VizBrick: Visual Brick Model Authoring Tool for Building Metadata Standardization

Sangkeun Lee[a], Borui Cui[b], Mahabir S Bhandari[c] and Piljae Im[c,*]

[a]Computer Science and Mathematics Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN 37830, United States
[b]Electrification and Energy Infrastructures Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN 37830, United States
[c]Buildings and Transportation Science Division, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN 37830, United States

ARTICLE INFO

Keywords:
building data
standardization
ontology
interoperability
automation
smart building

ABSTRACT

The Brick ontology is a unified semantic metadata standard for building assets and their relationships, serving as a key enabler for effective interoperability and automation of building systems and analytics. However, creating a Brick model, in other words, standard semantic metadata based on the Brick ontology for a building dataset, can be a complex task. This paper presents two case studies of the creation of Brick models for real-world residential and commercial building datasets, highlighting the challenges during the Brick model creation process. Additionally, the paper introduces VizBrick, an interactive authoring tool for creating semantic building metadata. VizBrick facilitates the creation of Brick models by providing an intuitive visual interface and interactive capabilities, such as keyword search, automatic mapping suggestions, and recommendations. The use of VizBrick is shown to significantly reduce the time and effort required during the Brick model creation process.

Acknowledgment

Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/doe-public-access-plan).

1. Introduction

Data collected from buildings and smart buildings are crucial assets for research aimed at enhancing human interactions in building environments. One of the major challenges in managing building data is its lack of standardization, which often results from the data being collected from various sources, such as research organizations, building managers and researchers. This lack of standardization undermines the interoperability and reuse of data assets, leading to redundant data, and significantly increasing the cost of time and labor.

Building data refers to all information related to a building's characteristics, operations, environment, performance, and more. Examples include, but are not limited to, records collected from sensors and devices associated with buildings, such as temperature and power sensors, or data detailing the physical characteristics of the building. When building data is collected and shared, publishers often provide metadata, which describes the data in question. Metadata is "data about data" that includes information such as the source and type of sensor, unit of measurement, frequency of data collection, and the significance of empty values, in the context of building data assets.

The simplest solution for enabling interoperability among different datasets is to use a common standardized building data schema, but this approach might not be practical. Defining a universal schema that covers all possible use cases is unrealistic, and converting existing datasets to another schema can be time consuming and complex. A more practical solution is to standardize the *metadata*, providing a machine-interpretable, common, and agreed-upon ontology of extensible semantic concepts and relations to describe building datasets. This can potentially facilitate interoperability and reuse of different building datasets.

To standardize metadata for building datasets, various unified metadata models have been developed [5, 7, 22, 34]. In this paper, we focus specifically on the Brick ontology [1], which is an open-source, state-of-the-art metadata schema for building metadata standardization. Brick is an emerging and increasingly popular open-source ontology-based unified metadata schema for building data that is supported by academia, industry, and federal agencies [27]. Brick is an ontology-based metadata schema that encapsulates the essential entities and relationships necessary for accurately representing buildings and their associated subsystems. It can semantically describe physical, logical, and virtual assets, covering concepts and relations. Brick's foundation on the World Wide Web Consortium (W3C)'s semantic ontology standards, such as OWL (Web Ontology Language)[28] and RDF (Resource Description Framework)[24], signifies a structured and standardized approach to representing knowledge about buildings and their subsystems. OWL allows for the definition of complex relationships, classes, and properties, thus enabling richer descriptions and classifications of

*Corresponding author
lees4@ornl.gov (S. Lee); cuib@ornl.gov (B. Cui); bhandarims@ornl.gov (M.S. Bhandari); imp1@ornl.gov (P. Im)
ORCID(s): 0000-0002-1317-5112 (S. Lee)

data. RDF, conversely, provides a standard model for data interchange and forms the backbone for linking, sharing, and analyzing data across various applications from cyber-physical systems and building energy use to enhancing human interactions in built environments[4]. Creating models with Brick for building datasets involves describing the dataset's contents using the standard terms defined in the Brick ontology.

The benefits of data standardization can be fully realized as more building datasets are converted to semantic metadata using the Brick ontology. For example, when working with datasets acquired from smart buildings, use of the Brick ontology allows us to describe the devices responsible for generating each column of the dataset. We can describe the locations of these devices within the buildings, the structural characteristics of the building, and the relationships between these assets. Leveraging the Brick ontology, we can create precise and comprehensive models that encompass the intricate details of a building and its associated datasets. Furthermore, as these descriptions are based on the standard terms, Brick ensures interoperability and facilitates seamless integration across different systems and platforms.

However, there is a major obstacle to overcome: the process of creating Brick models for building datasets based on the Brick ontology can be overwhelming for many data producers and owners. The process requires in-depth knowledge of building assets and semantic technology (e.g., ontology declarations and Resource Description Framework (RDF) syntax [8]), and it relies heavily on repeated manual trial and error steps. To address this issue, this work presents case studies of creating Brick models with real-world building datasets such as the NIST Net-Zero Energy dataset (NZERTF) and the Flexible Research Platform (FRP) dataset. We extracted the workflow for creating the Brick model from the analysis of two case studies and identified the challenges associated with Brick model creation. Based on these findings, we developed VizBrick, an interactive and visual Brick model [1] authoring tool with a graphical user interface, which helps mitigate the challenges of the Brick model creation process and saves time and effort.

The rest of this paper is organized as follows: In Section 2, we provide an overview of related work. Section 3 presents the case studies of creating a Brick model with real-world building datasets for metadata standardization. In Section 4, we summarize the common difficulties encountered during the Brick model creation process. In Section 5, we introduce VizBrick, our interactive Brick model creation tool, which addresses the difficulties identified in Section 4. Finally, in Section 6, we conclude our study and outline future work.

2. Related Work

2.1. Building metadata standards and ontologies

In smart building applications, standardized building metadata, including metadata from sensors and devices attached to the building, enables more intelligent automation. For instance, real-time data from standardized sensors can be crucial for optimizing energy efficiency and for quickly identifying and responding to threats or emergencies. Furthermore, in the age of the Internet of Things (IoT), where devices are constantly communicating, standardization serves as the foundation supporting seamless integration, interoperability, and future innovations in building technology.

There have been several approaches to standardizing metadata for building datasets. Industry Foundation Classes (IFC) [5] is a standardized Building Information Model that developed from the need to have a common exchange model for 3D architectural drawings with associated information. Although it is one of the most accepted standards, as it represents a comprehensive set of classes and relationships that define the physical and functional characteristics of building components throughout their life cycle, it does not offer explicit mechanisms for specifying the functional role of sensors and equipment.

Project Haystack [22] is an industry-driven project that aims to create a metadata schema that uses tags to represent various objects such as meters, HVAC components, and lighting. It provides a predefined tag dictionary for annotating sites, equipment, points, and their relations and allows for querying through an API. Additionally, it provides the option to use custom tags for an extension.

Ontology-based approaches based on semantic web technology are gaining attention. There have been efforts to create an OWL representation of the IFC model, namely ifcOWL [2] [28]. Haystack Tagging Ontology [6] is an effort to create a semantic technology wrapper for Haystack, while Building Topology Ontology (BOT) [34] focuses on the topological concepts of buildings such as sites, floors, zones, and rooms. Google's Digital Building Ontology [3] aims to create a unified schema for buildings, and Smart Appliances REFerence Ontology (SAREF) [7] focuses on smart appliances but does not cover the full spectrum of equipment and sensors. One of the primary reasons for this shift towards ontology-based approaches for standardization is that they provide a structured framework and common vocabulary for defining concepts, relationships, and attributes within a specific domain, which can be understood and processed by machines and humans.

2.2. Visual RDF editors

General-purpose visual RDF editors, such as OntoStudio [40], Protégé [30], RMLEditor [18], and TurtleEditor [32], can be valuable for creating ontologies or ontology-based models. These tools offer a wide range of features, including reasoning capabilities and query processing, so they can be used to create Brick models. Note, however, that they are not specifically tailored for building domains. They do not provide mechanisms to leverage existing building metadata or generate automatic suggestions based on the building-specific information embedded in the Brick ontology. In contrast, VizBrick was designed to address these limitations by offering a Brick model construction tool that does not require extensive programming knowledge or expertise in semantic technology. In Section 5, we will delve

into VizBrick and discuss its advantages over existing tools in more detail.

2.3. Brick ontology

Brick [1] provides a comprehensive and extensible vocabulary of building assets that can semantically describe physical, logical, and virtual assets (e.g., sensors, locations, devices, and so on), covering concepts and relations. Its coverage of building concepts makes it a clear choice for representing building components and their relations more effectively compared with other models. Building assets and subsystems can be described using Brick using a standardized, consistent framework, which can be seamlessly integrated with existing tools and databases. Various systems, such as HVAC, lighting, fire, and security, can collaborate and share data with one another. There are several advantages to using the Brick model over other approaches for managing and analyzing building data. Brick promotes greater consistency and interpretability, setting itself apart from systems like Project Haystack [22]. Industry Foundation Classes [5] focus primarily on structural information, while Brick captures functional relations between assets. The compatibility of Brick with other semantic Web-based ontologies, such as BOT [34], SAREF [7], and ifcOWL [2], allows for a more comprehensive representation and integration of building data. As an open-source solution under the BSD 3-Clause license, Brick encourages innovation, collaboration, and accessibility in the realm of building data management. Figure 1 depicts an example of a Brick model designed for a building dataset. The figure provides insights into the equipment, locations, and points related to the dataset, and more importantly, it illustrates the relations between these elements.

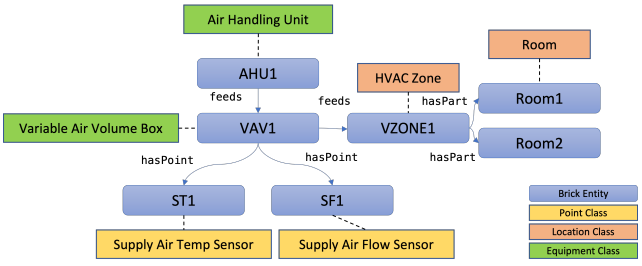


Figure 1: Visual representation of a Brick model created for an example building dataset. Dashed lines depict "is-a" relations, different colors represent different classes, and each block represents a class instance.

Conventional metadata, such as text files, word files, or natural language description embedded in comma separated values (CSV), is meant to be interpreted by humans, which makes it challenging for machines, such as software code, to process them. However, intelligent controls, analytics, and smart systems can leverage the Brick model, as it adopts standard syntax and preestablished semantic concepts and relations. Additionally, Brick can be used in conjunction with other ontology-based approaches.

Creating a Brick model means formulating standard building metadata based on the Brick ontology. The author of the Brick model needs to have not only a thorough understanding of the target building or building dataset but also knowledge of semantic technology and its tools, the Brick ontology, and how to represent a Brick model using the correct RDF representation syntax. This technical requirement can present a significant challenge for building data producers and owners. To address this issue, several tools have been developed to facilitate the creation of Brick models, such as Brickify [17], py-Brickschema [14], and BrickBuilder [13], which allow users to define transformation rules from tabular metadata sources, such as spreadsheets and CSV files, to Brick models or translation from other models to Brick. However, these tools have limitations in that they are text based and do not provide visual interactions during the model creation process.

