This paper has been accepted in the technical track of IEEE 3rd World Forum IoT (IEEE WF-IoT) to be held on 12th Dec. 2016.
This is an author copy for personal record.

SQenIoT: Semantic Query Engine for Industrial Internet-Of-Things Gateways

Charbel El Kaed*†, Imran Khan†, Hicham Hossayni†, Philippe Nappey†
Schneider Electric Industries: *Digital Services Platform, †Corp. Tech. - Architecture & Systems Team
Email: {charbel.el-kaed, imran2.khan, philippe.nappey} @schneider-electric.com
hicham.hossayni@non.schneider-electric.com

***Abstract*—The Advent of Internet-of-Things (IoT) paradigm has brought exciting opportunities to solve many real-world problems. IoT in industries is poised to play an important role not only to increase productivity and efficiency but also to improve customer experiences. Two main challenges that are of particular interest to industry include: handling device heterogeneity and getting contextual information to make informed decisions. These challenges can be addressed by IoT along with proven technologies like the Semantic Web. In this paper, we present our work, SQenIoT: a Semantic Query Engine for Industrial IoT. SQenIoT resides on a commercial product and offers query capabilities to retrieve information regarding the connected things in a given facility. We also propose a things query language, targeted for resource-constrained gateways and non-technical personnel such as facility managers. Two other contributions include multi-level ontologies and mechanisms for semantic tagging in our commercial products. The implementation details of SQenIoT and its performance results are also presented.**

***Index Terms*—Internet-of-Things; Semantic web; Ontology; Industrial Internet-of-Things; Embedded Systems**

I. Introduction

The Internet-of-Things (IoT) promises to interconnect devices and objects together on a massive scale [1]. Such amalgamation allows interactions and collaborations between these entities in order to fulfill specific tasks. Eventually, such tasks differ according to the context and environment of the application. They range from sensing and monitoring of a physical property such as temperature or humidity to controlling and optimizing a facility in order to achieve a larger objective, e.g. an energy management strategy.

Semantic Web technology [2] is gaining more popularity and over the years we have observed its use to develop rich and interactive applications [3]. In industrial environments, Semantic Web [4] has proved to be a potent solution for issues such as data interoperability. It is particularly useful to link cross domain data and to infer additional knowledge making applications better aware of the context.

In industrial domain, connected things are of heterogeneous types and range from low-end devices such as sensors and actuators to more capable ones such as gateways. Such diversified connected things embed different resources : CPU, memory and bandwidth which impact their communication mediums, protocols and data models. Due to the heterogeneity in various data models, data *interpretation* and analysis become seriously challenging. Another aspect of working in embedded domain are the inherit limitations that lead to scalability issues and high complexity in using standard technologies like SPARQL [5]. Therefore, we need lightweight solutions, that are scalable, less complex and user friendly.

In this paper we present our Semantic Query Engine for Industrial Internet-of-Things (SQenIoT) to tackle the semantic challenge we faced in our business due to plethora of devices as well as data heterogeneity coming from different domains. SQenIoT is a simple but robust solution to provide Schneider Electric gateways, the ability to process cross domain data from heterogeneous devices and sensors. The power of Semantic Web is used for two purposes: *first*, to tackle the data heterogeneity and *second*, to search, filter and aggregate data at the edge and push only the relevant data to our cloud platform.

The rest of the paper is organized as follows: Section II presents the real motivation behind our work. We draw a set of requirements from a scenario to illustrate the bottleneck faced by the current solution. Then we describe our proposal in section III. Section IV covers the implementation details along with results. We overview, in section V, related works then conclude and depict our future steps in section VI.

II. Background and Requirements

In the industry, more often, communication protocols and data representation are fairly concise and low level. Usually it consists on a set of registers exposing measured data which can be accessed thanks to a prior knowledge on how those registers are mapped to specific information. For example, on a specific Modbus[1] device, *register 3650* measures active energy in *KWh*. Such information as the measured quantity (energy, power, temperature) and their units (Watt, Kilo Watt, Celsius, and Fahrenheit) are explicit in a data sheet. Once retrieved, the data can be expressed in a data model with human friendly names such as 'Active-Energy-into-the-Load' or 'Temperature'. However, the naming convention is human dependent and is 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'. Therefore, it requires a lot of effort to make applications and machines to understand the meaning of the data before aggregating and analyzing it. Even though this process can be semi-automated as we depicted in our previous work [6], a human verification is still needed to

