Internet of Things

Metadata

Aslak Johansen asjo@mmmi.sdu.dk

Feb 8, 2021

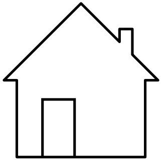
SDU

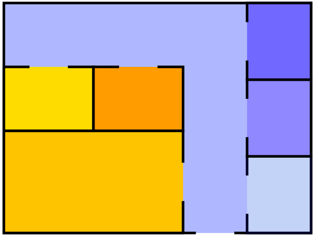
Part 0:
Introduction

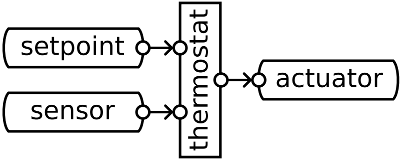
Motivation: A Cyber-Physical System

Consider a system consisting of

- A building with temperature sensors, actuators and setpoints per room
- A dashboard application showing a floormap colorcoded according to measured temperature
- A thermostat application which controls the temperature of each room







Motivation: Connecting the Data and Control Planes

Each point (sensor, actuator and setpoint) is named through a UUID but have different access rights.

How can we map the two applications to the points of a building?

1. Scatter constants throughout the code of each application
2. Introduce a per-application data structure:
   room name $\mapsto$ (sensor|actuator|setpoint) $\mapsto$ UUID
3. Share that structure

What happens when we add another building?

What happens when we add another application concerned with the power consumption of the lighting of a single floor of the building?

Problem

We want to write applications for buildings.

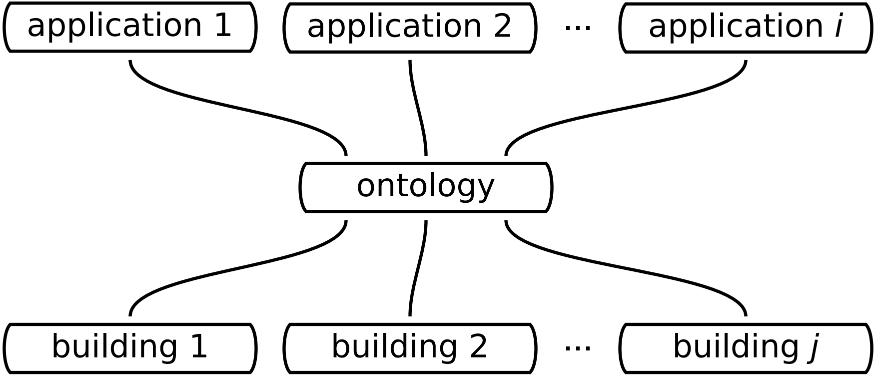
Attractive qualities:

- ***Portability*** An application can be executed on a many buildings without modification.
- ***Maintainability*** A change in a building does not translate to a need for changing an application.

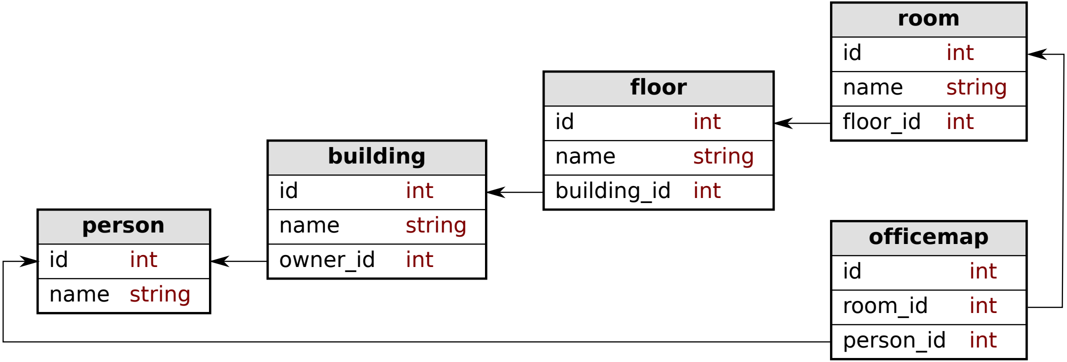
How do we accomplish this?

Approach: A Narrow Waist

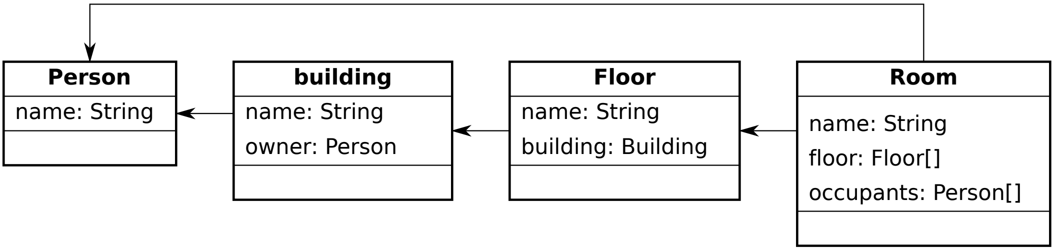
The labor intensive job of mapping out the equipment, data streams and relations between those may be shared between a portfolio of applications



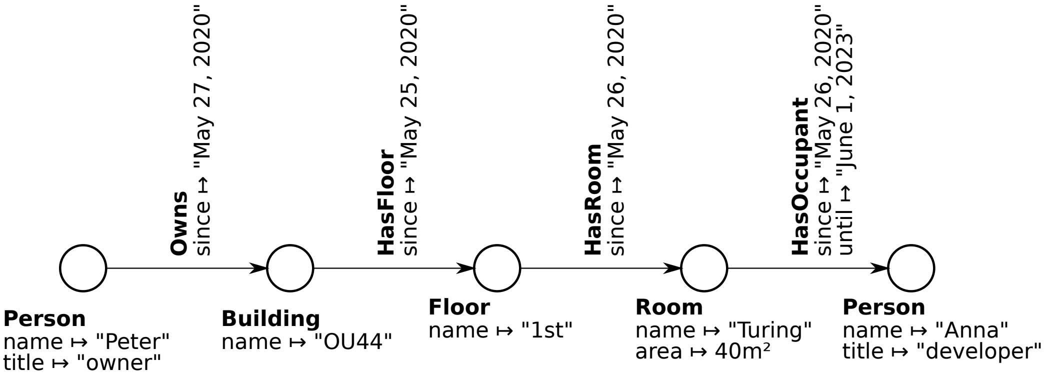
Relational Database



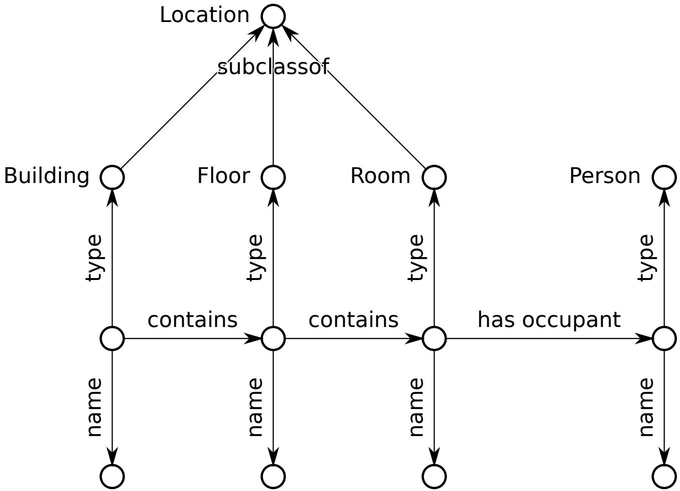
Object Database



Graph Database



Ontology-Based Information Model



Part 1:
Semantics

Context: Semantics

"cec78d2a-14b8-41d6-bf46-c6a4d07574a7 is in room U168"

What is the *intension* of this statement?

