Energon: A Data Acquisition System for Portable Building Analytics

Fang He[1], Yang Deng[1], Yanhui Xu[1], Cheng Xu[2], Dezhi Hong[3], Dan Wang[1]
[1]The Hong Kong Polytechnic University, [2]Johnson Electric, [3]University of California, San Diego
{fangf.he,yang2.deng,stephen.xu}@connect.polyu.hk
cheng.xu@johnsonelectric.com,dehong@eng.ucsd.edu,dan.wang@polyu.edu.hk

ABSTRACT

Emerging building analytics rely on data-driven machine learning algorithms. However, writing these analytics is still challenging—developers need to know not only what data are required by the analytics but also how to reach the data in each individual building, despite the existing solutions to standardizing data and resource management in buildings. To bridge the gap between analytics development and the specific details of reaching actual data in each building, we present Energon, an open-source system that enables portable building analytics. The core of Energon is a new data organization for building as well as tools that can effectively manage building data and support building analytics development. More specifically, we propose a new "logic partition" of data resources in buildings, and this abstraction universally applies to all buildings. We develop a declarative query language accordingly to find data resources in this new logic view with high-level queries, thus substantially reducing development efforts. We also develop a query engine with automatic data extraction by traversing building ontology that widely exists in buildings. In this way, Energon enables mapping of analytics requirements to building resources in a building-agnostic manner. Using four types of real-world building analytics, we demonstrate the use of Energon and its effectiveness in reducing development efforts.

CCS CONCEPTS

• **Information systems → Data access methods**.

KEYWORDS

Smart building, Data analytics, Machine learning, Declarative query

ACM Reference Format:
Fang He[1], Yang Deng[1], Yanhui Xu[1], Cheng Xu[2], Dezhi Hong[3], Dan Wang[1]. 2021. Energon: A Data Acquisition System for Portable Building Analytics. In *The Twelfth ACM International Conference on Future Energy Systems (e-Energy '21), June 28–July 2, 2021, Virtual Event, Italy*, 12 pages. https://doi.org/10.1145/3447555.3464850

e-Energy '21, June 28–July 2, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8333-2/21/06…$15.00
https://doi.org/10.1145/3447555.3464850

1 INTRODUCTION

Building analytics—that is, using data to develop machine learning (ML)-based methods for operation and control of building systems—have proven to be effective in reducing energy footprints and operational costs, and improving the maintenance efficiency of buildings [23, 25, 30, 38, 39, 41, 44]. While promising, developing building analytics requires not only expertise in machine learning and building science but also deep knowledge about the configuration of each individual building, as each building is unique. For example, developing a smart lighting control system in a particular building requires the knowledge about what algorithm fits, what control points are available in the space (switch, blinds, etc), and how to access these points in the specific configuration of each individual building. Currently, developing and deploying analytics still requires tremendous manual effort on a per building basis because of the varying system details in each building. This greatly impedes the adoption of building analytics.

Recently, efforts have been dedicated to standardizing the organization of resources and data in buildings and to facilitating access to building data. Schemas such as Brick [4] and BuildingSync [26] provide a standardized naming convention and hierarchy to name, organize, and manage resources in buildings. Solutions such as SEED [3] and Mortar [16] provide a uniform way to organize and query data. These solutions significantly simplify access to building data. Yet, building analytics apply to a building subsystem (e.g. a lighting system, a chiller system), and there is still a gap between the ability of practitioners to access data in an organized way at such level and to conveniently realize building analytics. Developers need to spend a considerable amount of time mapping their algorithm to the actual data resources in a specific building before they can leverage existing tools such as Mortar. Ideally, developers would only need to write a few simple SQL-like queries to locate the data and/or control points that they need and implement analytics, regardless of the building particulars. We envision a system that simplifies and standardizes the data acquisition process for building analytics so that the analytics become portable across buildings.

In this paper, we present Energon, a system that allows for the acquisition of data to develop analytics in a *building-agnostic* manner. Energon builds upon a standardized organization of building data resources (e.g. using the Brick Schema) and further provides developers with the capability to locate and retrieve desired data via *declarative queries*. In this way, a developer can develop and deploy analytics without the need to know the underlying building details, greatly simplifying and expediting the development process.

In designing Energon, the key challenge lies in how to efficiently map analytics requirements to the actual data resources in a building. This mapping process is currently deeply coupled with building

specifics, due to variations in buildings. For example, in one building the luminance of a lighting system is solely determined by the setpoint of the lamp, while in another building it could be determined by the setpoints of both the lamp and the ballast. Thus, to develop a data-driven smart lighting control system [13, 30], one needs to have the light level readings of the lighting system, and nuanced details about how to control the lighting system, which makes the process building-dependent and unnecessarily onerous.

To overcome this challenge, Energon presents a new *data organization* method for building as well as *tools* that can effectively manage building data and support building analytics development. More specifically, we propose an *abstraction* that divides the data resources in a building into two logic categories: subsystem (e.g. lighting, chiller, pump) and functionality (e.g. illuminance, temperature, power). Thus, the building data resources can be categorized into these two classes upon which analytics can develop. We then develop a *declarative query language*, with which developers can compose high-level queries to find the desired subsystem of a certain functionality. Our query language allows for common set operations (such as union, intersect, and join) on building resources, akin to relational algebra, which expedites analytics development and ensures that queries are complete and consistent. To execute these queries, we develop a *query engine* with a building-agnostic ontology extraction procedure, which automatically extracts the elements in a building ontology and assigns them into the two abstract categories. In this way, the query engine can automatically map the input of analytics to the concrete resources in a building. Energon can *adapt to* building specifics and *hide* building-dependent and analytics irrelevant knowledge. Analytics developed upon Energon are uniform across different buildings, hence portable.

We evaluate Energon and demonstrate its use in multiple types of analytics in real buildings. In summary, the contributions of this paper are as follows:

- We present a new abstraction of building resources so that developers do not need to have building-specific knowledge when developing building analytics; thus, the development of application can be simplified. The abstraction comprises two logic categories of building resources, namely, subsystem and functionality.
- We present the design and implementation[1] of Energon to materialize the abstraction. We develop a declarative query language to hide building-dependent and analytics irrelevant knowledge. We develop a query engine to automatically extract data by traversing a building ontology that widely exists in buildings. We further develop an indexing structure to optimize query execution time.
- We develop four types of building analytics through Energon, and qualitatively show that the development process becomes simpler when using Energon.
- We quantitatively evaluate Energon with regards to program length and development time. We evaluate our optimization schemes designed for execution time in Energon.

2 RELATED WORK

Standardized Resource and Data Management for Building. There have been various attempts to facilitate the management of and access to resources and data in buildings. Brick [4] and BuildingSync [26] are examples of unified schemas for organizing and managing resources in buildings, which helps to locate resources in a vendor-agnostic way. However, a schema can efficiently navigate users to resources *only if* the user knows exactly where they can locate the data in a building. The question of what resources users need remains to be answered per analytics, for which rich experience and often domain knowledge of individual buildings is required. On another front, platforms such as SEED [3] and Mortar [16] provide standard programming interfaces to query and retrieve data points in a building, which greatly eases and expedites data management in buildings. Yet, again, the users need to have the knowledge about which sensing and/or control devices they need the data points for, but such knowledge is sometimes beyond the ability of an algorithm or analytics developer to provide.

Besides these research efforts, commercial platforms available such as BuildingOS [27], CopperTree [11], Entronix EMP [14], and Skyspark [31] are also available for smart building solutions. These platforms are well integrated with Building Automation Systems (BAS), along with tools for monitoring and visualizing building data and generating reports. Analytics are often achieved by setting up rules or implementing pre-programmed templates for alerts and finding trends. However, the use of such customized solutions is often limited to specific analytics (e.g. only for fault detection and diagnostics), and hence can be difficult to extend.