3. Case Studies

3.1. Dataset overview

To understand the process and challenges of creating a Brick model for building datasets, we selected two building datasets for our case studies. These datasets were collected from two different buildings and were maintained by the data owners. Data owners already authored metadata for these two datasets in different formats, more specifically using the Word document and a CSV file; however, Brick models for these buildings did not exist. The goal of the case study is to create Brick models, standardized semantic metadata defined based on the Brick ontology, for these two datasets and identify the challenges during the process.

The first dataset was captured from the NIST Engineering Lab's Net-Zero Energy Residential Test Facility (NZERTF) [9] (see Figure 2), which is a residential home located in Gaithersburg, Maryland, USA. The dataset consists of diverse, high-resolution, high-precision measurements. The house has a living area of 252 m^2 and a basement floor area of about 135 m^2 [36]. The facility was intended to be built as a standard home for a family of four, with the goal of showcasing its ability to achieve an annual site energy usage of net-zero. [12]. This was to be accomplished through a combination of low energy loads resulting from a high-performance enclosure, efficient mechanical systems, and low-energy fixtures and appliances, in combination with site-generated energy using roof-mounted solar panels. This dataset was collected for researchers to study the performance of efficient homes. Data from 11 subsystems is collected across 379 channels (a channel is a data stream from a single sensor).

The second dataset, the Flexible Research Platform (FRP) dataset [20], was gathered from a two-story multizone unoccupied building that simulates common light commercial buildings found in the United States. The two-story FRP, which consists of slabs and a steel superstructure and covers an area of 179.56 m^2, is a typical example of the light commercial buildings that are commonly found in the country's current building inventory (see Figure 3). An FRP simulates occupancy by controlling lighting, humidifiers for human-based latent loads, and a heater for miscellaneous electri-

Figure 2: Net-Zero Energy Residential Test Facility

Figure 3: Flexible Research Platform building

cal loads. This dataset contains various information from sensors and HVAC systems, as well as system controls, occupant schedules, and plug loads. The validation tests for the multizone HVAC system used a 44 kW roof-top unit (RTU) and a natural gas furnace, with each room conditioned through a Variable Air Volume (VAV) box with electric resistance reheat. The Johnson Controls Metasys system was used for predefined control of room temperature and schedule. Various sensors were deployed to monitor temperature, humidity, flow rates, and energy consumption, with data available at 1 min, 15 min, and 60 min intervals. Additionally, a weather station on the roof of the FRP monitored outdoor weather data such as temperature, humidity, solar radiation, and wind speed and direction.

They were chosen to showcase how the Brick model can be applied to a broad range of buildings, from residential to commercial structures. HVAC systems in these buildings are commonly used in the United States. This makes them a suitable choice for demonstrating the versatility of the Brick model across a broad range of buildings. Moreover, the sensor/meter data collected in both buildings is consistent with data collected in many buildings, making it a valuable and relatable set of data for the Brick model creation case study.

3.2. Reviewing the existing metadata

The first step in creating a Brick model for the dataset was to thoroughly review existing metadata documents. These documents were prepared by the owners of the FRP and NZERTF datasets and describe the contents of these datasets. For the NZERTF dataset, the information was described in a Word document that detailed the data dictionary and its contents. The document was structured into two parts, starting with a taxonomy of building characteristics. The early sections discussed information about the building such as its location, size, number of floors, rooms, lighting, heating, cooling systems, and other relevant properties. The latter part of the document provided an in-depth description of each available data channel in the NZERTF dataset. To streamline the description, predefined keywords, referred to as *Keys*, were used to represent specific building-related objects within the document. Table 1 illustrates examples of the *Keys* used to describe the building in the document.

Then, the document presents a list of data channels within the NZERTF dataset, including their data label, subsystem, measurement location, measured parameter, unit, and natural language description. Examples of these data channels can be seen in Table 2. A total of 379 data channels are in the document. As noted, the *Keys* defined in Table 1 are used repeatedly in the names and descriptions of the data channels. The metadata document provides valuable information about the contents of the NZERTF dataset. Definitions of terms like *Keys* and concepts are applicable only within the NZERTF dataset and might have different meanings in a data dictionary for a different dataset, which could limit interoperability. The metadata document uses a consistent pattern; however, there are limitations. The document relies heavily on natural language descriptions to reduce ambiguity, so parsing and interpreting the document in computer code would require handling numerous exceptions and corner cases, and the code would not be compatible with other datasets.

On the other hand, the FRP dataset is described in a CSV file, which has a tabular format. The header of the table and the descriptions for each column are as follows:

- **Header**: Name of the column.
- **Units**: Unit of data (e.g., volt, Celsius degree, percentage, etc.)
- **Description**: Description of the column explained in natural language.
- **Location**: Location where the data was collected.
- **Notes**: Additional note for the column.
- **Conditional**: Conditions of the data collection for the column.

Table 3 presents a portion of the metadata table for the FRP dataset, which has a total of 247 rows in the actual CSV file. The column names in the table exhibit some patterns, such as the "T_VAV_202_Avg" column name indicating average temperature (implied by "Avg") collected from a VAV in room 203 (implied by "T_VAV_202"). However, not

Type	Key	Description
Subsystem Key	DHW	Domestic hot water
	SHW	Solar hot water
	Loads	Electrical and thermal loads by equipment and people
	HVAC	Heating and cooling system
	IndEnv	Indoor environment
Measurement Location Key	Bath1	First-floor bathroom
	Bath2	Hallway bathroom located on 2nd floor
	Bedroom2	Second-floor bedroom, west side of house
	Bedroom3	Second-floor bedroom, east side of house
	Bedroom4	First-floor bedroom/office
Measured Parameter Key	Current	Electric current from photovoltaic system
	Flow	Flow of water, glycol, or air; either total amount or a rate
	Frequency	Frequency of alternating current on circuits from the photovoltaic system
	Temp	Temperature measurements
	Energy	Accumulated energy consumption or delivery, either electrical or thermal

Table 1
The table presents examples of repeatedly used keywords, namely *Keys*, to describe the data, along with their types (categories) and descriptions. For instance, the "Measurement Location Key" represents a category of keywords that indicates the location where the data is collected. As an example, if a data label contains the keyword 'Bath1', it signifies that the data originates from the first-floor bathroom.

Data Label	Subsystem	Measurement Location	Measured Parameter	Description	Units
DHW_ClothesWasherColdFlow	DHW	Utility	Flow_ Water	The cumulative volume of cold water flowing into the clothes washer starting at midnight	Gallons
DHW_ClothesWasherHotFlow	DHW	Utility	Flow_ Water	The cumulative volume of hot water flowing into the clothes washer starting at midnight	Gallons
DHW_DishwasherHotFlow	DHW	Kitchen	Flow_ Water	The cumulative volume of hot water flowing into the dishwasher starting at midnight	Gallons
DHW_SHWGlycolTempIn	DHW	Basement	Temp_ Glycol	The instantaneous temperature of the glycol-water solution on the inlet side of the heat exchanger of the solar water heating system (from the solar panels)	°C
DHW_SHWGlycolTempOut	DHW	Basement	Temp_ Glycol	The instantaneous temperature of the glycol-water solution on the outlet side of the heat exchanger of the solar water heating system (returning to the solar panels)	°C

Table 2
Excerpt of data channel descriptions in the NZERTF metadata document. From the keys used in data labels, we can infer the information about the data label. For instance, from the data label 'DHW_SHWGlycolTempIn', we can infer that this data label pertains to Domestic Hot Water (DHW), Solar Hot Water (SHW), and Temperature (Temp) since the label incorporates these terms.

Header	Units	Description	Location	Notes	Conditional
TIMESTAMP		Date and Time stamp from datalogger			Average
RECORD		Sequential datalogger record number			Average
Batt_volt_Avg	Volt	Logger Battery Voltage			Average if 124. HVAC_Power(2) or 126.HVAC_Power(4) >100
Panel_Temp_Avg	°F	Logger Panel Temperature			Average
T_Air_Up_Avg(1)	°F	Room 201 Hanging	Room 201 Hanging	(There is currently no probe installed in the stairwell so this is a garbage value)	Average if 124. HVAC_Power(2) or 126.HVAC_Power(4) >100
T_Air_Up_Avg(2)	°F	Room 201 VAV	Room 201 VAV	(There is currently no VAV in the stairwell so this is a garbage value)	Average
T_Room_202_Avg	°F	Room 202 Room Ambient	Room 202 Room Ambient	Average if 124. HVAC_Power(2) or 126.HVAC_Power(4) >100	
T_VAV_202_Avg	°F	Room 202 VAV	Room 202 VAV		Average
T_Room_203_Avg	°F	Room 203 Room Ambient	Room 203 Room Ambient	Average if 124. HVAC_Power(2) or 126.HVAC_Power(4) >100	
T_VAV_203_Avg	°F	Room 203 VAV	Room 203 VAV		Average

Table 3
Excerpt of the FRP metadata CSV file

all column names follow this pattern consistently. For example, the "Panel_Temp_Avg" column name also indicates average temperature but does not start with "T," as many other temperature data columns do. Furthermore, not all column names contain location information, as seen in the "T_Air_Up_Avg" column name. In such cases, the description column provides additional information. This inconsistent pattern and reliance on natural language descriptions make it difficult to automate the process and result in the need for manual efforts when developing the Brick model.

Note that the two buildings used in our case studies are research-grade buildings, which means that their metadata files are relatively thorough and complete. However, this level of detail might not be typical for metadata files of regular buildings in the real world. In many cases, building datasets lack any metadata files that describe the datasets. Although creating these metadata files is not necessary for Brick modeling, in such cases it can be helpful to create a list of data columns in a dataset along with their simple descriptions. This list can serve as a useful reference document during the Brick modeling process.

In summary, both datasets are described in a tabular format with the information presented in either a Word document or a CSV file. Although their structures are not identical, they share similarities in listing data points, building-related entities, and their descriptions, such as location, units, and so on. The metadata is written primarily for humans and relies heavily on natural language text. Both tables exhibit patterns in the repeated use of keywords in the data label names or descriptions, but some are inconsistent and ad hoc throughout the data.

3.3. Creating Brick models

In this section, we describe the process of creating Brick models for the NZERTF and FRP datasets and summarize the challenges encountered during the process.

3.3.1. Describing the physical building structures

Brick models can be created in any order, but we started by creating Brick class instances that describe the physical building structure of our interest. Instances of Brick ontology classes are identified by uniform resource identifiers (URIs). RDF namespace prefixes such as *brick*, *nzertf*, and *frp* were used to represent long and complex URIs in RDF data. These prefixes make RDF statements more concise and readable.

Initially, we focused on instances of the Brick *Location* class instances and their relations, which define the overall structure of the physical building from which the datasets are generated. Brick classes are hierarchical. *Location* class has multiple subclasses such as *Space*, *Floor*, and *Zone*, and these subclasses can have their own subclasses, for instance, the *Space* class has subclasses such as *Room*, *Common Space*, and others. For example, the *Location* class has multiple subclasses such as *Space*, *Floor*, and *Zone*. These subclasses can further have their own subclasses. For instance, the *Space* class has subclasses like *Room*, *Common Space*, and others. We need to represent the building's physical structure using instances of these *Location* classes. We then use these instances to describe the locations from which data is collected. For example, if a sensor generates data from a room, we create instances of the *Sensor* class and the *Location* class representing the room, and establish a "hasLocation" relationship between these instances.

