Building Management Insights Driven by a Multi-System Semantic Representation Approach

Charbel El Kaed*, Brett Leida†, Tony Gray‡
Schneider Electric: *Digital Services Platform-Innovation, †EcoBuildings, ‡Power Solutions
Email: {first.last} @schneider-electric.com

***Abstract*—The Internet-of-Things paradigm has brought exciting opportunities to increase productivity and efficiency but also customer experiences. Thus, IoT promoted both the interconnectivity of devices and systems but also their cloud connectivity. However, such plethora of connected things capture part of the contextual information in different data models which makes them operate in silos. We depict in this paper an architecture and a real experiment to connect our on-premise systems to the cloud along with a semantic representation of the contextual information. Then, we integrate a BI component exposing data from our several systems to drive better insights in our facility.**

Keywords—IoT; System to Cloud, Semantic Representation

I. Introduction

The adoption of the Internet of Things promoted the connectivity of buildings' sensors, devices and systems to the cloud. Such cloud connectivity is sustained by the ambition of promoting applications which will make use of the collected data. The aim is to rely on the gathered information in order to drive new business opportunities and added-value services for customers. Applications are expected to provide several levels of analysis ranging from monitoring and visualisation, to prediction through machine learning. Advanced applications are also expected to offer facility monitoring and optimization to achieve a larger objective such as energy saving.

Connected things are of heterogeneous types and range from low-end devices such as sensors and actuators to more capable items such as systems which concentrate many devices. In such systems, contextual information of the connected sensors and gateways is organized and expressed more often in a standard, a convention or a notation such as the single-line diagram [1]. In a given facility, different systems are usually deployed, such as a Building Management System (BMS) which monitors temperature, humidity and CO2 levels in order to regulate cooling and heating along with the indoor air quality. A Power Monitoring System is deployed in order to monitor power quality and power consumption of electrical loads often classified by usage such as lighting, heating, cooling and plug loads. These systems co-exist in the same facility, and one can assume that they can provide great insights about a facility when their data is combined. Unfortunately, in reality, they are designed to be used by different personnel and are still in operational silos. Moreover, their contextual information is expressed in different data models and relies on various notations and conventions to represent their information during the commissioning phase and across the life-cycle of the system. Their information is expressed in data models with human-friendly names such as 'Active-Energy-into-the-Load', 'Temperature'. However, such naming conventions are human dependent and are only syntax based which leads to inconsistency and incompatibility. For example, a system might expose temperature data as 'Temperature' while another system will refer to it as 'T' or 'Temp'. Moreover, such names do not necessarily provide sufficient contextual information. In this case, temperature could refer to a room temperature or a chilled water supply temperature or an outdoor dry-bulb temperature. Due to the heterogeneity of the various data models, data interpretation and analysis become seriously challenging. In addition, such heterogeneity in data representation promotes applications to be siloed in a specific domain and highly tied to its data schema, typically making the reusability of applications across buildings a difficult task.

In addition to classical timeseries data (time-stamp, value) generated by the connected things, contextual information is no longer considered as a second class citizen of the data realm. For instance, the energy consumed by lighting, cooling, heating by zone and by person is more valuable than the total energy consumed by the building in an optimisation strategy context. Thus, contextual information with the timeseries data has to be collected and exposed for applications to query and analyze.

Semantic Web technology [2] has been gaining popularity in recent years. In industrial environments, ontologies have proved to be a potent solution for issues such as data interoperability as we depicted in our previous work [3]. They are particularly useful to represent contextual information and to infer additional knowledge making applications better aware of the context. Moreover, ontologies allow linking data and offering multi-level of abstractions which offers applications the ability to query information by relying on high-level concepts for general purpose applications or by relying on specialized concepts in the case of domain-specific fields.

