Brick v1.0 - Towards a Unified Metadata Schema For Buildings

Bharathan Balaji[1], Arka Bhattacharya[2], Gabe Fierro[2], Jingkun Gao[3], Joshua Gluck[3], Dezhi Hong[4], Aslak Johansen[5], Jason Koh[6], Yuvraj Agarwal[3], Mario Berges[3], David Culler[2], Rajesh Gupta[6], Mikkel Baun Kjærgaard[5], Joern Ploennigs[7], Kamin Whitehouse[4]

[1]UCLA, [2]UC Berkeley, [3]Carnegie Mellon University, [4]University of Virginia, [5]University of Southern Denmark, [6]UCSD, [7]IBM Research - Ireland

ABSTRACT

Commercial buildings have long since been a primary target for applications from a number of areas: from cyber-physical systems to building energy use to improved human interactions in built environments. While technological advances has been made in these areas, such solutions rarely experience widespread adoption due to the lack of a common descriptive schema that can reduce the now-prohibitive cost of porting these applications and systems to different buildings. Recent attempts have sought to address this issue through data standards and metadata schemes, but fail to capture the set of relationships and entities required by real applications. Building upon these works, this paper describes *Brick*, a uniform schema for representing metadata in buildings. Our schema defines a concrete ontology for sensors, subsystems and relationships among them, which enables portable applications. We demonstrate the completeness and effectiveness of Brick by using it to represent the entire vendor-specific sensor metadata of five diverse buildings across different campuses, comprising 15,700 data points, and running eight complex unmodified applications on these buildings.

CCS Concepts

•**Information systems** → **Ontologies;** *Process control systems;*
•**Computer systems organization** → **Sensors and actuators;**

Keywords

Smart Buildings, Building Management, Schema, Ontology

1. INTRODUCTION

Modern buildings are being integrated with a variety of networked sensors and equipment to improve convenience, accessibility and energy-efficient operations. These technological improvements hold the promise of significant advances in centralized operation and management, fault diagnosis, and integration to an emerging smart grid. As of 2012, 14% of the buildings in the U.S. deployed Building Management Systems (BMS) [9] to manage data collection and remote actuation of the connected building infrastructure. Innovations in "Internet of Things" (IoT) devices have led to connected lights, power meters, occupancy sensors and appliances that are capable of interfacing with the underlying SCADA systems used in building automation. New buildings are installed with BMS by design, and older buildings are being continuously retrofitted with networked systems for improved efficiency.

New networked devices thus present an opportunity for a building "applications plane" to provide new capabilities to building operators and occupants alike. However, as a platform, even the most modern BMS present a cacophony of data and information flows that vary by buildings, vendors and across locations. The lack of a common data representation prevents interoperability between buildings and limits scalability of applications as developers need to map the heterogeneous data of each building to a common format. NIST estimates that the U.S. building industry loses \$15.8 billion annually due to lack of interoperability standards [20].

Attempts have been made to address this problem. Building Information Models (BIM) [16] were introduced to address the interoperability concerns both for the design and operation of buildings. The resultant schemata – Industry Foundation Classes (IFC) [12] and Green Building XML (gbXML) [30] – were oriented towards design and construction efforts. As a consequence, only limited support was provided for BMS information. More recently, several schemata, e.g. Project Haystack [3], SAREF [18], have emerged to highlight the importance and use of building *metadata*. Brick builds upon these efforts to devise a practical schema that demonstrates real applications in a number of buildings across the U.S and Europe.

The challenge is to design a schema that can at the very least capture the points and relationships that actual building engineers and facilities managers chose to put into the BMS deployments across a heterogeneous set of buildings. The schema needs to be expressive enough to capture the contextual information for building subsystems and their data points so that canonical applications such as fault diagnosis and demand response can be easily developed and deployed. Recent work has shown that the existing schemata fall short in capturing important relationships and concepts necessary for applications for even one real building BMS [13].

Designing a comprehensive schema for the emerging IoT universe in order to run any possible application in any context is a difficult problem and beyond the scope of this work. Instead, we focus on creating an information exchange platform that is fo-

BuildSys'16 Nov 4–6, 2016, Stanford, USA
© 2016 ACM. ISBN 123-4567-24-567/08/06...\$15.00
DOI: 10.475/123_4

cused on commercial buildings where interactions among devices and people are core to sophisticated applications. In building such a platform, we are guided by the sensors, attributes and relationships that have been shown to be useful in the published literature with a view towards composability and extensibility. In designing Brick, we ask the following important questions and seek answers with demonstrated effectiveness:

- **Completeness:** Can Brick represent all the sensor metadata information (such as a sensor's location, type, etc.) contained in a building's BMS?
- **Expressiveness:** Can Brick capture all important relationships between sensors that are (a) (overtly or tacitly) mentioned in a building's BMS, and (b) expressed in canonical smart-building applications in published academic literature?
- **Usability:** Can Brick represent the information in a way that is easy to use for both the domain expert and the application developer? Can the schema provide support automation with machine readable data formats and querying tools?

Our design of Brick is grounded by the information from BMS across five buildings spread across two continents, comprising more than 615,000 sq-ft of floor space and more than 15,700 data points, whose BMS systems were set up by different vendors, and have vastly varying subsystems and sensors. We further refine our design requirements using eight canonical building applications that require integrated information across commonly isolated building subsystems: HVAC, lighting, spatial and power infrastructure.

We demonstrate that 98% of BMS data points across our five buildings can be mapped to Brick, and our eight applications can easily query the mapped building instances for required information. We open source the Brick schema files, the BMS metadata from our buildings, the application queries that run on top of Brick and tutorials on how to map existing building metadata to Brick.

2. BACKGROUND

Most large commercial buildings have monitoring and actuation sensor networks that are accessed through the BMS or SCADA (Supervisory Control and Data Acquisition) systems. These systems have programmable interfaces for higher level control, retain historical data and provide visualization. Sensors or control points in each building are generally set up with "labels": consistent naming conventions that describe the many aspects of a data point, such as its function, type, location, relationships to other subsystems, etc. Labels in some buildings are simply a terse alphanumeric representation, while in other buildings they are long-form and human readable. Typically, these labels are attached to various user interfaces of the specific BMS/SCADA systems, so that engineers and operators can check status and plot trends. However, "label" naming remains heterogeneous and is inconsistent between commercial vendors and even between buildings set up by the same vendor.

Thus, even with programmable access to labels, data, and other descriptive information, scaling analytics or intelligent control across commercial buildings to, for example, improve energy efficiency remains challenging. This is likely to be the case as long as the basic steps in interpreting the metadata involve labor intensive efforts by trained professionals with deep knowledge of building operations and specifics of each building.