Building Analytics. In recent years, building analytics have been developed and have proven to be effective for use in various kinds of building systems. Building Integrated Control (BIC) jointly controls multiple systems in a building to manage the indoor environment. For example, to ensure indoor visual comfort, Shen et al. developed ML models to predict the indoor illuminance under different operations of lighting and blind systems [30]. ML models have been developed for Indoor Air Quality (IAQ) by using indoor air data (e.g. CO2 and air flow rate) [37] . Energy Consumption Prediction (ECP) models energy usage of one or multiple system(s) in a building to manage and save energy. Neural Network models and Bayesian Network and Decision Tree models were developed to monitor the load of appliances and to identify appliances in use [21][5]. Multi-task learning was proposed to predict the coefficient of performance (COP) of the chillers, and improve chiller sequencing [44]. Model Predictive Control (MPC) [1, 20] is a common solution to Heating, Ventilation and Air Conditioning (HVAC) control optimization. Both model-based [9, 22, 43] and model-free [10, 36] algorithms have been developed. Detecting faults can improve system performance efficiency. Several ML-based methods for fault detection have been developed, either through simulations [12] or by using historical data [6, 24, 38].

To develop these analytics, developers not only need the skills in machine learning and data analytics, but must also have to have deep domain knowledge on how to reach the necessary data in individual buildings. In this paper, we aim to bridge the gap between analytics and building data extraction by developing a data acquisition system that can simplify and standardize the data extraction workflow, making building analytics portable across buildings.

Declarative Programming. Declarative programming is a non-imperative programming paradigm that describes the goal rather

[1] Our code is open-sourced and available: https://github.com/KGBUSH/Energon

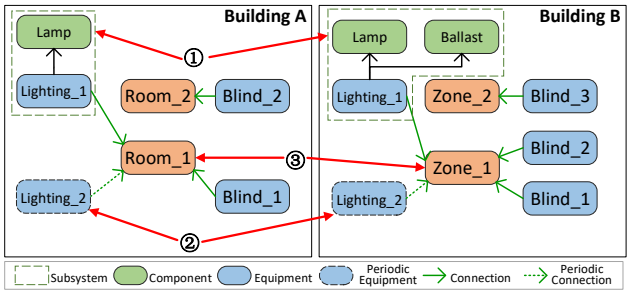

Figure 1: ① Spatial auxiliary knowledge; ② Temporal auxiliary knowledge; ③ Contextual auxiliary knowledge.

```
SELECT ?lpsp ?blsp WHERE {
    ?room(?zone) rdf:type/rdfs:subClassof* brick:Room(Zone) .
    ?lighting rdf:type/rdfs:subClassOf* brick:Lighting_System

    ?room(?zone) brick:isLocationOf ?lighting .

    ?lamp rdf:type/rdfs:subClassOf* brick:Luminance
    ?light brick:hasPart ?lamp .
    ?lamp brick:hasPoint ?lpsp .
    ?lpsp rdf:type/rdfs:subClassOf* brick:Setpoint

    ?blst rdf:type/rdfs:subClassOf* brick:Ballast .
    ?light brick:hasPart ?blst .
    ?blst brick:hasPoint ?blsp .
    ?blsp rdf:type/rdfs:subClassOf* brick:Setpoint .
    }
```

Data Tracing Flow

Figure 2: Example of a SPARQL query for extracting setpoints for BIC analytics (blue arrows show the steps involved in locating the desired data. Changes (marked in red) to the program are needed for another building due to variation in context (e.g. room to zone) and the addition of new components (e.g. Ballast setpoints).

than illustrates all the intermediate steps. In the database community, typical examples of declarative languages are SQL for relational databases [28], XQuery for XML documents [7], SPARQL for RDF triples [18], HiveQL for NoSQL databases [34], and more. In the building science community, building entities and relationships are defined as RDF ontology in the Brick Schema, which supports declarative SPARQL queries. As building ontology and the corresponding sensor data are usually stored separately, to retrieve the actual data, a two-step procedure is required: first, SPARQL queries are composed *per analytics* to search a building ontology for desired resources; then, in another step, the actual time-series data retrieval for these resources is performed, as is done in Mortar [16]. Thus, developers need to know the building ontologies which vary in different buildings. By contrast, Energon introduces an abstraction on top of the resources organized in Brick so that developers can easily extract building data without knowing the underlying details. To the best of our knowledge, Energon is the first system to provide the capability to write declarative queries over the abstraction of building data resources designed for analytics requirements.

3 MOTIVATION AND APPROACH

3.1 A Motivating Example

Building analytics involves the development of ML-based methods to analyze or predict the status of building systems so as to reduce wasteful or inefficient operations. An example is Building Integrated Control [13, 33], in which trains ML models are trained using data from multiple building systems for joint control, e.g. joint control of the lighting system and the blind system for indoor visual comfort.

Developing ML models for building analytics requires data. For example, in BIC, the setpoints for controlling the luminance level of the lighting systems and the slat angles of the blind systems, and readings on the intensity of outdoor sunlight, are used to develop a neural network to jointly consider these inputs to achieve indoor visual comfort [13]. To extract these data in a building, existing data acquisition systems, e.g. Mortar, need to deal with various types of building-dependent auxiliary knowledge. In particular, we show three types of auxiliary knowledge in Figure 1:

- Spatial (or structural) auxiliary knowledge: The structure of building subsystems often varies from building to building, e.g. in Figure 1 the lighting system in building A (i.e. lighting_1) only contains Lamp, the setpoint of which controls the luminance level, while in building B the setpoints of both the Lamp and the Ballast control the luminance levels.[2]
- Temporal auxiliary knowledge: a building subsystem may have a seasonal usage pattern, e.g. in Figure 1, lighting_2 is detached from a room to save energy because of ample light in the summer.
- Contextual auxiliary knowledge: a building subsystem may be used in different contexts, e.g. in Figure 1, lighting and blind systems are used in the context of "rooms" in building A but in "zones" in building B.

Such auxiliary knowledge, while necessary for pinpointing the correct data sources, is irrelevant to the analytics development and unnecessarily complicates the data acquisition process. For example, to develop the BIC ML model, a developer only needs to know the setpoints that control the luminance level of a room, yet whether such setpoints are for the lamp or the ballast or both is an unnecessary detail to the developer. This is also where existing building data acquisition systems fall short: they are designed for someone equipped with the knowledge to specify the exact point(s) to access [4, 16], rather than to provide users with the capability to make queries by high-level requirements such as *functionality*. We need a data acquisition that hides such building-dependent auxiliary knowledge.

Figure 2 shows an example of the extraction of the setpoints of a lighting system using an existing data acquisition system, Mortar. We see that 1) to find the setpoint, detailed knowledge about how to reach the data is needed; and 2) such details can differ from one building to another; thus the program needs to be revised according to the building.

3.2 Design Approach and Goals

A building-independent data acquisition system needs to evade the three types of building-dependent auxiliary knowledge when retrieving data. We would need to have a proper abstraction of the building data required by analytics and to build upon it, which is

[2] An electrical ballast is a device to limit the amount of current in an electrical circuit, and thus can control the luminance as well.

akin to how an Android program is developed regardless of the underlying specifications of the phone. To create such an abstraction, we examine a number of major categories of building analytics, as listed in Table 1. We observe that the data required by these analytics can be categorized according to the *subsystem* and *functionality* that they relate to. More specifically, building analytics are built upon building *subsystems*, e.g. lighting, chillers; the data associated with *functionality* are also required, e.g. temperature, power consumption, humidity. To reach the data of different subsystems, spatial, temporal, and contextual auxiliary knowledge is needed.

Thus, we abstract the data according to the subsystem and functionality involved, and then develop corresponding data extraction operations on these categories using declarative queries. Figure 3 shows an example of the extraction of data through referencing queries to the abstracted categories—obtaining the required data boils down to first locating the subsystem it involves and then finding the particular functionality.

We develop standard query operations (Select-From-Where, Union, Intersect, etc), which help to hide different types of auxiliary knowledge. For example, in Figure 3, to retrieve the setpoints of Lighting and Blind systems, the `Join` operation eliminates the contextual auxiliary knowledge on whether these two systems are reached via a *room* context (building A) or a *zone* context (building B).

To ensure the *completeness* and *consistency* of the query operations, we develop our query operations following relational algebra; for example, the Intersection, Union, and Join operations follow the convention of $(+, *, \bowtie)$. This algebra assists developers in writing error-free queries.

We develop a query engine to execute our query language. The core of the query engine is to leverage building ontology [4], which widely exists in buildings [19]. Building ontology embeds the organization of the building data, and we design a traversal algorithm to automatically reach the data. We further develop optimization schemes to improve the execution performance of the query engine.