In this paper, we connect different systems that are already deployed on a real facility : a building management system (SBO[1]), a power monitoring expert system (PME[2]), an external weather service, and electrical and gas utility billing information. Our work aims is to represent contextual information of these systems in a multi-layer ontology in order to drive insights from the shared data. We also share our return of experience in relying on semantic modelling to represent contextual information from a real facility and how multi-layer ontologies can achieve cross-system queries. The rest of the paper is organized as follows : Section II describes our solution in detail. We cover the implementation along with results in III.

[1]www.schneider-electric.com/b2b/en/solutions/enterprise-solutions/solutions/enterprise-software-suites-building-operation

[2]www2.schneider-electric.com/sites/corporate/en/products-services/product-launch/struxureware/power-monitoring-expert.page

The related work in reviewed in IV. Section V shares lessons learned, then we conclude and outline future steps in VI.

II. Proposed Architecture

In this section, we detail the components of our architecture, as shown in Fig. 1. The systems already deployed in our facility are briefly depicted. Then, our multi-level-ontology approach to represent semantic knowledge is then detailed. Next, we describe the software system which instantiated the ontology along with the storage choices. Then, we outline the selected indicators that we expose in our facility.

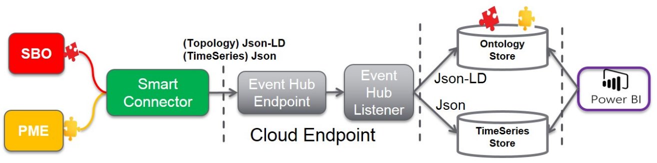


Figure 1. Overall Architecture

A. Existing Systems

There are various operational systems of the BoC, Boston One Campus ($247,000$ sq.ft.) of Schneider Electric, including the SmartStruxure Building Operation[1] (SBO) as the Building Management System (BMS), and Power Monitoring Expert[2] as the Power Monitoring System.

The role of the BMS is to monitor and control the mechanical equipment in the building in order to provide a safe and comfortable environment for the occupants. The types of mechanical equipment at the BoC include chillers, fan coil units, heat pumps, chilled beams, and energy recover ventilators (ERVs). Conditions such as temperature, humidity and indoor air quality are monitored by the BMS, and set points for these conditions are based on schedules.

More than 200 power meters have been installed in the building which are managed and monitored by the PM system. The power meters provide various types of measures, such as active and apparent energy and power, demand and power quality. The meters are placed on feeder circuits throughout the electrical distribution system as well as on critical infrastructure, such as the chillers and other HVAC equipment, data center equipment, generator, lighting and lab equipment. The extensive metering provides valuable insight into energy consumption patterns. The PM system also monitors digital protective circuit breakers which themselves include power monitoring and communication capabilities.

While the BMS has sensors for basic outdoor conditions (temperature, humidity), many additional weather observations are obtained from the Schneider Electric Weather Services. The types of observations include wind speed and direction, solar radiation, cloud cover and precipitation type and amount.

B. Multi-level Ontologies

Heterogeneity reigns between devices and systems across domains; however, they have a lot in common. Therefore, at Schneider Electric, while working with different business units and segments on their data models and applications, we defined two layers of ontologies: common ontologies and specific ontologies, as illustrated in Fig 2.

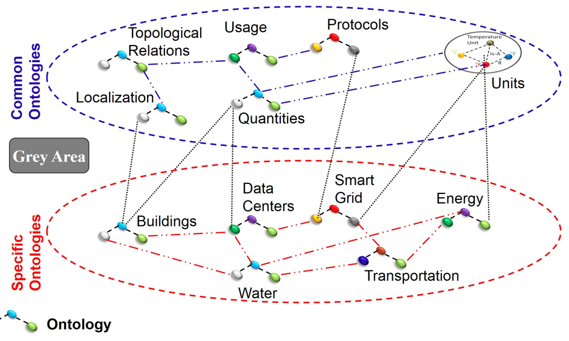


Figure 2. Common and Specific Ontologies

The common ontologies consist of common concepts used across the business units and segments such as :

