Core principles of a Building Operating System

Main objectives

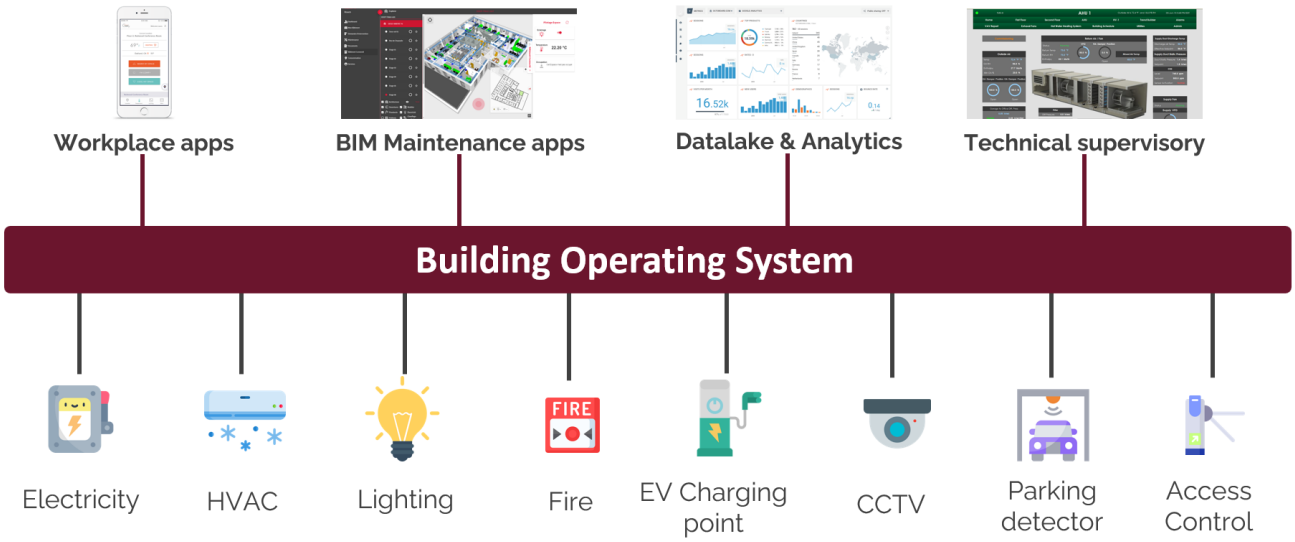
A building can be considered as a complex living structure which needs to provide comfort and safety to its occupants. In order to achieve its purpose, a building includes many different technical systems working more or less together. In the past, such systems would be self sufficient and would have no interactions. With the digital transformation and incoming new challenges like carbon footprint reduction or high standard air quality requirements, systems cannot work on their own anymore, they need to exchange their data to create more global behaviors.

The prop tech industry is seeking for data, there are many new players that want to make sure those challenges are efficiently met. These third parties want to provide their service to improve systems control, tenant experience, advanced analysis... They face the current building industry with multiple and heterogenous systems in buildings which doesn't make the task easy. This is primarily due to a missing part in many buildings: a digital core that can:

- Communicate with both regular OT systems (HVAC, Lighting, Metering...) and IT systems (Network, IoT, Access Control...)
- Transform technical data into intelligible pieces of information
- Create information exchanges between Systems and Services

That is the essence of a Building Operating System. It is a software platform integrating data from heterogeneous sources into a single unified interface to be shared with third parties.

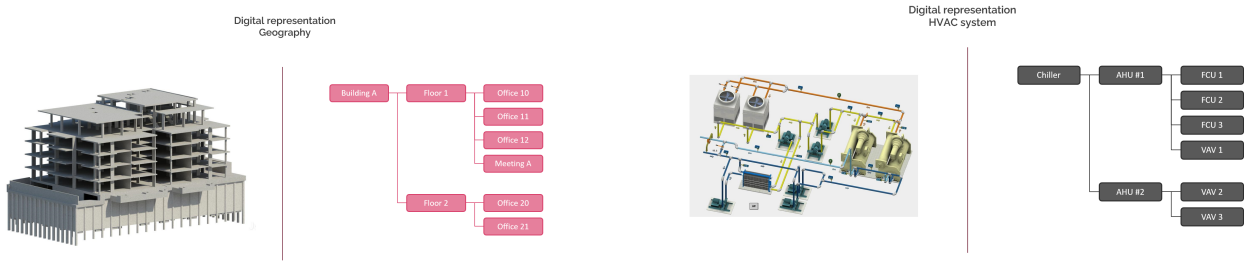
There are a few concepts and principles to understand the full extent of a Building Operating System.



