Life Is On | Schneider Electric

Smart Buildings: A Framework for Assessing the "Openness" of a Building Management System (BMS)

White Paper 501

Version 1

by Jeffrey Bowman
Chris Megede
Wendy Torell

Executive summary

As more building owners strive for "smart" buildings, the term "open" is often described as a requirement for their building management system (BMS). However, that term by itself is ambiguous and confusing, and can lead to a system selection that doesn't address present and future business needs. We believe there are 3 layers that must be understood in discussing the topic of open BMSs, which help avoid the confusion and allow for clear discussions on the topic – (1) Data acquisition/sharing, (2) System integration, and (3) Building orchestration. In this paper, we define this framework, clarify terminology, and discuss the key criteria associated with being open including how they influence the complexity and performance of the BMS. Example use cases for each are presented.

RATE THIS PAPER ★★★★★

Introduction

Building management systems (BMS) have historically served the functions of controlling and allowing the operation of building heating, ventilation, & air conditioning (HVAC) systems. But as societal pressures grow to improve the efficiency, sustainability, productivity, and tenant experience of buildings, the role of the BMS must also grow. To accomplish this, owners must interconnect their building systems that have traditionally been siloed, integrate internet of things (IoT) devices and 3rd party applications, and ultimately automate and optimize their entire building operations.

Schneider Electric's White Paper 500, Three Essential Elements of Next Generation Building Management Systems (BMS), discusses why traditional BMSs were not well positioned to grow their role, and describes the necessary attributes of a next generation BMS that serves as a smart building system platform for proactive monitoring, control, and automation. It discusses the ambition of many building owners to evolve their building(s) into smart buildings, and how the advancement of information technology (IT), IoT, and "smart building" technologies have enabled this vision of a building that:

- minimizes energy use by leveraging grid services and artificial intelligence (AI)
- orchestrates HVAC, security/safety, lighting, and occupancy systems (and others) together for more productive and satisfied tenants
- is automated to run in the most sustainable and reliable way possible

The term "open" is used by vendors as an attribute of a BMS and used by owners as a requirement for their BMS, but there is oftentimes confusion and ambiguity around the term, since **the industry lacks a standard definition for what it means to be open**. The generic definition of an "Open System" is this – a system that "provides some combination of interoperability, portability, and open software standards"[1] But with this broad definition, there is much left to interpretation. Note, in this paper, we don't cover "open source" which refers to freely available and modifiable software.

Many times, when people speak about open BMSs, the conversation is around technology capabilities (i.e. protocol compatibility), but sometimes the conversation is around how easy it is to commission and maintain the system, and yet others are referring to its ability to be interconnected with other building systems. Having these varying definitions makes it very challenging to have an intelligent conversation, which can lead to a system selection that doesn't address present and future business needs.

In this paper, we propose a framework, illustrated in **Figure 1**, for discussing the topic of open BMSs, so that building decision makers can assess a system's capabilities, have informed discussions about where it falls on the spectrum, understand the complexities and implications of being open, and ultimately make a more informed system selection based on the desired outcome of the building.

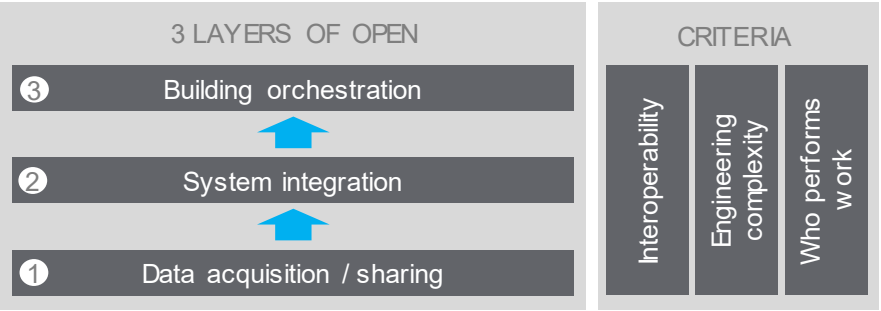


Figure 1
Framework for assessing "open" BMSs

[1] https://en.wikipedia.org/wiki/Open_system_(computing)

Open is a complex topic. Every building is unique and has different objectives. In this framework, we define three distinct layers, each with their own degree of openness. Each layer presents decision makers with varying expectations and challenges, and each layer builds from the previous layer. In other words, the capabilities from Layer 1 are pre-requisites for achieving the capabilities of Layer 2, and the capabilities of Layer 2 are pre-requisites for achieving the capabilities of Layer 3 (i.e., the ambition of a "smart building"). To help you determine how open a system is at each of the layers, we've identified three criteria:

- **Interoperability** – the ability for one part of the system to work with another part of the system, or one system working with another system. The more open the system is, the more data that can be shared and the more complex interoperability becomes.
- **Engineering complexity** – how difficult it is to achieve the interoperability. The amount of customization and programming required is a key consideration.
- **Who performs work** – does the work require skilled specialists? Is it limited to certified personnel? The requirement for specialized individuals and skillsets to perform the work is an important factor.

These criteria often pose tradeoffs. For example, you may be able to achieve a highly open system in terms of interoperability by sacrificing engineering simplicity.

In the next section, we'll clarify terminology and establish definitions, then go into each layer in the three sections that follow. For each, we'll walk through use-cases, and describe how the criteria influence the performance of the system in the context of being open. These use cases, all based on a hospital example, will demonstrate how the layers build on each other to achieve the end goal of a smart-orchestrated building.

Terminology

A number of terms that arise in BMS discussions are misinterpreted or misused. Therefore, before we describe the three layers, it's important to clarify the essential terms required for describing BMSs and what it means to be open.