Brick directly addresses this problem of building-specific "label" namespaces by devising a normalized vocabulary (i.e., a list of domain terms) and relationships which enable programmatic exploration of different facets of a building. Hence, building managers can represent their BMS metadata using Brick, and applications can

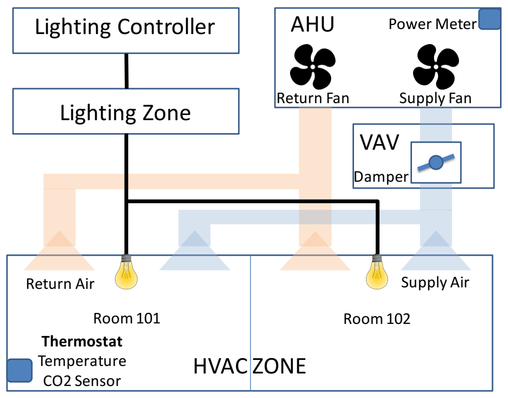

Figure 1: A simple example building that highlights the components to be modeled in a building schema.

be developed on a uniform data model. While it is possible to convert a particular building's custom labels to the Brick schema using semi-automated methods [13, 14, 11, 19, 21], we do not address those techniques in this paper since they are orthogonal.

2.1 An Example Building

We start with a hypothetical building to understand the requirements of uniform building data representation, outlining the current state of the art. Figure 1 shows the major components of this building that are of interest: an Air Handler Unit (AHU) supplying conditioned air to a Variable Air Volume Box (VAV), which modulates the air provided to an HVAC zone consisting of two rooms. The HVAC zone has a thermostat that contains a temperature and CO2 sensor. The same two rooms are part of a Lighting Zone, and the building Lighting Controller controls the zone lights based on a schedule. We model the HVAC and lighting systems as examples because these are the systems commonly found in a modern BMS.

At the very minimum, a schema should be able to model the components illustrated in Figure 1 as well as their relevant points such as temperature sensors and their related control parameters. More realistically, we should be able to model diverse infrastructure systems such as HVAC, lighting, water, and express the specifics of each installation as per end use and vendor requirements. For example, the AHU in an HVAC system can consist of equipment such as fans, pumps, heat exchangers, humidifiers, valves and dampers. Each component could have further types, e.g., fans can have types: supply fan, return fan, exhaust fan, and each fan would have its associated sensors measuring its speed, air flow and power consumption. In addition to this heterogeneity, the vendor may choose not to install certain sensors, or to expose esoteric data points whose functionality is unclear to others.

2.2 Current State of the Art

Project Haystack [3] aims to address heterogeneity in buildings using *Tags* to label different entities. For example, the temperature sensor in Figure 1 would have the Tags: `[zone, temperature, sensor]`. Tags provide a flexible and easy to use framework for annotating metadata to building data points. Haystack provides a vocabulary of tags that describes building equipment, weather, different types of data points and properties such as unit and data type. Their documentation provides guidelines for how to use the tags to create hierarchies and relationships between different entities of interest. Haystack uses a customized data format (Zinc) and there are no standard tools that enforce or verify the myriad set of rules laid out in their documentation. The lack of tools and a number of key missing concepts, such as rooms and floors, make it difficult to map existing buildings to Haystack.

IFC [12] is a standardized Building Information Model (BIM) that developed from the need to have a common exchange model for 3D architectural drawings. They are well designed to capture relationships within building components. For example, IFC is good at capturing space related information. However, the concept of **Sensors** was only added in the latest version and is still in its infancy. Thus, IFC lacks many of the common metadata attributes found in a typical BMS. Further, the focus on building CAD models makes IFC too verbose for a concise representation that application developers can grasp easily.

Semantic ontologies provide an alternative approach, with a formal language to represent essential concepts, domain hierarchies and relationships between the concepts. A number of ontologies have been proposed for smart homes and buildings. Most of these ontologies focus on realizing specific applications like controlling things [15], energy management [24], or automated design and operation [28]. Daniele et al. [18] combined these ontology modeling efforts in collaboration with industry to create a simple but unified model called SAREF. They identify 20 recurring concepts in homes and buildings across these ontologies, and lay out the steps to convert SAREF to a custom ontology. These common concepts do not effectively cover the diversity of devices and equipment in buildings [13]. Brick adopts similar design principles as SAREF, but our vocabulary and concepts are based on ground truth BMS deployments and representative smart building applications.

Bhattacharya et al. [13] performed a comparison of common metadata schemata in the buildings domain. The paper uses 89 building applications published in literature as a baseline to compare different schemata and shows that relationships between different pieces of information are essential to enable interoperability and portability of building applications. The paper compares the capabilities of Haystack [3], IFC [12] and SAREF [18]. They use three metrics to measure the effectiveness of a schema: (i) ability to completely map BMS metadata from three buildings to the schema, (ii) ability of the schema to capture the relationships required by applications, and (iii) the flexibility of the schema to deal with uncertainty as well as their extensibility to new concepts. The paper concludes that none of the studied approaches satisfactorily address these three requirements. The authors conclude that a new approach is needed that specifically addresses the metadata problem and normalizes the diversity of devices and their relationships.

Haystack Tagging Ontology (HTO) [17] maps the Haystack tags to an ontology, with each tag corresponding to an ontology class. Thus, HTO is able to combine the flexibility of tags and the formal modeling of ontologies to define essential BMS metadata and the relationships between entities. However, HTO confines the ontology to the defined tags, and the building entities which are a collection of tags (e.g. zone temperature sensor) are not modeled. HTO also does not provide a way to compose complex subsystems in a building and relies on Haystack tagging for mapping raw metadata to the ontology. Brick follows a similar methodology to combine tags and semantic models, but overcomes HTO's limitations with a vocabulary based approach and introduces encapsulation for composing complex systems. Thus, Brick provides a direct mapping to the data points and metadata exposed in a BMS and an enriched ontology that can be queried with ontology tools.

Brick builds upon these works in several ways. We utilize the tagging concept of Haystack and extend it with mechanisms to model relationships and entities. We use the location concepts from IFC. We use a semantic representation to utilize its flexibility and extensibility properties. The semantics allows us to formalize, restrict, and verify the usage of tags, entities, and relationships.

3. SCHEMA DESIGN

3.1 Design Principles

Brick's design focuses on the metadata and data points found in real building deployments and requirements defined by end use applications. We obtain ground truth information from five diverse buildings across the US and Europe, which have 15,700 data points and five different vendors in total (Table 4). We pick eight popular applications from the list of smart building applications compiled by Bhattacharya et al. [13], and formulate metadata queries for these applications to drive the basic requirements of Brick as well as evaluate how well our building metadata can be mapped to Brick. Section 6 contains our initial findings for the five buildings evaluated thus far.