Considering different types of data

Descriptive Data

If we take a physical representation of a building, a HVAC architecture or a submetering hierarchy, they all have something in common, they are made of **Descriptive data models** which is by definition information that doesn't change over time (apart from physical changes...). The descriptive data models are very useful to understand the relationships between different assets of a building (spaces, equipment...). They provide insights to help third parties (human or digital) to understand a building's philosophy and get over its complexity. Relationships between assets describe physical inclusions, fluid transfer, data exchanges, master-slave interactions... All together, they form different trees which are very useful to describe complex systems and furthermore allow a digital third party to learn a building autonomously.

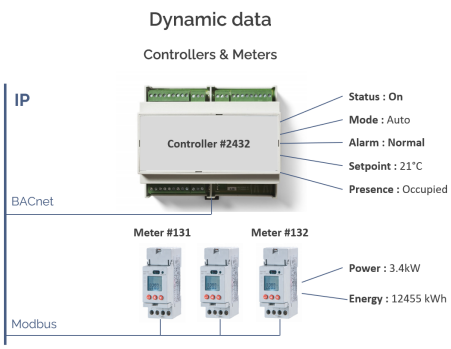


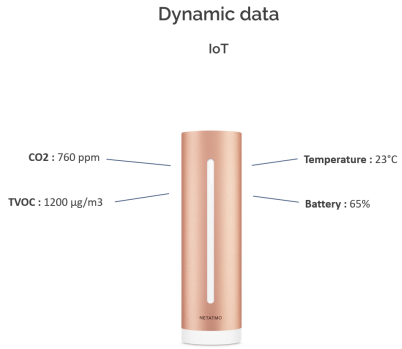
Live Data

In the meantime there are various systems in a building which produce or measure real-time data such as a temperature, a pressure, a valve position... **Live data** is a value varying over time. Live data is produced from BMS/BAS equipment (HVAC, lighting, meters...), from IoT devices (people counting, air quality...) and also from silos like access control, lifts or even from third party services that produce their own data. Live data can take many forms:

- An instant value called **datapoint** which represents a numeric variable, a boolean value...
- A series of values called **timeseries** or **trends** such as a weather temperature forecast, a logged pressure, a people count...
- An event such as a traditional BMS/BAS alarm, a CMMS work order, a meeting room booking...
- ....

A live data is usually acquired from a traditional BMS/BAS protocol (BACnet, Modbus...) or from IT systems (using APIs, connectors...)