- *cec78d2a-14b8-41d6-bf46-c6a4d07574a7* is a UUID
- *cec78d2a-14b8-41d6-bf46-c6a4d07574a7* identifies some data stream
- *U168* identifies a room
- The data stream identified by *cec78d2a-14b8-41d6-bf46-c6a4d07574a7* originates from some equipment in the room identified by *U168*

One of these intensions is expressed, the others are merely implied

Ontologies

According to Wikipedia:

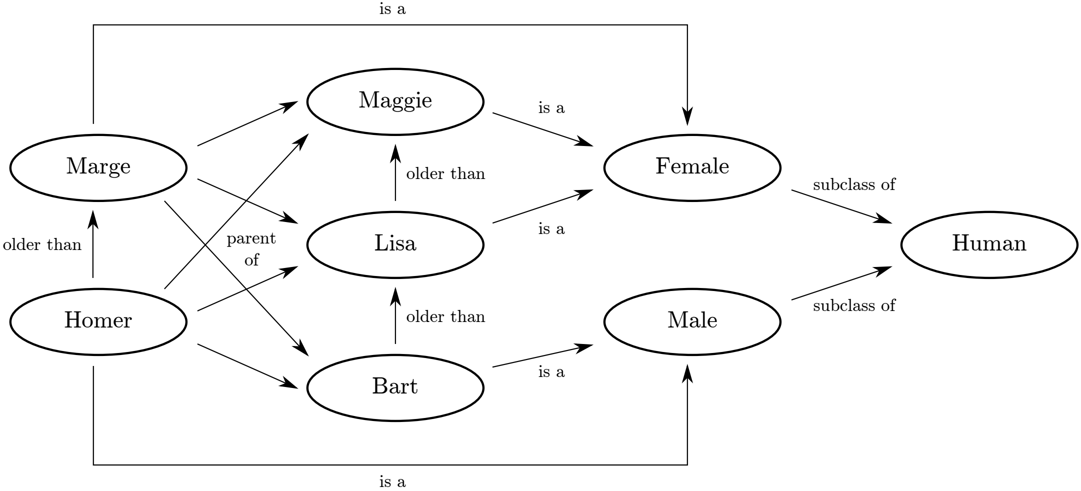
"In computer science and information science, an ontology is a ***formal naming and definition*** *of the* ***types, properties, and interrelationships of the entities*** *that really* ***exist in a particular domain*** *of discourse."*

Lets look into what this means :

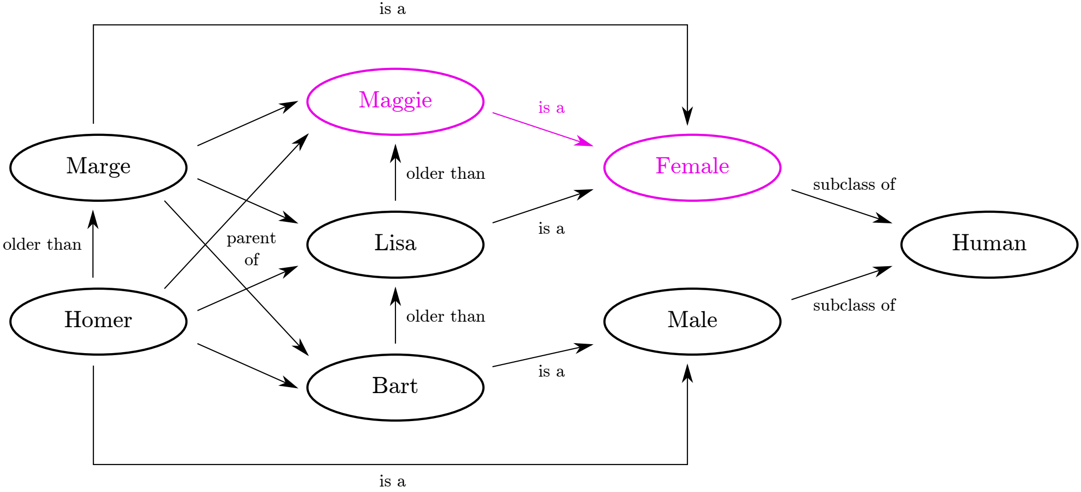
- ***Formal Naming and Definition*** A formalized language is employed
- ***Types, Properties and Interrelationships of Entities*** Things are defined through their type, parameters and relationships to other things
- ***Entities Existing in a Particular Domain*** Ontologies have limited coverage and focus on a particular field

Part 2:
The Resource Description Framework

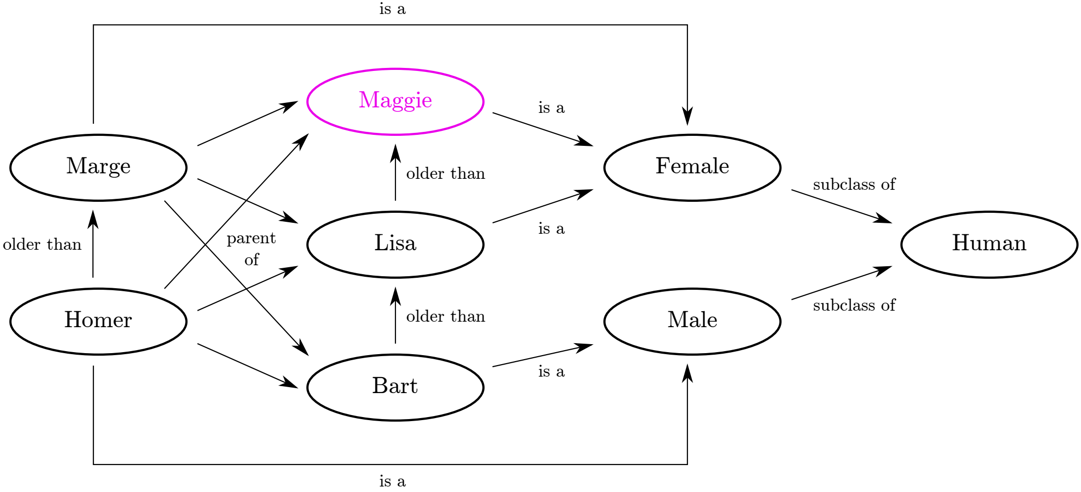
RDF Example


is a
Maggie
is a
Marge
Female
older than
subclass of
Lisa
Human
is a
older than
parent
of
Homer
older than
Male
subclass of
Bart
is a
is a

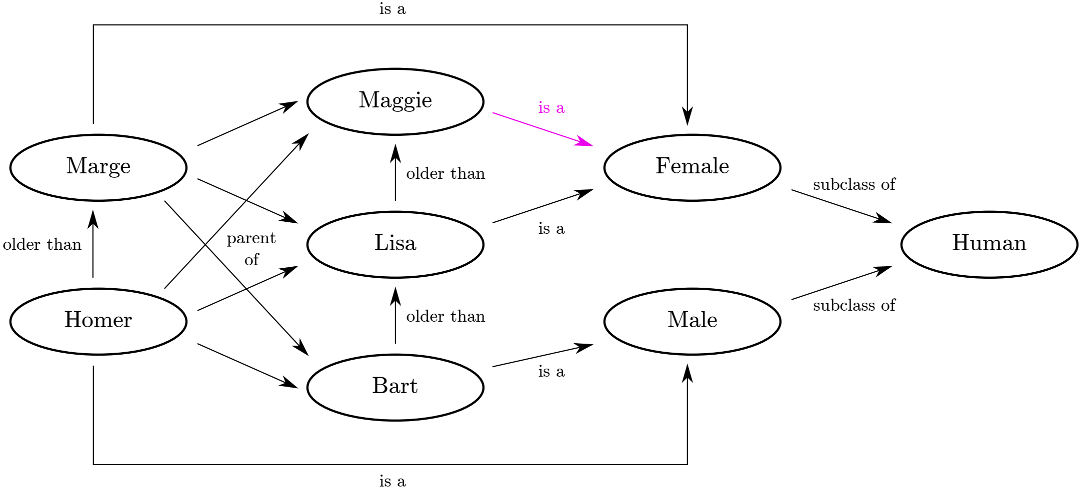
RDF Example


is a
Maggie
is a
Marge
Female
older than
subclass of
Lisa
is a
Human
older than
parent
of
Homer
older than
Male
subclass of
Bart
is a
is a