- Protocols : classifies the communication protocols along with information regarding the supported communication medium and range. Such information can be used during diagnostic and maintenance operations.
- Quantities : exposes several type of concepts. Physical quantities represent the sensed concepts (e.g. temperature) or calculated such as energy. Quantities also contains high-level concepts, e.g. Device and Server.
- Units : used by the physical quantities. It expresses the quantity symbol along with its SI unit. It also points out the conversion from one unit to another and handles scale factors like Ampere to μAmpere.
- Topological relations : classifies the relations between entities and specifies their property e.g. transitive, symmetric. It depicts general relations such as *isConnectedTo* which capture several dimensions like electrical wiring and network connectivity. Such relations are a necessity for measurement aggregation.
- Localization : sets a common definition for entities like building or floor. It defines such concepts along with the relations between, e.g. a room *isLocatedIn* floor where *isLocatedIn* is a transitive relation.
- Usage : general ontology which can be combined with the other common ontologies, for instance the active energy for lighting or the outside air temperature.

The common ontologies capture the shared concepts across business units and segments. However, these ontologies are extensible thanks to the expressiveness of the semantic web. The specific ontologies are silo and domain-oriented and rely on the common ontologies. For instance, in Fig. 2, the Buildings ontology would rely on the Physical Quantities ontology to express the measurements and on the Localization to reference a site or a floor. It also extends the concept of a *Server* from quantities to add a *BuildingServer*. In the Energy domain ontology, the concept *PowerServer* will extend the concept *Server*. Thus, when the general concept *Server* is used in a query, both Building Server and Power Server instances will be returned due to the inference capability. The ontology modeling required several iterations to achieve a relatively stable version. A set of grey area has been used as a buffering zone before deciding whether a concept used in more than two ontologies should become a common concept. Moreover, the specific ontologies can also rely on existing dictionaries such as the HayStack [4] from the building domain.

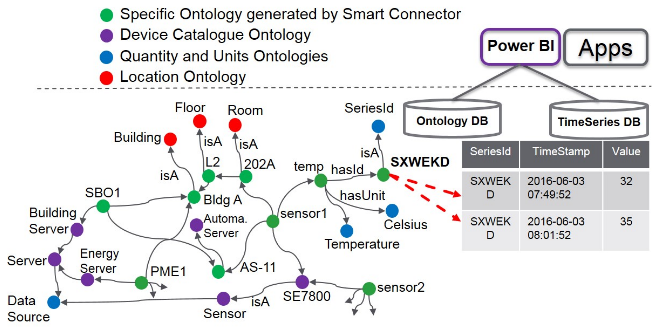
C. SmartConnector

The *SmartConnector*[3] is a software platform which serves two main roles in this architecture. First, it aims to capture the contextual information of the underlying on-premise systems and represent such knowledge in an ontology to be pushed to the cloud. *SmartConnector* creates the building instance ontology by combining information from multiple sources. Some information is discovered dynamically by reading the various objects in the BMS and PM, and other data is created by importing static information from various sources such as spreadsheets and structured files. In fact, the existing deployed systems usually do not capture all the contextual information during commissioning for various reasons such as restrained budget, limitations in the commissioning tools, or lack of interest from the commissioning personnel. The instantiated ontology references common and specific ontologies.

The second role of *SmartConnector* is to relay the sampled timeseries from the underlying on-premise systems to the cloud data store. Once the topology is instantiated by *SmartConnector*, it relies on the available timeseries unique identifiers to determine the series data to push. The topology and the timeseries are pushed to a cloud endpoint which handles the routing according to the content, as shown in Fig. 1.

D. Multi-Systems Ontology & Hybrid Storage

Our modular ontologies as discussed in II-B allow referencing any dimension as shown in Fig. 3. *SmartConnector* instantiates the building ontology by referencing the device catalogue, the quantity, units and the location ontologies. For example, *sensor1* is a *SE7800* sensor, connected to the Automation Server *AS-11*. The sensor measures *Temperature* in *Celsius* and it is located in room *202A*. The designated room is located on *L2* of building *BldgA*. Moreover, *sensor1* has a unique SeriesId: *SKWEKD*.