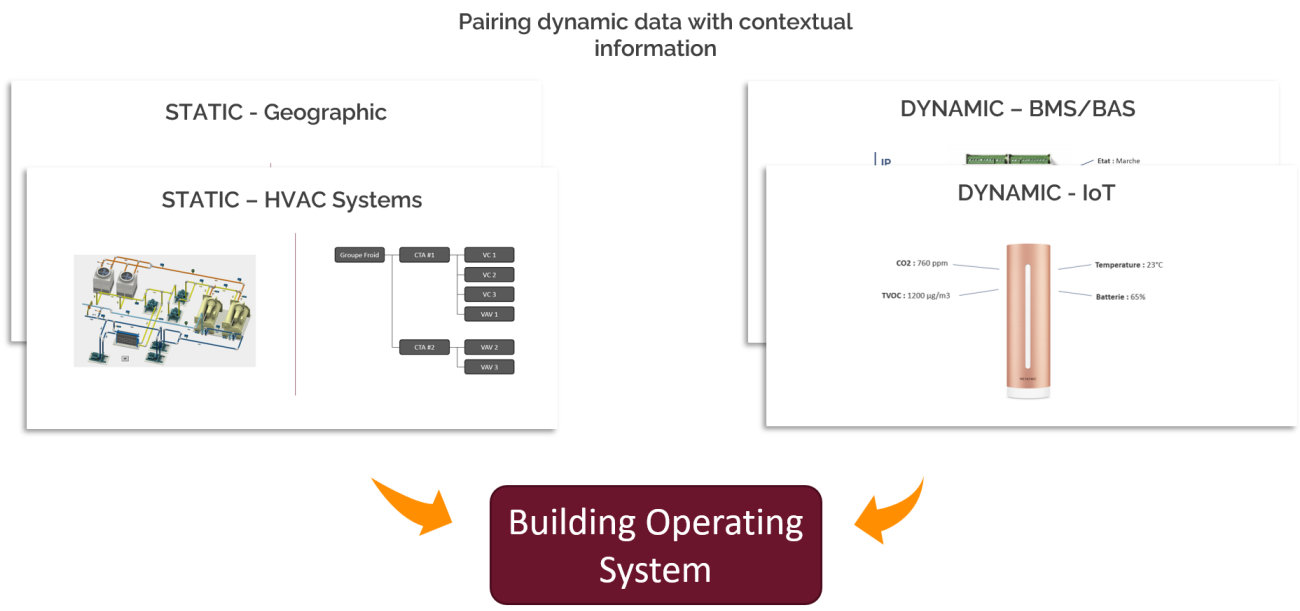
Pairing Descriptive & Live Data

Creating a context

A **Live data** has a very limited use on its own (it is just a value varying over time). Most of the traditional BMS/BAS systems apply a simple label to describe a live data (like "Temperature AHU 01") and displays it to a synoptic (a simplified schema of a system) with the single purpose to provide a technical interface for a Facility Manager. It was sufficient in the past. But, now, there are many applications out there looking for intelligible data from building systems to improve their service. And, it is really hard for a third party which collects data from a system like a BMS/BAS to automatically understand which equipment it is originated from, where is it located in a building, or what kind of interactions it has with other systems etc. **It is usually solved with a lot of manual engineering and guess**.

This is precisely why **Live Data are paired to Descriptive Data into a BOS.** Pairing a data point with some descriptive data generates a context, and from a simple value, it becomes an intelligible piece of information to be shared with third parties (analytics, tenant app, CMMS...). A Building Operating System is a core platform to handle this pairing process and to share these pieces of information to third parties. It simplifies data collect for services so they don't concentrate on how getting data from a building but on their core added value. It's very close to the revolution brought by any OS that handles hardware. App developers on a Smartphone can easily use a GPS information (transformed as the user position) without the need to decode raw data from the chip which may be different between various phone hardwares.

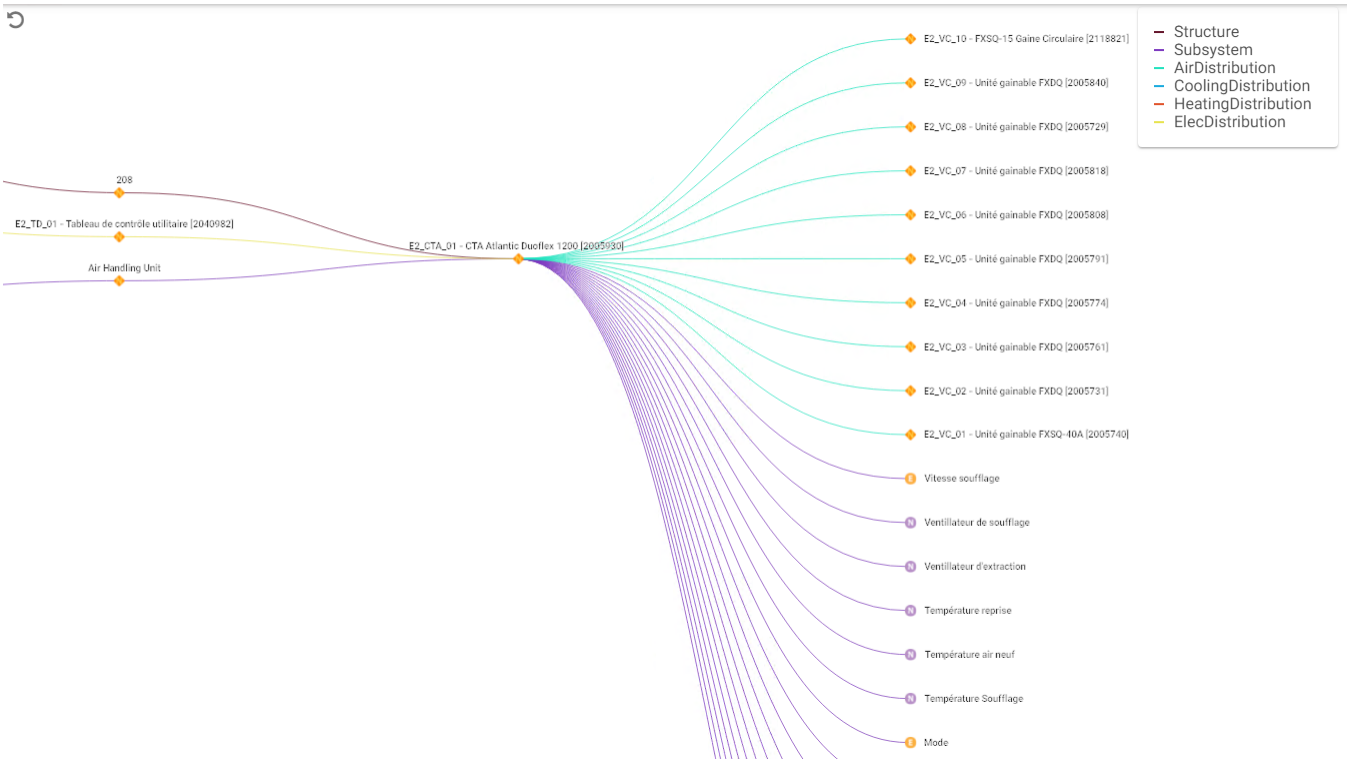
A third party can ask a BOS to provide information, not just raw data.