In summary, our new query language provides an abstraction for the developer to easily specify the desired data for developing building analytics, and our system helps with the task of reaching the data. Overall, our system design follows the following goals:

Completeness. The system should provide a complete set of components that may be used to acquire data to support the development of building analytics. Components should be decoupled, so that each component can be used as an independent module.

Usability. Building analytics often requires specific domain knowledge and expertise in data analytics. Therefore, our system should be easy for users of different backgrounds to use, from analytics developers with ample knowledge about buildings to data scientists who know little about buildings.

Extensibility. The system should be designed in an extensible manner so that adding new functions to existing modules or adding new modules is simple and straightforward.

Portability. It should be possible to deploy building analytics across buildings without major changes to the implementation.

Table 1: Analysis of seven building analytics: Fault Detection and Diagnosis (FDD) for Chiller (FDD-Chiller) [6, 40], for Variable Air Volume (FDD-VAV) [35], for Air Handling Unit (FDD-AHU) [29, 42], Chiller Profiling (CP) [44], Building Integration Control (BIC) [30], Indoor Air Quality (IAQ) [37], and Energy Consumption Prediction (ECP) [2, 23].

Category / Analytics	*Subsystem*			*Functionality*	Query Operation
	Spatial	Temporal	Contextual		
FDD-Chiler	✓			✓	(1) (3)
FDD-VAV	✓			✓	(1) (3)
FDD-AHU	✓			✓	(1) (3)
CP	✓	✓		✓	(1)(2)(3)
IAQ	✓	✓	✓	✓	(1)(2)(3)(4)
BIC	✓	✓	✓	✓	(1) (3)(4)
ECP	✓		✓	✓	(1) (3)(4)
Query Operation	(1) Select, From, Where; (2) Filter; (3) Union, Difference, Intersect; (4) Join;				

```
/* ontology algebra to perform ontology traversal */
SELECT Lighting(A) JOIN Blind(A) * Setpoint(A)
/* list of buildings to determine ontology boundings */
FROM Building A
/* predicate expression(s) to perform ontology selections */
WHERE A.BuildingID = 'building_A' AND A.Source = 'Local'
/* predicate expression(s) to perform data selections */
FILTER A.TIMESTAMP >'20190801' AND A.TIMESTAMP <'20191231'
```

Figure 3: Energon query for extracting the setpoints for BIC.

4 THE ENERGON SYSTEM DESIGN

4.1 Design Overview

Figure 4 shows the architecture of Energon. Energon has three parts: **Energon Query Language**, based on SPARQL for the data acquisition process of building analytics; **Energon Query Engine** for Energon Query Language interpretation and execution; **Energon Ontology Index**, developed for system performance optimization. The **database** stores the building data and the building ontology.

The Energon Query Engine presents a standardized *execution procedure* for building analytics. There are three steps. First, the **Declarative Query Processing** (§4.2) module processes the query written by analytics developers and generates a query execution plan. Second, the **Building Independent Ontology Extraction (BIOE)** module (§4.3) extracts the essential sub-ontology out of the building ontology based on user queries. There are two sub-steps to the process. The *Ontology Segment Extraction (OSE)* (§4.3.1) module takes an existing building ontology (e.g. based on the standard Brick schema) from the ontology storage and extracts a set of ontology segments. The *Algebra-based Ontology Composition* (§4.3.2) module takes the extracted set of ontology segments and performs the algebraic operations specified in the query to derive a sub-ontology. Finally, the **Data Extraction** module takes the sub-ontology to extract data via standard programming interfaces. We also develop a **Energon Ontology Index** (§4.4) to organize the building ontology hierarchically to complement traditional RDF triples. This index can improve the system performance of the Energon Query Engine.

4.2 Energon Query Language and Processor

4.2.1 EnergonQL: A declarative query for building analytics. As discussed, in current systems, when used to support building analytics, the data retrieval process requires writing analytics- *and*

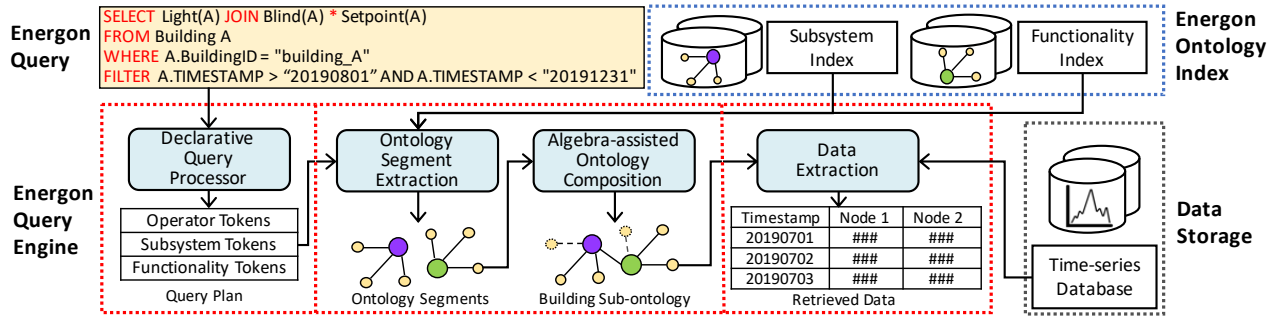

Figure 4: Energon System Architecture

Figure 5: Energon Query Processor

Table 2: Predicates of Subsystem Boundary

Predicates of Subsystem Boundary			
brick:hasPoint	brick:hasPart	brick:hasUuid	rdf:type

building-specific SPARQL queries; thus, it is time-consuming and not portable across buildings. Energon designs and implements a query engine which allows the user to write *declarative queries* to extract a subset of building data requiring *minimal* domain knowledge about the building and data extraction details. Following the concept of object database [8, 15], the Energon query language, *EnergonQL*, 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 illustrate the semantics of EnergonQL with an Energon query in Figure 3. The **SELECT**, **FROM**, and **WHERE** clauses specify ontology traversal, bounding, and selection, respectively. Specifically, the ontology from 'building_A' is selected and bounded to object *A*, from which the lighting and blind systems with setpoints-related nodes in *A* are traversed (more details about ontology traversal can be found in §4.3). The **FILTER** clause further specifies the filtering conditions for ontology and data selection. In this case, an ontology falling in the time window from 2019.08.01 to 2019.12.31 is firstly selected as the input of the query processor. Then in the data extraction stage, the data in the same time window are retrieved.

4.2.2 Query processor. Figure 5 shows the steps for processing queries in Energon. The *Query Parser* parses the query string into a set of tokens, called a *query syntax tree*. During the *Normalization* step, the predicates are normalized to the *Conjunctive normal form* (CNF). Then, the *Query Transformer* converts the normalized tree into a *query execution plan*, represented as ordered internal function calls accessing both the ontology and time-series databases for the building. The execution plan obtains a subset of building data that will further be used in data modeling.

4.3 Building Independent Ontology Extraction

A building ontology describes the building entities (e.g. sensors, chillers, pumps, etc.) and the relationships among them.

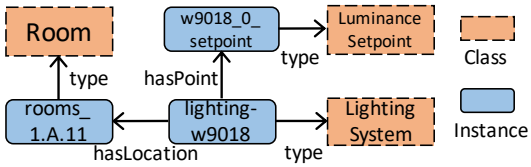

Figure 6: Ontology of a Building (Partial)

Definition 1. *A building ontology is a directed graph, comprising the building entities as nodes and the relationship between them as edges. This graph can be described by RDF triples* $\langle Sub, Pred, Obj \rangle$*, where the Subject (Sub) and Object (Obj) are nodes and the Predicates (Pred) are edges in the graph.*

Sub and *Obj* refer to concrete building entities, and *Pred* is the relationship among the entities (e.g. a lamp `is a part` of a lighting system, a lamp `has` a luminance setpoint). Figure 6 shows the building ontology of a real building. In this building, there is a room (named rooms_1.A.11) that is the location of a lighting system (named lighting-w9018) with a luminance setpoint (named w9018_0_setpoint). These can be described by RDF triples. For example, the triple <rooms_1.A.11, type, Room> indicates that rooms_1.A.11 is an instance of class Room, <lighting-w9018, hasLocation, rooms_1.A.11> indicates that the room (rooms_1.A.11) is the location of a lighting system (lighting-w9018), and so on.

