NAIADES

Webinar Series

IoT Technologies for Smart Water Systems

NOV 16, 2021
10-12.30 CET

ONLINE

Join Us!

disy

UDG Alliance

Jožef Stefan Institute

NAIADES Speakers

Fraunhofer IOSB

KOMPETENZZENTRUM WasserBerlin

DATA COVE

External Speakers

Data Model Validation

Julian Bruns
DISY

disy

Geointelligenz für Ihre Daten

Motivation behind Data Model Validation

- We want to enable and use the full power of IoT for the water sector.

- This results in the existance of highly heterogenous data with a manyfold of different data types and inputs from many different sources – each with their own reasoning.

- Therefore, we will also get highly conflicting data, incompatible inputs and the need for expensive transformations; e.g. sensors of different companies working together

- However, to ensure that we can utilize all this – with a reasonable effort – we need to use agreed upon, open data models and validate those before use.

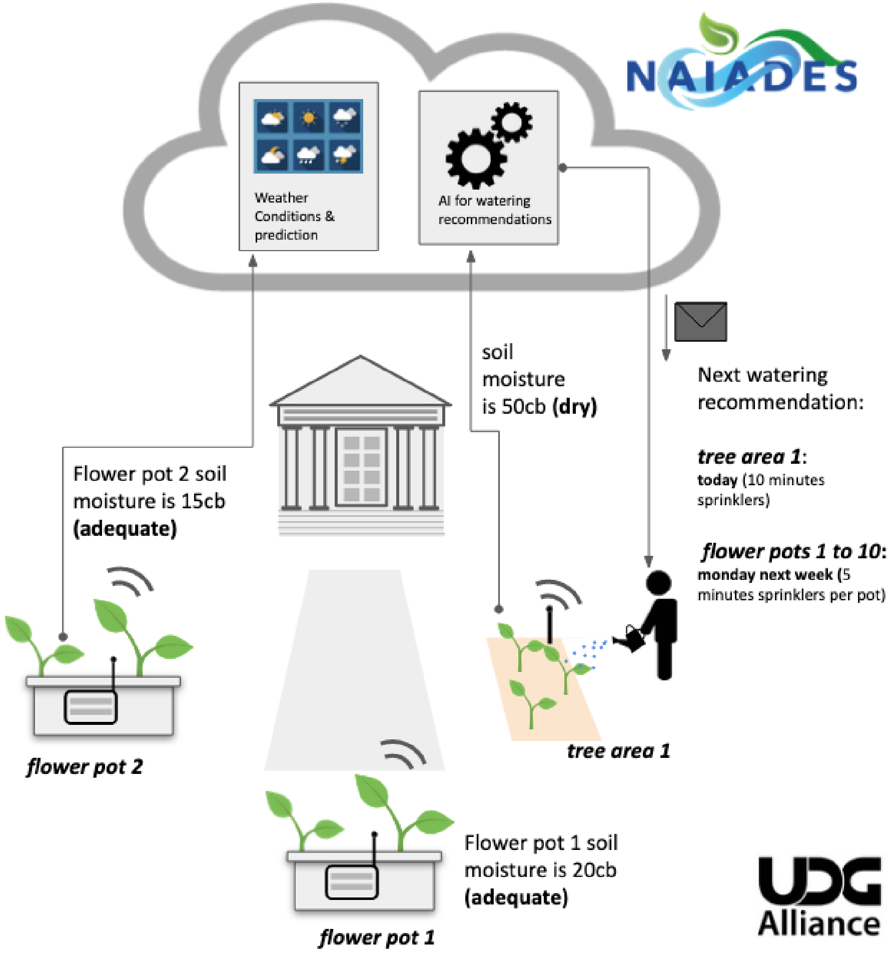
IoT and the Water Sector
–
An Introduction

What is IoT?

- IoT is the so called Internet of Things – the idea that everything is mapped to an online twin and that the data is gathered in real time
- This data can then be analyzed in real time, e.g. by AI, complex event processing or other methods – It is based on automated analytic pipelines
  - See e.g. the talk about teh early warning system for bathing water quality
- The most common examples are
  - Connected appliances
  - Wearables
  - Smart factory equipment
  - Tracked Shipping
  - Smarty City – mainly traffic, noise and pollution

What makes the water context special for IT?

- More than online sensors (even though quite important! – see the talks about SensorThings API)
- The water sector is highly interconnected and interdependent
- It always has a relation to time and space – e.g. weather or geography
- It needs many different data types apart from only measurements

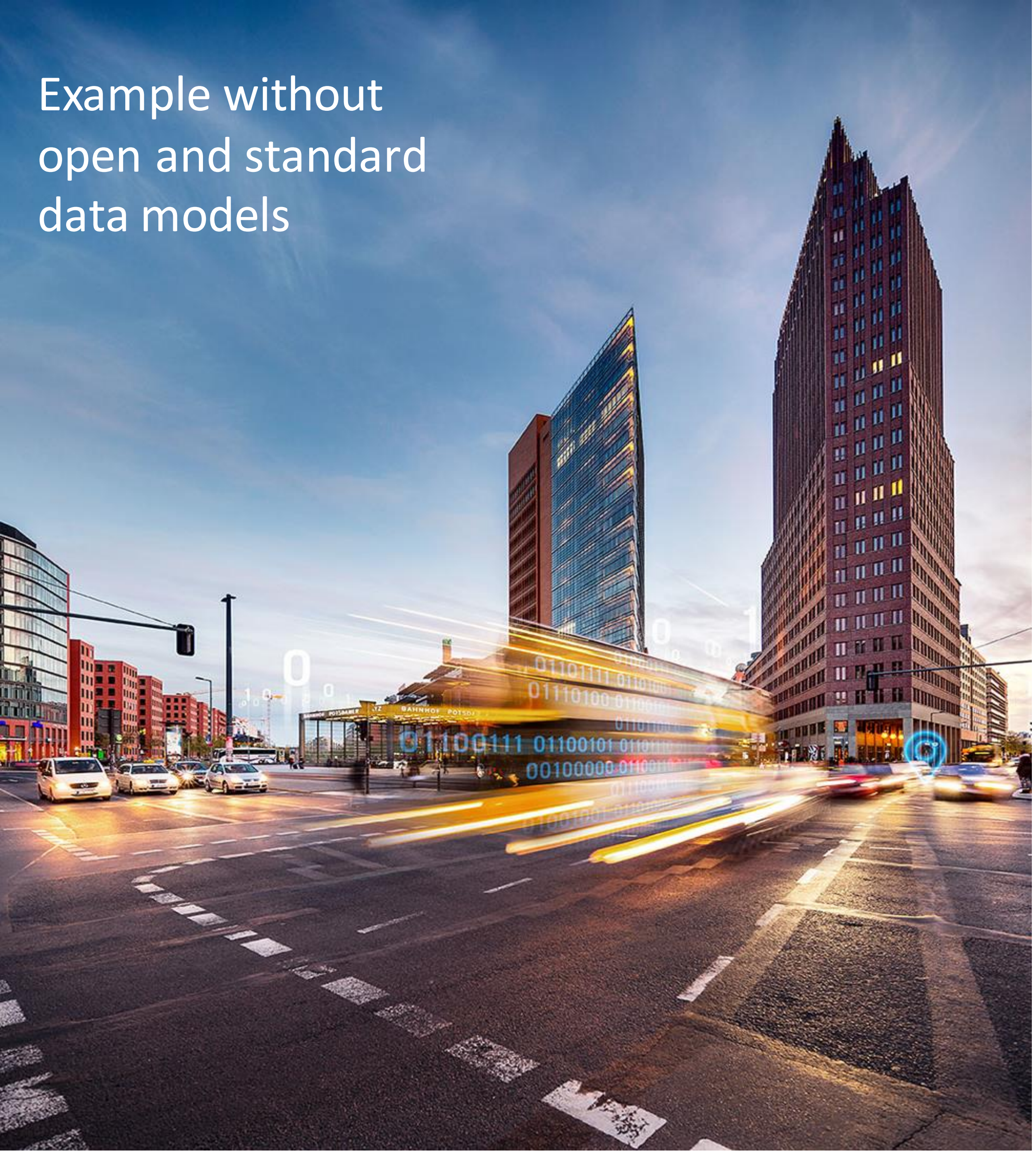

NAIADES
Weather Conditions & prediction
AI for watering recommendations
soil moisture is 50cb (dry)
Next watering recommendation:
tree area 1:
today (10 minutes sprinklers)
flower pots 1 to 10:
monday next week (5 minutes sprinklers per pot)
Flower pot 2 soil moisture is 15cb (adequate)
flower pot 2
tree area 1
Flower pot 1 soil moisture is 20cb (adequate)
flower pot 1
UDG Alliance

Data models:
An overview

What are data models - why do we need them?

- A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities. (Wikipedia)
- They fulfill the need of a certain domain – many different models are possible
- Data models are fundamental tools for the harmonization of data
- They enforce a set schema everyone adhers to (in principle)
- They enable interoperability and global governance

Example without open and standard data models



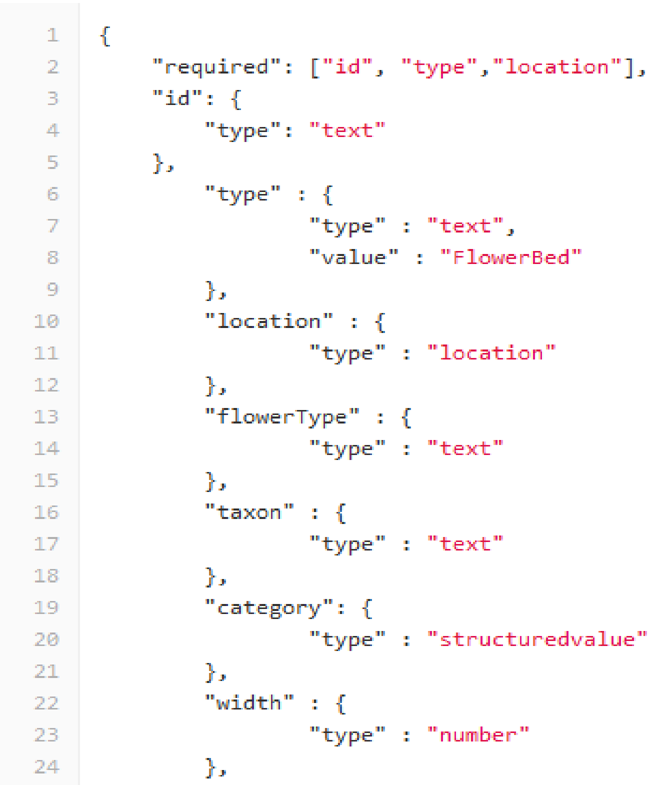
- Smart City solutions are becoming increasingly common
- They provide many benefits for the society
  - In particular for municipal agencies
- However:
  - There are many competing solutions
  - They have highly heterogenous standards and data formats
  - There is no agreed upon exchange format
  - The key hardware provider offer only their own, proprietary and closed solutions

→ This leads to a vendor lock-in and missing innovation

NAIADES approach

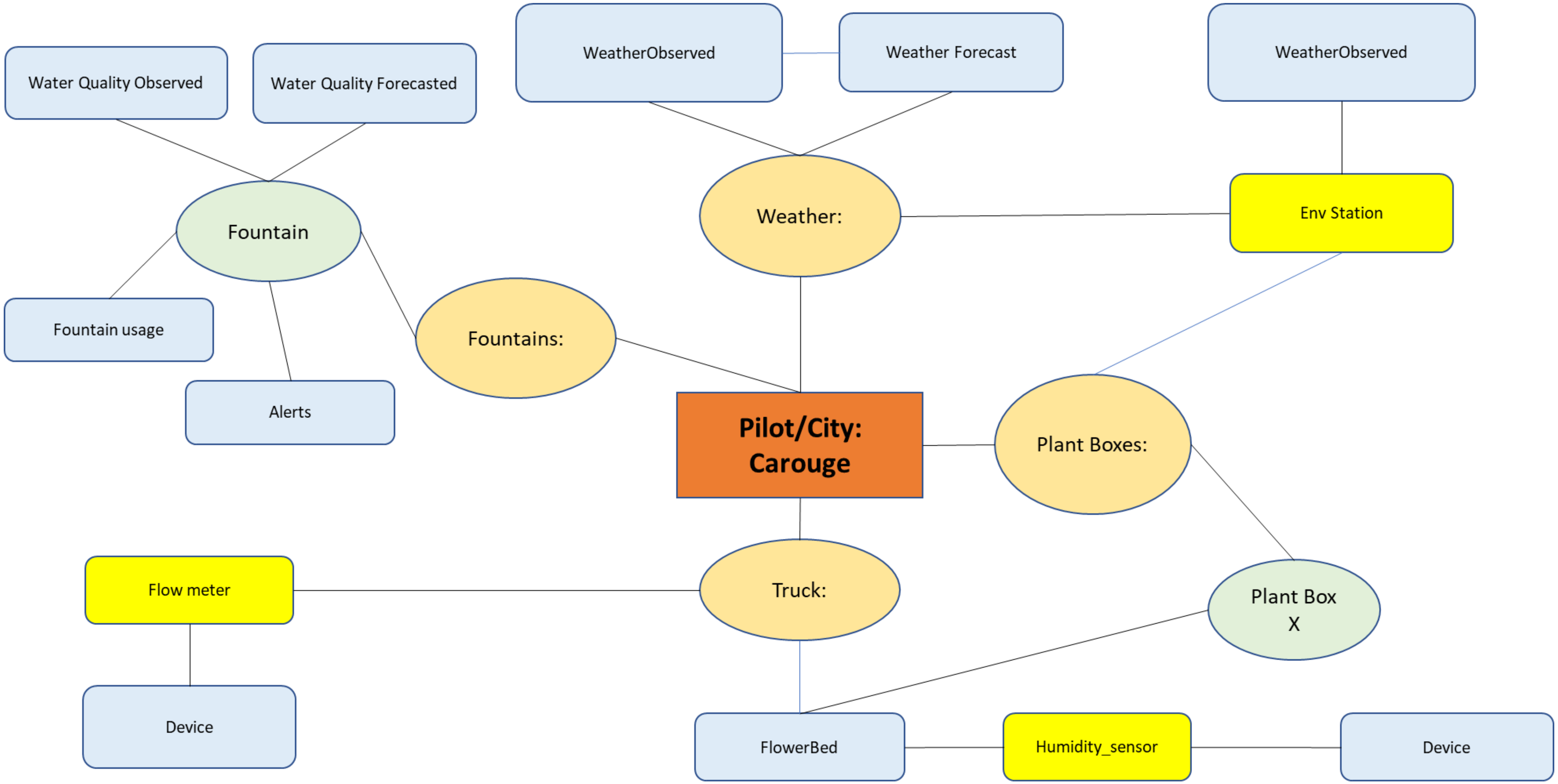
The NAIADES approach for data models

- We chose the existing FIWARE NGSI data models as our basis
- FIWARE data models are based on the JSON standard
- This allows the utilization of all extensions, e.g.
    - GEOJSON
    - JSON-LD
- In addition, as it is based on JSON, this allows it to be human readable and easily extendable



```
{
    "required": ["id", "type","location"],
    "id": {
        "type": "text"
    },
        "type" : {
                "type" : "text",
                "value" : "FlowerBed"
        },
        "location" : {
                "type" : "location"
        },
        "flowerType" : {
                "type" : "text"
        },
        "taxon" : {
                "type" : "text"
        },
        "category": {
                "type" : "structuredvalue"
        },
        "width" : {
                "type" : "number"
        },
```

