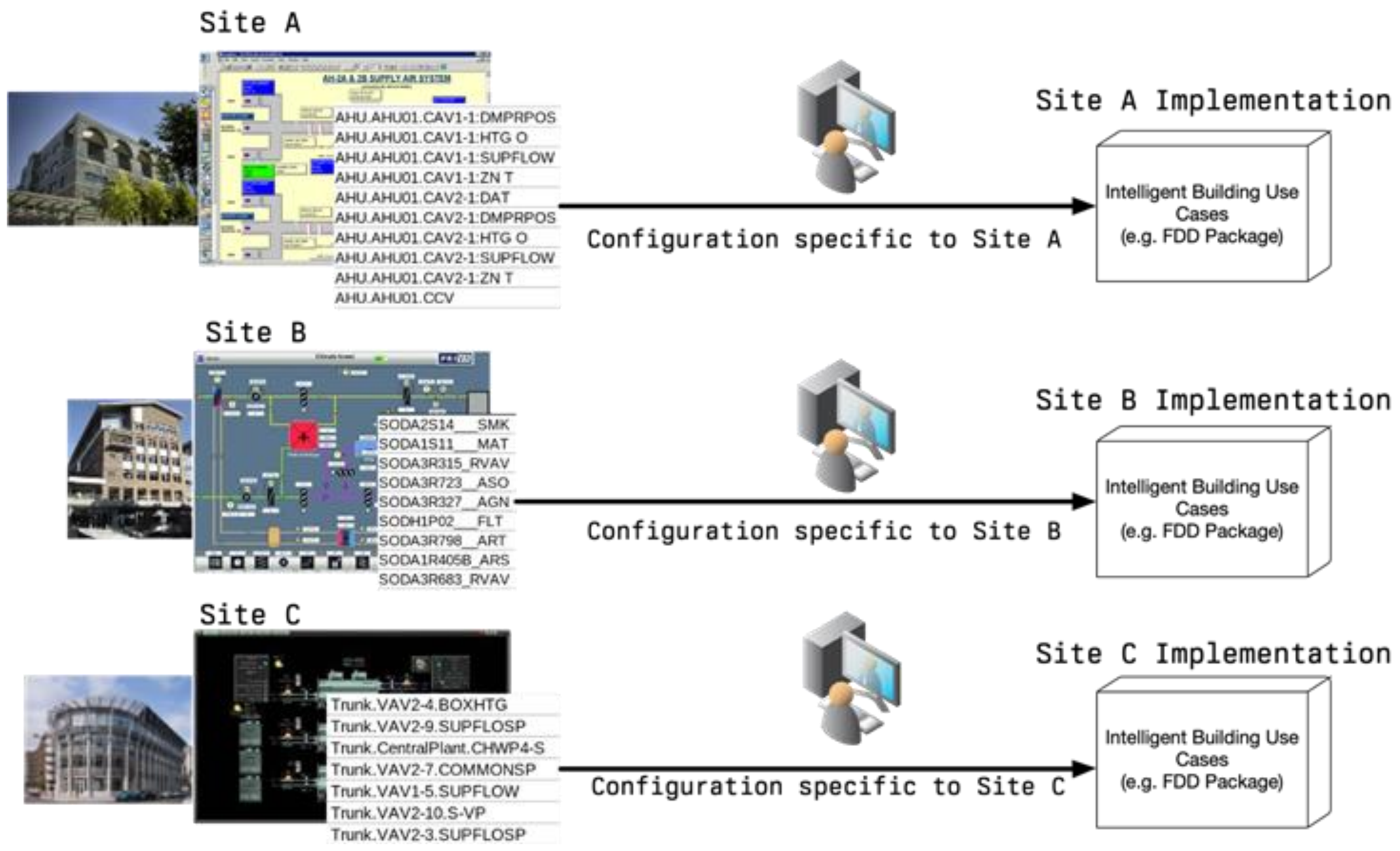
Data-driven Applications Require Extensive Configuration

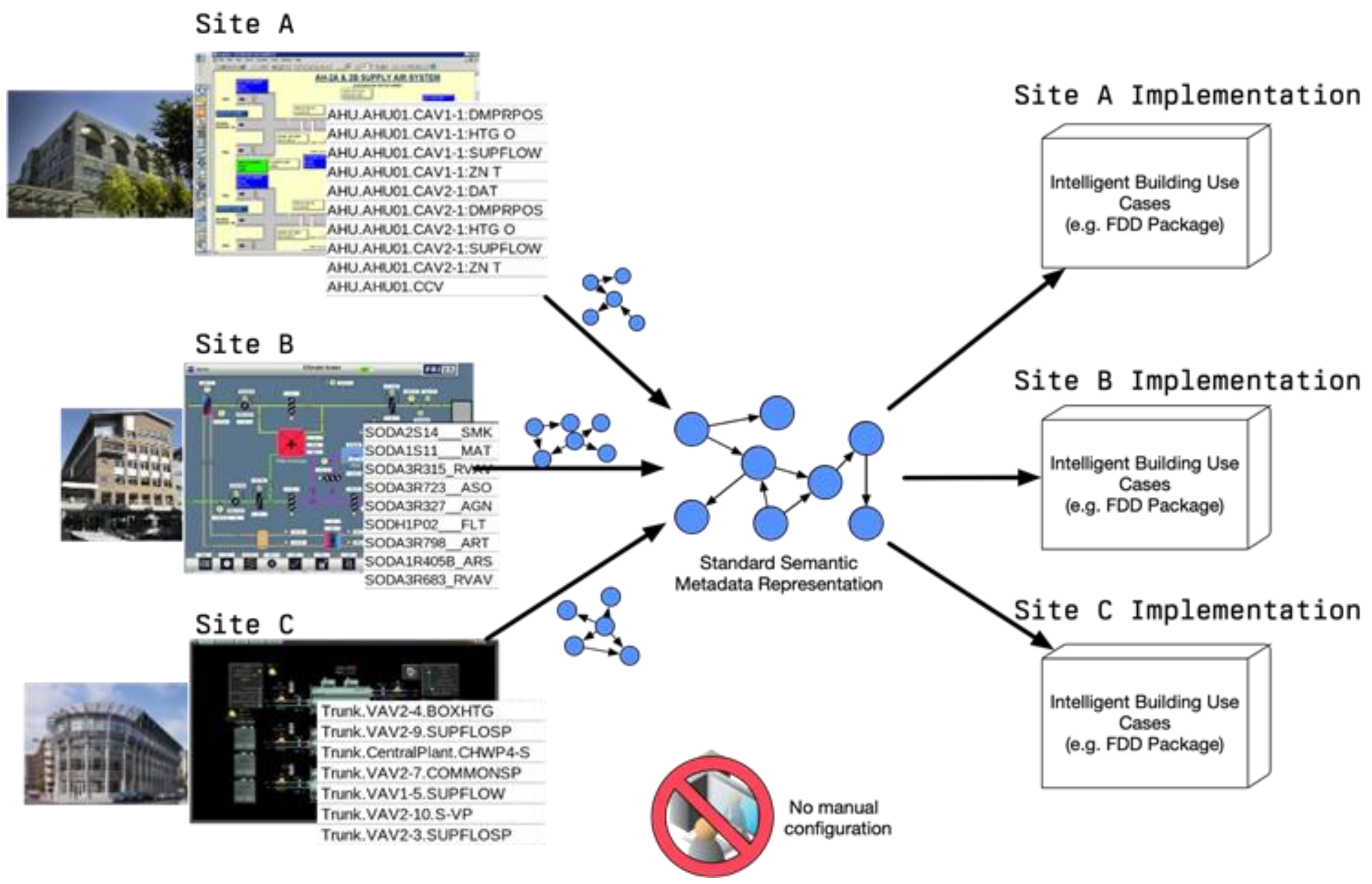


Configuration of applications requires discovery of data sources

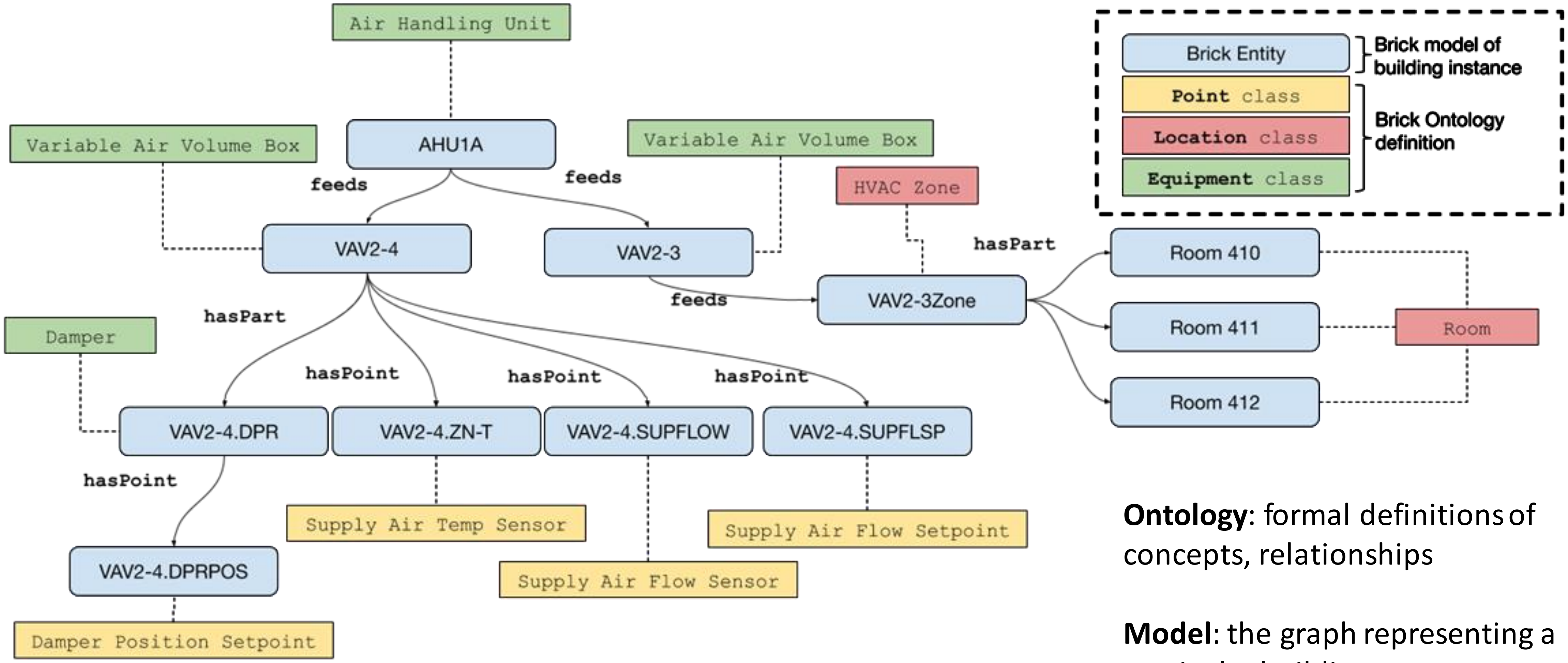
- Lack of standard representations of buildings during their operation
  - e.g., ad-hoc naming conventions for BAS/BMS points
- Existing representations not fit-for-purpose
  - e.g., BIM for geometry, not control loops