RDF Example

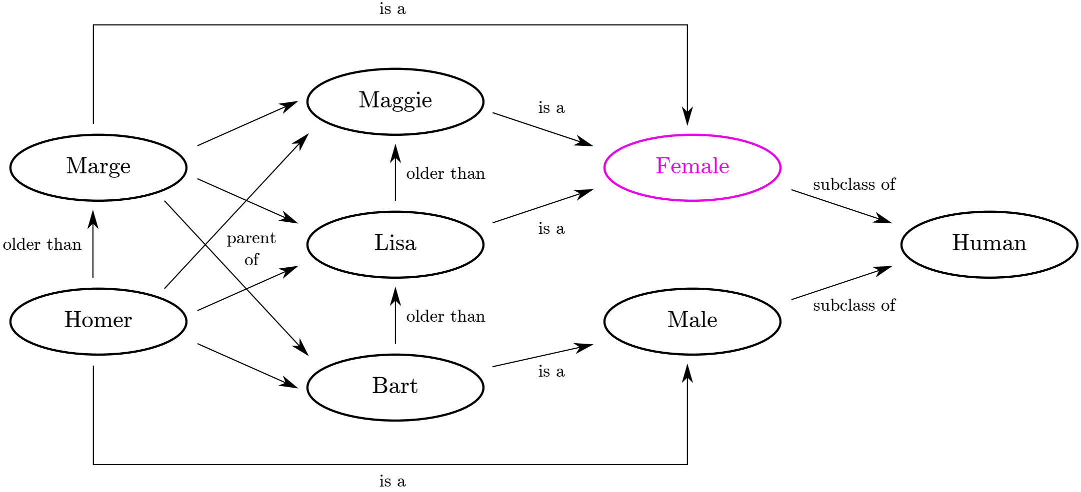

is a
Maggie
is a
Marge
older than
Female
subclass of
parent
of
Lisa
is a
Human
older than
Homer
older than
Male
subclass of
Bart
is a
is a

RDF Example



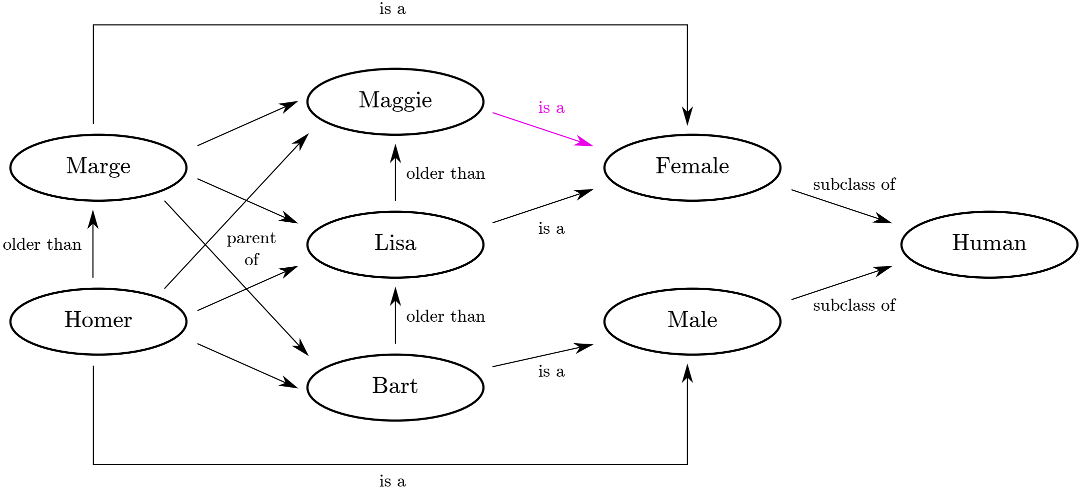
is a
Maggie
is a
Marge
Female
older than
subclass of
Lisa
is a
Human
older than
parent
of
Homer
older than
Male
subclass of
Bart
is a
is a

RDF Example



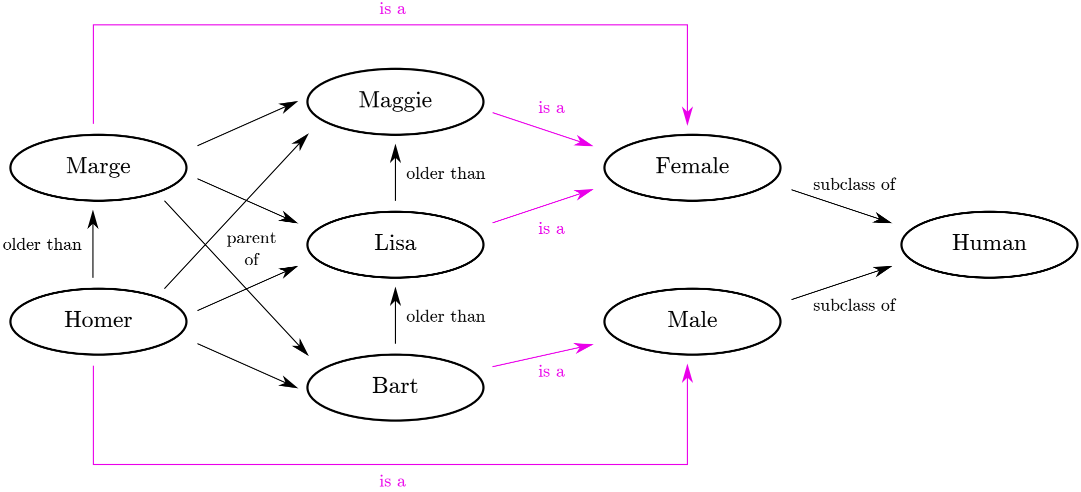
is a
Maggie
is a
Marge
older than
Female
subclass of
parent
of
Lisa
is a
Human
older than
Homer
older than
Male
subclass of
Bart
is a
is a

RDF Example



is a
Maggie
is a
Marge
Female
older than
subclass of
Lisa
Human
older than
parent
of
is a
Homer
older than
Male
subclass of
Bart
is a
is a

RDF Example



is a
Maggie
is a
Marge
Female
older than
subclass of
Lisa
is a
Human
older than
parent
of
Homer
older than
Male
subclass of
Bart
is a
is a

RDF - The Resource Description Framework

An ontology defined through triples

- Triples: subject $\times$ predicate $\times$ object
- A triple store is a model
- Namespaces
- Subclasses and subproperties

Wherein lies the information?

Hint: It is not in the name

In the relationships: A triple is a fact!

Subject	*Predicate*	*Object*
Marge	parent of	Bart
Marge	is a	Female
Bart	is a	Male
Female	subclass of	Human
Male	subclass of	Human

Turtle - The Terse RDF Triple Language

One (out of many) standardized ways of marshalling RDF.