When we describe a **BMS system**, we are referring to the comprehensive system made up of hardware and software, including the following main parts.

- **I/O (sensors, relays and actuators)**[2] – Sensors provide input on anything you want to know about the environment (examples include temperature, occupancy, pressure); Relays are electrically operated switches which are used to enable and disable equipment (i.e. start and stop a fan, turn off and on a light, etc.). Actuators are devices that move a mechanism that impacts the environment (i.e. opens a damper, adjusts a valve, etc.).
- **Field controllers** – These are programmable devices that provide the physical connections to the sensors and actuators and run the local control logic. These also send and receive building-level information to and from building controllers or other field controllers to influence local control logic to cause an action (i.e. close a damper).
- **Building controllers** – Sometimes referred to as building servers, these are devices that aggregate and coordinate the activities of multiple field controllers

[2] There's a generally accepted hierarchy to some of these devices starting at the bottom with I/O devices, followed by field controllers, and building controllers at the top.

and/or other building controllers. Coordination is done through its own control logic. The building controller may also serve the UI of the BMS.

- **Cloud Infrastructure** – Cloud technology is used to provide long term historical data storage, analytics and/or, to expose and interact with 3rd party apps.
- **Client applications (UI)** – This is the user interface to build, maintain, and operate the system. It may be a PC-based application, web-based application, or mobile application, and fulfill the functions of programming, as well as provide views into the system via graphics and dashboards.

When the topic of open is discussed, the focus often centers around communication protocols. But again, there is often confusion around related terms. For instance, a protocol may be open or closed. It may also be standard or proprietary. But words like open and standard are often interchanged, and words like closed and proprietary are as well. In fact, it is possible to have a standard and open protocol. It is also possible to have a proprietary and open protocol. The definitions below clarify these terms, as well as how application programming interfaces (API) are different than a protocol.

- **Protocols** – a communication protocol is a system of rules that allow two or more entities to transmit information. These rules define things like syntax and semantics[3]. In practice, this allows two devices to communicate with each other. Some are referred to as OT (operations technology) protocols, some as information technology (IT) protocols, and others as IoT protocols. White Paper 500, *Three Essential Elements of Next Generation Building Management Systems (BMS)*, describes each. All three generally exist in a smart building.
- **Open protocol** – An open protocol allows different vendors' equipment to interoperate without the need for a proprietary interface or gateway[4]. They talk the same language and no translation is needed. LonWorks and BACnet are examples of open protocols.
- **Closed protocol** – A closed protocol is one that is proprietary AND not open to communication with other vendor's products without an interface or gateway. In some cases, vendors may choose to create a closed protocol in order to enable vendor-specific functionality between two devices by that vendor.
- **Standard protocol** – A protocol established and controlled by a recognized third-party standards body to be an industry-accepted way of communicating, that is open to anyone. For example, BACnet is a standard (and open) OT protocol driven by ASHRAE, and TCP/IP is a standard (and open) IT protocol driven by IEEE.
- **Proprietary protocol** – A proprietary protocol is a communications protocol owned by a single organization or individual[5]; N2 Open, for example, is a proprietary (yet open) OT protocol owned and maintained by Johnson Controls.
- **Application programming interface (API)** – A computing interface that defines interactions between multiple software intermediaries. APIs simplify programming by abstracting the underlying implementation and only exposing objects or actions the software developer needs[6].

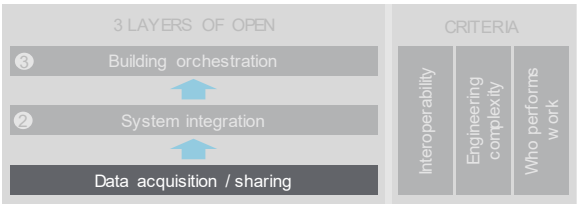
[3] https://en.wikipedia.org/wiki/Communication_protocol

[4] A gateway is, in simple terms, a translator between multiple protocols; it could be hardware or software.

[5] https://en.wikipedia.org/wiki/Proprietary_protocol

[6] https://en.wikipedia.org/wiki/API

Layer 1: Data acquisition / sharing



Layer 1, data acquisition and sharing, is the critical foundation of "open" for a BMS. BMSs control building systems, typically, the HVAC system of a building. Within this conventional scope of a single system, the BMS must be capable of receiving and sending data (control signals) to and from **sensors/actuators**, and other **building controllers**. Specifically, this creates the following four possible data flows (illustrated in **Figure 2**):

1. Sensors and actuators sharing/receiving data with controllers
2. Sensors sharing/receiving data with the cloud infrastructure
3. Controllers sharing/receiving data with other controllers
4. Controllers sharing/receving data with the cloud infrastructure

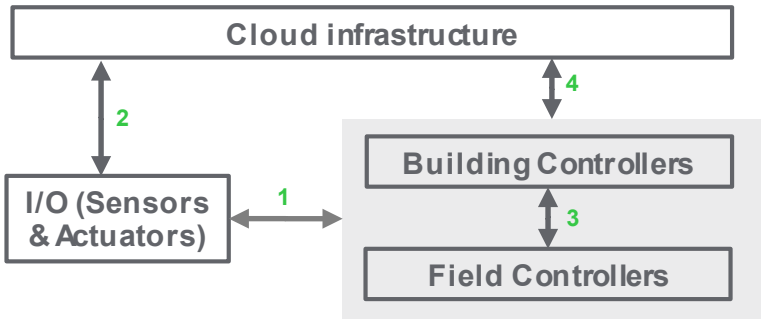


Figure 2
The four data flows of data acquisition/sharing in Layer 1

Each of these input and output data flows fall under one of three methods of communication via a wire using voltage/current/resistance signals, OT protocols, or IT and IoT protocols. Below we describe how sensors/actuators and controllers utilize these communication methods.