Result: manual site-specific configuration required

Standard Semantic Metadata Simplifies Configuration Effort

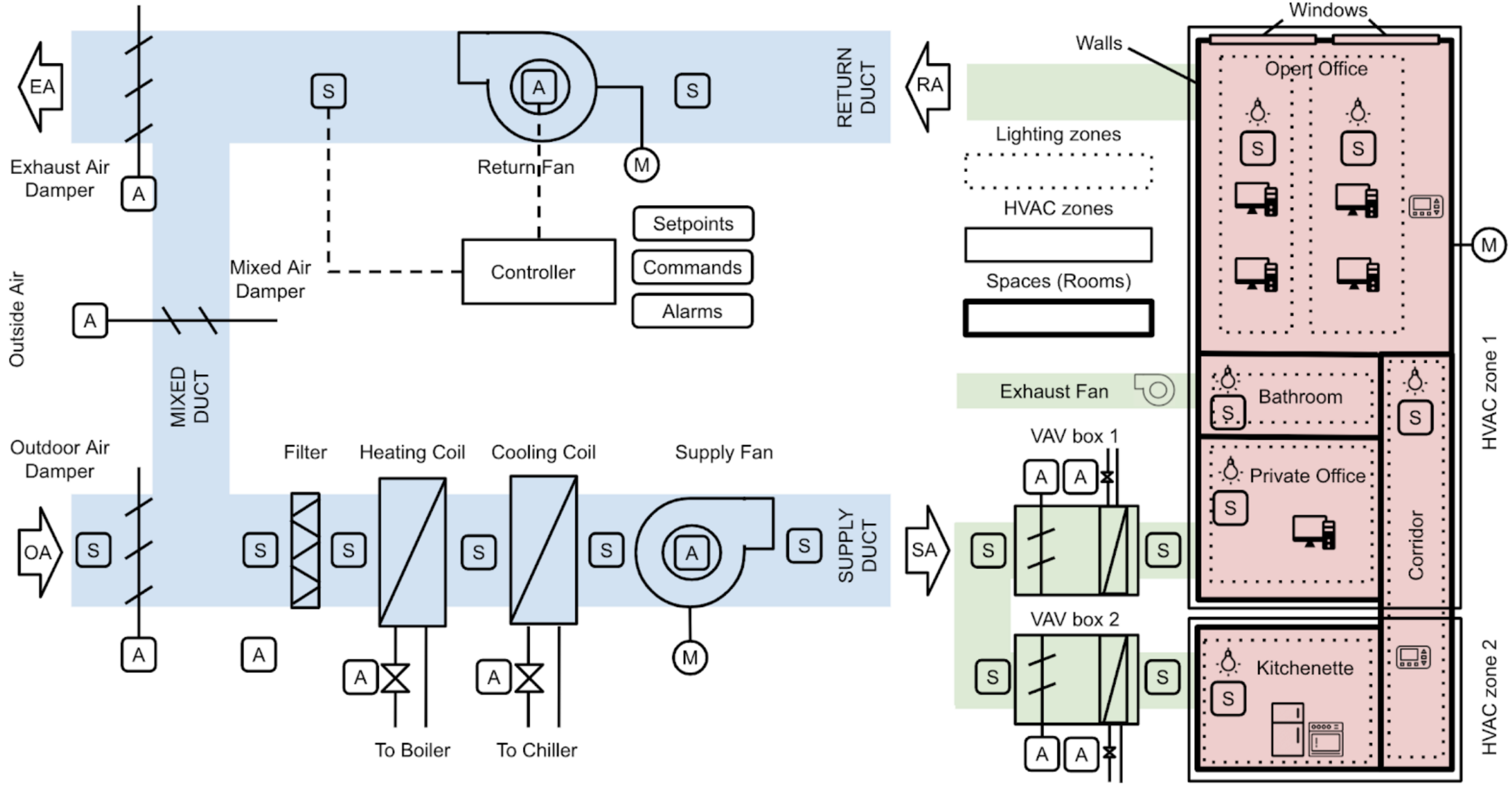


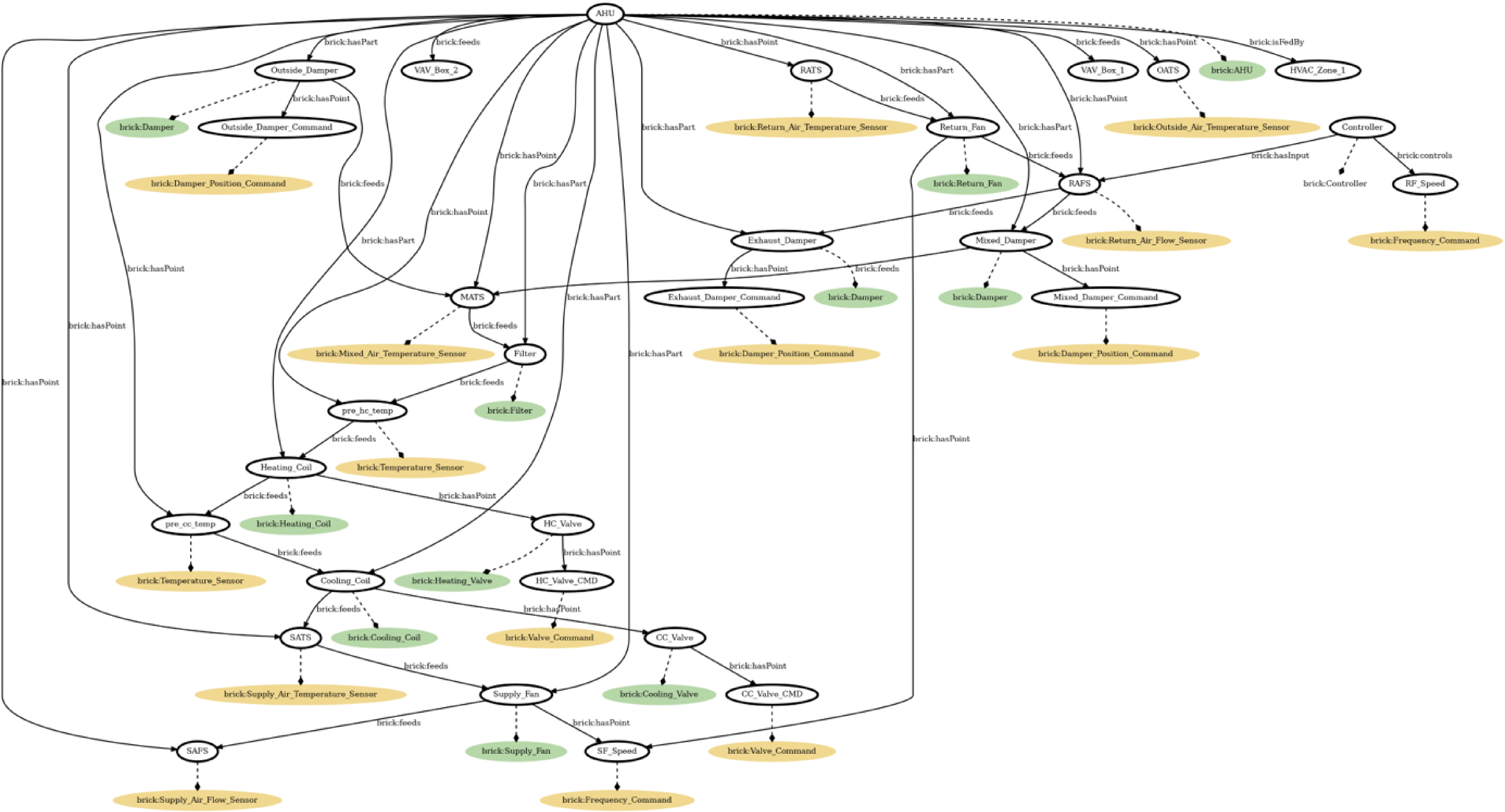
- Standard semantic metadata provides human- and machine- readable representation of:
  - Building data points
  - Building equipment
  - Building subsystems
  - Etc
- Significantly reduces site-specific configuration effort
- Enables automated reasoning and configuration



Ontology: formal definitions of concepts, relationships

Model: the graph representing a particular building

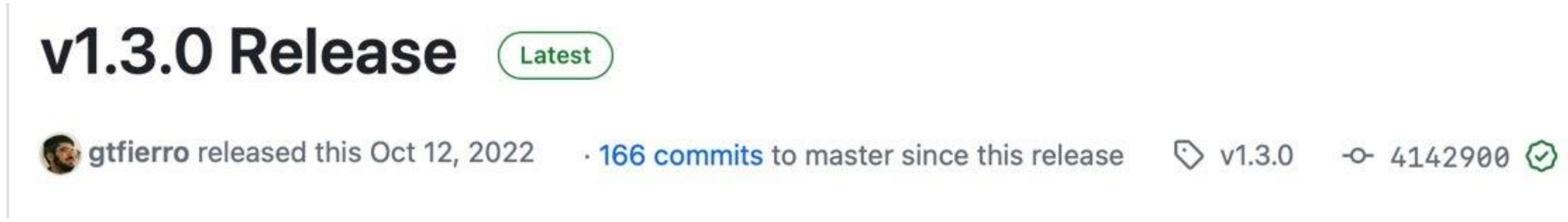

EA
Exhaust Air Damper
A
S
A
Return Fan
M
RETURN DUCT
S
RA
Setpoints
Commands
Alarms
Controller
Mixed Air Damper
Outside Air
A
MIXED DUCT
Outdoor Air Damper
OA
S
A
S
Filter
S
Heating Coil
S
Cooling Coil
S
Supply Fan
A
S
SUPPLY DUCT
M
A
A
A
To Boiler
To Chiller
SA
Windows
Walls
Open Office
Lighting zones
HVAC zones
Spaces (Rooms)
Exhaust Fan
Bathroom
Private Office
Corridor
Kitchenette
VAV box 1
VAV box 2
HVAC zone 1
HVAC zone 2
M
S
A


AHU
Outside_Damper
VAV_Box_2
RATS
VAV_Box_1
OATS
brick:AHU
HVAC_Zone_1
brick:Damper
Outside_Damper_Command
brick:Return_Air_Temperature_Sensor
Return_Fan
brick:Outside_Air_Temperature_Sensor
Controller
brick:Damper_Position_Command
brick:Return_Fan
RAFS
brick:Controller
RF_Speed
Exhaust_Damper
Mixed_Damper
brick:Return_Air_Flow_Sensor
brick:Frequency_Command
MATS
Exhaust_Damper_Command
brick:Damper
brick:Damper
Mixed_Damper_Command
brick:Mixed_Air_Temperature_Sensor
Filter
brick:Damper_Position_Command
brick:Damper_Position_Command
pre_hc_temp
brick:Filter
Heating_Coil
brick:Temperature_Sensor
pre_cc_temp
brick:Heating_Coil
HC_Valve
brick:Temperature_Sensor
Cooling_Coil
brick:Heating_Valve
HC_Valve_CMD
SATS
brick:Cooling_Coil
brick:Valve_Command
CC_Valve
brick:Supply_Air_Temperature_Sensor
Supply_Fan
brick:Cooling_Valve
CC_Valve_CMD
SAFS
brick:Supply_Fan
SF_Speed
brick:Valve_Command
brick:Supply_Air_Flow_Sensor
brick:Frequency_Command
brick:hasPart
brick:feeds
brick:hasPoint
brick:isFedBy
brick:hasInput
brick:controls

Brick Updates

Brick 1.4 Release Candidate

- Will be out for public review under a new tag today/tomorrow
  - Watch for post on Google Group
- Will contain
  - List of changes
  - Downloadable files of the release
  - Example models



v1.3.0 Release Latest

gtfierro released this Oct 12, 2022 · 166 commits to master since this release v1.3.0 4142900

New Concepts!