File extension: `.ttl`

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix brick: <https://brickschema.org/schema/1.1.0/Brick#> .
@prefix demo:  <http://ontologies.sdu.dk/Demo#> .

demo:floor a brick:Floor .

demo:room a brick:Room ;
          rdfs:label "U168" ;
          brick:isLocatedIn demo:floor .
```

OWL: The Web $\leftrightarrow$ Ontology Language

A set of extensions to RDF

Adds formal semantics:

- ***Constraints*** e.g., on cardinality
- ***Relationship Inferences*** transitive, symmetric, inverse . . .
- ***Equivalence Testing*** whether two concepts are similar enough
- ***Subsumption Testing*** whether one concept is more general than another
- . . .

Closed world assumption: What is not known to be true must be false!

Part 3:
Modeling Buildings with Brick

Brick An Ontology for Building Metadata

Builds on OWL

Subject	**Predicate**	**Object**
`my:building`	`type`	`brick:Building`
`my:building/floor2`	`type`	`brick:Floor`
`my:building/room1`	`type`	`brick:Room`
`my:building`	`contains`	`my:building/floor2`
`my:building/floor2`	`contains`	`my:building/room1`
`my:building/room1`	`label`	`Literal("U168")`

Quizztime: What is the name of the building?

Brick Concepts

Definitions for

- Typing (Room, Temperature Sensor)
- Flows (feeds, feeds air)
- Physical encapsulation (contains)
- Control (controls)

Uncertainty

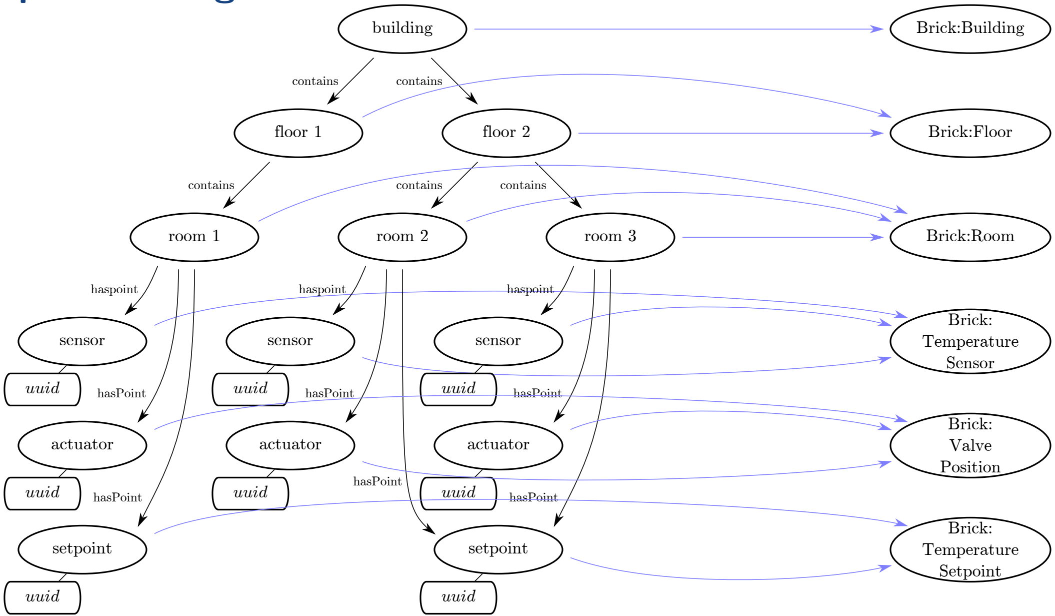
- Hierarchy of types (sensor $<$ temperature sensor $<$ celcius temperature sensor)

May be considered a graph ... but isn't

Quizz:

- What is a graph?
- How does a tripplestore differ?

Sample Building



Part 4:
Model Construction

Demo: Add Model

```
#!/usr/bin/env python3

from wrapping import *

g = model()
```

All code available at `https://github.com/aslakjohansen/sdu-iot-brick`

Demo: Add Building

```
building = N['/building']
g.add((building, RDF.type, BRICK['Building']))
```

Demo: Add Floors

```python
floors = []
for floor_number in range(2):
    floor = N['building/floors/%u' % floor_number]
    g.add((floor, RDF.type, BRICK['Floor']))
    g.add((building, BRICK.contains, floor))
    floors.append(floor)
```

Demo: Add Room Mapping

```
rooms = {
    'room 1': {
        'floor': 0,
        'temp-sensor':   'd0fcec33-af08-44a8-b74b-7a51f1902d13',
        'temp-actuator': '5c2e2bdb-5142-464d-ad91-ae483633dfd6',
        'temp-setpoint': 'c087d1ba-03f6-4c60-85de-b95d82e63248',
    },
    'room 2': {
        'floor': 1,
        'temp-sensor':   '18407632-8e81-4bb9-ae77-4ecc96f30f46',
        'temp-actuator': 'ec883cad-3235-46d5-a901-7bae169dda0a',
        'temp-setpoint': 'd2b54605-58b1-44ef-89f9-0a9adabbfbb1',
    },
    'room 3': {
        'floor': 1,
        'temp-sensor':   'e528f733-2e3a-4f65-93aa-6fb2e36d4b27',
        'temp-actuator': '9be52aed-0e12-49b5-8680-47778b9b2adf',
        'temp-setpoint': 'd2b54605-58b1-44ef-89f9-0a9adabbfbb1',
    },
}
```

Demo: Add Rooms

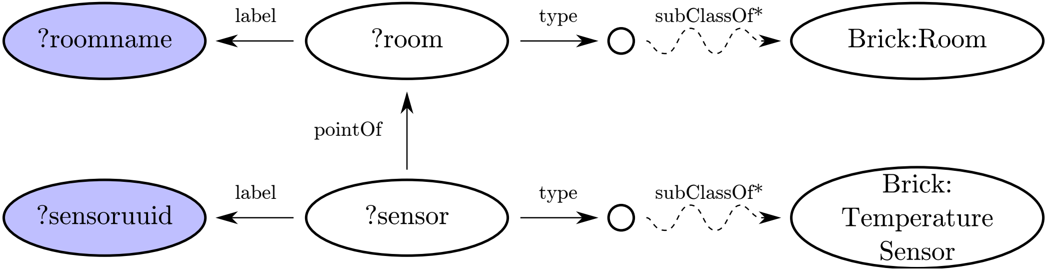
```
roommap = {}
for roomname in rooms:
    data = rooms[roomname]
    room = N['building/rooms/%s' % roomname.replace(' ', '_')]
    g.add((room, RDF.type, BRICK['Room']))
    g.add((room, BRICK.label, Literal(roomname)))
    g.add((floors[data['floor']], BRICK.contains, room))
    roommap[roomname] = room
```

Demo: Add Points

```
for roomname in rooms:
    data = rooms[roomname]
    room = roommap[roomname]

    sensor = N['building/rooms/%s/temp-sensor' % roomname.replace(' ', '_')]
    g.add((sensor, RDF.type, BRICK['Temperature_Sensor']))
    g.add((sensor, BRICK.label, Literal(data['temp-sensor'])))
    g.add((sensor, BRICK.pointOf, room))

    setpoint = N['building/rooms/%s/temp-setpoint' % roomname.replace(' ', '_')]
    g.add((setpoint, RDF.type, BRICK['Temperature_Setpoint']))
    g.add((setpoint, BRICK.label, Literal(data['temp-setpoint'])))
    g.add((setpoint, BRICK.pointOf, room))