Sensors & actuators – Traditionally, sensors and actuators have been the most generic and easiest to integrate within the BMS platform. These are one-way devices that share or receive their data to/from a controller. They primarily communicate via simple electrical signaling over a wire (i.e. voltage signal, current signal, resistance signal). This may be a simple sensor for pressure, temperature, occupancy, CO_2, etc., or an actuator that opens a valve, or a drive that adjusts a fan speed.

Smart sensors and actuators are now becoming more common because they offer several benefits including: (1) the ability to consolidate multiple sensors onto a single box, which reduces the number of connection points/wires and associated labor, (2) wireless communication which reduces engineering cost and installation, and (3) cloud connectivity which reduces capital expense for on-premise infrastructure. But with smart devices comes an increased need for integration. They may be capable of bi-directional communication, where they can sense and control multiple values at once. Depending on the features of the device and the communication protocols it supports, integration can be more or less complex. This is where the concept of "open" first comes into play.

Controllers – Unlike sensors and actuators where data flow can exist without protocols (i.e. simple signal over a wire), for controller to controller communication, there will always be protocols involved. Historically these have been closed, proprietary systems with little or no interaction between vendors. More recently these have

been “opened up” with protocols like BACnet and LonWorks. Just because a controller uses an open protocol, however, does not mean the controller is interoperable. There are complexities and degrees of openness to consider. We discuss this in the next subsection.

Use cases

Below we consider two use cases to help illustrate some of the complexities of sensor and controller integration. These use cases are based off the BMS system diagram shown in **Figure 3**.

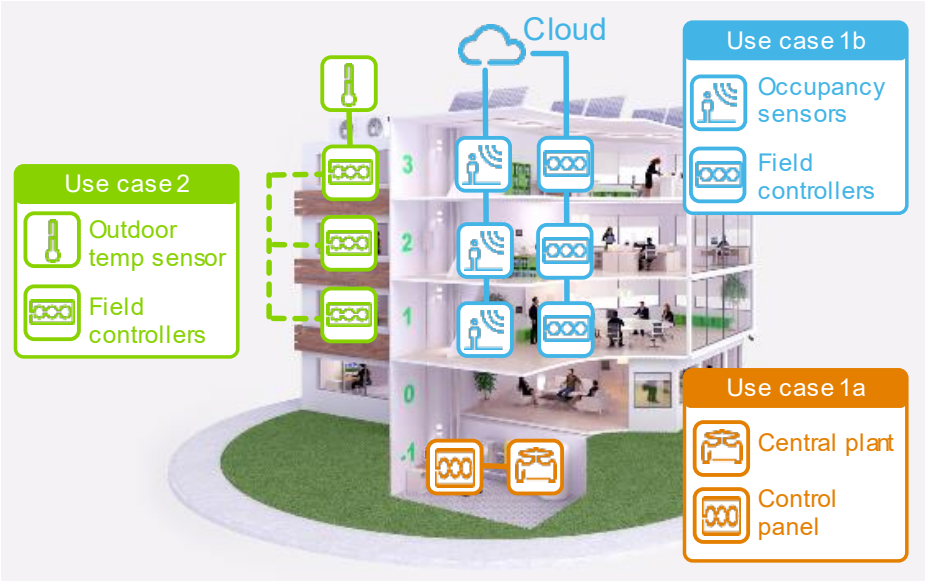


Figure 3
Data acquisition/sharing use case scenarios:

1a. on-premise sensors
1b. cloud sensors
2. controller to controller

Use case 1: Integration with communicating sensors
Sensors/actuators may need to communicate data directly to on-premise devices (1a); but sometimes they need to communicate data via the Cloud (1b).

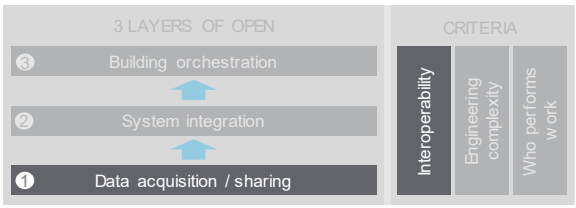
- On-premise – I want to be able to control my central chiller plant that has a variety of sensors. My plant includes smart chillers, and a variety of pumps with valves and sensors/actuators associated with it. For this use case, we need to control the variable speed drive for the pumps which includes control and system status and energy consumption monitoring.

- Via cloud – IoT sensors that connect to the Cloud are becoming increasingly common. I want to have the target temperature in my space react to occupancy changes. For this use case I need to connect my IoT occupancy sensor (which exposes an API) to my BMS as an input into my temperature control logic.

Use case 2: Integration between controllers
In order to control costs, I only have a single outdoor air temperature sensor that I would like to share as an input across all of my controllers that require this value. I would like the controller that has this sensor physically connected, to share this value directly with the other controllers on the network.

With these use cases, we can now assess the openness of a BMS as it pertains to Layer 1. To assess how “open” a BMS is, we must consider the three criteria introduced in **Figure 1** – (1) interoperability; (2) engineering complexity; and (3) who performs the work. Note that these criteria cut across all “3 layers of open”.

Open criterion 1 – Interoperability



Looking at use case 1, the sensors are designed to measure and record/report its values to the BMS system. But what good is a sensor if the system can't read its data? Not all sensors communicate in the same way, so it is important to understand the capabilities of the sensor(s) you want to deploy, to ensure it communicates in a way that is compatible with the target BMS. On-premise sensors generally communicate with OT protocols, like BACnet, LonWorks, Modbus, OPC, and KNX. Newer cloud-based sensors often support new IT protocols like Bluetooth and MQTT and/or communicate with an API.