Example from the city of Carouge



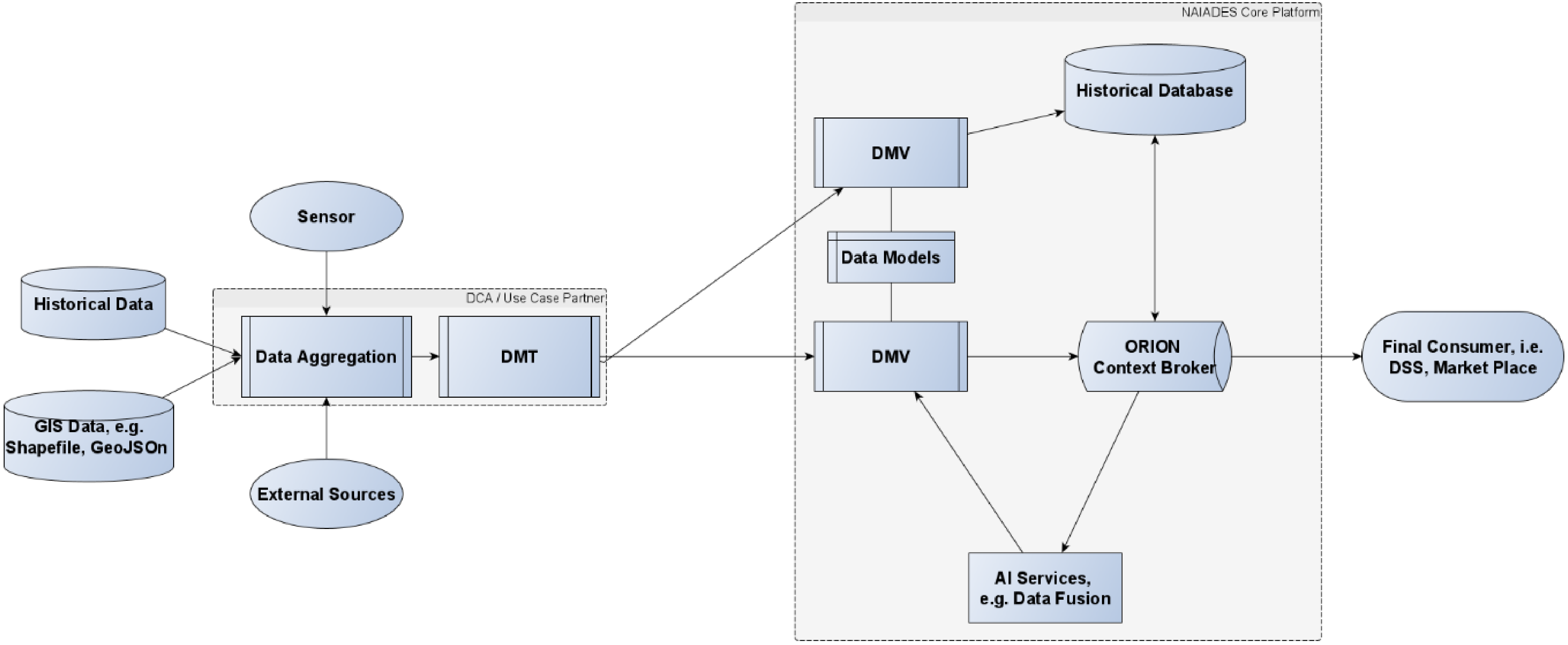
Put together:
Data Validation

Data model validation – reason and approaches

- Any processing of data is built upon pre-existing assumptions about the incoming data
  - Both its structure and its content – based on the data model
  - However, data can deviate from these assumptions, e.g. corruption, programming errors, …
- To ensure that these assumptions are fulfilled, data has to be validated

- Two basic approaches where to validate the data models
  - Validate locally at each time it is consumed
  - Validate centrally and trust the data from that point on
- Additional challenge: How to deal with changing data models?
  - Data models can change over time or additional models can be needed

Data Model Validation – The NAIADES way



THANK YOU

Follow us on social media

@naiadesproject

@naiadesproject

www.naiades-project.eu

https://gitlab.distantaccess.com/naiades/dmv_public

Integration onto FIWARE

Cédric Crettaz, UDG Alliance (UDGA)

Objectives

- Integration of water data onto FIWARE
- Multiple sources of water data:
  - IoT sensors measuring physical parameters in the environment (temperature, soil moisture, chemical components, etc.)
  - SCADA systems
  - Databases
  - Open data repositories

Steps for the integration

1. Install the NAIADES IoT platform.
2. Select a data model.
3. Create an entity.
4. Create a subscription.
5. Update the attributes of an entity.
6. Get all the historical data.

Installation of the IoT platform

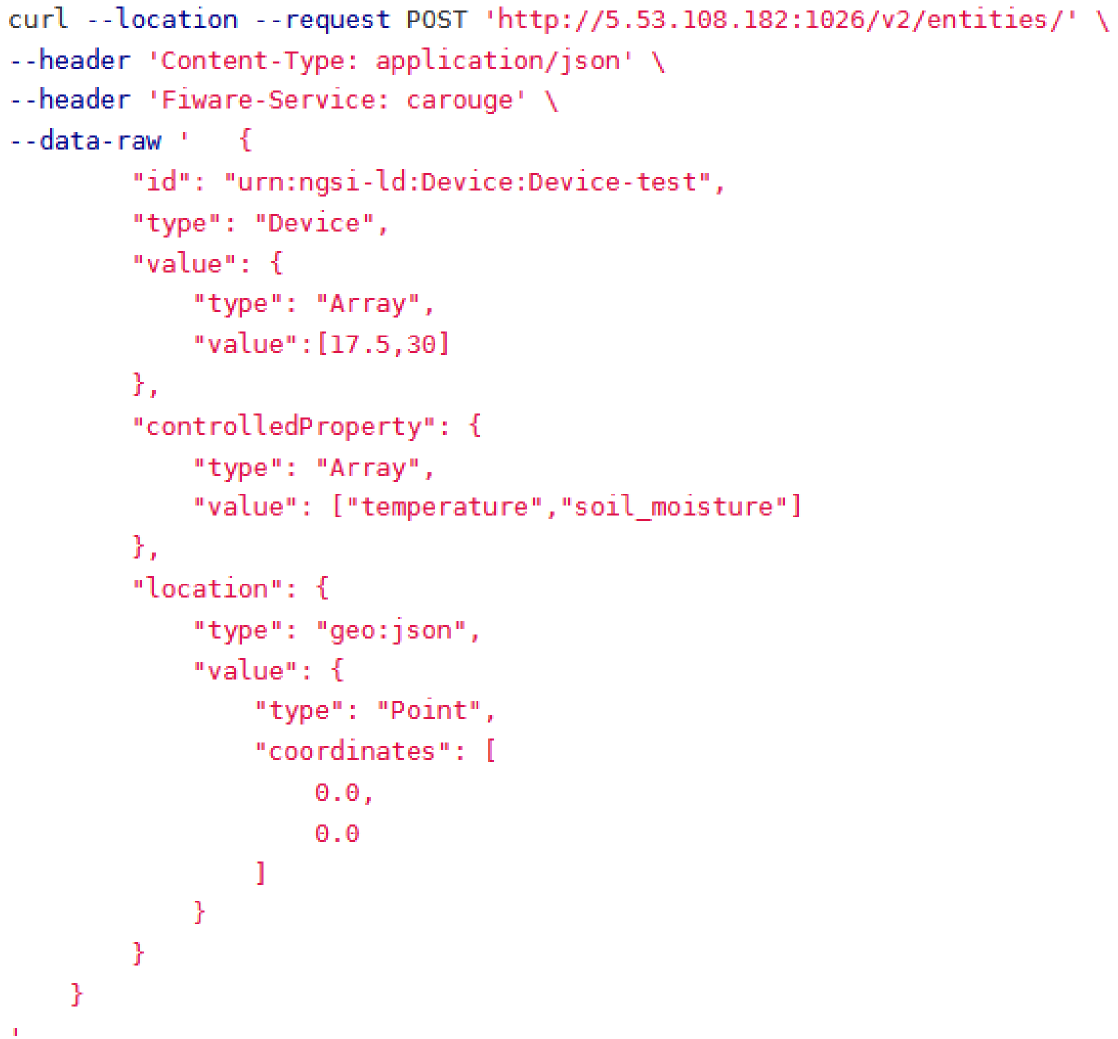
- The NAIADES IoT platform is using Docker containers, limiting the numbers of dependencies to git, docker and docker-compose.
- Steps for the installation:
  1. Clone the project from GitLab: *git clone https://gitlab.distantaccess.com/naiades/naiades-platform-poc.git*
  2. Create the minimal environment with *docker-compose up –d*
  3. Check if the IoT platform is running with *docker ps*
  4. Install other modules or services following the instructions on GitLab: https://gitlab.distantaccess.com/naiades/naiades-platform-poc/-/wikis/NAIADES-IoT-Platform-installation

Selection of a data model

- It depends on the data sources.
- There are several possibilities:
  1. Specific data models like WeatherObserved
  2. Generic data models like Device
- Parameters to be taken into account for the right choice:
  1. Data usage by the consumers of data
  2. Large heterogeneous data sets
  3. Single or multiple measurements at the same time
  4. Level of complexity and flexibility

Creation of an entity

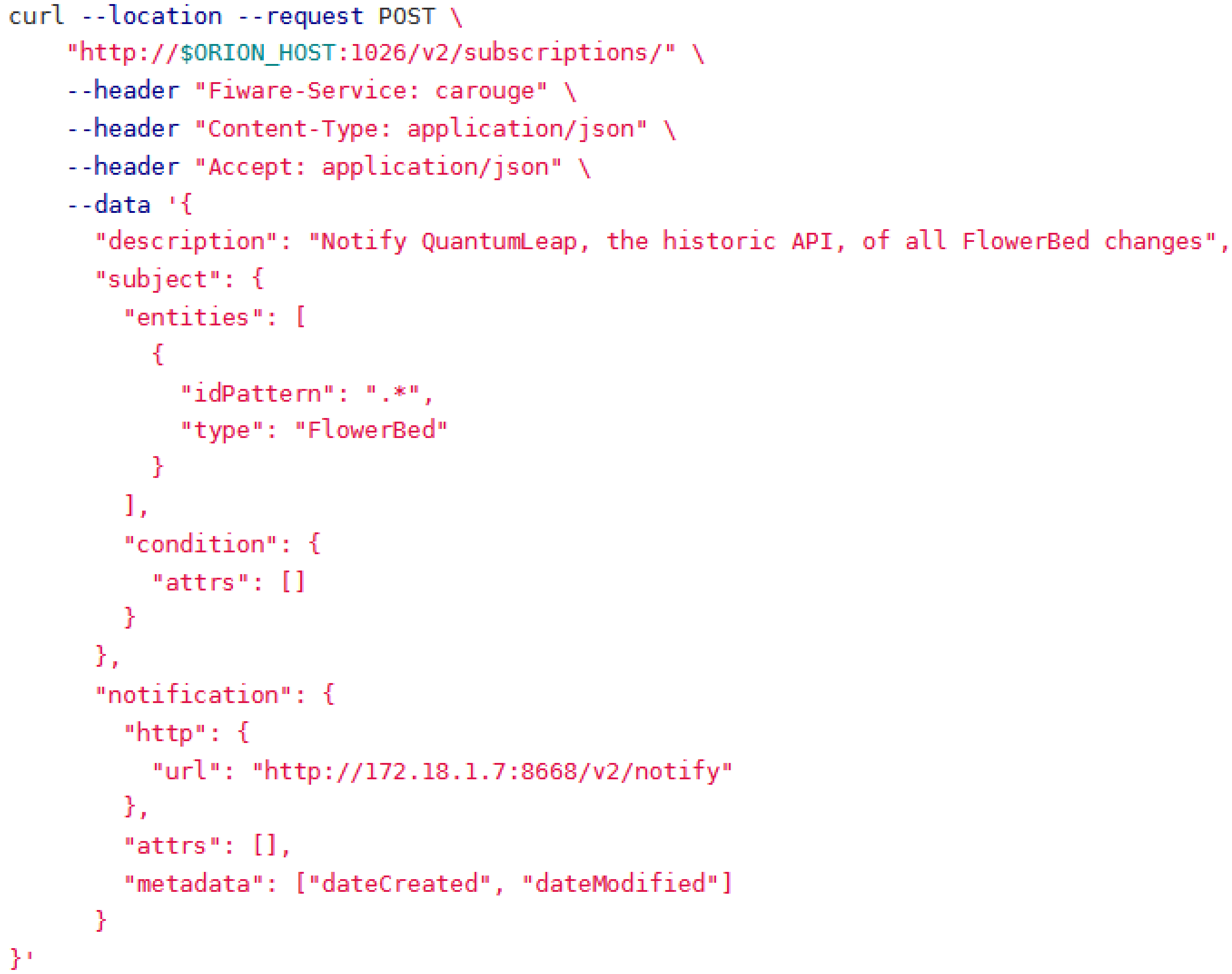
- HTTP POST request to the context broker:



```
curl --location --request POST 'http://5.53.108.182:1026/v2/entities/' \
--header 'Content-Type: application/json' \
--header 'Fiware-Service: carouge' \
--data-raw '   {
        "id": "urn:ngsi-ld:Device:Device-test",
        "type": "Device",
        "value": {
            "type": "Array",
            "value":[17.5,30]
        },
        "controlledProperty": {
            "type": "Array",
            "value": ["temperature","soil_moisture"]
        },
        "location": {
            "type": "geo:json",
            "value": {
                "type": "Point",
                "coordinates": [
                    0.0,
                    0.0
                ]
            }
        }
    }
'
```

