WOT CG MEETUP 2024-10-14

USING WEB OF THINGS THING MODELS

AS A BASIS TO MAKE TO HOUSING SECTOR SMART AND ENERGY-EFFICIENT

Date: 14.10.2024, by **Thomas Jäckle** (**@thjaeckle**)

ABOUT beyonnex.io

- → digitizing **real estate** / the housing sector
- → young **IoT startup** (< 3 years)
- → part of the **Noventic group**
- → connecting different types of devices to our own IoT backend
- → HQ: Hamburg, Germany - employees (>100) all over Europe

beyonnex.io

- → our world: millions of private households in apartment blocks in the housing industry
- → our customers: the housing companies / landlords
- → residents use the devices **as a service**
  (think of: battery, connectivity, maintenance)

SUBMETERING DEVICES

collect and allocate **heat/water consumption** of residents

SMART HEATING DEVICES

give residents more **control** over their **heat consumption** via App or manually

HOW ARE THOSE DEVICES CONNECTED?

M-Bus

→ Meter-Bus (European standard for the remote reading of water, gas or electricity meters)

→ uni-directional

LoRaWAN®

→ technology using license-free radio frequency band to connect many devices with few gateways over a **Lo**ng **Ra**nge, aiming for low power consumption of end devices

→ bi-directional

OUR DEVICES ARE MOST OF THE TIME OFFLINE

DEVICE ENVIRONMENT

- especially in the housing sector, the device's **environment** is important
- where (building, apartment, etc.) are devices **installed**?
- to which customer ("owner") do they **belong**?

Generated with Microsoft Bing Copilot

- who is the current **resident** allowed to **control** e.g. thermostats?

DIGITAL TWIN

more than just a buzzword

SPECIAL CHALLENGES TO SOLVE

- **heterogeneous device** types
- devices mostly **offline**
- various **user groups** (e.g. residents, installers, operators/supporters)
- **multiple development teams**
  - either connecting "their" device types
  - or "just" using device functionality (e.g. for mobile apps or operator/support dashboards)

A SOLUTION: DIGITAL TWIN

- → providing **APIs** for **accessing device data** and sending commands
- → **abstracting** from **protocols** devices are connected with
- → handling **access control** for various user groups
- → enabling **higher order twins** for rooms, apartments, buildings