SeriesId	TimeStamp	Value
SXWEK D	2016-06-03 07:49:52	32
SXWEK D	2016-06-03 08:01:52	35

Figure 3. Contextual and TimeSeries Data

The contextual information of the building is expected to be slowly changing. Updates are expected to occur occasionally, for example after a faulty device replacement or re-arrangement of the space. Moreover, the contextual information seems to be interlinked and complex; therefore, a reasoner would allow the extraction of additional information. For example, *find all the sensors in BldgA* is a query that could be formulated on the building ontology shown in Fig. 3. Since sensor1 is located in *202A* which is transitively connected to the floor and the building, then the returned results will contain sensor1. The ontology also allows to query multiple systems in a single query due to abstraction, such as *find all servers with their locations and connected elements along with their measure types and seriesid*. In fact, SBO (resp. PME) is an instance of *Building Server* (resp. *Energy Server*) which is based on a common abstract base class, *Server*. In addition, the contextual information is represented through the multi-level linked ontologies. Thus, a common query is used to extract information from several systems.

Timeseries, which include the time-stamp and value pair, are sampled and pushed to the cloud frequently; therefore, data storage requirements are expanding over time. Additionally, such data does not require any semantic reasoning. Therefore, it can be placed in a separate data store to avoid slow query performance, since a semantic reasoner [5], [6] explores the stored data to extract additional information.

Due to the heterogeneous nature between contextual information and timeseries, the difference in their updating frequency and the reasoning requirements, we opted for the data architecture and storage as outlined in Fig. 3. The contextual information is persisted in an ontology storage providing reasoning capabilities, while the timeseries is inserted in a tabular storage. Both repositories are linked through the SeriesId.

E. EU.BAC KPI's

The *European Building Automation and Controls Association*[4] represents the European manufacturers for Home and Building Automation and Energy Service Companies. It also promotes and influences the development and implementation of EU directives and regulation. *EU.BAC* publishes several performance indicators [7] in order to provide insightful information on building automation.

We selected a set of KPIs from the *eu.bac* KPI specification [7] and formulated queries to extract multi-system information from the stored data to compute the KPIs. These KPIs vary from simple raw meter measurements to more elaborate calculations requiring the combination of information from several measurements sampled by different meters. For example, the electrical energy consumption of the entire building *KPIEE.b* [7] is directly extracted from the main meters of the PM. However, *KPIEH.d* [7] which represents the heating degree days (HDD), requires the average outdoor temperature from the BMS and the energy consumption of the heating system from the PM system. Once these KPIs are selected and their queries are formulated, we connected Power BI[5] an interactive data visualisation tool to represent the retrieved information in order to be analyzed by a facility manager.

[3]https://www.youtube.com/watch?v=DXHvRRL4ycY

[4]www.eubac.org

[5]http://powerbi.microsoft.com

III. Implementation & Results

We depict in this section some implementation details and performance results of our aproach.

A. Multi-level Ontologies

The common and specific ontologies are designed over several iterations through the use of the *Protégé* desktop client[6]. The ontologies were rendered in *html* format and published on an internal server. The aim is to facilitate the generation of the facility's ontology by referencing the existing multi-level ontologies. The generation of the facility's ontology is handled programmatically by the SmartConnector as depicted next.

B. SmartConnector & Data

SmartConnector is a development framework for which processors are written in *.NET* to implement certain business logic. The processors can be run on-demand or on a schedule.

We developed an on-demand processor to create the ontology and to send it to the Event Hub Listener. As there is limited contextual data stored natively in the BMS and PM systems, much data was captured in spreadsheets and imported by the processor. Such data includes: building information (floors, rooms, their purpose), equipment (lighting, labs, HVAC, data center, plug loads), physical location of equipment, power meters and BMS controllers. In addition, various objects are discovered in the BMS (controllers, points and trends) and the PM system (power meters, points and trends), and all of this data is stitched together to instantiate the ontology.

