can control the HVAC system based on temperature measurements in rooms that are actually occupied.

One example of application that takes advantage of the rich sensor environment in this field test is shown in Figure 3. Smart meter electric data is combined with thermostat runtime and building operation hours to disaggregate energy use as a function of building activity. The program distinguishes the baseload energy use (standby power and constant operation of all the devices), the activity-based energy

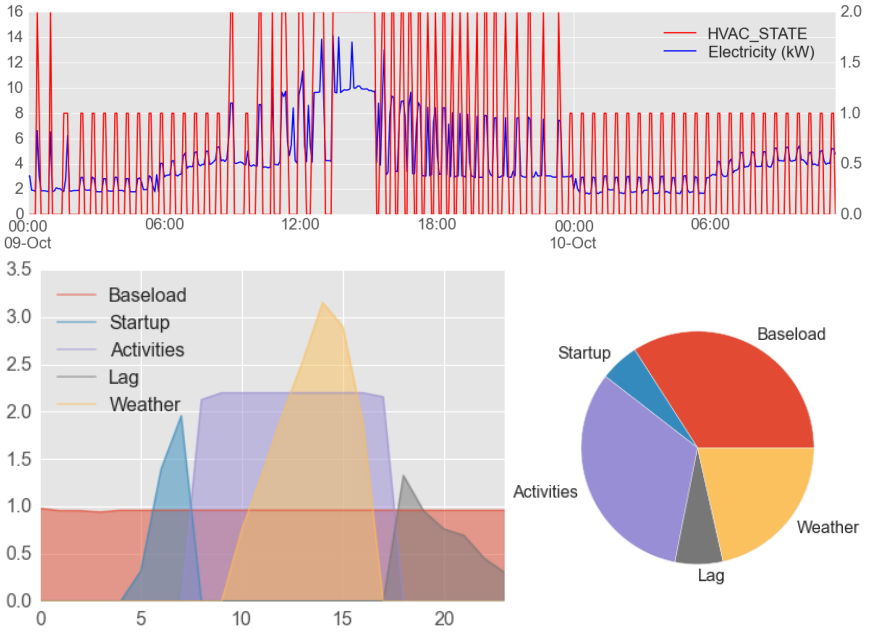

Figure 3: XBOS Energy Disaggregation Application. In the top graph, note the coincidence of spikes in KW consumption (left scale, blue line) and the HVAC state (right scale, red line). In this instance just one roof-top unit is operating. Small spikes occur when the fan comes on; larger spikes occur when both the fan and the compressor come on.

use (extracted using opening hours), the startup and lag energy use (before and after business time and above the baseload) and the weather-related energy use (calculated associating HVAC runtime with smart meter data).

Another key features of XBOS is that it operates on a 'canonical representation' of a building, a generalized description of functional relationships between buildings components. Because of this feature, new applications can be easily written, without knowing the details of the different devices and their communication protocols. The Hardware Abstraction Layer provides a common interface to devices like thermostats or lighting controllers. This type of architecture, commonly used in information technology (for example in Android smart phones), has not yet been adopted in building systems.

Conclusion

We believe there is a compelling case for building controls based on open software-architecture. As we have noted, the practical effect of the interoperability that open software-architecture can provide is that equipment suppliers and software developers can compete to supply established needs and can innovate to create new uses. This can foster both cost reductions and rapid innovation. However, there are significant impediments to the widespread adoption of this technology. Once they make a sale, purveyors of proprietary vertically-integrated control systems have a captive customer. Because only the original seller can perform maintenance and provide product upgrades, this customer lock-in is highly profitable. Companies with established products are not enthusiastic about changing their business models.

It is possible that there is more opportunity for open software-architecture in smaller buildings where the dominant controls companies do not have a large presence. This is part of the motivation for targeting XBOS at the small commercial building market. However, this strategy is not likely to succeed if the work is done in isolation. The XBOS software is open source[4] and we are hopeful that others will want to use it and continue its development.

Acknowledgment

The authors drew extensively from the work of the brilliant UC Berkeley graduate students in the LoCal and Software Defined Buildings research groups, namely Michael Andersen, Stephen Dawson-Haggerty, Andrew Krioukov, Jorge Ortiz, and Jay Taneja, as well as others. This work was supported by the National Science Foundation and the Department of Energy.

References

Blumstein, Carl. (2011). Energy Efficiency, Information Technology, and the Electricity System. *Proc. 2011 Conference on Energy Efficiency*, European Council for an Energy-Efficient Economy, Paris, France.

Dawson-Haggerty, Stephen, Krioukov, Andrew, Taneja, Jay, Karandikar, Sagar, Fierro, Gabe, Kitaev, Nikita, and Culler, David. (2013). BOSS: Building Operating

[4] https://github.com/SoftwareDefinedBuildings/XBOS

System Services. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*.

Fierro, Gabe et al. (2015) XBOS: An Extensible Building Operating System, in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments* (to appear) November 3-4, Seoul, South Korea