We have used terminology and organized the basic concepts in a way that is consistent with BMS deployments in our buildings and the vocabulary used by the building managers at our respective institutions. We follow standard ontology design methods so that developers can leverage available tools for data formatting (e.g., Turtle [8]) and querying (e.g., SPARQL [6]).

3.2 Tags and Tagsets

We borrow the concept of *tags* from Project Haystack [3] (Section 2.2) to preserve the flexibility and ease of use of annotating metadata. We enrich the tags with an underlying ontology that crystallizes the concepts defined by the tags and provides a framework to create the hierarchies, relationships and properties essential for describing building metadata. With an ontology, we can analyze the metadata using standard tools and place restrictions to prohibit arbitrary tag combinations or relationships. For example, we can restrict the units of temperature sensors to Fahrenheit and Celsius.

We introduce the concept of a *tagset* that groups together relevant tags to represent entities. With Haystack and related tagging ontologies [17], an entity such as `zone temperature sensor` from Figure 1 is defined by its individual tags, so its properties and relationships with other entities can only be specified at the tag level. With tagsets, we have a cohesive concept of a `zone temperature sensor`, and we can specify that the temperature is maintained between its `cooling setpoint` and `heating setpoint`. The concept of tagsets works well with an ontology class hierarchy - a `zone temperature sensor` is a subclass of a generic `temperature sensor`, and will automatically inherit all its properties. Further, we avoid use of complex tags such as `chilledWaterCool` and `hotWaterReheat` in Haystack. The vocabulary of Brick is defined by its list of tagsets.

3.3 Class Hierarchies

We define several high level concepts that provide the scaffolding for Brick's class hierarchy. As the central emphasis of our design is on representing points in the BMS, we introduce *Point* as a class, with subclasses defining specific types of points: Sensor, Setpoint, Command, Status, Alarm. Each point can have several attributes, and we divide them into *properties* and *relationships*. Properties are attributes that provide specifics about the point: units, data type, etc. Relationships are attributes that relate the data point to other classes: its location, equipment it belongs to, etc.

We define three concepts as high level classes to which a Point can be related to: Location, Equipment and Measurement (Figure 2). We can expand these concepts in future versions to expand the metadata covered by Brick (e.g. Network, People). Each con-

Relationship / Inverse	**Transitive?**	**Definition**	**Endpoints**
`contains` / `isLocatedIn`	Yes	`A` physically encapsulates `B`	Loc. / Sensor Loc. / Equip.
`controls` / `isControlledBy`	No	`A` determines or affects the internal state of `B`	Function Block / Equip.
`hasPart` / `isPartOf`	Yes	`A` has some component or part `B` (typically mechanical)	Equip. / Sensor Equip. / Equip. Loc. / Loc.
`hasPoint` / `isPointOf`	No	`A` is measured by or is otherwise represented by point `B`	Equip. / Sensor Loc. / Sensor
`feeds` / `isFedBy`	Yes	`A` "flows" or is connected to `B`	Function Block / Equip. Equip. / Equip.
`hasInput` / `isInputOf`	No	Function `A` has an input `B`	Function Block / Sensor
`hasOutput` / `isOutputOf`	No	Function `A` has an output `B`	Function Block / Sensor

Table 1: List of the Brick relationships and their definitions. All definitions follow the form `A <relationship> B`, where `relationship` is the first one listed, not the inverse. All Brick relationships are asymmetric, and transitive where marked. If a relationship $\rightarrow$ is transitive, then if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$ is a valid relation. Asymmetric simply means that if $A \rightarrow B$, then $B \rightarrow A$ is invalid.

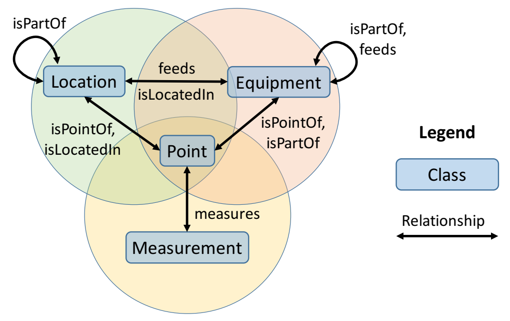

Figure 2: Information concepts in Brick and their relationship to a data point.

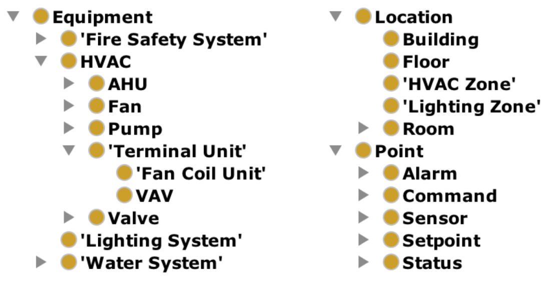

Figure 3: A subset of the Brick class hierarchy

cept has a class hierarchy to concretely identify each entity in the building. For example, the Equipment class has subclasses HVAC, Lighting and Power, each of which have their own subclasses. Figure 3 showcases a sample of Brick's class hierarchy.

It is common in a domain to use multiple terminologies for the same entity. For example, in HVAC systems, `Supply Air Temperature` and `Discharge Air Temperature` are used interchangeably. We identify these synonyms from our ground truth buildings, and mark the corresponding tagsets as being equivalent classes in Brick. Note that the class hierarchy does not strictly follow a tree structure, and we use multiple inheritance when appropriate. For example, a desk lamp can be a subclass of both the lighting system and office appliance classes.

3.4 Fundamental Relationships

Relationships connect the different entities in the building and are essential to providing adequate context for many applications. For example, an HVAC fault detection app running on our example building (Figure 1) needs to know the room in which the temperature sensor is located, the corresponding temperature setpoint and the status of the VAV that supplies conditioned air to this room.

Table 1 defines the basic set of relationships in Brick. We have designed these relationships to be minimal, multipurpose and intuitive so that it is easy for a user to specify a particular relationship. The `isPartOf` relationship is designed to capture the compositions among the entities in the building. For example, a room `isPartOf` a floor, an AHU `isPartOf` the HVAC system. The `feeds` relationship captures the different *flows* in the building - flow of air from AHU to VAV, flow of water from a tank to a tap or flow of electricity from a circuit panel to an outlet. Each of these relationships can have sub-properties. For instance, `feeds` can be extended to `feedsAirTo`, `feedsWaterTo`, etc. Figure 4 shows the relationships for a subset of example building in Figure 1.

The Brick schema includes possible relationships among classes as a guideline for users to add relationships to their instances. For example, using ontology class restrictions we say that a VAV can have points like `zone temperature sensor`, `discharge air flow setpoint`, `reheat valve command`, and it can have other equipment as its components such as `damper` and `reheat valve`. These can be exploited by a user interface to guide users while tagging raw metadata or while establishing relationships between entities. Note that we do not enforce these restrictions to enable the flexibility to compose building metadata as per user requirements.

