EnergonQL: A Building Independent Acquisitional Query Language for Portable Building Analytics

Fang He[1], Cheng Xu[1], Yanhui Xu[1], Dezhi Hong[2], Dan Wang[1]
[1]Department of Computing, The Hong Kong Polytechnic University
[2]Department of Computer Science and Engineering, University of California, San Diego

ABSTRACT

Emerging building analytics heavily rely on data-driven machine learning algorithms. However, writing these analytics is still challenging: developers not only need to know what data is required but also where this data is in each individual building when writing applications. To bridge this gap between analytics and the actual resources in buildings, we present EnergonQL, a building independent acquisitional data query language that extracts data for building analytics with a declarative query processor. EnergonQL provides logic views of building resources that universally apply to all buildings, thus allowing portable building analytics across buildings. We evaluate EnergonQL with four different building analytics and show that with EnergonQL the line-of-code and development efforts can be effectively reduced.

CCS CONCEPTS

• **Information systems → Query languages**.

KEYWORDS

Smart buildings, data analytics, Declarative query language

ACM Reference Format:
Fang He[1], Cheng Xu[1], Yanhui Xu[1], Dezhi Hong[2], Dan Wang[1]. 2020. EnergonQL: A Building Independent Acquisitional Query Language for Portable Building Analytics. In *The 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '20), November 18–20, 2020, Virtual Event, Japan*, 4 pages. https://doi.org/10.1145/3408308.3427979

1 INTRODUCTION

Data-driven building analytics have proven to be effective in profiling, control and fault prognosis of building systems, reducing energy footprints of buildings as well as improving the comfort for occupants [10, 12, 15]. While promising, developing building analytics nowadays requires not only expertise in data analytics, but also knowledge about each individual building. For example, developing a fault detection tool for air handling units (AHUs) requires the knowledge about what algorithms fit, components in the AHU involved, and how they function, as well as how to obtain the required data in a particular building. Currently, such expertise and knowledge cannot be easily translated into a *standardized* workflow using existing machine learning (ML) platforms such as scikit-learn [11]; developing and deploying building analytics still requires tremendous manual effort on a per building basis. This greatly impedes the adoption of smart building technologies.

There are recent efforts dedicated to standardizing resource and information organization in buildings and to facilitating access to building data [2, 4, 8]. While these solutions significantly simplify building data access, developers still need domain knowledge about building particulars before they can leverage these tools. Ideally, a developer should be able to write a few simple SQL-like queries to locate the data they need and then implement analytics in a *building-agnostic* manner. To this end, we present EnergonQL, a query language that extracts data for building analytics via a declarative query processor, agnostic to building specifics such as its subsystems and structure. EnergonQL builds upon a standardized organization of building resources and data (e.g., using the Brick Schema) and further provides developers with the capability to locate desired resources and retrieve their data. This is achieved by a new layer of abstraction of logic views of buildings and automated locating the required data through building ontology traversal. This way, a developer can develop and deploy building analytics without the knowledge about the underlying details of a building, and the building analytics become portable across buildings.

We evaluate EnergonQL using four building analytics, namely, chiller profiling [15], energy consumption prediction [1], fault diagnosis and detection [10, 13, 14], and building integrated control [12]. We show that with EnergonQL, the line-of-code and the development efforts can be significantly reduced.

2 BACKGROUND AND RELATED WORK

Building Analytics. In recent years, data-driven machine learning approaches have proven effective for numerous building applications. Sensory data from building systems (e.g., chillers, AHU) are periodically collected and stored in a database. To develop an ML model for building analytics, the first step is to retrieve essential data (e.g., power, airflow) from the database. The remaining steps follow a standard ML workflow, including data cleaning (e.g., missing value and outlier) and transformation (e.g., normalization and resampling), feature engineering (e.g., feature selection and generation), and model establishment using a specific algorithm. Note that the first step is highly building dependent, as the sensor name, building system structure, etc, vary across buildings. Thus, it requires the developers to have in-depth knowledge of individual buildings, making the development of building analytics time-consuming.

Standardized Management of Resources and Data in Buildings. There are existing efforts to standardize the management of

Fang He and Cheng Xu contribute equally to this work.