- New sensors, setpoints, commands
- Information/communication technology (ICT) concepts
    - Thanks, RealEstateCore!
    - Including sensor equipment for physical placement
- Storage tanks
- More PV stuff
- VRF, Heat Pumps and Chillers
- New terminal units, dampers, pumps
- Switchgears, switches, breakers

Consistency Fixes

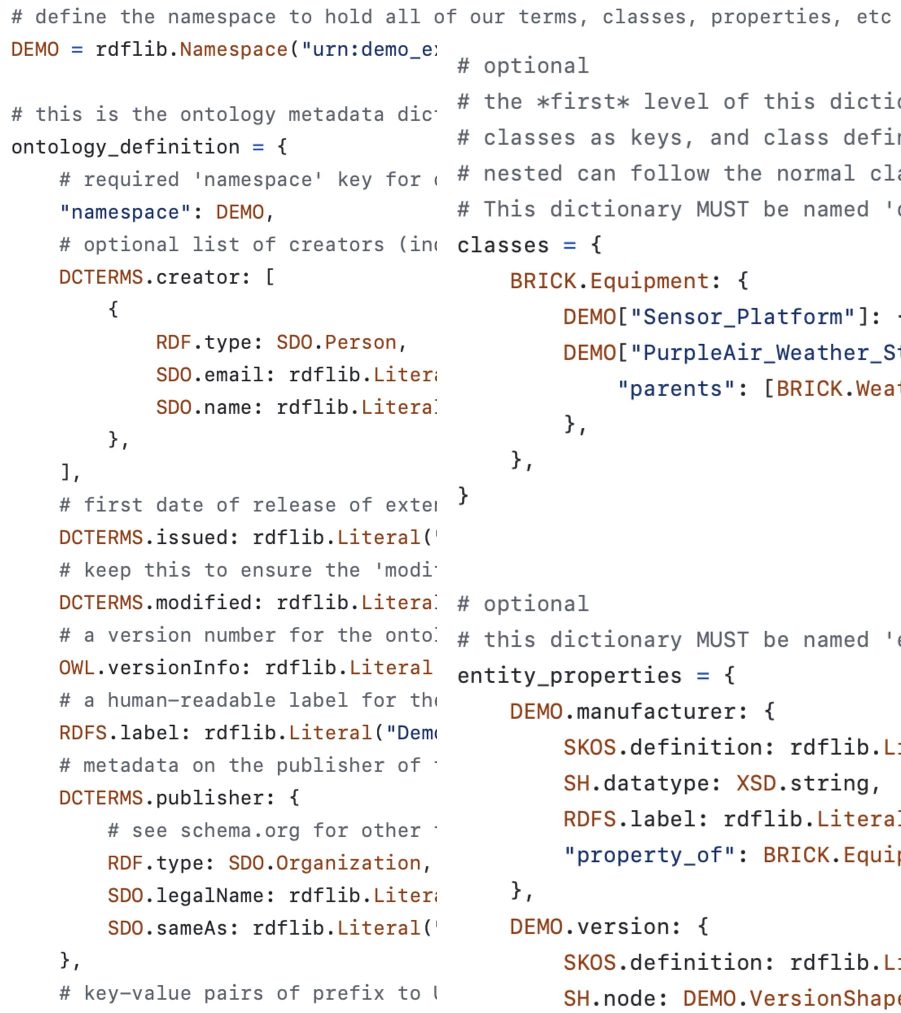
- Using QUDT types/concepts where possible
  - Interim: a few Brick-only "quantity kinds", e.g. for HVAC
  - Greater consistency with future ASHRAE 223P standard
- Improved use of SHACL for validation/inference
  - Enable future integration with ASHRAE 223P
  - Enable future integration with Project Haystack "Xeto" type system
- hasSubstance/hasQuantity annotations for Sensor, Setpoint, Command
- Rearrangement of setpoints, sensors, and some other class trees
  - Thanks Mapped!
  - Much more consistent
- And others! Please see the release notes

New Modeling Concepts

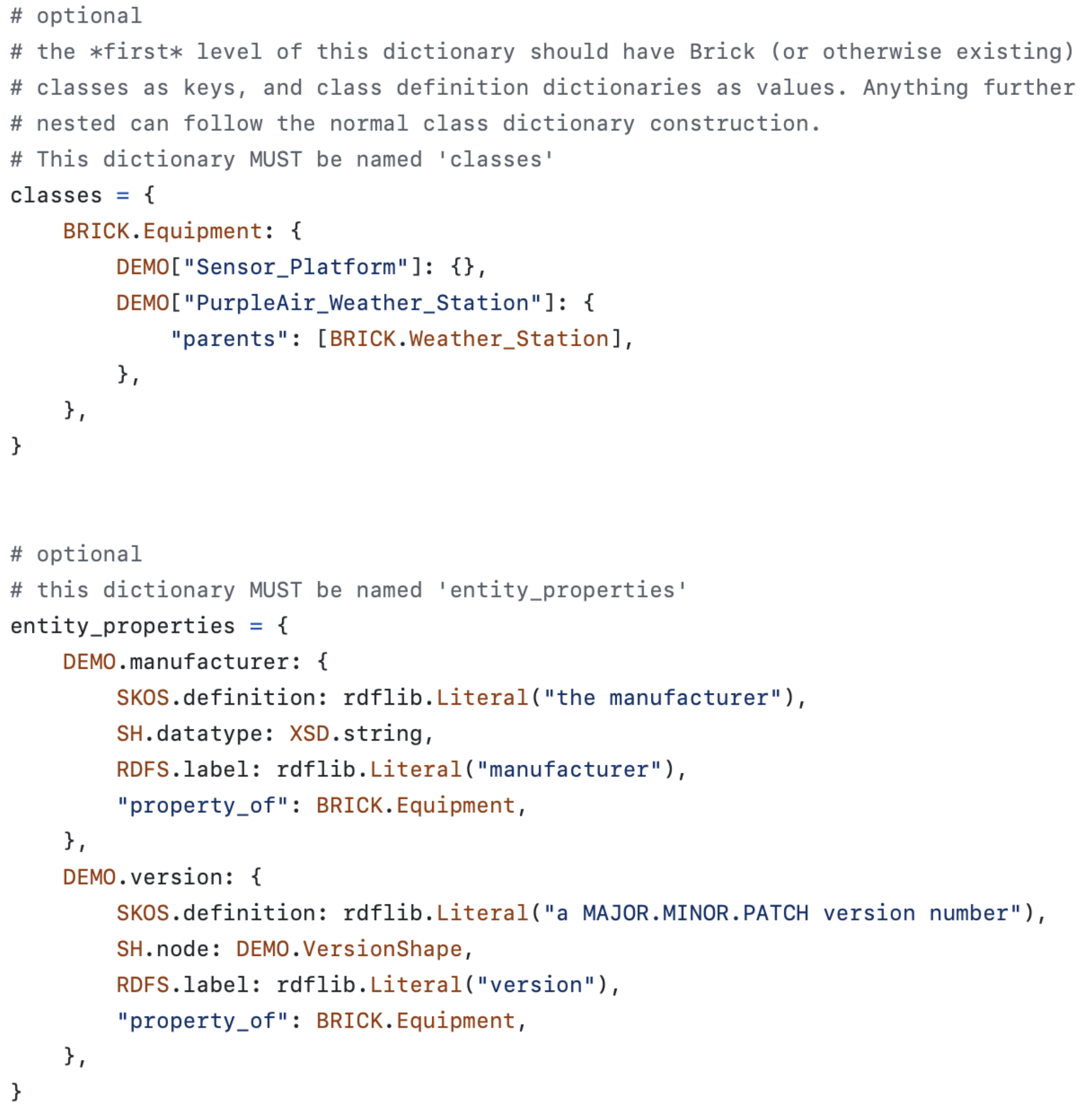
- **Aliases**: allow for "preferred" classes
    - e.g., Variable Air Volume Box over VAV
- Improved handling of deprecated classes, concepts
- Easier Brick extensions
    - More on this next
- Alignment with RealEstateCore
- Groundwork for future alignment with ASHRAE 223P

Easier Extensions for Brick

- Can now define your extensions in the same Python data structures as the core Brick distribution
- Ontology metadata
- Classes, entity properties, relationships, constraints, etc



```
# define the namespace to hold all of our terms, classes, properties, etc
DEMO = rdflib.Namespace("urn:demo_e

# this is the ontology metadata dic
ontology_definition = {
    # required 'namespace' key for
    "namespace": DEMO,
    # optional list of creators (in
    DCTERMS.creator: [
        {
            RDF.type: SDO.Person,
            SDO.email: rdflib.Liter
            SDO.name: rdflib.Literal
        },
    ],
    # first date of release of exte
    DCTERMS.issued: rdflib.Literal(
    # keep this to ensure the 'modi
    DCTERMS.modified: rdflib.Literal
    # a version number for the onto
    OWL.versionInfo: rdflib.Literal
    # a human-readable label for th
    RDFS.label: rdflib.Literal("Dem
    # metadata on the publisher of
    DCTERMS.publisher: {
        # see schema.org for other
        RDF.type: SDO.Organization,
        SDO.legalName: rdflib.Liter
        SDO.sameAs: rdflib.Literal(
    },
    # key-value pairs of prefix to
```



```
# optional
# the *first* level of this dictionary should have Brick (or otherwise existing)
# classes as keys, and class definition dictionaries as values. Anything further
# nested can follow the normal class dictionary construction.
# This dictionary MUST be named 'classes'
classes = {
    BRICK.Equipment: {
        DEMO["Sensor_Platform"]: {},
        DEMO["PurpleAir_Weather_Station"]: {
            "parents": [BRICK.Weather_Station],
        },
    },
}


# optional
# this dictionary MUST be named 'entity_properties'
entity_properties = {
    DEMO.manufacturer: {
        SKOS.definition: rdflib.Literal("the manufacturer"),
        SH.datatype: XSD.string,
        RDFS.label: rdflib.Literal("manufacturer"),
        "property_of": BRICK.Equipment,
    },
    DEMO.version: {
        SKOS.definition: rdflib.Literal("a MAJOR.MINOR.PATCH version number"),
        SH.node: DEMO.VersionShape,
        RDFS.label: rdflib.Literal("version"),
        "property_of": BRICK.Equipment,
    },
}
```

Steps Towards Semantic Interoperability

- Brick and RealEstateCore
- Brick and Project Haystack
- Brick and ASHRAE 223P
- Supporting tools (BuildingMOTIF)

Brick and RealEstateCore

```
bldg: a  owl:Ontology ;
    owl:imports <https://brickschema.org/schema/1.3/Brick> .

bldg:sample-device a bacnet:BACnetDevice ;
    bacnet:device-instance 123 ;
    bacnet:hasPort [ a bacnet:Port ] .

bldg:sensor1 a brick:Air_Temperature_Sensor ;
    brick:hasUnit unit:DEG_F ;
    ref:hasExternalReference [
        bacnet:object-identifier "analog-value,1" ;
        bacnet:object-name "BLDG-Z410-ZATS" ;
        bacnet:objectOf bldg:sample-device ;
    ] ;
    brick:lastKnownValue bldg:sensor1_reading1 .

bldg:sensor1_reading1  a rec:TemperatureObservation ;
        brick:value "72.0"^^xsd:double ;
        brick:timestamp "2024-01-01T00:00:00Z"^^xsd:dateTime ;
        rec:sourcePoint bldg:sensor1 .
```

