Final Technical Report

Award Number:	DE-EE0008681
Recipient:	Regents of the University of California (Berkeley) California Institute for Energy and Environment)
Project Title:	**Skewering the silos: using Brick to enable portable analytics, modeling and controls in buildings**
Principal Investigator:	Therese Peffer
Project team members:	Gabe Fierro, Colorado School of Mines/NREL Paul Raftery, Carlos Duarte Roa, Center for the Built Environment, UC Berkeley Marco Pritoni, Michael Wetter, Anand Prakash, Lazlo Paul, LBNL Erik Paulson, Johnson Controls International
Report Submission Date:	30 Jan 2023
Reporting Period:	1 October 2019- 30 September 2022

Please reference this report as:

Fierro, Gabe, Carlos Duarte Roa, Paul Raftery, Anand Prakash, Marco Pritoni, Lazlo Paul, Michael Wetter, Erik Paulson, and Therese Peffer. 2022. Skewering the silos: using Brick to enable portable analytics, modeling and controls in buildings. Final Technical Report for US Dept of Energy project EE0008681. DOE-CIEE-08681.

Table of Contents

Acknowledgement

This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Building Technologies Office (BTO) Award Number DE-EE0008681 from Oct 2019 - Sept 2022.

The authors would like to thank all the researchers and institutions that have helped develop Brick from its initial ideation at the BuildSys conference in 2014, to development of Brick v1.0[1] in 2016, and over the course of this project from 2019 to 2022. The initial institutions that contributed to the creation of Brick include Carnegie Mellon University, IBM Research, University of California Berkeley, University of California Los Angeles (UCLA), University of California San Diego, University of Southern Denmark, and University of Virginia. Initial support came from the National Science Foundation, US Department of Energy, Innovation Foundation Denmark, Intel and Johnson Controls and others as mentioned on the Brick website (https://brickschema.org/).

We are grateful for the partnership with Johnson Controls especially Dr. Younchoon Park, Dr. Young M Lee, and Erik Paulson, and including match funding, over the course of this project.

[1] See Bhattacharya et al, 2015 (https://doi.org/10.1145/2821650.2821669) and Balaji et al, 2016 (https://doi.org/10.1145/2993422.2993577)

Executive Summary

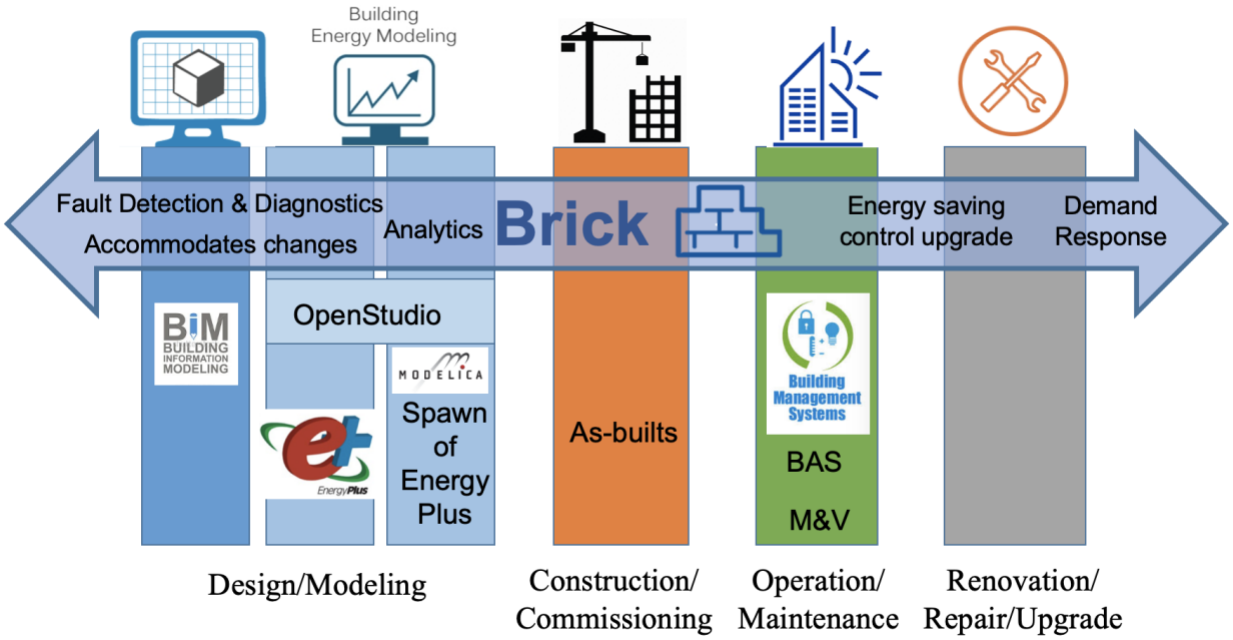
Nearly all large commercial buildings have heating, ventilation and air conditioning (HVAC) systems, lighting systems, safety and other systems controlled by a computer—a dedicated server with a building energy management system (BMS). However, these BMSs are proprietary with each building's assets (that is, fans, valves, pumps, and their setpoints) named and coded uniquely by the BMS vendor or engineer; building analytics and control algorithms are written specific to the assets and the building. Thus, any control updates or analytics to improve building performance—especially critical to reduce greenhouse emissions or improve load flexibility—are labor intensive and costly. In addition, the decreasing costs of sensors and IoT devices, as well as the increased penetration of networked monitoring and control systems, had led to an enormous amount of data available. Increased access to this operational data presents new opportunities to develop and deploy novel data-driven use cases including fault detection and diagnosis algorithms, intelligent controls, and grid interactivity. Such data-driven processes are particularly important because they can use data to automatically diagnose and understand a building, rather than relying upon expert-driven advice.

Although operational building data is now *available*, it is not *accessible* to software developers, building scientists, and other stakeholders for two primary reasons. First, this data is largely locked away in vendor-specific silos with proprietary interfaces and limited facilities for data discovery and analysis. Second, there is no standard labeling or organizational scheme for the data produced by buildings. Together, these characteristics pose significant challenges for the development and configuration of intelligent, data-driven building processes. Each piece of software must be manually configured (i.e., "ported") to each building, a painstaking and error-prone process.

To address these issues, the Brick schema was developed so the same analysis or control algorithms can work on a variety of buildings if each is digitally represented in a Brick data model. The motto is "write once, run anywhere": we want to "reuse" the diagnostics, analytics, and control algorithms (such as ASHRAE Guideline 36) in multiple buildings, not have to write a unique proprietary algorithm for each building. In addition, we want to translate across different data representations in the building lifecycle—from design to construction to operation and maintenance (Figure 1).

The Brick schema is an open source, permissively licensed development for standardizing digital representations of buildings and their data sources. Specifically, Brick is a graph-oriented metadata ontology that defines a taxonomy of types for categorizing common assets, data sources and other entities in buildings; it also defines a family of expressive properties that describe the relationships between these entities. Together, the taxonomy and properties enable the creation of machine-readable digital representations of buildings (a Brick *model*) that use a consistent vocabulary. A consortium of universities and industry partners collaborated in the development of Brick using field trials and testing in developing eight representative applications over six buildings. Brick succeeded in these initial experiments because of the flexibility of the underlying graph model and the extent of the vocabulary it defined.

Figure 1: Skewering the silos: the Brick schema allows translations across the data siloes in a building's lifecycle.



The goal of this project was to further the development of Brick to extend it beyond an academic project with demonstrated success in a small field study, to a practical choice for industrial and commercial stakeholders seeking to realize value from building data. To do this, we executed four objectives: (1) expand the Brick schema including its modeling capabilities and vocabulary, (2) develop tools for integrating Brick with existing digital technologies and representations in buildings, (3) develop an open-source analytics platform to facilitate use of Brick in delivering data value, and (4) demonstrate Brick-driven analytics and controls in real settings. Through these objectives, we have established and begun to execute a technology transition plan that has dramatically increased the impact and adoption of Brick in industrial, commercial, and academic settings.

Brick Schema Expansion

Over the course of this project, the team has delivered five new releases of the Brick ontology: 1.1.0 (Jul 2020), 1.1.1 (Jan 2021), 1.2.0 (Feb 2021), 1.2.1 (Aug 2021), and 1.3.0 (Oct 2022). Each of these releases expanded the set of building assets, data sources, properties and subsystems that can be modeled using the Brick schema. The specific additions were informed and implemented by a growing community of international contributors. Today, Brick defines over 1200 classes (equipment, data sources, assets, building components), 44 entity properties (datasheet and nameplate characteristics), and 33 relationships. This includes substantial efforts to expand Brick's capabilities to capture energy storage and generation systems, plug loads and end-use devices, radiant systems, and initial support for heat pumps. We have also published extensions to Brick which permit modeling of occupants and occupant data as well as

relationships between Brick entities and building information models, building control networks, and archival data storage.

Brick Integrations

The original release of the Brick schema was distributed as a standalone solution whose effective usage required software engineering as well as graph data model expertise. Over the course of this project, we have developed five integrations between Brick and existing technologies and standards that significantly simplify the process of bootstrapping a Brick model from for a real building. These integrations also make Brick part of a larger ecosystem, ensuring it is a "safe bet" for stakeholders: Brick *complements* existing technological investments rather than competing with them.

We have developed Brick translators for multiple technologies that span the lifecycle of a building and align with existing industry standards: BuildingSync (an energy auditing schema based in part on the ASHRAE 211 standard), Modelica (language for simulation models), Control Description Language (vendor-agnostic specifications of control sequences, being incorporated into the ASHRAE 231P standard), Project Haystack (a prevailing tag-based metadata solution), and BACnet (a standard network protocol formalized as the ASHRAE 135 standard). In addition, we have also developed alignments with emerging metadata efforts: the RealEstateCore ontology (focusing on spatial building elements and property management) and the VBIS asset management system. All of these integrations and alignments are released under permissive open-source software licenses, and have assisted in the adoption of Brick by making it easier for stakeholders to use Brick with their existing building technologies.

Open Source Analytics Platform

To demonstrate how Brick supports the execution of *portable* analytics applications, we created and currently host an open analytics platform—Mortar—that provides access to hundreds of millions of data points spanning over 50 buildings. Each of these buildings is modeled using Brick; users and applications use the Brick model to discover and retrieve the available data. The Mortar platform presents a fundamentally different approach to organizing and disseminating building data. Existing building data is available largely through building management systems, which organize data by annotating data streams with human readable but ultimately unstructured labels. Contextual information which relates these data streams to the building and its subsystems is typically not available in a machine readable form. By pulling this contextual information out into a standard form, a Brick model, it becomes possible for applications to discover and filter through data for many buildings at once without the user having to develop a specialized understanding of each of those buildings and their data. This introduced the notion of *portable building applications*: applications which can execute over many buildings with minimal or zero reconfiguration.

To prove the impact of portable building applications, we have released an initial library of portable building analytics, available online as open-source programs, which execute on the Mortar analytics platform.

Demonstration of Brick-Driven Controls

Through this project, we have demonstrated one vision of the next generation of intelligent, data-driven building analytics and controls. Standardized sequences of operation such as ASHRAE Guideline 36 for high-performance HVAC systems can be difficult to deploy on real buildings because of the heterogeneity of the building subsystems and their labels – the same issues that plague deployment of data-driven analytics. We have demonstrated the use of Brick to capture the required metadata about a building that allows a program to integrate a standardized building control language (CDL) with a standard communication protocol (BACnet) to implement these high-performance controls on buildings with minimal manual intervention. What is significant is that the entire controls deployment was achieved with publicly available, non-proprietary resources that can be executed over multiple buildings with

Thus, we used standardized sequences of operation (ASHRAE Guideline 36) to define the best practice controls, an open-source schema (Brick) to represent the required metadata in a structured format, a standardized communications protocol (BACnet) to communicate with the controllers, and (proposed) standardized building controls language (CDL) to implement these controls. These are all non-proprietary resources that are publicly available and can be used in a scalable manner that is *portable between buildings*. That applies even though those buildings have different automation systems, point names, HVAC topologies, etc. To our knowledge, this is the first time the ’full stack’ of these different components has been implemented to perform closed loop control in a real building. The controls continue to operate in the building as writing this report.

Goals and Objectives

The **goal** of this project is to develop the nascent open-source common data schema, Brick, into a demonstrated and tested tool that, through the implementation of advanced algorithms and analysis, can enable the scaling of major energy savings, improved building performance, and grid responsiveness across commercial buildings at low cost.

The **objectives** of the project are to:

1. Expand the Brick schema (the categories and names of building subsystems and equipment, and relationships among them)
2. Develop tools and translators for adding information to and from Brick models
3. Develop an open-source building analytics platform,
4. Test open-source building analytics and controls (e.g. ASHRAE Guideline 36[2], Open Building Controls (OBC)[3]) on several buildings, and
5. Develop a technology transition plan.

We have expanded the Brick schema: Over the course of the project, we have produced two minor and three major releases of the Brick schema: 1.1.0 (Jul 2020), 1.1.1 (Jan 2021), 1.2.0 (Feb 2021), 1.2.1 (Aug 2021), and 1.3.0 (Oct 2022). Each of these releases incorporated further extensions of the Brick ontology, as detailed in this report. In addition, we released an occupant-oriented extension of the Brick ontology for modeling building occupants and occupant data.

We developed tools and translators: We have implemented and released translators (which transform or infer Brick metadata from existing sources) for five protocols and data formats: BACnet (standard communication protocol), Modelica/CDL (modeling and control languages), IFC (building information model), Project Haystack (metadata tagging standard) and BuildingSync (energy auditing schema). In addition, we have established semantic alignments (obviating the need for translation) with the VBIS asset classification system and the RealEstateCore ontology.

We developed an open-source building analytics platform, Mortar: Mortar has been released under a permissive open-source license, and is currently being used to host more than 100 million data points spanning more than 50 real buildings. Each of these buildings is represented by a Brick model, facilitating data discovery and consistent interpretation of data with respect to the building. The platform implements an API for data discovery and retrieval over all buildings in the platform, and has been used to develop several building analytics applications.

We developed and tested several buildings analytics and controls: using the Mortar platform (which incorporates the Brick schema), we developed and tested 5 analytics and fault detection applications that each ran on multiple buildings with minimum reconfiguration. We also developed and tested two control applications—duct static pressure reset control and hot water supply temperature reset control—that were executed on real buildings with the help of Brick models of those buildings.

[2] http://gpc36.ashraepcs.org/
[3] http://obc.lbl.gov/

We developed a technology transition plan for Brick, including significant industry engagement, release of open-source tools, outreach to stakeholders, and public presentations and publications of the work achieved during this project. We have assisted in the creation of the Brick Consortium, Inc., a 501(c)(6) organization intended to support the research and development of the Brick ontology. The consortium currently consists of academic representatives from UC Berkeley, Colorado School of Mines, and Carnegie Mellon University; it also has industrial representation from Carrier, Clockworks Analytics, Johnson Controls Inc, Mapped, Schneider Electric, and Siemens. In addition, we have held several webinars and workshops, published 16 publications and 1 PhD thesis, and released many public materials that educate and evangelize Brick.

Project Activities

Brick endeavors to standardize descriptions of components in buildings through a dictionary of building terms, a set of relationships among these pieces, and a data model that integrates Brick with other tools and data models. This open-source effort "skewers the silos" of independent data models currently marking all points of the life of a building, from design to construction to operation. A flexible but standard schema can enable "portability" of analytics, modeling, and controls from one building to another.

This project brought together researchers in computer science, mechanical engineering, and building science with expertise in building modeling and controls to further develop Brick, to increase its usability, and increase its adoption.

The project was divided into three budget years with five tasks in each period. The following describes each task's progression over the three years.

Task 1: Expand Brick Schema

The goal of this task was to identify and select new metadata (descriptive annotations for data sources) to be added to the Brick schema, expand the Brick schema extensions after defining the most appropriate tools and applications, and then release the expanded schema in the public realm. These new metadata were informed by our experiences with (a) building equipment, sensors, and other data sources, and (b) the applications and other use cases which are potential consumers of a Brick model. A Brick *model* is a digital representation of a building and its composing assets (e.g., equipment, locations) and data sources.

Demonstrate Brick models for >50 buildings

The first milestone of this task (1.1.1) was to demonstrate Brick models for at least 50 buildings, with each model "complete" to the extent that they enable at least two control and/or analytics applications.

We developed Brick models capturing the structure of Heating Ventilation and Air Conditioning (HVAC) systems, lighting systems, spatial composition, and building-level metering for 108 buildings in California. 92 of the buildings are large office buildings with Air Handling Unit (AHU)-based HVAC systems, and the rest are smaller commercial buildings that are Roof Top Unit (RTU)-based.

For AHU-based systems, the Brick models capture the flow of air through the equipment and into the spaces (AHU to Variable Air Volume (VAV) box to HVAC Zone) as well as the points of actuation and control associated with the equipment in those flows. To create these Brick models, we scraped the point labels available in the building management system (BMS) and authored a Python program to infer from those point labels the names and types of equipment and in the building and the connections between them. We then associated the data sources (points) exposed in the BMS with the inferred equipment.

For the RTU-based systems, the Brick models capture the associations between thermostats, the packaged rooftop units, and the zones and rooms conditioned by the rooftop units. We authored a Python program to extract this structure from exported Revit models (a type of Building Information Model containing structural and mechanical information about a building). Because

buildings with RTU-based systems typically do not contain unified digital infrastructure such as a BMS, we augmented the information available in the Revit model with manufacturer-specific "scrapers" that extracted metadata about any networked lights, thermostats, plug load controllers and other consumer-level devices installed in the building. This allowed us to augment the Brick model with metadata beyond what was immediately available in the BIM.

These Brick models are complete enough to enable at least two applications: the detection of "rogue zones" (zones whose airflow or temperature is lower than the setpoint for extended periods) and stuck dampers.

Expand Brick schema to represent 80% of large commercial buildings

Through year 1, we identified the set of end-use applications which could benefit from access to a standard digital representation of building data source (i.e., Brick), and a set of existing building software tools which would either be potential sources or consumers of Brick metadata. We surveyed the metadata required by these tools and applications and expanded the Brick schema to include all necessary metadata definitions. To evaluate the coverage of the Brick ontology over large commercial buildings in the US, the research team developed two approaches: a *top-down approach* using a national survey and a *bottom-up approach* based on BMS data from several representative buildings.

Top-down approach

The U.S. Energy Information Administration (EIA)'s Commercial Buildings Energy Consumption Survey (CBECS) is the only nationally representative source of statistical information on the nation's commercial buildings, including their energy-related building characteristics and energy usage data. The CBECS survey has been conducted on a sample of 6720 buildings representative of commercial buildings from the 50 States and the District of Columbia.

To appraise the Brick schema ability in representing metadata information in commercial buildings, we used the most recent available version of the *U.S. Energy Information Administration's Commercial Buildings Energy Consumption Survey* (*CBECS*) Data variable codebook (2012)[4] (XLS) as a comprehensive benchmark for our analysis.

We evaluated the Brick schema (v1.2.α) coverage by mapping the CBECS variables to the Brick ontology. Prior to the mapping, we selected relevant variables that lie within the scope of the Brick schema. While the CBECS data survey includes building characteristics (such as building structural characteristics, energy sources and uses), energy usage data and energy providers, the Brick ontology aims at describing the multitude of subsystems in buildings (HVAC, lighting, electrical, security, control systems), which are core to the development of smart analytics and control applications.

We selected variables that fall within the following categories: Buildings Geographic Features and Locations, Cooling/Heating Equipment Types and Controls, Lighting, Electricity Generation and Storage Technologies, Occupancy Control and Others. It is important to note that Brick has the ability to represent HVAC, Security, Safety Equipment and Locations concepts in much

[4] https://www.eia.gov/consumption/commercial/data/2012/xls/2012microdata_codebook.xlsx

more depth than CBECS. This is the case also for the actual relationships that describe interactions among devices and building spaces.

In version v1.2.α, Brick only covered 55% of the selected CBECS variables, but had the capacity to represent 98% of them. Relevant use cases have already been defined for 28% of those unrepresented concepts and their additions will be covered in future versions of the Brick schema (bringing coverage up to 83%). Most of these additions have been incorporated into the Brick 1.3.0 release, and more will emerge in minor 1.3.x releases.

On average, Brick was able to cover 71% of the selected CBECS concepts out of those represented across the 6720 buildings sampled. We see the need for a more focused and integrated assessment of the existing overlap between CBECS and Brick buildings data contexts and use cases.

Bottom-up Approach

Based on BMS points from three representative buildings, we evaluated the expansion of the Brick schema by comparing the coverage of Brick at the time this project began (v1.1) to version 1.2.α, which was current at the time of the analysis.

We evaluated the coverage of the Brick ontology using the building management system (BMS) points from three buildings: San Mateo Office Building (SMC), Sacramento Municipal Utility District Office Building (SMUD), and David Brower Center Office Building (DBC). We compared the coverage of the previous schema version of Brick (1.1) to 1.2.α. We define "coverage" as the percentage of unique *types* of BMS points that were wholly describable in the given version of Brick. We only evaluated the BMS points where we could extract information on the point to determine its intended measurement or purpose. This is especially true for DBC where unidentifiable points were more prominent. The total number of points downloaded from DBC's BMS was 4,333 but were left with 1,218 points that were identifiable. Some example point names that were unidentifiable include the following:

- NAE00806603BECA/Field Bus1.CW Pump 8.Analog Values.AV43
- NAE00806603BECA/Field Bus1.HW Pump 5.Analog Values.AV-38
- NAE00806603BECA/Field Bus1.Radiant Pump 3.Analog Values.AV-36
- NAE00806603BECA/Field Bus1.CW Pump 7.Binary Values.BV-21

SMC is a typical office building with variable air volume with reheat zone terminal units; both SMUD and DBC have high thermal mass radiant heating and cooling systems for their primary HVAC system. The results are summarized in Table 1 below. Since SMC uses a typical VAV system, there was already a 100% coverage with Brick version 1.1. However, SMUD and DBC contained systems, hydronic system points, and measurements not previously defined in Brick. These buildings contained a radiant system with their specialized measurements and setpoints (e.g., embedded temperature sensors and setpoints), underfloor distribution ventilation system, heat pumps, and exhaust systems. The additional Brick classes added to version 1.2.α increased the coverage by at least 10%.