In the case of the NZERTF dataset, we could obtain information for modeling *Location* instances from its metadata documents, specifically, the list of *Measurement Location Key* in Table 1 and Measurement Location column from Table 2. To describe the locations, we needed to identify the most closely matching classes from the Brick ontology.

It was helpful to use the search toolbox provided on the Brick model website (http://brickschema.org/). Whenever a specific subclass could not be found in the Brick ontology schema, the most specific class in the upper hierarchy was chosen. For instance, in the case of *nzertf:Entry_Hallway*, we selected *brick:Space* class, as the Brick ontology did not have a class for a hallway or entry hallway. We also created *Location* instances that did not directly show up in Table 1 and Table 2 but were inferred to be useful to describe the building. *nzertf:First_Floor* and *nzertf:Second_Floor* were created to describe in which floor rooms are located. The following shows an excerpt of the triples that represent the defined *Location* instances in Terse RDF Triple Language (Turtle), a syntax used to encode data in the RDF data model [24]. The Turtle syntax expresses RDF data, including Subject, Predicate, and Object (SPO) triples, which are the foundation of the RDF data model. An SPO triple in Turtle format is represented as follows:

```
<Subject> <Predicate> <Object>
```

Each triple consists of a subject, a predicate, and an object. The subject is the resource (an entity that the statement is about), the predicate is the characteristic or aspect of the subject that the statement specifies, and the object is the value of this characteristic. The following shows part of a Brick model represented in the Turtle syntax.

Listing 1: Examples of RDF triples in the Turtle syntax

```
nzertf:Living_Room a brick:Room .
nzertf:Entry_Hallway a brick:Space .
nzertf:First_Floor a brick:Floor .
nzertf:Basement a brick:Basement .
```

The token "a" in the predicate position (second position) of a Turtle triple represents the Internationalized Resource Identifier "*http://www.w3.org/1999/02/22-rdf-syntax-ns#type*," which defines a type of an instance. So the first line describes that an instance defined for the NZERTF dataset is of type *Room* class, which is defined in the Brick model as a specific kind of location.

Once the location instances were defined, we created relations between the defined location instances. The following shows how we represented which rooms were located on which floor.

Listing 2: Excerpt of Location instances and their relationships (NZERTF)

```
nzertf:First_Floor brick:hasPart nzertf:Living_Room.
nzertf:Second_Floor brick:hasPart nzertf:Bedroom_2.
```

To describe how a location is part of another, we used the predefined relation *brick:hasPart* in the Brick ontology. These relation instances were inferred from the natural language description of the locations.

Overall, the workflow for the FRP dataset was similar. The metadata CSV file indicates that there are 12 rooms on 2 floors in the building of interest. The room number indicates which floor the room is located on, and each room is identified with a number. We defined *Location* instances and created relations between them as follows.

Listing 3: Excerpt of Location instances and their relationships (FRP)

```
frp:Room_101 a brick:Room .
frp:Upstairs a brick:Floor .
frp:Upstairs brick:hasPart frp:Room_201 .
frp:Downstairs brick:hasPart frp:Room_101 .
```

3.3.2. Describing the other building features

Next, for the NZERTF dataset we created *zone* and *equipment* instances to provide further details about the building. Controlled spaces can be grouped as physical locations within the building. As per the metadata document, we created two zone instances: (1) *nzertf:Conditioned_Places* and (2) *nzertf: Hot_Water_Supply_Zone* and mapped the physical locations to these defined zones.

Similarly, we created *equipment* instances to provide services to the defined zones and established relations between them. For instance, we described the presence of a solar system with a solar collector that supplies a hot water pump and a solar storage tank, which in turn feeds the hot water system and ultimately the *Hot Water Supply Zone*.

For the FRP dataset metadata, some of the building equipment is located outside. Additionally, some data points pertain to the entire building, not just a specific area. To handle such cases, we defined instances as follows.

Listing 4: Examples of building and outdoor class instance definitions

```
frp:Building a brick:Building .
frp:Outdoor a brick:Outdoor_Area .
```

We noticed that similar groups of data are collected from each room, so we defined zones for each room. Each zone is regulated by a VAV box, which controls the volume and temperature of the air delivered to the zone. We can say that a room is part of its corresponding VAV zone and that each VAV box serves its corresponding room. We also defined meters for the rooms and VAV boxes. Room 101 was an exception, as no probes were installed according to the description in the metadata.

Listing 5: Examples of VAV box and related sensor definitions

```
frp:Room_102_VAV_Zone a brick:Zone .
frp:Room_102_VAV a brick:VAV .
frp:Room_102_VAV brick:feeds frp:Room_102_VAV_Zone .
frp:Room_102_VAV_Zone brick:hasPart frp:Room_102 .
bldg:RTU a brick:RTU .
bldg:RTU brick:hasPoint bldg:RH_Ret_Avg .
bldg:RTU brick:hasPoint bldg:RH_RetDuctMain1_Avg .
bldg:RTU brick:hasPoint bldg:RH_RetDuctMain2_Avg .
bldg:RTU brick:hasPoint bldg:RH_Sup_dwn_Avg .
bldg:RTU brick:hasPoint bldg:RH_Sup_tot_Avg .
```

We inferred the existence of an RTU in the building from column names such as T_RTUintake_Avg, Tref_RTU_Dischg1_Avg, and so on. An RTU is a packaged air conditioner mounted on a roof that discharges conditioned air directly into the rooms below or through a duct system. It is composed of several parts, such as condensers, compressors, gas meters, fans, economizers, and so on.

These subcomponents were inferred from columns such as Tref_RTU_CondOut1_Avg (Cond implying *Condenser*) and WH_RTU_Comp1_Tot (Comp1 implying *Compressor*). External domain knowledge was required to clarify the type of equipment.

For example, WH_RTU_EvapFan_Tot is the total power measured from a supply fan, but this is not immediately apparent from the column name. The following shows how we described the RTU equipment and its subcomponents in our FRP Brick model.

As demonstrated by the example, the level of detail that can be inferred about a building from a dataset largely depends on the accuracy and completeness of the metadata describing the building. Inaccurate or incomplete metadata can result in errors and inconsistencies in the resulting Brick model, potentially leading to incorrect conclusions about the building's performance. Furthermore, creating a Brick model is a challenging task that requires significant expertise and technical knowledge. The person constructing the model must possess a deep understanding of the domain, the building, and semantic technologies such as RDF syntax and ontologies.

Identifying the most suitable Brick class for each instance or element and accurately describing the relationships between instances in the building are critical tasks in creating a Brick model. This process requires careful consideration of multiple possible options and often results in multiple revisions during the creation process. The creation of Brick models is not a one-time process but rather an ongoing effort that requires continuous updates and revisions as the building evolves over time. This highlights the importance of tools that can facilitate Brick creation practices and workflows to ensure the accuracy and completeness of building data without increasing complexity or requiring additional knowledge.

3.3.3. Describing the data points

The next step was to create instances for the data points, where measurements were taken. For the NZERTF dataset, we had to translate the full list of data channels (as seen in Table 2 in the metadata document). Similar to finding the

If data channel name contains	Corresponding entity is mapped to the Brick class
DewpointSupplyAir	*brick:Supply_Air_Dewpoint_Sensor*
DewpointReturnAir	*brick:Return_Air_Dewpoint_Sensor*
Air and Temp	*brick:Air_Temperature_Sensor*
ColdFlow or HotFlow	*brick:Water_Usage_Sensor*
Flow	*brick:Flow_Sensor*
flowrate	*brick:Air_Flow_Sensor*
Airflow	*brick:Air_Flow_Sensor*
EnergyTotal or EnergyPlugs or Energy or EnergywithStandby or PowerTotal	*brick:Energy_Usage_Sensor*
PowerwithStandby	*brick:Electrical_Power_Sensor*
PowerTotal	*brick:Energy_Usage_Sensor*
Power and Plug	*brick:Power_Sensor*
Power and not Plug	*brick:Thermal_Power_Sensor*

Table 4
Example rules for Brick class matching to *Point* instances for NZERTF dataset. For instance, the first row in the table represents a mapping rule where a data channel is mapped to the Brick class "$brick : Supply\_Air\_Dewpoint\_Sensor$" if the name contains the term "DewPointSupplyAir."

closest matching Brick classes for *Location* and *Equipment* instances, our aim was to identify the closest matching Brick *Point* class for each data channel in Table 2. However, the challenge was that unlike the *Location* and *Equipment* cases, we had numerous data channels (over 380 in this instance), making manual matching an arduous task.

To overcome this challenge, we used patterns in the data channel names, making use of the *Keys* in Table 1 to define the rules. We manually created such rules through multiple iterations of trial and error, as seen in Table 3. For example, we created a *Point* type entity named "ClothesWasherColdFlow" (as seen in the first row of Table 2) and mapped it to the "brick:Water_Usage_Sensor" class based on the rule we defined.

Defining the rules in Table 4 was a complex and demanding process, as it relied heavily on trial and error. The implementation of a script that would apply these rules and create RDF triples required significant effort. We accomplished this by writing a Python script that iterates over each row in Table 4 and applies the rules listed in the table, starting from the top and continuing until a match is found. Additionally, information about the location of each data label measurement can be obtained from Table 4 and translated into a relation between the created *Point* entity and *Location* entity in the final Brick Turtle file.

Similar to the approach taken for the NZERTF dataset, we also took a data-specific approach for the FRP dataset. This meant that the rules and the script developed for the NZERTF dataset could not be reused. To create the *Point* class instances for the FRP dataset, we went through each row in the CSV file and identified the most relevant Brick entity for each column name. In many cases, we could determine the appropriate Brick subclass for the *Point* class by relying on the string patterns in the column name. However, care was taken to handle exceptions such as typos in the metadata or corner cases.

Similar to the previous case, we implemented a Python script to apply the rules defined in Table 5 for the FRP dataset. The script created instances for each data point, identified the relevant Brick class for each column, and created relations between *Location*, *Point*, *Zone*, and *Equipment* class instances. The script iterated through each room and established relations between temperature and power sensors and their respective VAV boxes and meters. Relations were also established for other locations such as *Outdoor*, *Building*, *Downstairs*, and *Upstairs*.

We created relations between data points and RTUs or their subparts based on the column name or description from the metadata, but this required careful consideration of exceptions. For instance, columns such as Gas_SCF_Tot and T_ReturnMix1_Avg did not have RTU in their name or description, but our domain knowledge indicated that they should be associated with the RTU.

In both cases, defining rules and implementing the necessary scripts to apply those rules were ad hoc processes. We encountered corner cases and typos in both cases, underscoring the importance of carefully defining rules to address these challenges. The dataset and metadata column names and descriptions were not originally intended to be processed by software code, but rather to be read and interpreted by humans. As a result, the implemented scripts were data specific and could not be used for other datasets. This experience highlights the need for a tool that allows users to easily define and test various rules for matching building assets to Brick ontology classes. In Section 5.2.2, we discuss how the VizBrick tool can address this need.