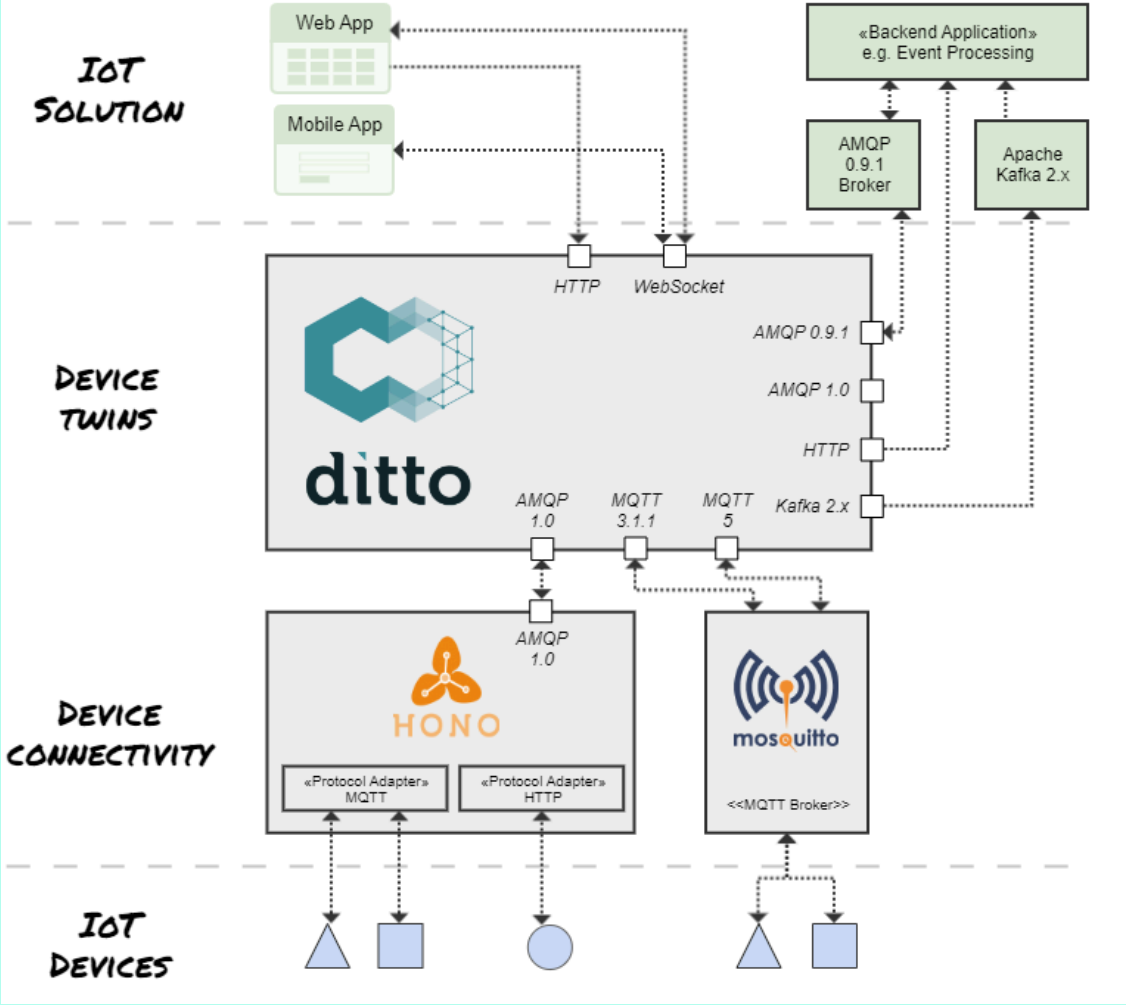
Generated with Microsoft Bing Copilot

- → used by various different development teams as **IoT middleware**

ECLIPSE DITTO

our preferred digital twin solution

ECLIPSE DITTO IN CONTEXT



Ditto as
IoT
and/or
Digital Twin
"middleware"

DEVICE DATA AS APIS

```
{
  "thingId": "io.beyonnex.therm
  "policyId": "io.beyonnex.room
  "attributes": {
    "serial": "4711-0815",
    "location": {
      "buildingId": "nice-build
    }
  },
  "features": {
    "temp": {
      "properties": {
        "value": 23.42
      }
    }
  }
}
```

JSON repr. of a Thing

```
GET/PUT/PATCH/DELETE /things/io.beyonnex.thermostat:my-t1
 /things/io.beyonnex.thermostat:my-t1/thingId
 /things/io.beyonnex.thermostat:my-t1/policyId
 /things/io.beyonnex.thermostat:my-t1/attributes
 /things/io.beyonnex.thermostat:my-t1/attributes/serial
 /things/io.beyonnex.thermostat:my-t1/attributes/location
 /things/io.beyonnex.thermostat:my-t1/attributes/location/l


 /things/io.beyonnex.thermostat:my-t1/features
 /things/io.beyonnex.thermostat:my-t1/features/temp
 /things/io.beyonnex.thermostat:my-t1/features/temp/proper
 /things/io.beyonnex.thermostat:my-t1/features/temp/proper
```

HTTP API of the Thing

DITTO'S WOT INTEGRATION

a Ditto managed digital twin (a.k.a "thing") can reference a (WoT Thing) model

```
{
  "thingId": "io.beyonnex.thermostat:my-t1",
  "definition": "https://models.some.domain.org/t
}
```

- → upon **thing creation**, a JSON skeleton is generated based on the defined model `properties`
- → for each thing, a **WoT Thing Description** is **generated** (on the fly) based on model, providing HTTP `forms` for defined `properties, actions, events`
- → **new** (since Ditto 3.6.0): validation of thing (`properties`) modifications and `action` + `event` payload based on WoT TM

TM BASED THING VALIDATION

- → **Ditto** now can **ensure** that interaction with a twin always **adheres to the defined model**
- → reasoning: **programming errors happen** - and can now already be detected early during development

```
PUT https://ditto.host/api/2/things/io.beyonne
false
```

```
{
  "status": 400,
  "error": "wot:payload.validation.error",
  "message": "The provided payload did not c
  "description": "The Thing's attribute </se
  "validationDetails": {
    "/attributes/serial": [
      ": {type=boolean found, string expecte
    ]
  }
}
```