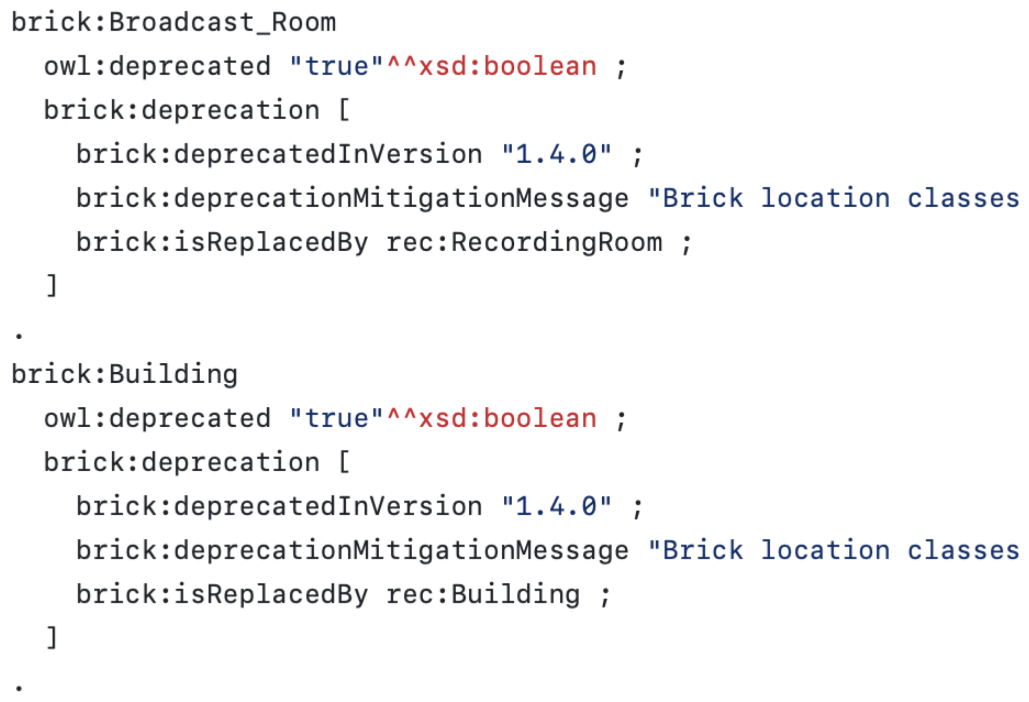
lastKnownValue relationship

```
brick:Temperature_Sensor
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path brick:lastKnownValue ;
      sh:class rec:TemperatureObservation ;
      sh:maxCount 1 ;
      sh:name "last known value" ;
      sh:nodeKind sh:IRI ;
    ] ;
.
```

Add REC Observation Types automatically

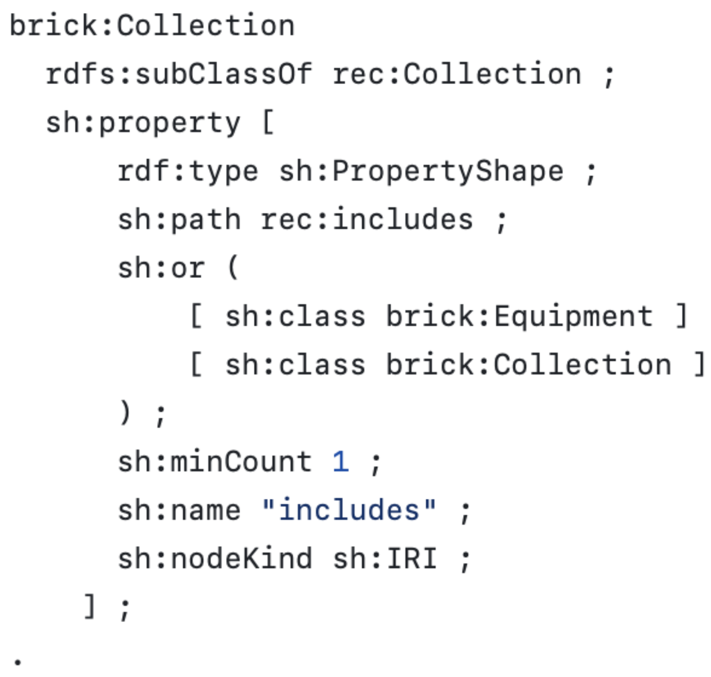
- Implemented in SHACL (*brickpatches.ttl*) in Brick repo
- Rules do the hard work for you; *automated backwards compatibility*

Brick and RealEstateCore



```
brick:Broadcast_Room
  owl:deprecated "true"^^xsd:boolean ;
  brick:deprecation [
    brick:deprecatedInVersion "1.4.0" ;
    brick:deprecationMitigationMessage "Brick location classes
    brick:isReplacedBy rec:RecordingRoom ;
  ]
.
brick:Building
  owl:deprecated "true"^^xsd:boolean ;
  brick:deprecation [
    brick:deprecatedInVersion "1.4.0" ;
    brick:deprecationMitigationMessage "Brick location classes
    brick:isReplacedBy rec:Building ;
  ]
.
```

Replace location types automatically

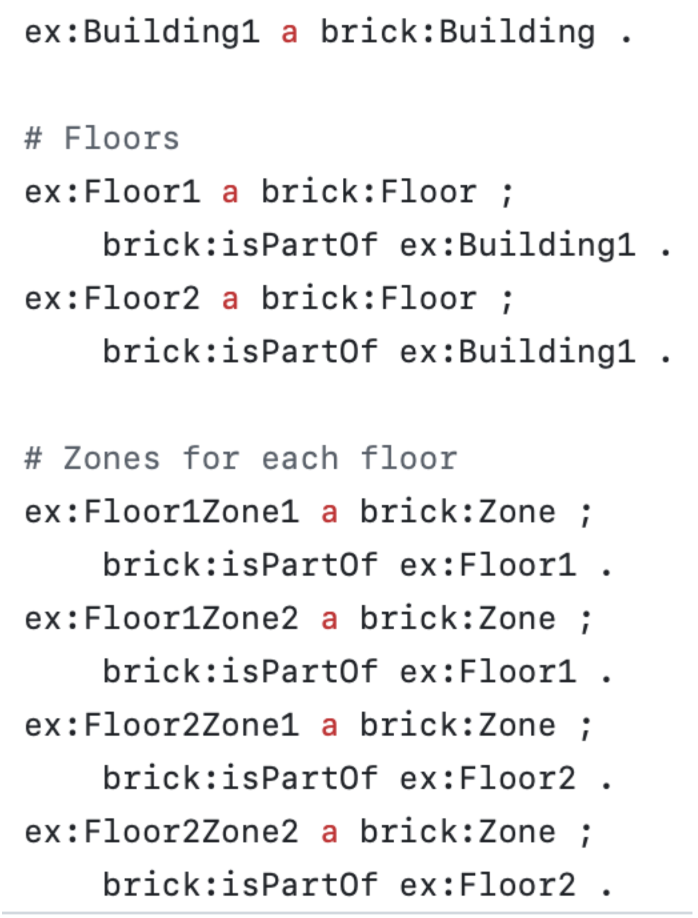


```
brick:Collection
  rdfs:subClassOf rec:Collection ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path rec:includes ;
      sh:or (
          [ sh:class brick:Equipment ]
          [ sh:class brick:Collection ]
      ) ;
      sh:minCount 1 ;
      sh:name "includes" ;
      sh:nodeKind sh:IRI ;
    ] ;
.
```

Infer REC properties/relationships when needed

- Implemented in SHACL (*brickpatches.ttl*) in Brick repo
- Rules do the hard work for you; *automated backwards compatibility*

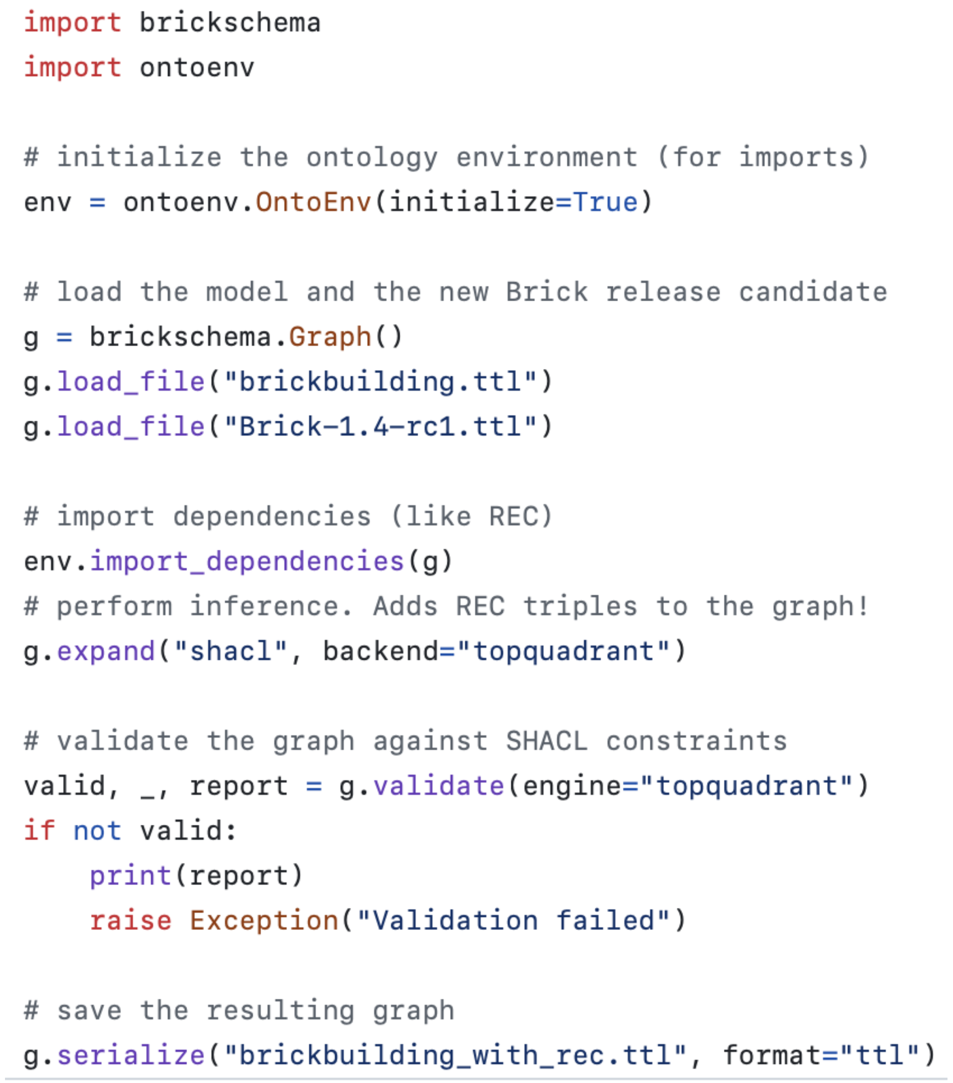
Brick + REC Model



```
ex:Building1 a brick:Building .

# Floors
ex:Floor1 a brick:Floor ;
    brick:isPartOf ex:Building1 .
ex:Floor2 a brick:Floor ;
    brick:isPartOf ex:Building1 .

# Zones for each floor
ex:Floor1Zone1 a brick:Zone ;
    brick:isPartOf ex:Floor1 .
ex:Floor1Zone2 a brick:Zone ;
    brick:isPartOf ex:Floor1 .
ex:Floor2Zone1 a brick:Zone ;
    brick:isPartOf ex:Floor2 .
ex:Floor2Zone2 a brick:Zone ;
    brick:isPartOf ex:Floor2 .
```