3.5 Function Blocks

The tags, tagsets, class hierarchies and fundamental relationships provide sufficient expressiveness to describe our building metadata and direct relationships. However, buildings equipment and points are often grouped by multiple logical views such as control view.

We use *Function Blocks* to encapsulate details of such logical groups that expose an interface through named inputs and outputs. These are defined through `isInputOf` and `isOutputOf` relations to the particular function block acting as context. Function Blocks may encapsulate other Function Blocks via the `isPartOf` relation.

Consider a heat exchanger as an example: a heat exchanger is an

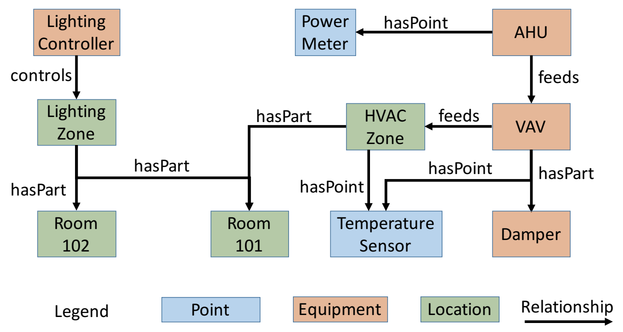

Figure 4: Brick classes and relationships for a subset of the example building in Figure 1.

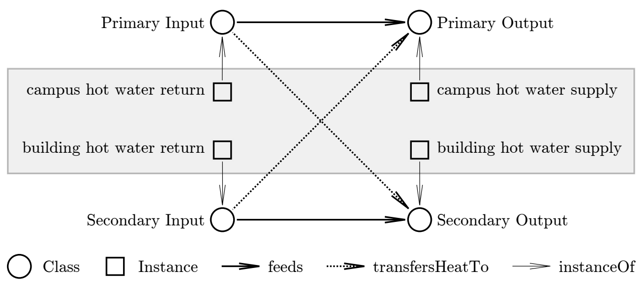

Figure 5: An example of a heat exchanger modeled in gray as a function block.

equipment that transfers heat between two fluid (air/water/steam) flows – a primary and a secondary. The primary is intended to heat or cool the secondary. It may be considered a simple piece of equipment with primary input, primary output, secondary input and secondary output. The primary input `feeds` the primary output, and it `transfersHeatTo` the secondary output.

A heat exchanger can be defined as a subclass of a function block and modeled as illustrated in Figure 5. Notice how the instance objects receive relations through the class they are instance of. Similarly, more complex equipment, like VAVs, can be constructed as function blocks and manipulated at this level.

4. RDF AND SPARQL

Brick represents knowledge as a graph of entities (nodes) connected by relationships (directed edges). This section briefly describes how Brick uses the RDF format to represent its knowledge, and how this knowledge is traversed and queried using SPARQL.

4.1 Representing Knowledge in RDF

Brick adheres to the RDF (Resource Description Framework) data model [26], which represents knowledge as a graph expressed as tuples of *subject-predicate-object* known as *triples*. All buildings in Brick consist of a collection of such triples. A triple states that some *subject* entity has some relationship *predicate* to some other entity *object* — essentially a directed edge in a graph. This simple structure enables the succinct and elegant composition of the large, interconnected structures typical of building subsystems.

All entities and relationships exist in some namespace, indicated by a `namespace:` prefix. Brick takes advantage of the standard RDF [4], RDFS [5] and OWL [2] namespaces, which come with their own graphs defining entities, relationships and restrictions.

```
1 example:myVAV rdf:type brick:VAV
2 example:myTempSensor rdf:type brick:Zone_Temperature_Sensor
3 example:myVAV brick:hasPoint example:myTempSensor
```

Figure 6: RDF triples instantiating a VAV and a Temperature Sensor and declaring that the VAV measures temperature via that sensor.

```
1  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX brick: <http://buildsys.org/ontologies/Brick#>
4  SELECT ?ahu ?room
5  WHERE {
6      ?zone rdf:type brick:HVAC_Zone .
7      ?room rdf:type brick:Room .
8      ?ahu rdf:type/rdfs:subClassOf* brick:AHU .
9      ?ahu brick:feeds+ ?zone .
10     ?zone brick:hasPart ?room .
11 }
```

Figure 7: A simple SPARQL query for retrieving all rooms connected to a given Air Handling Unit (AHU).

The collection of triples in Figure 6 gives the representation of the connection of the VAV to the Temperature Sensor using the `hasPoint` relationship from the building graph in Figure 4. Line 1 declares an entity identified by the label `example:myVAV`: this creates the `myVAV` entity in the `example` namespace. All entities are implicitly created the first time they are mentioned. `brick:VAV` is a class defined by the Brick ontology that represents a variable air-volume box. The use of the `rdf:type` relationship declares that `example:myVAV` is a `brick:VAV`. Similarly, line 2 of Figure 6 instantiates a Zone Temperature Sensor. Line 3 uses the Brick relationship `brick:hasPoint` to declare that `example:myVAV` is associated with the given temperature sensor.

4.2 Querying Knowledge with SPARQL

Applications query the Brick graph for entities and relationships using SPARQL (SPARQL Protocol and RDF Query Language) [6]. SPARQL queries specify constraints and patterns of triples, and traverse an underlying RDF graph to return those that match. For Brick applications, this underlying graph consists of all the entities and relationships in that building.

While SPARQL has many features, Brick is simple enough to support all of our intended applications with a simple subset of SPARQL. Figure 7, a query for retrieving all rooms which are connected to a given AHU, contains a representative example of each of these features.

Lines 1-3 declare the prefixes for the various namespaces to shorten the references to entities; for length, we omit these from all later queries in this paper. Line 4 contains the `SELECT` clause, which states that the variables `?ahu` and `?room` should be returned (the `?` prefix indicates a variable). The `WHERE` clause determines the types and constraints on these variables. Line 6 states that `?zone` is any entity in the graph that is an instance of the class `brick:HVAC_Zone`. Likewise, line 7 declares `?room` to be an instance of a `brick:Room`.

Brick provides both generic (such as **AHU**) and specific classes of equipment (such as a **RoofTop-Unit** AHU). A building represented in Brick can specify the specific subclasses, or if that information is not available, instantiate a generic class. Line 8 is a common construct in Brick queries which accounts for this type of uncertainty in how Brick represents buildings. This sub-query returns all entities `?ahu` that are either an instance of a subclass of `brick:AHU` or an instance of `brick:AHU` itself.