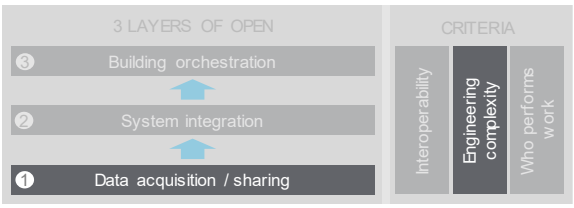
The OT protocols are supported by most BMSs, but for the newer IT protocols like in use case 1b (cloud sensors), there may be limited or no support. If this is the case, this may require a gateway or some other add-on to the BMS like middleware or custom code to enable data transmission.

In use case 2, the integration of controllers to each other introduces additional concerns for interoperability. There are two requirements here:

- **Both controllers use the same protocol** – As with sensors, the protocols used and supported by the controllers and the BMS dictate the ability to integrate. The OT protocols are the most likely options, with BACnet as the standard choice.
- **The data point(s) needed are exposed** – While the OT protocols are open and interoperable, ensuring that the right data is shared can be a challenge. If the manufacturer of the device has chosen to expose the needed data values, then other controllers speaking the same language will have access to that data.

As an example, in use case 2, where data is passed from one field controller to another (which is more efficient than bringing the data centrally), this can only happen if the two controllers are speaking the same OT protocol and capable of sharing the exact data needed. If the two criteria above are not true, the controllers cannot talk directly to each other, and the alternative is to move the data into the building controller directly and then distribute the value(s) from there. This is less efficient and requires that the building controller can act as the gateway between the different field controllers. If the data needed is only exposed in a proprietary way, then the only options are to replace your building controller with one that supports that proprietary standard, or to write a custom driver if the vendor exposes the protocol definition (which is unlikely).

Open criterion 2 – Level of difficulty / engineering complexity



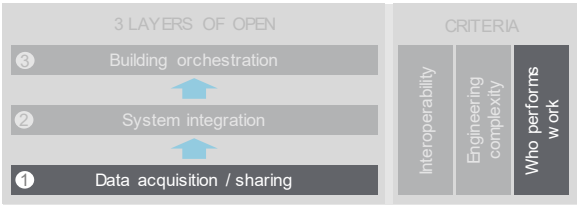
A consideration in the openness of a BMS is the level of difficulty or engineering complexity associated with making the sensors and/or controllers interoperable with the system. There are several actions necessary during integration and setup of your BMS that may be more or less complex and time consuming, depending on the level of openness. This applies to all three use cases described above.

Point discovery and mapping
On one end of the spectrum, you could have auto discovery, where the server sends out a broadcast to see what new devices are on the network, and automatically sees any new sensors or controllers (i.e. BACnet). On the other end of the spectrum, when there is no auto discovery, the complexity and manual nature of the point discovery and mapping increases. You would have to first specify the address of the controller (i.e. Modbus), then you'd have to manually look up the registers in the device specifications to find the specific register number(s) (addresses) for the desired data point(s).

Controller and System programming
BMS programming involves writing control loops, configuring alarms, building graphics and setting up schedules among other things. There are 3 controller programming variants: fixed function, configurable and freely programmable. In some cases, a fixed function device my be easier to integrate since it may not need specialized tooling for configuration. However, these devices will not satisfy all use cases.

A BMS that supports multiple protocols natively requires less engineering complexity than one that doesn't. With multiple protocols supported, points from different field controllers can interoperate during configuration setup without the need for additional hardware/software and interact with each other on graphical use interfaces (GUIs). When only one protocol is supported, allowing points from a field controller that speaks a different protocol requires a bridge or gateway to convert the protocol. That may mean an extra computer to make it run, plus the task of manual programming/mapping which is tedious and time consuming. In addition to these increased labor costs, there are some unexpected or "hidden" costs such as licensing and maintenance labor over time.

Open criterion 3 – Who performs work



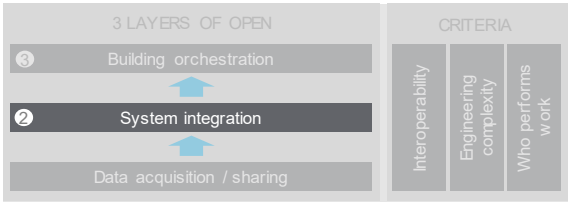
The third open criterion to consider is whether anyone can do the integration work or are specially-trained individuals necessary. You may have a difficult time operating your BMS if you need to call a third party every time you want to make a change. While it is generally expected that the control logic tasks in **Table 1** will require some level of specialized skills/resources, the day-to-day tasks shown should not.

The more custom work needed to accomplish the required tasks, the more "locked in" a user may feel, since they are dependent on the expert(s) to continue utilizing the system. It comes back to the previous discussion of protocols and exposing the necessary data. When those don't occur, more custom work is necessary, and therefore, more specialized knowledge as well. When specialized skills are necessary, well-trained individuals with a history of doing the work helps ensure successful implementation.

Control logic	Day-to-day operations
• Initial set up of HVAC controls in a new building – likely requires knowledge of the design, configuration, and programming BMS tools and someone who understands the specs of the components being integrated into the BMS. • Replacing a sensor/actuator – If it's a simple resistance-based temperature sensor, this could be a simple re-wiring at the I/O on the controller by a technician, but if it speaks its own protocol, it may require some software reconfiguration to update the connection to the new device. • Adding a new zone controller and exposing needed data point(s) – likely requires the same experts as the initial set up, to discover it, add to the control loop, etc.	• Modifying a set point like a room temperature – Whether it be at the controller/thermostat on the wall or within the GUI. • Creating trend logs and alarms to see what is happening over time and correlate data – This shouldn't require 3rd party expertise to create the trend logs, alarms, and link them to data points. • Updating schedules (i.e. daily, holiday, seasonal) – This should not require any highly trained person.