Creation of subscription

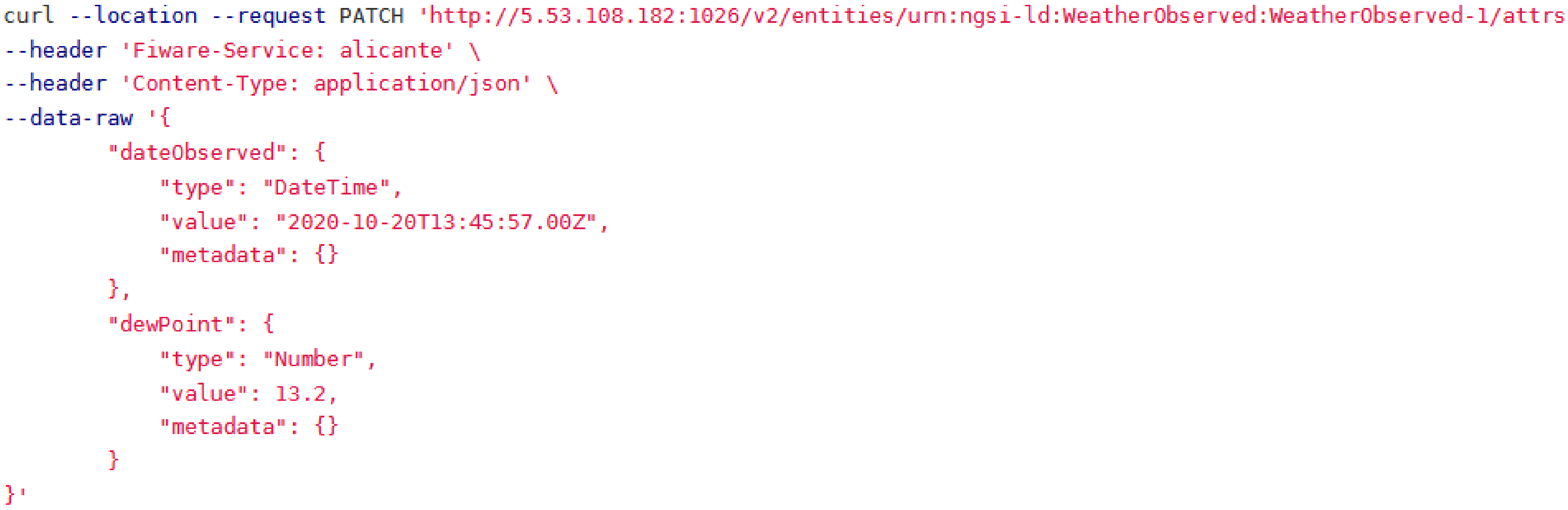
- A subscription permits the reception of a notification with the data to the historical database or to other applications.
- HTTP POST request to the subscriptions endpoint of the context broker at the right:



```
curl --location --request POST \
    "http://$ORION_HOST:1026/v2/subscriptions/" \
    --header "Fiware-Service: carouge" \
    --header "Content-Type: application/json" \
    --header "Accept: application/json" \
    --data '{
      "description": "Notify QuantumLeap, the historic API, of all FlowerBed changes",
      "subject": {
        "entities": [
          {
            "idPattern": ".*",
            "type": "FlowerBed"
          }
        ],
        "condition": {
          "attrs": []
        }
      },
      "notification": {
        "http": {
          "url": "http://172.18.1.7:8668/v2/notify"
        },
        "attrs": [],
        "metadata": ["dateCreated", "dateModified"]
      }
}'
```

Update of an entity

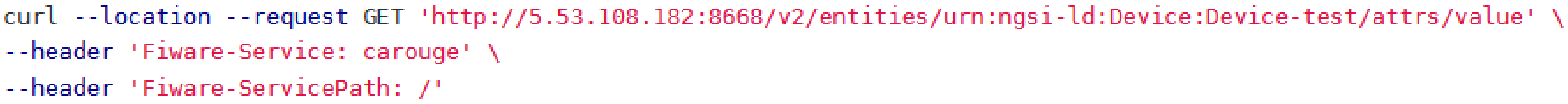
- HTTP PATCH request to the context broker, directly on the entity:



```
curl --location --request PATCH 'http://5.53.108.182:1026/v2/entities/urn:ngsi-ld:WeatherObserved:WeatherObserved-1/attrs
--header 'Fiware-Service: alicante' \
--header 'Content-Type: application/json' \
--data-raw '{
        "dateObserved": {
            "type": "DateTime",
            "value": "2020-10-20T13:45:57.00Z",
            "metadata": {}
        },
        "dewPoint": {
            "type": "Number",
            "value": 13.2,
            "metadata": {}
        }
}'
```

Get historical data

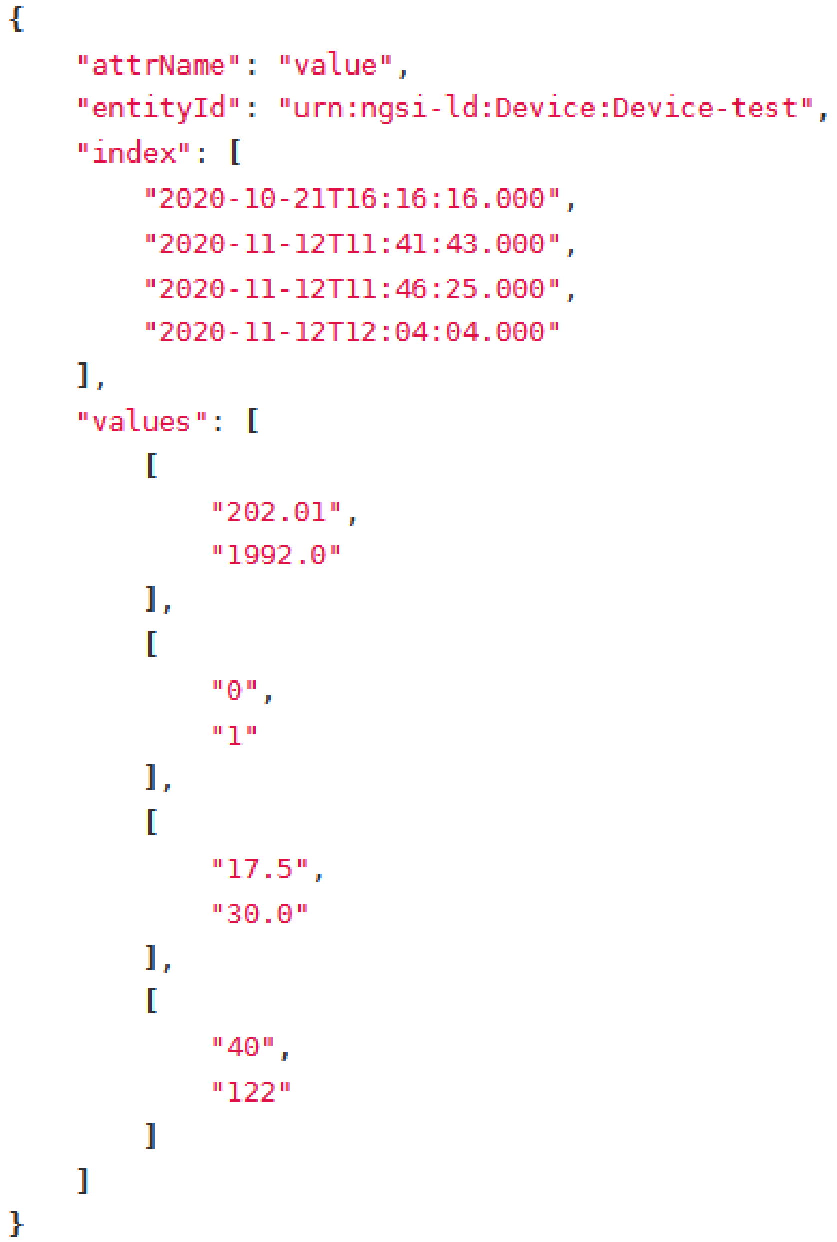
- GET HTTP request to QuantumLeap:



```
curl --location --request GET 'http://5.53.108.182:8668/v2/entities/urn:ngsi-ld:Device:Device-test/attrs/value' \
--header 'Fiware-Service: carouge' \
--header 'Fiware-ServicePath: /'
```

Get historical data

- Answer from QuantumLeap:



```
{
    "attrName": "value",
    "entityId": "urn:ngsi-ld:Device:Device-test",
    "index": [
        "2020-10-21T16:16:16.000",
        "2020-11-12T11:41:43.000",
        "2020-11-12T11:46:25.000",
        "2020-11-12T12:04:04.000"
    ],
    "values": [
        [
            "202.01",
            "1992.0"
        ],
        [
            "0",
            "1"
        ],
        [
            "17.5",
            "30.0"
        ],
        [
            "40",
            "122"
        ]
    ]
}
```

Final remarks

- The different elements presented in the slides are implemented by the Data Collection & Aggregation (DCA) of each pilot (Alicante, Braila and Carouge).
- The NAIADES IoT platform is supporting NGSIv2 and NGSI-LD.
- The IoT platform can use data from other IoT verticals using FIWARE components.

More information

- Source code of the NAIADES IoT platform: https://gitlab.distantaccess.com/naiades/naiades-platform-poc
- Documentation / Wiki: https://gitlab.distantaccess.com/naiades/naiades-platform-poc/-/wikis/home
- NAIADES website: https://www.naiades-project.eu/
- Contact: Cédric Crettaz ccrettaz@udgalliance.org

Thank you

'This project (NAIADES) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 820985'.

'The opinions expressed in this document reflect only the author's view and reflects in no way the European Commission's opinions. The European Commission is not responsible for any use that may be made of the information it contains.'

NAIADES

Webinar Series

IoT Technologies for Smart Water Systems

NOV 16, 2021
10-12.30 CET

ONLINE

Join Us!

disy

UDG Alliance

Jožef Stefan Institute

NAIADES Speakers

Fraunhofer IOSB

KOMPETENZZENTRUM WasserBerlin

DATA COVE

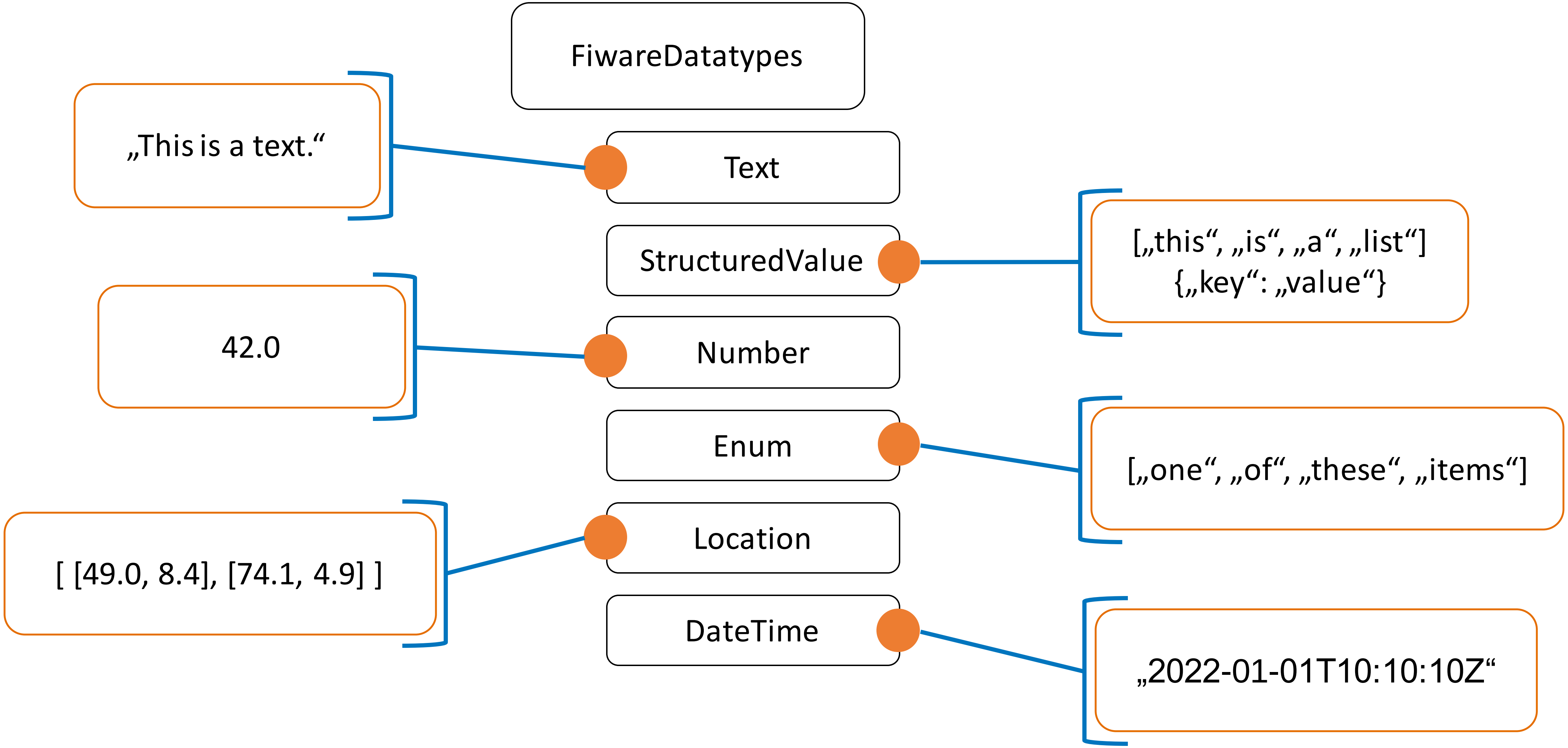
External Speakers

Adding new Data Models

- Shaping Models
  - Available Datatypes
  - Creating custom Specification Model
  - Creating custom Blueprint Model
- Integration
- Live Demo