Table 1: Summary of Brick ontology coverage on three different buildings using Brick version 1.1 and 1.2.α

Building	No. Points | No. Entities	Brick Version	Coverage of all point	Coverage of unique classes
SMC	506 | 15	1.1	100%	100%
		1.2.α	100%	100%
SMUD	1581 | 138	1.1	84%	81%
		1.2.α	99%	98%
DBC	1218 | 136	1.1	89%	88%
		1.2.α	99%	99%

Based on system types and points described in ASHRAE Guideline 36 (public standard used in industry), we evaluated whether 80% of the points/classes are covered in Brick v1.2.α. ASHRAE Guideline 36, High-Performance Sequences of Operation for HVAC Systems, was created to develop and maintain best-in-class standardized (air-side) HVAC control sequences. The document presents these sequences in reference to a set of HVAC equipment configurations and common Building Automation System "points". The coverage of Brick is evaluated based on the full set of systems and points mentioned in the Guideline 36 - 2018. The results of such analysis are illustrated in Table 2.

Table 2: Summary of Brick ontology coverage based on ASHRAE guideline 36 concepts

Type	Coverage in Brick 1.2.α
Points	86%
Equipment	81%

Feedback on expanded Brick schema

Another task in year 2 was to report on feedback from the Brick community. In 2021, the Brick forum had 178 members, which grew to 296 members in 2022. New members regularly post questions and answers to other questions. Technical users continue to engage on the Brick GitHub issue tracker, requesting extensions to and clarifications of the Brick ontology. Examples of Brick community engagement include expanding Brick to model district heating systems, expanding the set of definitions for electrical metering, expanding the set of room and location types supported by Brick, and a larger family of minor bug fixes and concept additions.

In order to better communicate with the Brick community, the Brick Roadmap page was launched (http://roadmap.brickschema.org), which tracks the immediate and future development plans for Brick.

Dr. Fierro with Johnson Controls International, who has helped to organize an Industry Consortium for Brick, organizes a set of bi-weekly Working Groups that bring community members together with the Brick development team around four distinct efforts: ontology development, conversion tools for Brick, curation of public reference models with timeseries data, and the development of Brick applications.

Finally, we note that Brick compares favorably to other ontology and schema (Table 3).

Table 3: Summary of Brick compared to other schema

Modeling Support	Brick	Project Haystack	IFC	BOT	SAREF
HVAC Systems	**yes**	**yes**	**yes**	no	no
Lighting Systems	**yes**	partial	**yes**	no	no
Electrical Systems	**yes**	**yes**	**yes**	no	no
Spatial Information	**yes**	no	**yes**	**yes**	no
Sensor Systems	**yes**	**yes**	generic	no	**yes**
Control Relationships	**yes**	no	generic	no	no
Operational Relationships	**yes**	no	generic	no	no
Formal Definitions	**yes**	no	**yes**	**yes**	**yes**

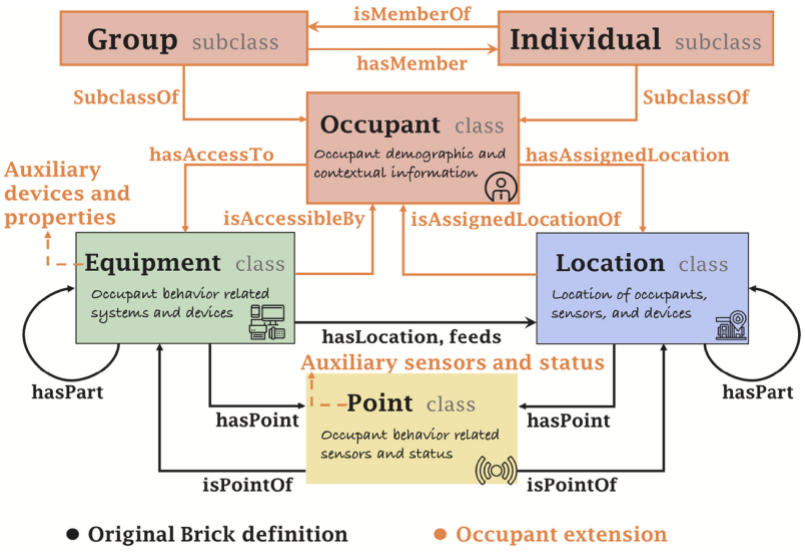
Review and release the expanded schema

In the final year of the project, we reported on several extensions to the Brick schema; all of which can be found on GitHub in the Brick repository (https://github.com/BrickSchema/Brick) unless indicated elsewhere.

Occupancy

One of the most recent set of extensions, the Occupancy modeling extensions may be found here: https://github.com/gtfierro/brick-occupancy-extension. The extension introduces some classes for describing properties of the building occupants.

Figure 2: schematic drawing of occupancy modeled in the Brick schema (Figure from Brick Occupancy Extension paper[5])



The extension introduces several new classes of equipment:

- Ceiling Fan
- Personal Devices:
- Portable Fan
- Portable Heater
- Desktop Light
- Envelope Equipment:
- Door
- Window
- Plug Meter

The extension also incorporates plug meters as a new subclass of brick:Electrical_Meter. The model directly supports modeling the relationships between occupants, the devices they interact with, and the energy consumption of those devices.

Interoperability with other ontologies: REC, Project Haystack and *ref-schema*

As is described further in the Tools and Translators section below, we developed a Brick extension to harmonize the Brick ontology with the RealEstateCore (REC) ontology. This extension (https://github.com/RealEstateCore/REC4/tree/main/Ontology/SHACL/Brick) clarifies the semantic relationships between Brick and REC concepts. It also lays the groundwork for future changes to both BRICK and REC to make them more complementary. This extension marks many months of collaboration with the core REC development team. More recently, the

[5] https://www.sciencedirect.com/science/article/abs/pii/S0926580522001807

development teams for Brick and REC held multiple seminars[6][7] showing how Brick and REC can be used together in the same metadata model. This high level of interoperability was the result of more than a year's worth of development by the Brick and REC development teams. REC 4, the most recent version of the REC ontology, incorporates Brick concepts directly in its definition. Brick 1.3, the most recent version, likewise contains direct support for REC concepts.

We have developed interfaces between Brick and both Project Haystack tags and the *ref-schema* (https://github.com/gtfierro/ref-schema) for interoperability between ontologies and other digital models. The *ref-schema* has been incorporated into the current draft of the ASHRAE 223P standard. The *ref-schema* includes support for relating parts of a linked data model (like a Brick, REC or 223P model) to IFC models, BACnet networks and timeseries databases.

We are continuing to develop other extensions and have community involvement in several areas, such as heat pumps and meters. The Brick v1.3.0 release notes are listed at https://github.com/BrickSchema/Brick/releases/tag/v1.3.0 : this details the set of extensions and changes to Brick. The Brick consortium approval process was utilized for the first time in order for this release to take place.

[6] https://memoori.com/will-a-unified-smart-building-metadata-standard-become-reality/
[7] https://www.realestatecore.io/reccon22/

Task 2: Develop Tools and Translators

The second major task that evolved through the three-year project was developing the tools and translators between Brick and other data models.

Identify list of required interfaces with Brick

In order to consider a wide adoption of Brick, we identified general categories of end users who would benefit from improved data schema translation across data, data models, and tools. Table 4 lists these potential end users, what goals each has, and what application each would use.

Table 4: General categories of Brick applications

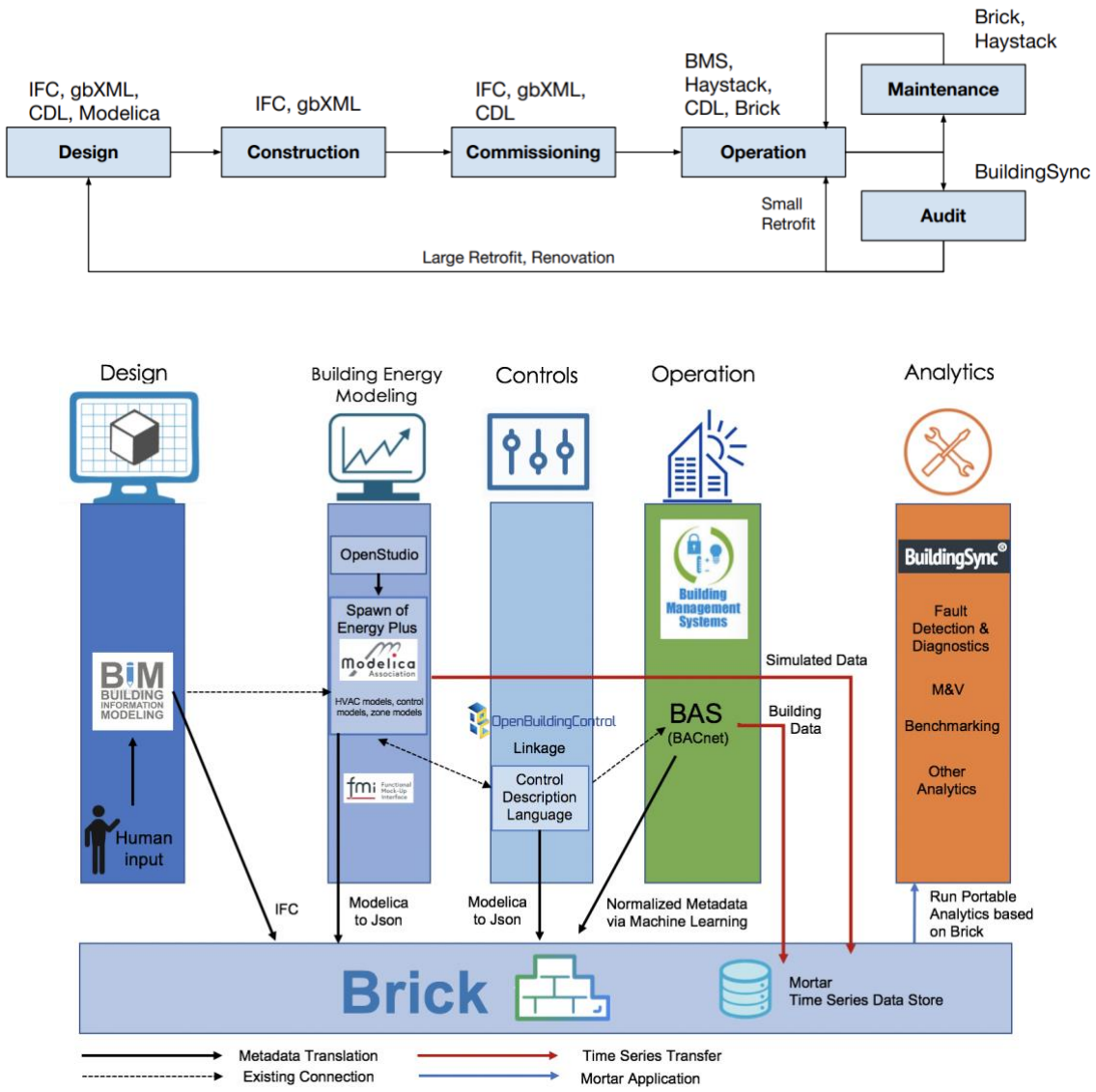
User	*Application*	*Goal*	*Relationship to Brick*	*Applicability*	*Source Data Model:*
Building facilities managers	Operation	Improve building comfort (reduce complaint), or energy performance.	Get data out of existing BAS/BMS into a timeseries database for ease in visualization, analytics, and to aid in performance	High priority to help FM control buildings better	BACnet
Energy services providers	Analytics	Improve building comfort (reduce complaint), or energy performance (energy efficiency, reduce demand charges, reduce energy during peak demand).	Get data out of existing BAS/BMS in large commercial (or from networked thermostats/whole building data in small buildings) in a format that can be used for third party/proprietary tools analysis, FDD, controls (EE, DR)	Lots of existing buildings--high need to improve EE, implement DR, and integrate DER	BACnet (using tools such as "The Building Adaptor" for large commercial buildings)
Energy controls engineers	Controls	Develop controls specifications (Guideline 36) to upgrade existing buildings.	OBC and CDL, ASHRAE 231p, Ctrl-Flow	Upgrading legacy building controls	Open Building Controls (OBC) Control Description Language (CDL)
Energy controls engineers	Controls	Test controls on existing large commercial buildings through models	Spawn of Energy Plus, Modelica	Testing new control sequences	BACnet
Energy controls engineers	Design	Create Brick models from IFC	IFC translator	Relatively new buildings, so fewer in number, but perhaps quicker to	IFC

				make Brick model.	
Energy controls engineers	Modeling	Create Brick models from Modelica/Spawn of Energy Plus/Energy Plus to be able to design and test controls	Spawn of Energy Plus, Modelica	Testing new control sequences	BACnet
Project Haystack users	Controls/ Analytics	Reuse analytics developed using Haystack	Convert Haystack schema to Brick schema or Brick to Haystack	Value for Project Haystack users	Project Haystack
Energy data analyzers (evaluation, program development, verification, benchmarking)	Analytics	Use BuildingSync	Translator to BuildingSync	Value for BuildingSync users	BuildingSync

Outline integration pathway for at least four data models

We identified four translations to focus on: BACnet to Brick, Modelica energy models, and Control Description Language (CDL) control sequences to Brick, IFC to Brick, and Project Haystack to Brick. We also considered BuildingSync to Brick. Figure 3 shows the various silos of building data at various stages of the building lifecycle, from design, modelling, controls, operation, and analytics.

Figure 3: Many different metadata standards and technologies are applied over the course of a building's lifecycle, but are relatively siloed and thus non-interoperable. We propose a method for continuously mining a single, coherent semantic metadata from these various representations over time.



BMS Point labels from a building management system are perhaps the most common source of digitized building metadata, but also the most varied and least standardized. Point labels act as identifiers for sensors, actuators, alarms, parameters and other "registers" in a SCADA or BMS system; as a result, they may contain some semantic information such as the point's location, function and some related equipment. However, they are often unstructured, building-specific, inconsistent, and reliant upon vendor-specific conventions for consistent interpretation. This motivates the creation of tools and techniques which can extract structured and standardized metadata—such as Brick—from these representations.

Although several research efforts exist to automate this process[8] [9], these techniques rely upon heuristics and learned statistical techniques whose efficacy is hard to measure for a more general population of buildings, whose labels may not be as regular. To this end, we have released a dataset of building labels[10] that seeks to make a broader diversity of building labels available for researchers. We are also pursuing the creation of tools that are less automated but offer a structured workflow for non-computer scientists to parse and convert their building label metadata to Brick.

In collaboration with JCI and using data donated by UC Davis, we have been working on an open-source tool which converts specially structured CSV files and spreadsheets to functioning Brick models[11]. Rather than requiring users to be fluent in semantic web technology, these new tools only ask that users can fill out a spreadsheet template. The tool integrates with the open-source OpenRefine tool[12], which presents users with a powerful web interface for cleaning and parsing labels. After parsing labels in OpenRefine, users can export the data to a CSV file which can then be converted into a Brick model. We have documented this workflow in a YouTube video[13]. We have organized an informal working group of individuals from several "smart building" startups, government bodies and other researchers to discuss and evaluate this tool and other metadata conversion techniques.

We have used this technique to extract Brick models for more than 10 different buildings, all with different point label structures, and are continuing to develop the tool.

The recently-released BuildingMOTIF tool[14] from NREL also incorporates the CSV import feature, inspired by the work in this project.

Industry Foundation Classes (IFC) is a standard format and data model for building information modeling, designed for the exchange of data related to the design and construction of a building. The IFC standard describes many common types of HVAC and lighting equipment as well as sensors. However, because of the focus on the design and construction phases of the lifecycle, IFC does not explicitly represent the context or configuration of these entities within the structure of their respective subsystems (for example, is a particular fan a supply fan or an exhaust fan?) This information can be inferred by expert inspection of an IFC model, but cannot be easily determined in a programmatic manner.

---

[8] Jason Koh, Bharathan Balaji, Dhiman Sengupta, Julian McAuley, Rajesh Gupta, and Yuvraj Agarwal. 2018. Scrabble: Transferrable Semi-Automated Semantic Metadata Normalization using Intermediate Representation. In The 5th ACM International Conference on Systems for Built Environments (BuildSys '18), November 7–8, 2018, Shenzen, China. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3276774.3276795

[9] Bhattacharya, Arka A., et al. "Automated metadata construction to support portable building applications." *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. 2015.

[10] Gabe Fierro, Sriharsha Guduguntla, and David E. Culler. 2019. Dataset: An Open Dataset and Collection Tool for BMS Point Labels. In The 2nd Workshop on Data Acquisition To Analysis (DATA'19), November 10, 2019, New York, NY, USA. ACM, New York, NY, USA, 3 pages. https://doi.org/ 10.1145/3359427.3361922

[11] https://github.com/gtfierro/brick-builder

[12] https://openrefine.org/

[13] https://youtu.be/LKcXMvrxXzE

[14] https://github.com/NREL/BuildingMOTIF/

Green Building XML (gbXML) is an XML-based data exchange format for building information modeling, similar to IFC. While it only represents simple building geometry compared to IFC, it can represent a variety of HVAC and Lighting equipment and — importantly — groups related equipment, rooms and zones together. This kind of contextual information permits greater inference of Brick relationships than IFC. Both gbXML and IFC are common export formats for BIM tools such as Autodesk Revit.

BuildingSync is an XML-based schema designed to capture energy audit data in line with ASHRAE Standard 211. The standard requires the reporting of high-level operational parameters of the building (floor area, occupancy classification, operating hours, etc.), primary system information (heating, cooling, lighting, process loads, etc.), as well as historical energy consumption, benchmarking information, and target performance objectives. This information is then used by the energy auditor to provide the building owner with recommendations for achieving energy reduction goals or mandates. Energy audits, such as those enabled by BuildingSync, are often conducted at various points in a building's lifecycle. For example, New York City's Local Law 87 requires an energy audit to be performed every 10 years for existing buildings over 50,000 square feet. This strategy is seen as a key feature in many Building Energy Performance Standards (BEPS) and other mandates issued by cities to reduce the energy footprint of the existing building stock.

Modelica Buildings Library and Control Description Language (Modelica/CDL): Modelica is a declarative, equation-based modeling language used to model engineered systems. In Modelica, components are represented by modular models coupled to each other through connectors to form systems. Connectors can represent input/output ports for control signals, or physical ports such as for representing a flange of a valve through which fluid flows. The Modelica Buildings Library contains component and system models for building and district energy and control systems. The Control Description Language (CDL) is a subset of Modelica used to express control sequences for building automation systems in a vendor-independent format. CDL aims at enabling the digitization of the design, specification, deployment and verification of building control sequences. CDL sequences can be integrated with building models and the Modelica Buildings library already contains CDL representations of high-performance HVAC control sequences (e.g., ASHRAE Guideline 36). Simulations of these models can aid in comparing the performance of different control sequences, testing their correct specification, and commissioning their correct implementation in buildings.

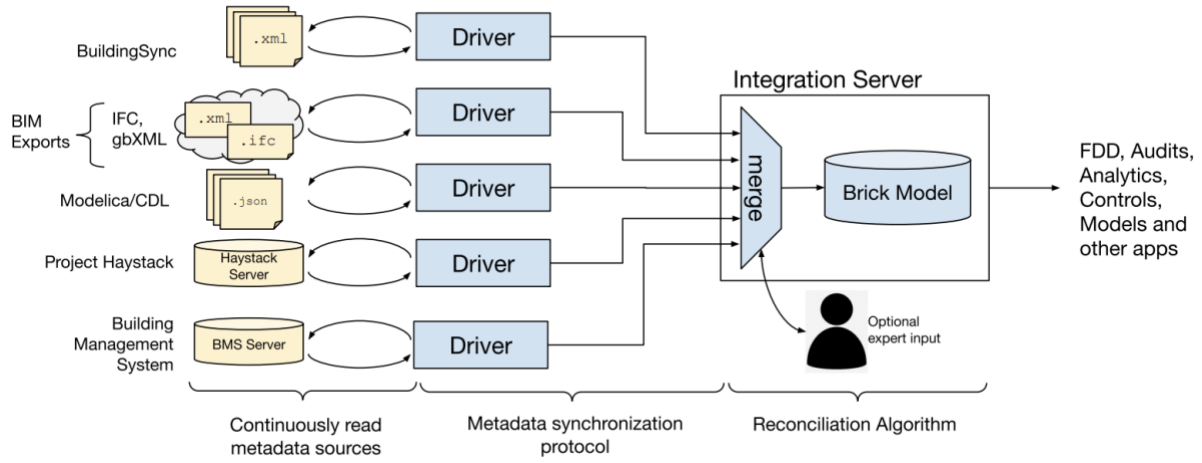
Project Haystack (Haystack) is a popular tag-based data model that describes equipment and points (data sources) in buildings for use during the operational phase of a building. Haystack does not formally define how well-known concepts should be described. As a result, tag usage is inconsistent between Haystack models, which limits the interpretability of the resulting model and the extent to which it can be integrated with other metadata sources. Recent work ameliorates these issues through a method for automatically aligning Haystack tags with existing formal definitions.

Brick is built over the extensible RDF graph-based data model, which can be easily extended to include additional properties and concepts. The RDF data model organizes information into triples: 3-tuples of a subject, predicate and object. The subject and object represent classes (groups of entities with common properties) or entities (instances of classes); the predicate

represents a relationship between the subject and object or a property of the subject whose value is the object.

The proposed approach to continuous metadata integration, represented in Figure 4, is as follows. A **metadata source** is a digital representation of a building whose structure, syntax and semantics are informed by some metadata standard or data model. The representation may change over time, e.g., to add additional detail during commissioning or to reflect the impact of repairs and retrofits. A **driver** is a software process that produces Brick metadata from an underlying metadata source. For structured or standardized metadata sources such as BuildingSync and IFC models, this may be accomplished through direct translation of the source's concepts and structures to Brick. For less structured and ad-hoc sources such as Haystack models and BMS labels, a driver may infer Brick metadata through a statistical or heuristic-based approach. The driver continually produces Brick metadata from the most recent version or "snapshot" of the underlying metadata source, even as that source changes. The driver places no requirements on how the Brick metadata is produced or inferred, but represents the Brick metadata as a set of RDF triples. An integration server is a logically centralized process that consumes Brick metadata from a collection of drivers and produces a unified Brick model representing the union of the collected metadata. Because different metadata sources are traditionally created at different times and by independent stakeholders, the unified metadata is likely to contain mistakes, disagreements and inconsistencies. To address this issue, the integration server incorporates a novel reconciliation algorithm (analogous to the "merge" operation in Git) that attempts to resolve the differences between the metadata reported by the drivers. The integration server maintains a Brick model which is accessible by applications and other external services such as Mortar.

Figure 4: Overview of the proposed approach: drivers interface directly with existing metadata sources stored in local file systems, or accessed via file shares or networked services. Drivers continuously publish inferred Brick metadata to a central server, which produces a unified model.



Drivers are software processes that produce Brick metadata from an underlying metadata source. Drivers report the Brick metadata to the integration server via a metadata synchronization protocol, described below. The metadata synchronization protocol decouples the method of inferring or producing Brick metadata from how that metadata becomes integrated into the

authoritative model. This allows the proposed system to incorporate new metadata sources and novel methods of Brick metadata inference.