If data channel name contains	Corresponding entity is mapped to the Brick class
T_Room	*brick:Zone_Air_Temperature_Sensor*
T_VAV	*brick:Discharge_Air_Temperature_Sensor*
*RH_*	*brick:Zone_Air_Humidity_Sensor or brick:Discharge_Air_Humidity_Sensor*
*Pref_ or Patm_*	*brick:Pressure_Sensor*
*Mdot_*	*brick:Flow_Sensor*
*AirACFM_*	*brick:Air_Flow_Sensor*

Table 5
Example rules for Brick class matching to *Point* instances for FRP dataset.

3.3.4. Validating the created Brick model

The final step in creating the Brick model involved validating the RDF syntax and visualizing the final result. To ensure the correctness of the RDF Turtle file's syntax, we used an online RDF validator tool [39] and confirmed that the created model could be used in external Brick tools such as BrickStudio [15]. Visualizing the model was crucial for checking the semantic accuracy of the hierarchy and overall structure of the classes.

Although BrickStudio [15] can visualize the Brick model, it does not provide a means to modify the model while visually navigating it. The ability to visually confirm the network's structure is often required not only at the final step of the workflow but also during the creation process. However, in our case, we had to revert to the Brick model files, make the necessary edits, save them, and then reimport the updated file. This process could potentially cause significant delays.

4. Challenges of Brick Model Creation

In previous sections, we present two case studies showcasing the creation of Brick models using real-world datasets.

Figure 4 summarizes the workflow of Brick model creation during the case studies. Rather than being a straightforward, step-by-step process, it involves repeated trial and error. The need for external tools for searching Brick class definitions, visualization, and syntax checks adds to the complexity of the process, making it challenging. More specifically, we summarize the challenges of creating a Brick model as follows.

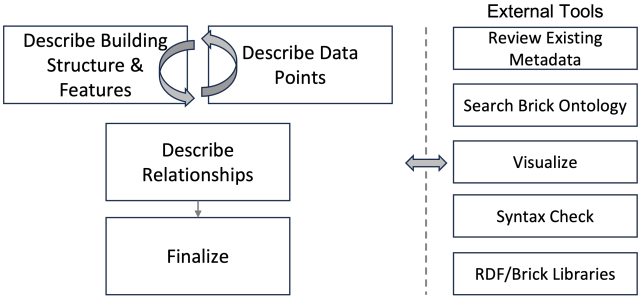


Figure 4: Overview of the Brick model creation workflow

Metadata documents might not exist or have a good level of detail: In the case studies we conducted, we had access to text-based metadata for the research-grade buildings we were studying. However, in many real-world scenarios, metadata for buildings might not exist or might not contain the level of detail required to create an accurate Brick model. Without access to reliable information about a building's physical and logical structure, equipment, measurements, and relationships, it is not possible to create a Brick model that accurately reflects the building's characteristics.

Metadata documents might be inconsistent for automation: The challenge in using building datasets lies in the diverse ways in which they are described in metadata documents, which are often written for humans. These metadata documents usually contain tabular information, where each row describes a column in the dataset. However, the structure of the table and the information available can vary greatly depending on the dataset. For instance, we noted similarities in the metadata of the NZERTF dataset and the FRP dataset, such as data column names, natural language descriptions, and locations, but the way the information was presented differed greatly. We then created relations between the instances, including their location and connections with other instances, such as component and subcomponent relationships. Our case studies showed that string patterns found in the metadata, such as column names, can be used to create these relations. For example, information about the location, equipment, unit, or other details about the data point can be gleaned from these strings.

Automating the process through script creation can be useful, but care must be taken as these patterns might have exceptions and metadata documents are primarily meant to be read by humans, not computers. The rules and scripts developed for one dataset might not be applicable to another dataset, adding to the complexity of the process. Metadata documents might omit important details because they are frequently written with the assumption that readers have prior knowledge of the domain. These omitted details can vary from one metadata document to another. Having more domain knowledge can result in more accurate and efficient creation of a Brick model. If crucial information about the building is missing or unclear in the metadata and cannot be inferred, it is necessary to reach out to the data collector or owner for clarification.

Choosing from large number of classes and relation definitions is difficult: After reviewing the metadata, we define Brick ontology instances for classs such as *Location*, *Zone*, *Equipment*, and *Point*. The next step involves determining the most appropriate Brick class that represents the

entity. This task can prove challenging, as choosing the right Brick ontology class is not always straightforward, since there exists over thousand class definitions in the Brick ontology. More specifically, 1438 classes in the Brick ontology version 1.3.0. The Brick ontology website offers a keyword search tool and a tree browser to help users navigate the class definitions, but there are still hundreds of predefined classes and relations to choose from.

Visually interacting with the Brick model is limited: The process of creating a Brick model involves building a graph, and it is often more natural to visually modify and interact with the model using the graph structure. Although some existing tools can visualize and validate Brick models, these tools do not always allow for easy interaction, modification, and validation of in-progress models. This limitation can make it difficult to make changes to the model as it evolves, which can hinder the model-building process.

Brick models of existing building datasets can be quite time consuming since they require many iterations of manual steps (e.g., understanding column names, units, frequency of data collection, checking exceptions, using different tools at different steps). Even with detailed building information, the person creating the Brick model must be capable of interpreting the information correctly. Because of their abstract nature, these steps cannot be fully automated. It is also a huge hurdle for most people who are interested in creating metadata for their building dataset to obtain knowledge of semantic technology (e.g., RDF data model and Terse RDF Triple Language representation), predefined Brick ontology classes and relations, domain knowledge, script programming, and related libraries such as Brickify [17] and pyBrickSchema [14].

5. VizBrick for Visually Authoring Brick Models

5.1. Overview

To address the challenges discussed in Section 4, we have developed a user-friendly tool called VizBrick that simplifies the creation of Brick models through a visual and interactive interface, eliminating the need for extensive knowledge of the RDF and the details of Brick ontology. VizBrick offers several helpful features, including keyword search, which enables users (e.g., data owners, generators, or facility owners) to easily locate relevant Brick concepts and their corresponding data columns, as well as automatic mapping suggestions. Our aim is to provide a robust and intuitive GUI-based tool that stream lines the process of creating Brick models, particularly for those with a strong understanding of the targeted building. By removing the barriers associated with programming and semantic technology, we hope to encourage broader adoption of Brick and improve the overall efficiency of building automation systems. With VizBrick's help, more building data can be accompanied by Brick models that accurately describe the data, enabling better understanding and use of the information.

The user interface of VizBrick is web based and was constructed using a blend of various web development libraries including JavaScript, jQuery, vis.js, and Bootstrap. The backend is implemented using Python 3 and the Tornado Python web framework, as described in [10]. Despite its web-based architecture, the software can also be installed on a local machine and accessed via standard web browsers. Figure 5 presents the initial user interface of the VizBrick tool upon startup.

VizBrick, as outlined in Figure 6, provides three key functions to its users. First, it offers an interface for reviewing existing tabular-formatted metadata associated with a building dataset. The user can load a CSV file containing data labels and descriptions and then review and modify the metadata before proceeding with the entity and relation creation process. Second, VizBrick enables users to perform keyword searches against the Brick ontology, select specific classes and relations, and create or update entity and relation instances. The tool also features a suggestion function, offering a suitable matching class in the Brick ontology for each data point in the original metadata. Last, during the modeling process, VizBrick visualizes the intermediate and final model, allowing users to interactively modify class and relation instances without the need for coding or RDF expressions. The tool provides the ability to save intermediate or final results and export the result as a standard Turtle file.

Section includes a list of the challenges of creating a Brick model. VizBrick addresses these challenges by providing useful functionalities as follows:

Metadata documents might not exist or have a good level of detail: Although metadata documents are very useful for Brick creation with VizBrick, they are not necessarily required. Users can start defining nodes and edges within the Brick model from scratch or begin with a simple metadata document that includes specific information. The Brick model can be enhanced and updated over time, and the intermediate models can be saved and loaded.

Metadata documents might be inconsistent for automation: VizBrick does not aim to fully automate the Brick model creation workflow. Instead, it assists in human decision-making to make the process more efficient. Erroneous matching results or inconsistent representations of the same elements in the metadata documents can be identified and corrected more easily through visual interaction with the model the user is currently working on.

Choosing from hundreds of class and relation definitions is difficult: VizBrick provides search and suggestion capabilities that help users find the desired class and relationship definitions within the Brick model.

Limited visual interaction with the Brick model: With VizBrick, visualization, modification, and improvement of the Brick model can be performed within the same tool.

5.2. Revisiting the case study

Section 3 includes a case study demonstrating creation of a Brick model for NZERTF without the use of VizBrick. In this section, the same task is described and shown to be