[1] www.modbus.org

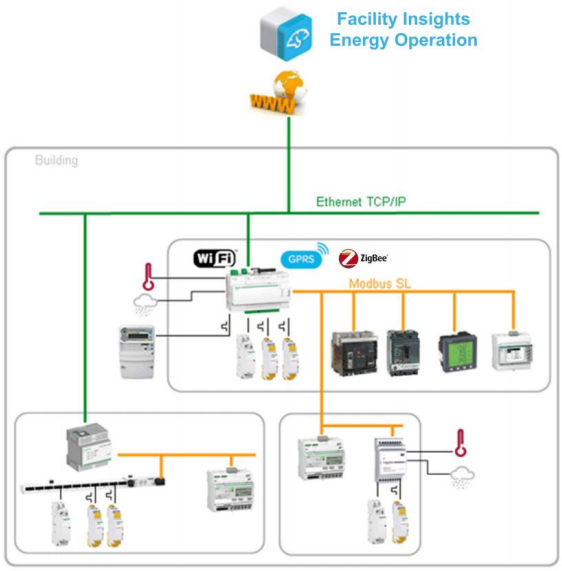

Fig. 1. Comx Architecture Deployment Example

validate the semantic matching. In addition to the classical telemetry data sensed 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. Therefore, in order to have an interoperable data that can be analyzed across domains, a common metadata vision is required.

Schneider Electric introduced a commercial IoT gateway, the Comx'200[2], targeting small to medium facilities. It is capable of integrating heterogeneous devices and sensors with different communication mechanisms, protocols and data models as shown in Fig. 1. Additionally, the Comx'200 can push data to Schneider Electric Cloud Platform for storage and analysis. For example, at the application level, the Facility Insights Services[3] query the collected data for several purposes such as energy efficiency, asset maintenance and management in different market segments like retail or health-care.

The Comx'200 is designed to be installed and commissioned by an electrician. However, after its installation and commissioning, a Comx'200 has several stakeholders. In this work, we consider a facility manager who is mainly responsible to ensure : *1)* the comfort of the facility's occupants, *2)* air quality (CO2 and humidity), and predefined temperature range according to the existing weather conditions, *3)* facility budget regarding energy consumption by lighting, heating and cooling equipments, *4)* maintenance of the facility and the equipment.

Typically a facility manager is a non-technical person and based on our real-world interactions and deployment experiences, he expects from a gateway the ability to : *a)* Search for specific devices/sensors easily. *b)* Configure alerts or notification messages using customizable conditions. *c)* Execute specific queries and quickly look up his facility's real-time data like energy consumption by area, by time and other parameters. *d)* Achieve all these without being compelled to manipulate different data models or use complex query language which may involve extra learning steps.

Based on these requirements and the general feedback, it is evident that not only we need to provide a technical solution for the business requirements but also we need to make it user friendly with right level of abstraction to hide the complexity and to provide rich interactive experience.

III. Contributions

In this section we discuss our contributions, we first present our approach relying on multilevel ontologies to support a common metadata vision. Later we present our solution to perform semantic tagging of the device data, i.e. add additional metadata using the concepts in ontologies. Finally we discuss SQenIoT and our proposed domain specific query language.

A. 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 sets of ontologies: common ontologies and specific ontologies, as illustrated in Fig 2.

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.
- Physical quantities: expose the sensed concepts e.g. temperature or calculated concept such as energy.
- Units: used by the physical quantities to express the quantity symbol along with its SI unit. It also points out the conversion from a unit to another and handles scale factors like Ampere to μAmpere.
- Topological relations: classifies the relations between entities and specifies the property of such relations e.g. transitive, symmetric. It depicts general relations such as *is-ConnectedTo* which capture several dimensions like the electrical wiring, the network connectivity. Such relations have become a necessity for measurement aggregation.
- Localization: sets a common definition for entities like building or floor. This ontology also 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 ontology web language. The specific ontologies are silos & domain oriented and rely on the common ontologies. For instance, in Fig. 2, the

[2]www.schneider-electric.com/en/product-range/62072-enerlin-x-com-x
[3]www.schneider-electric.com/en/product-range/63092-facility-insights-services