An ontology can be serialized in several formats [8], [9]. In our work we selected Json Linked Data (Json-ld) [10] which is an extension of Json since it is easier to understand and to generate than other formats like OWL/XML or RDF/JSON. The SmartConnector processor that we designed takes about *30* seconds to generate the ontology in json-ld format from several sources. The generated json-ld size is $500KB$. The processor sends the generated file in less than 2 seconds to the cloud endpoint, which is a Microsoft Event Hub[7] endpoint.

Regarding the timeseries, there are other SmartConnector processors that manage the extraction of the timeseries from the systems (using a Schneider-specific web service, *EcoStruxure Web Services*), and reliably push the records to the Event Hub Listener. These processors are scheduled to run every 15 minutes, and take $30 - 120$ seconds per system to transfer all timeseries records remotely, depending on the number of configured timeseries. The timeseries are also serialized in a json format with series id, time stamp and value. Both processors encapsulate the content with a json header indicating the content type (graph or timeseries), the client name and unique identifier, the operation type, along with ontology Uri and version as shown in listing 1.

Once the topology and the timeseries are pushed to our Event Hub, the Listener will handle the data as discussed next.

[6]http://protege.stanford.edu

[7]https://azure.microsoft.com/en-us/services/event-hubs/

C. Event Hub Listener & Storage Technologies

We implemented an Event Hub Client in Java based on the available open-source code[8] and deployed it on an Linux VM, A4 basic with 8 cores and 14 GB RAM. The Client listens to a Microsoft Event Hub endpoint[7] and receives a message sent by SmartConnector with a header as shown in 1. The content-type indicates to the client the nature of the payload, graph (json-ld) or timeseries (json). Then based on the operation (*Update, Init*) attribute, the client either appends received data to an existing database/table or creates a new one by combining the client name and his unique id (GUID). As shown in Fig. 1, 3, the client inserts the received data into two types of storage : an ontology store and and timeseries store.

```
{ "content-Type": "Graph or TimeSeries",
  "client-GUID": "XVSFD-SDF",
  "client-Name": "BOC-SBO-PME",
  "operation": "Init or Update",
  "generatedBy": "SmartConnector_SBO_PME",
  "ontology-Uri":
    "http://schneider-electric.com/Clients/BoC#",
  "payload":
    {"ontology(json-ld) or timeseries(json)"}}
```

Listing 1: Content Header Received by the Client

As an ontology storage, we selected Stardog 4.0.2[9] database with a community license with 2GB of RAM. It is installed on the same virtual machine as the Event Hub Client. Our client relies on Stardog Native API (SNARL) to bulk insert the json-ld payload. The EventHub Client inserted 7353 triples in Stardog with an average speed of 35K triples/sec. Additionally, json-ld references the multi-level ontologies, such as the Location, Units and Quantities. Therefore, the Client injects these dependencies in the ontology store as well. For the BoC ontology, the overall imports with json-ld reaches 8221 triples.

For the timeseries storage, we selected MySQL Database as a service in Azure which can store up 5GB of data. We instantiated a timeseries table with three columns (series id, time stamp, value) with series id and time stamp as a primary key. Due to the simplicity of the timeseries model, any other storage technology could have been chosen. Our EventHub Client is also capable of writing to MongoDB, HBase and HDFS storage technologies. However, we selected MySQL due to its ease of use with Power BI as we depict next. SmartConnector pushes an average of 229 series entries per file. The Client inserts around 229 series entries between [4..35] seconds. The maximum amount occurs when there are multiple timeseries ids in the same file and the minimum when there is only one timeseries id.