Figure 5 describes at a high level the operation of a driver and its interaction with the integration server per the metadata synchronization protocol. The protocol operates over HTTP. The driver POSTs a list of JSON messages to the integration server when the metadata source changes. Each message contains the Brick metadata inferred for an entity (virtual, logical or physical "thing") along with the following fields:

- **id:** a name or other identifier for the entity, as given by the metadata source
- **raw:** identifies the encoding (e.g. JSON, XML) and content of the original metadata that defined this entity. May contain additional metadata not expressed in Brick
- **source:** identifies the metadata source
- **timestamp:** denotes the time at which the metadata source was read, corresponding to a consistent "version" of the underlying metadata
- **triples:** a list of RDF triples constituting the Brick metadata produced from the raw record

Figure 5: Example record published by the BuildingSync driver, showing the original metadata (raw) and the inferred Brick metadata (triples).

```
{
 "id": "RTU-1",
 "raw": {
   "content": "<auc:Delivery ID=\"RTU-1\">
    <auc:DeliveryType>
      <auc:CentralAirDistribution>
        <auc:AirDeliveryType>Central fan</auc:AirDeliveryType>
        <auc:FanBased>
          <auc:CoolingSupplyAirTemperature>73</auc:CoolingSupplyAirTemperature
        </auc:FanBased>
      </auc:CentralAirDistribution>
    </auc:DeliveryType>",
    "encoding": "XML"},
  "source": "BuildingSyncDriver",
  "timestamp": "2020-07-16T20:02:50",
  "triples": [["http://example.com/building#RTU-1",
               "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
               "https://brickschema.org/schema/1.1/Brick#Rooftop_Unit"]]
}
```

The server saves all received messages in a local database, indexed by the timestamp field. When the server performs the reconciliation algorithm to produce a unified metadata model, it by default only considers the messages corresponding to the most recent timestamp per driver. By extension, the server can also produce a unified metadata model for any point in the Brick model's history, provided a driver was publishing metadata at that time. This allows applications to access the history of changes in a building, but through the interface of a standardized, unified representation rather than an ad-hoc collection of diverse metadata sources.

The triples field can contain arbitrary Brick metadata to be relayed to the server. Typically, this involves type information (*vav1 is a brick:VAV*), system composition information *(vav1 is downstream of ahu1),* telemetry association *(vav1 has temperature setpoint temp_sp1)* and

location information *(tstat1 is in Room 410)*. The triples may also define extensions to the Brick ontology, such as to describe additional properties of an unusual device or point.

Drivers / Metadata Source

We implement the following drivers for integrating several common building metadata standards and data models into a Brick model. For each driver, we explain the structural and semantic mapping between the metadata source and Brick. We only focus on a particular version of each metadata source below. The architecture permits the development of separate drivers for different versions of each metadata source.

BuildingSync: Buildings represented via the BuildingSync schema are captured in an XML document; a single document may contain multiple buildings. Elements in a BuildingSync document describe systems (e.g., the kinds of heating and cooling subsystems in an HVAC system), properties of systems (e.g., expected energy consumption, cooling or heating capacity) and properties of buildings (e.g., year of construction, floor area). The children of a given element in the document describe both the properties of that element as well as relationships to other elements. The driver uses the following approach to map BuildingSync elements to Brick entities: First, the driver uses XPath expressions to find elements with certain attributes or properties that match a Brick class definition. Then, this type is refined by searching for additional properties that may further elucidate the type of the entity described by that element. For example, the BuildingSync auc:Chiller element aligns with the Brick brick:Chiller entity type. If the auc:Chiller element in the BuildingSync document contains a auc:ChillerType property with the value “Absorption”, then the driver can infer the more specific Brick class of brick:Absorption_Chiller.

The initial type mapping exercise identified 27 direct concept mappings, primarily for locations and equipment types. BuildingSync captures limited information pertinent to point information (in comparison to Haystack and CDL / Modelica), although certain limits and set points may be captured.

Project Haystack: Popular Haystack implementations can export a Haystack model as a collection of documents. Each document contains the tags and key-value pairs that describe each Haystack entity; Haystack entities represent sites, points and pieces of equipment. As the Haystack model is updated and maintained, the driver can request additional exports in order to update the produced Brick metadata.

The structure of a Haystack model has a straightforward mapping to Brick: each Haystack entity corresponds to one or more Brick entities, and the generic links between Haystack entities (called refs in Haystack parlance) correspond roughly to Brick relationships. However, because the semantics of a Haystack model are not well-defined, the precise types of Haystack entities and relationships between them must be inferred. The Haystack driver adopts the inference engine described in (Fierro et al, 2019) to produce a Brick model from an exported Haystack model.

The driver operates as follows: the driver loads the most recent exported Haystack model into memory and partitions it by the documents corresponding to each Haystack entity. For each entity, the driver uses the inference engine from (Fierro et al, 2019) to produce a set of RDF triples representing the Brick metadata for that entity. Each original Haystack document and the

inferred triples are packaged into the standard message schema, using the value of the id field from the Haystack document as the name of the entity. The set of these messages is then transmitted to the integration server.

Modelica/CDL: Modelica and CDL models consist of a set of connected objects; these can be exported to a JSON document using the modelica-json tool. The JSON export contains detailed information about building components, their connections and the control sequences. Control sequences in CDL and Modelica models of the HVAC system can also embed Brick information (and other semantic information) as annotations, which are also included in the JSON export when they are made available in the source model. The driver operates as follows. From a JSON export, the driver first extracts all the Brick semantic information from the annotations, if present. The next is an inference step, where the translator uses the Modelica class of each instantiated object and assigns a Brick type if such a mapping exists. This treats each Modelica object as a Brick entity: for example, every instance of the Modelica class Buildings.Fluid.Sensor.Temperature can be translated into a Brick:Temperature_Sensor entity. Then, the driver traverses the connect statements in the Modelica model to extract the sequential (brick:feeds) and compositional (brick:hasPart) relationships between objects and add these to the corresponding Brick entities. connect statements in the Modelica model can also provide spatial metadata such as rooms and thermal zones. Lastly, the driver uses the CDL representation of control sequences to extract the inputs and the outputs of controllers in order to identify brick:hasPoint relationships between equipment, actuators and measurement sensors. Although there is a good deal of semantic metadata that can be obtained from Modelica/CDL models, because the models target simulation, they also contain information that cannot be directly represented in a Brick model. This includes properties such as heat transfer parameters for a cooling coil.

Industry Foundation Classes. For IFC, the data format description includes the EXPRESS file format and requires a special parser. Colleagues at BuildingSmart have an IfcOWL project with EXPRESS to RDF converter2.[15] We may also look at OpenStudio. The mapping consists of a few steps: instantiate location and zone entities from IFC, instantiate points from IFC, and record each IFC document.

After the release of IFC4, Dr. Fierro and JCI worked with the developers of BlenderBIM (an open-source IFC editor and visualizer) to incorporate support for Brick in a release of the BlenderBIM plugin[16]. This included support for (a) linking a Brick model to an IFC model (and vice versa), and (b) automatically inferring Brick metadata for IFC entities (and vice versa).

Metadata Reconciliation

Because the set of metadata sources for a building are developed independently over time, they may present incomplete or incompatible perspectives on the building, its subsystems and data sources. We propose a reconciliation algorithm for merging the Brick metadata proposed by a set

[15] https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/

[16] https://twitter.com/BlenderBIM/status/1461098241970384898

of drivers into a single logically and semantically valid model. This unified model can be used to configure portable applications. The algorithm is executed at the integration server.

First Stage: Record Linkage. The first stage of the algorithm performs two kinds of record linkage on the associated names or labels of each entity. The record linkage step uses traditional string matching techniques to produce clusters of entities based on the names of those entities. The name of an entity can be derived from string-valued properties such as rdfs:label or the URI of the entity if no string-valued properties are found. The goal of this step is to use the semantic information sometimes encoded in entity labels as one heuristic for linking. Due to different naming conventions between metadata sources, there can often be greater similarity scores between entities from the same source than between entities of different sources. For this reason, the algorithm assumes that all entities reported by a metadata source are distinct and only clusters entities from different metadata sources.

The second record linkage step leverages semantic information from the proposed types of each entity to do type-aware clustering. The algorithm identifies all entities that have a proposed type that is a Brick class (entities with an rdf:type property with a value that is a class from the Brick ontology) and associates with the entity all of the Brick classes which are equal to or are superclasses of the given type. If two or more sources have the same number k of entities of a given type, the algorithm produces k clusters containing one entity from each source that have the highest pairwise similarity between their names. The clusters produced by this second step are added to the set of clusters produced by the first step.

Second Stage: Graph Union. The second stage of the algorithm takes as input the clusters of entities from the first stage and builds and validates the graphs formed by merging their associated triples. The algorithm also adds statements to the Brick model to associate the different identifiers for the same entity (this uses the owl:sameAs property).

Unlike many other metadata sources, Brick is built over formal logic. This allows continuous validation of a Brick model as metadata is added to it, which helps the algorithm produce a logically valid model through the reconciliation process. The logical validation is implemented by a process called an OWL-RL reasoner; this process also generates logical consequences of the statements in a Brick graph. The reasoner examines the graph G for each cluster and produces a set of exceptions. These exceptions indicate that either the entities in the cluster are not equivalent, or the metadata associated with those entities is incorrect. Examples of exceptions include:

- incompatible entity types, e.g., if a cluster contains entities of two types which are disjoint
- incompatible relationships, e.g., if the values of an entity's properties and relationships do not match the definition of those properties and relationships
- semantic "sniff tests": qualities of the Brick graph that are not logical violations but may indicate deeper issues. The primary example of this is an entity's types should all be subclasses or superclasses of each other.

When exceptions occur, the algorithm can optionally re-cluster entities using more selective thresholds, or, as in the implemented prototype, it can request human input on the failing cluster. The algorithm then repeats the graph union step. These steps are iterated until no exceptions are

logged, after which all of the clusterproduced graphs are merged into a single graph. The algorithm performs one last verification pass on this unified graph; if this passes, the unified graph is returned as the authoritative metadata model.

Human-aided Disambiguation. When the algorithm logs exceptions for the entities in a given "bad" cluster, the algorithm can ask for external input on how to proceed. First, the algorithm asks if it should split the bad cluster into two or more smaller clusters; this can be performed automatically by adjusting the clustering hyperparameters or manually by specifying the new clusters explicitly. If reclustering occurs, then the algorithm begins another iteration of the graph union phase above using the new clusters.

If reclustering does not occur, then the algorithm asks for manual resolution of the graph contents before proceding to the next cluster. This typically involves choosing which Brick class to assign to a group of entities, but may also require editing properties and relationships of entities.

We illustrate the behavior of the algorithm with the following example. Consider two metadata sources $shaystack$ and $sbsync$ which correspond to a Project Haystack and BuildingSync representation of a building, respectively. The drivers for these sources produce the Brick metadata listed in Figure 6. The algorithm begins by clustering the entities. The first string matching phase places bldg:bsync-meter and bldg:ph-meter into the same cluster because their labels are sufficiently similar ("main-meter" vs "Main Meter"). The labels bldg:bsync-ahu-1 and bldg:ph-rtu-1 are not grouped because the labels are too dissimilar.

Figure 6: Example Brick metadata produced by BuildingSync and Project Haystack drivers. The rdfs:label property denotes the original name or identifier of the entity in the metadata source.

```
# BuildingSync: ahu-1
bldg:bsync-ahu-1    rdf:type   brick:Air_Handler_Unit ;
    rdfs:label "AHU-1" .
# BuildingSync: main-meter
bldg:bsync-meter    rdf:type   brick:Building_Power_Meter ;
    rdfs:label "main-meter" .

# Haystack: rtu-1
bldg:ph-rtu-1    rdf:type   brick:Rooftop_Unit ;
    rdfs:label "RTU 1" ;
    brick:hasPoint   bldg:oat-1 .
# Haystack: main-meter
bldg:ph-meter    rdf:type   brick:Power_Meter ;
    rdfs:label "Main Meter" .
```

The second type-aware phase examines the Brick-defined classes for the remaining entities. Using the Brick ontology, the algorithm infers that because brick:Air_Handler_Unit is a superclass of brick:Rooftop_Unit, each source has metadata for one air handler unit. Because each source has the same number of instances of that type, the algorithm clusters those entities by label similarity. This results in bldg:bsync-ahu-1 and bldg:ph-rtu-1 being placed in the same cluster. The difference in specificity between the original sources is due to the fact that BuildingSync does not differentiate between subclasses of air handler units, but Haystack does.

The algorithm proceeds by unifying the triples for the entities in each cluster and validates the logical and semantic soundness of the resulting graph. In this simple example, the algorithm only needs to verify that the types of each pair of entities are compatible. This is true:

brick:Air_Handler_Unit is a superclass of brick:Rooftop_Unit and brick:Power_Meter is a superclass of brick:Building_Power_Meter. Finally, the two graphs are merged into a single Brick model (Figure 7).

Figure 7: The inferred unified metadata model for the triples in Figure 5. Note that the more specific type for each entity was chosen and that extra properties from the original source are carried through.

```
bldg:rtu-1   rdf:type    brick:Rooftop_Unit ;
    brick:hasPoint bldg:oat-1 .
bldg:meter   rdf:type    brick:Building_Power_Meter .
```

Evaluation

We evaluate the proposed approach by implementing a fully functional prototype and measuring aspects of its execution on a set of real and artificial sites. As structured, digital representations of buildings become more standardized and widely available, we envision this framework to serve a vital role in integrating this information over time. The prototype is open-source and is available online under a permissive license at https://github.com/gtfierro/shepherding-metadata.

Models and Sites

Table 5 lists the sites with more than one available metadata source that were used through the evaluation. The Carytown site is drawn from the set of example Project Haystack sites. We developed a BuildingSync model for the Carytown based on available metadata. The "DOE Medium Office" is the reference building for a new construction, medium office in Chicago, United States that has been described in the set of U.S. Department of Energy Commercial Buildings Benchmark. We used the Modelica model that had been developed for a single floor (four perimeter zones and one core zone) of this building as part of the Modelica Buildings library and developed Project Haystack and BuildingSync representations for this building.

Table 5: Sites and Metadata sources for evaluation

Site Name	Metadata Sources
Carytown	BuildingSync, Project Haystack
DOE Medium Office	Modelica, BuildingSync, Project Haystack

In order to understand the behavior and performance of the drivers, we assembled a set of publicly available models for each of the targeted metadata sources. In total, we measured 9 Haystack models, 18 BuildingSync models, and 16 gbXML models. These models are not for the same set of buildings. Instead, the population of buildings for each metadata source offers an empirical measurement of Brick metadata availability.

Driver Implementation

We developed three drivers for extracting Brick metadata from existing metadata sources. The prototyped drivers are all implemented in Python 3 and are built over a simple framework that provides:

- an API for interacting with the metadata synchronization protocol

- an embedded web server for viewing and extracted records and inferred Brick metadata from an executing driver
- tools for detecting and reacting to changes in metadata sources.

This set of features reduces the developer overhead of producing a driver by abstracting away common elements of the protocol and implementation. The embedded web server presents a simple read-only JSON API that permits a user or automated tool to debug the driver's output without the use of an external server. Although presented prototype is implemented in Python, its implementation is short (~100 LOC) and uses modules from the standard library, so there are few technical barriers to implementing its functionality in other languages.

We now review the implementation of each of the three drivers to understand how well the metadata synchronization protocol fulfills the inference needs of existing metadata sources.

Project Haystack Driver: The Haystack driver is built over the inference engine described in (Fierro et al, 2019) and available as part of the opensource brickschema Python package. The driver only required a few lines of code to read a JSON export of a Haystack model, feed this to the inference engine, and extract the inferred Brick metadata. The division of a Haystack model into a set of entities is natural: each Haystack document becomes one or more Brick entities, with relationships between them. Due to the high degree of overlap in the modeling domain of Haystack and Brick models, most of the Haystack metadata is translated into its Brick equivalent. One exception is the timeseries information embedded in Haystack models (such as the current value and timestamp of a point), which has no direct representation in Brick.

BuildingSync Driver: The BuildingSync driver operates by using XPath expressions to conditionally extract parts of a BuildingSync XML document and translate the information to Brick metadata. The driver accepts a list of BuildingSync-to-Brick mappings in tabular form, making the driver easy to extend with additional mappings. However, the amount of Brick metadata obtained from a BuildingSync model is limited compared to what can be inferred from a Modelica or Project Haystack model. This is due to a difference in scope: BuildingSync describes properties and performance characteristics of building systems, rather than the individual components and relationships addressed by other metadata sources. As a result, a BuildingSync model may be a better target for exporting data from a unified Brick model.

gbXML and IFC: Both gbXML and IFC perform data exchange during the design, construction and commissioning phases of a building's lifecycle. However, it was much more straightforward to extract Brick metadata from gbXML models than IFC models; this is supported by the higher degree of Brick metadata extracted from gbXML models as seen in Figure 8. (Figure 8 shows the distribution of the number of Brick triples that each driver produced per entity across all of the buildings). This is for two reasons. Firstly, although newer versions of IFC (e.g., IFC 4.1) can provide more equipment and sensor information which is relevant to Brick, we only had access to older models (targeting IFC 2x3) which do not have as broad a vocabulary. Secondly, the flexibility and generic approach of the IFC data model results in a complex schema in which related pieces of information are often separated by many intermediate objects. The relative simplicity of gbXML resulted in a more complete metadata driver.

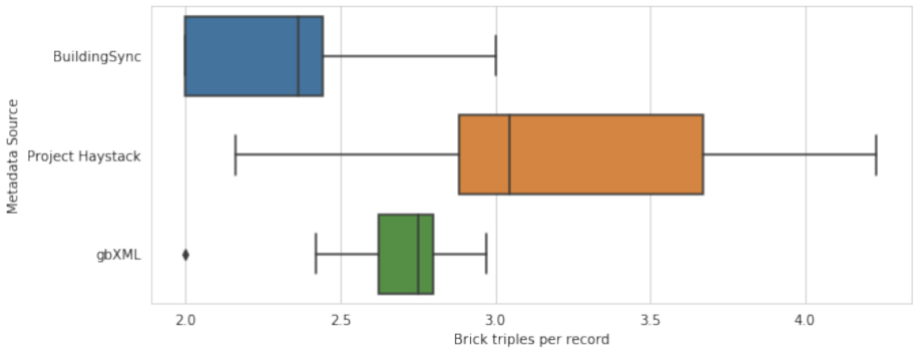
Modelica/CDL: The Modelica/CDL driver initially attempts to assign Brick types to objects in the Modelica model based on its the Modelica class. Next, Brick relationships (brick:feeds,

brick:hasPart, brick:hasPoint) are extracted by traversing the different connect statements in the model and from the inputs and outputs of CDL blocks. However, as with any programming language, Modelica allows the developers to create custom components and models that do not use the Buildings library templates and conventions. In such cases, the type mappings mentioned earlier might fail and the driver has to rely on heuristics on the element names, comments and Brick-specific annotations included in the models to extract information.

For instance, Modelica declarations are of the form “className instanceName comment;”. Hence from the declaration Buildings. Fluid.Actuators.Dampers.Exponential damRet(...) "Return damper"; the driver can extract the brick type (brick:Damper) from the Modelica class name, and it can infer the location (i.e., the return section of the AHU), from the Ret string in the instance name damRet or from the comment, allowing to refine the Brick type to brick:Return_Damper.

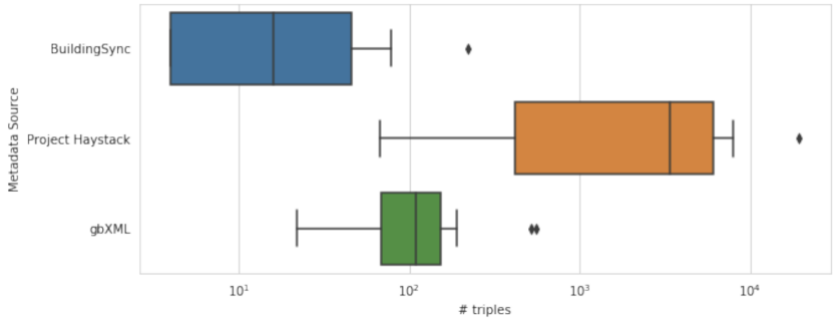
Although Modelica/CDL models can provide a significant amount of semantic metadata, there are still gaps that have to be addressed (modeling fire equipment, thermostat equipment etc. in Modelica, representing CDL control specifications in Brick etc.). We are working with the developers of the Modelica Buildings library to continue the development of this driver to address these gaps and to possibly encode the Brick mappings directly in the library. A key update is the explicit inclusion of semantic information as annotations so that the Modelica/CDL translator can extract more granular information (e.g.: Brick:Return_Air_Temperature_Sensor instead of a simple Brick:Temperature_Sensor). The syntax of these annotations is undergoing review as part of ASHRAE 231p committee. Once finalized, it can support providing more information about the points and equipment present in the energy models and control sequences.

Figure 8: The distribution of the number of triples inferred per record for each driver.



To compare the behavior of each of the drivers, we executed each driver on a collection of publicly available metadata sources and measured the number of records and triples each produced. Recall that a record is a Brick entity represented in a metadata source, and that triples each contain two entities and a relationship between them. Figure 9 shows the distribution of the total number of triples obtained for each model.

Figure 9: The distribution of the total number of triples inferred each driver. Note the log-scale on the X axis



Evaluation of Reconciliation Algorithm.

To develop an understanding of how the algorithm behaves, we collected sites with more than one metadata source and executed the system end-to-end to produce a unified metadata model for each site. The list of sites is contained in Table 6.

Table 6 contains the results of executing the reconciliation algorithm on the metadata from each site. The Union column contains the proportion of triples in the unified model that were contributed by each source; this includes redundant triples. The Unique column contains the proportion of triples in the unified model that came only from that source. The Common column contains the proportion of triples that came from all of the available metadata sources. Between these quantities, we can understand the diversity of the metadata from each of these sources and how complementary they are to one another.

Table 6: Results of executing the reconciliation algorithm on the metadata from each site.

Site Name	Metadata Source	Union	Unique	Common	Model Size (Triples)
Carytown	Haystack BuildingSync	15.3% 4.7%	100% 100%	0	280
DOE Medium Office	Haystack Modelica BuildingSync	16% 41.9% .8%	100% 100% 100%	.8%	1,698

Although there are only a few sites and models, we can observe some general behavior about the metadata extracted from the available drivers. First, the metadata from Haystack and BuildingSync drivers are mostly complementary and there is little overlap between them. This aligns with the respective scopes of each metadata source: BuildingSync describes holistic properties of systems that may not be covered by Project Haystack models (at least in a standard way). Secondly, Modelica drivers provide more Brick metadata than Haystack drivers. This also aligns with the detailed treatment of HVAC systems found in Modelica models compared with the coarse-grained modeling found in Haystack.