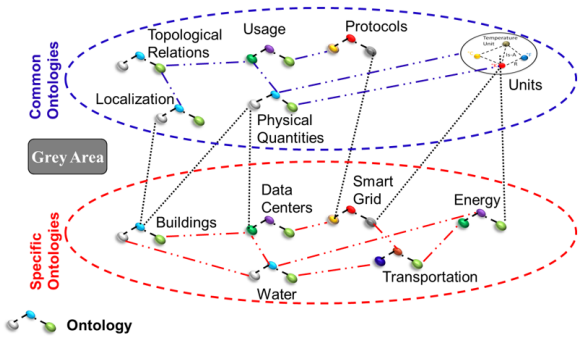

Fig. 2. Common and Specific Ontologies

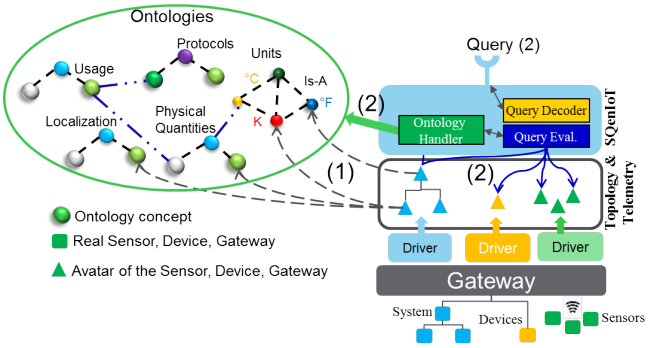

Fig. 3. Architecture: Semantic Tagging and Query Principals

Data Centers ontology would rely on the Physical Quantities ontology to express the measurements and on the Localization ontology to add the concept of a *rack*. This concept will be added as a sub-concept of the general class *Location* from the Localization ontology. Therefore, any instance of the *rack* concept can still be queried thanks to the inference capability. The ontology modeling requires 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 specific ontologies should become a common concept or not. Moreover, the specific ontologies can also rely on existing ontologies such as the HayStack project[4] which is focused on the Building Automation domain.

The global and local ontologies are hosted at Schneider Electric Cloud Platform and a subset is deployed at the gateways level to help in device data annotation.

B. Semantic Tagging

The annotation is the process of linking concepts from common or/and global ontologies with the actual data exposed by the gateways, as shown in Fig. 3(1). Like any multi-protocol gateway, the real sensors, devices and gateways are represented internally in a data model or an avatar which resides in memory or is persisted on disk. A semantic tag is a triplet composed of an ontology module reference, a concept or a verb and an optional instance from the ontology. The semantic tag is added to the internal representation (data model or avatar) at the gateway level. A semantic tag is added on a device or a variable representation. For example *Unit:* $^{\circ}C$ and *Protocol:Modbus* are two semantic tags referencing respectively the *Unit* (resp. *Protocol*) and the instance $^{\circ}C$ (resp. *modbus*) in the ontology modules. The *Protocol:modbus* tag can be added at a meter level, while the *Unit:* $^{\circ}C$ tag is added at the variable level. The conversion rule between $^{\circ}C$ and $^{\circ}F$ for example is expressed and stored as a rule in the *Units* ontology. We distinguish between the two different semantic tagging mechanisms as automatic tagging and commissioning.

Automatic Tagging: a driver handles the protocol communication along with the data decoding up to its representation in the gateway's data model. For instance, a *Modbus* driver is expected to decode and extract a data frame from a specific Modbus register along with identifying its structure in order to expose it in the gateway's data model. The automatic tagging is performed at the driver level for at least the *Protocols*, *Units* and *Quantities*. Although the driver is capable of automatic annotation of part of the data, the contextual information like usage and location are only known at the gateways and are specified at the commissioning phase.

Commissioning: it is handled through a user interface during the gateway installation phase after all the wiring and pairing has been performed. For instance, the usage of the sensor along with its location are known only at the commissioning phase hence, the installer relies on a commissioning tool in order to tag the data from both the *Usage* and *Location* ontologies. Fig. 4 shows an example from our gateway, where the installer can select a sensor and tag its physical location by relying on the *Location* ontology module. The commissioning tool displays the options, based on the *Location* ontology, to instantiate the facility (e.g. buildings, floors, rooms).