Nowadays, building ontologies are widespread. A building ontology represents a complete view of a building's resources and data. A building analytic requires a subset of the building data, and thus can be represented by a sub-ontology of the entire ontology. Thus, we leverage the building ontology and develop a systematic way to extract a sub-ontology that can satisfy analytics needs.

As discussed in §3, there are two logic views of the building entities, subsystems and functionalities, which are commonly used in building analytics. We introduce the *ontology segment*, defined as a set of building entities with a logic partition of building ontology. We will first extract *subsystem ontology segments* and *functionality ontology segments* from the building ontology. Then, we will perform a set of algebraic operations on these ontology segments to generate the sub-ontology.

4.3.1 Ontology Segment Extraction.

Subsystem Ontology Segment Extraction. A subsystem in a building is a complete, independently functioning part of the building's

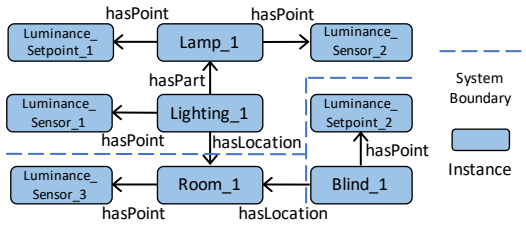

Figure 7: Subsystem Boundary divides an ontology into subsystems (e.g. lighting system, blind, and room in the figure).

interior. This means that the components that make up the subsystem can work together for a certain task. For example, a lighting system is a common type of subsystem in modern buildings, which typically consists of a group of lamps, switches and ballasts. It works in a single lighting zone that conditions the intensity of illumination for thermal visual comfort. We call this a subsystem. To logically separate subsystems, we define *subsystem boundary* through a *subsystem boundary predicate set*, which contains the belonging relationship. Table 2 shows an example subsystem boundary predicate sets of Brick predicates.

A subsystem boundary exists between two nodes in the building ontology where there is no predicate (i.e. no relationships) or where the predicate does not belong to the system boundary predicates. For example, in Figure 7, the predicate `hasPart` suggests that Lamp_1 is part of Lighting_1. Thus, they belong to the same subsystem segment. The predicate `hasLocation` is not in the subsystem boundary. Thus, Room_1 and Lighting_1 do not belong to the same subsystem.

With the subsystem boundary defined, we can automatically extract subsystem ontology segments. We note that the Brick schema has already defined a naming convention which we can use to select a node in the building ontology. In the example in Figure 7, to get the lighting subsystem segment, Lighting_1 is defined as <Lighting_1, rdf:type, brick:Lighting_System> in Brick, and the SPARQL query `SELECT ?light WHERE ?light rdf:type/rdfs:subClassOf* brick: Lighting_System` can find it because it is of the type `brick:Lighting_System`. If there are multiple lighting systems, all of them will be returned. Breadth-first search (BFS) can be used to find nodes recursively. Specifically, the subsystem segment type in user query can be followed to pinpoint the subsystem (e.g. lighting system) and extract the initial nodes. BFS can then be conducted from the initial nodes (e.g. Lighting_1 in Figure 7) with a search stopping condition defined by the subsystem boundary.

Functionality Ontology Segment Extraction. As discussed, only a logic view subsystem cannot meet all building analysis requirements. For example, to develop a control model for lighting system, the lighting conditions data (e.g. illumination intensity) and operation data (e.g. status and setpoints) from the lighting system are needed. Extra lighting system data, such as power, may be a curse (e.g. irrelevant and redundant variables), rather than a blessing.

We also note that building analytics can require the same types of functional sensors in different subsystems. For example, to develop the energy consumption model used in ECP, we need the temperature data of various subsystems. Therefore, we introduce the functionality ontology segment, referring to a group of entities in the building with the same function.

For the functionality segment extraction, we directly make use of a SPARQL query to find all the nodes whose type matches a particular functionality segment type. For example, the functionality segment of the type Temperature will include all the nodes of the type `brick:Temperature_Sensor`.

4.3.2 Algebra-assisted Ontology Composition. With subsystem and functionality ontology segments, we now present how to generate the sub-ontology for a specific query. Energon allows operations on top of the ontology segments to compose the final sub-ontology by following Energon algebra defined as follows: (a general introduction on set algebra can be found in [32]).

Energon Algebra. An Energon query Q, as illustrated by Figure 3, is a *select-from-where* liked query. Conceptually, such queries have the "canonical" form of Formula (1) in terms of relational algebra:

$$Q = \pi_{\mathcal{P}(p_1,\cdots,p_k)}\sigma_{\mathcal{S}(\varphi_1,\cdots,\varphi_m)}\tau_{\mathcal{F}(\omega_1,\cdots,\omega_n)}(G_1,\cdots,G_r) \quad (1)$$

That is, upon the ontology relations $(G_1, G_2, \cdots G_n)$, the following two types of clauses **WHERE** and **FILTER** denoted by σ (with $\mathcal{S}$ function over conditions $\varphi_1, \cdots, \varphi_m$) and τ (with $\mathcal{F}$ function over conditions $\omega_1, \cdots, \omega_j$) respectively are performed to determine the pending ontology. Then ontology extraction π (with function $\mathcal{P}$ indicates) with projected attributes $(p_1, \cdots, p_k)$ is performed to compose a building sub-ontology.

Let A, B be two sub-ontologies, U be the universe, i.e. the complete building ontology, $\varnothing$ be the empty set, and x, a and b be a single node of ontology. Energon algebra defines four fundamental operations to conduct an ontology composition:

- **Union** $A + B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Intersection** $A * B = \{x \mid x \in A \text{ and } x \in B\}$.
- **Difference** $A - B = \{x \mid x \in A \text{ and } x \notin B\}$.
- **Join** $A \bowtie B = \{x \mid x = a \text{ or } b, \text{where } a \in A, b \in B \text{ and } a \leftrightarrow b)\}$, "$\leftrightarrow$" represents a situation where at least one predicate exists between a and b, i.e. a and b are connected in the ontology graph.

We define that any query that fits Formula (1) as a normal form of Energon query. For example, the query in Figure 3 is an Energon query and can be formalized as:

$$Q = \pi_{(Light \bowtie Blind) * Setpoint}\,\sigma_{\varphi_1 \wedge \varphi_2}\tau_{\omega_1 \wedge \omega_2}(G_A)$$

Here, φ_1 is BuildingID='building_A', and φ_2 is Source='Local'. ω_1 is A.timestamp > '20190801', and ω_2 is A.timestamp < '20191231'.

It is easy to verify that the Energon algebra has three algebraic equivalence laws: *Commutative*, *Associative*, and *Distributive* (Table 3). This ensures consistency when developers write Energon queries in different ways. For example, to extract the luminance data and setpoint data from the Lighting subsystem in a building, the algebra can be expressed as $Lighting * Luminance + Lighting * Setpoint$. With the distributive property, the algebra can also be expresssed as: $Lighting * (Luminance + Setpoint)$.

We now present the implementation of these four operations: Union, Intersection and Difference operations are implemented by overloading the +, * and − operators, respectively.

The Join operation is the set of all combinations of tuples in two tables that are equal in their common attribute names. In building ontology, to jointly control multiple subsystems (e.g., BIC and ECP), we introduce Join operator to merge the subsystems with

relationships. In Energon algebra, Join operator is a binary operator that can only be performed on two subsystems, and thus its results can be regarded as a further subsystem. We implement the Join operator in a three-step process as follows:

(1) Determine the two subsystems to be joined from the Energon query, and extract the sub-ontology of these two subsystems;

(2) For each pair of equipment in the both side subsystems, traverse the entire ontology, and try to find a path (a sequence of predicates in building ontology that connects a sequence of entities) between the equipment. Once there is a path exists, the pair of equipment is added to the resulting subsystem ontology;

(3) For each piece of added equipment, an algorithm like BFS can be conducted recursively to find underlying sensors and components. As a result, the targeted joint subsystem is extracted.