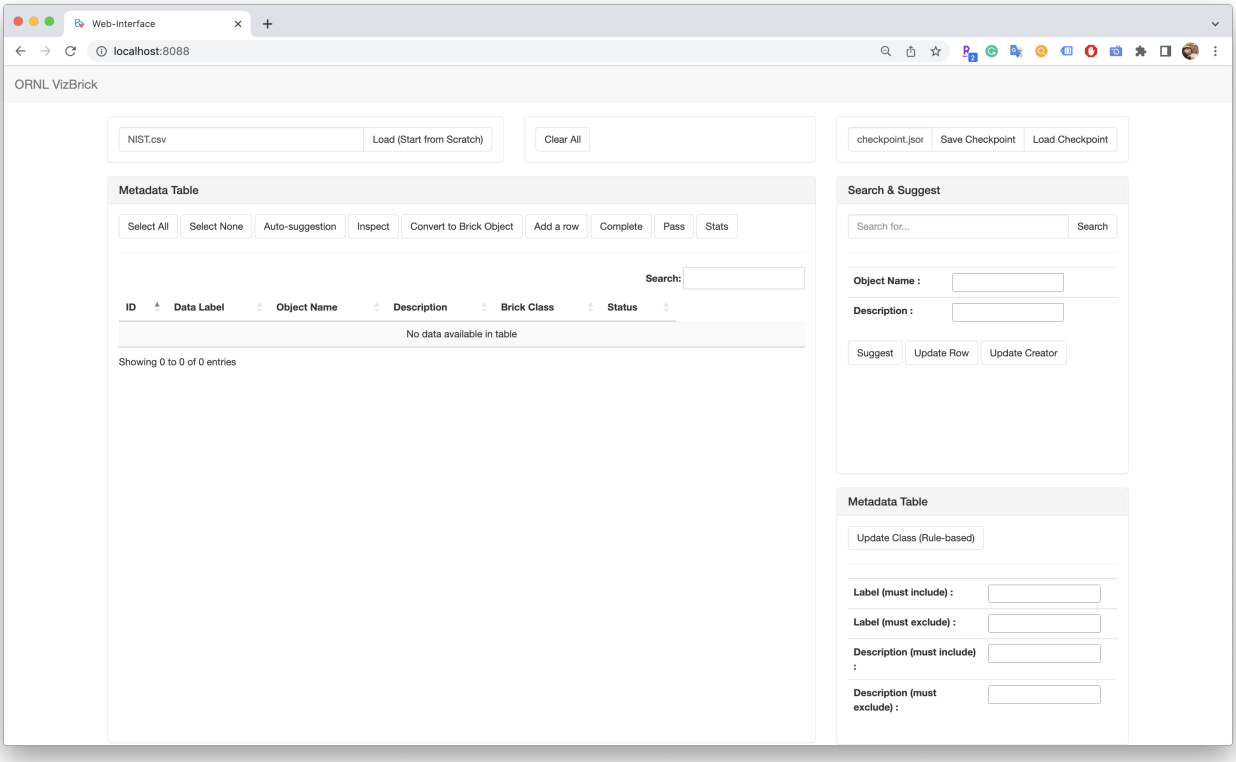


Figure 5: A screenshot of the VizBrick graphical user interface

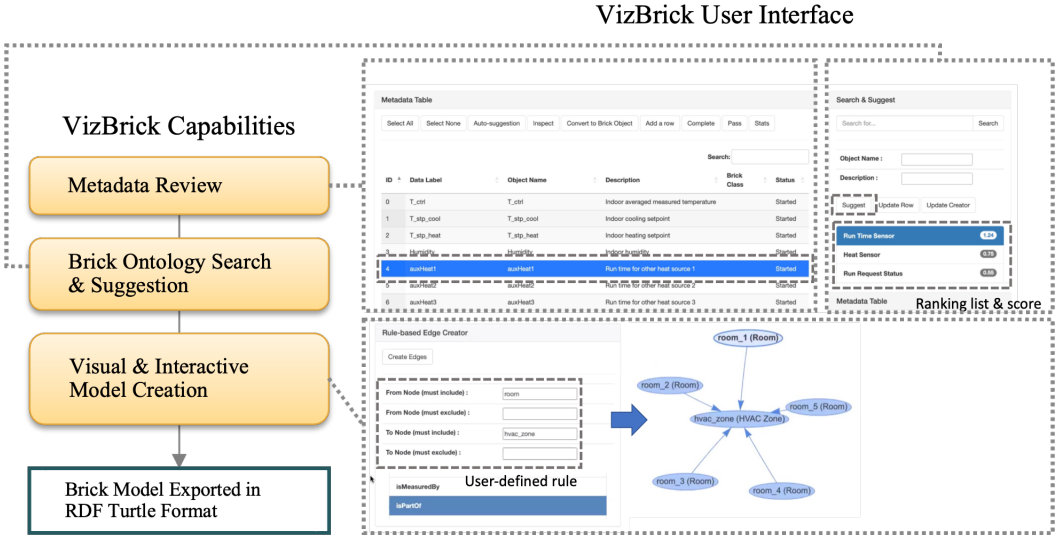


Figure 6: Overview of VizBrick capabilities

more efficient with use of the VizBrick tool. Additionally, the technical details of the capabilities offered by VizBrick and their implementation are discussed in the following section.

5.2.1. Reviewing the existing metadata

Building datasets might exhibit varying metadata formats or lack metadata altogether, as demonstrated in Section 3.2. To address this challenge, VizBrick provides the option to either begin building a Brick model from scratch or import a metadata file in CSV format, if a metadata file is available.

In the case of the NZERTF and FRP datasets used for the case studies, we have access to metadata documents that we could use for Brick model creation with VizBrick. VizBrick can leverage a metadata file that contains information such as data points, location, area, and any other entities related to building assets. Note, however, that the metadata file does not have to be complete. Although the metadata file provides

ID	Data Label	Description
0	*DHW_ ClothesWasherColdFlow*	DHW Utility Flow-Water The cumulative volume of cold water flowing into the clothes washer starting at midnight Gallons
1	*DHW_ ClothesWasherColdFlow*	DHW Utility Flow-Water The cumulative volume of hot water flowing into the clothes washer starting at midnight Gallons

Table 6
Excerpt of the NZERTF database metadata file converted for VizBrick

a starting point, it is not a requirement for VizBrick. The tool can be used without relying on metadata, allowing users to create and enhance the Brick model from scratch. In case of lack of a metadata file, users can skip this step and manually create Brick entities by typing in the name and description.

The CSV file input format used by VizBrick is straightforward and easily adaptable to most existing metadata formats. The CSV file should consist of three columns, as follows.

- **ID:** Row number sequentially increasing
- **Name:** Name of a data measurement point, location, or any entity in the dataset
- **Description:** Natural language or keyword description

Table 2 displays the original metadata for the NZERTF database, which was subsequently converted into a VizBrick-compatible CSV file format, as shown in Table 6. The conversion process was straightforward, requiring only the merging of multiple columns (Subsystem, Measurement Location, Measured Parameter, Description, Units) into a single column (Description). Once the input metadata file is prepared, it can be imported into the VizBrick tool, as depicted in Figure 7. The contents of the CSV file are then visualized, allowing the user to navigate and review the metadata.

Although some preprocessing of metadata files might be necessary in most use cases, as metadata files can vary, preparing a CSV file for VizBrick does not involve complicated data processing. The requirements and restrictions for the CSV file are minimal. The ID column needs to have unique values, and the Name and Description columns need to contain text. Apart from these basic criteria, there are no significant constraints or complexities involved in preparing the CSV file for VizBrick.

5.2.2. Modeling Brick class instances

With the use of the VizBrick tool, the next step is to convert each data label in Table 6 into a Brick entity. A Brick entity is an instance of a class defined in the Brick ontology, identified by a URI. The conversion of a data label into a Brick class instance involves describing the data label in terms of the Brick ontology. For instance, the data label "DHW_ClothesWasherColdFlow" can be transformed into a "Water Usage Sensor" type entity, indicating that the data label represents data measured by a device of the "Water Usage Sensor" class in the Brick ontology.

VizBrick offers search and suggestion capabilities to aid users in accurately identifying the appropriate data label. **Search and Suggestion Functionality**: VizBrick offers efficient search and suggestion capabilities, allowing users to access Brick class definitions in the Brick Ontology without the need for external tools, such as the Brick ontology website. These capabilities are implemented using the TF-IDF algorithm [33], a widely used information retrieval technique. In VizBrick, each Brick class definition is treated as a document. Figure 8 shows an example of a class definition in the Brick ontology. Each document is then preprocessed by performing lowercasing, removing stop words, stemming, and removing punctuation or special characters.

Next, these documents are transformed into Term Frequency-Inverse Document Frequency (TF-IDF) vectors. Each dimension of the vector corresponds to a unique term in the document corpus, and the value represents the TF-IDF score of that term in the document. TF-IDF is the multiplication of two components: TF and IDF, reflecting the importance of a term in a document collection.

The TF component measures the frequency of a term (word) within a document. It is calculated as the ratio of the number of occurrences of a term to the total number of terms in the document. It aims to capture the relevance of a term within a specific document. Formally, the TF score of $\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$, where $f_{t,d}$ is the frequency of a term in a document d. The denominator is simply the total number of terms in document d.

The IDF component measures the rarity of a term across the entire document collection. Mathematically, it can be denoted as $\text{idf}(t, D) = log \frac{N}{|d \in D : t \in d|}$, where N is the total number of documents and the bottom term $|d \in D : t \in d|$ represents the number of documents where the term t appears. IDF helps to give higher weight to terms that appear in a smaller number of documents, thus emphasizing their importance in distinguishing documents.

The TF-IDF score quantifies the importance of a keyword within a specific document, reflecting its significance in distinguishing the document from others. When a query is given for a search, the query is converted into a TF-IDF vector in the same way, considering the set of given query terms as a document. Then, we calculate the cosine similarity between the TF-IDF vector of the query and the TF-IDF vectors of the documents. Finally, the documents are ranked based on the scores, and the top-ranked documents are returned as the result.

As an example, consider the scenario in which a user searches for the "Water Usage Sensor" class using the key-

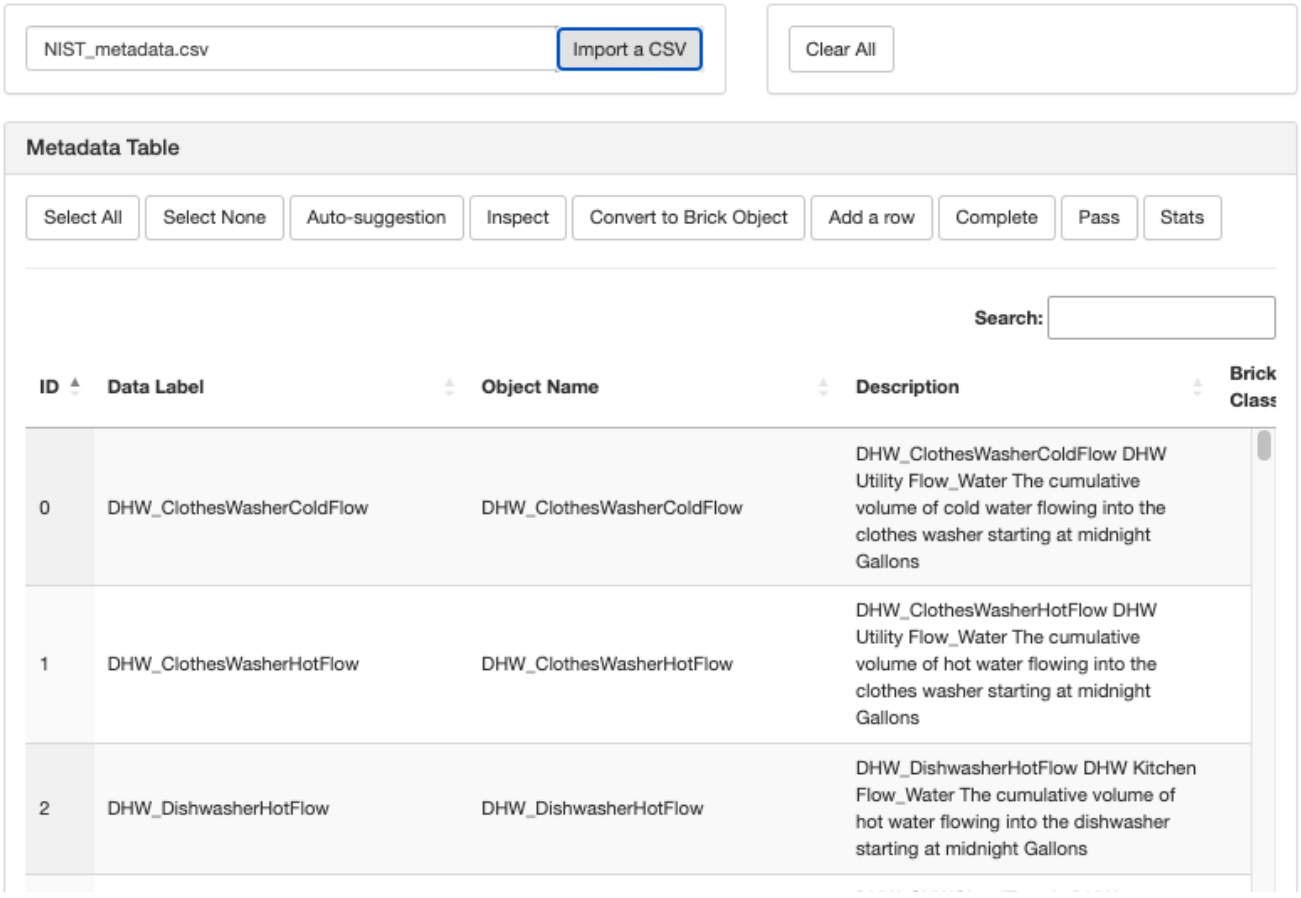


Figure 7: A metadata CSV file of the NZERTF dataset, imported into the VizBrick tool

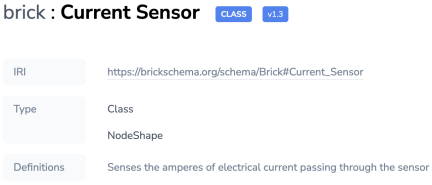


Figure 8: Definition of "Current Sensor" class in the Brick ontology

words "water usage." Figure 9 illustrates the results of this search, which returns a ranking list of Brick classes with their corresponding relevance scores. In this instance, the "Water Usage Sensor" class is ranked first.