Table 1
Control logic and day-to-day operations tasks; The expectation for an open system is that internal staff can fulfill day-to-day operations

Layer 2: System integration



Up until now, we've spoken about integration or connectivity with finite components like sensors and controllers that have finite data points. As the BMS scope expands beyond HVAC control, the BMS must integrate with other systems, such as a lighting system that has its own underlying complexity of controllers and sensors (lighting sensors, actuator to adjust blinds, etc.). A further level of integration is needed to achieve this that goes beyond what we addressed in Layer 1. The BMS must be able to consume data in a different way, it must be able to push data to 3rd party systems and query data from those systems. It also may need to tie in with analytics services that may communicate to a proprietary cloud.

This requires us to think differently about the term "open". Whereas with Layer 1, openness was centered around the protocols and exposed data, here we must expand the scope and abstract further out in the ecosystem to include other methods of sharing data between the BMS and 3rd party system(s). Two use cases for Layer 2 are described below and illustrated in **Figure 4**.

Use case 1: Integration with other building systems
A hospital building wants to tie their admission/discharge/transfer system to their BMS. This would enable them to make smart decisions about when to set patient rooms into setback control. By knowing when a space is vacated, the hospital can save money on its energy bill.

Use case 2: Integration of an AI/Analytics system
The hospital's building manager wants to apply a fault detection and diagnostics system to the BMS. This system requires data from all aspects of the BMS, from the sensors, to the inputs, the outputs, and the internal logic. This system uses this data to run a series of algorithms in order to determine if the system is running efficiently, if there are any faults, or if there is any degraded comfort, and if so, help diagnose this problem and point to potential causes.

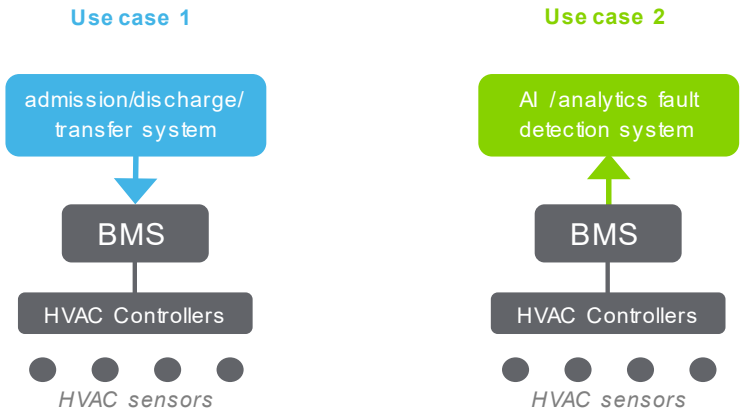
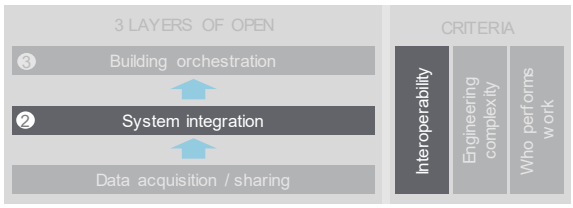


Figure 4
System integration use case scenarios

Open criterion 1 – Interoperability



At this level of system-to-system integration, things are generally less "out-of-the-box", compared to sensor and control integration. Key topics to consider that impact interoperability include (1) the use of APIs vs files/databases, (2) Semantics/ontologies, (3) type of data needed, and (4) authentication.

Use of APIs vs files/databases vs. OT protocols
While it is possible to integrate some systems together with OT protocols (and this could be the preferred approach since it doesn't require specialized software engineers to do custom coding), more than likely you will run into REST APIs[7] at this layer. If a vendor wants their system to be "open", and pass data along to another system, data must be exposed through either APIs or OT protocols.

In the example use case 1 above, the vendor of the hospital admission/discharge/transfer system creates the API that the BMS uses. In use case 2, the BMS vendor would create the API for the AI/analytics system to use.

Some systems which are less open, however, don't actively try to expose the data. They may have their data stored away and shared through .csv files or directly to a database that are abstracted away from the systems that want to consume that data. These schemas can make it very difficult and confusing to understand the data structure when you're not the one that wrote the database. The more complex the database schema, the more time it will take, increasing costs of integration. When selecting systems to integrate, look for ones that expose data through APIs or protocols.

Semantics/ontologies
Semantic tagging (labeling of things) and ontologies (relationships of those things to each other) are critical to interoperability. Consider the analytics use case, where we want to integrate AI. This requires that the AI system has a good understanding of the context of the points within the BMS. If that analytics system wants to discover room temperature set points for the building, it may be difficult to discover relevant points and figure out how each data point relates to the building, such as what

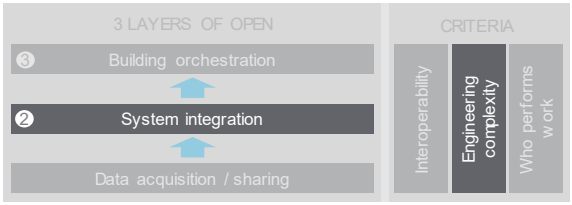
[7] https://www.redhat.com/en/topics/api/what-is-a-rest-api

room(s) they are in, and what equipment they are feeding. Without standard semantics and ontologies, this requires a human to interpret an individual building's point naming convention (i.e. location, controlled equipment), map everything, understand the hierarchy, etc. Exposing data in a standard way can avoid this manual work. There are different semantic standards such as Brick and Haystack, but this is still evolving for many vendors. In some cases, there are data standards for specific industries (i.e. HL7 for healthcare) that make it easier to understand the data schema. When applicable, look for systems that follow these standards.