Sharing a context

Let's take a simple use case: a Boiler brings hot water to an AHU, which brings itself prepared air to terminal units, which deliver themselves warm air to their respective space. The question is very simple: what happens when the boiler is malfunctioning, which rooms are impacted? Is there any impact on the air quality? Is the number of people in the zone critical without recycled air? Apart from a human analysis behind a screen, there was no way to automatically create a chain of alerts up to tenants.

While pairing both descriptive & live data, a BOS platform creates a knowledge graph that provides a more easy way to represent complex systems such as buildings. A building is made of floors, spaces and equipment which produce a lot of data. There are different angles to represent interactions and relationships between assets. A global knowledge graph of a building allows any third parties to understand all the subsystems and assets being involved. Of course visualizing the graph on the BOS platform UI is not the most beneficial feature (since it requires a human interaction) but sharing this graph easily with third parties (API, Connectors...) is very valuable.



Hiding OT complexity

Hardware agnostic

What made the use of OS very popular was initially the availability to plug-in any devices and to be recognized by the machine (remember when you plug a mouse or a USB key into a computer, there are no drivers to install manually). A building is even more complex by the quantity of different systems it uses to provide comfort to its occupants, to ensure safety, to measure its impact on the environment. This is achieved using **many different technologies and hardwares** from many manufacturers.

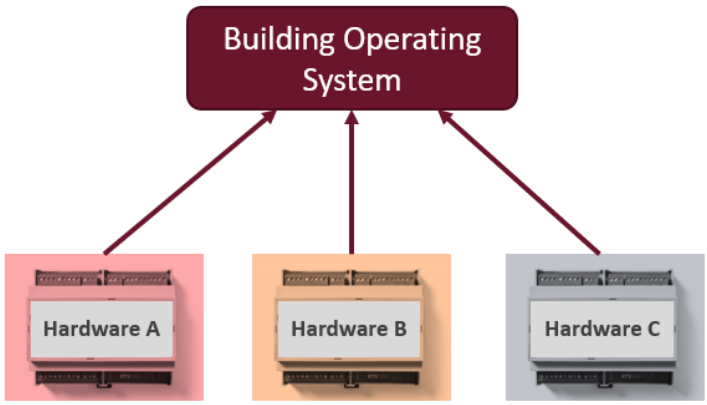
A BOS creates a convergence of these many systems, integrating all of them into a single platform no matter the hardware being used (controller A, B...), the communication technology (API, BACnet, Modbus, proprietary...) or the targeted silo (HVAC, Energy, Access control, IoT, Lockers...).

A BOS is hardware and solution agnostic, meaning that:

- A BOS is not tight to be installed on a specific hardware (and therefore to a specific manufacturer)
- A BOS is independent from any hardware it integrates
- The installation of a new hardware has no impact on the way a third party interacts with a BOS

We can definitely use the analogy of a computer or mobile OS which can be installed on different hardwares from different manufacturers with a capacity of communicating with external devices (printers, keyboard, GPS sensor, docks...) from almost any manufacturers, the third parties being the applications installed on the OS. Each application doesn't need to create their own specific driver, the OS plays the intermediary role to simplify data accessibility to apps (An image editor app didn't create every mouse driver to provide interactions on its user interface). In a similar way, **a BOS has this capacity to be installed on any hardware (server, PC...), to collect data from any devices, to translate them into a unified language, to enrich them with some context and to share them to third parties**.

As a natural evolution, a BOS extends the original purpose of a BMS/BAS (integrating usually HVAC, lighting and energy systems) to a broader perimeter (there are new other concepts and challenges to consider as well...).



An abstraction layer

Think about a service that needs to collect data from a building to provide energy insights, or from an analytics platform that needs to monitor all the equipment valves position. The developers/integrators in charge of collecting data don't really mind whether values are obtained through a series of gateways, the use of two communication protocols or by room controllers from brand A and B. What they seek about is the associated consumption values (with some context) or the valve position values. Their interest should be focused only on the data itself, not on its acquisition method (which needs to be as easy as possible). **This is precisely one of the main role of a BOS: hide the acquisition complexity to highlight exchanged data**.