D. Power BI

Business Intelligence tools are becoming more popular as a means to derive insights from available data. Their popularity is due generally to their friendly user interface and high-level abstract functions for easy data manipulation. In our architecture we selected Power BI Desktop[5] as a BI tool. It has a built-in connector to MySQL and a native REST client. As any ontology storage, Stardog exposes a Sparql/REST endpoint and supports the following results formats: JSON and CSV.

[8]https://github.com/hdinsight/eventhubs-client

[9]www.stardog.com

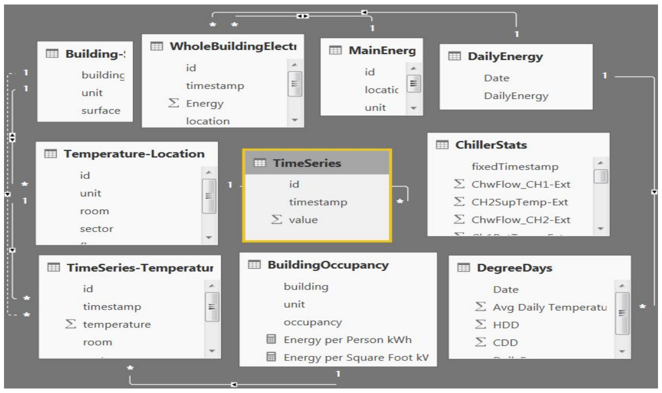


Figure 4. Power BI tables

In Power BI, we carried out two types of queries : SPARQL/REST where we set the result format to CSV, since Power BI is capable of processing csv formats natively. We also connected Power BI to our MySQL database instance where the timeseries are stored. Figure 4 outlines the different tables in Power BI. The timeseries table (in the center) is the MySQL table while all the others are the Sparql query table results. These queries return different tables such as the the temperature timeseries id along with the unit and the location of the sensor by its room, sector, floor and building. Each table represents a selected KPI from the EU BAC document, as mentioned in II-E. Moreover, all of these tables are linked with the timeseries table by relying on the id attribute. The join operations between these tables and the timeseries table are performed internally in Power BI. We were not able to measure the join performance carried out by Power BI.

Performance of Sparql queries varies according to the complexity of the queries, number of returned attributes and whether the inference is activated or not. Our most complex queries, which return a table of $6 * 43 = 258$ elements representing the temperature sensors along with their details location by building, floor, sector and room, takes on average 9 seconds. This performance time is dependent on the network latency between the Power Bi Desktop Client and the Azure Cloud VM where Stardog is deployed. When both client and Stardog are on the same server, the query takes approximately 3 seconds. The other less complex queries without any need for the inference executes in less 40 ms remotely and less then 25 ms when placed on the same VM.

The MySQL DB contains 2 months of data, $1,565,335$ rows occupying 100 MB. Power BI loads around $300,000$ rows in around 3 min. However, most of the time it cannot load the rest. Moreover, Power BI loads the entire MySQL table in memory and perform join operations with the Sparql returned tables.

Power BI was used to further generate derived quantities from retrieved timeseries data and to otherwise transform quantities from their stored format to one more suitable for presentation. For example, a common environmental measure correlated to building energy consumption is the 'Degree Day'. Computation of heating and cooling degree day values from the interval temperature readings was done using the query language supplied with Power BI [11]. Temperature values were averaged over each day and then either subtracted from a reference temperature (HDD) or had a reference temperature subtracted from the average (CDD) with the result truncated at zero for both cases, removing any negative quantities.

Similar manipulations combined the coolant flow rate, inlet temperature and outlet temperature of a chiller unit to determine the cooling power output. Pump power consumption from the cooling loop was subtracted from the total chiller power to determine the chiller electrical consumption prior to calculation of the coefficient of performance (CoP) of the chiller unit.