- → also for e.g. action "input" payloads, guiding 3rd party users of published twin API with helpful error messages

MODELS

WoT TMs ensure that (API) contracts are defined and met

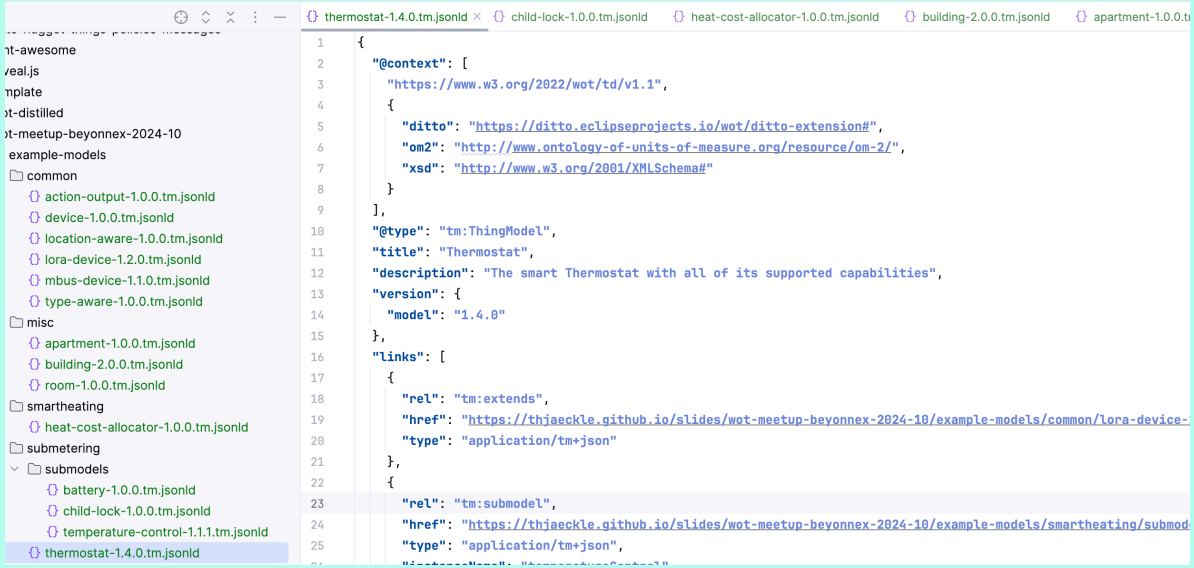
THING MODELS AND HOW WE USE THEM

- in our Thing Models, we use inheritance (`tm:extends`) **and** composition (`tm:submodel`), applying "object-oriented" modeling, e.g.:
  - a thermostat **is-a** device, a device **is** aware of its identity
  - a thermostat **has-a** functionality regarding temperature control, child lock and battery usage
  - a room as well **has-a** functionality regarding temperature control (but not child lock and battery usage)
- occasionally as well importing single capabilities cross-model (using `tm:ref`)

MODELED AS CLASS DIAGRAM

HOW WE CREATE MODELS

- → write new (or copy&paste from existing ones) TMs in our favorite **IDE**
- → apply **semantic versioning**



- → commit and push TMs to version control (**Git**)
- → CI/CD pipeline automatically **publishes TMs** to a "development" **website**
- → creating **Git tags** to **release TMs**
- → released TMs are deployed to a QA and productive website

MODEL TOOLING

WoT TMs are great, and we invest in building a tooling ecosystem around them

MODEL TOOLING: OPENAPI GENERATION

- **writing/enhancing** TMs **is hard**
- **reading TMs also** needs quite some understanding
- so we created some **tooling to help our development** teams
  - **generating OpenAPI definition** for complete Ditto **Things** based on TM and Ditto endpoints
  - during that step **ensuring** that WoT TM can be parsed
  - doing that **automatically for merge requests**

SERVING GENERATED OPENAPI IN SWAGGER-UI

Thermostat 1.4.0 OAS 3.1

The smart Thermostat with all of its supported capabilities

Servers

https://ditto.eclipseprojects.io/api/2/things - Ditto: Sandbox

Authorize

Thing Complete Thermostat

- GET /{thingId} Retrieves the complete 'Thermostat' thing
- PATCH /{thingId} Merges the complete 'Thermostat' thing