We illustrate the Join execution process in Figure 1, where BIC needs the data from the Lighting system and the corresponding Blind system. The algebra can be expressed as $Lighting \bowtie Blind$. To extract the sub-ontology, the query engine traverse the RDF triples recursively to search the paths between pieces of equipment in Lighting subsystem ($Lighting\_1$ and $Lighting\_2$) and the equipment in the Blind subsystem ($Blind\_1$ and $Blind\_2$). As shown in Figure 8, there are two paths were searched by the query engine, i.e. $Lighting\_1$ to $Blind\_1$ as well as $Lighting\_2$ to $Blind\_1$, thus all the sensors from these three equipment are extracted.

Completeness and Consistency. Based on the Energon algebra, an Energon query has the properties of completeness (Theorem 1) and consistency (Theorem 2). Theorem 1 states that an Energon query can meet building analytics requirements, i.e. for building analytics with a targeted sub-ontology, an Energon query exists and our implementation above can extract such a sub-ontology.

Theorem 1. *(Completeness) Given a building ontology and the requirement that a building analytic be represented as a logic partition of this building ontology, there exists a set of subsystem ontology segments, functionality ontology segments, and algebra operations on these ontology segments, that can construct this logic partition.*

We now examine the issue of consistency. Given two individual Energon queries, Q_1 and Q_2, if the sub-ontologies extracted with Q_1 and Q_2 are the same, we refer to Q_1 and Q_2 as *consistent*.

Note that the Energon algebra does not have a distributive law for intersection ($*$) over join ($\bowtie$). For example, $Lighting \bowtie (Room * Zone)$ should be different from $Lighting \bowtie Room * Lighting \bowtie Zone$, as the result of the former algebra can be empty (zones and rooms have no intersection set) and the latter algebra can be non-empty (some lights are public equipment of zones and rooms).

Theorem 2 states that in the case of two Energon queries, as long as they follow the Energon algebra equivalence laws (without the distributive law for intersections over joins), they are consistent.

Theorem 2. *(Consistency) Given two individual Energon queries, Q_1 and Q_2, if the algebra of Q_1 can be equivalent to Q_2's with Energon algebra equivalence laws, then Q_1 and Q_2 are consistent.*

We defer the formal proof of these theorems to Appendix A.

4.4 Energon Ontology Index

Energon stores the RDF triples constituting a building ontology graph in an ontology storage structure. This heavily influences the execution of an Energon query, especially the ontology extraction process. A common ontology storage structure is to store the entire ontology as RDF triples (see Figure 9 (a)). This can incur significant execution overhead for processing Energon queries. Specifically, the subsystems of EnergonQL have a hierarchical structure, i.e. Lamp and Ballast are children of Lighting; yet a storage in RDF triples does nothing to maintain such a hierarchical structure. As such, extracting an entity from each individual segment (§4.3.1) first requires an initial node to be located and then the entire ontology to be traversed, an $O(n)$ operation conducted, and the entities of a subsystem extracted. This process needs to be repeated for each entity, leading to a time complexity of $O(n^2)$. Even worse, once there is a Join operator, the traversal to specify the relationship between two ontology segments is needed, leading to another $O(n)$ and the overall complexity becomes $O(n^3)$.

Proposition 1: Commutative law
• $S_1 \Theta S_2 \equiv S_2 \Theta S_1, \forall \Theta \in \{+, *, \bowtie\}$
Proposition 2: Associative law
• $(S_1 \Theta S_2) \Theta S_3 \equiv S_1 \Theta (S_2 \Theta S_3), \forall \Theta \in \{+, *, \bowtie\}$
Proposition 3: Distributive law
• $(S_1 + S_2) \Theta S_3 \equiv S_1 \Theta S_3 + S_2 \Theta S_3, \forall \Theta \in \{*, \bowtie\}$

Table 3: Energon Algebra Equivalence Laws.

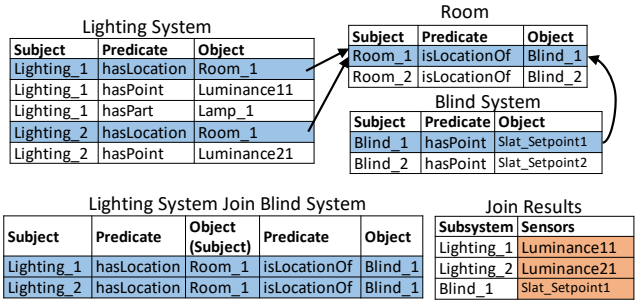

Figure 8: Join Execution in Energon

We present the Energon Ontology Index structure to maintain the hierarchical structure of a building ontology into an adjacency list (see Figure 9 (b)). This allows for an immediate positioning of the substructure of an entity. We maintain a list for each subsystem as building analytics are based on subsystems. This Energon Ontology Index structure basically sacrifices a memory space of $O(n)$ to substitute a time complexity of $O(n^2)$. We present the implementation details as follows.

Subsystem Segment Index. As Figure 9 (b) shows, a fully elaborated two-layer index is represented to store the subsystem structure for a certain building. In the first layer, the keys are the unique identifications of a subsystem index and values are lists of belonging equipment segments index. In the second layer there are three lists, for each type of equipment: sensor list, component list, and segment list. The sensor list is used to associate the data points of the equipment. The component list is used to associate the components of the equipment. The segment list stores the equipment connected with each segment.

There are two benefits to this structure. Firstly, when a single subsystem is extracted (e.g. Lighting in Figure 9 (b)), its own sensors ($Luminance\_11$, $Setpoint\_11$ and others in $Ballast\_1$ and $Lamp\_1$)

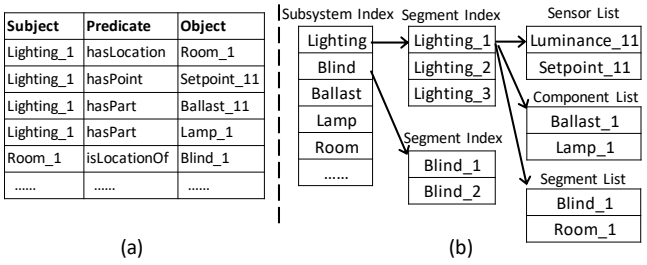

Figure 9: (a) Ontology store in RDF triples; (b) Energon Ontology Index for optimized performance

can be easily found by recursively indexing the component list and the sensor list instead of conducting a full search of the ontology. For each iteration, the sensors in the sensor list are added and the components in the component list figure out the next object. Secondly, the segment list significantly accelerates the Join process (especially in a dense ontology graph) by recording both a directly and indirectly connected subsystem. When Energon sees a query pattern like $A \bowtie B$, e.g. *Blind* $\bowtie$ *Lighting*, it checks the segment lists from each equipment index of the subsystem *Blind*, to extract the equipment of the subsystem *Lighting*. Thus, there is a saving of the traversal time among subsystems.

Functionality Segment Index. The functionality segment has a simpler index structure compared with the subsystem segment index. It is a key-value structure, where the index keys are the all identifications of functionality segments in the ontology graph (e.g. *Luminance* and *Setpoint*). Each value contains a sensor list associated with the functionality key.

Both segment index structures can contribute to the ontology extraction process in two ways. First, in the extraction of ontology segments, the segments can be easily retrieved instead of having an iterative traversal with an integral RDF triples, as the sensor list records the data points within the segments. Secondly, in the algebra-assisted ontology composition, the operators mentioned in §4.3.2 can be well supported for constructing an objective sub-ontology. The sensor list and component list contribute to the Union, Intersect and Difference operation between segments. The segment list in subsystem segment index contributes to the Join operation between subsystem segments.

5 QUALITATIVE EVALUATION

We demonstrate how Energon simplifies analytics development using four different types of analytics.

5.1 Building Integrated Control (BIC)

Given that indoor comfort is affected by more than one subsystems [13], BIC is an MPC method that jointly controls multiple subsystems to manage the indoor environment in a building.