C. SQenIoT: An IoT Semantic Query Engine

Once the data is annotated and exposed by the gateway, the query execution can take place. As shown in Fig. 3(2), SQenIoT exposes a query interface along with a Domain Specific Query Language. After several discussions with our marketing and technical personnel, we converged on the fact that a query language can be proposed but it has to be simple with a natural language-like grammar. We designed

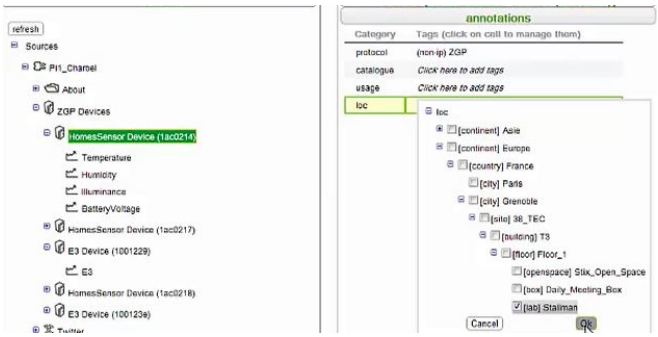

Fig. 4. Semantic Tagging through the Commissioning Tool

[4]project-haystack.org

and proposed a query language[5] which takes into account a combination of tags and expressions to filter the data. SQenIoT is capable of handling the following types of queries:

i) Search queries that include any block of semantic tags along with a block of filtering expressions. For example, the following query can be handled by SQenIoT. *Search Device protocol:ZigBee and quantity:temperature and location:Lab101 with value* > 22. The protocol and location tags are attached to the device level, however, the quantity is attached to the variable measuring temperature. SQenIoT will take into account the variables of a device when performing the search. For this reason, the tagging should be as much accurate as possible. Moreover, the query engine allows to combine common elements from the data model to be used from the query such as name, value and unit.

ii) Basic aggregation functions are supported by SQenIoT such as Min, Max, Sum, Avg. Such queries are used to retrieve, for example the average temperature of all sensors in a building or the sum of consumed energy by usage or by zone like *Sum Variable measures:ActiveEnergy and usage:Lighting and location:Building2*.

iii) Subscribe function allows our customers to handle situations where devices appear/disappear at a specific location for a given measurement type. It also comes handy to generate alerts and notifications when an event of interest occurs, e.g. an event on change compared to a user defined threshold. Such subscriptions are configurable according to the business requirements. For example, it is possible to subscribe to an event by checking the value of a temperature sensor every 10 minutes for the next month at a particular location and generate an alert every time the temperature value is higher than a specific value. SQenIoT is also capable of collecting data and pushing to Schneider Electric Cloud or to a remote REST endpoint. For example : *Collect Device (quantity:temperature or quantity:humidity) or (quantity:ActiveEnergy and usage:mainMeter)) and @loc:floor1 From 2016-03-21 To 2017-03-21 every 00:10:00 towards http://MyRestEndpoint.com/rest*. This query collects the two types of devices on floor1. SQenIoT will push this data for a year every 10 min to the indicated REST endpoint.

iv) Localized Inference SQenIoT implements an inference engine to reason and infer additional knowledge. The inference feature is specified at query time. As shown in Fig. 4, the given device was annotated with *Stallman* lab which is located in *Floor_1* and building *T3*. An example of a query relying on the inference is *Search device @location:T3*. Although there is no device tagged with location:T3, this query will still return the device after applying the inference (since Stallman is located in T3). The special character @ on a tag is a request to apply the inference feature. Currently the inference is applicable on the location and the device type tags such as *@type:sensor* which is the parent class of all the sensors.

[5]sites.google.com/site/charbelweb/query/QueryGrammar.g

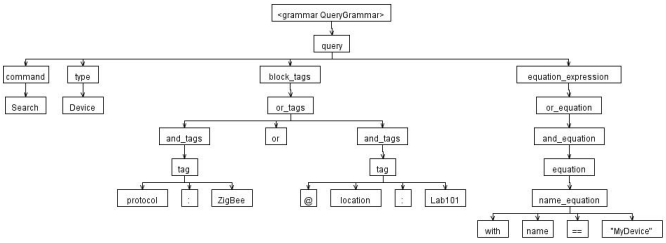

Fig. 5. Abstract Syntax Tree Query Example

IV. Implementation & Evaluation

SQenIoT has been implemented as a standalone software component and can use any annotated data model. In this section we present our implementation in detail. First we discuss the implementation choices we made, and then we present our setup and the performance metrics. We end this section with a discussion of the results.