Types of data needed
Different types of data (i.e. historical data like trend logs, real-time data, alarming data) may be collected in different systems, which may impact your ability to access the data. While you may have access to live data, you may not have access to historic data which may be collected in a separate historian or even archived. In the context of open, it comes down to the system having methods to access the varying types of data.

Authentication
Another important consideration with regard to interoperability from system-to-system is authentication[8] mechanisms, as these can be a limiting factor and increase the complexity of the integration. APIs pass unsecure data, so when authentication is used, middleware often exists. It is hard to genericize authentication. It can't be done very easily through a UI and is oftentimes very obscure, requiring code to be written for some of it. There are standard authentication protocols that could simplify this process and reduce custom coding (i.e. OAuth), but they are not always used. In the context of open, look for a system that supports non-obscure authentication types like OAuth, as well as allows custom development of middleware to access 3rd parties which use proprietary authentication.

Open criterion 2 – Level of difficulty / engineering complexity



When standard protocols are not used at this layer (which is commonly the case), custom development becomes necessary to connect the systems together. This custom development can require only a day in some cases, but months in other cases. As a general rule, the less open a system is, the more engineering time it will take to integrate. Engineering complexity is minimized when the systems use best practices like web services, semantics, and ontologies.

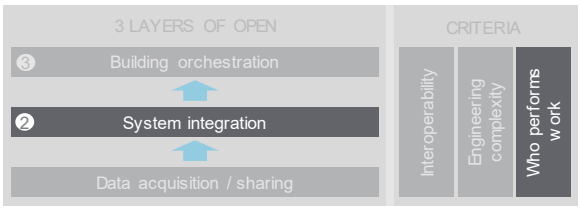
In order to simplify facility operations, a user interface is often desired that pulls in information from the integrated systems (i.e. the hospital admissions/discharge system and the BMS from use case 1). Integrating video, web pages, programs, set points, and values into a single dashboard ("single pain of glass") adds to the engineering complexity. If there's not a straight-forward way to customize or modify the UI to display the different types of data streams, this limits the usefulness of the integration. A system that doesn't support this type of customization could be considered less open.

Another layer of complexity is added when you want a system to have 3rd party control over another system. For example, in use case 2, if we want the analytics system to control a valve in the HVAC system, changes may be necessary to the BMS's

[8] https://en.wikipedia.org/wiki/Authentication

control loop logic to allow for 3rd party input, which could require custom work. A system is more open if the work required to accomplish this is minimized.

Open criterion 3 – Who performs work

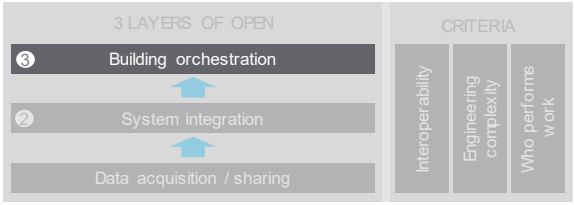


The distinction between control logic tasks and day-to-day tasks that we discussed in Layer 1 also applies here. When 'mapping' is not automatic, there needs to be a consideration of who does this 'mapping'. The person who does this needs to understand both domains, so that the data can be mapped accurately. In some cases, knowing the ins and outs of a specific building may drive who must do this work, and could require a team of people.

Often, the BMS vendor has a team that specializes in integrations with 3rd party systems. If the vendor doesn't have this, the building owner would need to hire a 3rd party to write the integration. Finding and hiring a consulting firm, that then has a learning curve to understand the systems and the building, all adds time and cost. Access to the right documentation, licenses, and technical support can further complicate the learning curve.

In the AI/analytics use case above, the 3rd party analytics system vendor may have to write middleware to extract and map the data and the BMS vendor would require additional work on the BMS to allow for third party control. The complexity of the middleware may lead to more specialized people involved in the integration, which may be viewed as less open.

Layer 3: Building orchestration



Layer 3, building orchestration, involves complete coordination across systems. Think of it as Layer 2, but at scale. The ambition of a smart building requires going past the system-to-system integration of Layer 2 and expanding the scope of the integration to all building systems. This orchestration helps buildings optimize their energy efficiencies, operational efficiencies, tenant well-being, and tenant productivity by streamlining and automating complex building tasks. There should always be clear objectives of the integration, to ensure tangible value, vs. integration for the sake of integration.

We believe the BMS will either play the role as the "orchestrator" or allow itself to be integrated as a subsystem into an orchestration layer. (The latter scenario becomes a Layer 2 discussion, as its focus is on system-to-system integration.) In the role as building orchestrator, the BMS will not only be responsible for managing the HVAC control of the building, but be able to ingest data, apply context, and manipulate and apply workflows to any other system within the building, in any combination, at scale. A single cohesive system is necessary to achieve a sustainable, resilient, efficient, and people-centric building.

Use case: A smart hospital
The hospital has the ambition of being a "smart hospital". It has countless different systems, that need to interoperate and share data to ensure patient satisfaction, comfort, and safety while maintaining an energy efficient building. Consider some of the systems utilized in a hospital, as illustrated in **Figure 5**.