Shaping Models

Shaping models
Integration
Live Demo



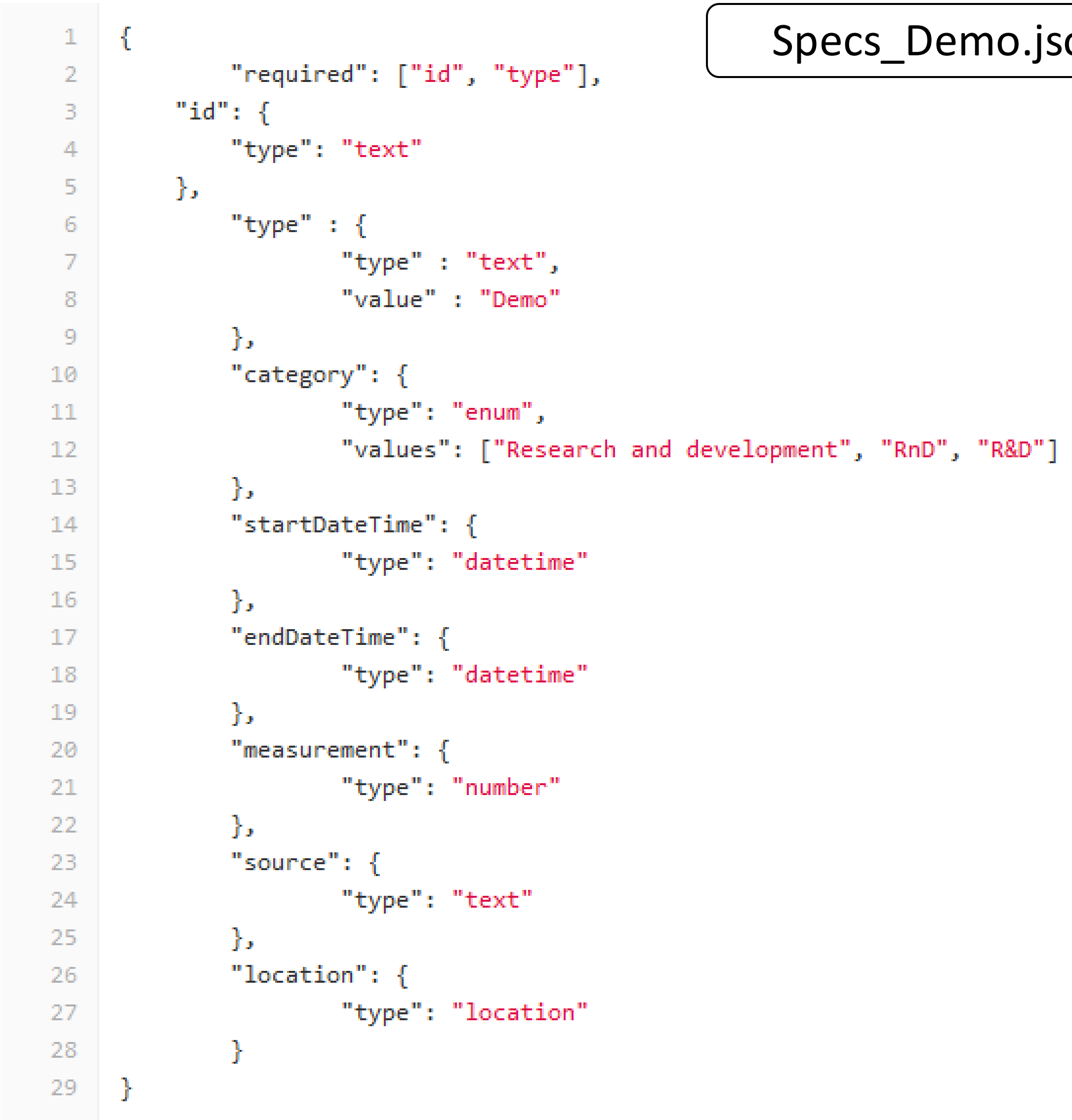
Shaping Models

- What data is needed to characterize a use case?
- Which attributes are sufficient to describe it?

Shaping Models

Shaping models — Integration — Live Demo

Specs_Demo.json



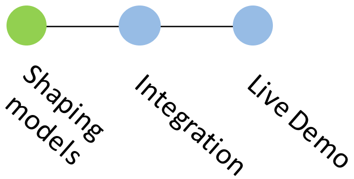
```
{
        "required": ["id", "type"],
    "id": {
        "type": "text"
    },
        "type" : {
                "type" : "text",
                "value" : "Demo"
        },
        "category": {
                "type": "enum",
                "values": ["Research and development", "RnD", "R&D"]
        },
        "startDateTime": {
                "type": "datetime"
        },
        "endDateTime": {
                "type": "datetime"
        },
        "measurement": {
                "type": "number"
        },
        "source": {
                "type": "text"
        },
        "location": {
                "type": "location"
        }
}
```

- What data is needed to characterize a use case?
- Which attributes are sufficient to describe it?

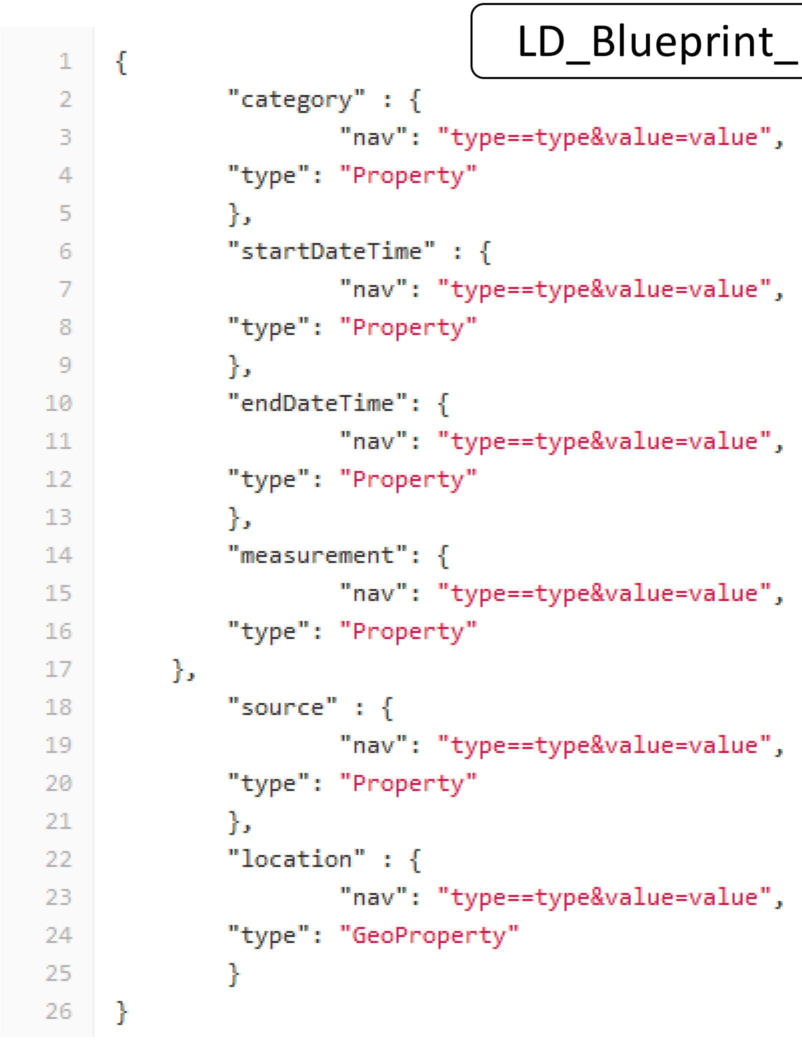
Shaping Models

How can conversion between Fiware Model Versions be managed?

Shaping Models



LD_Blueprint_Demo.json

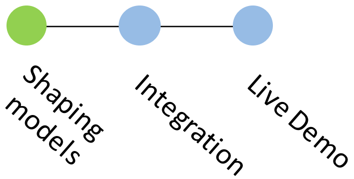


```
{
        "category" : {
                "nav": "type==type&value=value",
        "type": "Property"
        },
        "startDateTime" : {
                "nav": "type==type&value=value",
        "type": "Property"
        },
        "endDateTime": {
                "nav": "type==type&value=value",
        "type": "Property"
        },
        "measurement": {
                "nav": "type==type&value=value",
        "type": "Property"
    },
        "source" : {
                "nav": "type==type&value=value",
        "type": "Property"
        },
        "location" : {
                "nav": "type==type&value=value",
        "type": "GeoProperty"
        }
}
```

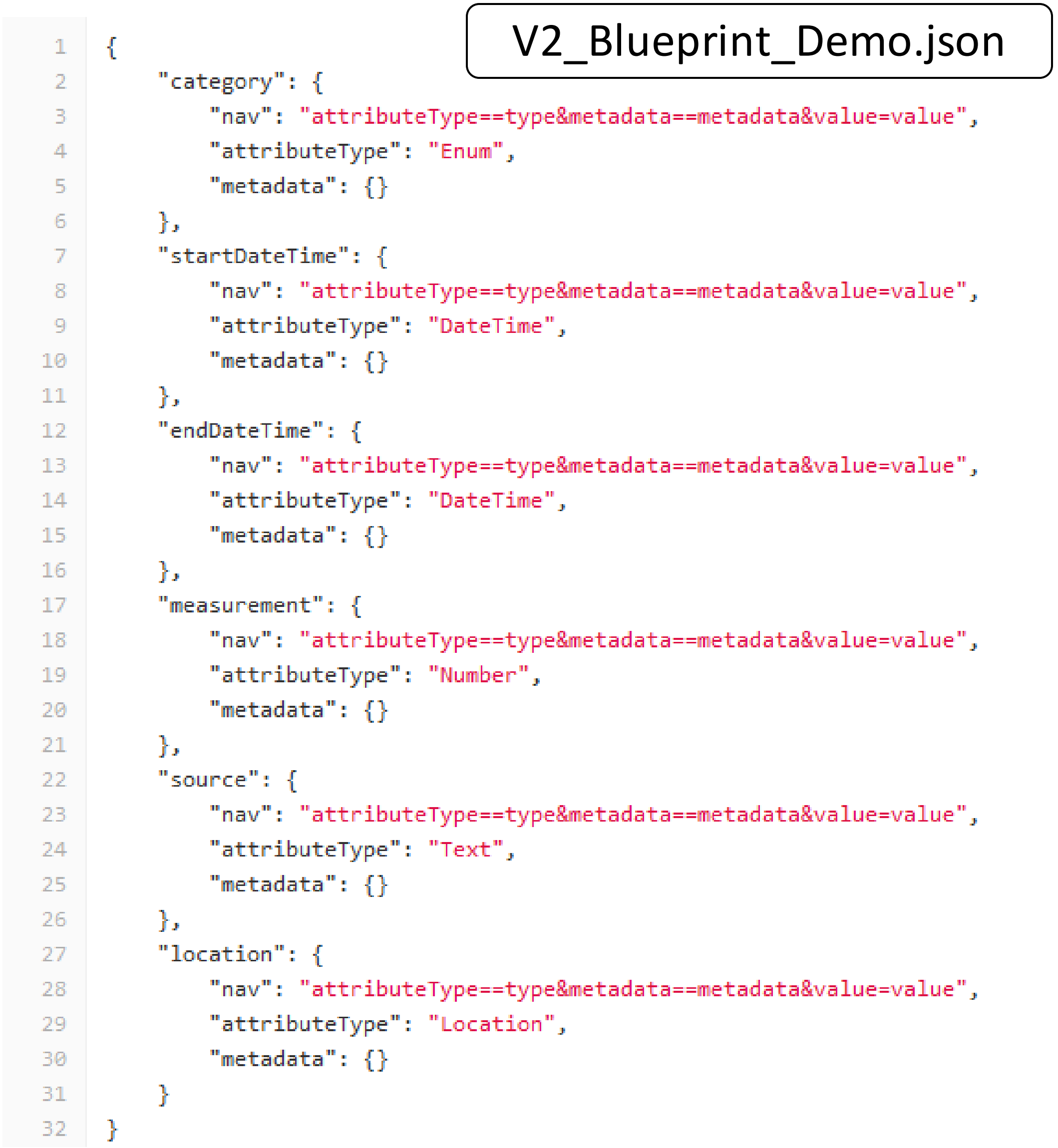
- Comparison ==
- Assignment =
- Additional **type** specification

How can conversion between Fiware Model Versions be managed?

Shaping Models



V2_Blueprint_Demo.json

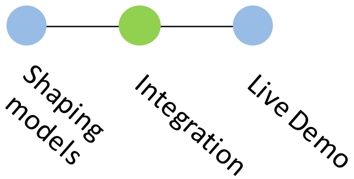


```
{
    "category": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "Enum",
        "metadata": {}
    },
    "startDateTime": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "DateTime",
        "metadata": {}
    },
    "endDateTime": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "DateTime",
        "metadata": {}
    },
    "measurement": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "Number",
        "metadata": {}
    },
    "source": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "Text",
        "metadata": {}
    },
    "location": {
        "nav": "attributeType==type&metadata==metadata&value=value",
        "attributeType": "Location",
        "metadata": {}
    }
}
```

- Comparison ==
- Assignment =
- Additional **type** specification

How can conversion between Fiware Model Versions be managed?