Brick-only model (v1.3 compliant)

Run script on model

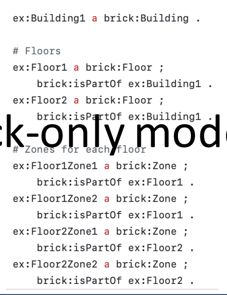


```
import brickschema
import ontoenv

# initialize the ontology environment (for imports)
env = ontoenv.OntoEnv(initialize=True)

# load the model and the new Brick release candidate
g = brickschema.Graph()
g.load_file("brickbuilding.ttl")
g.load_file("Brick-1.4-rc1.ttl")

# import dependencies (like REC)
env.import_dependencies(g)
# perform inference. Adds REC triples to the graph!
g.expand("shacl", backend="topquadrant")

# validate the graph against SHACL constraints
valid, _, report = g.validate(engine="topquadrant")
if not valid:
    print(report)
    raise Exception("Validation failed")

# save the resulting graph
g.serialize("brickbuilding_with_rec.ttl", format="ttl")
```

Brick + REC Model

```
ex:Building1 a brick:Building, rec:Building .

# Floors
ex:Floor1 a brick:Floor, rec:Storey;
    brick:isPartOf ex:Building1 ;
    rec:isPartOf ex:Building1 .
ex:Floor2 a brick:Floor, rec:Storey ;
    brick:isPartOf ex:Building1 ;
    rec:isPartOf ex:Building1 .

# Zones for each floor
ex:Floor1Zone1 a brick:Zone, rec:Zone ;
    brick:isPartOf ex:Floor1 ;
    rec:isPartOf ex:Floor1 .
ex:Floor1Zone2 a brick:Zone, rec:Zone ;
    brick:isPartOf ex:Floor1 ;
    rec:isPartOf ex:Floor1 .
ex:Floor2Zone1 a brick:Zone, rec:Zone ;
    brick:isPartOf ex:Floor2 ;
    rec:isPartOf ex:Floor2 .
ex:Floor2Zone2 a brick:Zone, rec:Zone ;
    brick:isPartOf ex:Floor2 ;
    rec:isPartOf ex:Floor2 .
```

Brick+REC model (classes *and* relationships)



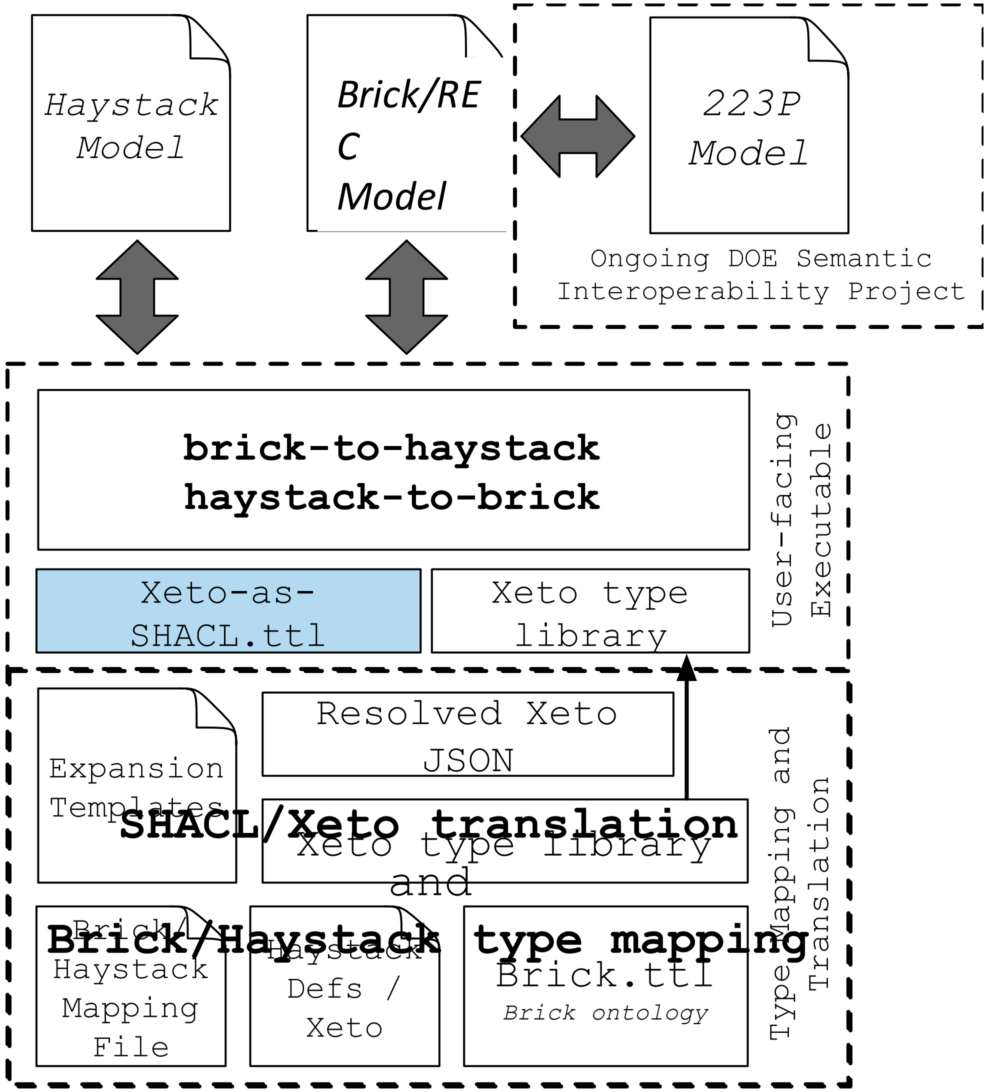
```
ex:Building1 a brick:Building .

# Floors
ex:Floor1 a brick:Floor ;
    brick:isPartOf ex:Building1 .
ex:Floor2 a brick:Floor ;
    brick:isPartOf ex:Building1 .

# Zones for each floor
ex:Floor1Zone1 a brick:Zone ;
    brick:isPartOf ex:Floor1 .
ex:Floor1Zone2 a brick:Zone ;
    brick:isPartOf ex:Floor1 .
ex:Floor2Zone1 a brick:Zone ;
    brick:isPartOf ex:Floor2 .
ex:Floor2Zone2 a brick:Zone ;
    brick:isPartOf ex:Floor2 .
```

Brick-only model

Run script on model

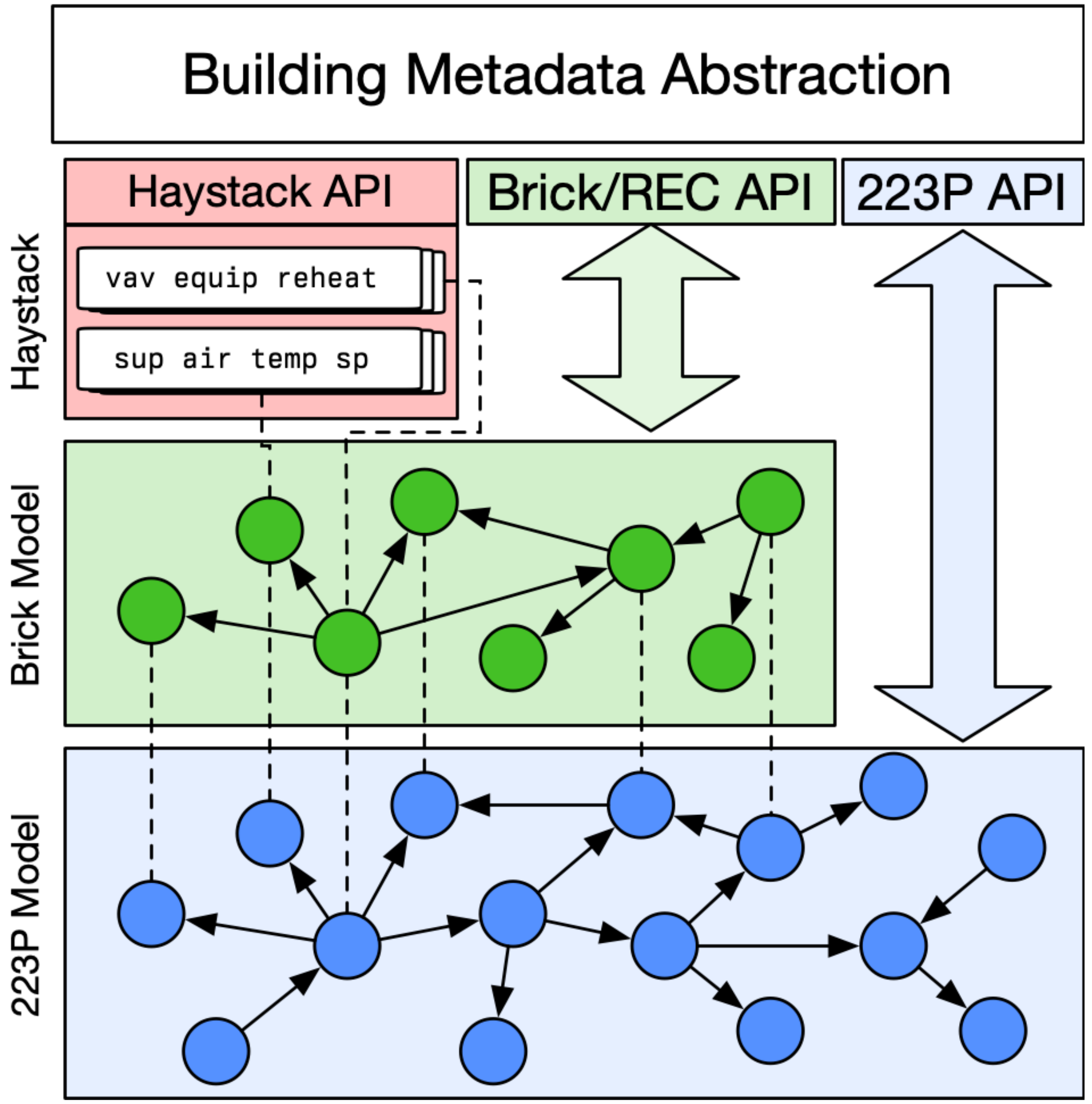
```python
import brickschema
import ontoenv

# initialize the ontology environment (for imports)
env = ontoenv.OntoEnv(initialize=True)

# load the model and the new Brick release candidate
g = brickschema.Graph()
g.load_file("brickbuilding.ttl")
g.load_file("Brick-1.4-rc1.ttl")

# import dependencies (like REC)
env.import_dependencies(g)
# perform inference. Adds REC triples to the graph!
g.expand("shacl", backend="topquadrant")

# validate the graph against SHACL constraints
valid, _, report = g.validate(engine="topquadrant")
if not valid:
    print(report)
    raise Exception("Validation failed")

# save the resulting graph
g.serialize("brickbuilding_with_rec.ttl", format="ttl")
```