Attributes Attributes of the Thermostat

- GET /{thingId}/attributes Retrieves all attributes
- GET /{thingId}/attributes/devEUI Retrieves the 'DevEUI'
- GET /{thingId}/attributes/id Retrieves the 'Device ID'
- GET /{thingId}/attributes/location Retrieves the 'Location'
- GET /{thingId}/attributes/manufacturer Retrieves the 'Manufacturer'
- GET /{thingId}/attributes/serial Retrieves the 'Serial number'

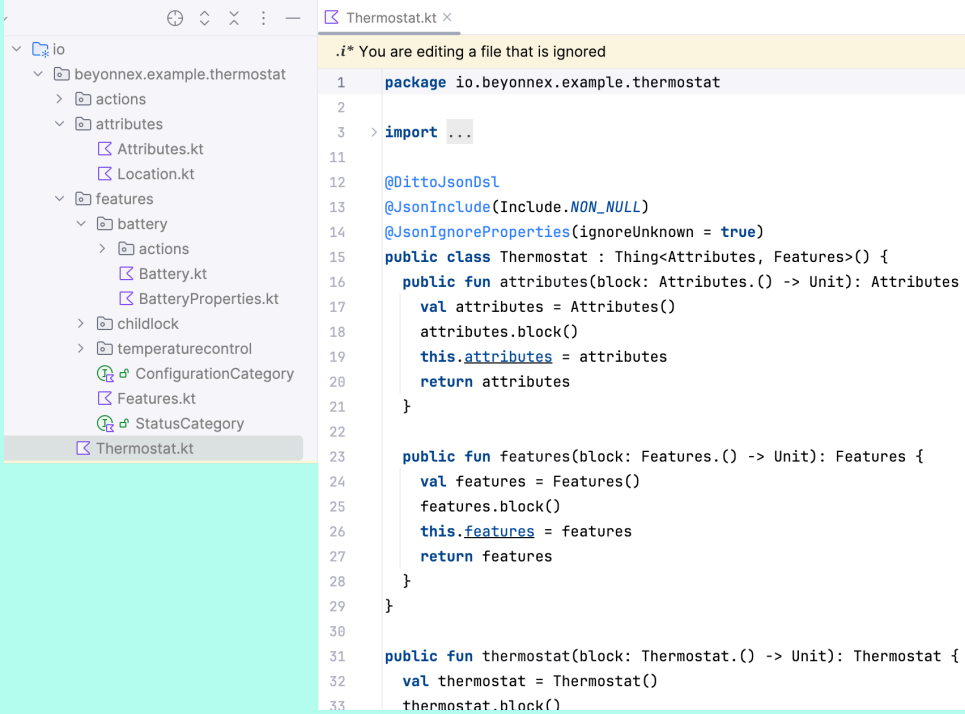
Example Value | Schema

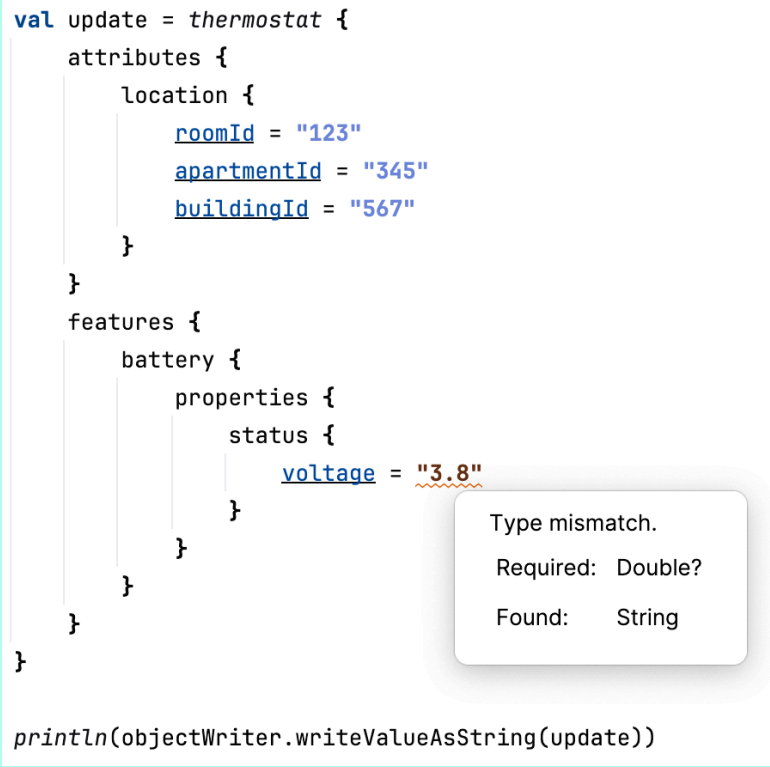
```
thing ^ Collapse all object
  thingId string
  policyId string
  definition string
  attributes ^ Collapse all object
    devEUI > Expand all read-only string matches ^[0-9a-f]{16}$
    id > Expand all read-only string matches ^[0-9a-f]{8}-([0-9a-f]{4}-){3}[0-9a-f]{12}$
    location > Expand all read-only object
    manufacturer > Expand all read-only string
    serial > Expand all read-only string
    type > Expand all read-only string
  features ^ Collapse all object
    battery > Expand all object
    childLock > Expand all object
    temperatureControl ^ Collapse all object
      The temperature control for the device or virtual thing
      properties ^ Collapse all object
        configuration ^ Collapse all object
          targetTemperature ^ Collapse all read-only number [1, 40]
            Target temperature set on the device in celsius
      desiredProperties > Expand all object
```