The search functionality in the Brick ontology classes is a valuable feature as it eliminates the need for users to rely on external tools for searching or navigating the Brick classes. To use this feature, users are required to input search keywords. The suggestion capability, on the other hand, provides a means of finding a Brick class even when the search keyword is not immediately apparent. The implementation of the suggestion capability is similar to the search capability with modifications. The name and description of the data label are used, in addition to any keywords provided by the user, as the search query. In cases where no keywords are provided, the words present in the name and description of

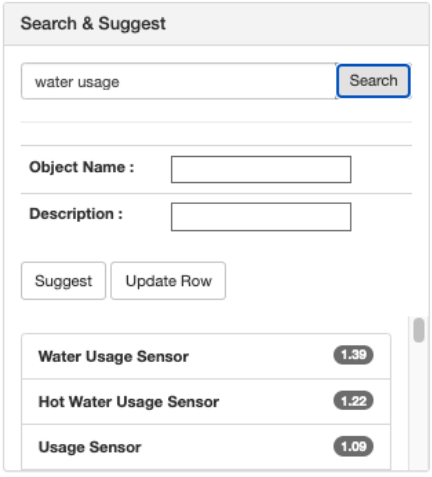


Figure 9: A user can search for classes in the Brick ontology using keywords. The figure illustrates an example of searching for Brick ontology classes using the keywords 'water usage'. The resulting Brick classes, along with their relevance scores, are displayed in the search results.

the selected data label are used as keywords. If additional keywords are given, they will be used to augment the search. The quality of the search results might vary depending on the richness and descriptiveness of the data label descrip-

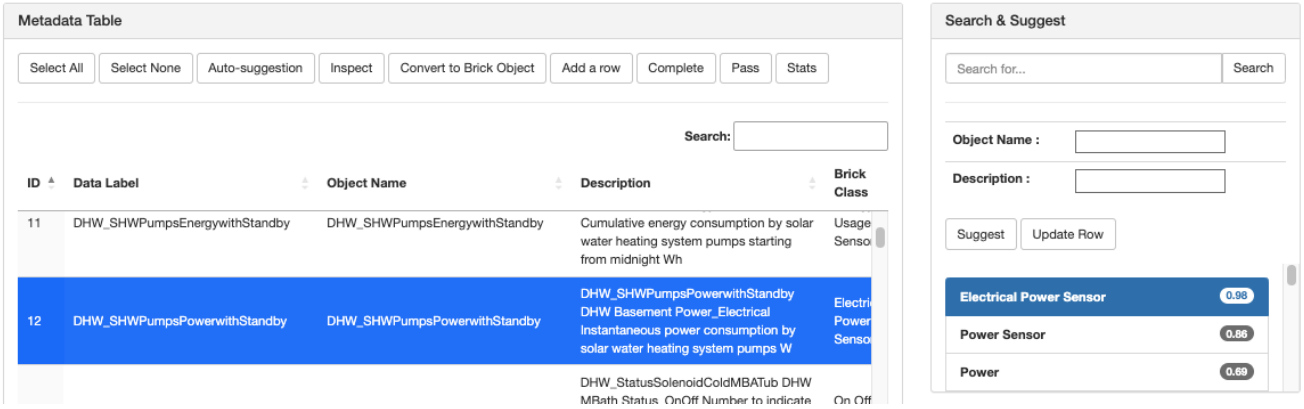


Figure 10: Brick class suggestion result generated by VizBrick for the data label "DHW_ClothesWasherColdFlow."

tion. In instances where the description is rich and descriptive, there is a higher likelihood that users will not need to provide additional keywords. Conversely, if the description is lacking, user-specified keywords will be necessary to retrieve the most relevant Brick classes for the selected data label.

Figure 10 illustrates the result of the suggestion capability for the "DHW_SHWPumpsPowerwithStandby" data label, which is presented as a ranked list of the most relevant Brick classes. In this example, the class "Electrical Power Sensor" was ranked first with the highest score. The retrieval was performed using the data label name and description, without the use of any search keywords. However, users can provide additional keywords in the search bar to refine the results.

Enhancement of Search and Suggestion Results via Explicit Positive User Feedback: VizBrick leverages users' interactions to continually improve the accuracy of its search and suggestion results. During the modeling process, users are able to indicate the relevance of a given Brick class in the search or suggestion results by selecting it and clicking the "Update Row" button, which confirms the selected class is correct. This explicit positive feedback is recorded in the database, allowing subsequent search and suggestion results to benefit from this information. Specifically, when a user confirms the relevance of a given result, the words appearing in the corresponding query (e.g., user-specified keywords, data label name, and description) are appended to the original Brick class document and its vector representation is updated accordingly.

Over time, this process enables the system to match words that were not originally present in the definition of Brick classes. VizBrick also provides an auto-suggestion function that generates suggestions for every data label in an imported metadata table and automatically selects the Brick class with the highest matching score. Although the highest-scoring suggestion might not always be the best match, the use of descriptive text for each data label in the metadata can help users save time by streamlining the review and modification process. Rather than conducting searches and suggestions for each data label individually, users are able to review and modify auto-suggested results quickly.

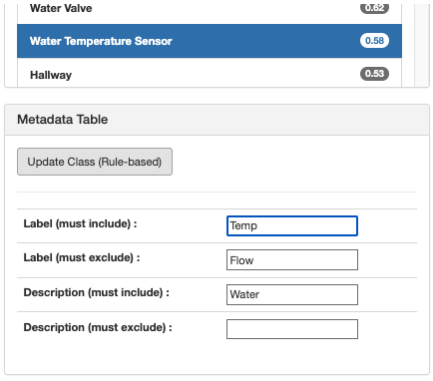


Figure 11: An example of rule-based Brick class mapping based on a user-defined rule. In this example, the system updates all data labels with the words "Temp" and "Water" in their names and descriptions to the class "Water Temperature Sensor," as long as the word "Flow" is not present in the name. This update is performed when the button "Update Class (Rule-based)" is clicked. When there are multiple common patterns in the data label, as shown in Table 4 and Table 5, rule-based class matching can be an efficient way to find the correct Brick classes.

Rule-Based Class Matching: In addition to VizBrick's powerful search and suggestion functions, the system also provides a way for users to find Brick classes based on user-defined rules in instances where repeated patterns are present in the data label or description text. A user-defined rule consists of four parts, namely lists of words that must be included and excluded from the data label name and description, respectively. An example of using a user-defined rule is depicted in Figure 11.

Evaluation of VizBrick's Suggestion Capability: The effectiveness of VizBrick's suggestion capability is closely tied to the quality and quantity of the available descriptions for data labels. Note that VizBrick is not intended to be a fully automated tool; instead, it works in conjunction with a user and uses additional information provided, such as keywords, to optimally function. Consequently, quantifying the usefulness of VizBrick's suggestion capability poses a challenge. Nonetheless, we conducted an evaluation using the

following approach.

Considering the intended users of VizBrick, it is reasonable to assume that they possess adequate knowledge about building datasets and terminologies related to building assets. However, they might not have extensive familiarity with Brick ontology and programming, which justified the reason for the user to use VizBrick. When the ranking list contains relevant suggestions among the higher ranks, users can more efficiently locate the desired Brick ontology class they are seeking. The evaluation of VizBrick's performance uses the top-k hit ratio (HR@k) metric [23], which measures the proportion of correct answers found within the top-k suggestions and quantifies the effectiveness of VizBrick in delivering the desired class within the top-k suggestions. In other words it measures the effectiveness of VizBrick in providing accurate suggestions within top-k. The evaluation was conducted separately using scripts but using the same suggestion code that we used in the GUI.

For NZERTF and FRP datasets, the ground truth for each data label was determined by the manually selected Brick class designated by subject matter experts who have expertise in the NZERTF and FRP building and datasets.

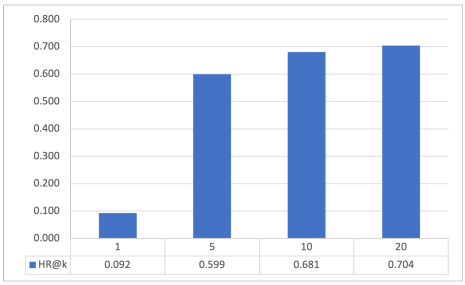
To compute the HR@k metric, we measured the HR@k at k = 1, 5, 10, and 20 for all data labels. This allows us to quantitatively assess the effectiveness of VizBrick's suggestion capability.

Figure 12a presents the results of the evaluation of the VizBrick tool for the NZERTF and FRP datasets. For NZERTF, we generated 379 suggestions, which is the number of data labels, and the results indicate that the top 1 accuracy was 9.2%, with a significantly higher hit ratio of 59.9% at the top 5 (HR@5). This is promising, as it suggests that users will predominantly use the VizBrick tool to focus on a small number of highly ranked Brick classes, which is a common scenario in real-world use cases.

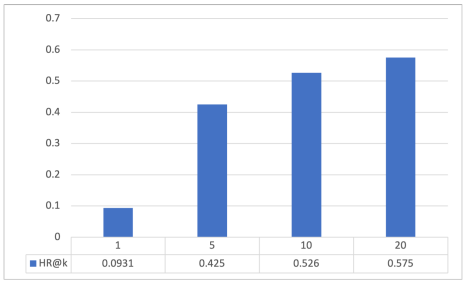
For the FRP dataset, the descriptions for each data label, available in the metadata, were quite sparse. Out of 247 data labels, 106 did not have any descriptions for the data columns. As presented in Table 3, when descriptions were available, they primarily concentrated on the constraints and additional information about each data column, rather than their definitions. In the context of FRP, the names of data columns suggest their nature, but abbreviations were used extensively. For example, *T* was used for *Temperature*, and *RH Humidity* was used for *Relative Humidity*. In this case, without additional information from a user, VizBrick's suggestion capability would not perform well. The following shows an example of VizBrick's top 5 suggested Brick classes with ranking using the available description text and data name for "T_Room_202_Avg" data label.

Listing 6: Example of VizBrick suggestions

```
NAME = T_Room_202_Avg
DESCRIPTION = Room 202 Room Ambient
1 Room 1.78
2 Room Air Temperature Setpoint 1.20
3 Pump Room 0.91
4 Electrical Room 0.89
5 Equipment Room 0.89
```



(a) NZERTF



(b) FRP

Figure 12: The HR@k of VizBrick's suggest capability measured at k = 1, 5, 10, and 20.