    actuator = N['building/rooms/%s/temp-actuator' % roomname.replace(' ', '_')]
    g.add((actuator, RDF.type, BRICK['Radiator_Valve_Position']))
    g.add((actuator, BRICK.label, Literal(data['temp-actuator'])))
    g.add((actuator, BRICK.pointOf, room))
```

Part 5:
Querying a Model

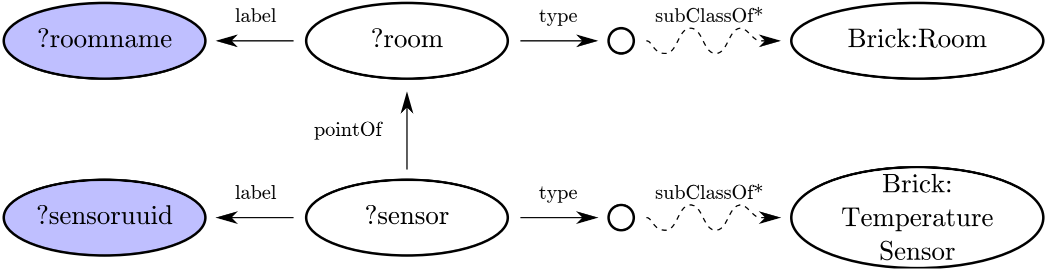
Pattern Matching: SparQL Query for Dashboard Application



Dashboard Query SparQL Form

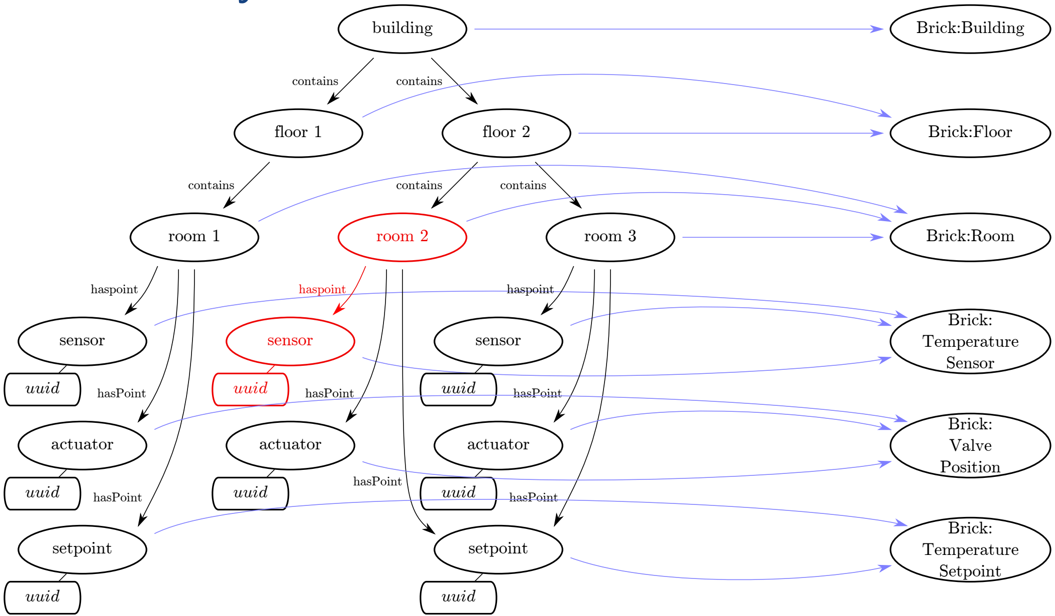
```
SELECT DISTINCT ?room_name ?sensor_uuid
WHERE {
    ?room     rdf:type/brick:subClassOf* brick:Room .
    ?sensor   rdf:type/brick:subClassOf* brick:Temperature_Sensor .

    ?sensor   brick:pointOf ?room .

    ?room     brick:label ?room_name .
    ?sensor   brick:label ?sensor_uuid .
}
```

Dashboard Query Visual Form

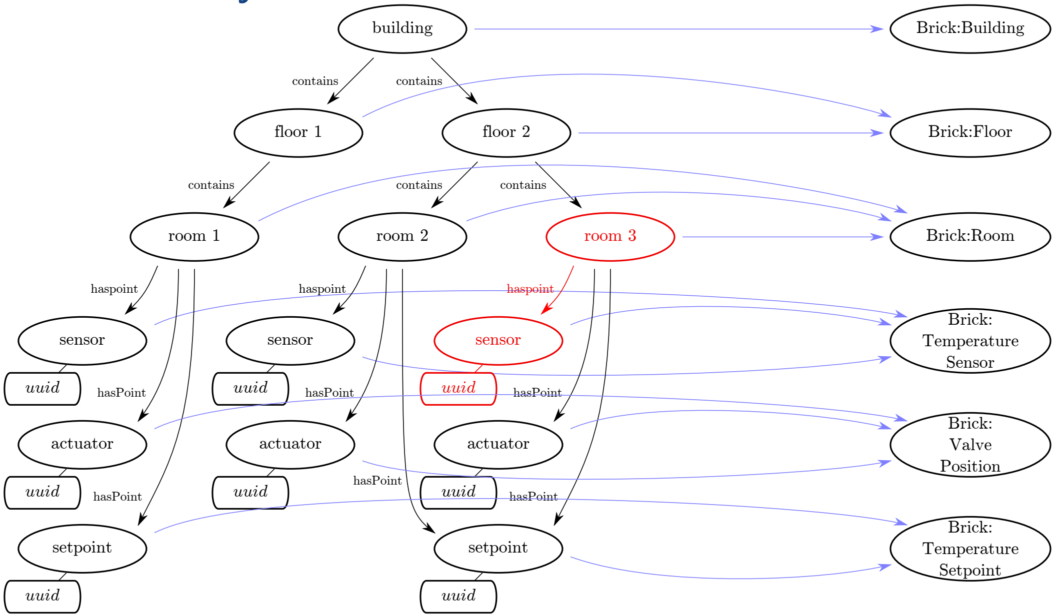


Dashboard Query Match 1

Dashboard Query Match 2



Dashboard Query Match 3



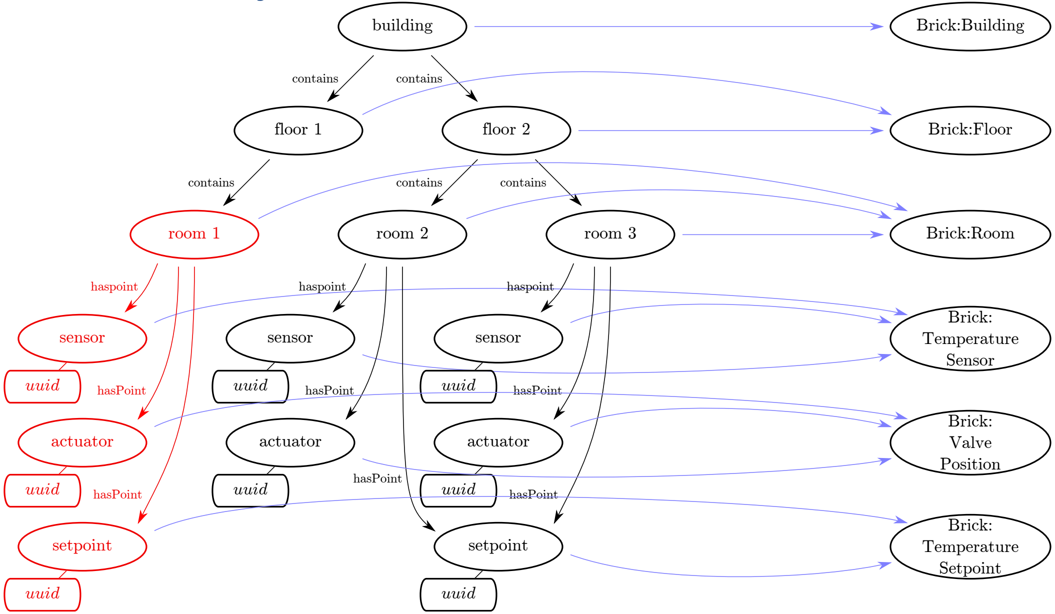
Dashboard Query Result Set