A typical analytics is to adjust indoor light levels for visual comfort through the integrated control of the Lighting System and Blind System. During the day, in addition to electric lighting, daylight provides indoor lighting. The incident daylight intensity is managed by adjusting the blind slat angle. In this way, such an integrated control allows the interior illuminance to be maintained at a proper level (e.g. ~500 lux), meanwhile saving energy for electric lighting

```
1  # 1. SPARQL query for ontology extraction
2  bic_query = '''
3      SELECT ?lpsp ?llum ?bsp ?blum ?ans ?srs WHERE {
4      ?room rdf:type/rdfs:subClassof* brick:Room .
5      ?lighting rdf:type/rdfs:subClassOf* brick:Lighting_System .
6      ?blind rdf:type/rdfs:subClassOf* brick:Shading_System .
7
8      ?room brick:isLocationOf ?lighting .
9      ?room brick:isLocationOf ?blind
10
11     ?lamp rdf:type/rdfs:subClassOf* brick:Luminance .
12     ?lpsp rdf:type/rdfs:subClassOf* brick:Setpoint .
13     ?llum rdf:type/rdfs:subClassOf* brick:Luminance_Sensor .
14     ?light brick:hasPart ?lamp .
15     ?lamp brick:hasPoint ?lpsp .
16     ?lamp brick:hasPoint ?lum .
17
18     ?bsp rdf:type/rdfs:subClassOf* brick:Setpoint .
19     ?blum rdf:type/rdfs:subClassOf* brick:Luminance_Sensor .
20     ?blind brick:hasPoint ?bsp .
21     ?blind brick:hasPoint ?blum .
22
23     ?wea rdf:type/rdfs:subClassOf* brick:Weather .
24     ?ans rdf:type/rdfs:subClassOf* brick:Angle_Sensor .
25     ?srs rdf:type/rdfs:subClassOf* brick:Solar_Radiance_Sensor .
26     ?wea brick:hasPoint ?ans .
27     ?wea brick:hasPoint ?srs .
28     }
29     '''
30
31 # 2. data extraction and encapsulation
32 request = pymortar.FetchRequest(
33     # Define building 'building_A' as data source
34     sites=['building_A'],
35     views=[
36         pymortar.View(
37             name='data_points',
38             query=bic_query,
39         ),
40     ],
41     # Data format is omitted here, e.g. time series interval
42     # and aggregation method
43     ...
44     # Define the time window
45     time=pymortar.TimeParams(
46         start='2019-08-01T00:00:00Z',
47         end='2019-12-30T00:00:00Z',
48     )
49 )
50
51 result = fetch(request)
52 data = result['data'][data_list]
```

Figure 10: Current data acquisition process (e.g. in Mortar [16]) for developing analytics (e.g. BIC) requires detailed knowledge about a building.

as much as possible. As an example of such integrated control, Shen et al [30] utilized the setpoint of the Lighting System (e.g. intensity level of lamps), the Blind System (e.g. blind slat angle) as well as environmental data (e.g. solar incident angle and solar radiation rate) to build an ML model to predict the indoor illuminance.

Figure 10 shows how this analytics would be implemented using existing platforms such as Mortar (see in appendix). The analytics program has two parts. The first part (line 2 to line 29) is a SPARQL query to retrieve the data from lighting, blind, and weather systems. The second part (line 31 to line 52) executes the query and gets the data for further analytics development. Clearly, this implementation requires the developers to have an in-depth knowledge of the structure and relationships of building systems.

```
1  from Energon.EnergonQL import *
2  from ontology.ontology_bic import global_ontology
3
4  # Load the complete building ontology for BIC
5  global_ontology()
6
7  # 1. Energon Query for ontology and data extraction
8  bic_query = '''
9      SELECT Light(A) JOIN Blind(A) * (Luminance(A) + Setpoint(A)) +
           Weather(A) * (Solar_Angle(A) + Solar_Radiance_Rate(A))
10     FROM Building A
11     WHERE A.BuildingID = 'LightZone' AND A.Source = 'Local'
12     FILTER A.TIMESTAMP > '20190801' AND A.TIMESTAMP < '20191231'
13     '''
14 # 2. execute the query to retrieve the data
15 data = fetch(bic_query)
```

Figure 11: Retrieving Data for BIC in Energon

By contrast, Figure 11 shows the same analytics developed in Energon, still with two parts. The first part (line 8 to line 14) defines a BIC query in the EnergonQL. The second part (line 16) executes the query and retrieves the data. We see that, in Energon, the developer does not need to have auxiliary knowledge about the building systems. Specifically, the spatial auxiliary knowledge is hidden by the subsystem and functionality partitions (e.g., lighting and blinds); the temporal auxiliary knowledge is addressed by filter clause (e.g., timestamp conditions); and the contextual auxiliary knowledge is hidden by the Join operator (e.g. $Lighting \bowtie Blind$). We also see that the Energon program is portable across buildings and easier to extend. For example, if a lighting system has a ballast (e.g., a ballast is installed with sensors to collect its setpoint data), there is no change in the Energon program.

Energon provides the level of abstraction with which developers can focus on analytics-specific design and implementation, thus simplifying the development of analytics.

5.2 Energy Consumption Prediction (ECP)

ECP often takes an MPC-based approach to save energy for a building system by predicting the energy consumption of the system with a group of available operations. In general, ECP establishes a (usually nonlinear) model with the control strategies as inputs and the energy consumption according to the control strategies as outputs. An optimization algorithm then searches the control strategy space for the control strategy that would result in the least amount of energy consumed.

We implement an ECP based MPC approach to VAV control in a building while maintaining thermal comfort. For this ECP analytics, we use environmental data (e.g. weather and zone condition) and data on the historical operations of AHU and VAV systems (e.g., setpoint) [2, 22]. In Figure 13 (see in Appendix B), we show an example of an Energon query that extracts data for the ECP analytics.

5.3 Fault Detection and Diagnosis for AHU (FDD-AHU)

Traditional FDD methods follow rule-based models. Recently, data-driven ML model-based approaches have been put forward [25, 29]. As faults often occur in parallel, multi-objective ML models are used. Specifically, a multi-layer diagnostic model has been developed to detect multiple types of faults (e.g., stuck dampers and stuck cooling coil valves) in an AHU system [25].

Analytics	Method	Lines of Code	Development Time (minute)
BIC	Mortar	42	86.2
	Energon	11 (-73.8%)	46.2 (-46.4%)
ECP	Mortar	54	90.2
	Energon	13 (-75.9%)	35.2 (-60.9%)
FDD-AHU	Mortar	73	66.6
	Energon	11 (-84.9%)	30.4 (-54.4%)
CP	Mortar	52	60.4
	Energon	10 (-80.8%)	25.0 (-58.6%)

Table 4: Development Effort of Mortar and Energon

The common data used for these models are air condition data (e.g. temperature, humidity, pressure, and flow rate) and operation data (e.g. control signals and setpoints). To retrieve data for FDD-AHU, an example of an Energon query is shown in Figure 14 (see in Appendix B).

5.4 Chiller Profiling (CP)

Chillers are core components of an HVAC system. Chiller profiling involves estimating the performance of a chiller, which can be used for maintenance, operation decisions, and so on. The performance of a chiller is called the Coefficient of Performance (COP). Intuitively, COP is an indication of the amount of cooling load a chiller can output given a unit of electricity.

To build a COP prediction model, three kinds of data are needed: (1) temporal features such as the age of the chillers (2) meteorological features such as outdoor air temperature, and (3) mechanical features such as the inlet/outlet node water temperature, mass flow rate, and the power consumption of the chiller as well as its associated parts. We also show an Energon query to extract these data in Figure 15 (see in Appendix B).

6 QUANTITATIVE EVALUATION

6.1 Analytics Development Effort

We compare the development effort (in terms of program length and program development time) required when using Energon to that for a state-of-the-art system, Mortar. Our focus is to demonstrate how much development effort can be reduced.

We recruited five developers in this evaluation study to implement the four analytics mentioned in §5, each associated with a different building ontology. All five developers are data scientists with limited knowledge of RDF semantics and building analytics. We intend to evaluate the development efficiency of using these two approaches; thus, we do not count the time spent learning background knowledge such as RDF, SPARQL, and the task-specific requirements of these four analytics into the development time. The developers only recorded the time spent on implementing the analytics according to the order in which they appear in Table 4.