An application that does not require specific features of such subclasses may want to query for the generic class rather than exhaustively specify every possible subclass. Because SPARQL and RDF do not support the object-oriented programming style of classes, the SPARQL query itself must specify the semantics of the type-inheritance: entities that instantiate the generic class directly, or entities that instantiate a subclass of the generic class.

After declaring the types of the entities involved, the query restricts the set of relationships between the entities on lines 9 and 10 to determine which pairs of entities are connected. Line 9 finds all HVAC zones downstream of a particular AHU by following a chain of `brick:feeds` relationships (the + indicates that 1 or more edges can be traversed as long as the edges are of type `brick:feeds`). Line 10 links the identified HVAC zones with the rooms they contain. The correct relationships to use can be determined from the Brick relationship list (Table 1).

This example query also illustrates an important quality of Brick queries: establishing a link between two entities (even across different subsystems such as HVAC and spatial) does not require explicit knowledge of all intermediary entities. Rather, the query denotes the relevant entities and relationships: the query in Figure 7 is indifferent to whatever building-specific equipment and details lie between an Air Handler Unit and the end zones. This is possible because the relationships between those entities all use Brick's `brick:feeds` relationship. What's more, the query accomplishes this using only a few, straightforward expressions to return the relevant triples from the collection of thousands of entities and relationships present in the building.

5. APPLICATIONS

A successful building schema must be able to capture the necessary relationships and entities required by the family of possible applications. Bhattacharya et al. establish that current industrial standards lack the ability to sufficiently describe how those pieces of equipment relate to each other [13]. Brick instead chooses to represent the set of useful entities and relationships required by the family of possible building applications. In this section, we construct this set by pulling a representative example from each of the eight common application dimensions identified by Bhattacharya et al. [13]. We determine the effectiveness of the schema to be how many of these entities and relationships it can capture.

5.1 Designing Relationships

We use these representative applications to establish the set of required relationships as well as the domains of those relationships. Relationships define how entities are associated, which for a given entity may include:

- **Taxonomy:** what class or classes of things define an entity
- **Location:** which building, floor and room an entity is in, but also where in the room it is
- **Equipment Connections:** what equipment an entity is connected to, and how it is connected
- **Equipment Composition:** what equipment an entity is a part of, or what equipment is a part of it
- **Subsystem:** how the entity is situated in a building subsystem such as HVAC, Electrical or Lighting
- **Monitoring:** what measures the entity or what it measures

Portability and orthogonality are two primary concerns in designing the set of relationships to include in an effective ontology. When describing or reasoning about a building, the set of possible relationships between any two entities (i.e. set of named edges between any two nodes) should be small enough and well-defined

Entities	Occupancy Modeling [23]	Energy Apportionment [22]	Web Displays [10]	Model-Predictive Control [32]	Participatory Feedback [25]	Fault Detection and Diagnosis [31]	NILM [27]	Demand-Response [33]
Sensors								
Temp Sensor	X					X		
CO2 Sensor	X							
Occ Sensor	X	X			X			
Lux Sensor		X			X			
Power Meter	X	X	X		X		X	X
Airflow Sensor			X					
Equipment								
Generic							X	X
HVAC	X	X	X			X		
Lighting	X	X			X			
Reheat Valve			X			X		
VAV			X	X				
AHU				X		X		
Chilled Water			X			X		
Hot Water			X			X		
Locations								
Building				X		X		
Floor				X	X		X	
Room	X	X	X	X	X		X	
HVAC Zone	X		X	X				
Lighting Zone	X				X			
Relationships								
Sensor `isLocIn` Loc.	X	X			X		X	
Equip `isLocIn` Loc.		X			X		X	X
Loc. `hasPart` Loc.		X		X	X			
Loc. `hasPoint` Sensor	X	X			X		X	X
Equip `hasPoint` Sensor	X		X			X	X	X
Equip `hasPart` Sensor			X			X	X	X
Equip `feeds` Zone	X			X	X			
Equip `feeds` Room	X			X			X	
Equip `feeds` Equip			X	X			X	
Zone `hasPart` Room	X			X	X			

Table 2: This table shows at a high level which entities and relationships are required by each of the eight representative applications.

such that the "correct" relationship should be obvious. This *orthogonality* reduces the possibility of inconsistency across buildings. Taken to its extreme, orthogonality informs a set of relationships that are specific and non-redundant, which can lead to overfitting the set of relationships for a particular building or subsystem. To support the goal of designing a unified metadata across many buildings, these relationships must also be sufficiently generic to be *portable* to many buildings.

Resolving these two tensions leads to the set of relationships listed in Table 1. We demonstrate here that this set of relationships is sufficient to cover the requirements of the representative applications. The specific entities and relationships each application requires are listed in Table 2. We implemented the eight applications in Table 2 and ran them on the five buildings; the results are collated in Table 3. The actual points exposed for each building by the BMS are the primary limiting factor for whether or not each application runs on a building: if a BMS exposes no lighting points, then

	Building				
Application	Soda	EBU3B	GTH	GHC	Rice
Occupancy [23]	232	244	139	366	11
Energy Apportionment [22]	-	-	302	-	4
Web Displays [10]	513	697	81	65	106
MPC [32]	482	482	69	428	110
Participatory Feedback [25]	-	-	253	-	-
FDD [31]	136	229	12	229	-
NILM [27]	-	6	82	-	-
Demand Response [33]	144	1428	24	2490	4

Table 3: Number of matching triples in each building for the SPARQL queries consisting the eight applications. A non-zero number indicates that the application successfully ran on the building. Buildings with '−' did not have any relevant points exposed in the BMS.

a lighting application cannot run. In addition, applications have to account for the diversity of points across buildings: Brick defines synonym tagsets where possible, but there will always be a degree of disambiguation that is application-specific.

Brick allows applications to write *portable queries* that identify relevant resources in a building-agnostic manner. An application can then adapt its behavior to the set of returned resources, likely using some API to interact with the required points. For this reason, we implement each of the applications as a set of SPARQL queries that return the set of relevant entities and relationships.

5.2 Results

We implement eight applications — one from each of the application categories in [13] — as a set of SPARQL queries identifying the relationships in Table 2. These queries do not contain the full operating logic of the application, but rather serve as a bootstrapping step for the application to discover the set of available and relevant resources. Table 3 contains the results of running these queries over the five buildings for each of the applications.

The applications that ran on the majority of buildings did so because they rely on HVAC and construction/spatial information readily exposed by the BMS. This includes VAVs, AHUs, HVAC zones, relevant sensors, and how these connect to each other. The Participatory Feedback application operates entirely on lighting controls, which rarely appear in a BMS, thus limiting its portability. Likewise, the NILM application relies on power meters, which also may not be integrated into the BMS.