For all sites, the metadata common to all drivers was very low. This is to some extent due to the completeness of the drivers at time of writing, but is also limited different levels of detail and different perspectives of a building that are communicated by different metadata sources. The metadata from each driver was also completely unique. Even there is some overlap in the entities described by each driver, no two drivers actually produced Brick metadata at the same level of

detail or level of completeness. For example, one sensor was identified as a brick:Flow_Sensor by the Modelica driver and a brick:Return_Air_Flow_Sensor by the Haystack driver.

Demonstration of three interfaces

In the third year, we reported on the demonstration of at least two of these interfaces: Modelica-Brick, BACnet-Brick, and IFC-Brick.

Demonstration of Modelica-Brick Translator

The Modelica-Brick translator takes as input a JSON export of a Modelica/CDL model (the JSON export has been generated using the modelica-json tool). The translator treats each instance of a Modelica model in the document as a Brick entity, and assigns a Brick class to entities whose class is defined in the Modelica Buildings Library. To infer relationships between these entities, the translator examines the ports for each Modelica instance; these are connected by connect statements to other instances of Modelica models. From these statements, it can infer the Brick relationships brick:feeds, brick:hasPart and brick:hasPoint.

We demonstrate the operation of the Modelica-Brick translator using the publicly available Modelica instance of the DOE Medium Office. It is a reference building for a new construction, medium office in a large U.S. city that has been described in the set of U.S. Department of Energy Commercial Buildings Benchmark. We used the Modelica model that had been developed for a single floor (four perimeter zones and one core zone) of this building as part of the Modelica Buildings library. The translator we developed has been published at: https://github.com/gtfierro/shepherding-metadata/.

We developed a web server that allows us to interact with the output of the translator. In this example, we hosted the server at http://localhost:8081. Using this application, Figure 10 shows the list of elements extracted from the Modelica model (using the endpoint http://localhost:8081/ids). Now, Figure 11 shows in detail the relevant Brick triples for a particular element, *'heaCoi'*, which is the heating coil of the air handling unit (http://localhost:8081/id/heaCoi). Now the actual representation of the *heaCoi* in the model and the graphical representation of the Brick triples are shown in Figure 12 and 13. More details about the software, the methodology and the demonstration can be found in the papers published.

Figure 10: List of elements extracted from Modelica


```
[
  "cor",
  "sou",
  "eas",
  "nor",
  "wes",
  "flo",
  "eco",
  "heaCoi",
  "cooCoi",
  "fanSup",
  "senSupFlo",
  "senRetFlo",
  "TSup",
  "dpDisSupFan",
  "TRet",
  "TMix",
  "VOut1",
  "TSupCor",
  "TSupSou",
  "TSupEas",
  "TSupNor",
  "TSupWes",
  "VSupCor_flow",
  "VSupSou_flow",
  "VSupEas_flow",
  "VSupNor_flow",
  "VSupWes_flow",
  "senRelPre",
  "vav",
  "terHea",
  "damOut",
  "damExh",
  "damRet",
  "cor.vav",
  "cor.terHea",
  "sou.vav",
  "sou.terHea",
  "eas.vav",
  "eas.terHea",
  "nor.vav",
  "nor.terHea",
  "wes.vav",
  "wes.terHea",
  "flo.sou",
  "flo.eas",
  "flo.nor",
  "flo.wes",
  "flo.cor",
  "flo.senRelPre",
  "eco.damOut",
  "eco.damExh",
  "eco.damRet"
]
```

Figure 11: The list of triples relevant to the heaCoi element in the Modelica model


```
{
  "id": "heaCoi",
  "record": {
    "content": "{\"content\": \"N/A\"}",
    "encoding": "JSON"
  },
  "source": "ModelicaJSONDriver",
  "timestamp": "2022-04-25T11:54:17",
  "triples": [
    [
      "http://example.com/doe-building#heaCoi",
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
      "https://brickschema.org/schema/Brick#Heating_Coil"
    ],
    [
      "http://example.com/doe-building#heaCoi",
      "http://www.w3.org/2000/01/rdf-schema#label",
      "heaCoi"
    ],
    [
      "http://example.com/doe-building#heaCoi",
      "https://brickschema.org/schema/Brick#feeds",
      "http://example.com/doe-building#cooCoi"
    ],
    [
      "http://example.com/doe-building#heaCoi",
      "http://www.w3.org/2000/01/rdf-schema#label",
      "heaCoi"
    ]
  ]
}
```


Figure 12: A partial view of an air handler unit in the Modelica model, focusing on the heaCoi element

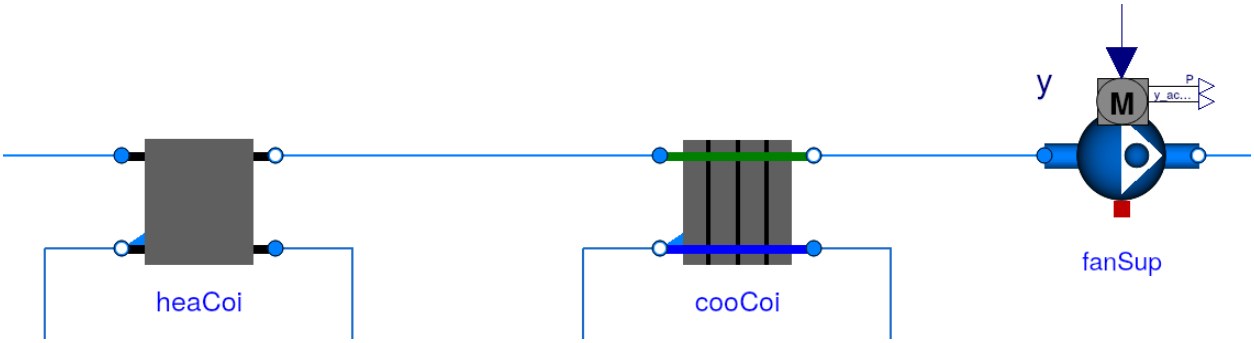
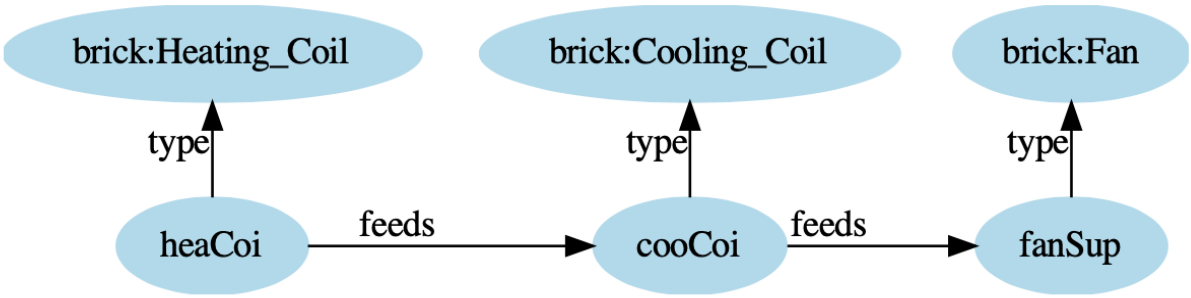


Figure 13: Corresponding Brick instance of the components shown in Figure 12



Demonstration of BACnet-Brick Interface

We have developed an extension to the open-source py-brickschema library[17] which uses the BAC0 library to discover BACnet networks, scan for BACnet objects, and produce a basic Brick model which contains relevant metadata about each of the BACnet objects. Figure 14 contains an example of what this basic Brick model may contain.

Figure 14: Snippet of a discovered BACnet network expressed as a Brick model

```
<urn:bacnet-scan/dev123> a bacnet:BACnetDevice ;
    bacnet:device-instance 123 ;
    bacnet:hasAddress "101:3" .

<urn:bacnet-scan/dev123/1> a brick:Point ;
    ref:hasExternalReference [ a ref:BACnetReference ;
            bacnet:object-description "FANA&D" ;
            bacnet:object-identifier 1 ;
            bacnet:object-name "FANA&D" ;
            bacnet:object-type "binaryOutput" ;
            bacnet:objectOf <urn:bacnet-scan/dev123> ;
            bacnet:units "('Off', 'On')" ],
        [ a ref:BACnetReference ;
            bacnet:object-description "INLET" ;
            bacnet:object-identifier 1 ;
            bacnet:object-name "INLET" ;
            bacnet:object-type "analogInput" ;
            bacnet:objectOf <urn:bacnet-scan/dev123> ;
            bacnet:units "degreesFahrenheit" ] .
```

The produced Brick model uses the recently-developed *ref_schema*[18] to capture the relationship between instances of the Brick I/O Point class (including sensors, alarms, setpoints, etc) and the BAcnet objects they represent. This initial model does not attempt to classify each of the discovered BACnet objects, but does contain all of the relevant metadata for each BACnet

[17] https://github.com/BrickSchema/py-brickschema/
[18] https://github.com/gtfierro/ref-schema

object. This can be accessed by downstream tools to classify objects as Brick points or extract other useful information such as system composition and topology.

Embedding the object identifier and device address inside the Brick model makes it possible for different tools to use BACnet and Brick together. For example, a tool could automatically determine which BACnet points to trend by querying a Brick model. Another tool could retrieve the current value of a Brick point by accessing the "present-value" property on the linked BACnet object when necessary.

To facilitate inferring the Brick types for each of the discovered points, we have created a web-based tool for inferring Brick point types from the labels gathered from BMS systems. The tool is available online and is open-source[19]. It implements the W3C Reconciliation API, which works with other data cleaning tools like OpenRefine[20]. We have also produced a YouTube video showing how OpenRefine works with our web tool to produce a Brick model[21]. We anticipate that the output of the BACnet-Brick interface could work with other tools and techniques such as The Building Adapter.

Demonstration of IFC-Brick Interface

We have also produced initial work on interfacing between the Industry Foundation Classes (IFC) BIM standard and Brick. This consists of two components.

The first is a mapping between IFC concepts and Brick classes that facilitates determining the corresponding "types" of an entity in both standards. This is reinforced by the development of "foreign keys" between an IFC and Brick model using the "ref" schema --- any Brick entity can point to its corresponding representation in an IFC model. The upshot is Brick does not have to duplicate any of the semantic information which is best captured by an IFC model, such as geometry. Instead, consumers of a Brick model can access an IFC model in a standard way to retrieve the information they require.

The second component is an extension of the open-source BlenderBIM plugin[22] for Blender which enables the co-authoring of a Brick model while an IFC model is created. A simple drop-down suggests the most appropriate Brick type for each IFC entity and will maintain the Brick model for the user. The plugin also leverages IFC's native "library" features to populate the bidirectional link from entities in an IFC model to their corresponding representations in a Brick model. In addition, the plugin leverages traversal algorithms over the IFC model to populate some of the relationships between Brick entities.

Demonstration of Haystack-Brick translation

We have developed and demonstrated a translation from Project Haystack to Brick, and released it in Brick 1.1. Table 7 describes the results from inferring Brick entities from five tagged Haystack entities.

[19] https://github.com/BrickSchema/reconciliation-api
[20] https://openrefine.org/
[21] https://www.youtube.com/watch?v=LKcXMvrxXzE
[22] https://blenderbim.org/

Table 7: Results from Haystack to Brick Translation

Site Name	Haystack Entities	Inferred Brick Entities	% Classified Entities	Unclassified Entities	Avg % Custom Tags per Entity	Unique Custom Tags
1	22	23	86.4%	3	7.4%	4
2	147	168	89.8%	15	5.0%	6
3	149	145	73.8%	39	6.6%	7
4	2183	1755	86.7%	290	17.6%	46
5	6474	6236	93.0%	451	19.5%	41

Task 3: Develop an Open Source Analytics Testbed (Mortar)

The third task of the project was to develop a testbed filled with data and Brick models from many existing buildings in order to test analytic algorithms across these data.

General architecture of platform

We have designed and implemented an open-source analytics platform named Mortar, which integrates expressive semantic metadata expressed in Brick with high-performance timeseries storage and retrieval. Through the Mortar API, applications describe their data requirements using queries against Brick models. Brick queries allow applications to describe the context of the requested data rather than the names of specific data sources which. Thus, Brick queries can be re-used across buildings without needing to be rewritten. This reduces the effort in running a Mortar-based analytics application across multiple buildings.

Figure 15 below contains a sample interaction with the Mortar API from the Python programming language. The program describes and retrieves a dataset containing a year of data for air flow sensors and setpoints as well as their associated equipment (such as a VAV) and HVAC zone.

Figure 15: Sample Mortar interaction

```python
import pymortar

client = pymortar.Client()

request = pymortar.FetchRequest(
    # sites=[ <some list of sites here> ]
    views=[
        pymortar.View(
            name="airflow_points",
            definition="""SELECT ?sensor ?setpoint ?equip WHERE {
                ?sensor rdf:type/rdfs:subClassOf* brick:Air_Flow_Sensor .
                ?setpoint rdf:type/rdfs:subClassOf* brick:Air_Flow_Setpoint .
                ?sensor bf:isPointOf ?equip .
                ?setpoint bf:isPointOf ?equip
            };
            """
        )
    ],
    dataFrames=[
        pymortar.DataFrame(
            name="sensors",
            aggregation=pymortar.MEAN,
            window="30m",
            timeseries=[
                pymortar.Timeseries(
                    view="airflow_points",
                    dataVars=["?sensor"],
                )
            ]
        ),
        pymortar.DataFrame(
            name="setpoints",
            aggregation=pymortar.MAX,
            window="30m",
            timeseries=[
                pymortar.Timeseries(
                    view="airflow_points",
                    dataVars=["?setpoint"],
                )
            ]
        ),
    ],
    time=pymortar.TimeParams(
        start="2018-01-01T00:00:00Z",
        end="2019-01-01T00:00:00Z",
    )
)
response = client.fetch(request)
# => <pymortar.result.Result: views:1 dataframes:2 timeseries:835 vals:14628365>
```

Figure 16: Mortar architecture

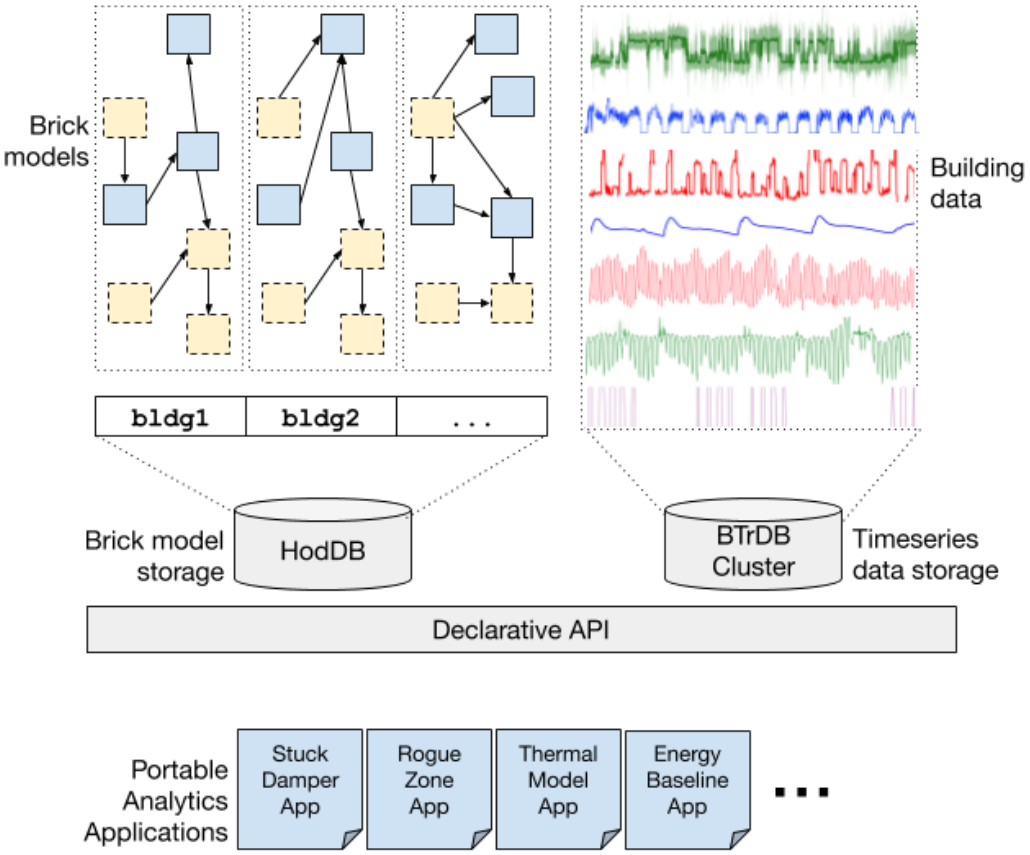


Figure 16 describes the architecture of the Mortar platform. Mortar integrates with HodDB (https://github.com/gtfierro/hoddb), a compact database for Brick models, to provide storage of Brick models and to serve queries against them. Mortar also integrates with BTrDB (https://github.com/BTrDB/btrdb-server), a high-performance timeseries database, to provide storage of collected telemetry: values of sensors, setpoints, meters and so on. The Brick models in HodDB incorporate pointers to the timeseries data in BTrDB, so that the results of a Brick query can be used to retrieve the relevant data from the other database.

The core of Mortar is a declarative API frontend—providing authentication, authorization and error reporting—and a scalable query execution engine that evaluates client queries against the backend HodDB and BTrDB databases.

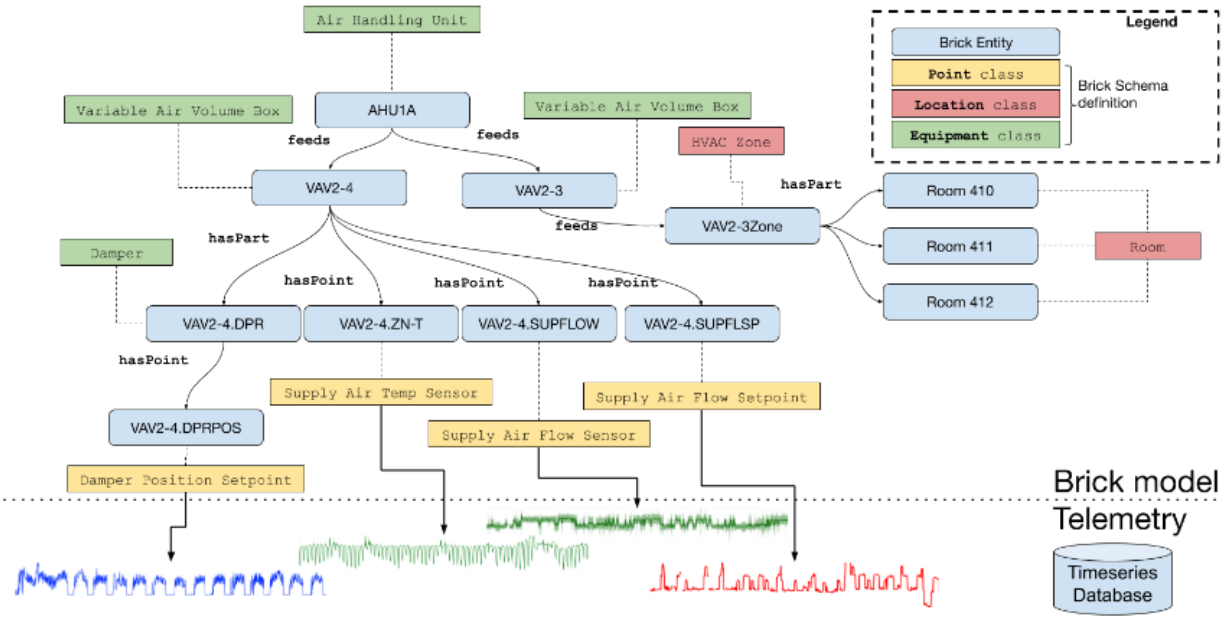
The Mortar platform is implemented and deployed at https://mortardata.org/. We have also published a code repository demonstrating how many features of the Mortar platform could be provided on modern cloud platforms (https://github.com/gtfierro/mortar-parquet-support) using the Apache Parquet data format to efficiently store data.

Develop the testbed

A testbed for building data must support the storage and retrieval of historical timeseries data and its metadata in order to enable the development and execution of applications over the data. Metadata describes properties of each timeseries data source and places it in a larger context. This may include the type or nature of the data source, or some representation of its physical or logical location relative to the structure and composition of the building and its subsystems.

This set of requirements poses a set of challenges for an effective testbed. The first challenge is how to handle the diverse family of data models and metadata representations that define different perspectives on a building. The testbed should standardize on a single data model that can capture the salient aspects of these other representations—this will ensure that metadata can be queried and traversed in the same manner regardless of which metadata sources are available for a particular building.

Figure 17: Brick data model

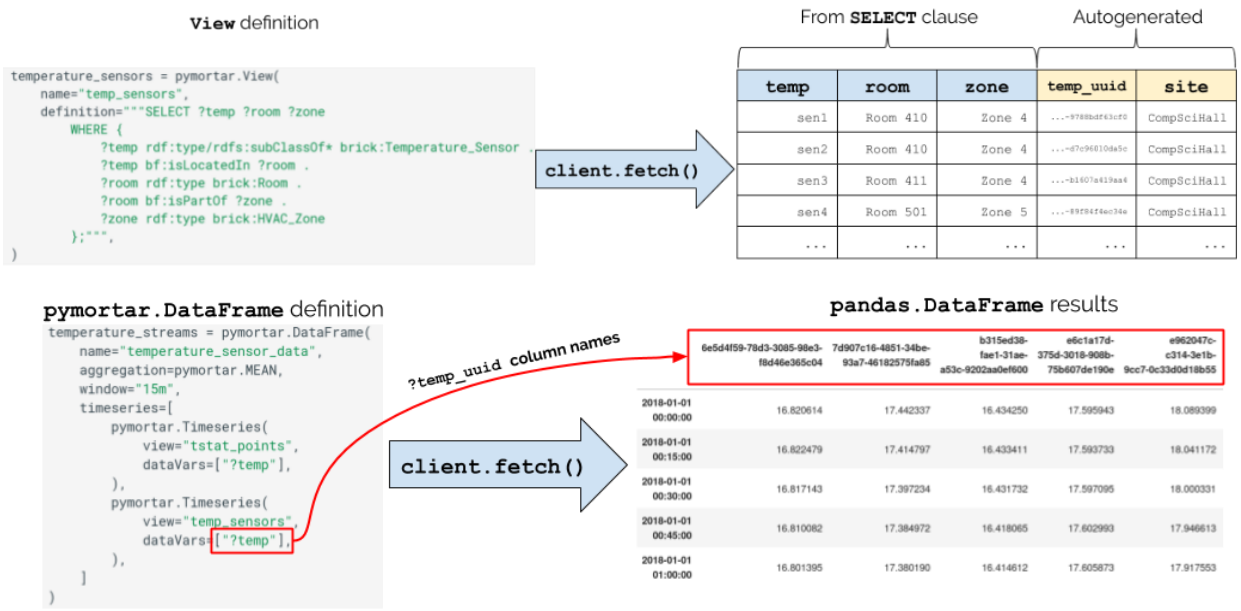


The second challenge for an effective testbed is how to provide a unified query interface over the timeseries and metadata data models, which can be quite different. Crucially, the query interface should support the retrieval of timeseries data that matches metadata predicates. This requires some linking between the two data models which may reside in different databases.