Brick and Project Haystack



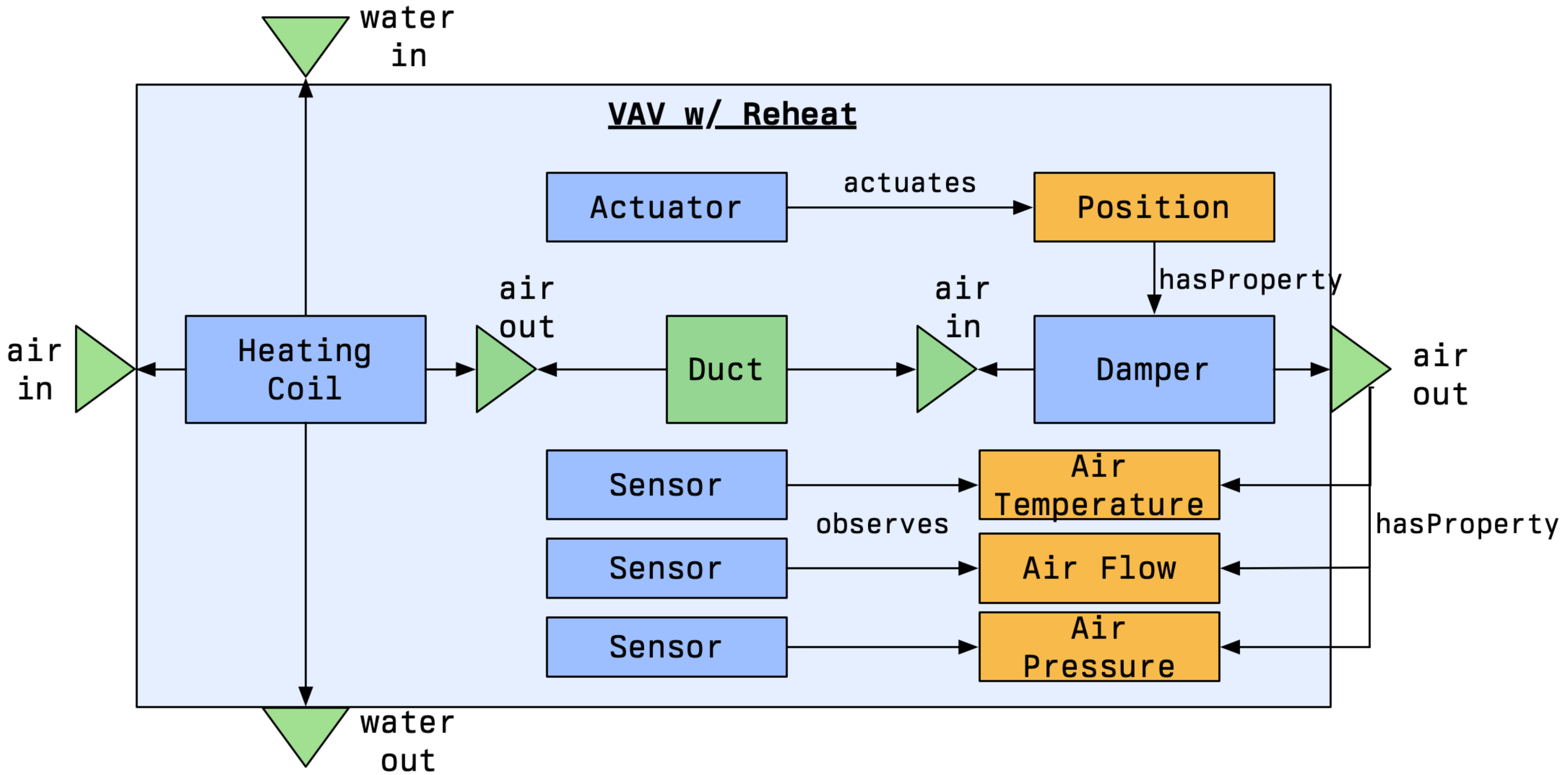
- Building on new "Xeto" PH type system
- Haystack → RDF:
  - Produce Brick/REC and 223P descriptions of Haystack entities
  - <u>Validate Haystack models against 223P standard</u>
- RDF → Haystack:
  - Use Xeto type system to add Haystack tags to Brick/REC, 223P entities
  - <u>Produce valid Haystack models from valid Brick/REC, 223P models</u>
- Along the way:
  - Resolve differences between Brick/Haystack types

Brick and 223P (and REC and Haystack)



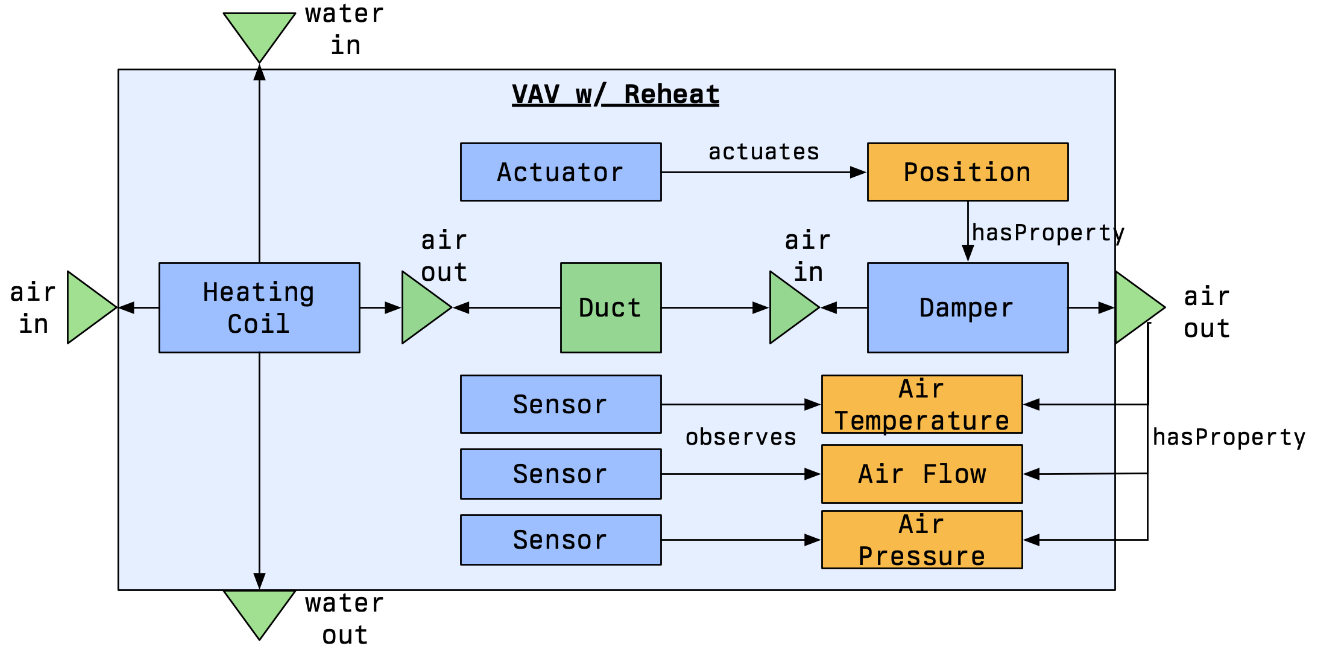
- 223P models the fundamental components of a building
- Brick/REC provide human-facing application-centric vocabulary
  - **Programmatically generated from 223P**
- Haystack provides one possible high-level abstraction over RDF / graph models
  - **Programmatically generated from Brick/REC**

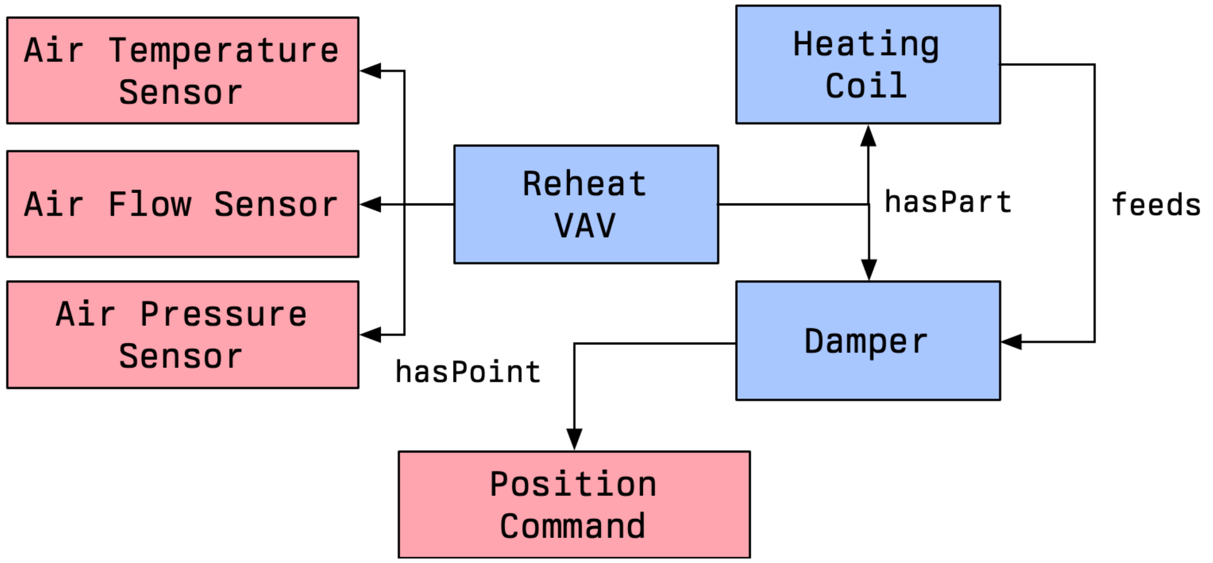
VAV w/ Reheat: 223P



- Directionality, substance, properties propagated through process flows
- Equipment composition, explicit sensor observation relationships
- *Some details removed for ease of visualization*

