Software Technology of Internet of Things

Metadata Exercises: Adventures in Semantic Madness

Aslak Johansen asjo@mmmi.sdu.dk

February 18, 2025

Congratulations on reaching level 2! This is an important step; the first of many.

Quest 1: Exploring the Lecture Example (5XP)

Over the last decade or so the Elders have been constructing the World of Things. But it didn't turn out quite right. Rumors has it that the streams have crossed, and the consequences of that could reach far and wide. We need you, young student, to help us sort it out! I sure hope you have kept your Elixir skills sharp ...

First, you must attend the lecture, or – if tardy – study the lecture slides. Next, you should study the lecture building model as it contains many details and complexities. Once you have familiarized yourself with the model and have a good understanding the high-level structure you may venture into Quest 2!

Quest 2: Exploring the Source (5XP)

The remainder of this quest line is solvable using any of the two data models. You should do both, and compare the two solutions. Choose your starting point from the Repository of Wisdom:

- ***RDF*** While working on the RDF data model, focus on the `rdf.livemd` file.
- ***Neo4j*** while working on the Neo4j data model, focus on the `neo4j.livemd` file.

Two paths lie in front of you, and you must choose wisely. Do you focus on one data model at a time, or do you focus on one exercise at a time? In any case, now is the time to explore, and figure out what makes them tick!

Quest 3: A New Query (5XP)

Lets start out easy by adding a query that lists all the rooms that – according to the model – (i) are contained within a building, and (ii) have a live data stream that reports in Kelvin.

Quest 4: Universe Expansion (10XP)

Next, you should introduce a building concept:

- Add a `Building` type and have instances of this `contain` floors.
- Add such an instance and connect it the rest of the model.
- Add a query that demonstrates that this has been achieved.

Quest 5: Temperature Actuator (15XP)

We now have a model of a building with a couple of rooms. According to the model, the building has sensors for temperature, humidity and occupancy, and they all produce data. However, these data are not being used for anything; control-wise the building is dead! Lets change that, at a model level, by adding a thermostat.

In order to run a thermostat, we need – for the same room – the live temperature, the setpoint[1] and a way of adjusting the temperature. This could be a signal to a radiator valve of the HVAC system[2]. Lets just call this a temperature actuator, and imagine that we can address it through some ID.

In this exercise you should:

1. Add the relevant terms to the vocabulary (aka the modeled typesystem).
2. Consider whether *provides* is the right relationship between the actuator and the modality, or if another needs to be added.
3. Add instances of the temperature actuator to each room.
4. Use the web interface to Neo4j to verify that you have modeled this information according to your intention, or a (number of?) queries if you are using RDF.
5. Write a new query that – for each room – extracts all necessary IDs, so that an application (through these) could implement a thermostat. Naturally, that would require some system for obtaining real temperature readings and realizing actuation requests. But for the purpose of this exercise, we will just imagine that this is available to us.

[1] The setpoint is the temperature that the thermostat should attempt to make the sensor output match.

[2] HVAC is an abbreviation of Heating, Ventilation and Airconditioning.

6. How does a room that is missing one of the needed sensors affect the resultset?

Quest 6: Schema (20XP)

One problem that we are facing, is that we can add anything to the model: There are no checks. Giles – who is otherwise a very thorough individual – discovered that the hard way, and burned a small building to the ground. Luckily, No-one was harmed, but it is still scary to think of the possible consequences of a small human mistake.

RDF has RDFS, OWL and SHACL to help defining a schema, but a schema is really just a set of rules and they are not really enforced here. With that in mind:

1. What would a *model checker* look like?
2. Which – if any – rules are missing?
3. How would you describe them?
4. How would you you implement them?

Note: You don't need to implement all the rules. Just enough for it to get repetitive.