Because Brick is a graph-based data model, it is the most natural to query it with a graph query language such as SPARQL, which returns to the user the set of nodes and edges in the graph that match user-provided patterns (Figure 18). In contrast, timeseries databases are more naturally queried by handing the database a list of data sources, a temporal extent, and optionally aggregations or window functions. The Brick Point class describes sources of digital data in a building; instances of the Point class represent a particular sensor, setpoint, command, alarm or other data source. The names of the Point instances in a Brick model can therefore serve as foreign keys into a timeseries database (Figure 17). This leads us to a potential design for a unified query interface: accept queries against either data model (timeseries or graph) and

convert the results into relational tables that can then be joined using the names of Point instances.

Figure 18: Schematic of the graph query process



Mortar consists of a timeseries database, a Brick database, and an API frontend which performs the necessary federation of queries. These three services are managed using Kubernetes, a container orchestration framework that facilitates the reactive scaling of the backend computation in response to load in addition to continuous monitoring and crash recovery. Mortar supports two timeseries databases: BTrDB and TimescaleDB. BTrDB, developed at Berkeley and now developed at PingThings, provides best-in-class performance and compression for the storage and retrieval of timeseries data and natively supports Kubernetes. However, it only supports the use of UUIDs as timeseries identifiers, so for the BTrDB backend, Mortar annotates Brick point instances with their corresponding uuids. TimescaleDB, built over the well-established Postgres RDBMS, is an open-source timeseries database that provides good performance and the ease-of-use benefits of SQL as a query language, in addition to the ability of using Brick point names as foreign keys. Mortar uses HodDB for storage and querying of Brick models; HodDB[23] (https://github.com/gtfierro/hoddb) is an open-source RDF database and SPARQL query processor that executes Brick queries 3-700x faster than existing open source and commercial offerings, which often take seconds or even minutes to execute a single query on a moderately-sized Brick model. Below, we describe the architecture of Mortar as it relates to the use of HodDB and BTrDB.

[23] Gabe Fierro and David E. Culler. 2018. Design and Analysis of a Query Processor for Brick.ACM Trans. SensorNetw.1, 1, Article 1 (January 2018), 25 pages.

In order for the Mortar[24] platform to handle the current and projected scale of requests, we have designed the Mortar query processor around a SEDA architecture. The SEDA architecture is characterized by distinct process stages connected by explicit queues. Each SEDA stage has a number of parallel workers who take jobs from the stage's incoming queue, do some processing, and insert the job into the stage's outgoing queue. When a stage is overwhelmed – such as due to a lack of compute or memory resources – its incoming queue fills up. This means that the prior stage's workers will be unable to place their finished jobs in the incoming queue for the next stage, which influences the prior stage's incoming queue, and so on (this is called backpressure). A stage can react to backpressure in a few ways: it can add more workers to its stage to handle more requests at a time, or it may choose to do nothing or even reduce the number of workers, which can propagate backwards through each stage and reduce the load on the whole system. If the system is unable to cope with the incoming request load by adjusting internal resources, it can rate limit client requests.

The query processor consists of three stages: frontend processing, metadata processing and timeseries processing. The client request (Figure 18) is processed over these three stages as follows:

Frontend Processing Stage: The frontend stage receives requests from clients in the form of calls to Mortar's GRPC API, prepares requests for execution, and delivers results back to the client using GRPC. The frontend authenticates each incoming request using the JWT (JSON Web Token) included in the request; clients can generate valid JWTs using a *GetAPIKeyAPI* call. After authentication, the frontend stage checks that the request is valid and well-formed. The frontend stage then wraps the client's request in a *Context* object which encapsulates all resources allocated during query execution before attempting to place the request in the metadata stage's incoming queue. Requests to Mortar's *GetAPIKeyAPI* call are answered from the frontend stage alone. Requests to Mortar's `Qualify` API call flow through the frontend and metadata stages. Requests to Mortar's *FetchAPI* call flow through all three stages. Rather than passing the full Fetch response between stages, the metadata and timeseries stages forward incremental parts of the response that can be reassembled by the client into the full dataset.

Metadata Processing Stage: The metadata stage evaluates queries from incoming client requests by executing their Brick query definitions (called *Views*) against the Brick database, which contains the Brick models for all sites in the Mortar testbed. View definitions do not include the binding between points and timeseries identifiers; instead, the metadata stage rewrites View definitions to query for the timeseries identifiers for points that are used in a DataFrame. This simplifies dataset definitions by making transparent the binding between an entity in the Brick model and its corresponding timeseries. To support this query rewriting functionality, we implemented an alternative query frontend to HodDB that accepts parameterized versions of SPARQL queries.

Timeseries Processing Stage: The timeseries stage pulls data from BTrDB for each of the timeseries identifiers derived from the evaluation of the metadata stage's Views. The TimeParams field in the client request defines the temporal extent of the data to be retrieved, and the DataFrame definitions indicate which aggregation function to apply to each stream along

[24] https://github.com/SoftwareDefinedBuildings/mortar

with the desired resolution. Because a client can request tens or hundreds of megabytes of data, the timeseries stage takes care to minimize the performance impact of large queries or slow clients on the rest of the platform. To achieve this, the timeseries stage decomposes the requested timeseries data into small batches that are buffered in memory and enqueued for delivery to the client. The amount of buffered timeseries data is constant for all queries, so the amount of server memory dedicated to serving a request is independent of the size of the requested dataset. The timeseries stage only pulls data from BTrDB as needed, so if a client is slow to read incoming data or terminates the connection, the ongoing query to BTrDB can be terminated and the resources released without having read data unnecessarily.

Demonstrate the testbed

The Mortar testbed, along with a Brick application that reports all the mapped physical, logical, and virtual assets of a building, proved useful as a screening tool. The team used these tools to provide a shortlist of buildings that merit further investigation and target extra metering for an ongoing California Energy Commission (CEC) project called *Getting out of hot water: Reducing gas consumption in existing large commercial buildings.* This project requires the evaluation of large buildings with spaces conditioned through heating systems with hot water reheat coils e.g., variable air volume (VAV) reheat boxes to reduce natural gas consumption. The evaluation requires VAV discharge air temperature and air flow rate for the analysis. Thus, the team queried all the sites in the Mortar dataset for the required data points and combined with external metadata to produce the shortlist.

Furthermore, the team are currently using the "detect_passing_valves" application, described in the next section, on the Mortar dataset to determine the frequency of passing valves in existing buildings. The Mortar dataset contains over 1,000 VAV boxes with relevant data points. It is a rich dataset that has not been publicly available before and will shed light on the issue of passing valves.

Rogue Airflow Analysis

Another fault detection application developed on the Mortar platform focuses on identifying "rogue" behaviors in the air flow within an HVAC system in a building, such as zones whose indoor air temperatures are substantially different than the setpoint temperature. The Brick query for this application filters all buildings whose equipment have both the points of type (or sub-type) brick:Air_Flow_Setpoint and brick:Air_Flow_Sensor. This includes Air Handler Units (AHUs) that have, for example, both supply air sensors and setpoints and return air sensors and setpoints. VAVs that also track the zone discharge air flow will also be included in this analysis. Once the shortlist of buildings and equipment have been identified, the timeseries data is retrieved and it undergoes a level of data processing and cleaning to prepare for the analysis.

This application identifies "rogue" behavior in an equipment when the air flow value measured by its sensor does not meet the corresponding air flow setpoint. The tolerance and minimum period of the out of bounds behavior are configurable, and a snippet of the output of this application running on the Mortar platform (with 10 °F deviation from setpoint for at least two consecutive hours) is show in Figure 19. It shows the start time of the incident, the duration, the building it happened in, the equipment name and the average deviation in actual air flow from the setpoint (in cfm).

Figure 19: Output of Mortar for rogue zone query

start	hours	building	equip	deviation
2018-06-03 14:00:00+00:00	4.0	vm3a	VAVRM3300H_LAB	18.707500
2018-06-04 22:30:00+00:00	3.5	vm3a	VAVRM3300H_LAB	14.841429
2018-06-06 05:30:00+00:00	2.0	vm3a	VAVRM3300H_LAB	16.395000
2018-06-12 06:00:00+00:00	2.0	vm3a	VAVRM3300H_LAB	18.395000
2018-06-12 10:00:00+00:00	3.0	vm3a	VAVRM3300H_LAB	19.020000

Automatic selection of analytics

Dr. Fierro developed an API call and implementation supporting the automatic selection of analytics for a suite of Brick models. Specifically, the solution enables:

A) Given an analytics application, the solution automatically determines the set of Brick models that support the application, and

B) Given a Brick model, the solution automatically determines the set of analytics applications that will run

The solution depends on having access to a library of analytics implementations, each with a "manifest" of the Brick queries defining the metadata the implementation requires to operate. Such a library has been developed as part of the Mortar platform. The API call, named "qualify", takes as arguments a list of Brick queries and a list of Brick model names and returns a matrix where cell *(i,j)* contains the number of rows returned when query *i* is executed on model *j*. By providing the application manifest's queries as one argument and the set of Brick models in a database (such as Mortar) as the other argument, the *non-zero* columns of the matrix correspond to Brick models that can run the application. This enables the feature (A) above.

The automatic selection of analytics is implemented by adjusting the above process such that the call to "qualify" takes as an argument the Brick queries for *all* analytics implementations in the library, with the second argument being a single Brick model. The *non-zero* rows of the matrix correspond to queries that are supported by the model. The analytics which have only non-zero rows are supported by the model; this list can be provided to the user.

Version control of Brick models

An important concern when using metadata to drive applications is how to manage and deal with churn in the metadata. Here, we describe a simple version control solution to graph-based metadata models such as those based on Brick.

Metadata churn has two primary sources:

- change in the environment / deployment, which is reflected in the contents of the model

- change in the definition of the Brick ontology

With a new minor release of Brick every 6 months, and patch releases arriving even more often[25], it is important to distinguish between backwards-compatible and backwards-incompatible changes. The version control mechanism described below focuses on backwards-compatible changes; a discussion of the implications and challenges in dealing with backwards-incompatible changes is provided at the end.

Recall that applications execute by querying a Brick model describing the context of data sources in a particular building. To handle versioning, we must attach a pair of timestamps t_i,t_j to a Brick model, indicating that the Brick model represents the content of a building in the range [t_i,t_j) (inclusive lower-bound). A Brick model at t_i,t_j is immutable—that is, the content of the model will not change. Changing the Brick model requires the creation of a new version. When executing, applications must specify which version of the Brick model is required.

In most cases, this will be the most recent version.

It is the role of a Brick database and query processor to execute application inquiries against the indicated version. This can be implemented as follows. A database D stores a collection of (g,m,s,p,o,t) tuples. The graph g is the name of a Brick model, representing a single site. Updates to a graph come from metadata sources. A metadata source m is a particular source of triples for a particular graph over time. The content of the metadata source will change, but the identity does not. Examples of metadata sources are the content of the Brick ontology, or an inferred Brick model produced by some process such as Fierro et al, 2020[26]. s, p and o are the three components of an RDF triple at a point at time t for the metadata source m.

This structure allows a database to derive the contents of the Brick model at any point in time. To derive the Brick model for timestamp t and graph g, the database searches for the most recent timestamp for each metadata source for that graph that matches or is before the query timestamp t. The union of the triples for each recent metadata source constitute the Brick model at that timestamp.

The advantage of this approach is that it allows both the ontology definition, which is expressed as triples, and the building instance definition to evolve independently. Brick models can be updated to the latest backwards-compatible version of Brick simply by inserting the new ontology definition into the database at a newer timestamp but using the same metadata source name. One crucial assumption being made is that the data sources described in the Brick model do not change identity unnecessarily. Updating a Brick model, for example to add a recently installed thermostat, should not invalidate historical data. For this reason, most of the Brick model is likely to remain the same between versions. Data sources should only be assigned new names when there are changes to the data source that affect its context or interpretation; this will coincide with the creation of a new Brick Point entity to identify the data source.

Backwards-incompatible changes to the ontology present a challenge for automated version control. In the context of a version history of a Brick model, backwards-incompatible changes in

[25] https://github.com/BrickSchema/Brick#versioning

[26] Fierro, G., Prakash, A. K., Mosiman, C., Pritoni, M., Raftery, P., Wetter, M., & Culler, D. E. (2020, November). Shepherding Metadata Through the Building Lifecycle. In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (pp. 70-79).

the ontology reflect a barrier across which automated tools or queries cannot cross without knowing exactly how to interpret the different ontology version on the other side. For example, early versions of the Brick ontology contained so-called "equipment-flavored points" such as "AHU Outside Air Temperature Sensor". Subsequent versions of Brick decided to make the separation between equipment and point explicit. The "AHU Outside Air Temperature Sensor" concept is best modeled as an AHU instance and an Outside Air Temperature Sensor instance that are related to one other with the Brick hasPoint relationship.

How to handle these kinds of backwards-incompatible changes is being actively researched. The proposed technique is called "segmented query generation": it uses succinct rules to automatically rewrite queries so that the correct semantics are preserved across versions of an ontology.

Task 4: Develop and Apply Analytics and Controls Applications

The research team first identified potential analytics and controls, develop at least two simple algorithms, and demonstrate at least two more complex algorithms.

Identify potential analytics and controls

In the first year, the research team collaborated with researchers at NREL[27] and Center for the Built Environment (CBE) partner firms to develop a list of 21 representative applications that highlight the descriptive capabilities of Brick models. As part of this process, we identified quantities and constants that applications make use of but that are not currently captured in the Brick data model, for example zone floor area or equipment rated capacity. We investigated existing standards and vocabularies such as bSDD (http://bsdd.buildingsmart.org/) that we can build upon and incorporate into Brick. We have identified a mechanism for formalizing the definitions of these quantities and constants in a manner that is consistent with the rest of Brick. In addition, we are developing a plugin-based software architecture to support the retrieval, storage and standardization of such external metadata. Table 8 lists a number of applications; we have started the list with HVAC applications since they are the most complex and systems that can most benefit by semantic interoperability technologies.

Table 8: Prioritized list of applications for Brick

Application Name	High Level Functional Description
Detect Cooling Coil Valve Passing	Cooling Coil Valve Passing/Leaking: If there is a considerable temperature difference between mixed air temperature and supply air temperature while the cooling coil valve is closed, the fault will be triggered. The purpose of this use case is to check if there is a mechanical problem/leakage in the cooling coil valve.
Detect Coil Valve Passing	Generalized Coil Valve Passing (i.e. fluid flows when valve is commanded closed): If there is a considerable temperature difference between the next upstream and downstream air sensors in the system while the valve has been closed for a given time period. This could also include a test to evaluate what other components are in the brick model between the upstream and downstream air temperature sensors.
Calculate Cooling Plant Efficiency	Accessing power (kW) and cooling load (ton) data from a chilled water system (e.g., chiller power, pump power and cooling tower power), and utilize the data to estimate and report system energy performance (kW/ton). The purpose of this use case is to monitor the performance of the system and take corrective action when needed.
AHU Outside Air Economizer Operation	AHU Airside Economizer Assessment: Energy conscious control strategies should utilize free or economized cooling when able. The purpose of this use case is to assess how well the economizer control strategy is working. Specifically, this test will look at the outside air damper and cooling coil valve position. Economizer dampers switch to the minimum ventilation position when outside air conditions are not favorable for cooling and vice versa.

[27] Semantic Interoperability R&D Project Kick-Off Workshop at NREL, December 12-13, 2019

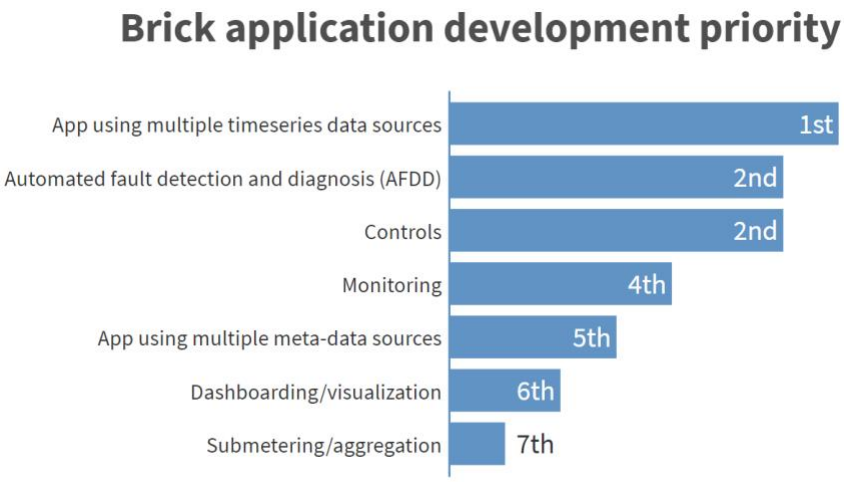
Improve HVAC Sequence of Operation: Occupancy-Based Equipment Scheduling	Airside Occupancy and Seasonal Scheduling: Depending on the season, time of day, and day of the week (week / weekend / holiday), airside equipment shall use different setpoints for heating and cooling. The purpose of this use case is to evaluate the performance of seasonal and temporal setpoint control strategies, specifically using occupied and unoccupied schedules. Usage of occupancy sensors and dynamically generated occupancy schedules will also be evaluated to determine how energy performance changes compared to season, time of day, and day of week control strategies.
Optimize Chilled water supply setpoint	Chilled water supply setpoint optimization: Develop a model-based optimization that will generate optimized chilled water supply setpoint by comparing the energy consumption from the pump and chiller side. The purpose of this use case is to evaluate the performance of chilled water supply setpoint reset strategy. Specifically, this optimizer will evaluate the the energy consumption by the chillers and chilled water pumps corresponding to a set of chilled water supply setpoints and pass the one with the least total energy consumption.
Estimate Time to Replace Air Filters for AHU	Air Filters for AHU Preventive maintenance: Depending on the pressure drop across the air filter, the filter should be maintained/replaced. A high pressure drop means that your air handler has to work harder and consume more energy in order to maintain proper air flow, and the filter should be replaced
VAV Box Commissioning	VAV Box Commissioning: Perform all relevant functional performance tests for a VAV box to support (and reduce the time/cost associated with) initial or retro
Automatically Generate BAS Graphics for a VAV box	Automatically generate the equipment graphics for a VAV BOX
Assess Long-Term Ability to Meet Thermal Comfort Needs	Assess long term ability to meet thermal comfort needs within the building. Identify zones where setpoints are frequently not met, and identify zones that have unusual setpoint temperatures.
Detect Boiler Short Cycling	Identify how often a boiler operates below it's minimum turndown and is short cycling. Boiler efficiency drops of very significantly under these conditions.
Estimate Hot Water Intentional Reheat Use	Hot water intentional reheat use. Estimate the amount of heating power used intentionally by a reheat coil in a VAV box.
Detect Fixed Equipment Setpoint	Identify fixed equipment setpoints as this likely indicates an opportunity for improved energy performance at low cost.
Improve HVAC Sequence of Operation: G36 Trim and Respond Logic	Generalized trim and respond logic implementation to perform demand based reset of a setpoint. See ASHRAE Guideline 36 section: 5.1.14. Example setpoints controlled using trim and respond using feedback from a VAV reheat box: 5.6.8 (includes duct static pressure, ahu supply air temp, hot water temp, chilled water temp, pump differential pressure setpoint resets)
Improve HVAC Sequence of Operation: G36 Supply Air Temperature Reset	AHU Supply air temperature reset based on zone level requests and outside air temperature conditions; G36 section describing trim and respond: Cooling SAT request: 5.7.8.1. See Fig 5.16.2.2 showing how SAT setpoint is constrained based on OAT.

Assess VAV box minimum airflow	Identify incorrect high minimum airflow setpoints, as this wastes fan, reheat and cooling energy consumption (but does not increase ventilation rates)
Identify Exterior Zones	Identify exterior zones of a building.
Improve HVAC Sequence of Operation: G36 Split Signal Damper Control Strategy	Split signal damper control strategy. This reduces fan power by keeping dampers more open than traditional logic. See ASHRAE Guideline 36 Fig 5.16.2.3-2.
Improve HVAC Sequence of Operation: G36 Time-Averaged Ventilation	Use time averaged ventilation to achieve the desired minimum airflow rate from a VAV box where the lower control limit is constrained by existing hardware and instrumentation. See ASHRAE Guideline 36 section 5.2.2.
Detect sensor faults	Identify sensors reporting incorrect values - fixed values, values out of range, etc - to be replaced.
Detect adjacent open zone heating/cooling mode conflicts ('fighting' zones)	Identify zones that are adjacent to each other, open spaces, and frequently operate in conflicting heating/cooling modes. This often occurs in commercial building open plan spaces due to conflicting zone thermostat temperature changes.

Discuss list of applications with stakeholders

We met with our Technical Advisory Group to present the Brick project, and to receive feedback on the preliminary application list developed. We presented several applications and asked the TAG to rank the priority of importance to them. The figure below shows the ranking.

Figure 20: Priority of Brick application development



The top five Brick application category types as voted by TAG members:

A. Application using multiple timeseries data sources
B. Automated fault detection and diagnosis (AFDD)
C. Controls
D. Monitoring
E. Application using multiple meta-data sources

Table 9 below shows a summary of suggested applications compared to the data available in the Mortar database (over 100 building sites).

Table 9: Summary of apps' applicability and category.

	App type	Multiple timeseries sources	AFDD	Controls	Monitoring	Multiple meta-data sources
Suggested apps to develop	Potential applicability: Sites on current database	Category				
1. Assess thermal comfort	52					
2. Detect passing valves in VAV	51					
3. Saving potential of correct ventilation minimum	51					
4. Dynamic zone ACH	51					
5. Calculate dynamic building GHG emission	46					
6. Control duct static pressure	20					
7. Control boiler temperature	1					

These seven are our prioritized list of applications to fulfill the top voted category types.