The primary challenge in developing portable queries was accounting for the variance in relationships across buildings. For example, a zone temperature sensor may have *either* of the two connections indicated in Figure 4: it may have an `isPointOf` relationship with an HVAC zone entity or a VAV entity. These inconsistencies arise from differences in building construction and the representation of the points in the BMS. It is possible to account for these differences in SPARQL to construct truly portable queries.

5.3 Example Application: ZonePAC

The ZonePAC [10] application incorporates monitoring and modeling of HVAC zone behavior and power usage with occupant feedback to provide a platform for occupants to directly contribute to the efficacy and efficiency of a building's HVAC system. ZonePAC requires the following relationships:

- the mapping of VAVs to HVAC zones and rooms
- the heating and cooling state of all VAVs in the building
- the mapping of VAV airflow sensors to rooms

```
SELECT ?airflow_sensor ?room ?vav
WHERE {
  ?airflow_sensor rdf:type/rdfs:subClassOf*
    brick:Supply_Air_Flow_Sensor .
  ?vav rdf:type brick:VAV .
  ?room rdf:type brick:Room .
  ?zone rdf:type brick:HVAC_Zone .
  ?vav brick:feeds+ ?zone .
  ?room brick:isPartOf ?zone .
  ?airflow_sensor brick:isPointOf ?vav .
}
```

Figure 8: ZonePAC query for airflow sensors and rooms for VAVs. The query returns all relevant triples for ZonePAC to bootstrap itself to a new building.

- all available power meters for heating or cooling equipment

Immediately, the requirements of this application outstrip the features provided by other metadata solutions. ZonePAC needs to relate entities across subsystems typically isolated or ignored in modern BMS: the spatial construction of the building, the functional construction of the HVAC system, and the positioning of power meters in that infrastructure. Brick simplifies this cross-domain integration and makes it possible to retrieve all relevant information in a few simple queries.

To identify the airflow sensors and rooms served for each VAV, the application uses the query in Figure 8. The application uses Brick's synonyms to capture both `Discharge Air Temperature Sensors` as well as `Supply Air Temperature Sensors`. Airflow sensors have an `isPointOf` relationship with the VAVs, and the rest of the relationships in the application mirror those in Figure 4. The "Web Displays" row of Table 3 contains the results of running ZonePAC over the five buildings.

6. CASE STUDIES

We showcase the effectiveness of our schema by converting five buildings with a wide range of BMS, metadata formats, and building infrastructure into Brick. We discuss the challenges faced in converting various buildings into Brick to demonstrate Brick's robustness as well as to provide guidance for those facing similar challenges when using Brick.

Table 4 contains a summary of the construction and infrastructure of the five buildings, and how well Brick was able to capture their exposed BMS points. To evaluate the effect of "overfitting" Brick's tagsets to the set of known BMS points, we examined the % of BMS points covered by Brick's tagsets for Rice Hall and Soda Hall both before and after we incorporated their specialized points into Brick. Using an unaltered Brick, we matched 93.5% and 93.1% of Rice and Soda Hall's BMS points respectively. After incorporating the BMS-specific points, they scored 98.5% and 98.7% respectively, using Brick's class hierarchy to avoid compromising generalizability. Thus, we can conclude that Brick's tagsets do not overfit the set of five buildings. Examining Table 4, we can see that Brick matches the majority of points in all five buildings.

6.1 Gates Hillman Center at CMU

The Gates and Hillman Center (GHC) at Carnegie Mellon University is a relatively new building, completed in 2009, with 217,000 square feet of floor space, 9 floors, and 350+ rooms of various types (offices, conference rooms, labs), and contains over 8,000 BMS data points for various HVAC sensors, setpoints, alarms, and commands. CMU contracts with Automated Logic for building management.

Building Name	Location	Year	Size (ft^2)	# of Points	% Tagsets Mapped	# Relationships Mapped
Gates Hillman Center (GHC)	Carnegie Mellon Univ., Pittsburgh, PA	2009	217,000	8,292	99%	35,693
Rice Hall	Univ. of Virginia, Charlottesville, VA	2011	100,000	1,300	98.5%	2,158
Engineering Building Unit 3B (EBU3B)	UC San Diego, San Diego, CA	2004	150,000	4,594	96%	8,383
Green Tech House (GTH)	Vejle, Denmark	2014	38,000	N/A	N/A	N/A
Soda Hall	UC Berkeley, Berkeley, CA	1994	110,565	1,586	98.7%	1,939

Table 4: Case Study Buildings Information. GTH does not expose any BMS points, so numbers are not available.

The GHC includes 11 AHUs of different sizes serving multiple zones: three small AHUs serve one giant auditorium, one big laboratory and three individual rooms respectively. Eight large AHUs supply air to more than 300 VAVs. GHC's HVAC system also contains computer room air conditioning (CRAC) systems which are equipped with additional cooling capacity to maintain the low temperature in a computer room and fan coil units systems to provide cooling and ventilation functions. Despite the existence of these more esoteric subsystems, Brick matched 99% of GHC's BMS points, with the remaining points being too uncommon to be required by most applications (such as a `Return Air Grains Sensor` which measures the mass of water in air). The direct translation of BMS tags into Brick was relatively simple, only requiring a mapping between the human-readable BMS data points and Brick for each unique data point type.

The major challenge in converting the GHC to Brick was determining the relationships between pieces of equipment, which were not encoded in the BMS's labels. While the information is available through an Automated Logic GUI representation of the building, there was no machine readable method of understanding which VAV was related to which AHU. This required examining the building plans directly to incorporate these relationships (of which there were over 400). While a barrier to generating a Brick representation of a building, this example also shows the benefits that such a representation provides. Instead of being reliant upon manually examining a GUI to determine relationships between equipment, the Brick representation shows these relationships in both human and machine readable formats once represented in Brick.

6.2 Rice Hall at UVA

Rice Hall hosts the Computer Science Department at the University of Virginia. The building consists of more than 120 rooms including faculty offices, teaching and research labs, study areas and conference rooms distributed over 6 floors with more than 100,000 square feet of floor space. The building contracts with Trane for building management.

Rice Hall contains 4 AHUs associated with more than 30 Fan Coil Units (FCU) and 120 VAVs serving the entire building. Besides the conventional HVAC components, the building features several different new air cooling units, including low temperature chilled beams and ice tank-based chilling towers, an enthalpy wheel heat recovery system, and a thermal storage system. The building also contains a smart lighting system including motorized shades, abundant daylight sensors and motion sensors. Rice Hall's BMS points are easily interpretable for conversion to Brick despite it containing some uncommon equipment such as a heat recovery and thermal storage systems, as part of the building design as an energy-efficient "living laboratory". Moreover, the set of relationships defined by Brick sufficiently captured how the uncommon equipment related to other components of the HVAC system.