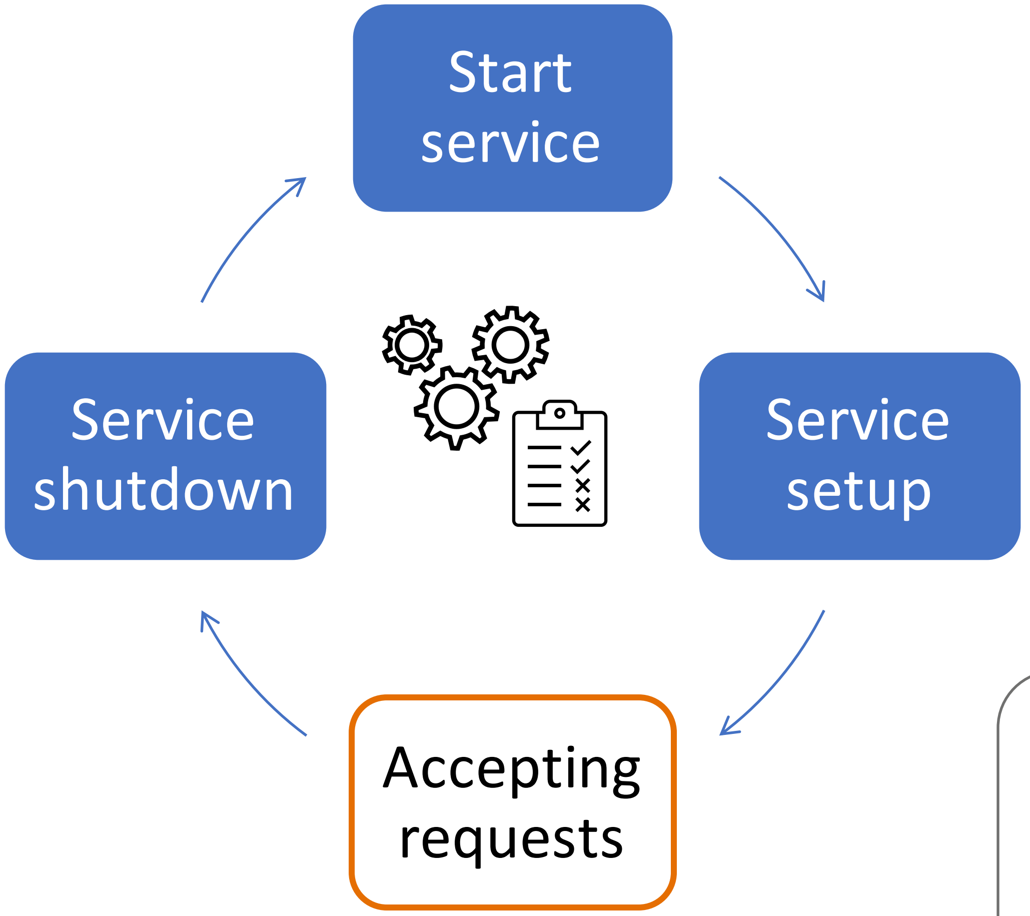
Uploading Models



Shaping models

Integration

Live Demo

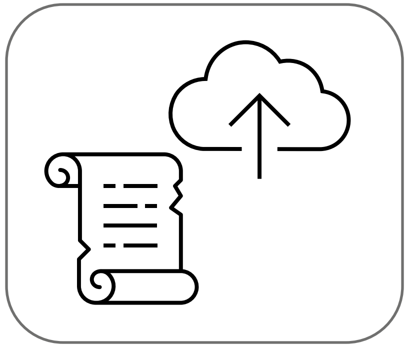


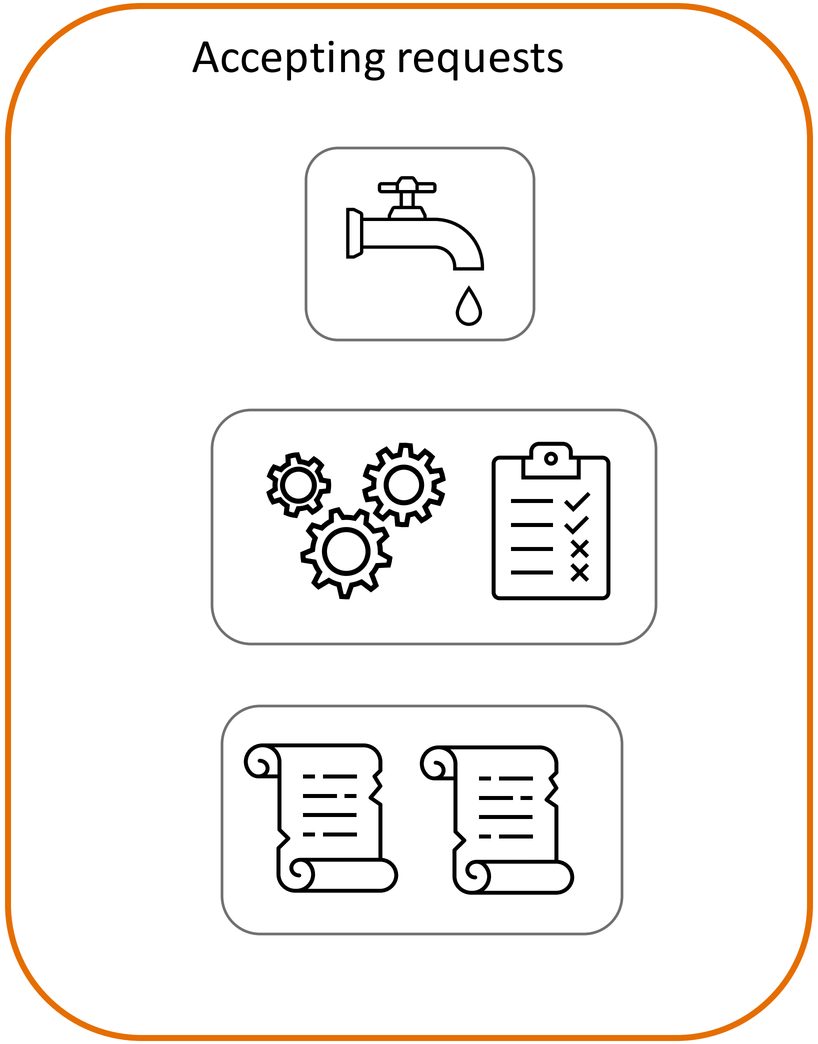
Start service

Service setup

Accepting requests

Service shutdown





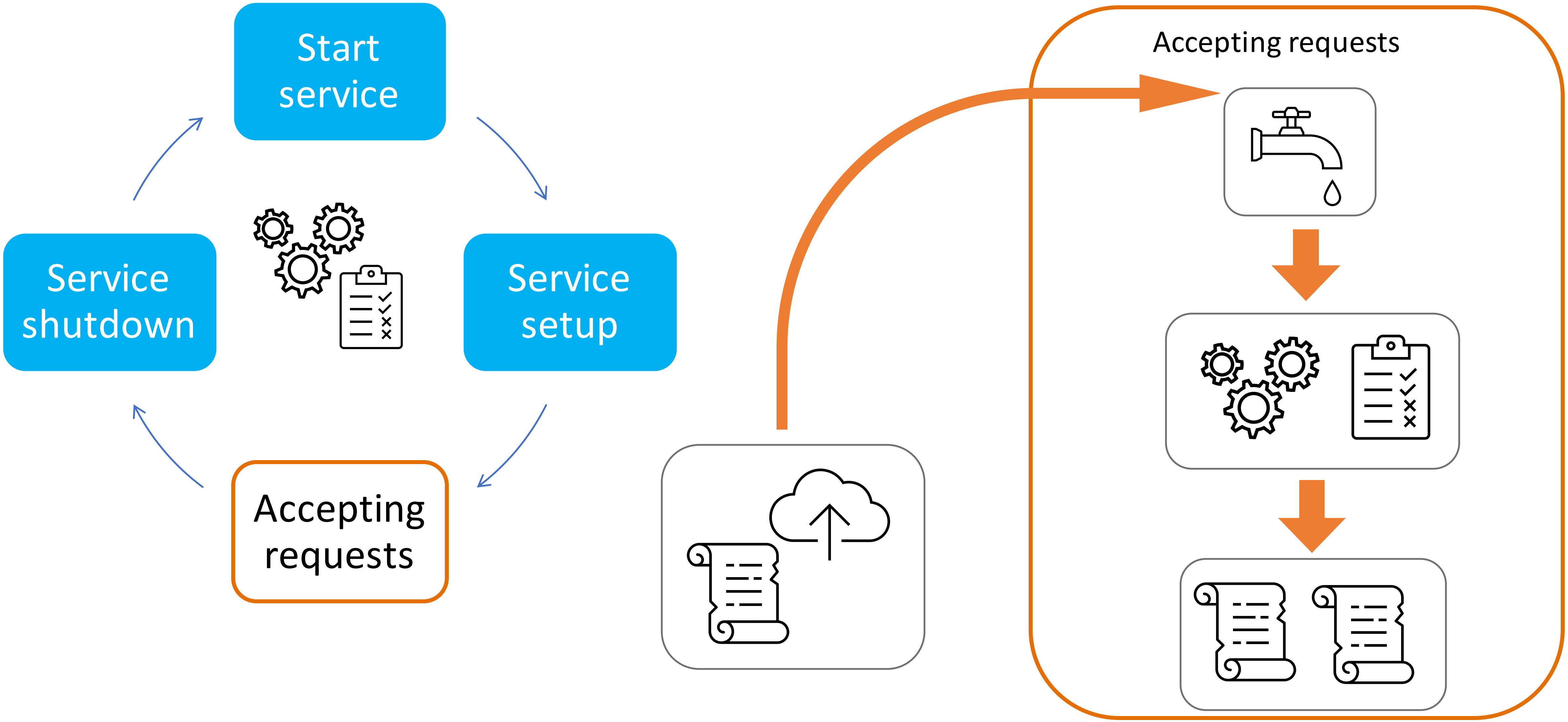
Accepting requests

Uploading Models

Shaping models

Integration

Live Demo



Start service

Service setup

Accepting requests

Service shutdown

Accepting requests

Live Demo

Shaping models

Integration

Live Demo

THANK YOU

Follow us on social media

@naiadesproject

@naiadesproject

www.naiades-project.eu

& Samples!

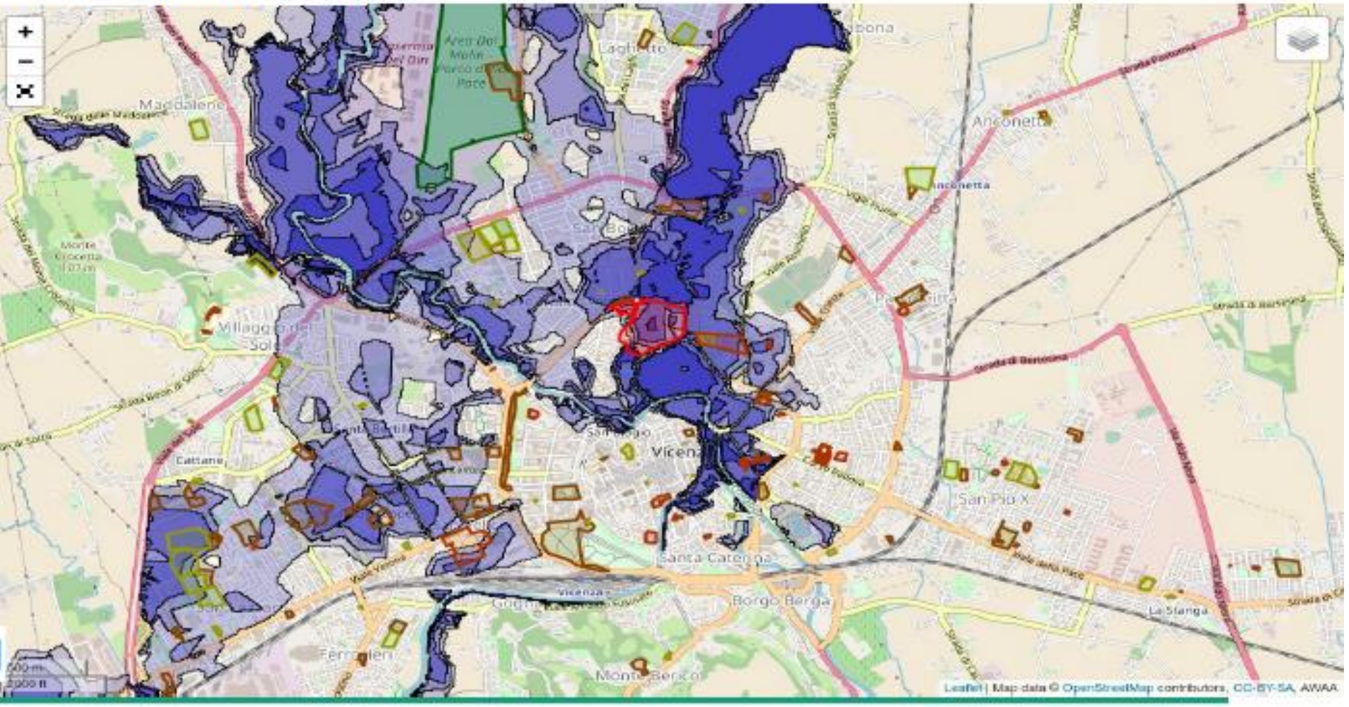
OGC/ISO Observations & Measurements and the OGC SensorThings API

Hylke van der Schaaf

Fraunhofer IOSB

Open Geospatial Consortium

- International consortium
  - over 540 companies, government agencies and universities
- "Geo-enable" mainstream IT
- Develop publicly available interface standards
  - Maps (Web Map Service)
  - CityGML
  - WaterML
  - Earth Observations
- Conformance testing

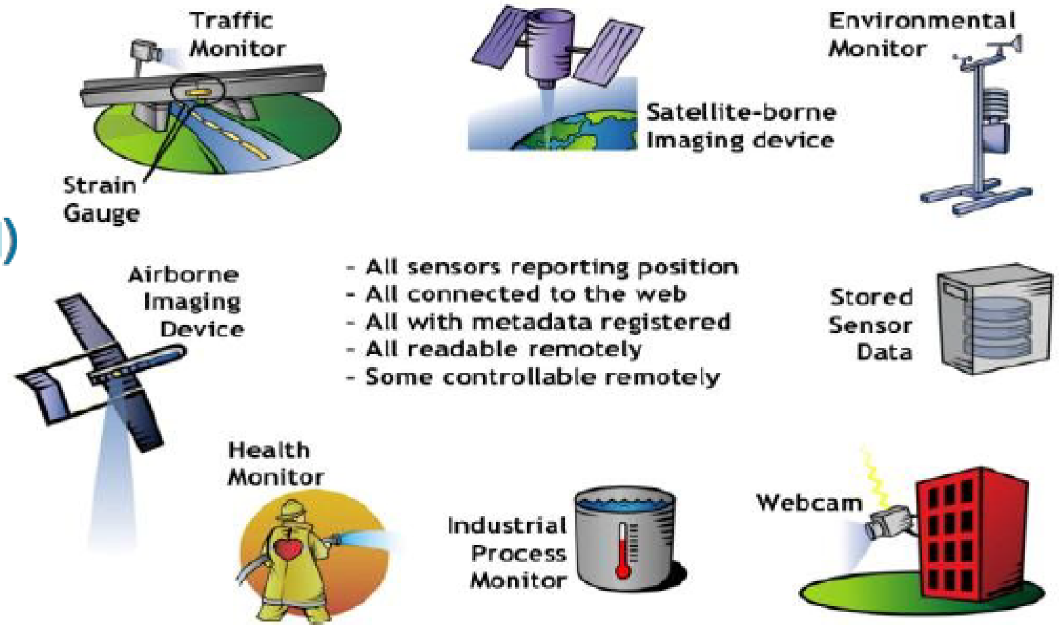


Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, AWAA

- http://www.opengeospatial.org

OGC & Observations?