1. **Assess long-term ability to meet thermal comfort needs by zone in a building.** The analysis can be performed using existing *temperatures* and associated *setpoints* or based on thermal comfort models. The output can be:
    i. raw number of discomfort hours per zone or;
    ii. a percentage of total occupied hours or;
    iii. weighted by the number of hours at specific delta temperature to get a "degree-hours" metric.

iv. Long term thermal comfort metrics (such as those recently evaluated by CBE researchers (Li et al, 2020[28]).

2. **Detect passing reheat valves in VAV.** Compare temperatures upstream (*discharge temperature* from AHU) and downstream (*discharge temperature* from VAV) of the reheat coil.
3. **Savings potential of using designed zone minimum ventilation requirements.** Perform a comparison between the implemented zones' minimum *ventilation* requirements and the designed zone *ventilation* requirements as found from architecture drawings.
4. **Dynamic monitoring of air changes per hour (ACH) in zones.** Combine real-time *airflow measurements* with *architectural data* to return ACH. Useful for analyzing COVID-19 risks.
5. **Calculation of a building's actual greenhouse gas emissions (GHG).** Sync CAISO data, electricity meter, NOAA, and/or marginal emissions data to calculate actual GHG.
6. **Supply air duct static pressure reset control using trim and respond logic.** Guideline 36 Trim and Respond Logic
7. **Boiler temperature reset control using trim and response logic.** Hot water reset based on zone level requests and outside air temperature conditions; ASHRAE RP 1711 /Guideline 36 Trim and Respond Logic. Adaptable depending on the time constant of the system e.g., a hot water loop serving VAV or TABS systems will be different.

Develop two simple algorithms

The following describes two simple algorithms developed at the end of the first year: Comparing sensor measurements with setpoints, and detecting passing valves.

Compare sensor measurements against setpoints[29]

This application compares sensor measurements against their respective setpoints. The application retrieves sensor points where measurements 1) are above or below its setpoint, 2) are in between both its minimum and maximum setpoint values, 3) exceeds either its minimum or maximum setpoint value, or 4) exceeds a user-defined absolute threshold from its setpoint. The retrieval criteria can also incorporate a minimum timeframe threshold in which to return the sensor point e.g., sensor measurement exceeds setpoint for at least half an hour. The application produces a CSV file with relevant information that includes site, equipment name, start and end date and number of hours that sensor point meets the selected criteria, and average sensor measurement and setpoint values. Table 10 below shows an example of the CSV output file for supply air temperature sensors. The source code for this application is found at

[28] Li, Peixian, Thomas Parkinson, Stefano Schiavon, Thomas M. Froese, Richard de Dear, Adam Rysanek, and Sheryl Staub-French. 2020. "Improved Long-Term Thermal Comfort Indices for Continuous Monitoring." Energy and Buildings 224 (October): 110270. doi:10.1016/j.enbuild.2020.110270.
[29] Sun, Ruiji, Carlos Duarte Roa, Paul Raftery, and Gabe Fierro. 2022. "Enabling Portable and Reproducible Long-Term Thermal Comfort Evaluation with Brick Schema and Mortar Testbed." In 2022 ASHRAE Annual Conference. Toronto, ON, Canada.

https://github.com/SoftwareDefinedBuildings/mortar-analytics/tree/compare_sensors_against_setpoints/.

Table 10: Example of the output file that the application produces.

Site	Equipment	Hours	Start	End	Avg. value	Avg. SP	Exceed. Diff.
stor	http://buildsys.org/ontologies/STOR#AHU00	0.5	2018-05-07 09:30:00+00:00	2018-05-07 09:45:00+00:00	61.8	64.4	-2.7
stor	http://buildsys.org/ontologies/STOR#AHU00	0.75	2018-05-10 08:45:00+00:00	2018-05-10 09:15:00+00:00	61.7	64.7	-3.0
stor	http://buildsys.org/ontologies/STOR#AHU00	0.5	2018-05-19 09:15:00+00:00	2018-05-19 09:30:00+00:00	61.8	64.7	-2.9
stor	http://buildsys.org/ontologies/STOR#AHU00	0.5	2018-05-31 10:15:00+00:00	2018-05-31 10:30:00+00:00	61.5	63.8	-2.3
rech	http://buildsys.org/ontologies/RECH#AHU_AC4	1	2018-05-31 17:45:00+00:00	2018-05-31 18:30:00+00:00	67.8	67.8	0.0
rech	http://buildsys.org/ontologies/RECH#AHU_AC4	8.25	2018-05-31 21:00:00+00:00	2018-06-01 05:00:00+00:00	66.5	69.7	-3.2
rech	http://buildsys.org/ontologies/RECH#AHU_AC4	0.5	2018-06-01 05:30:00+00:00	2018-06-01 05:45:00+00:00	63.0	61.5	1.5
rech	http://buildsys.org/ontologies/RECH#AHU_AC4	0.5	2018-06-01 17:00:00+00:00	2018-06-01 17:15:00+00:00	68.0	51.7	16.2

Detect passing valves[30]

This application detects valves in HVAC equipment that do not close fully even when actuated to a fully closed position, also known as "passing valves". The application compares fluid temperatures upstream and downstream from the coils that the valve controls and calculates the expected long-term difference between the two fluid streams when the valve is currently closed, and has been closed for some time. The app then analyzes the expected trends with the actual data to determine if the valve is in good operating condition or malfunctioning e.g., passing valve, sensor fault, loss of communication with building automation system, or forgotten overrides. Figure 21 below shows an example where the application detected a good operating valve and a malfunction valve in one site. The source code for this application is found at https://github.com/SoftwareDefinedBuildings/mortar-analytics/tree/detect_passing_valves/ .

Viewing Figure 21, the top plot shows the data where the application detected a correctly operating valve and the bottom plot shows data where the application detected a possible malfunctioning (passing) valve. The solid green horizontal line shows the average temperature difference between the downstream and upstream fluids when valve is commanded closed, the

[30] Duarte Roa, Carlos, Paul Raftery, Rupam Singla, Marco Pritoni, and Therese Peffer. 2022. "Detecting Passing Valves at Scale Across Different Buildings and Systems: A Brick Enabled and Mortar Tested Application." In Climate Solutions: Efficiency, Equity, and Decarbonization. Pacific Grove, CA: ACEEE. doi:10.20357/B7VP5H.

dashed purple line shows the expected correct operating behavior trend of the valve (based on the green points), the solid pink horizontal line shows the average temperature difference when the application detected a possible passing valve (based on the red points). The ‘Bad ratio’ value is a ratio of the number of bad operating point values to good operating point values.

In summary, we ran application in 20 buildings including two large commercial buildings in which we were able to 'ground-truth' the findings by field investigation of several of the identified passing valves by the on-site facilities team members. We analyzed 1,335 VAV terminal units where 5% returned a sensor fault and 14% returned a valve fault. We calculated an average temperature rise of about 6 °F between the air upstream and downstream of the reheat coils. The temperature rise is about three times higher when compared to the temperature difference on a VAV terminal unit with no fault detected which is about 2 °F. We estimated an 8% heat loss of intentional reheat energy due to a passing valve. In the two large commercial buildings, we reported six VAV terminal units with faults. Three VAV boxes were verified with faults and one additional VAV terminal unit was found as faulty by the on-site facilities when conducting the investigations. Two units were found to have no faults and last unit was not checked. Figure 22 shows two verified faults found in the two large commercial buildings. The faults we reported included valves in operator override, with loss of communication with building automation system, and with its operation reversed e.g., valve opened when it should have closed.

Figure 21: Using Brick to enable the detection of malfunctioning valves

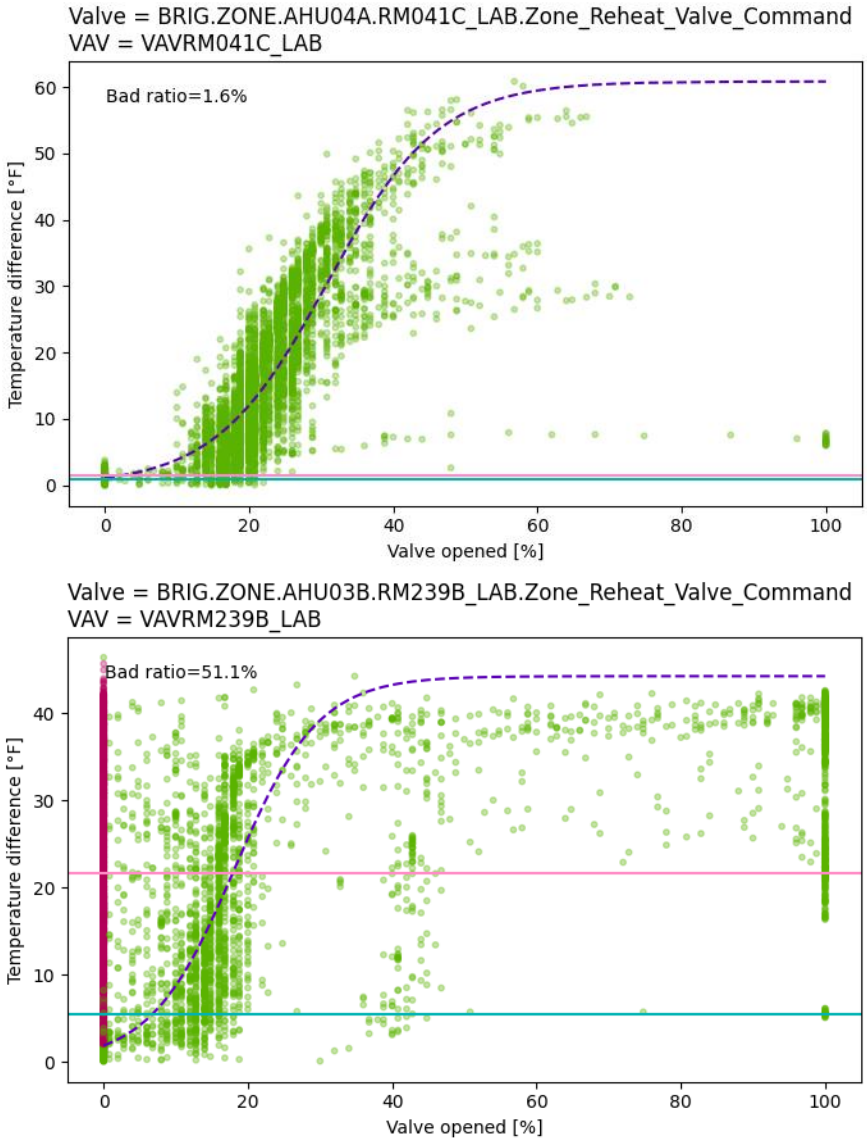
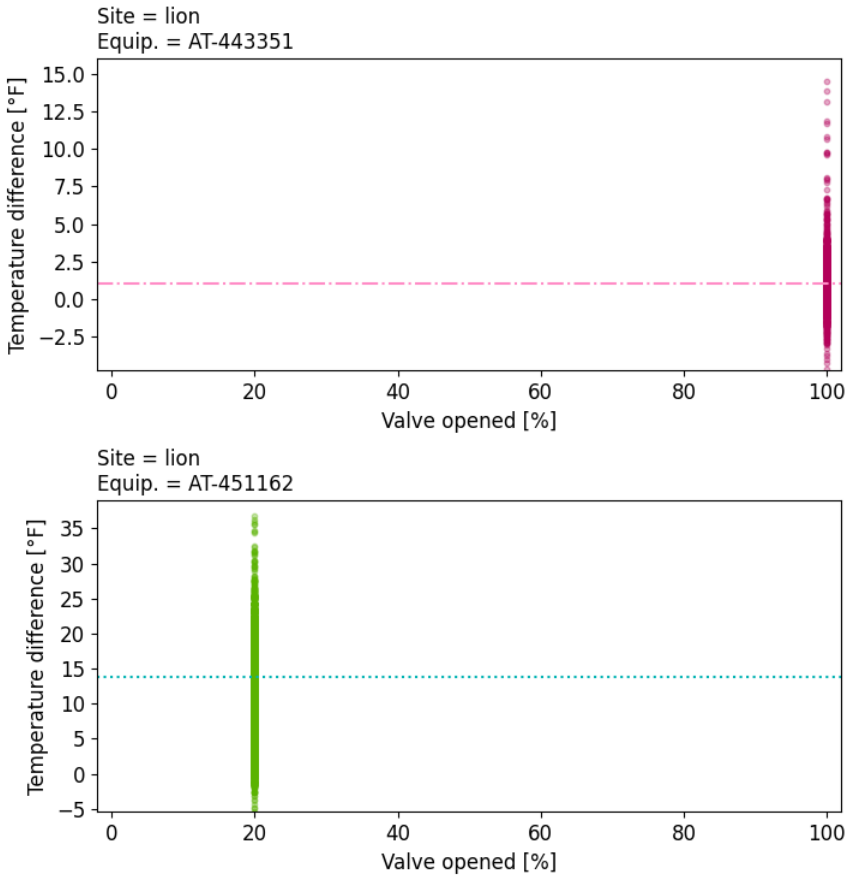


Figure 22: The detect passing valve application detected valve faults that were verified in two large commercial buildings.



Open-source library of applications

We maintain an open-source public GitHub repository for Brick applications, found at https://github.com/BrickSchema.

Here one can find packages (py-brickschema: Python package for working with Brick), services (reconciliation-api for OpenRefine and other tools), and other connectors (brick-BACnet). This is the repository for tools and applications, such as Brick versioning, graph inference, conversion from Haystack, VBIS translation, web-based interaction, Brick model validation.

Many applications meant to run on buildings and tested first on the Mortar platform can be found at the UC Berkeley research GitHub site: https://github.com/SoftwareDefinedBuildings/mortar-analytics. These include an application to find simultaneous heating and cooling zones, possible inefficient zones, “rogue” zones, detect passing valves, temperature reset, and so on.

We will continue adding to this repository over time.

Demonstration of two more complex applications

The project developed two additional applications: hot water supply temperature reset (tested in the David Brower Center) and supply air duct static pressure reset control (designed for Sutardja Dai Hall). The goal of these applications is to show how developing a Brick model for a building HVAC system enables implementation of Guideline 36 sequences. First introduced in 2018 by ASHRAE, Guideline 36 (G36) contains standardized advanced control sequences to facilitate the implementation of high-performance HVAC control. However, G36 sequences are currently presented as English language specifications of controls and not the actual programming code syntax uploaded to the building's BAS. Another enabling mechanism is the OpenBuildingControl project, which aims to digitize the design, specification, deployment and verification of building control sequences such as G36. Researchers in the project developed the Control Description Language (CDL), built on the equation-based modeling language called Modelica[31], and used this to express control sequences for BAS in a vendor-independent format. CDL has already been used to program G36 and other high-performance HVAC control sequences and these CDL representations are currently available in the Modelica-Buildings library.

These applications were both developed for buildings in Berkeley, CA, which enjoys year-round moderate temperatures and is located in ASHRAE climate zone 3C.

Hot Water Supply Temperature Reset

The demonstration building is a LEED Platinum, four-story mixed-use building, the David Brower Center, located in downtown Berkeley, California with about 39,000 ft^2 (3,600 m^2) of conditioned space. The building's program consists of private and open-plan offices, conference rooms, an auditorium, and a gallery. The HVAC system includes a thermally activated building (TABS) radiant system for the primary heating and cooling in the office spaces. Two air-handling units (AHU) supply 100% outdoor air to an underfloor air distribution (UFAD) system and combined with natural ventilation through operable windows provide ventilation to the building. The radiant system does not thermally condition the first floor. Instead, seven water-to-air heat pumps provide heating and cooling, and an eighth heat pump is located in a second-floor conference room. The heat source for the radiant system, heat pumps, and the two AHUs for the ventilation system is provided through two gas condensing boilers each with an input capacity of 26 Btu/hr-ft^2 (82 W/m^2). The boilers have a lead-lag operation and efficiency ranges between 85% and 95% depending on the operating mode and return water temperature when operating at or above the minimum turndown capability of the boilers. This G36 field demonstration facilitated with a Brick data model is focused on a hot water supply temperature setpoint (HWST) reset on the building's hot water plant illustrated in Figure 23.

[31] modelica.org/modelicalanguage

Figure 23: Schematic of the demonstration building's hot water plant.

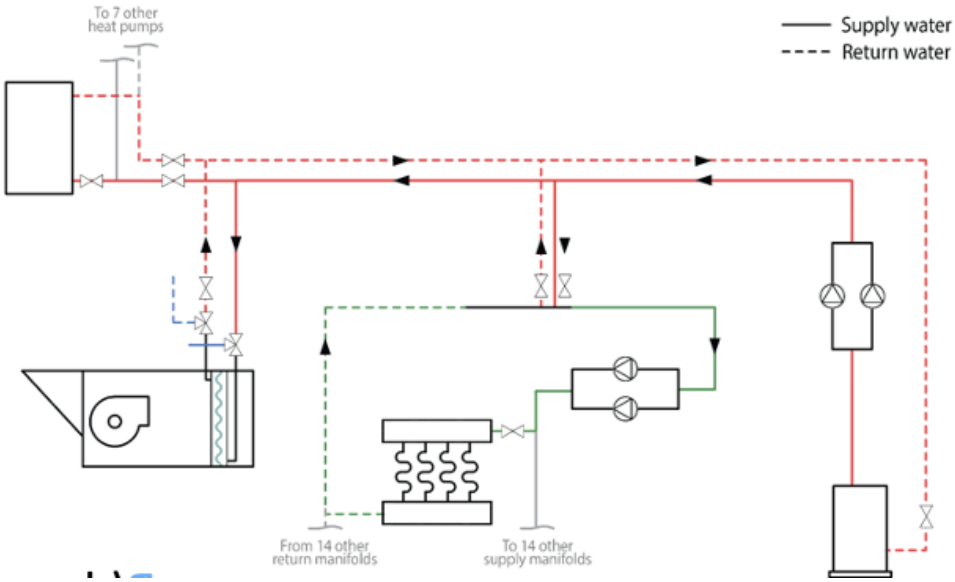
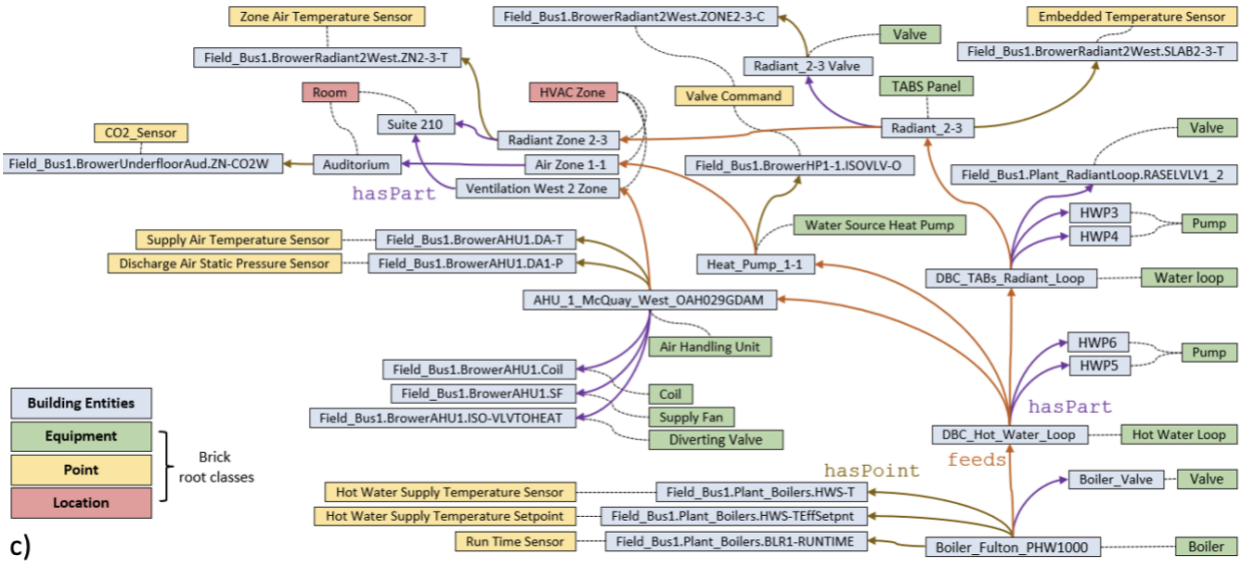


Figure 24 below shows how Brick can be used to digitally represent a hot water plant.

Figure 24: Brick data model example for the hot water plant shown in Figure 22



The existing HWST reset strategy is based on the outdoor air temperature (OAT). The HWST was designed to be at 95 °F (35 °C) when OAT was at 40 °F (4.4 °C) and 75 °F (24 °C) when OAT was at 65 °F (18 °C). However, the building manager changed it to a more conservative operation (HWST=130 °F (54 °C) at OAT=55 °F (13 °C) and HWST=90 °F (32 °C) at OAT=77 °F (25 °C)) since the building manager believed the designed temperature setpoints were not high enough to maintain occupant thermal comfort during cold days. On some occasions, the

building manager would override the reset operation and maintain a constant 130 °F (54 °C) HWST.

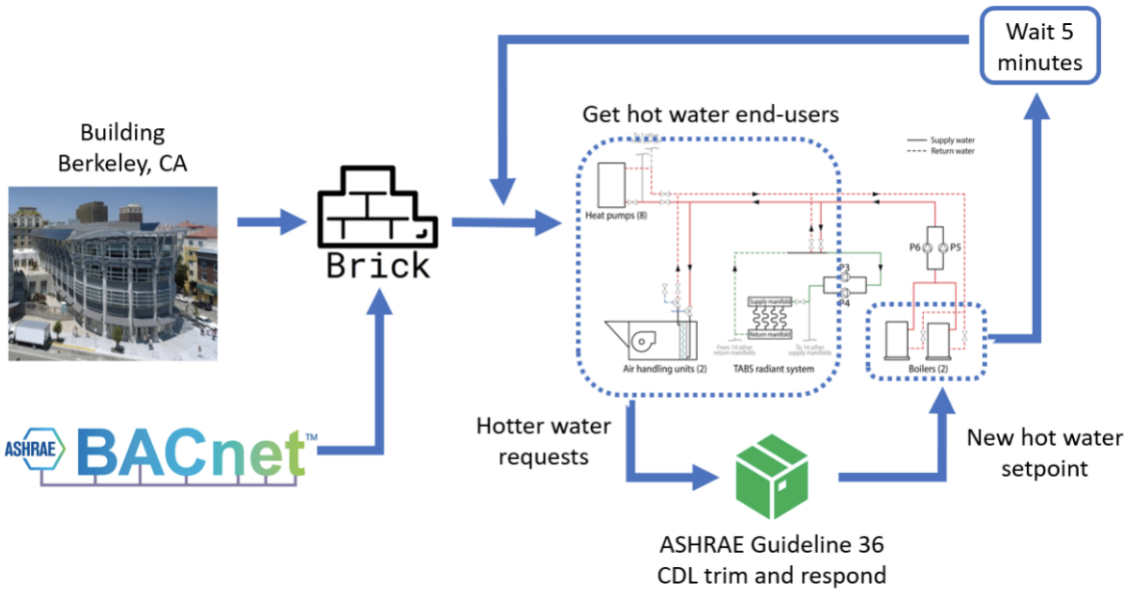
We changed the radiant system control strategy based on Raftery et al. (2017)[32] as part of an earlier study (Bauman et al. 2018)[33], which allowed the building manager to feel more confident in reinstating an HWST reset strategy. This provided an opportunity to implement an HWST reset strategy based on G36's trim and respond control strategy, using Brick and CDL.