A BOS creates an abstraction layer between heterogenous systems (data producers) and services (primarily data consumers). This layer hides complexity and simplifies data access to third parties. It works both ways, it's easier to collect data from a building, but it is also easier to apply control-command.

To hide systems complexity, there are several levels of abstraction provided by a BOS:

- **Level 1: a BOS hides the acquisition hardwares and provide data to third parties from individual equipment perspective**. To illustrate it, a single controller may acquire data points that are related to several equipment (lights, a VAV, multi-sensors...). The BOS creates a representation of each individual equipment and present the data to third parties accordingly (without any correlation with the controller). At the opposite, an individual equipment (usually a production, a power distribution panel...) may contain data points acquired by several hardwares (and even from several communication protocols). In that case again, the BOS provides a single equipment representation no matter the number of hardwares being used. It simplifies a lot the understanding of the system.
- **Level 2: a BOS provides a representation of a higher abstraction through syntheses and global commands**, most of the time from a spatial perspective: per room, per floor... To illustrate it: each office of a building may have 2, 3, 4... different lights to control. One way for a workplace app to control lights is to send a command to each light (through the BOS API or through a BOS connector). The level 1 hides

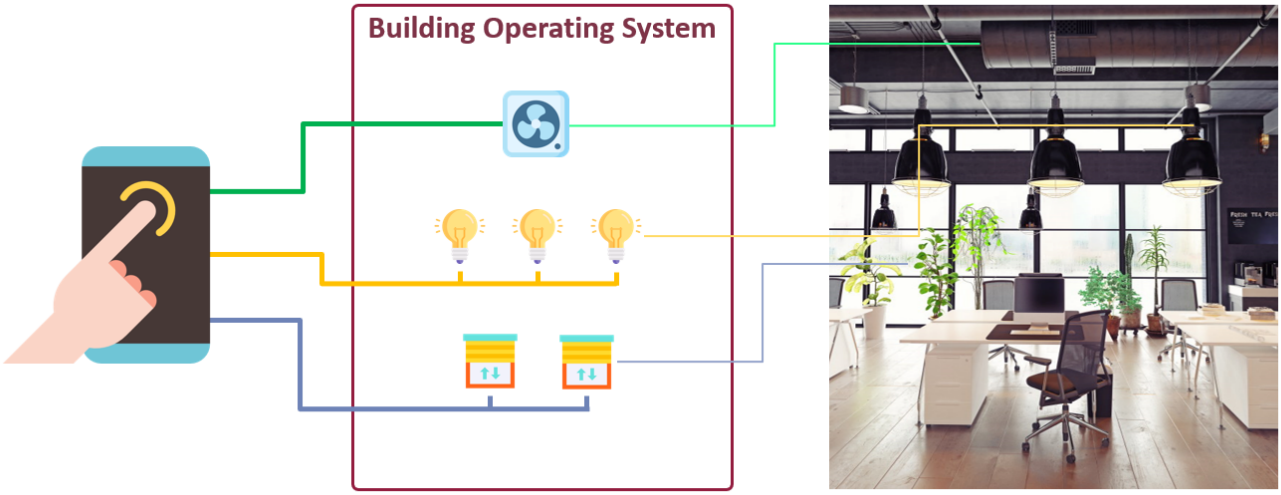
already the hardware complexity. Although it would work, it requires this third party to learn the number of lights per room, to register for changes... Alternatively, the BOS can provide a global command per space, so the app only applies a command relative to the space, leaving the BOS the mission to apply commands to the x lights of the room.

- **Level 3: a BOS provides a knowledge graph to deeper understand the relationships between equipment and assets** as seen before. This higher level will allow a third party to fully understand fluid exchanges, data exchanged between systems and create more advanced scenarios.

Depending on the third party, one level is more appropriate, the BOS should be able to provide these three no matter.