BuildSys '20, November 18–20, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-8061-4/20/11…$15.00
https://doi.org/10.1145/3408308.3427979

and access to resources and data in buildings. Brick [2] and BuildingSync [8] are examples to unify vendor-specific nomenclature into a standard schema. The developers can be relieved to handle vendor-specific nomenclature when retrieving data. Platforms such as Mortar [4] provide standard programming interfaces for developers to *query* data points in a building.

Yet, the developers still need to have the knowledge about which sensing and/or control devices they need the data points for, and the specific structure to retrieve the data (see a concrete example in §3.1), which is sometimes well beyond the forte of an algorithm or analytics developer. These motivate our design of EnergonQL.

Declarative Programming. Declarative programming is a paradigm that expresses the logic of a computer program without describing its control flow, in contrast to imperative programming that specifies how a program operates in a step-by-step manner. Typical examples of declarative languages are SQL [9] for relational databases, XQuery [3] for XML documents, SPARQL [5] for RDF triples, and more. Recently, building entities and relationships are defined as RDF triples in the Brick Schema, which support declarative SPARQL queries. Our proposed EnergonQL introduces an abstraction on top of resources organized in Brick so that developers can easily extract data for building analytics without knowing the underlying details of buildings.

3 ENERGONQL DESIGN

3.1 A Motivating Example

Building analytics are data-driven; a simple means to access data is critical for the developers to focus on analytics-specific development (e.g., the features and the algorithms for model establishment). As said, while solutions such as Brick and Mortar provide unified query APIs to find data in a building, developers still need to have nuanced knowledge about the building at different levels. We illustrate this by an example of retrieving the sensor data of a chiller, which is needed by many predictive and diagnostic analytics.

The overall data of a chiller includes its own data and that of the associated sub-components, such as pumps and compressors. Figure 1 shows an example that extracts such data via SPARQL queries. The developer needs to have knowledge of the entire chiller system as well as the sensors installed. We see that Line 5 extracts the power data of the chiller; Line 7 - 10 and Line 12 - 15 go to the pump and compressor, the sub-components of the chiller; and Line 10, 15 extract the power data of the pump and compressor, respectively. This is for one particular type of chiller, and different chiller systems can contain different sub-components, as illustrated in Figure 2. A chiller may or may not have a compressor sub-component either because sensors are not installed at the compressor level[1] or the chiller does not have a compressor at all. A developer has to have such knowledge and it is building dependent, yet this is irrelevant from the viewpoint of analytics development as the analytics just need the overall data of the chiller.

The need to deal with such detail at different levels incurs overhead in analytics deployment and remains a barrier to algorithmic and analytics advances for scientists. Current solutions require developers to have both analytics- and building-specific knowledge;

[1]Note that a chiller profiling analytic can still be developed, only with lower accuracy.

```
1  SELECT ?cps ?pps ?cpps WHERE {
2      ?chiller rdf:type/rdfs:subClassOf* brick:Chiller .
3
4      ?chiller brick:hasPoint ?cps .
5      ?cps rdf:type/rdfs:subClassOf* brick:Power_Sensor .
6
7      ?chiller brick:hasPart ?pp .
8      ?pp rdf:type/rdfs:subClassOf* brick:Pump .
9      ?pp brick:hasPoint ?pps .
10     ?pps rdf:type/rdfs:subClassOf* brick:Power_Sensor .
11
12     ?chiller brick:hasPart ?cp .
13     ?cp rdf:type/rdfs:subClassOf* brick:compressor .
14     ?cp brick:hasPoint ?cpps .
15     ?cpps rdf:type/rdfs:subClassOf* brick:Power_Sensor .
16 }
```

Figure 1: SPARQL Query for Chiller Profiling

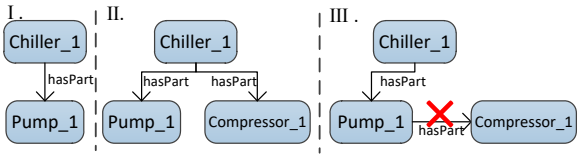

Figure 2: Chiller Structure (in RDF) in Different Buildings

this impedes *usability*. The data extraction query needs to be revised if new sensors are installed or new components are added; this impedes *extensibility*. Lastly, the structure of building (sub)systems differs from one building to another, and this impedes *portability*.