To integrate with the building's BAS, we connected a miniature computer (PC) to the building's BAS network infrastructure, allowing us to read from and write to BACnet objects. We used the open-source project BACpypes to provide the BACnet application and network layer to establish communication for our PC to the BAS. We performed a BACnet network scan and point list retrieval using open-source network discovery utilities such as Nmap. The BACnet point list gave us the starting point to follow a five-step process to build a Brick data model of the building. The five-step process is as follows: 1) collect siloed metadata for the building, 2) organize it into more manageable formats, 3) transform metadata into a Brick model, 4) apply inference and reasoning to the initial Brick model to discover implied information, and 5) validate the Brick model to ensure we used Brick classes and relationships correctly. We also embed BACnet object information within the Brick model and access information for an external database collecting historical data from the building's BAS.

We used the building's Brick model to retrieve HVAC system design information and pertinent BACnet object information to obtain the HVAC's current operating status. This operation status is forwarded to the CDL G36 trim and respond control sequence so it can calculate a new HWST. The new HWST is written back to the hot water plant, and once again, we use Brick to retrieve the required HWST BACnet object information so the boilers can use it. Figure 25 shows a schematic of the field demonstration implementation. The Brick model allows us to retrieve the hot water plant's end-users and their current operation, which is used by G36 trim and respond to calculate a new HWST. The HWST is written back to the BAS and the process repeats every 5 minutes. The dotted boxes denote where we query the Brick model in the programming implementation.

[32] Raftery, Paul, Carlos Duarte, Stefano Schiavon, and Fred Bauman. 2017. "A New Control Strategy for High Thermal Mass Radiant Systems." In Proceedings of Building Simulation 2017. San Francisco, CA: International Building Performance Simulation Association. http://escholarship.org/uc/item/5tz4n92b.
[33] Bauman, Fred, Paul Raftery, Stefano Schiavon, Caroline Karmann, Jovan Pantelic, Carlos Duarte, Jonathan Woolley, et al. 2018. "Optimizing Radiant Systems for Energy Efficiency and Comfort." EPC-14-009. Sacramento, CA: California Energy Commission. https://escholarship.org/uc/item/6qx027rh.

Figure 25: Schematic of the ASHRAE Guideline 36 (G36) hot water supply temperature setpoint (HWST) reset strategy implementation in a Berkeley, California building facilitated with Brick.



The Modelica-Buildings library contains the CDL representation of different standardized control sequences including G36 sequences for hot water plant control. These sequences and any model within the library can be exported as a Functional Mockup Unit (FMU), conforming to the Functional Mockup Interface (FMI) standard. FMI was developed to allow model exchange and co-simulation of models created in different simulation environments. The FMU is a packaged file that contains details about the model parameters, variables, equations, and other relevant information to run a simulation. We exported the CDL representation of the G36 HWST reset control strategy as an FMU. The G36 HWST reset control requires signals to denote the current status of the hot water plant, whether or not it is in the staging process, the status of the pumps, the current HWST, and the number of *requests* for higher hot water supply temperature.

The trim and respond is a demand-based reset control strategy that can control a single variable subject to multiple input sources. This control can be applied to reset setpoints such as pressure, temperature, and other variables. The control logic *trims*, or reduces, the controlled setpoint at a fixed rate until the downstream equipment generates requests for higher pressure, temperature, or whatever the logic is applied. The control logic *responds* by increasing the setpoint when sufficient *requests* are generated and the cycle is repeated.

We developed generalized programmatic queries to first search the building's Brick model for the hot water plant boilers and the hot water end-users. Then, we queried each end-user to retrieve its flow control valve and pertinent BACnet object information. Once we retrieved pertinent BACnet information for the hot water end-user, we used BACpypes[34] to send messages over BACnet to determine if the end-user is enabled and its water valve is open over 95%; this formed the basis for a *request* needed for the trim and respond logic. The total sum of *requests* is sent to the FMU package containing the CDL G36 reset control to calculate a new HWST. We

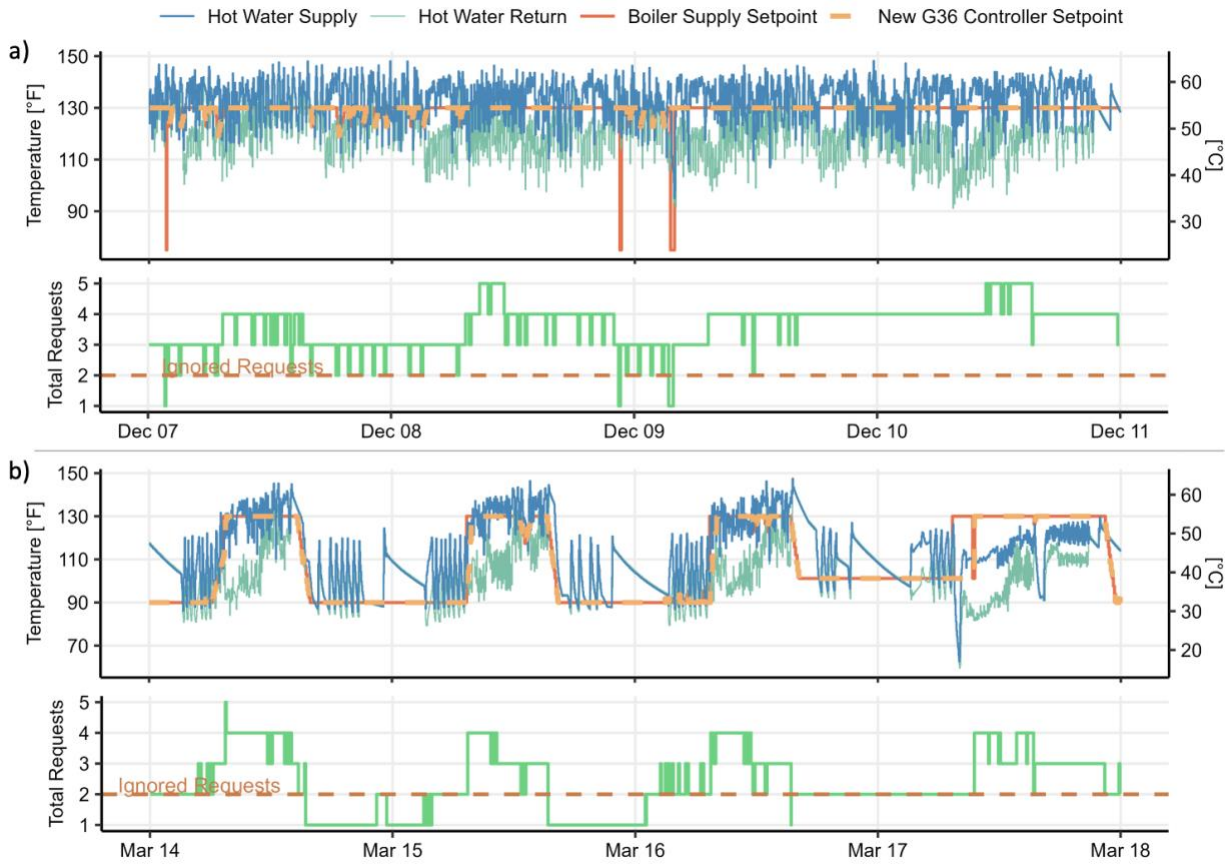
[34] https://github.com/JoelBender/bacpypes

set the total number of ignored *request*s to two, so it takes at least three *requests* for the HWST to start increasing. The new HWST is written back to the BAS with BACpypes using BACnet object information retrieved from the building's Brick model. The process is repeated every five minutes. We setup a Python 3.6 environment on our BAS connected PC with Python packages pyfmi[35], brickschema[36], asyncio[37], and other supporting packages to load, initialize, and simulate the FMU, read and query the Brick model, and setup periodic intervals to repeat the control strategy. We started running the Brick enabled G36 HWST reset control on November 30, 2021 with 130 °F (54 °C) and 90 °F (32 °C) as the upper and lower setpoint limits, respectively. These controls have been running through late July 2022.

Figure 26 shows the Brick enabled G36 HWST reset control implementation. The first few weeks of the Brick enabled controls were during winter when the outdoor temperatures were lower. Thus the number of calculated hotter water requests was above the ignored request threshold. The HWST mostly operated at the upper temperature setpoint limit we defined, as shown in Figure 26 (a). However, we started to see more variation in the HWST during the shoulder season months when the outdoor temperatures began to increase, as shown in Figure 26 (b). During these months, the HWST varied the full range between the upper and lower temperature limits we defined. In particular, the HWST increased during occupancy hours when heat pumps' heating setpoints came out of their nighttime setbacks.

[35] https://pypi.org/project/PyFMI
[36] https://github.com/BrickSchema/py-brickschema
[37] https://pypi.org/project/asyncio

Figure 26: Example of Brick enabled ASHRAE Guideline 36 hot water supply temperature setpoint reset strategy during the (a) winter season and (b) during shoulder season when outdoor temperatures are milder.



This Brick enabled G36 control implementation demonstrates that there are no hard-coded parameters that prevent the implementation from being ported over to a new building. The only requirements for the new building are to have a Brick data model and have similar type of equipment to control. The Brick data model allows us to develop general programmatic queries. If we were to represent them in simple language format, the ones we used in this field demonstration would read as “Get me all the hot water end-users for the building’s hot water plant” and “How many of these end-users are requesting hotter water?”. The generic queries allow us to avoid using unique BAS point naming conventions as it is typically done. This field demonstration of Brick enabled controls shows us a path forward where advanced control strategies for building systems can one day be as easy as installing a new application on our mobile phones. (Duarte Roa, paper in progress).

Thus, we used standardized sequences of operation (ASHRAE Guideline 36) to define the best practice controls, an open-source schema (Brick) to represent the required metadata in a structured format, a standardized communications protocol (BACnet) to communicate with the controllers, and (proposed) standardized building controls language (CDL) to implement these controls. These are all non-proprietary resources that are publicly available and can be used in a scalable manner that is *portable between buildings*. That applies even though those buildings

have different automation systems, point names, HVAC topologies, etc. To our knowledge, this is the first time the 'full stack' of these different components has been implemented to perform closed loop control in a real building. The controls continue to operate in the building as writing this report.

Duct Static Pressure Reset Control

The second application developed was a supply air duct static pressure reset control and follows a very similar implementation strategy as the first application.

Variable air volume (VAV) systems are one of the most common types of heating ventilation and air conditioning (HVAC) systems for commercial buildings in North America. The air handling unit (AHU) in a VAV system is typically single duct with an airside economizer, a cooling coil, a supply fan driven by a variable frequency drive, optional heating coil, and either a return or a relief fan. Each individual thermal zone in the building has a VAV terminal unit that measures airflow and controls flow with a damper, and often also a reheat coil. There are many controllable setpoints to manage in a VAV system, from heating and cooling temperature setpoints, discharge temperature setpoints, and minimum airflow setpoints at the zone level, and to minimum outside airflow, supply air temperature and duct static pressure setpoints at the air handling unit (AHU) level.

In early implementations of VAV systems, building operators used constant values for duct static pressure and SAT setpoints. These constant setpoint strategies were improved to become linear resets that increase static pressure and decrease supply temperature with respect to increasing outside air temperature. With the advent of Direct Digital Control with feedback from every zone in the building, demand-based reset approaches are used where static pressure and SAT setpoints vary based on the requirements of the most demanding (''critical') zone, often using 'trim and respond' logic as described in the last section. The duct static pressure should be just high enough so that the most demanding VAV terminal unit in the building (the 'critical' unit) has sufficient pressure to meet its current airflow setpoint. This control strategy is known as a duct static pressure reset and is now part of Guideline 36.

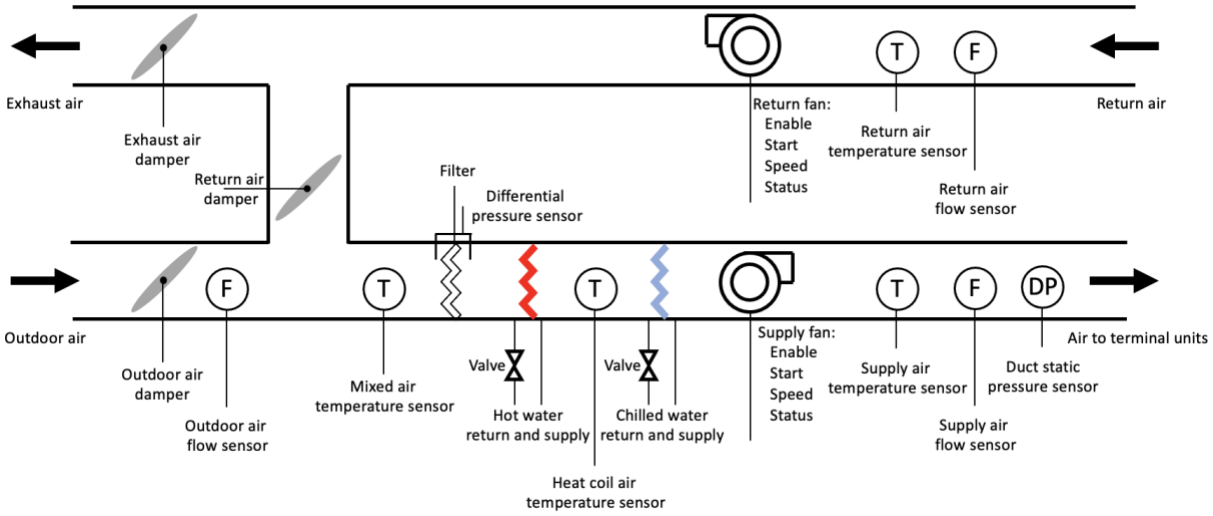
Duct static pressure reset is typically achieved by 'resetting' the duct static pressure setpoint upwards when a zone requests increased pressure (typically when a VAV damper is nearly wide open and the airflow is still below the maximum airflow setpoint) and allowing the setpoint to slowly decrease when there are no requests. This reduces static pressure (and fan power) to the minimum needed to meet the current airflow requirements for all of the zones in the building and can generate fan energy savings from 30-50% compared to fixed duct static pressure setpoints. Typically, there is also a user-defined number of requests that will be ignored, particularly in systems with many zones, as one faulty ('rogue') zone would otherwise drive the entire reset strategy (Raftery 2018[38]).

Sutardja Dai Hall is a seven-floor 141,000-square-foot building on the campus of UC Berkeley. The building houses research labs, faculty offices, a nanofabrication lab, an auditorium, and a café. The building runs one of two 600 ton chillers (absorption using steam and centrifugal using

[38] Raftery, Paul, Shuyang Li, Baihong Jin, Min Ting, Gwelen Paliaga, and Hwakong Cheng. 2018. "Evaluation of a Cost-Responsive Supply Air Temperature Reset Strategy in an Office Building." Energy and Buildings 158 (Supplement C): 356–70. doi:10.1016/j.enbuild.2017.10.017.

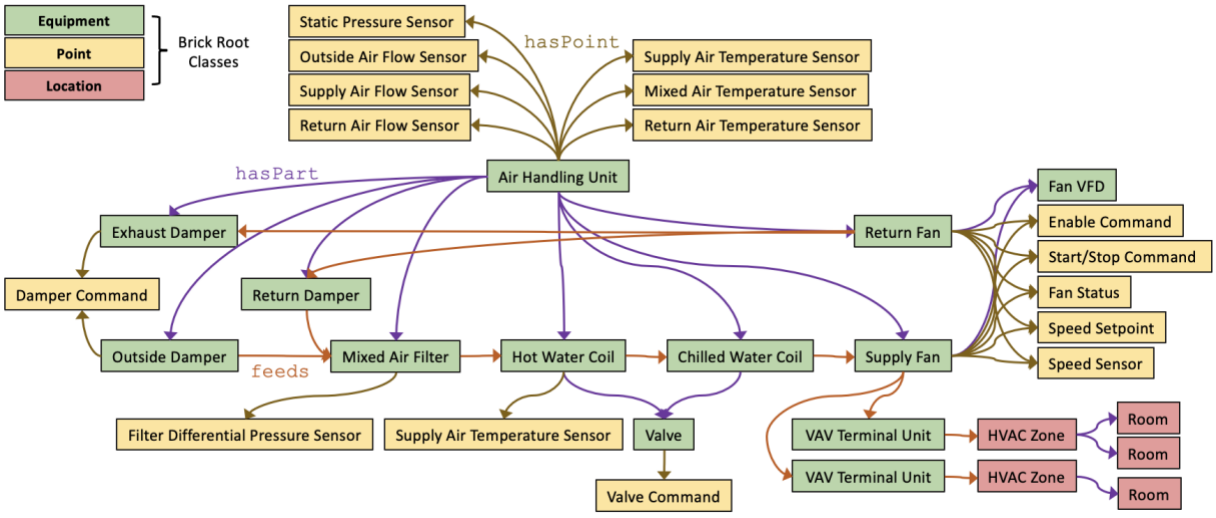
electricity), depending on the time of year; 135 Variable Air Volume (VAV) boxes with reheat supply conditioned air to the office portion of the building. SDH has over 6000 sensing points from the Siemens BAS as well as other additional sensors, such as temperature sensors on the chilled water supply and discharge air temperature sensors at most of the 130 zones. Figure 27 shows a general schematic of the VAV system sensor and control points.

Figure 27: Multi-zone VAV system schematic



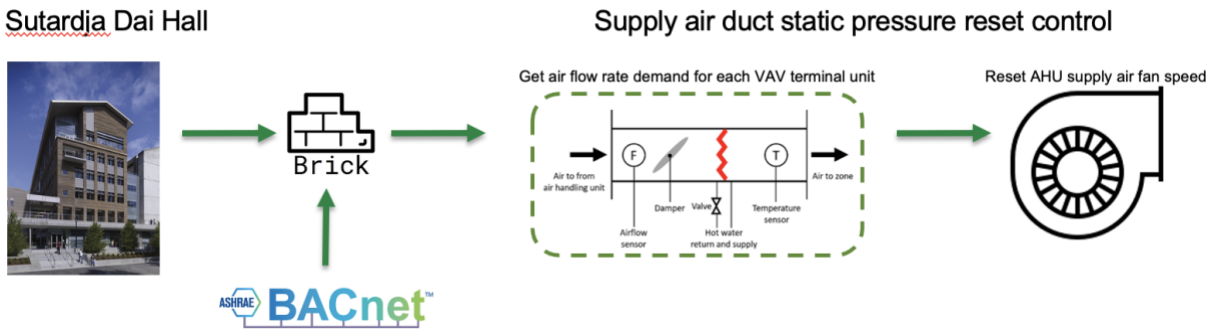
We developed a data model in Brick based on this multi-zone VAV HVAC system, as seen in Figure 28.

Figure 28: Brick data model of a multizone VAV system



We worked with the facilities manager to get access to the system, developed the code and debugged it. Similar to the previous trim-and-respond strategy, this application gets the air flow rate demand for each VAV terminal unit and resets the AHU supply air duct static pressure accordingly (Figure 29).

Figure 29: Duct Static pressure reset control application



Additional Applications in progress

In addition to the two applications described above, several Ph.D. students at UC Berkeley that we advise are taking advantage of the Brick data models that the two demonstration buildings have with BACnet network information embedded to develop advanced control applications. The objective of the first application is to automatically generate suitable thermal models for the building to optimize building operations, forecasting building energy consumption, participate in demand response events or detect faults. The Brick model provides relationships between sensor data points and, as a result, eases the process of mapping data points to the model. The application will use a bottom-up approach to generate a thermal resistor-capacitor (RC) network model. The application will be tested on both the DBC and SDH buildings described above.

The second application in progress has the objective to create a controller in which HVAC system carbon emissions are minimized using model predictive control and thermal storage (e.g., ice storage). The control logic developed would enable HVAC to charge the thermal storage when grid emissions are low and discharge to reduce the building cooling load when emissions are high. The controller will evaluate multiple variables such as HVAC system efficiency, grid marginal carbon emissions, peak power load, and occupant comfort temperature range in multiple U.S. grid regions to find the optimal HVAC control in regard to carbon emissions. The application will be first tested through simulation and if successful plan to test in DBC since it has a high thermal mass HVAC system.

The students developed abstracts on these two Brick applications and submitted to the 2023 IBPSA Building Simulation Conference.

In summary, we have demonstrated several applications and continue to develop applications that take advantage of the Brick schema. We have demonstrated applications that compare the sensor measurements against their setpoints, identify “rogue” behaviors in the air flow rate within an HVAC system, and detect passing valves in HVAC systems. We have developed applications to perform the boiler supply temperature setpoint reset control, assessment of HVAC’s long-term ability to meet thermal comfort needs, and assessment the savings potential of using the designed zone minimum ventilation version the measured zones’ minimum ventilation air flow rate. These applications are all available through the following publicly accessible GitHub repository: https://github.com/SoftwareDefinedBuildings/mortar-analytics.

Brick and energy saving

One of the goals of the project was to demonstrate through simulations using building energy models from the buildings in the Brick model at least 25% energy savings through the use of advanced algorithms.

Previous research studies have shown the importance of standardizing the control sequences for HVAC systems. For example, HVAC VAV system performance can vary as much as 67% when starting from "poor" to "good" control implementations (Pang et al., 2017)[39]. ASHRAE Guideline 36 (G36) aims to reduce this discrepancy and provide standardized HVAC control sequences to maximize energy efficiency and performance, provide control stability, and allow for real-time fault detection and diagnostics. The Brick schema is well positioned to take this effort to the next level by providing the common language to implement G36 sequences in buildings' energy management systems by retrieving and processing the necessary control points and metadata. The combination of these two efforts have the capability of providing consistent energy savings and performance across many buildings. To estimate the potential savings from G36 sequences, Pritoni et. al. (2020)[40] developed a simulation-based energy savings estimator. The researchers calculated over 39% in energy savings when evaluating the supply temperature reset, static pressure reset, and zone air flow rate control with varying zone minimums in a five-zone commercial building. Pritoni et. al. (2020)'s research and savings estimator demonstrate that over 25% is achievable through advanced algorithms.