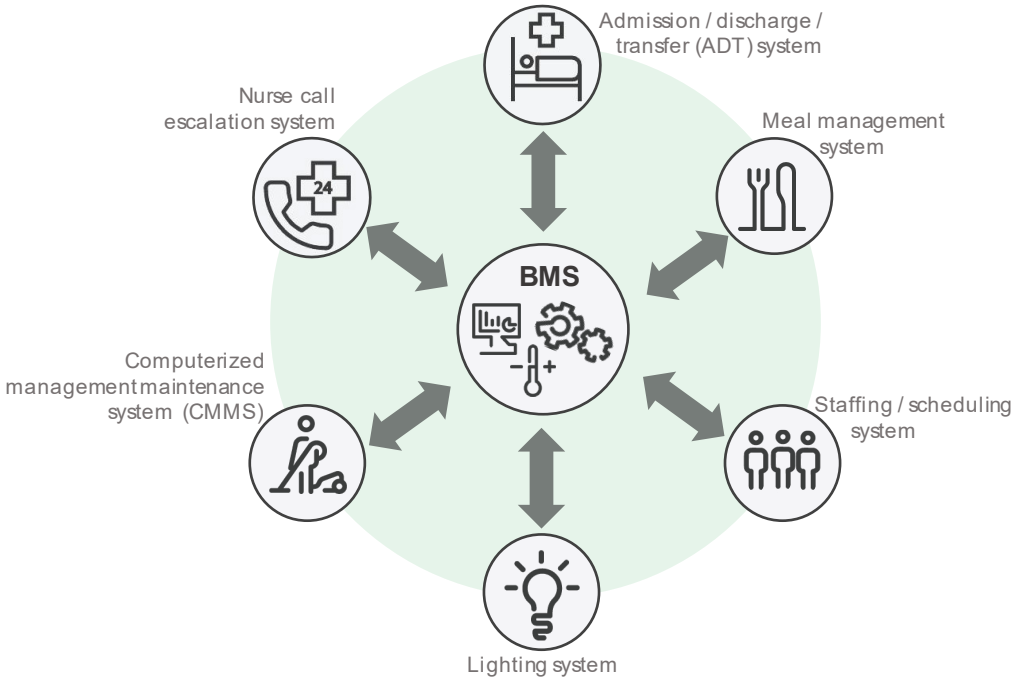
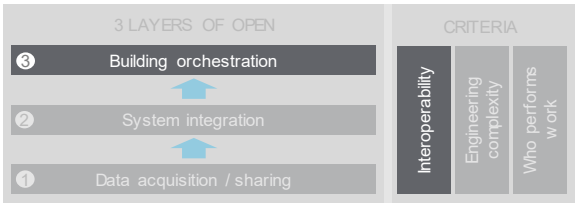


Figure 5
Hospital use case of building orchestration

As a subset of this smart hospital, consider this example. If I know when patients are going to be scheduled for a procedure, and I know when the patient is being moved from the procedure to recovery, then I can adjust the setpoints in that patient's room to reduce energy while they are out of the room, and then readjust the setpoints in the room while they are in recovery so that the room is conditioned before they are returned. While that patient is out of the room, the maintenance staff could be automatically scheduled to go into the room to perform routine maintenance without disrupting the patient. All of this should be wrapped up into a single UI, so that the facility managers, nurses, doctors, office staff, and maintenance crews all have access to contextualized views based on the same core data.

Open criterion 1 – Interoperability

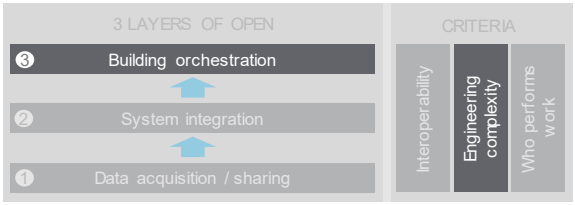


The topics discussed regarding interoperability in Layer 2 apply here as well. This includes (1) the use of APIs vs. files/databases, (2) semantics/ontologies, (3) the type of data needed, and (4) authentication. But, at this layer, it is more expected that the BMS will have more tools available to make integrations between the systems much simpler, whether it be UI-based or programming, or a combination of the two.

These integrations are generally bi-directional, so it's likely to include more complicated object types, whereas, at the other layers, it was less complex data such as *data values*. Here, you are more likely to encounter more complex data structures such as *time schedules*. With this more complex data structure, there is a greater likelihood of issues with semantic differences between systems, and possibly different functional capabilities that don't map directly from system-to-system. When this happens, custom rules and workflows may be necessary to fill in the gaps, which adds to the complexity. A more open BMS would have a toolset that supports these types of custom rules and workflows.

Open criterion 2 – Level of difficulty / engineering complexity

Accomplishing the objectives of Layer 3 is similar to those of Layer 2, but at a grander scale. This adds to the complexity and to the need for customization. There are four key considerations in this layer that impact the engineering complexity:



Presentation
In an open system, the orchestrator should be able to view the data from all the subsystems in a single-pane-of-glass. This view should be customizable and tailored to the expectations of the operators and tenants. It is also important that the user's dashboard have normalized data (i.e., same units of measurement, time scales, and so on) from the subsystems and calculate data based on rules and workflows in order to give the viewer actionable insights. For example, in the use case above, it would be helpful for the maintenance staff to have a visual dashboard so they can see at-a-glance what rooms in the hospital are available and in need of maintenance actions. The ability to create these dashboards in a simple way (i.e. drag & drop widgets, auto-creation) is important to avoid unnecessary programming/developing.

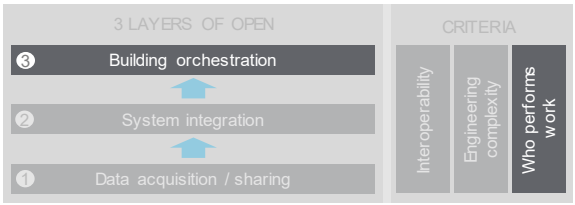
Point discovery/mapping
With an unspecified amount of systems that need to interoperate and understand each other, it's likely there will be some protocols and semantics differences between the systems (i.e. the admissions system in the hospital uses HL7 protocol, whereas the BMS uses BACnet). The orchestrator needs to understand these different "languages" and serve as the translator between the systems. This can be very complex to solve, and, in many cases, these translations require custom development or modeling that can only be performed by an expert. Doing these at scale likely means there is a very complex toolkit to allow these translations. In assessing the openness of a BMS, you should consider to what degree these integrations can occur through the UI. Only the most complex things should require entirely custom development, and even then, should ideally be done within the confines of the orchestrator, and not as a separate application that is then installed alongside the orchestrator. As the industry progresses forward, we expect to see more APIs available, making this easier.