3.2 EnergonQL Design

In designing EnergonQL, the key challenge lies in how to map analytics requirements to the actual resources in a building. We need a proper *abstraction* of analytics needs and a tool to automatically *map* such abstraction to the concrete resources (e.g., equipment, devices, sensors) in a building. To create a proper abstraction, we carefully analyze seven building analytics (see Table 1), representing profiling, model predictive control (MPC), and fault diagnosis and detection (FDD), which are used for maintenance and building operation efficiency (e.g., energy efficiency) improvement.

We observe that data required by building analytics fall into two categories: subsystem (e.g., AHU, chiller, pump) and functionality (e.g., temperature, humidity, power), as summarized in Table 1. These can be considered as two logical views of building data. Thus, we categorize building data into these two classes (or two logical partitions) so that analytics can be built upon them. As building analytics require final integration, we further define common set operations (such as union and intersect) on building data, akin to relational algebra. Analytics developed on top of such an abstraction are uniform across different buildings, hence portable. On top of these two categories of building resources, we designed the Energon query language, *EnergonQL*, following the concept of object database, comprises *select-from-where* expressions. The basic primitives are objects and functions, where objects can be bounded to buildings and functions can be used as predicate conditions.

We present an example EnergonQL query in Figure 3. The **SELECT**, **FROM**, and **WHERE** clauses specify traversal, bounding, and selection, respectively. Specifically, the data resource from building 'PU' is selected and bounded to object *B*, from which the chiller

Table 1: Subsystems and functionalities commonly required in different applications.

Subsys. / Func.	AHU system	VAV system	Chiller System	Weather	Zone	Lighting	Blind
Temperature	(3)(5)	(6)	(1)	(1)(2)(3)	(2)(3)		
Humidity	(5)	(6)		(2)			
Pressure	(5)	(3)(6)	(7)				
Flow Rate	(3)(5)	(6)	(1)(2)(7)	(3)	(2)		
Power	(3)	(3)	(1)(3)(7)				
Solar Radiance				(3)(4)			
Solar Angle				(4)			
Control Signal	(5)	(6)					
Setpoint	(5)	(3)(6)				(4)	(4)

Analytics	**Profiling**	**(1) Chiller Profiling [15];**
	MPC	**(2) PMV Prediction [6]; (3) ECP [1]; (4) Building Integrated Control [12]**
	FDD	**(5) FDD for AHU [10]; (6) FDD for VAV [13]; (7) FDD for Chiller [14]**

```
/* algebra to perform building traversals */
SELECT Chiller(B) * (Temperature(B) + Flow_Rate(B) + Power(B))
+ Temperature(B) * Weather(B)
/* list of buildings to determine boundings */
FROM Building B
/* predicate expressions to perform resource selections */
WHERE B.BuildingID = 'PU' AND B.Source = 'LOCAL'
/* predicate expressions to perform data selections */
FILTER B.TIME_STAMP >'20190801' AND B.TIME_STAMP <'20191231'
```

Figure 3: EnergonQL Query for Chiller Profiling

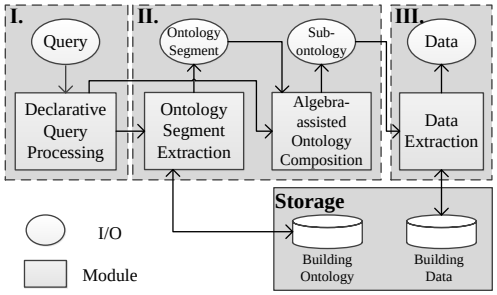

Figure 4: EnergonQL Query Processor

and weather system with various related nodes in B are traversed The **FILTER** clause further specifies the filtering conditions for data selection. In this case, data falling in the time window from 2019.08.01 to 2019.12.31 are selected.

3.3 EnergonQL Query Processor

Data in building database is deeply coupled with building specifics, and thus subject to the variations of buildings. To overcome this issue, we leverage building ontology, widely available in practice, to automatically extract the correct subset of data that can be used for building analytics.

Figure 4 overviews the architecture of EnergonQL query processor. It presents a standardized *execution procedure* for building analytics data extraction. There are three steps. First, the **Declarative Query Processing** module processes the query written by analytics developers and generate a query execution plan. Second, the **Building Independent Ontology Extraction (BIOE)** module extracts the sub-ontology out of the building ontology-based on user queries. There are two sub-steps. The *Ontology Segment Extraction (OSE)* module takes an existing building ontology (e.g., in standard Brick format) and extracts a set of ontology segments, corresponding to the two categories, sub-system and functionality. The *Algebra-based Ontology Composition* module takes the extracted set of ontology segments and performs operations specified in the query to derive a sub-ontology. Our operations follow algebra (e.g., Union, Intersection, etc), ensuring the operations to be correct and conflict-free. Finally, the **Data Extraction** module takes the sub-ontology for data extraction via standard programming interfaces.