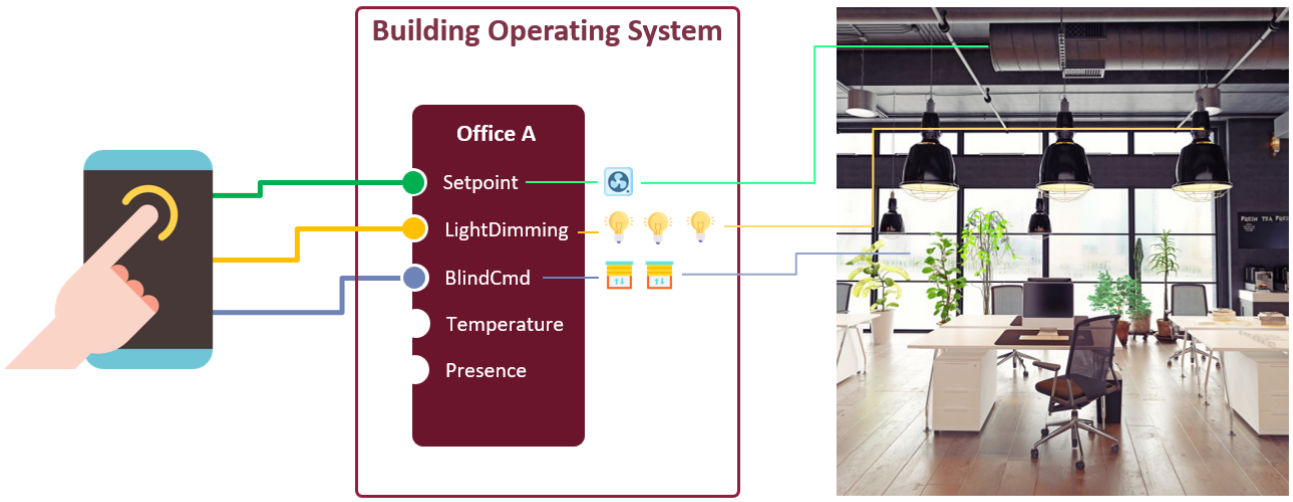
LEVEL 1

Hides the acquisition hardwares, uniforms data and provides context



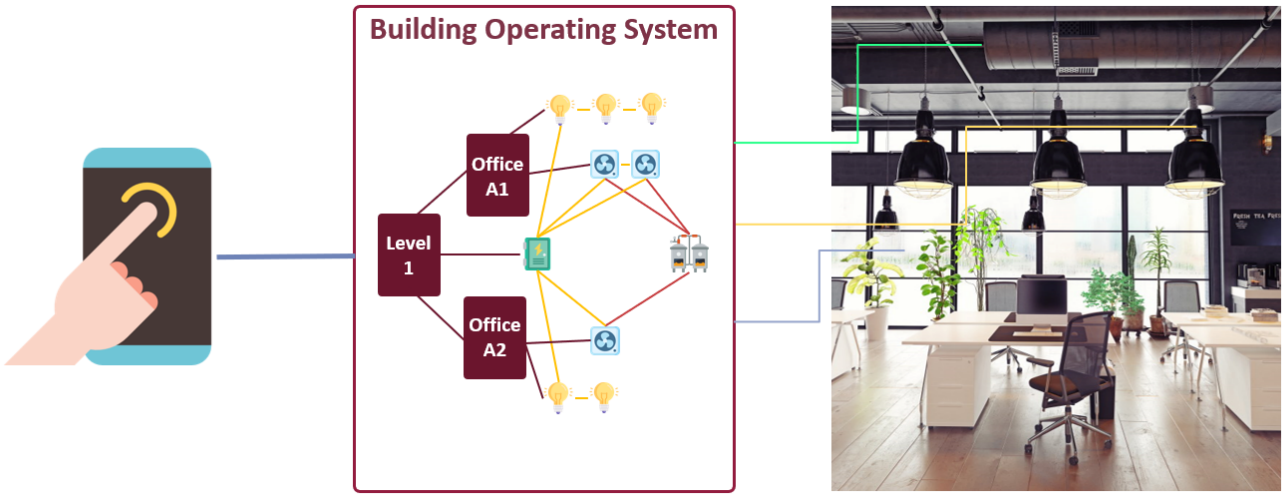
LEVEL 2

High level representation per space, floor with syntheses and global commands no matter the number of equipment



LEVEL 3

Knowledge graph to deeper understand the relationships between assets



By giving a "practical representation" of heterogenous data with syntheses by space, by floor..., a BOS helps a third party to provide its service more efficiently and in some cases, makes it plug & play. This is a big change in the industry and brings many new perspectives!

To summarize

- A BOS integrates data from any hardware (must be at least communicating)
- A BOS provides a representation made of individual equipment to third parties unrelated from the acquisition hardware
- A BOS provides aggregated data (syntheses) and global commands at a higher level, per space or per floor for example
- A BOS provides different angles to describe mutual relationships between equipment

Avoiding a Spaghettiware

On the IT side more and more services are deployed now for tenants, facility managers, property managers, owners... Although being sometimes self-sufficient, services usually require interactions with building systems (OT) or from other services to improve their added value. For example a meeting rooms booking application may require comfort data to suggest the most appropriate room or to liberate bookings based on the actual presence. In parallel, a portfolio monitoring app could require both the presence and the bookings made in the other app.

A BOS has a key role to play to create interactions between services.