```
[
    [
        "room 1",
        "d0fcec33-af08-44a8-b74b-7a51f1902d13"
    ],
    [
        "room 2",
        "18407632-8e81-4bb9-ae77-4ecc96f30f46"
    ],
    [
        "room 3",
        "e528f733-2e3a-4f65-93aa-6fb2e36d4b27"
    ]
]
```

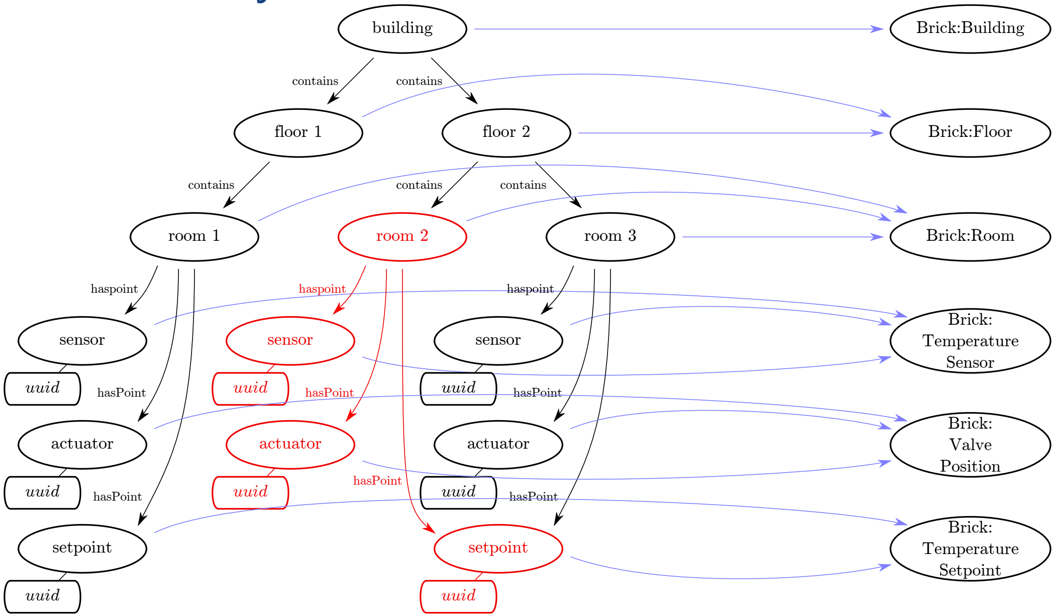
Demo: Thermostat Query

```
SELECT DISTINCT ?room_name ?sensor_uuid ?setpoint_uuid ?actuator_uuid
WHERE {
    ?room     rdf:type/brick:subClassOf* brick:Room .
    ?sensor   rdf:type/brick:subClassOf* brick:Temperature_Sensor .
    ?setpoint rdf:type/brick:subClassOf* brick:Temperature_Setpoint .
    ?actuator rdf:type/brick:subClassOf* brick:Radiator_Valve_Position .

    ?sensor   brick:pointOf ?room .
    ?setpoint brick:pointOf ?room .
    ?actuator brick:pointOf ?room .

    ?room     brick:label ?room_name .
    ?sensor   brick:label ?sensor_uuid .
    ?setpoint brick:label ?setpoint_uuid .
    ?actuator brick:label ?actuator_uuid .
}
```

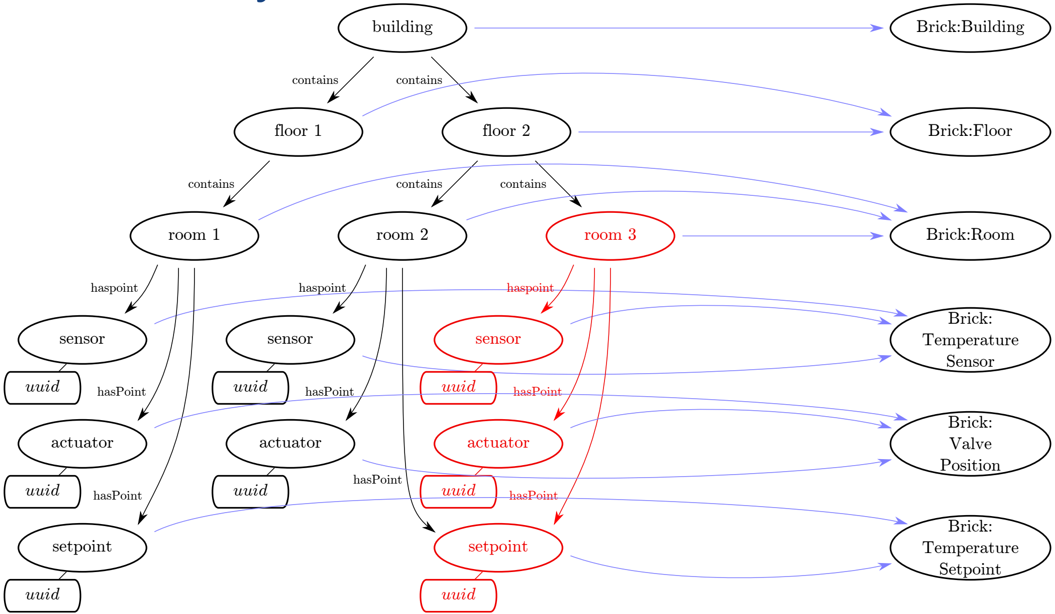
Thermostat Query Match 1



Thermostat Query Match 2



Thermostat Query Match 3



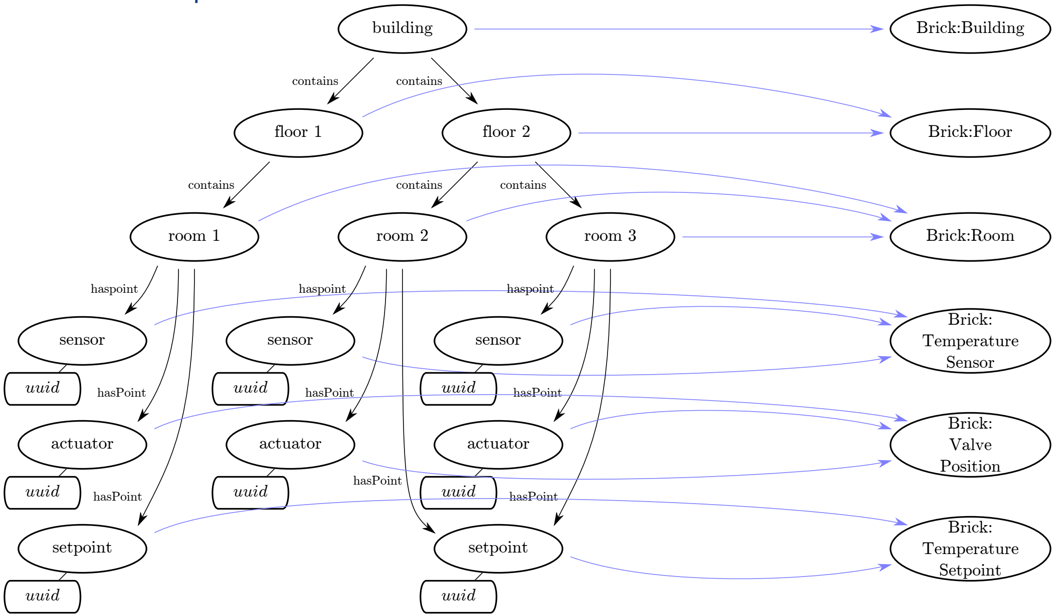
Thermostat Query Thermostat Result