The BIOE module contains an ontology traversal algorithm, which incurs certain complexity since the ontology can be big. We choose to offline decompose and cache the ontology into smaller sub-ontology, which effectively accelerates online extraction. Note that the state-of-the-art SPARQL can be used together with EnergonQL; it is efficient in retrieving data if the location of the data can be specified by the developer.

4 EVALUATION

We evaluate EnergonQL by implementing four building analytics.
Chiller Profiling (CP). Chiller profiling estimates the performance of chillers, which is called the Coefficient of Performance (COP) and can be used for maintenance, operation decisions, etc. Intuitively, COP indicates the amount of cooling load a chiller can output given a unit of electricity. In the past, physical models were developed for COP, and recently, data-driven ML models have shown better performance. We implement the ML model in [15].
Energy Consumption Prediction (ECP). In general, ECP establishes a (usually nonlinear) model with the control strategies as inputs and predicts the energy consumption according to the control strategies. An optimization algorithm then searches the control strategy space for the least-energy control strategy. We implement a specific ECP based analytics [1] on VAV control.
Fault Detection and Diagnosis for AHU (FDD-AHU). Traditional FDD methods follow rule-based models. Recently, there are data-driven ML models developed. We implement a specific multi-layer diagnostic model [7] for detecting multiple types of faults (e.g., damper stuck and cooling coil valve stuck) in AHU systems.
Building Integrated Control (BIC). BIC requires jointly control of multiple subsystems in a building to achieve indoor comfort. We

Table 2: Development Effort of Mortar and EnergonQL

Analytics	Method	Lines of Code	Development Time (min)
Chiller Profiling	Mortar	48	84.2
	EnergonQL	11 (-77.1%)	46.2 (-45.1%)
Energy Consumption Prediction	Mortar	49	75.2
	EnergonQL	13 (-73.4%)	38.2 (-49.2%)
FDD for AHU	Mortar	73	69.6
	EnergonQL	12 (-83.6%)	31.4 (-54.9%)
Building Integrated Control	Mortar	39	67.4
	EnergonQL	12 (-69.2%)	30.0 (-55.5%)

implement a typical BIC analytic [12] for maintaining indoor visual comfort via an integrated control of the indoor lighting system and the blind system in conjunction with outdoor sunlight. We recruited five developers in this evaluation study to implement the four analytics per the order we introduce them. All five developers are data scientists with limited knowledge of RDF semantics and buildings. We do not consider the learning time of background knowledge such as RDF, SPARQL, and task requirement of these four analytics, and only record lines of code for data extraction and the actual time spent on implementing the four analytics.

We compare the development effort (in terms of program length and development time) when using EnergonQL to that of Mortar.

Table 2 shows the results. We see that the total lines of code are reduced by 77.1%, 73.4%, 83.6% and 69.2%, respectively. With EnergonQL developers spent less time on implementing the analytics. The time spent was reduced by 45.1%, 49.2%, 54.9% and 55.5% compared to using Mortar. We also see that as they develop one analytic after another, the developers become more proficient over time. The speedup when using EnergonQL is more significant. Specifically, for EnergonQL, the development time is reduced from 46.2 minutes to 30.0 minutes (35.06% reduction), while using Mortar, the development time is reduced from 84.2 minutes to 67.4 minutes (19.95% reduction). This is partial because that for each analytic in Mortar, the developer needs to spare extra time to understand the details of every new building ontology.

We also examine the execution time of EnergonQL as well as of data extraction and model traning in Figure 5. Note that EnergonQL is the very first step of building analytics and does not incur overhead to the analytics, since the model training dominates the execution time as expected.For example, in the chiller profiling, model training accounts for 80.4% of execution time. We comment that the EnergonQL execution overhead can increase if the ML model only needs special-purpose data (e.g., CO) from a fraction of subsystems (e.g., a subset of VAV terminal boxes). To avoid ontology traversal, such cases may be handled with customized pre-processing. We plan future studies to investigate such a problem.