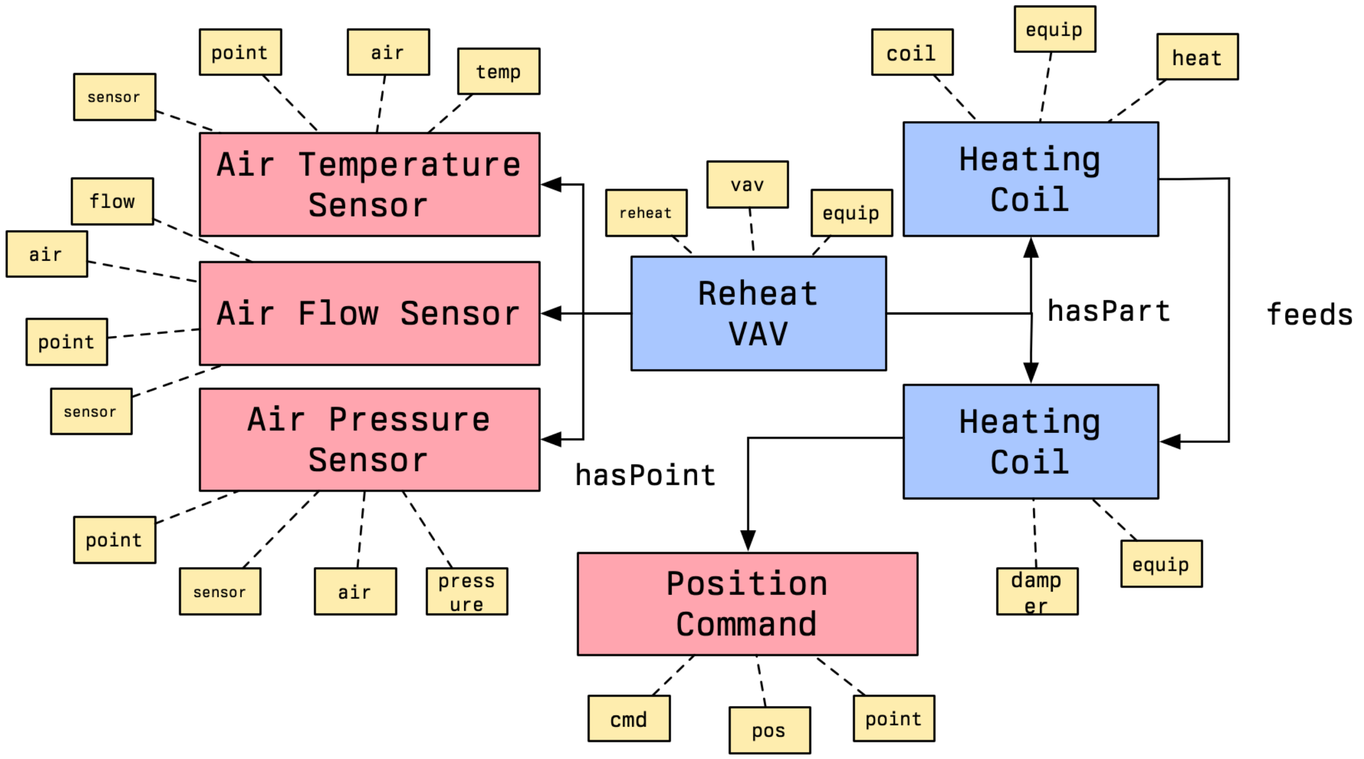
VAV w/ Reheat: 223P → Brick/REC

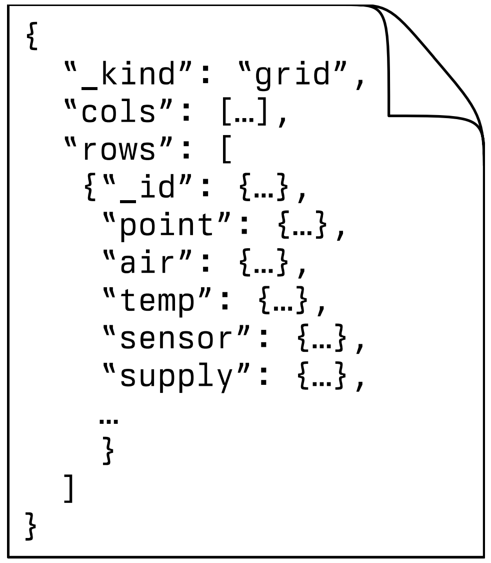




- Brick/REC model captures composition, flow, relationship to BMS points
  - Simplification of the 223P model with more specific names
- Brick/REC model is <u>programmatically generated</u> from the 223P model
  - Uses SHACL to infer types, add necessary relationships

VAV w/ Reheat: 223P → Brick/REC → Haystack

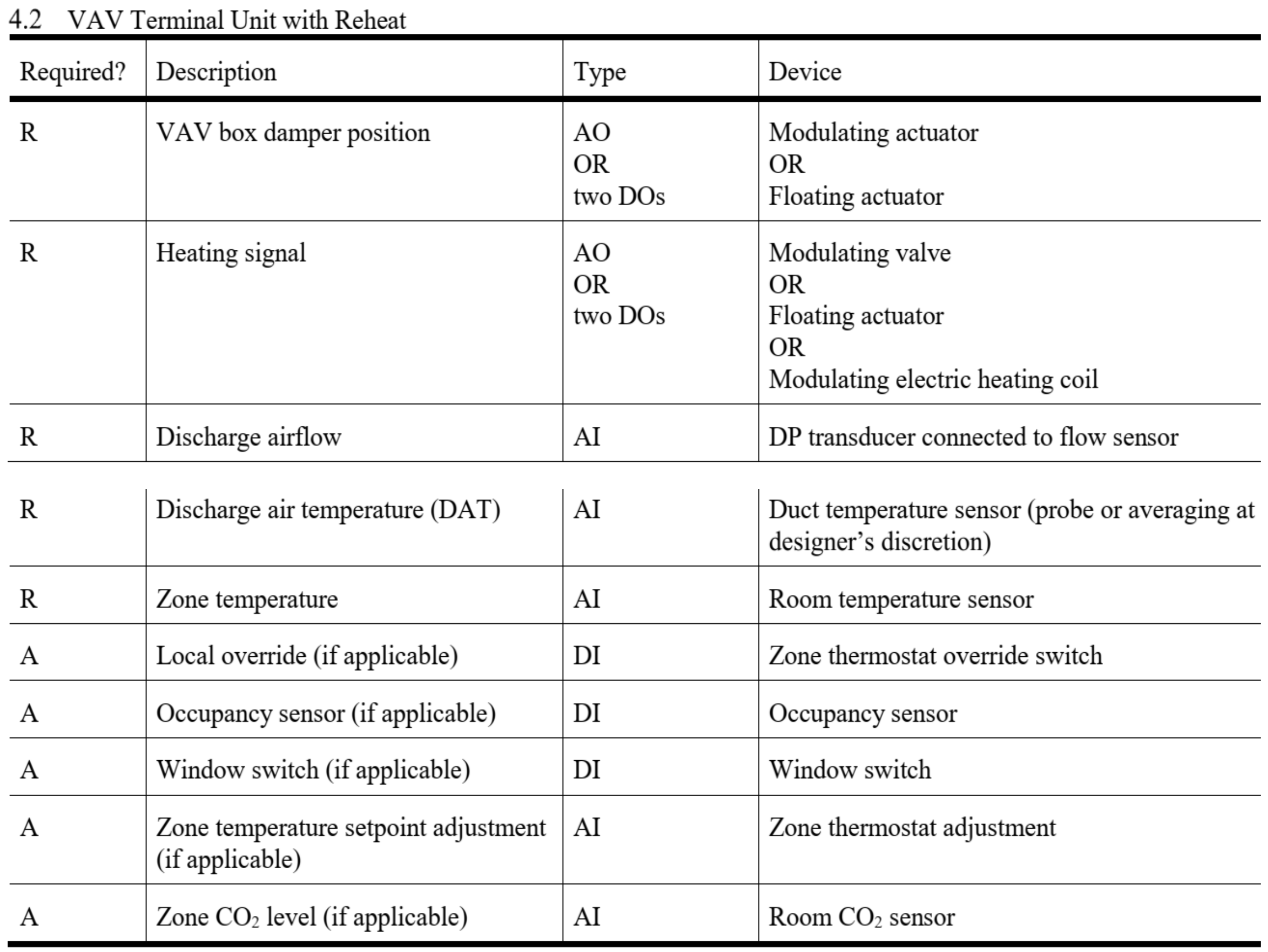




```
{
  "_kind": "grid",
  "cols": […],
  "rows": [
   {"_id": {…},
    "point": {…},
    "air": {…},
    "temp": {…},
    "sensor": {…},
    "supply": {…},
    …
    }
  ]
}
```

- Application of SHACL rules adds Haystack tags to Brick/REC entities
  - Rules constructed automatically from Xeto type definitions
  - Adds ref tags and (*soon!*) value tags
- Haystack import formats (e.g. JSON) generated from the augmented RDF model

Shapes: Formalizing Application Metadata Requirements



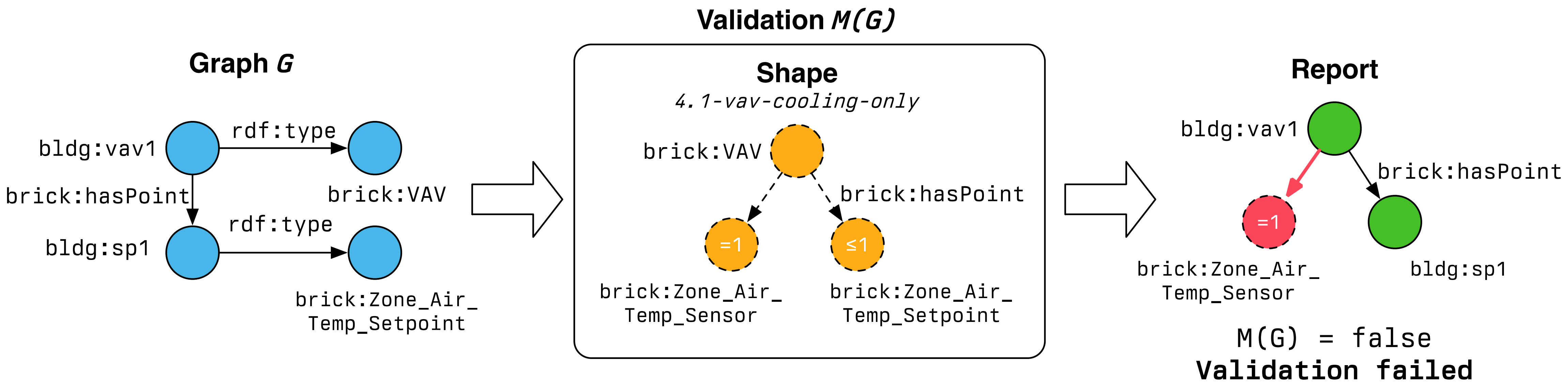
4.2 VAV Terminal Unit with Reheat

Required?	Description	Type	Device
R	VAV box damper position	AO OR two DOs	Modulating actuator OR Floating actuator
R	Heating signal	AO OR two DOs	Modulating valve OR Floating actuator OR Modulating electric heating coil
R	Discharge airflow	AI	DP transducer connected to flow sensor
R	Discharge air temperature (DAT)	AI	Duct temperature sensor (probe or averaging at designer's discretion)
R	Zone temperature	AI	Room temperature sensor
A	Local override (if applicable)	DI	Zone thermostat override switch
A	Occupancy sensor (if applicable)	DI	Occupancy sensor
A	Window switch (if applicable)	DI	Window switch
A	Zone temperature setpoint adjustment (if applicable)	AI	Zone thermostat adjustment
A	Zone CO_2 level (if applicable)	AI	Room CO_2 sensor

- Come in many different forms…
- **At right**: point lists from ASHRAE Guideline 36
- SOO for high-perf forced-air systems
- How to ensure Brick model contains the right metadata?