A. Implementation Choices

Figure 3 shows the implemented SQenIoT components, i.e. query decoder, query evaluator & ontology handler.

1) Query Decoder (QD): It relies on our domain specific query language. Its grammar follows the LL1 derivation. QD uses the Antlr3[6] library which takes as input our grammar[5] and then generates the required code to verify the queries lexical and syntactic conformity. Once validated, Antlr generates an Abstract Syntax Tree as shown in Fig. 5.

2) Query Evaluator (QE): QE implements our query algorithm and performs the following functions : search the devices according to user queries, publish data to remote platform, handle subscriptions and aggregation functions such as Min, Max, Sum, and Average. QE looks up for the devices and variables in the data model using the tags specified in the query and then compares them with the conditions specified in the expressions block (e.g. $value > 25$). When inference is required on a tag QE invokes the ontology handler module.

3) Ontology Handler (OH): OH implements a simple inference engine, instead of relying on existing ones like Apache Jena[7] due to less resources of the gateways. The inference capability is specified on a tag. For instance, as shown in Fig. 4, the given device was annotated with *Stallman* which is located in *Floor_1* which in turn is located in building *T3*. OH relies on the transitive nature of the object properties between the instances in the case of the location. On the device catalogue ontology, OH would rely on the hierarchical relations between the device classes. OH uses ontologies such as Units and Physical Quantities are already deployed on the gateway. The Location ontology is built during the commissioning phase using a GUI and stored on the gateway.

B. Implementation Setup

The implementation setup is shown in Fig. 3. SQenIoT is implemented on our new hardware platform called ECP. The

[6]www.antlr3.org
[7]www.jena.apache.org

ECP board consists of a *SoCA9*, based on dual core Cortex A9 chip clocked at 900Mhz with 1GB RAM. It runs Linux[8] for embedded systems.

We implemented a Random Query Generator (RQG) to associate different combinations of tags and expressions to handle various possible queries. Each configuration consists of d devices, where $d \leq 1000$. Each device has two tags: location and protocol with one possible value for each tag, e.g. a device can have tags *location:Grenoble, protocol:Zigbee*.For variables, we randomly associate between 1 to 5 variables to each device. Each variable has quantity tag with random value.

C. Performance Metrics

The performance of SQenIoT is assessed in terms of the following metrics: Query Decoding Time (QDT), Query Evaluation Time (QET) and Inference Time (INFT). QDT and QET are measured in Microseconds while INFT is measured in Milliseconds All are average values of 100 iterations. We do not consider the network or communication delays.

QDT measures the time difference between the reception and the processing of the query by the QD component. It includes the time taken to parse the query according to our grammar and to generate the AST. Each set contained 1000 random queries created with N tags and M equations where $N \in [1-12]$ and M is either 0, 1 or 2.

QET measures the time taken to evaluate conditions specified in a given query along with the returned response. For QET, we executed two types of queries: Search Device Queries (SDQ) and Search Variable Queries (SVQ). For SDQ type, we generated random queries containing T Location + T Protocol tags. For SVQ type, we generated random queries containing T Quantity tags. For both query types, $T \in [1-6]$.

INFT measures the difference between the evaluation time of a query that uses inference against the one that does not. For instance, in our Location ontology, Stallman and Floor1 are located in T3 building, so the query *Search device @Location:T3* is equivalent to *Search device Location:T3 or Location:Stallman or Location:Floor1* query.

D. Results

Fig. 6 shows the QDT. Our results show that the time to decode a query is negligible. The minimum time is about 0.3 ms for a query containing a single tag. For more complex queries, the QD takes on average 0.033 ms and 0.045 ms extra for each additional tag and equation respectively.

Figure 7 shows that the QET, in SDQ, is a linear function of the configuration size and the number of tags used. In the worst case, (850 devices with 12 tags), the average QET is ≈85 ms. We found that the presence of equations in SDQ increases its QET We also determined that the increase in QET also depends on the number of returned results. For example, for 180 returned results, when there are 500 devices, QET increases by ≈0.2 ms per result.