Programming
With many systems all working together, the orchestrator should *allow for custom workflows*. These workflows take the existing conditions in the building from the various subsystems, enabling the actions that lead to the desired experiences and outcomes for the operators and tenants (such as in the hospital use case). In an open system, these workflows should be developed within the orchestrator's UI *without the need for entirely custom development*. Another important consideration is *how resilient the workflows are to changes throughout the lifecycle of the building*. In other words, if something is added/moved/deleted from a subsystem, how do you make sure the workflows adapt. A resilient, open system ensures the orchestrator is aware of the changes made and the new conditions can be handled without the need for human interaction.

APIs
The data used by the orchestrator to create workflows from an unspecified number of building subsystems is very valuable. In Layer 2, we explained the importance of APIs in achieving this. That data may *also* be useful to other types of systems, perhaps even outside the context of the building. A completely open system means that

this data should be available for other third-party applications to consume. For example, the data contained within the orchestrator is needed to create a mobile app; or a power utility wants the data to perform city-wide analytics. APIs allow this data to be exposed externally.

Open criterion 3 – Who performs work



As we discussed in Layers 1 and 2, when integration work moves further away from standard protocols, the need for a software programmer to perform building-specific customization becomes greater. Since, in Layer 3, we are dealing with a lot of subsystems being tied together, the probability of needing to integrate one that is not very "open" is much more likely, which increases the probability of software developer involvement, unless the UI and driver layers are incredibly sophisticated. Below are three examples of integration activities that can occur that require varying skills and expertise.

- **Set up workflows** – The workflows within the orchestrator should be configurable in most cases by the integrator[9] using a tool with a minimal amount of training. However, expect that in more complex workflows, custom functionality may require the help of a software developer.
- **Set up custom UI** – The UI within the orchestrator should be configurable, in most cases, by the integrator using a tool with a minimal amount of training. However, it is expected that in more complex UIs, there might be a need to extend the UI with custom functionality in order to create unique visuals or calculate values in the UI that requires the help of a software developer.
- **Connect to mobile and 3rd party applications** – The data that is exposed outside of the orchestrator through APIs should be descriptive enough for a 3rd party application integrator or software developer to interpret and utilize within their application without the need for expert support.

Conclusion

The role of the BMS is evolving. Its scope reaches far beyond the traditional function of HVAC control. It now must play a key role in reaching the ambition of a smart building.

"Open" is a commonly stated requirement during the discussion and selection of a BMS, but the industry lacks clear definitions, categorizations, and criteria to allow for intelligent conversations on this subject. In this paper, we introduced that missing framework. It consists of 3 layers: (1) data acquisition, (2) system integration, and (3) system orchestration.

Every building is unique and has different objectives. At each layer, there are varying expectations and challenges the system exhibits, and each layer builds from the previous layer. There are three key criteria for assessing how open a BMS is at each layer: (1) interoperability, (2) engineering complexity, and (3) who performs the work.

[9] An integrator is a person or company that specializes in bringing together component subsystems into a whole and ensuring that those subsystems function together (https://en.wikipedia.org/wiki/Systems_integrator)

Table 2 provides key takeaways regarding "open" in the context of each layer of the framework.

Table 2
Key takeaways, for each layer of the Open BMS framework

Layer	An "open" BMS should…
1 – Data acquisition / sharing	• be interoperable across multiple OT protocols. • support extension of native protocols, limiting the number of gateways required to communicate with sensors, actuators, and controllers. • not depend on "experts" for day-to-day operations.
2 – System integration	• have the ability to integrate system-to-system through any means required. • minimize the time required to integrate systems together. • simplify integration between systems with a vendor-provided team of experts.
3 – Building orchestration	• be able to implement custom rules and workflows, as the orchestrator. • be able to support customization with the presentation, point discovery/mapping, programming, and APIs. • allow for programmers to expand functionality with setting up workflows, setting up custom UIs, and connecting to third party applications.

About the authors

Jeffrey Bowman is a Senior Software Engineer in the CTO Office of Digital Buildings at Schneider Electric. He has over 10 years of experience in Schneider Electric's Digital Buildings group and is an Edison expert in integrating and aggregating different building systems together.

Chris Megede is a Software Architect in the CTO office at Schneider Electric. He has over 20 years of experience in Schneider Electric's Digital Buildings group where he has served as the Director of Global BMS architecture and as Product Manager leading IoT and cloud initiatives

Wendy Torell is a Senior Research Analyst at Schneider Electric's Science Center. In this role, she researches best practices in data center and building design and operation, publishes white papers & articles, and develops TradeOff Tools to help clients optimize the availability, efficiency, and capex/opex costs of their facilities. She also consults with clients on availability science approaches and design practices to help them meet their performance objectives. She received her bachelor's degree in Mechanical Engineering from Union College in Schenectady, NY and her MBA from University of Rhode Island. Wendy is an ASQ Certified Reliability Engineer.

RATE THIS PAPER ★★★★★

Resources

Three Essential Elements of Next Generation Building Management Systems (BMS)
White Paper 500

Browse all white papers
https://www.se.com/ww/en/download/

Browse all TradeOff Tools™
tools.apc.com

Note: Internet links can become obsolete over time. The referenced links were available at the time this paper was written but may no longer be available now.

Contact us

For feedback and comments about the content of this white paper:

Schneider Electric Science Center
dcsc@schneider-electric.com