The development workflow consists of two main steps: ontology extraction and data extraction. Table 4 shows the breakdown of the lines of code for Mortar and Energon. We exclude auxiliary snippets such as comments and library imports. We see a drastic reduction in the total number of lines of code for all four analytics, by 73.8%, 75.9%, 84.9%, and 80.8%, respectively. This is attributed to the effective abstraction of ontology extraction and data extraction, which can be done with standard *select-from-where* query expressions in

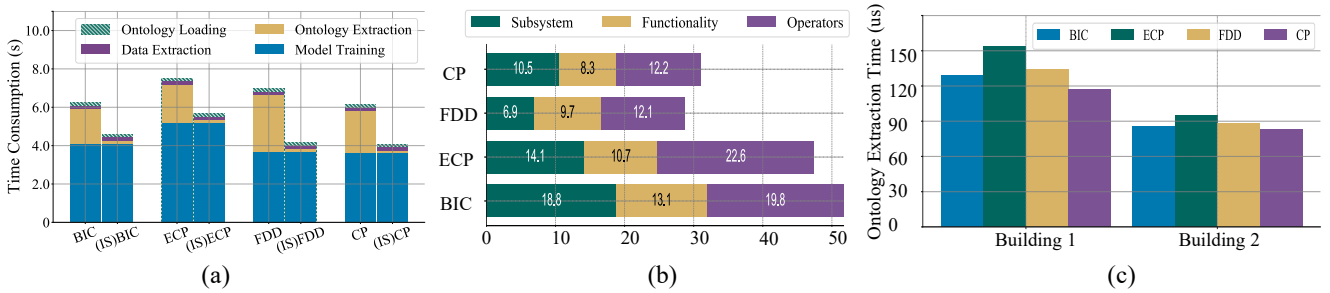

Figure 12: (a) Execution Time with and without the Indexing Structure Optimization; (b) Time Consumption of Subsystem, Functionality, and Operator in Ontology Extraction Stage; (c) Execution Time of Ontology Extraction across Buildings

Energon, whereas with Mortar building-specific SPARQL queries need to be written with an understanding of the specifics of the building, a key difficulty during the development of the analytics.

In Table 4 shows a record of development times. We see that with EnergonQL developers spent less time on implementing the analytics. The time spent was reduced by 46.4%, 60.9%, 54.4%, and 58.6%, respectively, compared to using Mortar. We see that as they developed one analytics after another, the developers became more proficient and spent less time on each analytics over time. The speedup in this process was more significant when Energon was used. Specifically, with Energon, the development time of the last building analytics dropped from 46.2 minutes to 25.0 minutes (a 45.9% reduction), compared with the first building analytics. When using Mortar, the development time was reduced from 86.2 minutes to 60.4 minutes (a 29.0% reduction). This is partially because for each analytics in EnergonQL, the developer does not need to understand the details of every new building ontology.

These results suggest that Energon can effectively shorten development effort via a standardized workflow.

6.2 System Execution Time

We now study the execution time of Energon. Figure 12 (a) shows the results on the four analytics. We report on the execution time of the four modules – ontology loading, ontology extraction, data extraction, and model selection and training. We see that model training dominates the execution time of analytics. This is expected since ML analytics have significant model training time. We see that the ontology extraction module also requires a long execution time, due to the traversal of the building ontology, which takes 30.2%, 27.3%, 43.2% and 36.8% in the four analytics respectively.

We then evaluate our Indexing Storage (IS) optimization approach (§4.4), where we offline pre-extract and cache ontology segments for an online lookup. Figure 12 (a) shows that our IS scheme substantially reduces the ontology extraction time, and that the average execution time among the four analytics is reduced by more than 1000x times for all four mentioned building analytics. This is attributed to the effective indexing mechanism, which avoids time-consuming traversals with RDF triples in this stage.

In addition, we study two factors that influence the time consumption of ontology extraction – the first one is query constituents and the second one is building ontology size. As shown in Figure 12 (b), the execution time of the ontology extraction of these four analytics is respectively 51.7, 47.4, 28.7, and 31.0 μs (from bottom to top). The execution mainly consists of three parts: subsystem extraction, functionality extraction, and operator performing. We find that no part dominates the whole process. The Energon queries for BIC and ECP take longer than those for FDD and CP for two reasons: 1) more subsystems are involved and need to be extracted; 2) The JOIN operator also increases the complexity of ontology extraction. Thus, it takes longer for the queries for BIC and ECP to execute, especially the part of operator performing.

We note that, even for the same analytics, ontology extraction time varies across buildings because the scale of their ontology differs. Figure 12 (c) shows the ontology extraction time in two simulated buildings, where the ontology size of Building 1 is five times over that of Building 2. However, we see that the time consumption in Building 1 only increased by 50.0%, 62.1%, 52.3% and 40.9%, for the four analytics, respectively. Our proposed Subsystem Segment Index stores both directly and indirectly connected equipment, and therefore even when the building scale increases dramatically, the traversal time only grows at a moderate pace.

7 CONCLUSION

In recent years, ML model-based building analytics have emerged and proven to be effective for the operation, control, and maintenance of buildings. Such analytics range from the profiling of building systems and energy conservation, to fault detection and diagnosis. While promising, each of these applications still requires non-trivial and building-specific development efforts to deploy in practice. The key difficulty is that developers need both analytics-specific knowledge to develop applications and building-specific knowledge to extract building data.

In this paper, we presented Energon, a data acquisition system that can support the development of building analytics with an abstraction that decouples the process of analytics development from the nuances details of the building system. With Energon, developers can focus on application development, and applications become portable across buildings. We evaluated Energon both qualitatively and quantitatively, and showed that Energon simplifies development in terms of lines of code and development effort.

8 ACKNOWLEDGEMENTS

Dan Wang's work is supported by GRF 15210119, 15209220, ITF-ITSP ITS/070/19FP, CRF C5026-18G, C5018-20G, PolyU 1-ZVPZ and a Huawei Collaborative Project. Dezhi Hong's work is supported by National Science Foundation 1940291, 1947050, and 2040727.

REFERENCES

[1] A. Afram and F. Janabi-Sharifi. 2014. Theory and applications of HVAC control systems–A review of model predictive control (MPC). *Building and Environment* 72 (2014), 343–355.

[2] 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.

[3] E. Alschuler, J. Antonoff, R. Brown, and M. Cheifetz. 2014. Planting SEEDs: Implementation of a Common Platform for Building Performance Disclosure Program Data Management. In *Proc. ACEEE Summer Study.*

[4] 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.

[5] L. Basu, K.and Hawarah, N. Arghira, H. Joumaa, and S. Ploix. 2013. A prediction system for home appliance usage. *Energy and Buildings* 67 (2013), 668–679.

[6] A. Beghi, R. Brignoli, L. Cecchinato, G. Menegazzo, M. Rampazzo, and F. Simmini. 2016. Data-driven fault detection and diagnosis for HVAC water chillers. *Control Engineering Practice* 53 (2016), 79–91.

[7] 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).

[8] R. G. G. Cattell, R. G. Cattell, D. K. Barry, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, et al. 2000. *The object data standard: ODMG 3.0.*

[9] B. Chen, Z. Cai, and M. Bergés. 2019. Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy. In *Proc. ACM BuildSys'19.* 316–325.

[10] Y. Chen, L. K Norford, H. W Samuelson, and A. Malkawi. 2018. Optimal control of HVAC and window systems for natural ventilation through reinforcement learning. *Energy and Buildings* 169 (2018), 195–205.

[11] CopperTree. 2020. CopperTree Analytics. https://www.coppertreeanalytics.com/

[12] D. Dehestani, F. Eftekhari, Y. Guo, S. H. Ling, S. Su, and H. Nguyen. 2011. Online Support Vector Machine Applicationfor Model Based Fault Detection and Isolationof HVAC System. *International Journal of Machine Learning and Computing* 1 (01 2011), 66–72.

[13] X. Ding, W. Du, and A. Cerpa. 2019. OCTOPUS: Deep reinforcement learning for holistic smart building control. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation.* 326–335.

[14] Entronix. 2020. Entronix EMP. https://entronix.io/

[15] L. Fegaras, C. Srinivasan, A. Rajendran, and D. Maier. 2000. lambda-DB: An ODMG-Based Object-Oriented DBMS. *SIGMOD Rec.* 29, 2 (2000), 583.

[16] 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.

[17] M. R. Garey and D. S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* USA.

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

[19] F. He, C. Xu, Y. Xu, D. Hong, and D. Wang. 2020. EnergonQL: A Building Independent Acquisitional Query Language for Portable Building Analytics. In *Proc. ACM BuildSys '20.*

[20] H. Huang, L. Chen, and E. Hu. 2015. A new model predictive control scheme for energy and cost savings in commercial buildings: An airport terminal building case study. *Building and Environment* 89 (2015), 203–216.