For instance, consider that the query *Search device location:Europe and protocol:ModBus* returns Y results in a configuration of 500 devices in $T0$ ms. Then the query *Search device location:Europe and protocol:ModBus with value* > 25 will be executed in about $(T0 + 0.2 * Y)$ ms. The same results were obtained for the SVQ case.

[8]www.windriver.com/products/linux/

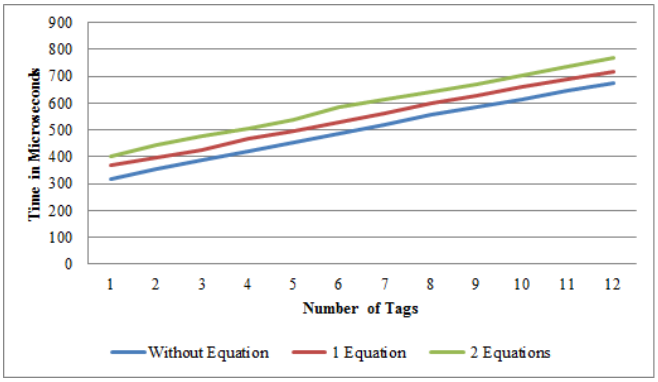

Fig. 6. Query Decoding Time (QDT)

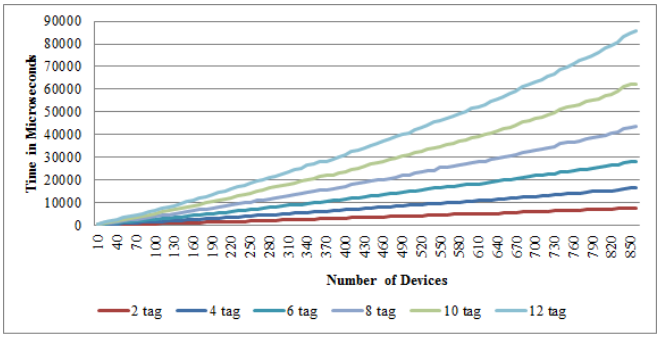

Fig. 7. Query Evaluation Time (SDQ Type)

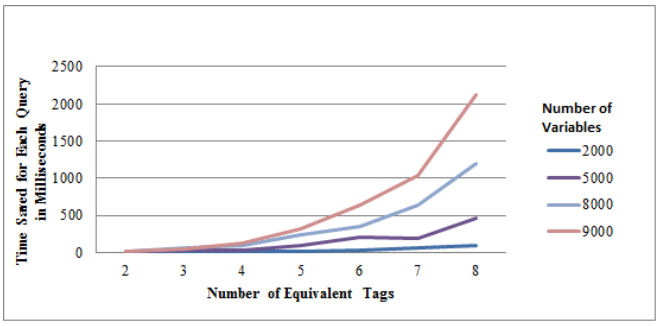

Fig. 8. Time Saved by using Inference

In this work, we found that the inference capability can also help to reduce the QET. To evaluate INFT, we created random configurations of sizes ($k * 1000$ where $k \in [1-9]$) and executed predefined queries using inference over 2, 3, and up to 8 tags. For each query, we also executed its equivalent query (without inference) to get the difference.

Figure 8 shows the time saved by using inference We observe that inferring concepts from ontologies takes less time than the parsing of tags and conditions of individual queries one by one. This is especially true when the number of tags deduced by the inference are large enough.

Other type of queries that can be handled by SQenIoT

include : *i) aggregation*, *ii) subscribe*, and *iii) collect* queries. Aggregation queries require computation of the aggregated values on the results returned by the search queries. For example, the query *avg variable quantity:Temperature* is evaluated by computing the average of the values returned by the query *Search variable quantity:Temperature*. The subscribe and collect queries are, in fact, periodic tasks which send the results of a search query to the defined targets (e.g. a remote application). So, we can generalize the previous evaluation results of the search queries to these specific query types.

The SQenIoT implementation fulfills the identified requirements in Section II and has been tested at several pilot sites.

V. Related Work

There have been various efforts in providing semantic-aware gateways in different domains. However none of them fulfills our requirements identified in Section II.