Since available text is very limited (e.g., Room), the result was not good. In real-world scenarios, a user might opt for rule-based matching over automatic matching, as human experts can comprehend the patterns appearing in the data columns. For instance, a user could match all the data columns with the Brick class *Temperature Sensor* if a column contains the abbreviation *T* or implement more complex rules as illustrated in Figure 11. Alternatively, to use the automatic suggestion feature, a user could supply extra information as keywords to enhance the VizBrick tool's suggestion results. The following shows the result with additional information, expansion of abbreviations.

Listing 7: Example of VizBrick suggestions using additional keywords

```
NAME = T_Room_202_Avg
DESCRIPTION = Room 202 Room Ambient Temperature Average
1 Room 1.45
2 Room Air Temperature Setpoint 1.22
3 Average Zone Air Temperature Sensor 0.77
4 Pump Room 0.74
5 Electrical Room 0.72
```

As we can see, with few additional keywords, VizBrick was able to find a good answer at rank 3. To simulate the latter situation, we assumed that a user provides the full names of the abbreviations used in the data column names. We generated 247 suggestions. The results showed that the top 1 accuracy was 9.31%, and there was a substantial increase in the hit ratio to 42.5% at top 5 (HR@5). This shows that VizBrick's suggestion capability will be useful for users to find appropriate Brick classes more easily using interactively provided additional information during the Brick creation process.

By using search, rule-based class matching, and suggestion features, users can effortlessly locate the Brick class that best describes each data label. This semantic description of the data labels, consistent across various datasets, constitutes a notable enhancement over the original metadata, which was a dataset-specific description of the information.

The next step in the process is to model the physical building structures and related Brick class instances, as described in Sections 3.3.2 and 3.3.3. Note that the imported metadata table includes only a list of data labels, thus additional rows need to be added to describe physical and logical entities, such as locations (e.g., floors, rooms, hallways) and equipment (e.g., heat pumps, washers, dishwashers). Each entity should be assigned a unique name to facilitate identification. Subsequently, we will use search and suggestions to find a Brick class for each entity in a similar manner as we did for the data labels.

5.2.3. Creating Brick relations

Once the Brick class instances (vertices) have been modeled, the next step is to model their relations (i.e., to create edges between vertices). Figure 13 provides an example of creating relations, where the "First_Floor" entity is linked to five other Brick class instances using the Brick relation "hasPart." This represents the semantic relation that the "First_Floor," which is an instance of the Brick class "Floor," has five other locations as its parts. This process can be repeated to create edges that describe the relations between previously created instances. The VizBrick tool facilitates this process by providing a GUI that interacts with a visual representation of the graph. The user can select, edit, and update both vertices and edges, while interacting with the visualized graph. Figure 14 depicts the graph visualization of the physical and logical features of the building that were created using the VizBrick tool. The tool also provides a way to visualize a selected part of the graph; therefore, in the event that the graph becomes large, the user can interact with and visualize a portion of the graph without having to view the entire graph, which can be very complex.

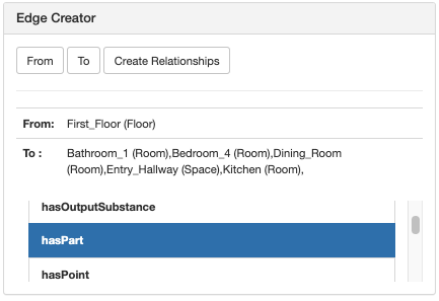


Figure 13: Example of creating relations between Brick class instances using 'hasPart' relationship

The subsequent step is to connect all of the instances created from the data labels to instances representing building features, such as locations and equipment. For the NZERTF dataset, the measurement locations information provided in Table 2 can be used to accomplish this connection.

5.2.4. Validating and exporting the Brick model

The final model encompasses the building, building features such as locations and zones, equipment, and sensors, all of which describe the NZERTF building and the data collected from the building for the dataset. The final result can be exported to a standard RDF turtle file for further use and analysis.

Listing 8: An excerpt of the turtle representation of the final Brick model for the NZERTF building.

```
@prefix bldg: <https://url_of_this_data#> .
@prefix brick: <https://brickschema.org/schema/1.1/Brick#> .

bldg:Attic a brick:Room .
bldg:Elec_EnergyLightsAttic a brick:Energy_Usage_Sensor .
bldg:Elec_EnergyMakeUpAirDamper a brick:Energy_Usage_Sensor .
bldg:Elec_EnergyPlugsAttic a brick:Energy_Usage_Sensor .
bldg:Elec_PowerLightsAttic a brick:Power_Sensor .
bldg:Elec_PowerMakeUpAirDamper a brick:Power_Sensor .
bldg:Elec_PowerPlugsAttic a brick:Power_Sensor .
bldg:IndEnv_RoomTempAtticNE a brick:Air_Temperature_Sensor .

bldg:Attic brick:hasPoint bldg:Elec_EnergyLightsAttic .
bldg:Attic brick:hasPoint bldg:Elec_EnergyMakeUpAirDamper .
bldg:Attic brick:hasPoint bldg:Elec_EnergyPlugsAttic .
bldg:Attic brick:hasPoint bldg:Elec_PowerLightsAttic .
bldg:Attic brick:hasPoint bldg:Elec_PowerMakeUpAirDamper .
bldg:Attic brick:hasPoint bldg:Elec_PowerPlugsAttic .
```

As the Turtle file format is a widely accepted standard, the exported RDF files can be used with other standard Brick-related tools. For example, the file can be validated using any RDF syntax validator as specified in [39], and it can also be imported into other external tools, such as BrickStudio [15].

5.2.5. Advantages of VizBrick

In this section, we revisit the case study on the NZERTF dataset and describe how its Brick model can be created end to end using the VizBrick tool. Figure 15 summarizes the workflow of Brick model creation with VizBrick. Although the process still involves repeated trial and error, the need for external tools for searching Brick class definitions, visualization, and syntax checks is now significantly reduced compared with the previous workflow (Figure 4). More specifically, the use of the VizBrick tool provided several advantages during the Brick model creation process.

First, the need for writing code was eliminated. Although the original metadata document needed to be converted into CSV file format, this was a straightforward process that could be accomplished using spreadsheet software without writing any program codes. Once the input file for the VizBrick tool was ready, the rest of the Brick model authoring process was performed within the tool itself.

Second, external tools were not necessary. In our original case study, we used the Brick website to search and navigate the Brick class and relation definitions and external tools such as BrickStudio to visualize the intermediate or final result. With VizBrick, searching and visualizing the

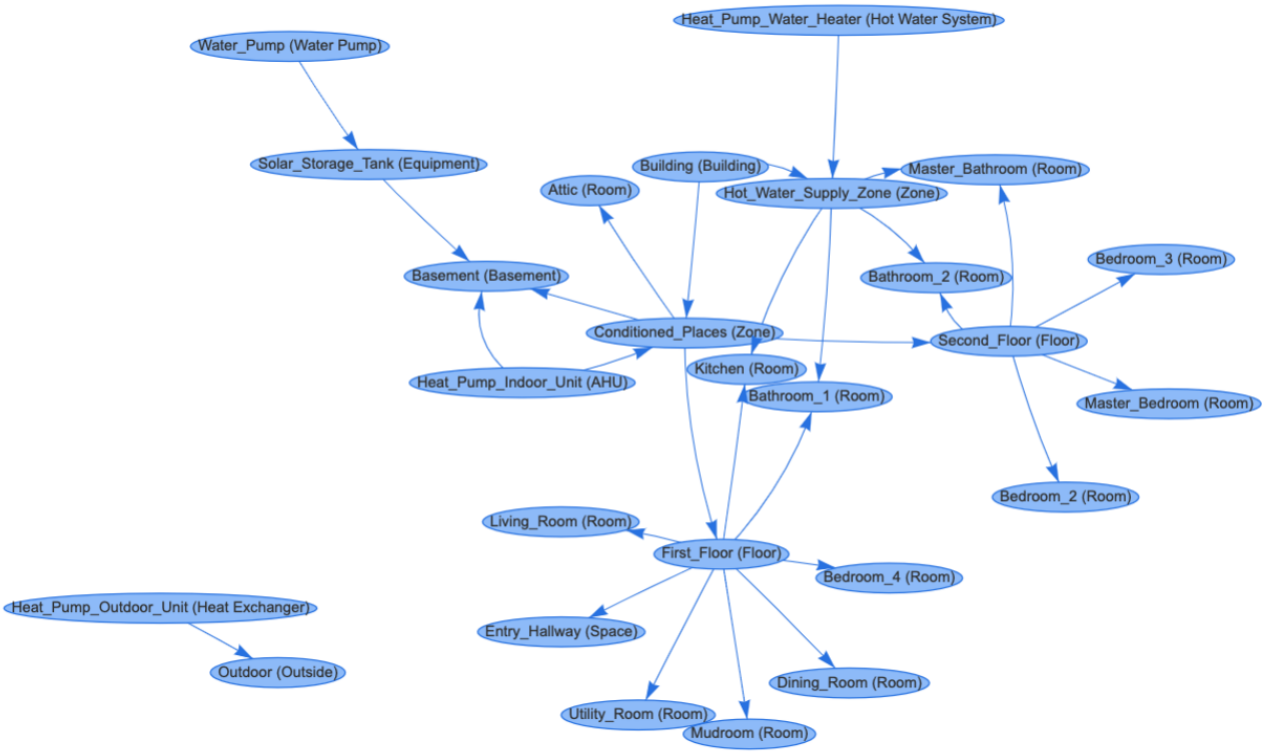


Figure 14: Graph visualization of the building's physical and logical features

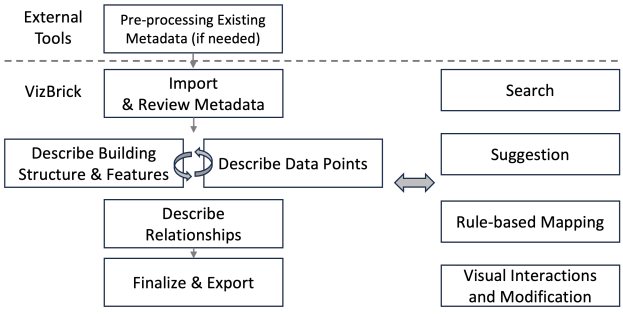


Figure 15: Brick model creation workflow with VizBrick

Brick ontology and current model can be performed in one place.

Third, VizBrick provides a suggestion capability to support faster and more efficient Brick model creation. In our original case study, finding the most appropriate Brick class for a data label relied solely on the user's knowledge of the Brick ontology, which could be challenging given the large number of Brick class definitions (1438 class definitions in the Brick ontology version 1.3.0). VizBrick leverages the text information available for the data labels and the Brick ontology class definitions to generate a ranking list, allowing users to quickly find the most suitable Brick class. As demonstrated in Section 5.2.2., VizBrick's suggestion capability was promising, with a hit ratio of 59.9% at top 5 for our example dataset.

Fourth, the intermediate Brick model can be saved and loaded during the authoring process. Brick model creation can be time consuming, especially for large datasets collected from complex buildings. For example, a dataset might contain hundreds of data points, as we have seen in the case of NZERTF (379 data points) and FRP datasets (247 data points). VizBrick provides the ability to save checkpoints to be resumed in multiple sessions.