Shapes: Formalizing Application Metadata Requirements

- **Shape**: set of constraints, requirements, conditions on parts of the graph
- Function returns T/F when evaluated on a graph
    - Plus some information about ”what went wrong”

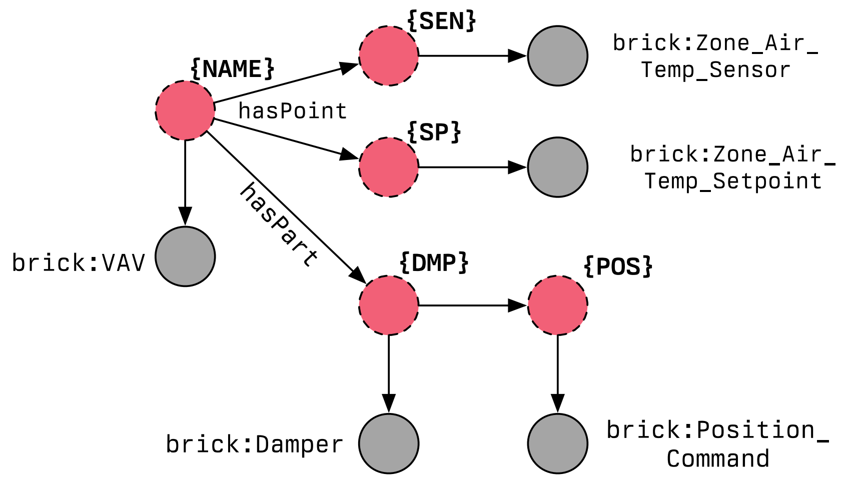


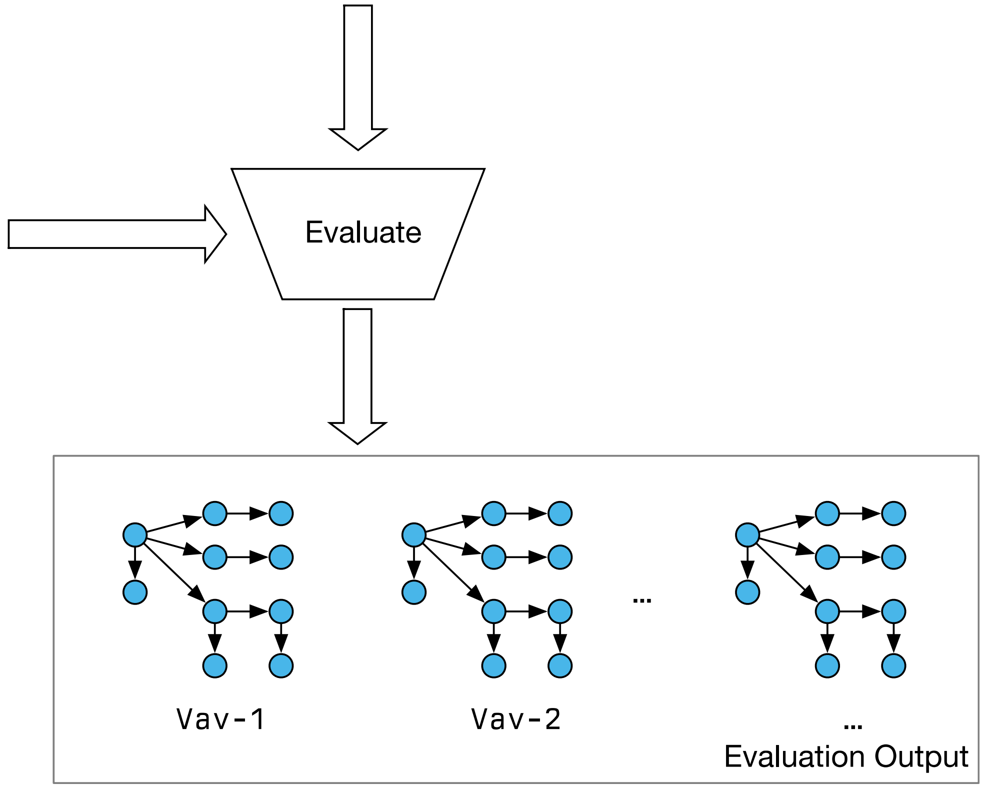
- Defined using W3C SHACL standard

Templates: Simplifying Model Creation

- **Template**: function that generates a graph

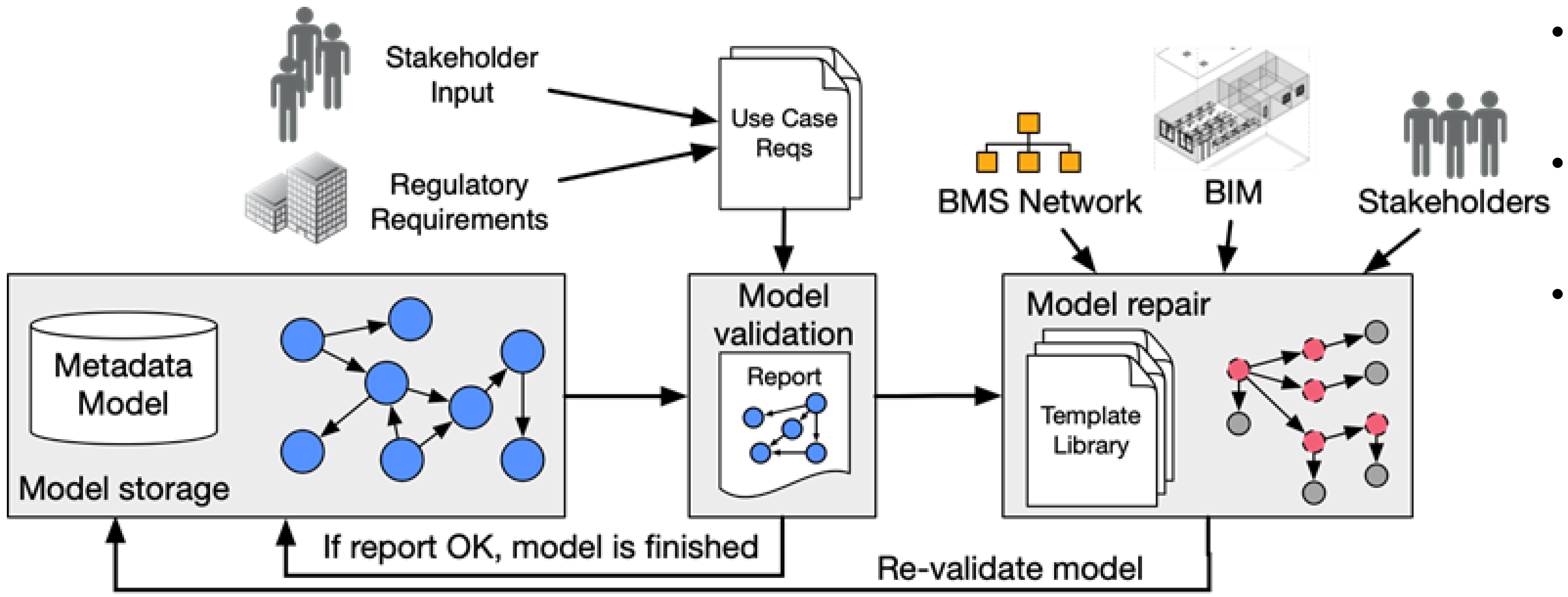
{NAME}	{SEN}	{SP}	{DMP}	{POS}
Vav-1	ZTS-A	ZTSP-A	DMP-12	POS-12
Vav-2	ZTS-B	ZTSP-B	DMP-23	POS-23
…	…	…	…	…





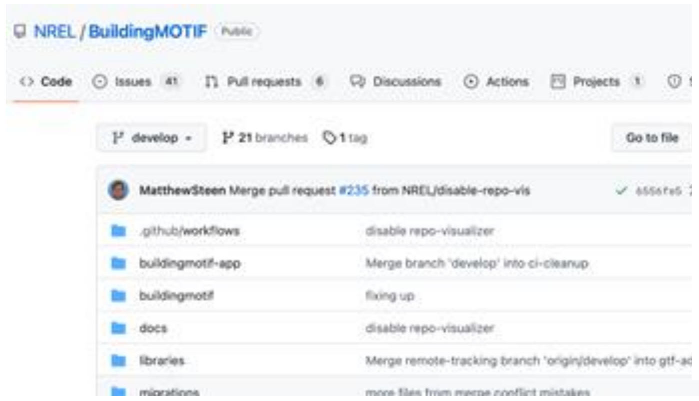
- Simplifies RDF graph creation
  - Tabular input: web forms! Spreadsheets!
  - Easier to integrate (pull from BACnet, etc)

BuildingMOTIF SDK



- Incorporate formal use case requirements into iterative workflow
- Ensure that delivered metadata model fulfills all use cases
- Automate / simplify authoring through templates, imports from other sources

- Code open-sourced on GitHub
- Publication on underlying theory / algorithms in BuildSys 2022
- Currently supports Brick, REC and 223P

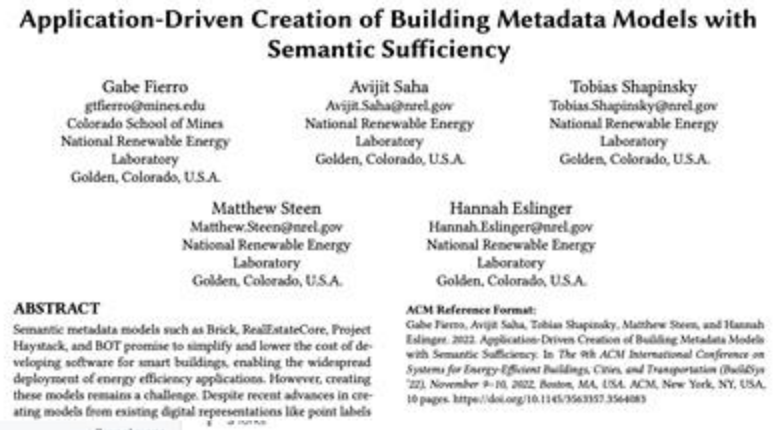

NREL / BuildingMOTIF Public

Code · Issues 41 · Pull requests 6 · Discussions · Actions · Projects 1

develop · 21 branches · 1 tag · Go to file

MatthewSteen Merge pull request #235 from NREL/disable-repo-vis ✓ 6556fe5

.github/workflows	disable repo-visualizer
buildingmotif-app	Merge branch 'develop' into ci-cleanup
buildingmotif	fixing up
docs	disable repo-visualizer
libraries	Merge remote-tracking branch 'origin/develop' into gtf-ac
migrations	more files from merge conflict mistakes


Application-Driven Creation of Building Metadata Models with Semantic Sufficiency

Gabe Fierro
gtfierro@mines.edu
Colorado School of Mines
National Renewable Energy Laboratory
Golden, Colorado, U.S.A.

Avijit Saha
Avijit.Saha@nrel.gov
National Renewable Energy Laboratory
Golden, Colorado, U.S.A.

Tobias Shapinsky
Tobias.Shapinsky@nrel.gov
National Renewable Energy Laboratory
Golden, Colorado, U.S.A.

Matthew Steen
Matthew.Steen@nrel.gov
National Renewable Energy Laboratory
Golden, Colorado, U.S.A.

Hannah Eslinger
Hannah.Eslinger@nrel.gov
National Renewable Energy Laboratory
Golden, Colorado, U.S.A.

ABSTRACT

Semantic metadata models such as Brick, RealEstateCore, Project Haystack, and BOT promise to simplify and lower the cost of developing software for smart buildings, enabling the widespread deployment of energy efficiency applications. However, creating these models remains a challenge. Despite recent advances in creating models from existing digital representations like point labels

ACM Reference Format:

Gabe Fierro, Avijit Saha, Tobias Shapinsky, Matthew Steen, and Hannah Eslinger. 2022. Application-Driven Creation of Building Metadata Models with Semantic Sufficiency. In *The 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '22), November 9–10, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3563357.3564083