The control sequences used and simulated in Pritoni et. al. (2020)[41] were developed using Control Description Language (CDL), on top of the Modelica-Buildings[42] library (OBC project) and are available at this public repository: https://github.com/LBNL-ETA/G36SavingsCalculator. As demonstrated in the boiler plant control application, we are now able to implement CDL control sequences on a building with its Brick representation and access to the control system. Additionally, as G36 sequences have already been developed in CDL as part of the Modelica-Buildings library, these advanced algorithms can now be easily ported to other Brick modeled buildings.

[39] Pang, X., Piette, M. A., & Zhou, N. (2017). Characterizing variations in variable air volume system controls. Energy and Buildings, 135, 166–175. https://doi.org/10.1016/j.enbuild.2016.11.031

[40] Pritoni, M., Prakash, A., Blum, D., Zhang, K., Tang, R., Granderson, J., Cheng, H., Engineering, T., & Paliaga, G. (2020). Advanced control sequences and FDD technology. Just shiny objects, or ready for scale? . Lawrence

[41] Berkeley National Laboratory. https://eta-publications.lbl.gov/sites/default/files/advanced_control_sequences_and_fdd_technology_m_pritoni_0.pdf

[42] https://github.com/lbl-srg/modelica-buildings/

Task 5: Technology Transition Plan

The final task that evolved through the three-year project involve taking this technology to Market, through the Industrial Consortium, outreach to stakeholders, presentations, papers, and a workshop.

Industrial Consortium

The initial intention behind the Brick Industrial Consortium was that Johnson Controls was leading this effort to bring a consortium of industry members together, including Microsoft, construction industry members, and so on. This process started a few years ago and has been delayed by negotiations around Intellectual Property (IP) issues. The conclusion was that it is easier for industry to carefully define IP, but at universities this is a more complex process. The solution was to ask university members to join as individuals instead of as institutions.

Feedback from industry partners

The initial thought was that we could interact with the consortium members at related events such as ASHRAE Winter Meeting or Annual Conference in the summer. However, with conferences becoming virtual due to the coronavirus, we created a 10-12 member Technical Advisory Group (TAG) to solicit feedback on applications, use cases, and best ways of "getting Brick out there." The request included the following areas for input:

- The further development of the Brick schema itself,
- Ideas for applications that would highlight how that schema can be used (example list here),
- Ways to demonstrate the benefits of those applications (e.g., by applying them to the Mortar Dataset)
- Contributions of additional datasets (time-series data from automation systems, models of buildings, etc.) that are either private (for our own use in meeting the project goals), or public (and contributed to the next version of the Mortar Dataset).
- Ways to encourage adoption and further development within the broader building industry, and support for the related standard currently under development (ASHRAE 223P).

TAG members include representation from Mechanical-Electrical-Plumbing and Commissioning firms (e.g., Taylor Engineering), Building Retrofits (e.g., Carbon Lighthouse), Controls manufacturers (e.g., Johnson Controls, others from ASHRAE Guideline 36 committee), building owners/managers (e.g., Joel Bender from Cornell), HVAC optimization services (e.g., Comfy/Siemens, KGS), and research organizations (e.g., DOE, NREL, academics).

The following people met in late August 2020:

- Amir Roth, Technology Manager (BEM), DOE
- Stephen Frank, Instrumentation & Controls Engineer, Planning, Management, Engineering, and Construction (PMEC), NREL
- Ambuj Shatdal, Manager Platform Engineering, Johnson Controls Inc

- Nick Gayeski, Partner/Co-founder, KGS Buildings (Clockworks)
- Dezhi Hong, Post-doc, UCSD
- Hongning Wang, Assistant Professor CS, University of Virginia (PI of The Building Advisor)
- Reece Kiriu, Senior Engineer, Taylor Engineering
- Soazig Kaam, WeWork
- Tyler Hoyt, Integrations Team Lead, Comfy
- Joel Bender, Programmer/Analyst, Cornell University (ASHRAE 223P)

We presented the Brick project in the CBE Industry Advisory Board meeting in April 2021 and received overall positive feedback for the need to standardize the description of building data for use in building applications. CBE meeting attendees are enthusiastic to apply and extend Brick and use the Mortar database for their projects. For instance, an HVAC manufacturer is interested in extending Brick concepts so it can apply to their variable refrigerant volume equipment. Another firm wants to leverage the available building data in Mortar for their ongoing project while other firms want to establish a Brick/Mortar like databases to facilitate their current data collection and storage needs they normally perform for commissioning or research studies. However, there were also concerns. One of the prevailing concerns among the attendees is the relationship and standing of Brick among other ontologies, schemas, and other ongoing efforts for building metadata standardization e.g., Project Haystack, ASHRAE 223P, etc. They currently perceive too many players in the semantic data model field and may not embrace a specific semantic data model until they see the clear benefits of one model versus another or one comes out on top. Another concern mentioned was in regards of who would take responsibility for developing a Brick model for new construction e.g., control vendors, architects, or mechanical engineers.

Technology to Market (T2M) Plan

The pathways to market to expand the use and application of Brick include through the Brick Industry Consortium and aligning with DOE-NIST-ASHRAE efforts with the Semantic Interoperability working group developing the 223P standard; barriers include the time/effort to learn to create and develop the Brick model and utilize the query tools and applications to make use of the Brick model.

We see several paths towards increased adoption of Brick, such as through:

- continued support of Brick and Mortar, through the User Forum
- the Brick Industrial Consortium
- participation in the development of the ASHRAE 223P standard.

The Brick Industrial Consortium provides a means for commercial and academic entities to participate in the development of the Brick ontology. Participation can be as direct as contributing documentation, extensions, fixing bugs and developing tools or it can be advisory in the form of voting on future directions for Brick or approving releases of the standard. Industry participation helps ensure that Brick addresses the metadata and data needs of the constituents. Paid membership in the Consortium provides Brick with the resources necessary to advertise and support the ontology, but also indicates to the industry that Brick is a worthy investment. Via the

Brick Consortium's working groups and monthly office hours, we have also learned of several efforts to adopt Brick internally at a variety of companies. In these cases, Brick is replacing ad-hoc or homegrown metadata schemas and providing a more comprehensive set of concepts with which to support data-driven applications. Anecdotally, most of the Brick-enabled use cases are characterized by cloud-based data analytics, data visualization and dashboard creation.

Several members of the project team are actively involved in the development of the ASHRAE 223P standard. As a result, we are well-positioned to ensure the future compatibility between the two complementary standards. We envision Brick as an ontology layer on top of 223P that provides abstractions designed specifically for data analytics and controls applications. In the future, a Brick model could be automatically produced from a 223P model, enabling analytics and controls applications without having to manually curate the metadata. The work in this project on producing a Brick model from varying metadata sources could also be adapted to support the creation of 223P models.

Workshop

Dr. Pieter Pauwels and Dr. Gabe Fierro presented a 90-minute interactive session at the CLIMA 2022 conference (May 22-25, 2022) titled "An Introduction to Semantic Metadata for Data-Driven Buildings". The session was well-attended, with upwards of 30 people in the room. Dr. Pauwels gave an overview of semantic metadata and its relation to existing technologies for digitizing buildings. Dr. Fierro then presented the Brick and Linked Building Data (LBD) ontologies in the context of a real building model and demonstrated common queries against semantic metadata models. Dr. Fierro then demonstrated a basic analysis of the deviations from temperature setpoints in the building, using real building data. Finally, Dr. Fierro and Dr. Pauwels took questions from the audience. Most of the questions concerned how semantic metadata technologies could integrate with BIM, BEM and other digital representations. Many of the attendees seemed enthusiastic about the potential for semantic metadata in facilitating adoption of data-driven practices in buildings, and wanted to know more about how they could adopt and deploy these technologies. Dr. Fierro had several follow-on conversations with researchers and developers at the Czech Technical University in Prague who are working on novel authoring interfaces for Brick models.

The session materials are available online at the following GitHub repository: https://github.com/gtfierro/clima-2022

Session description: https://clima2022.org/programme/scientific/workshops-interactive-sessions/?theme%5B%5D=Digitization

End of project goals

Over the course of the project (2019-2022), Brick released three updates (v1.1, v1.2, and v1.3); Brick 1.0 was released in 2016. Brick 1.1 had an expanded set of classes, provided support for converting from Haystack, and implemented a Python-based ontology compilation system that made it easier for non-ontologists and non-computer scientists to contribute to Brick. Brick v1.2 dramatically expanded the number of concepts and classes covered by the ontology, making Brick more useful for modeling chillers, electrical subsystems, nameplate characteristics, and other static properties. Brick 1.3 added support for some heat pumps, more complete electrical

subsystem modeling, basic safety and security equipment, and PV and battery systems. We have also developed a Brick model validation framework to help verify that users are building models correctly.

Mortar has released one update (v.1.5). Mortar has been released (https://mortardata.org/), with a tutorial developed (https://tutorial.mortardata.org/). A more easily deployable version of Mortar is being developed at https://github.com/gtfierro/mortar, with documentation available at https://beta.mortardata.org/intro.html. The team found a stable, permanent, and performant home for the Mortar testbed dataset so that it is accessible by the community and can be easily maintained by Dr. Fierro as he moved to his new role at Colorado School of Mines in 2021.

The team has developed an alternate, cost-effective solution for the Mortar dataset that can be hosted on major cloud providers or local infrastructure, with the tradeoff of only supporting static datasets. The solution stores timeseries data using Apache Parquet, which is an efficient columnar data format. The dataset is partitioned by building and by I/O point and the resulting structure can be stored on an S3-compatible object store. A simple Python library uses the metadata exposed by the Apache Parquet to scan and download timeseries data matching user queries into DataFrames, CSV files or local Parquet archives. The documentation, client library and supporting tooling for the Parquet-based Mortar dataset is available online at https://github.com/gtfierro/mortar-parquet-support.

The Industry Consortium was finalized, has 6 industrial members, and was involved in the release of Brick v1.3.

The project developed a partnership between RealEstateCore (REC) (https://www.realestatecore.io/) and Brick (https://github.com/RealEstateCore/REC4/tree/main/Ontology/SHACL/Brick) to clarify the semantic relationships between Brick and REC concepts.

The *ref-schema* for interoperability between ontologies, other digital models (https://github.com/gtfierro/ref-schema) has been incorporated into the current draft of the ASHRAE 223P standard.

We maintain an open-source public GitHub repository for Brick applications, found at https://github.com/BrickSchema.

Here one can find packages (py-brickschema: Python package for working with Brick), services (reconciliation-api for OpenRefine and other tools), and other connectors (brick-BACnet). This is the repository for tools and applications, such as Brick versioning, graph inference, conversion from Haystack, VBIS translation, web-based interaction, Brick model validation.

Many applications meant to run on buildings and tested first on the Mortar platform can be found at the UC Berkeley research GitHub site: https://github.com/SoftwareDefinedBuildings/mortar-analytics. These include an application to find simultaneous heating and cooling zones, possible inefficient zones, “rogue” zones, detect passing valves, temperature reset, and so on.

d. Links to publicly available STI are provided

See publications on page 71.

e. Products developed (if applicable) are identified

Brick and Mortar have been described and are available at:

Brick: https://brickschema.org/

Mortar: https://mortardata.org/

f. Computer modeling info (if applicable) is identified

N/A

Products Developed and Technology Transfer Activities

6. Identify products developed under the Award and technology transfer activities, such as:

Publications

Bennani, Imane Lahman, Anand Krishnan Prakash, Marina Zafiris, Lazlo Paul, Carlos Duarte Roa, Paul Raftery, Marco Pritoni, and Gabe Fierro. 2021. Query relaxation for portable brick-based applications. In Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '21). Association for Computing Machinery, New York, NY, USA, 150–159. https://doi.org/10.1145/3486611.3486671

Duarte Roa C., Raftery P., Sun R., Paul L., Prakash A., Pritoni M., Fierro G., Peffer T. (2022). *Towards a Stronger Foundation: Digitizing Commercial Buildings with Brick to Enable Portable Advanced Applications.* ACEEE Summer Study on Energy Efficiency in Buildings 2022. https://doi.org/10.20357/B7ZG6R

Duarte Roa C., Raftery P., Rupam S., Pritoni M., Peffer T. (2022). *Detecting Passing Valves at Scale Across Different Buildings and Systems: A Brick Enabled and Mortar Tested Application.* ACEEE Summer Study on Energy Efficiency in Buildings 2022. https://doi.org/10.20357/B7VP5H

Fierro, Gabe, Jason Koh, Yuvraj Agarwal, Rajesh K. Gupta, and David E. Culler. 2019. Beyond a House of Sticks: Formalizing Metadata Tags with Brick. In The 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '19), November 13–14, 2019, New York, NY, USA. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3360322.3360862

Fierro, Gabe, Sriharsha Guduguntla, David E. Culler. 2019. Dataset: An Open Dataset and Collection Tool for BMS Point Labels. 2nd Workshop on Data Acquisition To Analysis (DATA), New York, NY, USA, November 2019. [pdf]

Fierro, Gabe, Jason Koh, Shreyas Nagare, Xiaolin Zang, Yuvraj Agarwal, Rajesh K. Gupta, and David E. Culler. Formalizing Tag-Based Metadata with the Brick Ontology. Frontiers in Built Environment, Vol 6 (September 2020). DOI: https://doi.org/10.1145/3360322.3360862

Fierro, Gabe, Anand Krishnan Prakash, Cory Mosiman, Marco Pritoni, Paul Raftery, Michael Wetter, David E Culler. Shepherding Metadata Through the Building Lifecycle. In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys 2020). November 18, 2020, Virtual Event. DOI: https://doi.org/10.1145/3408308.3427627

Fierro, Gabe, Anand Krishnan Prakash, Cory Mosiman, Marco Pritoni, Paul Raftery, Michael Wetter, David E Culler. Demo Abstract: Interactive Metadata Integration with Brick. In Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient

Buildings, Cities, and Transportation (BuildSys 2020). November 18, 2020, Virtual Event. DOI: https://doi.org/10.1145/3408308.3431125

Fierro, Gabriel T. 2021. Self-Adapting Software for Cyberphysical Systems. Dissertation, University of California, Berkeley. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-159.pdf.

Fierro, Gabe, Avijit Saha, Tobias Shapinsky, Matthew Steen, Hannah Eslinger. 2022. Application-Driven Creation of Building Metadata Models with Semantic Sufficiency. In Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '22). Association for Computing Machinery, Boston, USA

Fierro, Gabe, Anand Prakash, David Blum, Joel Bender, Erik Paulson, Michael Wetter. 2022. Notes Paper: Enabling Building Application Development with Simulated Digital Twins. In Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '22). Association for Computing Machinery, Boston, USA

Luo, Na, Gabe Fierro, Yapan Liu, Bing Dong, Tianzhen Hong. 2022. Extending the Brick schema to represent metadata of occupants. Automation in Construction, Volume 139, 2022, 104307, ISSN 0926-5805, https://doi.org/10.1016/j.autcon.2022.104307.

Pauwels, P., & Fierro, G. (2022). A Reference Architecture for Data-Driven Smart Buildings Using Brick and LBD Ontologies. *CLIMA 2022 Conference*. https://doi.org/10.34641/clima.2022.425

Pritoni, Marco, Drew Paine, Gabe Fierro, Cory Mosiman, Michael Poplawski, Avijit Saha, Joel Bender and Jessica Granderson. Metadata Schemas and Ontologies for Building Energy Applications: A Critical Review and Use Case Analysis Energies, Volume 14, April 2021.

Roth A., Wetter M., Benne K., Blum D., Chen Y., Fierro G., Pritoni M., Saha A., Vrabie D., (2022). *Towards Digital and Performance-Based Supervisory HVAC Control Delivery.* ACEEE Summer Study on Energy Efficiency in Buildings 2022. https://doi.org/10.20357/B70G62

Sun, Ruiji, Carlos Duarte Roa, Paul Raftery, and Gabe Fierro. 2022. “Enabling Portable and Reproducible Long-Term Thermal Comfort Evaluation with Brick Schema and Mortar Testbed.” In 2022 ASHRAE Annual Conference. Toronto, ON, Canada.

Wetter, M., Hu J., Prakash A., Ehrlich P., Pritoni M, Fierro G., Grahovac M., Rivalin L., Robin D. (2021). Modelica-json: Transforming energy models to digitize the control delivery process. Building Simulation 2021 Conference. (conference in Aug 2021)

Presentations

Duarte, Carlos, Paul Raftery, Ruiji Sun, Lazlo Paul, Anand Prakash, Marco Pritoni, Gabe Fierro, Therese Peffer. 2022. "Towards a Stronger Foundation: Digitizing Commercial Buildings with Brick to Enable Portable Advanced Applications." Presented at the 2022 ACEEE Summer Study on Energy Efficiency in Buildings. Pacific Grove, CA. August.

Duarte, Carlos, Paul Raftery, Ruiji Sun, Therese Peffer. 2022. "Scalable Fault Detection Using the BRICK Schema." Presented at the CBE Industry Advisory Board Meeting. Berkeley, CA. April.

Duarte, Carlos, Ruiji Sun, Paul Raftery, Anand Prakash, Michael Wetter, Karthikeya Devaprasad, Gabe Fierro, Therese Peffer. 2021. "Digitizing Buildings with BRICK to Enable Portable Analytics, Modeling, and Controls." Presented at the ASHRAE Ireland Chapter. Remote Presentation. October.

Duarte, Carlos, Ruiji Sun, Paul Raftery, Anand Prakash, Michael Wetter, Karthikeya Devaprasad, Gabe Fierro, Therese Peffer. 2021. "Digitizing Buildings with BRICK to Enable Portable Analytics, Modeling, and Controls." Presented at the CBE Industry Advisory Board Meeting. Berkeley, CA. October.

Fierro, Gabe. 2021. "Brick and Mortar: Semantic Metadata for Cyberphysical Telemetry and its Context." Presented at Google (virtual). October.

Fierro, Gabe. 2021. "Self-Adapting Data-Driven Software for Buildings." Presented for the Global AI Challenge Conference (virtual). October.

Fierro, Gabe. 2021. "Brick: Consistent Semantic Metadata for Data-Driven Buildings." Presented for IBPSA-USA Building Data Exchange Sub-Committee (virtual).

Fierro, Gabe, Jason Koh, Erik Paulson. 2021. "Catching Up with the Brick Schema for Smart Buildings." Presented as Memoori Webinar. January.

Fierro, Gabe, Karl Hammar, Joel Bender, Erik Paulson, Akshay Johar, Erik Wallin. 2022. "Major harmonization effort between two smart building metadata standards." Presented as Memoori Webinar. August.

Fierro, Gabe, Karl Hammar. 2022. "Getting started using Brick and RealEstateCore: examples and tools." Presented at Brick-RECCon22. https://www.realestatecore.io/reccon22/. November.

Fierro, Gabe. 2022 "A Solid Foundation: Harmonizing Brick and Haystack to Simplify the Building Metadata Landscape." Presented at HaystackConnect 2022. September.

In addition, Dr. Fierro has posted multiple YouTube videos and tutorials about Brick and related topics.

Using Brick and TimescaleDB for building data analytics: https://www.youtube.com/watch?v=kZYNXoiM8gk (852 views)

Converting between Brick and VBIS: https://www.youtube.com/watch?v=zF1M7Z2APSY (75 views)

Dissertation Talk: Self-Adapting Software for Cyber-Physical Systems: https://www.youtube.com/watch?v=Z5OKXIJtvYs (202 views)

Tutorial for online SPARQL query execution: https://www.youtube.com/watch?v=zJTuizwSAks (41 views)

b. Web site or other Internet sites that reflect the results of this project;

Brick: https://brickschema.org/

Mortar: https://mortardata.org/

Brick developer documentation: https://docs.brickschema.org/

c. Networks or collaborations fostered;

Johnson Controls Inc: JCI was a project partner throughout the project and led the creation of the Brick Industrial Consortium. Dr. Fierro (CO School of Mines and NREL) met with Erik Paulson of JCI on a regular basis throughout the project

Industrial Consortium: While the process of creating the Consortium took longer than expected, the Consortium now has 6 industrial members and was involved in the release of Brick v1.3 in 2022. Membership includes Johnson Controls, Siemens, Schneider Electric, Carrier, Mapped, and Clockworks Analytics, in addition to multiple academic members.

Brick collaborates with:

- ASHRAE (especially the 223p BACnet Standard working group):
- Virtual Building Information System (VBIS, https://vbis.com.au/): In Aug 2020, Brick and VBIS announced a Memorandum of Understanding (MOU) to collaborate and integrate. VBIS has a comprehensive classification structure and asset specific metatag for all assets that make up the built environment as well as a mechanism to allow linking asset data that is stored in disparate locations to allow discovery and use of information. The VBIS classification structure, which captures detailed asset properties not covered by Brick, will be mapped to the asset classes in Brick. This will provide a comprehensive data model definition that supports deploying analytics, energy efficiency measures, automation as well as Asset Management and Facility Management activities such as lifecycle planning, maintenance planning and performance benchmarking.

- RealEstateCore (https://www.realestatecore.io/): Brick and REC are cooperating to make it easier to create rich semantic models of smart buildings and real estate portfolios.

As of this writing (2022) Brick is used by:

- Bractlet (https://bractlet.com/)
- Carrier (https://www.carrier.com/carrier/en/worldwide/)
- Johnson Controls (https://www.carrier.com/carrier/en/worldwide/)
- Onboard (https://www.onboarddata.io/)
- Mapped (https://www.mapped.com/)
- Schneider Electric (https://www.se.com/us/en/)

d. Technologies/Techniques;

The Brick data schema includes the ontologies, interfaces and tools, and a testbed (Mortar) of building data described in Brick data models.

Brick: https://brickschema.org/

Mortar: https://mortardata.org/

Brick developer documentation: https://docs.brickschema.org/

e. Inventions/Patent Applications, licensing agreements

Brick is free and open-sourced under the BSD 3-Clause license; Brick is publicly available at https://brickschema.org/resources. The source code for Brick, the website, and related tools developed by the Brick team are available on GitHub.

f. Other products, such as data or databases, physical collections, audio or video, software or netware, models, educational aid or curricula, instruments or equipment

Videos describing Brick:

- Using Brick and TimescaleDB for building data analytics: https://www.youtube.com/watch?v=kZYNXoiM8gk (852 views)
- Converting between Brick and VBIS: https://www.youtube.com/watch?v=zF1M7Z2APSY (75 views)
- Dissertation Talk: Self-Adapting Software for Cyber-Physical Systems: https://www.youtube.com/watch?v=Z5OKXIJtvYs (202 views)
- Tutorial for online SPARQL query execution: https://www.youtube.com/watch?v=zJTuizwSAks (41 views)

Software may be found at:

https://brickschema.org/resources

Brick: https://brickschema.org/

Mortar: https://mortardata.org/

Brick developer documentation: https://docs.brickschema.org/