```
[
    [
        "room 1",
        "d0fcec33-af08-44a8-b74b-7a51f1902d13",
        "c087d1ba-03f6-4c60-85de-b95d82e63248",
        "5c2e2bdb-5142-464d-ad91-ae483633dfd6"
    ],[
        "room 2",
        "18407632-8e81-4bb9-ae77-4ecc96f30f46",
        "d2b54605-58b1-44ef-89f9-0a9adabbfbb1",
        "ec883cad-3235-46d5-a901-7bae169dda0a"
    ],[
        "room 3",
        "e528f733-2e3a-4f65-93aa-6fb2e36d4b27",
        "d2b54605-58b1-44ef-89f9-0a9adabbfbb1",
        "9be52aed-0e12-49b5-8680-47778b9b2adf"
    ]
]
```

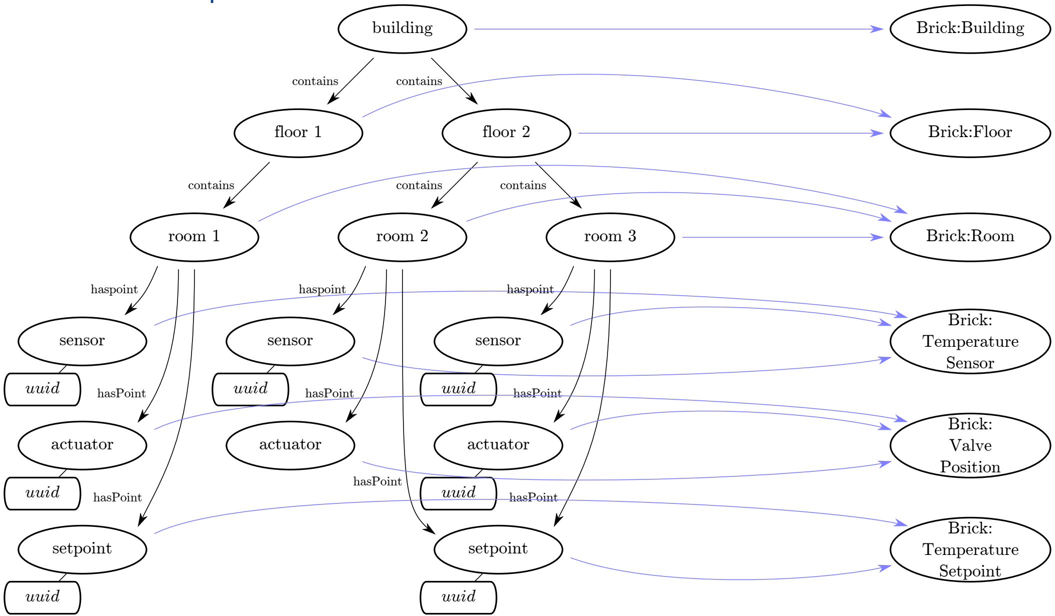
Modification Deletion

```
DELETE {
    ?actuator brick:label "ec883cad-3235-46d5-a901-7bae169dda0a" .
}
WHERE {
    ?actuator rdf:type/brick:subClassOf* brick:Radiator_Valve_Position .
    ?actuator brick:label "ec883cad-3235-46d5-a901-7bae169dda0a" .
}
```

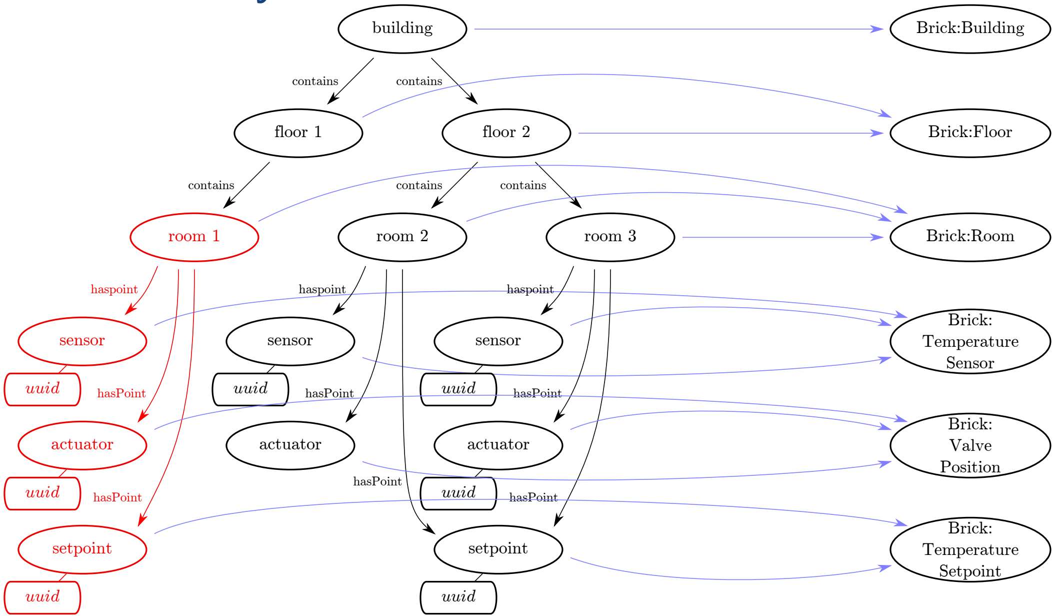
Deletion Pre Update



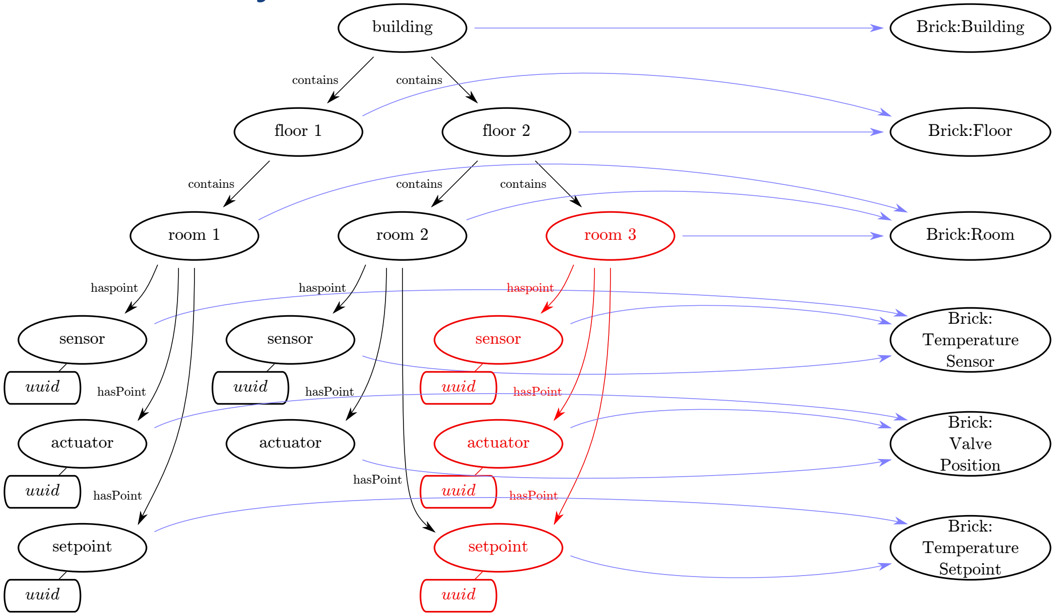
Deletion Post Update



Thermostat Query Match 1



Thermostat Query Match 2



Thermostat Query Results