The authors in [7] discuss a unified semantic engine for IoT and Smart Cities. The issues like data interoperability, reasoning, abstraction, scalable, unified access through web services, secure, and real-time are discussed. However, the work does not provide any details regarding the implementation or the technologies that are used to address the identified issues.

The authors in [8] present a semantic smart gateway framework to achieve a loosely-coupled interoperability for interconnection between devices from heterogeneous vendors and to allow 3rd party application developers to write applications for these devices. The proposed semantic smart gateway has 'on-the-fly' ontology learning and ontology alignment features. While it tackles the device heterogeneity, the authors have suggested to use a SPARQL end-point for querying which is not a good choice for embedded systems. This work also lacks any implementation details and results.

The work in [9] is related to industrial domain thus is comparable to our work. The authors propose a solution to tackle the typical industrial use cases, e.g. industrial field service where service engineers use plethora of tools to identify issues of the installed devices. The semantic aspect only deals with the mapping of the SOAP bindings to the networking platform. The proposed gateway abstracts network services and translates them to a standard DPWS [10] interface.

In [11], a SOA-based architecture is presented to deal with the commonly operational disruptions (device break down or malfunctions) in industry. The proposed gateway contains a *semantic assistant* that uses semantic reasoning to find replacement devices using semantic identification of devices, services, and by mapping their features. However, there is no support for querying for the devices and their solution is tailored to deal with only a particular use case.

The authors in [5] propose a Haystack Tagging Ontology (HTO) to complement the haystack entities and to be able to query the data. The authors propose to annotate data using HTO, and then by using a set of rules, generate the knowledge (ontology) in order to query it. Transformation rules, written in SPARUL/SPARQL, are used to populate the knowledge. However, according to the authors, the transformation rules are not always accurate. Moreover, using transformation rules and SPARQL in embedded devices/gateways is not scalable and aim to develop a lighter version of their approach in future.

The work in [12] proposes to use embedded devices for storage and processing of RDF data. An in-network query processor is presented to efficiently handle RDF data and to execute SPARQL queries. This processor splits input queries into sub-queries and sends them for execution on local devices.The partial results are collected by the query processor and the final answer is computed. However, it is challenging to use embedded devices for forwarding intermediate results to the base station.

VI. Conclusion

In this paper we presented SQenIoT allowing Schneider Electric facility managers to query their devices in order to retrieve details of the environment and to subscribe to events and get notifications based on their preferences. We implemented SQenIoT in our next generation hardware platform and evaluated its performance. We have also identified several future work items. Searching and aggregation queries are useful for a first step, however, providing means to control the behavior of these devices based on their status or application requirements is also requested. We would also like to improve the query algorithm that we used in this work to efficiently deal with large number of devices. On the multi-level ontology layers, we look forward to integrate Haystack as a specialised ontology to enable cross domain data analytics.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.
[2] J. Ye *et al.*, "Semantic web technologies in pervasive computing: A survey and research roadmap," *Pervasive and Mobile Computing*, 2015.
[3] I. Khan *et al.*, "A data annotation architecture for semantic applications in virtualized wireless sensor networks," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, 2015.
[4] E. Kharlamov *et al.*, "How semantic technologies can enhance data access at siemens energy," in *The Semantic Web–ISWC*, 2014.
[5] C. Victor *et al.*, "An ontology design pattern for iot device tagging systems," *5th International Conference on the Internet of Things*, 2015.
[6] 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.
[7] A. Gyrard and M. Serrano, "A unified semantic engine for internet of things and smart cities: From sensor data to end-users applications," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*. IEEE, 2015, pp. 718–725.
[8] K. Kotis *et al.*, "Semantic interoperability on the web of things: the semantic smart gateway framework," in *Complex, Intelligent and Software Intensive Systems, Sixth International Conference on*, 2012.
[9] T. Riedel *et al.*, "Using web service gateways and code generation for sustainable iot system development," in *Internet of Things (IOT)*, 2010.
[10] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *Industrial informatics, IEEE transactions on*, 2005.
[11] G. Cândido *et al.*, "Enhancing device exchange agility in service-oriented industrial automation," in *Industrial Electronics (ISIE), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–6.
[12] D. Boldt, H. Hasemann, A. Kröller, M. Karnstedt, and C. von der Weth, "Sparql for networks of embedded systems," in *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. IEEE, Dec 2015, pp. 93–100.