5 CONCLUSION

Nowadays, a key obstacle to building analytics is that developers need analytics-specific knowledge for application development and building-specific knowledge for building resource extraction. In this paper, we presented EnergonQL, a building structure independent query language that can support building analytics development

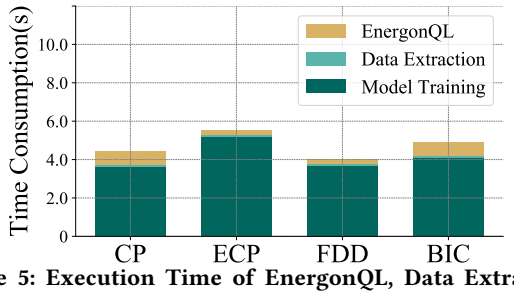

Figure 5: Execution Time of EnergonQL, Data Extraction and Model Training

with an abstraction that decouples the building resources from analytics development. With EnergonQL, developers can extract data with limited building-specific knowledge, and queries are portable across buildings. We evaluated EnergonQL with four building analytics showing that EnergonQL can reduce analytics effort in terms of lines of code and development time.

6 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their constructive comments. Dan Wang's work is supported in part by ITF-UICP UIM/363, ITF-ITSP ITS/070/19FP, CRF C5026-18G, GRF 15210119, PolyU 1-ZVPZ. Dezhi Hong is supported in part by National Science Foundation 1940291 and CA-2040727.

REFERENCES

[1] A. Afram, F. Janabi-Sharifi, A. S. Fung, and K. Raahemifar. 2017. Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system. *Energy and Buildings* 141 (2017), 96–113.

[2] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M. Kjærgaard, M. Srivastava, and K. Whitehouse. 2016. Brick: Towards a Unified Metadata Schema For Buildings. In *Proc. ACM BuildSys'16*. 41–50.

[3] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. 2002. XQuery 1.0: An XML query language. (2002).

[4] G. Fierro, M. Pritoni, M. Abdelbaky, P. Raftery, T. Peffer, G. Thomson, and D. Culler. 2018. Mortar: An Open Testbed for Portable Building Analytics. In *Proceedings of the 5th Conference on Systems for Built Environments (BuildSys '18)*. 172–181.

[5] S. Harris, A. Seaborne, and E. Prud'hommeaux. 2013. SPARQL 1.1 query language. *W3C recommendation* 21, 10 (2013), 778.

[6] R. Z. Homod, K. S. M. Sahari, H. A. F. Almurib, and F. H. Nagi. 2012. Gradient autotuned Takagi–Sugeno Fuzzy Forward control of a HVAC system using predicted mean vote index. *Energy and Buildings* 49 (2012), 254–267.

[7] J. Liang and R. Du. 2007. Model-based fault detection and diagnosis of HVAC systems using support vector machine method. *International Journal of Refrigeration* 30, 6 (2007), 1104–1114.

[8] N. Long, J. DeGraw, M. Borkum, A. Swindler, K. Field-Macumber, E. Ellis, et al. 2018. *BuildingSync®*. Technical Report.

[9] J. Melton and A. R. Simon. 1993. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[10] M. Najafi, D. M. Auslander, P. L. Bartlett, P. Haves, and M. D. Sohn. 2012. Application of machine learning in the fault diagnostics of air handling units. *Applied Energy* 96 (2012), 347–358.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12 (2011), 2825–2830.

[12] E. Shen, J. Hu, and M. Patel. 2014. Energy and visual comfort analysis of lighting and daylight control strategies. *Building and Environment* 78 (2014), 155–170.

[13] S. Wang and J. Qin. 2005. Sensor fault detection and validation of VAV terminals in air conditioning systems. *Energy Conversion and Management* 46, 15-16 (2005), 2482–2500.

[14] K. Yan, Z. Ji, and W. Shen. 2017. Online fault detection methods for chillers combining extended kalman filter and recursive one-class SVM. *Neurocomputing* 228 (2017), 205–212.

[15] Z. Zheng, Q. Chen, C. Fan, N. Guan, A. Vishwanath, D. Wang, and F. Liu. 2018. Data Driven Chiller Sequencing for Reducing HVAC Electricity Consumption in Commercial Buildings. In *Proc. ACM e-Energy '18*. 236–248.