Without a BOS

While new services are deployed, they are looking at interacting with other apps or with the OT systems, and this is either poorly and hardly done or it costs a lot of money. Very often more than half the budget is spent on developing "connectors" but not on the core value of each service. This is called a Spaghettiware as everyone tries to create its own connectors with others. For example, a portfolio monitoring app would create a connector to collect data from a workplace app and a workplace app would create a connector itself to get the list of available rooms from the portfolio app... All the these connectors development result with:

- An increased budget (multiplication of connectors)
- An increased inter-dependency between apps (making it harder to change one as it would impact the others)
- An increased complexity

Unfortunately, this happens every time a middleware is lacking. It may work for the deployment of a couple of services, but becomes a **spaghettiware** as the number of services increases.

With a BOS

While a BOS acts as a middleware, each application exchanges data with this intermediate layer that easily translates and enriches data from one service to another. Back to our analogies, a text editor uses an OS to store the work on a file locally so he can be sent using an email editor (another application). The text editor developers didn't try to create a "connector" with every email editor on the market. The interaction with the OS is the key to allow end users to choose (and change) their text editor independently from their email editor or any other app.

Back to our industry: let's take a popular example where a BOS collects work orders (tickets) from a CMMS app. A BOS retrieves tickets from the app and enriches them (associates it with a corresponding equipment if it's not done at the CMMS level), and once in the BOS, tickets can be shared to any third parties such as an analytics platform. The CMMS platform doesn't need to create a specific connector to the analytics app, the BOS already has these two connectors since it already communicates with each app for other purposes (generate ticket for example, and feed the analytics app with other types of data). Both apps can be changed, only the connector to the BOS will need to change.

Apply this logic to 3 services and the amount of work is even more reduced.

blocked URL

Exchanging data with third parties

What also made Operating Systems so popular at a time was their capacity to install any software found out there, install them quickly and start using them. The OS was providing a single User Interface to launch dozens of applications at the same place.

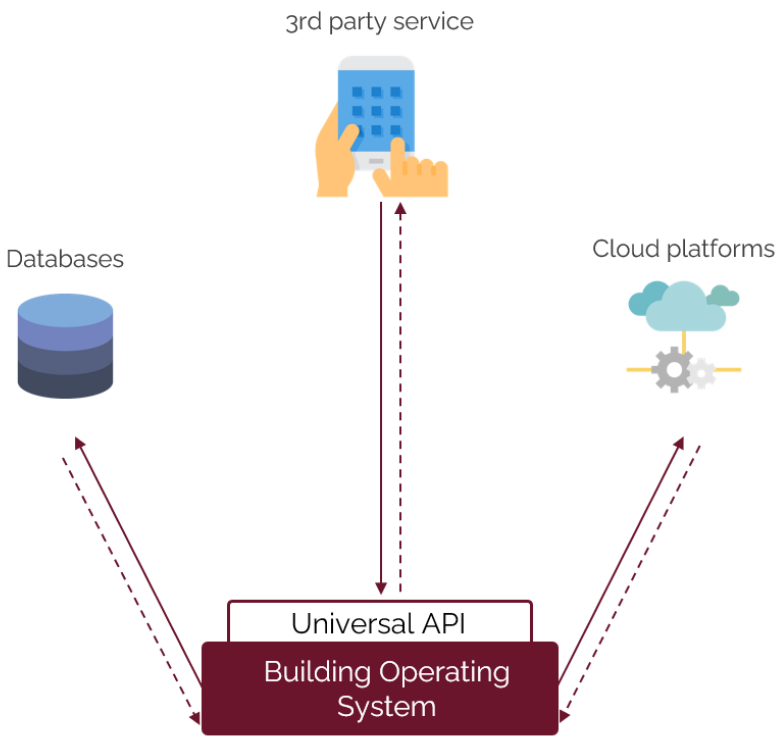
Time has changed, with the rapid development of the Internet and Software as a Service (SaaS), most of applications today are hosted in the Cloud using their own infrastructure to run efficiently and provide their continuous service, the computer or phone acting as a local interface. A BOS is then not really the place where applications should be installed (like on a computer or Smartphone) and be running within the BOS where every user should use the BOS as the local interface.

Tenants want to book a room though their smartphone, property managers want to check energy consumptions from their own office (Why should they be in their customer's basement). Even Facility managers want to operate buildings remotely as much as possible. Also, applications are already there on the market, they didn't wait for BOS to emerge to create their platforms. They don't want to create specific portability of their service on existing BOS, what they seek for is collecting data to feed their service.

Therefore **the core mission of a BOS is not to host applications but to exchange data with them**. This is again redefining our approach with BMS and middleware as they stood so far for most of them as a single place to attract all the users. A BOS may have no User Interface, it wouldn't be a big deal (a UI is still very useful to configure, monitor, debug... wait to see what Linksper can offer on this subject).