A few of these points, such as `Ice Tank Entering Water Temperature Sensor`, are specific to Rice Hall among the set of five buildings we examined. Nonetheless, Brick's structure allows for the clean integration of new tagsets into the hierarchy without disrupting the representation of existing buildings.

6.3 Engineering Building Unit 3B at UCSD

The Engineering Building Unit 3B (EBU3B) at University of California, San Diego hosts the department of Computer Science and contains offices, conference rooms, research laboratories, an auditorium and a computer room. The building was constructed in 2004 and has 150,000 square feet of floor space with over 450 rooms. The BMS of EBU3B is provided by Johnson Control Inc., and contains more than 4500 data points, most of which belong to the HVAC system and power metering infrastructure.

The HVAC system consists of a single AHU that supplies conditioned air to 200+ VAV units and some FCUs. There is a CRAC system serving the computer room and there are exhaust fans for all kitchens and restrooms. The HVAC system also consists of Variable Frequency Drives (VFD), valves, heat exchangers and cooling coils to facilitate operation of AHU and CRAC. Brick's schema provides the necessary tagsets and relationships to account for all of these components and their data points. The university central power plant provides the hot and cold water necessary for the HVAC and domestic hot water system. The corresponding sensors that measure the hot and cold water use were modeled in Brick, but the central plant was left out as it was not part of the building. The building contains meters that measure power consumption of various subsystems: lighting, computer room, HVAC system and elevator. The meters were associated to the corresponding systems with `isPointOf` relationship as required by the applications.

An issue that arose in mapping EBU3B to Brick was that the AHU supply air was divided into two parts that supplied air to two wings of the building. Brick currently does not provide a means to model this division of supply air which has proven relevant to the diagnosis of various faults. Moving forward, Brick can address this by modeling the AHU discharge air as a *resource*, which can also help model other concepts such as cold water supply from a central plant. Alternatively, the discharge air can be attributed to the cooling coil that modulates its temperature, and the cooling coil can be said to feed discharge air to the terminal units.

Additionally, EBU3B's BMS contains data points corresponding to Demand Response events, which exposes an interesting conflation of the representation and operation of the building. Because BMS are typically written as monolithic applications over building-specific representations, they must incorporate external signals such as Demand Response into the set of BMS points. Conversely, Brick decouples the resources and infrastructure of a building from the processes managing the building.

6.4 Soda Hall at UC Berkeley

Soda Hall, constructed in 1994, houses the Computer Science Department at UC Berkeley. It comprises mostly of closed small to medium sized office spaces, where either faculty or groups of graduate students sit. The BMS system, provided by the now-defunct Barrington Systems, exposes only the sensors in the HVAC system.

The HVAC system of the building runs on pneumatic controls, and comprises 232 thermal zones. The zones on the periphery of the building have VAVs with reheat, while the other zones do not. For a VAV with reheat, the same control setpoint indicates both the amount of reheat and the amount of air flowing into a zone, by using a proprietary value mapping mechanism. While the value mapping is building-specific, Brick can express the fact that the same sensor controls both the reheat and air flow by labeling the point as a subclass of both reheat and airflow tagsets. The logic for communicating with the point correctly would be handled by some other system; Brick simply identifies the available points.

Unique to the set of buildings presented here, the operational set of Soda Hall's HVAC components is not static. Soda Hall contains a redundant configuration of chillers, condensers and cooling towers. At any point of time, one of each of these systems is operational, while the others are kept as hot standby. An isolation valve setpoint indicates which of the redundant subsystems is currently operating. Brick completely expressed the redundant subsystem arrangement, but the equipment contained several unique points such as `On Timer` for the chiller subsystem that had to be added to Brick's tagsets.

6.5 Green Tech House

The Green Tech House (GTH), constructed in 2014, is a 38,000 square feet office building that houses a range of organizations and companies with different business purposes. It is a three-story building containing 50 rooms comprising office spaces, a cafeteria, meeting rooms and bathrooms. GTH's BMS limits our access to a subset of the available points, so calculations of how well Brick covered the points (Table 4) are not applicable.

GTH's BMS exposes some lighting points, but has a substantially different HVAC system. The HVAC system heats air centrally in order to distribute air to zones with cooling capabilities. Although the building documentation don't refer to groups of equipment as AHUs or VAVs, equivalents are present.

A single AHU recovers heat from the return air using a rotary heat exchanger. This heat exchanger is the first in a cascade of two between outside air and supply air. Outside and return air are never mixed. The pressure of return and supply air of the north and south side of the building are measured separately.

AHU heating relies on the second heat exchanger which uses a hot water loop. Additionally, most rooms have either radiators or floor heating. These are supplied by independent hot water loops, heated by district heating.

The main challenge in converting GTH's points to Brick was accounting for the acute difference in infrastructure compared to Brick's construction. The current version of Brick assumes that an HVAC system contains VAVs and AHUs, which is not strictly true for GTH. For the purposes of running Brick applications, we mapped GTH's HVAC components onto Brick's model; however, this is not an ideal solution. It is not Brick's goal to disguise the construction of building subsystems, but rather to abstract away the intricacies of subsystem composition between buildings. Future versions of Brick will account for such variation in subsystem equipment and construction.

7. CONCLUSION AND DISCUSSION

This study has addressed an important open problem that was referenced in [11, 13, 14, 19, 21] — Can there be a building metadata schema that is complete, expressive and usable? Completeness entails that the schema expresses the vast majority of the points found in large commercial buildings that facilities managers thought worthy to include in the BMS tags, as well as the points and relationships necessary for important building applications. Expressiveness entails that the schema's namespace is well-defined, and one can implement applications using it rather than using the adhoc namespace of the particular building. Usability entails that the schema is understandable, and the total amount of work required to convert the existing buildings into the schema is limited and bounded, and can perhaps be automated at a reasonable scale using solutions such as [11] and [14].

We have defined a schema, Brick, that we believe is a strong candidate to solving this open problem. Brick builds upon prior work and introduces a number of novel concepts that we believe adresses this open problem. Brick uses clear tags and tagsets to specify sensors and subsystems in a building. It defines an ontology and a class hierarchy for this list of tags and tagsets. Relationships are represented as triples, which allows us to leverage existing tools to build and query the resulting building representations. Brick proposes Functional Blocks to abstract out complexity but also aid in system composition and hierarchies. Finally, Brick uses the notion of synonyms to equate sensors and subsystems similar in function.

Brick is complete, capturing an average of 98% of BMS data points across five diverse buildings comprising almost 15,700 data points and 615,000 sq-ft of floor space. Brick is expressive, successfully running eight canonical applications on these buildings. Four applications ran on all five buildings, while the remaining applications ran on buildings whose BMS exposed the requisite points. Brick is usable, as converting each of the buildings' legacy metadata to the normalized schema took no more than 20 manhours. The resulting schema is understandable and easy to query as shown in Figures 6, 7 and 8.