Figure 5 shows an assortment of simple visualizations of the raw and aggregated data. It also has a sample of correlated dashboard widgets where selection of sectors, rooms and floors on the lower row of widgets interactively generates matching timeseries displays in other displays. In Fig. 5, from upper left: (a) the daily energy by day, (b) the CoP performance. The average temperature by room with the ability to drill up to the sector, floor level is depicted in (c). Moreover, a quick overview of the average temperature is shown in (d). Energy per HDD as mentioned previously is previewed in (e). While the cost by Electricity and and Gas are shown in (f). Energy per Person and Square foot are shown in (g) and (h). In (i), we show the power consumption by room, sector and floor by type of load such as lights, kitchen, lab, IT.

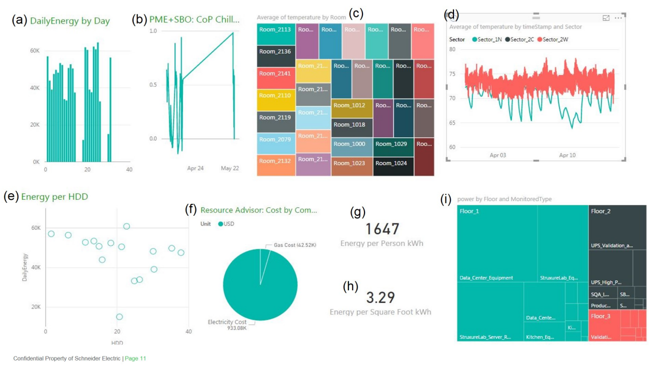


Figure 5. Power BI Screen Shot: Overview KPIs

IV. Related Work

BOSS : the Building Operating System Services [12] supports fault-tolerant applications on top of the distributed physical resources present in buildings. BOSS relies on two layers, the Hardware Presentation Layer (HPL) and the Hardware Abstract Layer (HAL). HPL abstracts all sensing and actuation by mapping each individual sensor or actuator into a point. These points produce timeseries data. HPL allows metadata tags to be attached in the form of key value pairs to describe the data. The HAL provides an approximate query language to search through multiple views of underlying building systems, in a three-dimensional space : electrical, HVAC and lighting. Even though the approach seems scalable, it is not clear how the tags can be attached to the HPL and how expressive the semantic representation is behind. As for the query language, it seems only tag and timeseries oriented; there is no support of inference, which is essential in our approach to query multiple systems.

The authors in [13] propose a Haystack Tagging Ontology (HTO) to complement the haystack entities and to be able to query the data. They propose annotating data using HTO, and then using a set of rules to generate the knowledge (ontology) to query it. Transformation rules, written in SPARQL, are used to populate the knowledge. Once the data is 'assembled' by relying on these rules, the next step is to query it using SPARQL. However, according to the authors, the transformation rules are not always accurate and there is no backward reversibility between the tags and the generated model.

An interesting comparative study between IFC, HayStack [4] and SAREF metadata concludes in [14] the non-completeness of these models, which helps to justify our use of multi-layer ontologies and their extensibility.

Gao et al. present in [15] a data-driven approach based on analyzing timeseries and labels to infer the metadata's sensors. This approach seems promising and can partially substitute the programmatic generation of the topology (json-ld) in SmartConnector based on external, manually added knowledge. However, since Schneider Electric design such systems, we lean more towards the evolution of our systems to better capture such tags.

Ploenings et al. present in [16] BASont, a building automation system ontology as a knowledge representation to capture the various use cases of the system's life cycle. The BASont ontology is generated by AUTEG tool [17], an automated commissioning tool to be used on a Building Automation System. This approach pushes towards the evolution of the commissioning tools of our BMS and PME in order to better capture the contextual information during commissioning instead of injecting it programmatically in SmartConnector.