MODEL TOOLING: CODE GENERATION

- before using WoT TMs and tooling around it, we had **challenges** to keep our **things "consistent"**
- e.g. often data classes were manually written, typos and wrong data types were happening often
- so we created some **tooling to help our development** teams
  - **generating Kotlin (data) classes** for complete Ditto **Things**, e.g. of type "Thermostat"
  - **generating a Kotlin DSL** to simply build new instances in code (incl. compiler checks)
  - providing that as **Maven plugin**

GENERATED KOTLIN CODE





MODEL TOOLING: MODEL MIGRATION

- our newest tool
- **evolving Things** over time with **new capabilities**
- providing tooling in order to **migrate** (as in a DB migration) **Things** from one referenced TM version to the next increment in Ditto, **patching** the "delta" into the things

MIGRATION TOOL IN ACTION

Model Migration Tool

Model Name

thermostat

Model Version

1.5.0

Environment

DEV

Migration Payload

```
{
  "definition": "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/thermostat-1.5.0.tm.jsonld",
  "attributes": {
    "addedProperty": "its default"
  },
  "features": {
      "partyMode": {
          "defintition": [
              "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/submodels/party-mode-0.0.1.tm.
          ],
          "properties": {
              "partyModeEnabled": true
          }
      }
  }
}
```

Search Filter

like(definition,'*thermostat-1.5.0*')

Things Count:

N/A

Patch Condition

Ditto JWT

Enter your Ditto Bearer Token

Submit

DIFF TOOL IN ACTION

Diff Tool

Model Name

Thermostat

Main Version

1.4.0

Update Version

1.5.0

DITTO Token

••••••••••••••••••••••

Generate Diff

Diff Output

response-diff CHANGED

```
@@ -1,8 +1,8 @@
 {
   "thingId": "io.beyonnex.smartheating.srt.mock:eui0123456789",
   "policyId": "io.beyonnex:connect-go",
-  "definition": "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/thermostat-1.4.0.tm.jsonld",
+  "definition": "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/thermostat-1.5.0.tm.jsonld",
   "attributes": {
     "type": "SMART_RADIATOR_THERMOSTAT",
     "serial": "_",
     "manufacturer": "_",
@@ -15,9 +15,9 @@
   },
   "features": {
     "temperatureControl": {
       "definition": [
-        "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/submodels/temperature-control-1.1.1.tm.jsonld",
+        "https://thjaeckle.github.io/slides/wot-meetup-beyonnex-2024-10/example-models/smartheating/submodels/temperature-control-1.2.0.tm.jsonld",
       ],
       "properties": {
         "targetTemperature": 19.5,
+        "offset": 0.0
       }
     },
```

WRAPPING UP

- we mainly use WoT **Thing Models**
- Ditto can now **ensure changes** to the twin always **adhere to the TM**
- Ditto **generates Thing Descriptions** to be prepared for future integrations, e.g. API description for partners
- we continue to **invest in tooling** around **TMs** and Ditto integration
- we **aim** to make the generic **tooling** around **Ditto and TMs OSS**

thank you for attending and your interest

Q & A