Brick tries to maintain orthogonality in describing tagsets and relationships, i.e. there should be a single straightforward way to describe an entity, collection of entities and their inter-relationships. The functional block model in Brick proves very helpful in abstracting out the complexity and heterogeneity of particular subsytems in buildings, while aiding system composition and hierarchies. For instance, functional blocks' easy replication, instantiation and composition can help Brick quickly specify the numerous VAV zones and its associated points in a building. Functional blocks also help hide the subtle differences between complex subsystems such as an AHU.

Brick distinguishes itself from other industry standards in the building-metadata domain by virtue of its process of the production of open reference implementations on real buildings serving as a means of evaluating the effectiveness of the solution. Developing a reference standard through such a process has been successful in other fields, most notably the IETF [7] for internet protocols and algorithms. The code, schema, and reference implementations of all the buildings in our testbed are available at [1].

We hope that our solution to this well-defined open metadata problem lays the foundation for industry and academic collaboration to produce bonafide standards that could be transformative in producing energy efficient buildings and portable applications.

8. FUTURE WORK

There are two primary directions of future work on Brick. First, Converting legacy label/SCADA-based metadata to Brick can be automated further drawing from techniques in [14, 11, 19, 21, 29]. Such techniques should also be augmented to help a facilities manager infer which are the best tags/tagsets and relationships that a particular label should be mapped to. Second, Brick needs an equivalent of a model-checker to ensure the correctness of the generated metadata. Manually debugging why certain applications were not running on certain buildings in our case study required quite a bit of effort, and the answer was invariably some error in translating existing labels to tags/tagsets, misuse of tags when they should not have been used, or expressing semantically wrong relationships.

9. REFERENCES

[1] Brick Schema. https://github.com/BuildSysUniformMetadata/GroundTruth.

[2] OWL Namespace. http://www.w3.org/2002/07/owl#.

[3] Project Haystack. http://project-haystack.org/.

[4] RDF Concepts Namespace. http://www.w3.org/1999/02/22-rdf-syntax-ns#.

[5] RDF Schema Namespace. https://www.w3.org/2000/01/rdf-schema#.

[6] SPARQL Query Language. https://www.w3.org/TR/rdf-sparql-query/.

[7] The Internet Engineering Task Force (IETF®). https://www.ietf.org/.

[8] Turtle. https://www.w3.org/TR/turtle/.

[9] U. E. I. Administration. User's guide to the 2012 cbecs public use microdata file. *Commercial Buildings Energy Consumption Survey (CBECS)*, page 33, May 2016.

[10] B. Balaji, H. Teraoka, R. Gupta, and Y. Agarwal. Zonepac: Zonal power estimation and control via hvac metering and occupant feedback. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 1–8. ACM, 2013.

[11] B. Balaji, C. Verma, B. Narayanaswamy, and Y. Agarwal. Zodiac: Organizing large deployment of sensors to create reusable applications for buildings. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 13–22. ACM, 2015.

[12] V. Bazjanac and D. Crawley. Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings. In *Proceedings of Building SimulationâĂŹ99*, volume 2, pages 661–667, 1999.

[13] A. Bhattacharya, J. Ploennigs, and D. Culler. Short paper: Analyzing metadata schemas for buildings: The good, the bad, and the ugly. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 33–34. ACM, 2015.

[14] A. A. Bhattacharya, D. Hong, D. Culler, J. Ortiz, K. Whitehouse, and E. Wu. Automated metadata construction to support portable building applications. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 3–12. ACM, 2015.

[15] D. Bonino and F. Corno. DogOnt – ontology modeling for intelligent domotic environments. In *ISWC - Int. Semantic Web Conf.*, volume 5318, pages 790–803. 2008.

[16] T. Cerovsek. A review and outlook for a âĂŸbuilding information modelâĂŹ(bim): A multi-standpoint framework for technological development. *Advanced engineering informatics*, 25(2):224–244, 2011.

[17] V. Charpenay, S. Kabisch, D. Anicic, and H. Kosch. An ontology design pattern for iot device tagging systems. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 138–145. IEEE, 2015.

[18] L. Daniele, F. den Hartog, and J. Roes. Study on semantic assets for smart appliances interoperability: D-s4: Final report. Technical report, European Union, 2015.

[19] J. Gao, J. Ploennigs, and M. Berges. A data-driven meta-data inference framework for building automation systems. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 23–32. ACM, 2015.

[20] N. GCR. Cost analysis of inadequate interoperability in the US capital facilities industry. *National Institute of Standards and Technology (NIST)*, 2004.

[21] D. Hong, H. Wang, J. Ortiz, and K. Whitehouse. The building adapter: Towards quickly applying building analytics at scale. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 123–132. ACM, 2015.

[22] M. Jahn, T. Schwartz, J. Simon, and M. Jentsch. Energypulse: tracking sustainable behavior in office environments. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*, pages 87–96. ACM, 2011.

[23] D. Jung, V. B. Krishna, N. Q. M. Khiem, H. H. Nguyen, and D. K. Yau. Energytrack: Sensor-driven energy use analysis system. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 1–8. ACM, 2013.

[24] M. J. Kofler, C. Reinisch, and W. Kastner. A semantic representation of energy-related information in future smart homes. *Energy and Buildings*, 47:169–179, 2012.

[25] A. Krioukov, S. Dawson-Haggerty, L. Lee, O. Rehmane, and D. Culler. A living laboratory study in personalized automated lighting controls. In *Proceedings of the third ACM workshop on embedded sensing systems for energy-efficiency in buildings*, pages 1–6. ACM, 2011.

[26] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. 1999.

[27] A. Marchiori and Q. Han. Using circuit-level power measurements in household energy management systems. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 7–12. ACM, 2009.

[28] J. Ploennigs, B. Hensel, H. Dibowski, and K. Kabitzsch. Basont-a modular, adaptive building automation system ontology. In *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*, pages 4827–4833. IEEE, 2012.

[29] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[30] S. Roth. Open green building xml schema: A building information modeling solution for our green world, gbxml schema (5.12). 2014.

[31] J. Schein, S. T. Bushby, N. S. Castro, and J. M. House. A rule-based fault detection method for air handling units. *Energy and Buildings*, 38(12):1485–1492, 2006.

[32] D. Sturzenegger, D. Gyalistras, M. Morari, and R. S. Smith. Semi-automated modular modeling of buildings for model predictive control. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 99–106. ACM, 2012.

[33] T. Weng, B. Balaji, S. Dutta, R. Gupta, and Y. Agarwal. Managing plug-loads for demand response within buildings. In *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 13–18. ACM, 2011.