To exchange data, a BOS needs to provide an extensive range of possibilities to interact with third parties. At least:

- A BOS should be able to interact with all field hardwares (OT) using open protocols (and with the possibility for using proprietary protocols when necessary)
- A BOS should be able to interact with any solution/silos opening their data with an API (Similar to open protocols, any specific development should be avoided to reduce costs integrations)
- A BOS should provide an open, dynamic and fully documented API for third parties
- A BOS should provide other means of data exchange (MQTT, Real-time databases using SDK...)



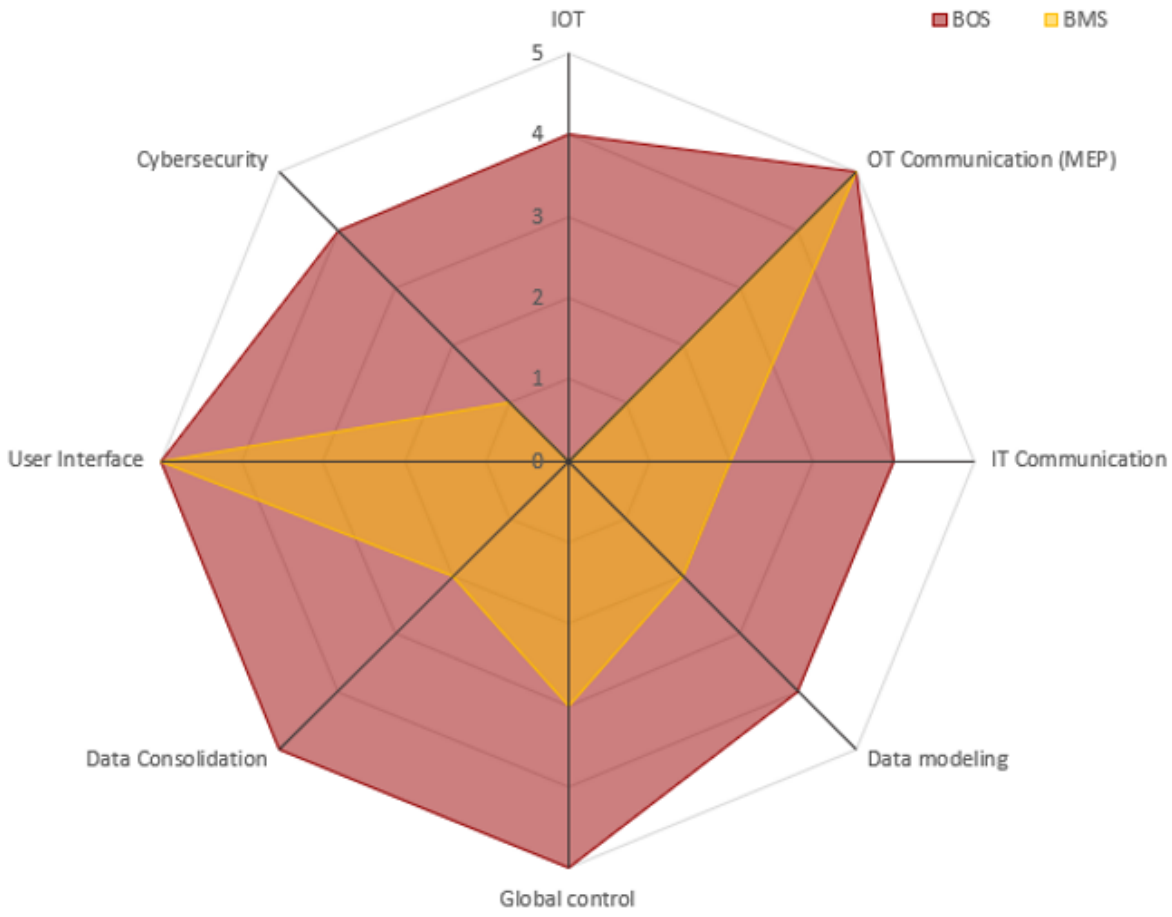
Differences between a BOS and a BMS/BAS

A BOS provides all the features of a traditional BMS

- Communication with Mechanical, Electrical and Plumbing equipment (OT)
- Global automation
- Technical User Interface (for a FM)

A BOS also provides advanced features:

- Communication with IoTs and silos (Access control, lifts…)
- Data organization (Modeling, Consolidation)
- Easy data exchanges with cloud services
- Global scenarios between services
- Cybersecurity


BMS vs BOS features
BOS
BMS
IOT
OT Communication (MEP)
IT Communication
Data modeling
Global control
Data Consolidation
User Interface
Cybersecurity
5
4
3
2
1
0