```
[
    [
        "room 1",
        "d0fcec33-af08-44a8-b74b-7a51f1902d13",
        "c087d1ba-03f6-4c60-85de-b95d82e63248",
        "5c2e2bdb-5142-464d-ad91-ae483633dfd6"
    ],[
        "room 3",
        "e528f733-2e3a-4f65-93aa-6fb2e36d4b27",
        "d2b54605-58b1-44ef-89f9-0a9adabbfbb1",
        "9be52aed-0e12-49b5-8680-47778b9b2adf"
    ]
]
```

Modification Inserts

Inserts can be done in the same way as you do deletes

Insert and delete clauses may be combined

Part 6:
Building a New Ontology

Ontology Construction Introduction

An ontology defines a the equivalent of a schema

A model contains instances of types and relates these instances to each other according to rules defined in such ontologies

A model may refer to multiple ontologies

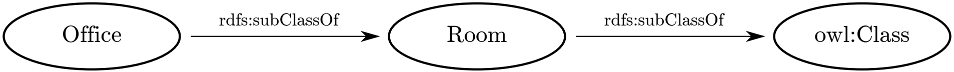
Ontologies are simply RDF stores

They usually define a namespace and are stored in a file

Ontology Construction Defining Types

We can define a type by declaring the name a subclass of something else.

Everything that exists in the `owl:subClassOf` DAG rooted in `owl:Class` is a type.

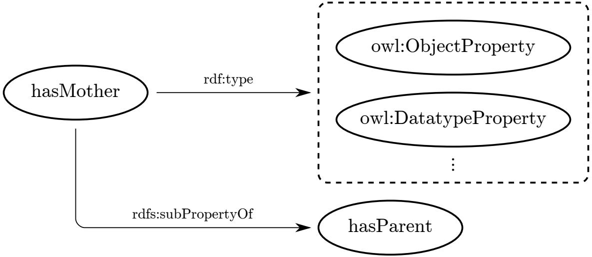


Ontology Construction Defining Relationships

Relationships are defined through their use and their place in the DAG of types

Their use is based on whether it points at a literal (`owl:DatatypeProperty`) or not (`owl:ObjectProperty`)

Their place is based on the `rdfs:subPropertyOf` relation.



Note: In this example `hasMother` and `hasParent` are properties

Ontology Construction Relationship Restrictions

A relationship p can be restricted to only be allowed on some set S of subjects using `owl:domain` properties. Hereby you define the set of allowed triples of the form (S,p,*)

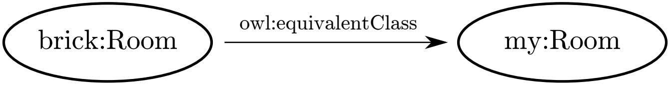
A relationship p can be restricted to only be allowed on some set O of objects using `owl:range` properties. Hereby you define the set of allowed triples of the form (*,p,O)

What happens when you combine them?

Ontology Construction Type Equivalence

We can state that two classes (e.g., types) are equivalent using `owl:equivalentType`

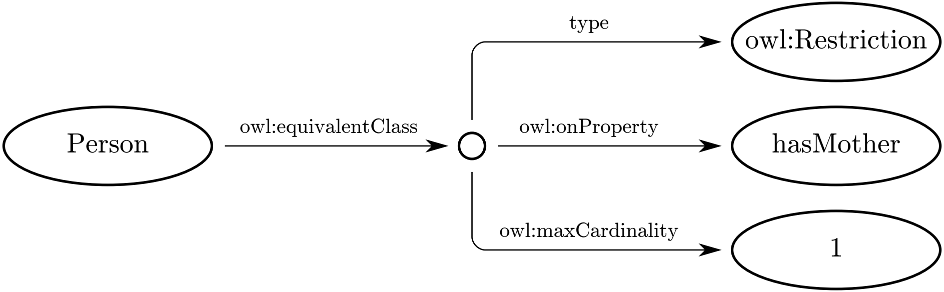
A common usecase is to bridge two ontologies in a model



Ontology Construction Restrictions Between Type and Relationship

We can limit how many triples are allowed to follow the pattern (s, p, *)

This is known as a restriction on cardinality



Note: In this example `hasMother` is a property

Ontology Construction Literals

The XSD namespace defines a set of types whose values can be encoded as literals

https://www.w3.org/2011/rdf-wg/wiki/XSD_Datatypes

Examples:

- int
- double
- boolean
- string
- dateTime

Part 7:
Validation of Model

Role of a Validator

A validator tests the stated rules of the applied ontologies.

Examples:

- The brick:isLocatedIn relationship has as subject something that is a subclass of brick:Equipment and as object something that is a subclass of brick:Location
- Any instance of a person:Human has exactly two parents
- The birthdate property of an instance of a person:Human has the date type

Without a validator triples can be added at will

- Spelling mistakes are accepted
- Entities can be connected in any way
  (in general, the order of definitions don't matter)
- Entities can be floating

Part 8:
Reasoners

Role of a Reasoner

A reasoner can apply rules

Examples:

- Given the facts that Marge is a Female and Marge is parent of Maggie a reasoner can conclude that Marge is the mother of Maggie
- Given a fact stating that some entity is a temperature sensor and another stating that temperature sensor is a subclass of sensor a reasoner can conclude that this entity is a sensor

Note: Each of these require a rule

If you define such a rule, a reasoner will be able to apply it

Consequences of using Reasoner

When to apply the reasoner:

1. When inserting
2. On triplet store before querying
3. While querying

Instead of:

```
?maggie person:hasName "Maggie" .
?mother person:isParentOf ?maggie .
?mother person:isA person:Female .
```

We can write:

```
?maggie person:hasName "Maggie" .
?mother person:isMotherOf ?maggie .
```

Consequences of using Reasoner

When to apply the reasoner:

1. When inserting
2. On triplet store before querying
3. While querying

Instead of:

```
?sensor rdf:type/rdf:subClassOf* brick:Sensor .
```

We can write:

```
?sensor rdf:type brick:Sensor .
```

Part 9:
Model Management

Hosting a Model

Many options:

1. Filesystem
2. Webserver
3. Dedicated RDF server
   - HodDB – Fast, read-only, stability issues
   - Fuseki – Supports writes

At SDU we are working on

- rdfserv – slow, supports writes and multiple protocols
- A next-generation RDF server supporting query-based subscriptions

Interacting with a Model

Most popular high-level languages have libraries for manipulating and querying a triple store

- ***Python:*** `rdflib`
- ...

Typical qualities:

- No validation
- Limited or no reasoners
- "Slow" query resolution available
  - Simple queries are fast
  - Large complex queries takes forever

This is where dedicated RDF servers come into play!

Further Reading

RDF and OWL:

- Google has plenty of information

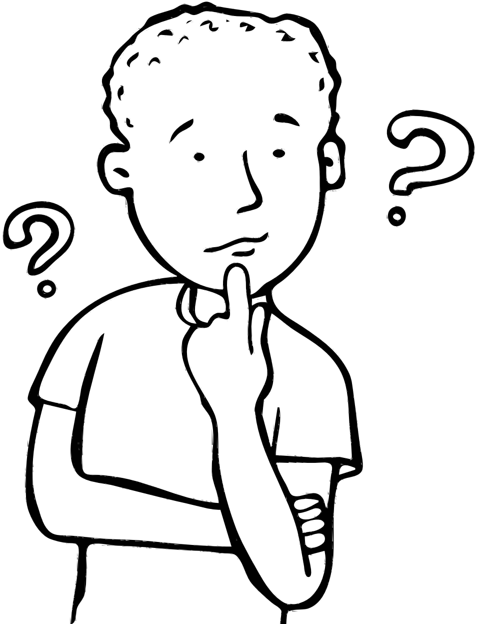
Brick:

- Official homepage at `http://brickschema.org`

Evaluating SparQL Queries:

- ***Library*** For python there is the `rdflib` module at `https://github.com/RDFLib/rdflib`
- ***Web Service***
  - Fuseki: `https://jena.apache.org/documentation/serving_data/`
  - HodDB: `https://hoddb.org`

Questions?



https://openclipart.org/detail/238687/boy-thinking-of-question