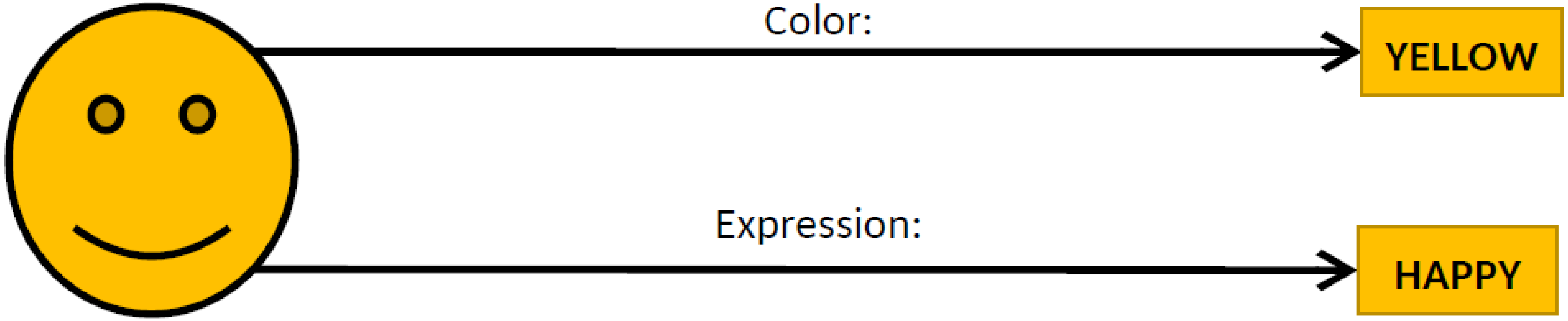
- Observations are made **somewhere**!
- Often by Sensors
- OGC Sensor Web Enablement (SWE)
  - Enable developers to make *all types* of sensors, transducers and sensor data repositories discoverable, accessible and useable via the Web
  - Since 1990 by NASA
  - Since 2001 in OGC
  - SensorML
Observations & Measurements (O&M)
SensorThings API
  - Sensor Data & Metadata



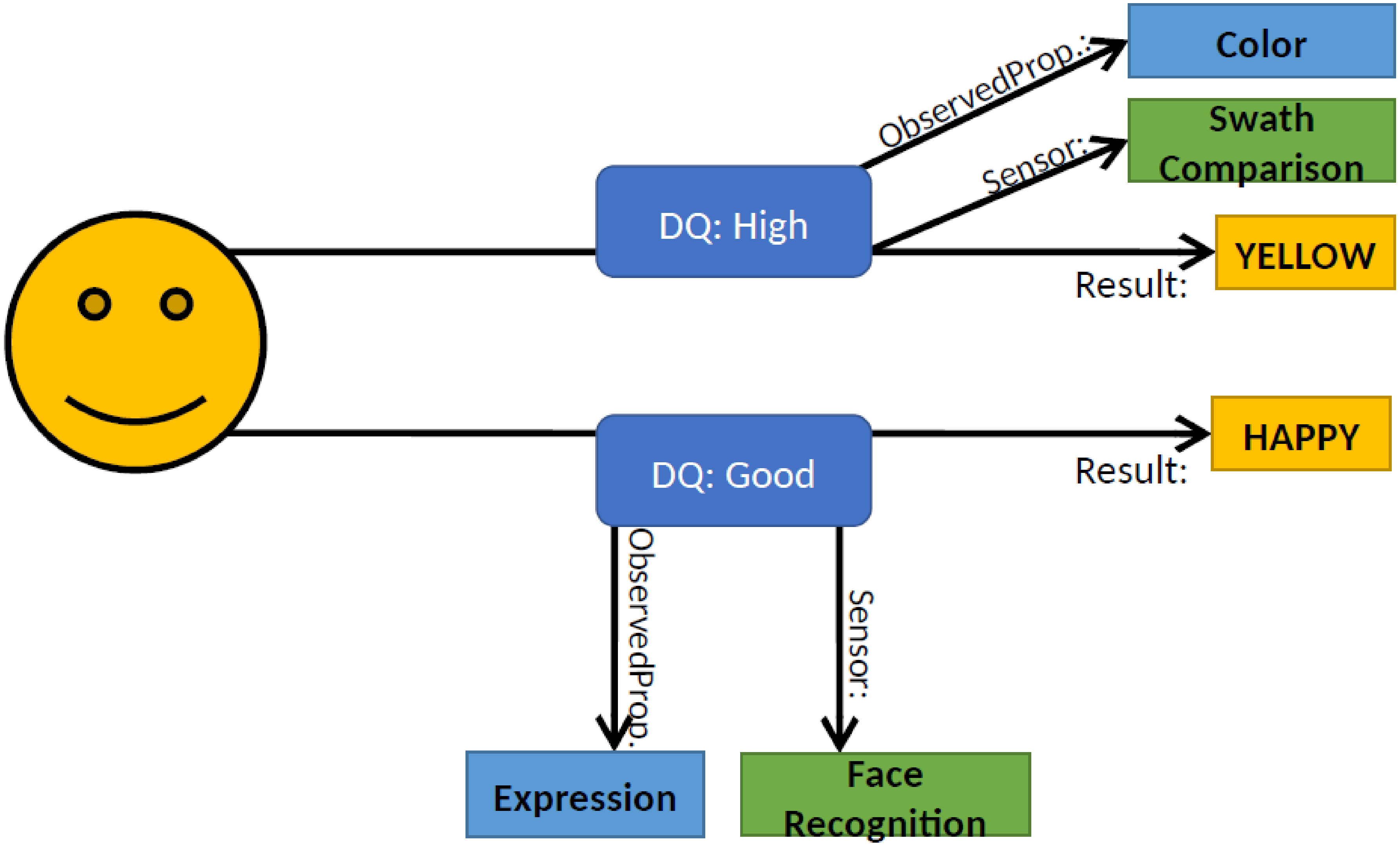
Traffic Monitor
Satellite-borne Imaging device
Environmental Monitor
Strain Gauge
Airborne Imaging Device
- All sensors reporting position
- All connected to the web
- All with metadata registered
- All readable remotely
- Some controllable remotely
Stored Sensor Data
Health Monitor
Industrial Process Monitor
Webcam