Finally, VizBrick generates data in the standard RDF format, capitalizing on the benefits of RDF-like interoperability, which enables seamless collaboration with various RDF tools. Examples include Apache Jena [35] and Protégé [30]. It is also compatible with Brick tools such as BrickStudio [15] and BrickViewer [16].

Overall, the case study demonstrates the usefulness of the VizBrick tool. The tool provides effective search and suggestion capabilities, allows for visualization and interactive modifications, and has the potential to significantly reduce the time and effort required for Brick modeling.

VizBrick has been released as open-source software [26] and has been registered with the US Department of Energy Office of Scientific and Technical Information [25].

The VizBrick tool was developed as part of the Benchmark Datasets of Building Environmental Conditions and Occupancy Parameters project. This project was a collaborative effort by Oak Ridge National Laboratory, Pacific Northwest National Laboratory, Lawrence Berkeley National Laboratory, and National Renewable Energy Laboratory, aiming to collect and curate a selection of high-resolution building systems datasets with broad applicability to address high-impact use cases. The development process garnered positive feedback from various team members. Besides the Brick

models for FRP and NZERTF case studies presented in this paper, Brick models were also created for the Habitat for Humanity Zero Energy Home [31] and the Ecobee Donate Your Data dataset [11], curated by Lawrence Berkeley National Laboratory and National Renewable Energy Laboratory, correspondingly. These diverse applications demonstrate the versatility of VizBrick, illustrating its capability to facilitate the development of Brick models across a range of building datasets. Furthermore, VizBrick accommodates users with varying levels of expertise, highlighting its potential as a comprehensive tool.

6. Conclusion

Building datasets are crucial for the successful interoperability and automation of smart building applications. To achieve this, it is essential to standardize building datasets. One way to achieve this standardization is by creating Brick models for building datasets. However, creating Brick models is a challenging task that requires knowledge of semantic technology, such as ontology declarations and RDF syntax, and often results in repeated manual trials and errors.

In this paper, we present the challenges encountered during the creation of Brick models for two real-world residential and commercial building datasets. We then introduce VizBrick, a novel interactive Brick model authoring tool that enables users to visually and interactively create semantic building metadata. This tool was designed to mitigate the difficulties encountered during the creation of Brick models. By revisiting one of the case studies, we demonstrate how VizBrick's handy features, such as keyword search for relevant Brick concepts and automatic concept mapping suggestions and recommendations, can significantly reduce the time and effort required for creating Brick models.

We believe that there are multiple directions for future improvements in the VizBrick tool. One potential area is the incorporation of semantic ranking to enhance the accuracy of suggestions and recommendations. The current version of VizBrick cannot understand the synonyms or implicitly related terms. Techniques such as Word2Vec [29] for word embeddings and Latent Dirichlet Allocation [21] for topic modeling can be explored to handle synonyms and related queries more effectively. Furthermore, we plan to explore extending the tool to integrate with other ontology-based data models. This integration would leverage the collaborative nature of these models to provide more precise and comprehensive descriptions of buildings and building datasets. References to relevant works in this area include [2, 7, 6]. We also recognize that enhancing the validation capability of VizBrick can be achieved by incorporating external tools such as Shapes Constraint Language (SHACL) [38], RDFUnit [37], or OWL API [19]. Additionally, conducting user studies and gathering feedback will allow us to assess the tool's usability and effectiveness in real-world scenarios. Understanding the needs and perspectives of users will provide valuable insights for refining and enhancing the tool accordingly.

Moreover, we aim to evaluate the performance of the tool on larger and more diverse datasets to ensure its scalability and generalizability. This evaluation will help validate the tool's capabilities and identify any potential limitations or areas for further optimization.

Overall, the case studies and the introduction of VizBrick demonstrate the importance of standardizing building datasets and provide a solution to the challenges encountered during the creation of Brick models for building datasets. By providing a visual and interactive tool for creating Brick models, VizBrick makes it easier for users to standardize building datasets, thus facilitating the interoperability and automation of smart building applications.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used Grammarly, WordTune, and ChatGPT to perform spell check and correct grammatical errors. After using this tools/services, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., et al., 2016. Brick: Towards a unified metadata schema for buildings, in: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, pp. 41–50. https://doi.org/10.1145/2993422.2993577.

[2] Beetz, J., Van Leeuwen, J., De Vries, B., 2009. IfcOWL: A case of transforming EXPRESS schemas into ontologies, in: Ai Edam, Cambridge University Press. pp. 89–101. https://doi.org/10.1017/S0890060409000122.

[3] Berkoben, K., El Kaed, C., Sodorff, T., 2020. A Digital Buildings Ontology for Google's Real Estate., in: Proceedings of the 19th International Semantic Web Conference, pp. 392–394. ISSN 1611-3349.

[4] Brick Consortium, . Brick Ontology. https://brickschema.org/ontology. (Date accessed: 2023-7-20).

[5] buildingSMART International, . Industry Foundation Classes. https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/. (Date accessed: 2023-7-20).

[6] Charpenay, V., Käbisch, S., Anicic, D., Kosch, H., 2015. An ontology design pattern for IOT device tagging systems, in: The 5th International Conference on the Internet of Things), IEEE. pp. 138–145. https://doi.org/10.1109/IOT.2015.7356558.

[7] Daniele, L., Hartog, F.d., Roes, J., 2015. Created in close interaction with the industry: the smart appliances reference (SAREF) ontology, in: International Workshop Formal Ontologies Meet Industries, Springer. pp. 100–112. https://doi.org/10.1007/978-3-319-21545-7_9.

[8] Decker, S., Melnik, S., Van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I., 2000. The semantic web: The roles of XML and RDF, IEEE. pp. 63–73. https://doi.org/10.1109/4236.877487.

[9] Domich, P.D., Pettit, B., Fanney, A.H., Healy, W.M., 2015. Research and development opportunities for the NIST net zero energy residential test facility https://doi.org/10.6028/NIST.TN.1869.

[10] Dory, M., Parrish, A., Berg, B., 2012. Introduction to Tornado: Modern Web Applications with Python. O'Reilly Media. ISBN-10 9781449309077.

[11] Ecobee, . Ecobee Donate Your Data. https://www.ecobee.com/en-us/donate-your-data/. (Date accessed: 2023-7-20).
[12] Fanney, H., Healy, W.M., 2014. Design challenges of the NIST net zero energy residential test facility. https://doi.org/10.6028/NIST.TN.1847.
[13] Fierro, G., . brick-builder GitHub repository. https://github.com/gtfierro/brick-builder. (Date accessed: 2023-7-20).
[14] Fierro, G., 2023. py-brickschema. https://pypi.org/project/brickschema/. (Date accessed: 2023-7-20).
[15] Fierro, G., Nagare, S., a. Brick Studio. https://brickschema.github.io/brick-studio/. (Date accessed: 2023-7-20).
[16] Fierro, G., Nagare, S., b. Brick TTL Viewer. https://viewer.brickschema.org/. (Date accessed: 2023-7-20).
[17] Fierro, G., Nagare, S., 2020. Brickify tool. https://brickschema.readthedocs.io/en/latest/brickify/index.html. (Date accessed: 2023-7-20).
[18] Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E., Walle, R.V.d., 2016. RMLEditor: a graph-based mapping editor for linked data mappings, in: Proceedings of the European Semantic Web Conference, Springer. pp. 709–723. https://doi.org/10.1007/978-3-319-34129-3_43.
[19] Horridge, M., Bechhofer, S., 2011. The Owl API: A Java API for Owl ontologies, IOS Press. pp. 11–21. ISSN 1570-0844.
[20] Im, P., Bhandari, M., 2012. Use of flexible research platforms (FRP) for BIM and energy modeling research, American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc. (ASHRAE). pp. 197–205. ISSN 0001-2505.
[21] Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., Zhao, L., 2019. Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey, Springer. pp. 15169–15211. https://doi.org/10.1007/s11042-018-6894-4.
[22] John, J., et al., 2020. Project haystack data standards, in: Energy and Analytics, River Publishers. pp. 237–243. ISBN 9781003151944.
[23] Koren, Y., 2008. Factorization meets the neighborhood: a multi-faceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 426–434. https://doi.org/10.1145/1401890.
[24] Lassila, O., Swick, R.R., et al., 1999. Resource description framework (RDF) model and syntax specification. https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. (Date accessed: 2023-7-20).
[25] Lee, S., Im, P., Bhandari, M., Cui, B., 2022. Vizbrick. doi:10.11578/dc.20220609.1. https://www.osti.gov/biblio/1871804 (Date accessed: 2023-7-20).
[26] Lee, S., Im, P., Bhandari, M., Cui, B., 2023. VizBrick Github Repository. https://github.com/liza183/vizbrick. (Date accessed: 2023-7-20).
[27] Luo, N., Pritoni, M., Hong, T., 2021. An overview of data tools for representing and managing building information and performance data, Elsevier. p. 111224. https://doi.org/10.1016/j.rser.2021.111224.
[28] McGuinness, D.L., Van Harmelen, F., et al., 2004. Owl web ontology language overview. W3C recommendation 10, 2004.
[29] Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. https://doi.org/10.48550/arXiv.1301.3781.
[30] Musen, M.A., 2015. The protégé project: a look back and a look forward, ACM New York, NY, USA. pp. 4–12. https://doi.org/10.1145/2757001.2757003.
[31] Norton, P., Xiong, J., . Habitat for Humanity Zero Energy Home (ZEH) Dataset. https://bbd.labworks.org/ds/bbd/habitatzeh. (Date accessed: 2023-7-20).
[32] Petersen, N., Similea, A., Lange, C., Lohmann, S., 2017. TurtleEditor: A web-based RDF editor to support distributed ontology development on repository hosting platforms, World Scientific. pp. 311–323. 10.1142/S1793351X17400128.
[33] Ramos, J., et al., 2003. Using TF-IDF to determine word relevance in document queries, in: Proceedings of the first instructional conference on machine learning, pp. 29–48.
[34] Rasmussen, M.H., Lefrançois, M., Schneider, G.F., Pauwels, P., 2021. BOT: the building topology ontology of the W3C linked building data group, IOS Press. pp. 143–161. https://doi.org/10.3233/SW-200385.
[35] The Apache Software Foundation, . Apache Jena. https://jena.apache.org/. (Date accessed: 2023-7-20).
[36] Ullah, T., Poppendieck, D., Healy, W., Fanney, A., Teichman, K., 2016. Energy and indoor air quality benchmarking of the NIST Net-Zero Energy Residential Test Facility (NZERTF), pp. 21–26. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=920720.
[37] W3C, a. RDFUnit - RDF Unit Testing Suite. https://github.com/AKSW/RDFUnit. (Date accessed: 2023-7-20).
[38] W3C, b. W3CShapes Constraint Language (SHACL). https://www.w3.org/TR/shacl/. (Date accessed: 2023-7-20).
[39] W3C, 2006. RDF Validator. https://www.w3.org/RDF/Validator/. (Date accessed: 2023-7-20).
[40] Weiten, M., 2009. Ontostudio® as a ontology engineering environment, Springer. pp. 51–60. ISBN-10 3540888446.