V. Lessons Learned

We share in this section, our lessons learned from our real experimental prototype to connect our on-premise systems to the cloud. Accuracy in capturing contextual information through on-premise systems is key to provide precise insights at the application level. In fact, we had to request the facility sheets in order to prepare such contextual information in excel sheets for SmartConnector. Such tasks makes our approach time consuming to reproduce unless our commissioning tools and methods evolve to better capture contextual information and generate the facility's ontology. The multi-level ontology modelling requires several iterations before achieving a stable version. The modularity of the ontologies between specificity and abstraction offers great flexibility in the queries depending on the domain of the cloud applications. One benefit is the ability to reuse the reports and queries across multiple sites to save time and effort. Similar insights can be gained through analyzing traditional relational data stores. However, doing so requires much effort to combine disparate data models and schemas from heterogeneous systems. The use of an ontology provides a common way to homogenize data from these systems and abstract queries which rely on inference to retrieve multi-systems data. Power BI as on the shelf component seems promising however, we doubt its scalability, since the join operations are made in memory locally.

VI. Conclusion

We presented in this paper our experimental work to connect multiple on-premise siloed systems to the cloud. We relied on a software brick (SmartConnector) and several sources to capture and augment contextual information. A facility's ontology representing such contextual information was generated programmatically and pushed to the cloud along with timeseries data. The generated facility's ontology references our multi-layer ontologies which allows high flexibility when performing queries. In fact, queries can rely on high level abstract concepts or specific concepts to target a specific domain. We propose two types of data storage, an ontology database for the contextual information and a simple column table for the timeseries data. We used Power BI as an interactive business intelligence tool to visualise actual data in our facility retrieved from the EU BAC KPIs. For the next steps, we will be focusing on the analytics to be applied on the represented data along with another solution to bridge the gap between ontologies and timeseries outside of Power BI to improve performance and achieve better scalability.

References

[1] T. Nasser, *Power Systems Modelling and Fault Analysis : Theory and Practice*. Elsevier Ltd., 2007.

[2] J. Ye *et al.*, "Semantic web technologies in pervasive computing: A survey and research roadmap," *Pervasive and Mobile Computing*, 2015.

[3] C. E. Kaed, Y. Denneulin, and F. G. Ottogalli, "Dynamic service adaptation for plug and play device interoperability," in *Proceedings of the 7th International Conference on Network and Services Management*. International Federation for Information Processing, 2011, pp. 46–55.

[4] "Project haystack," http://project-haystack.org/.

[5] F. Baader, I. Horrocks, and U. Sattler, "Description Logics," in *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, and B. Porter, Eds. Elsevier, 2008, ch. 3, pp. 135–180.

[6] J. Urbani *et al.*, "Querypie: Backward reasoning for owl horst over very large knowledge bases," in *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ser. ISWC'11, 2011.

[7] eu.bac, "Certifying energy efficiency of building automation and control systems, at first delivery and during the lifetime, part 4: Specification of key performance indicators," www.eubac.org, 2015.

[8] Bechhofer *et al.*, "Web ontology language," http://www.w3.org/TR/owl-features/, 2004.

[9] W3C, "Resource description framework," http://www.w3.org/RDF, 1999.

[10] ——, "Json-ld 1.0: A json-based serialization for linked data, w3c recommendation," https://www.w3.org/TR/json-ld/, 2014.

[11] Microsoft, "Power query (informally known as "m") formula reference," https://msdn.microsoft.com/en-us/library/mt211003.aspx, 2016.

[12] S. Dawson-Haggerty and others., "Boss: Building operating system services," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013.

[13] C. Victor *et al.*, "An ontology design pattern for iot device tagging systems," *5th International Conference on the Internet of Things*, 2015.

[14] A. Bhattacharya *et al.*, "Short paper: Analyzing metadata schemas for buildings: The good, the bad, and the ugly," in *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys '15. ACM, 2015, pp. 33–34.

[15] J. Gao *et al.*, "A data-driven meta-data inference framework for building automation systems," in *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys '15. ACM, 2015, pp. 23–32.

[16] J. Ploennigs *et al.*, "Basont - a modular, adaptive building automation system ontology," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 4827–4833.

[17] H. Dibowski *et al.*, "Automated design of building automation systems," *IEEE Transactions on Industrial Electronics*, 2010.