©OGC: http://www.opengeospatial.org/ogc/markets-technologies/swe

Observational (Meta)Data

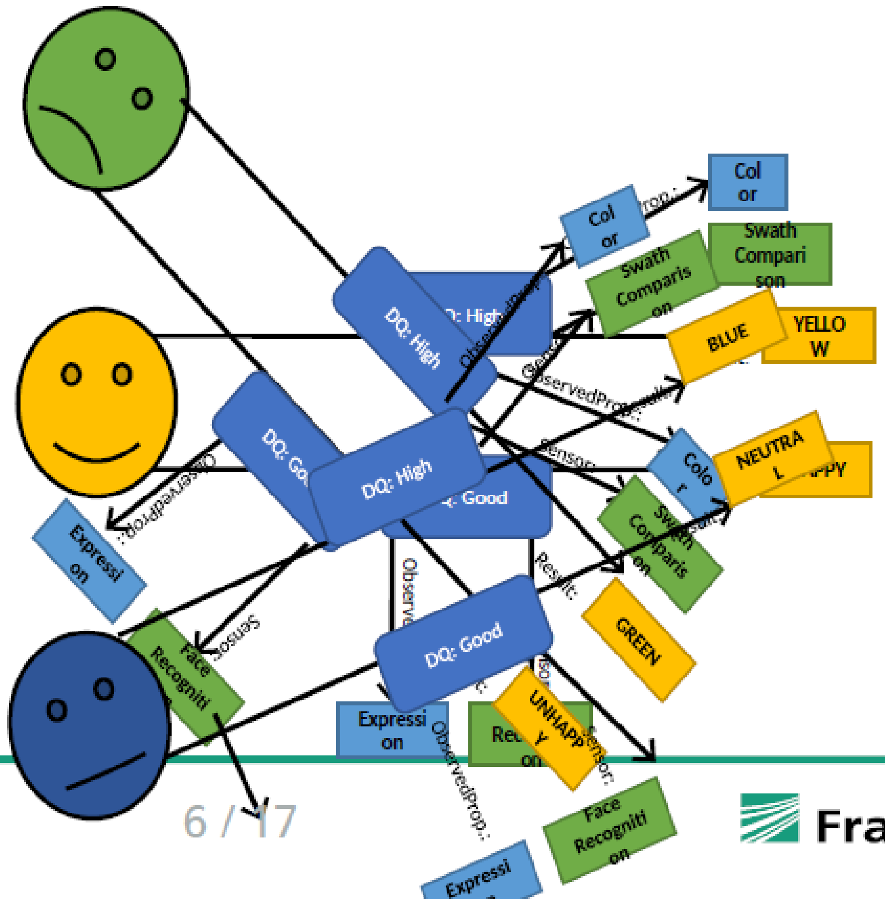


Observational (Meta)Data



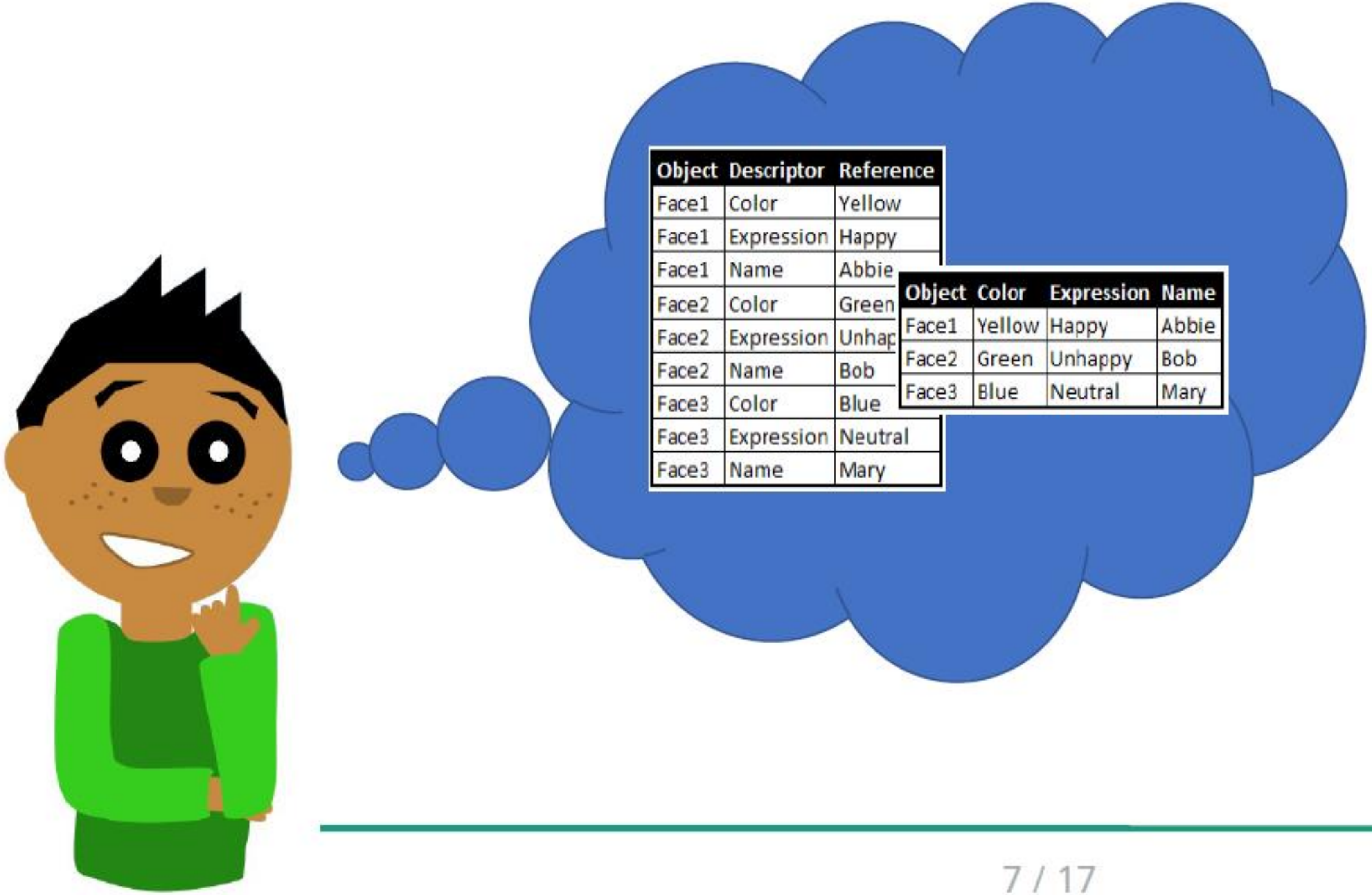
Using Observational (Meta)Data

Using Observational (Meta)Data



Fraunhofer
IOSB

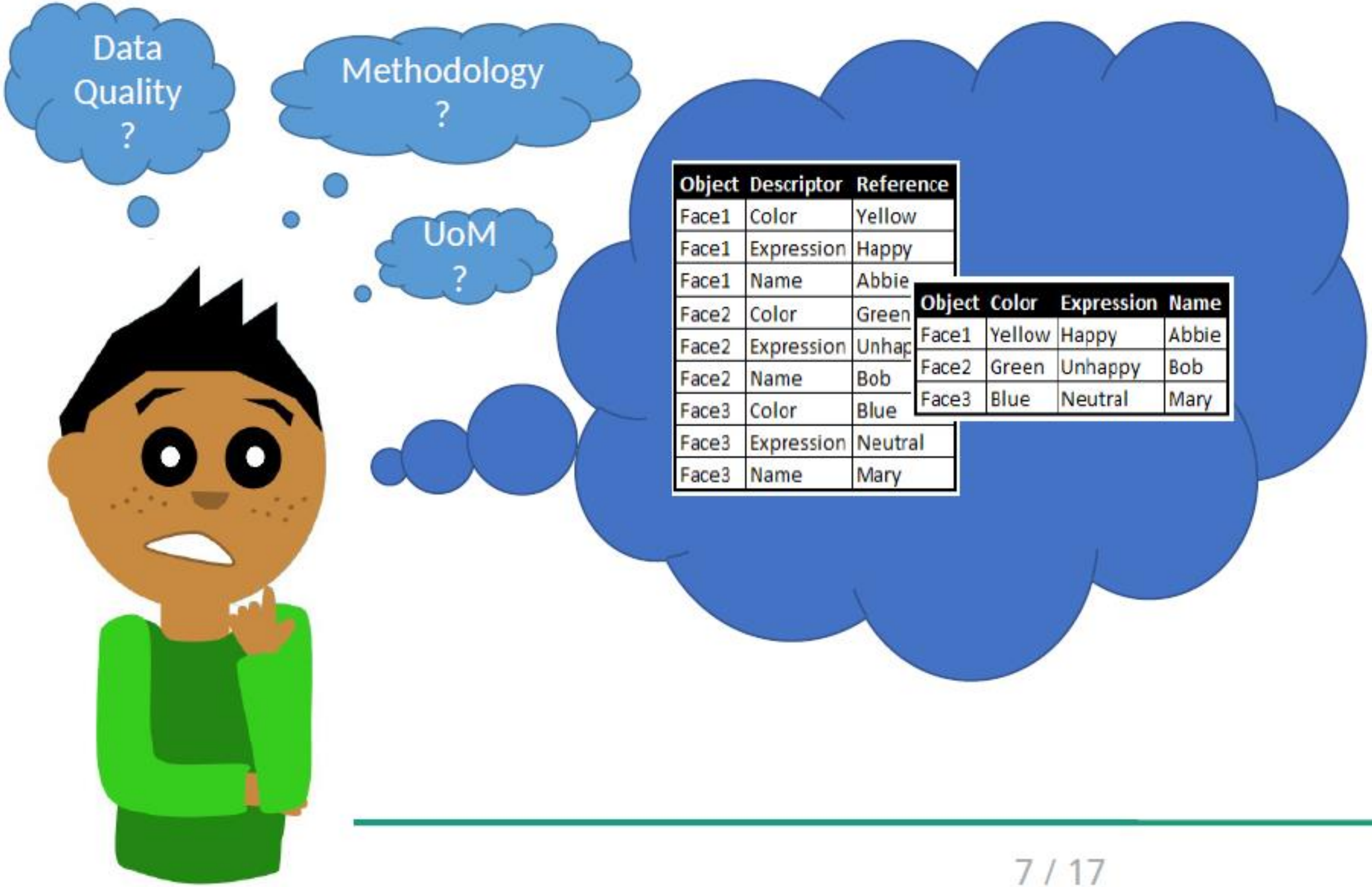
Using Observational (Meta)Data



Object	Descriptor	Reference
Face1	Color	Yellow
Face1	Expression	Happy
Face1	Name	Abbie
Face2	Color	Green
Face2	Expression	Unhap
Face2	Name	Bob
Face3	Color	Blue
Face3	Expression	Neutral
Face3	Name	Mary

Object	Color	Expression	Name
Face1	Yellow	Happy	Abbie
Face2	Green	Unhappy	Bob
Face3	Blue	Neutral	Mary

Using Observational (Meta)Data

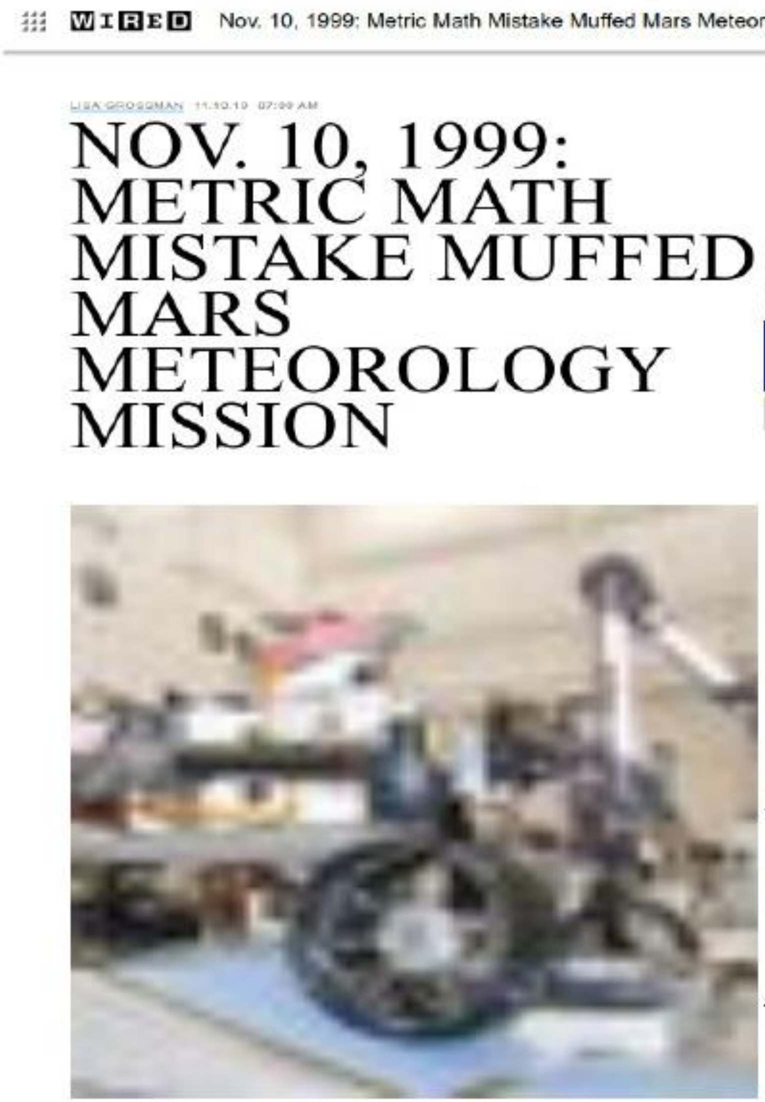


Object	Descriptor	Reference
Face1	Color	Yellow
Face1	Expression	Happy
Face1	Name	Abbie
Face2	Color	Green
Face2	Expression	Unhap
Face2	Name	Bob
Face3	Color	Blue
Face3	Expression	Neutral
Face3	Name	Mary

Object	Color	Expression	Name
Face1	Yellow	Happy	Abbie
Face2	Green	Unhappy	Bob
Face3	Blue	Neutral	Mary

Observations & Measurements Model

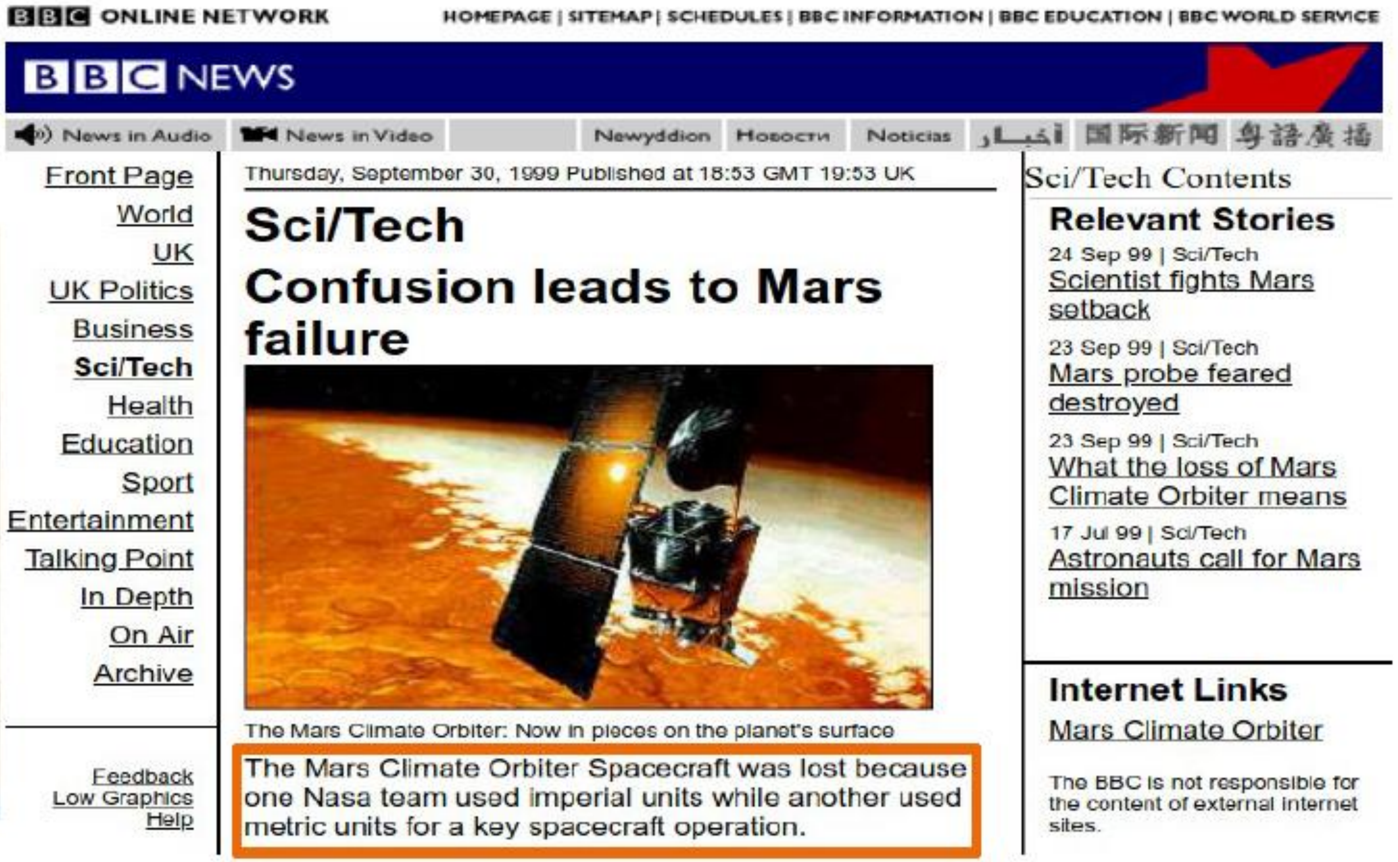
Sensor Metadata!



WIRED Nov. 10, 1999: Metric Math Mistake Muffed Mars Meteorology Mission

LISA GROSSMAN 11.10.10 07:00 AM

NOV. 10, 1999: METRIC MATH MISTAKE MUFFED MARS METEOROLOGY MISSION



BBC ONLINE NETWORK

HOMEPAGE | SITEMAP | SCHEDULES | BBC INFORMATION | BBC EDUCATION | BBC WORLD SERVICE

BBC NEWS

News in Audio | News in Video | Newyddion | Новости | Noticias | أخبار | 国际新闻 | 粤語廣播

Front Page
World
UK
UK Politics
Business
Sci/Tech
Health
Education
Sport
Entertainment
Talking Point
In Depth
On Air
Archive

Feedback
Low Graphics
Help

Thursday, September 30, 1999 Published at 18:53 GMT 19:53 UK

Sci/Tech

Confusion leads to Mars failure

The Mars Climate Orbiter: Now in pieces on the planet's surface

The Mars Climate Orbiter Spacecraft was lost because one Nasa team used imperial units while another used metric units for a key spacecraft operation.

Sci/Tech Contents

Relevant Stories

24 Sep 99 | Sci/Tech
Scientist fights Mars setback

23 Sep 99 | Sci/Tech
Mars probe feared destroyed

23 Sep 99 | Sci/Tech
What the loss of Mars Climate Orbiter means

17 Jul 99 | Sci/Tech
Astronauts call for Mars mission

Internet Links

Mars Climate Orbiter

The BBC is not responsible for the content of external internet sites.

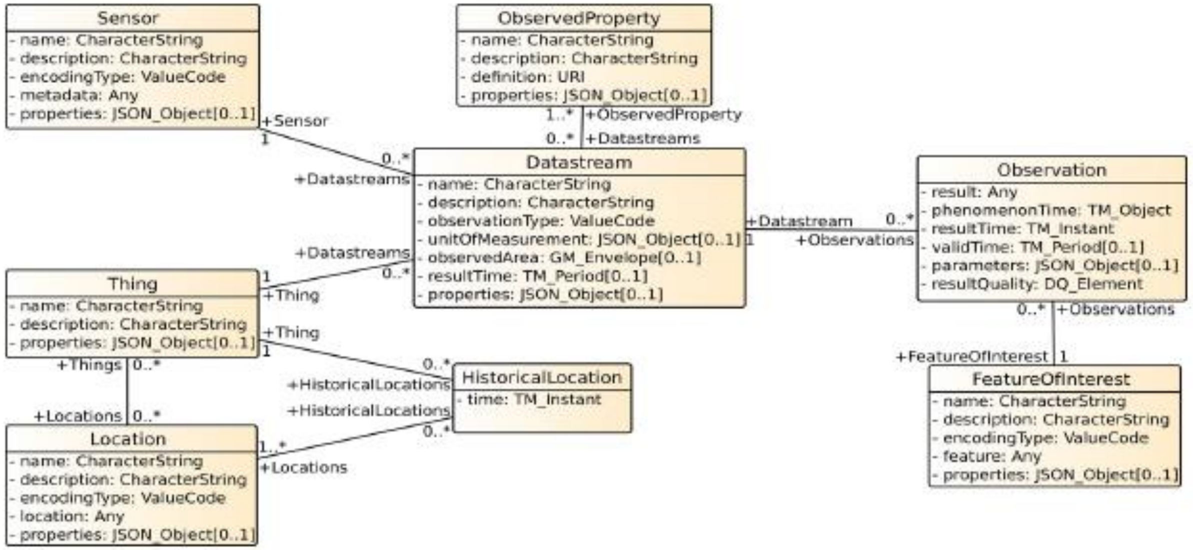
OGC SensorThings API

- A standard for exchanging sensor data <u>and metadata</u>
  - Historic data & current data
  - JSON Encoded
  - RESTful
  - Adapting OASIS OData URL patterns and query options
  - Supporting ISO MQTT messaging

- Easy to use & understandable
  - Discoverable with only a web browser

How does it work?