[21] J. Kelly and W. Knottenbelt. 2015. Neural nilm: Deep neural networks applied to energy disaggregation. In *Proceedings of the 2nd ACM international conference on embedded systems for energy-efficient built environments.* 55–64.

[22] A. Kusiak, M. Li, and F. Tang. 2010. Modeling and optimization of HVAC energy consumption. *Applied Energy* 87, 10 (2010), 3092–3102.

[23] A. Kusiak, G. Xu, and Z. Zhang. 2014. Minimization of energy consumption in HVAC systems with data-driven models and an interior-point method. *Energy Conversion and Management* 85 (2014), 146–153.

[24] S. Li and J. Wen. 2014. Application of pattern matching method for detecting faults in air handling unit system. *Automation in Construction* 43 (2014), 49 – 58.

[25] 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.

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

[27] Lucid. 2020. BuildingOS. https://lucidconnects.com/

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

[29] 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.

[30] 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.

[31] SkyFoundry. 2020. SkySpark. https://skyfoundry.com/

[32] R. R. Stoll. 1979. *Set Theory and Logic.* Dover Publications, Upper Saddle River, NJ, USA.

[33] B. Sun, P. B. Luh, Q. Jia, Z. Jiang, F. Wang, and C. Song. 2012. Building energy management: Integrated control of active and passive heating, cooling, lighting, shading, and ventilation systems. *IEEE Transactions on automation science and engineering* 10, 3 (2012), 588–602.

[34] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1626–1629.

[35] 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.

[36] T. Wei, Y. Wang, and Q. Zhu. 2017. Deep reinforcement learning for building HVAC control. In *Proc. ACM DAC'17.* 1–6.

[37] W. Wei, O. Ramalho, L. Malingre, S. Sivanantham, J. C. Little, and C. Mandin. 2019. Machine learning and statistical models for predicting indoor air quality. *Indoor Air* 29, 5 (2019), 704–726.

[38] S. West, Y. Guo, R. Wang, and J. Wall. 2011. Automated Fault Detection And Diagnosis Of HVAC Subsystems Using Statistical Machine Learning. In *Proc. IBPSA'11.* 2659–2665.

[39] F. Xiao, Y. Zhao, J. Wen, and S. Wang. 2014. Bayesian network based FDD strategy for variable air volume terminals. *Automation in Construction* 41 (2014), 106–118.

[40] 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.

[41] K. Yan, W. Shen, T. Mulumba, and A. Afshari. 2014. ARX model based fault detection and diagnosis for chillers using support vector machines. *Energy and Buildings* 81 (2014), 287–295.

[42] Y. Yu, D. Woradechjumroen, and D. Yu. 2014. A review of fault detection and diagnosis methodologies on air-handling units. *Energy and Buildings* 82 (2014), 550–562.

[43] C. Zhang, S. R Kuppannagari, R. Kannan, and V. K Prasanna. 2019. Building HVAC scheduling using reinforcement learning via neural network based model approximation. In *Proc. ACM BuildSys'19.* 287–296.

[44] 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.

A APPENDIX

Theorem 1. *Given a building ontology and the requirement of a building analytics represented as a logic partition of this building ontology, there exists a set of subsystem ontology segments, functionality ontology segments, and algebra operations on these ontology segments, which can be used to construct this logic partition.*

Proof. For an individual building, let the complete ontology be the universe B. We know that the subsystem and functionality abstraction are two different partitions of B, where subsystem partition $S = \{SP_1, SP_2, ..., SP_m\}$ such that $SP_i \subset B$ and $SP_1 \cup SP_2 ... \cup SP_m = B$, functionality partition $F = \{FP_1, FP_2, ..., FP_n\}$ such that $FP_i \subset B$ and $FP_1 \cup FP_2 ... \cup FP_n = B$. Because F is partitioned based on sensor functionalities, F is an exact set cover of all the sensor data in B [17]. Since no two sensors of the same type are installed in one piece of equipment, we can always find a subsystem partition S such that $\nexists y[y \in SP_i \text{ and } x \in SP_i \text{ and } x \neq y]$. As a result, for every sensor entity $x \in B$, we can find exactly one FP_o where $x \in FP_o$ and we can find a collection of subsystems $CSP = \{SP_j ... SP_k\}$ such that $x \in SP_p, SP_p \in CSP$; therefore, x can be extracted as: $SP_p \cap FP_o$. For any combination of building data, $C = \{c_1, c_2, ...c_l\}$, we know that $c_i \in C$ can be extracted; therefore, C can also be extracted by the unions of all sensors in C □

Theorem 2. *Given two individual Energon queries, Q_1 and Q_2, if the algebra of Q_1 can be equivalent to Q_2's according to Energon algebra equivalence laws, then Q_1 and Q_2 are **consistent**.*

Proof. The Energon algebra equivalence laws can be easily proved by set theory, with the exception of the distributive law for Union(+) over Join(⋈). Here we would like to prove the following equation:

$$(A + B) \bowtie C = A \bowtie C + B \bowtie C$$

Let A, B, and C be three individual sub-ontologies of a certain building ontology. First, we expand the both sides of the equation as follows:

$$(A + B) \bowtie C = (1)\ (A + B) * [(A + B) \bowtie C] + (2)\ C * [(A + B) \bowtie C]$$
$$A \bowtie C = (3)\ A * (A \bowtie C) + (4)\ C * (A \bowtie C)$$
$$B \bowtie C = (5)\ B * (B \bowtie C)\ + (6)\ C * (B \bowtie C)$$

Such an expansion separates the result of the Join query. We then try to find if (1) = (3) + (5) and (2) = (4) + (6). Easy to have, (3) $A * (A \bowtie C) = A * [(A + B) \bowtie C]$. Recap that the Join(⋈) Operator is for mining the relationships between subsystems; hence, both of them represent the part of A that is related to C. Similarly, there exists (5) $B * (B \bowtie C) = B * [(A + B) \bowtie C]$. According to the Distributive Law of Intersection (∗), we have:

$$\begin{aligned}(3) + (5) &= A * [(A + B) \bowtie C] + B * [(A + B) \bowtie C] \\ &= (A + B) * [(A + B) \bowtie C] \\ &= (1)\end{aligned}$$

Also, (4) $C*(A \bowtie C)$ represents the part of C related to A, and (6) $C*(B \bowtie C)$ represents the part of C related to B. Combining these two items, we have (4) $C * (A \bowtie C) + (6)\ C * (B \bowtie C)$ which represents the part of C that is related to either A or B. Such a representation is the same as the representation of (2) $C * [(A + B) \bowtie C]$. In other words, (4) $C * (A \bowtie C) + (6)\ C * (B \bowtie C) = (2)\ C * [(A + B) \bowtie C]$. Putting these items together, we have the equation we wanted to prove at the beginning.

□

B APPENDIX

Here we give the example of Energon query for the three building analytics mentioned in §5: ECP, FDD-AHU, and CP in Figure 13, 14, and 15 respectively.

```
SELECT Weather(B) * (Temperature(B)  + Solar_Radiance_Rate(B)) +
      Zone(B) * Temperature(B) + AHU(B) JOIN VAV(B) * (Temperature(B)
       + Flow_Rate(B) + Setpoint(B))
FROM Building B
WHERE B.BuildingID = '5ZoneAutoDXVAV' AND B.Source = 'Local'
FILTER B.OCCUPANCY_FLAG = 1
```

Figure 13: Energon Query for ECP

```
SELECT AHU(C) * (Temperature(C) + Humidity(C) + Pressure(C) +
      Flow_Rate(C) + Signal(C) + Setpoint(C))
FROM Building C
WHERE C.BuildingID = 'MZVAV' AND C.Source = 'Local'
FILTER C.TIME_STAMP > '20070828' AND C.TIME_STAMP < '20090515'
```

Figure 14: Energon Query for FDD-AHU

```
SELECT (Chiller(D) * (Temperature(D) + Flow_Rate(D) + Power(D)) +
      Weather(D) * Temperature(D)
FROM Building D
WHERE D.BuildingID = 'CP1' AND D.Source = 'Local'
FILTER D.TIMESTAMP > '20190630' AND D.TIMESTAMP < '20190831'
```

Figure 15: Energon Query for CP