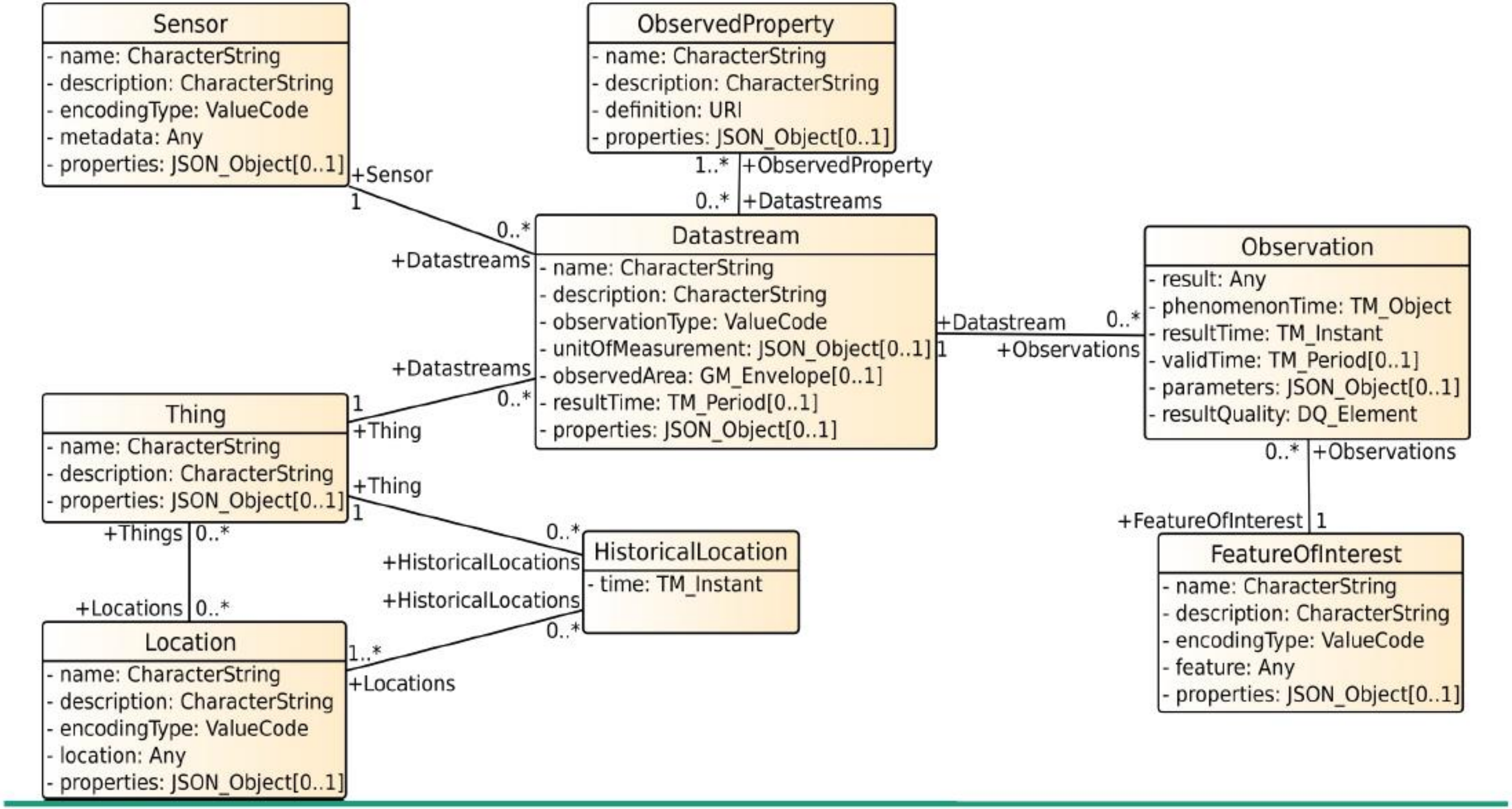
- Part 1: Data model
  - Which entities exist
  - How are they linked


Sensor
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- metadata: Any
- properties: JSON_Object[0..1]
ObservedProperty
- name: CharacterString
- description: CharacterString
- definition: URI
- properties: JSON_Object[0..1]
+Sensor
1
0..*
+Datastreams
1..*
+ObservedProperty
0..*
+Datastreams
Datastream
- name: CharacterString
- description: CharacterString
- observationType: ValueCode
- unitOfMeasurement: JSON_Object[0..1]
- observedArea: GM_Envelope[0..1]
- resultTime: TM_Period[0..1]
- properties: JSON_Object[0..1]
+Datastream
1
0..*
+Observations
Observation
- result: Any
- phenomenonTime: TM_Object
- resultTime: TM_Instant
- validTime: TM_Period[0..1]
- parameters: JSON_Object[0..1]
- resultQuality: DQ_Element
0..*
+Observations
+FeatureOfInterest
1
FeatureOfInterest
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- feature: Any
- properties: JSON_Object[0..1]
+Datastreams
0..*
1
+Thing
Thing
- name: CharacterString
- description: CharacterString
- properties: JSON_Object[0..1]
+Thing
1
0..*
+HistoricalLocations
HistoricalLocation
- time: TM_Instant
+HistoricalLocations
0..*
1..*
+Locations
+Things
0..*
+Locations
0..*
Location
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- location: Any
- properties: JSON_Object[0..1]

- Part 2: URL patterns for queries
  - How do I get & search data
  - How do I add data
  - How do I modify data
  - How do I delete data

REST
&
MQTT

Data model



Sensor
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- metadata: Any
- properties: JSON_Object[0..1]

ObservedProperty
- name: CharacterString
- description: CharacterString
- definition: URI
- properties: JSON_Object[0..1]

Datastream
- name: CharacterString
- description: CharacterString
- observationType: ValueCode
- unitOfMeasurement: JSON_Object[0..1]
- observedArea: GM_Envelope[0..1]
- resultTime: TM_Period[0..1]
- properties: JSON_Object[0..1]

Observation
- result: Any
- phenomenonTime: TM_Object
- resultTime: TM_Instant
- validTime: TM_Period[0..1]
- parameters: JSON_Object[0..1]
- resultQuality: DQ_Element

Thing
- name: CharacterString
- description: CharacterString
- properties: JSON_Object[0..1]

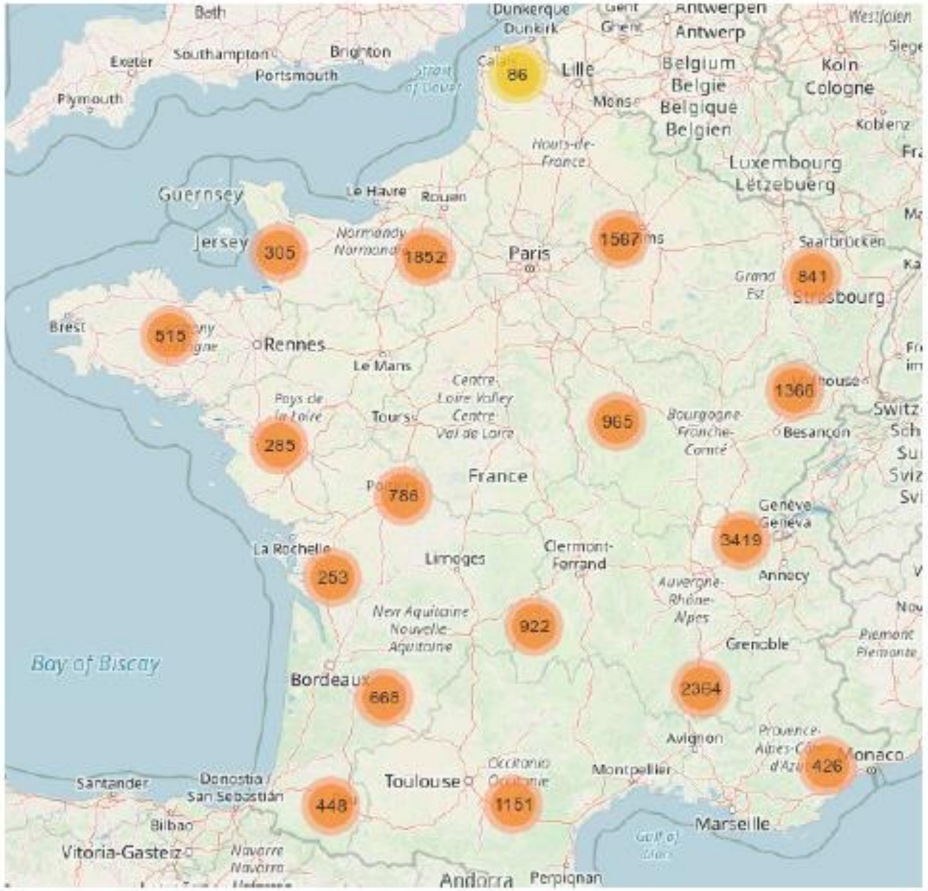
HistoricalLocation
- time: TM_Instant

Location
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- location: Any
- properties: JSON_Object[0..1]

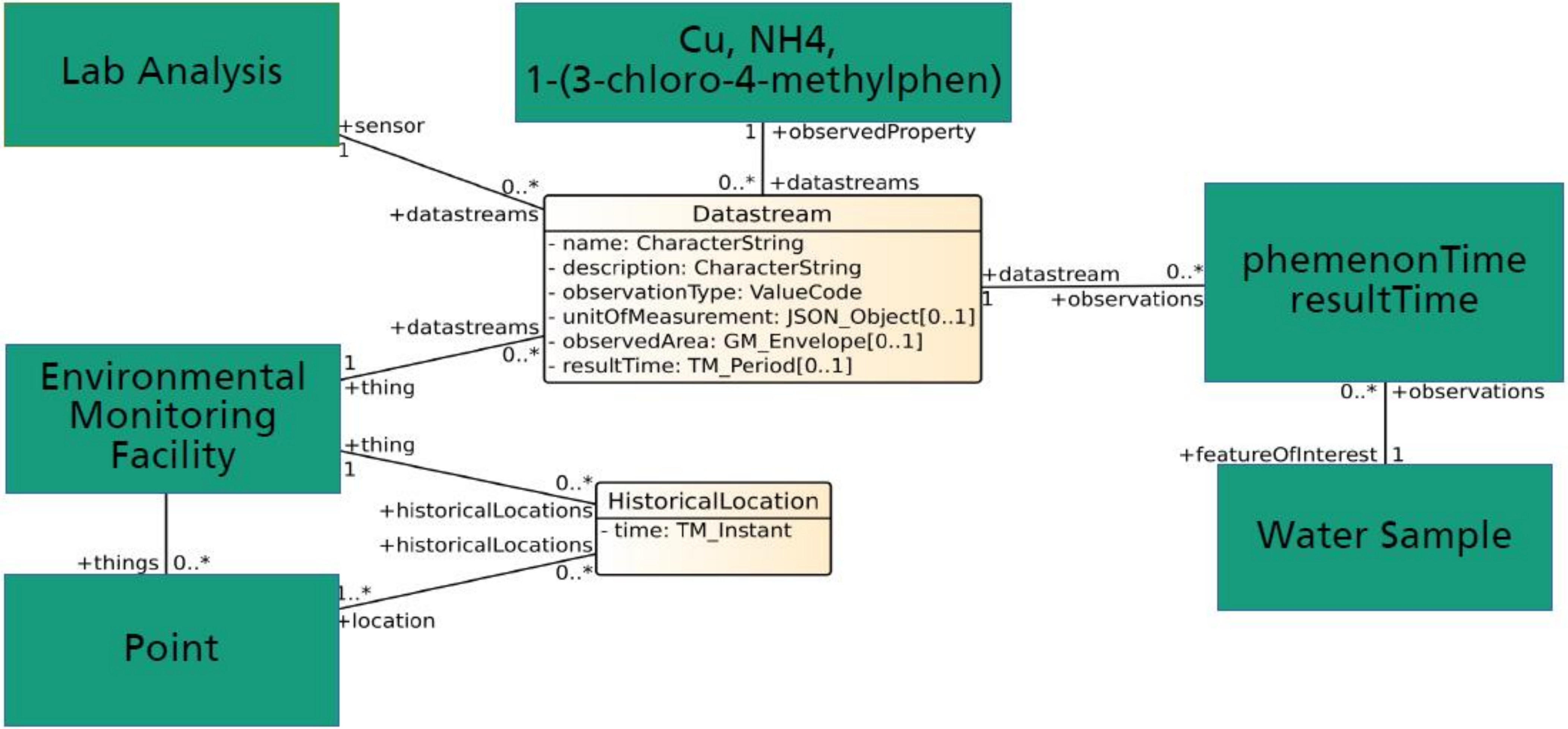
FeatureOfInterest
- name: CharacterString
- description: CharacterString
- encodingType: ValueCode
- feature: Any
- properties: JSON_Object[0..1]

Examples: BRGM – French surface water database

- French surface water quality database
  - 18478 Stations
    1874 Observed Properties
    136000000 Observations
  - INSPIRE Aligned
  - Water samples
    - analysed in laboratory
    - many results per sample



Data model – BRGM Water Quality



Lab Analysis

+sensor 1

0..* +datastreams

Cu, NH4,
1-(3-chloro-4-methylphen)

1 +observedProperty

0..* +datastreams

Datastream
- name: CharacterString
- description: CharacterString
- observationType: ValueCode
- unitOfMeasurement: JSON_Object[0..1]
- observedArea: GM_Envelope[0..1]
- resultTime: TM_Period[0..1]

+datastream 1

0..* +observations

phemenonTime
resultTime

0..* +observations

+featureOfInterest 1

Water Sample

Environmental
Monitoring
Facility

+datastreams 0..*

1 +thing

+thing 1

0..* +historicalLocations

HistoricalLocation
- time: TM_Instant

+historicalLocations 0..*

1..* +location

+things 0..*

Point

Getting to your data

- Based on OASIS OData
- Base URL: http://server.de/FROST-Server/v1.1
- Read: GET
  - v1.1 → Get collection index
  - v1.1/Collection → Get all entities in a collection
  - v1.1/Collection(id) → Get one entity from a collection
  - v1.1/Collection(id)/Relation → Get related entities
- Create: POST
  - v1.1/Collection → Create a new entity
- Update: PATCH
  - v1.1/Collection(id) → Update an entity
- Update: PUT
  - v1.1/Collection(id) → Replace an entity
- Delete: DELETE
  - v1.1/Collection(id) → Remove an entity

Getting to your data

- **$top:** Limit returned # of items
- **$skip:** Skip first # items
- **$count:** Count items
- **$orderBy:** Sort items
- **$select:** Limit returned properties
- **$filter:** Filter items
- **$expand:** Return related items

Questions?

- Hylke van der Schaaf
  - hylke.vanderschaaf@iosb.fraunhofer.de
- O&M
  - https://www.ogc.org/standards/om
  - https://en.wikipedia.org/wiki/Observations_and_Measurements
- SensorThings API
  - https://www.ogc.org/standards/sensorthings
- FROST-Server
  - https://github.com/FraunhoferIOSB/FROST-Server
  - https://fraunhoferiosb.github.io/FROST-Server/