PLATOON

Digital platform and analytic tools for energy

Task 2.4 – Data Integration (V2)

Status of FQP-Federated Query Processing Component

Requirements

- Data Formats
  - Data is transformed by ENGIE to JSON-LD Flatten form
  - JSON-LD flat form (but not compacted!) have a simple embedded item formats (see https://www.w3.org/TR/json-ld11/#flattened-document-form)
    - **Flattening** collects all properties of a node in a **single map** and labels all blank nodes with blank node identifiers. This *ensures a shape of the data and consequently may drastically simplify the code required* to process JSON-LD in certain applications.
- Data Storage
  - JSON-LD documents are stored in MongoDB
- Query requirement
  - Simple SPARQL BGP queries over MongoDB documents
  - Transform results to SPARQL JSON Format

PLATOON

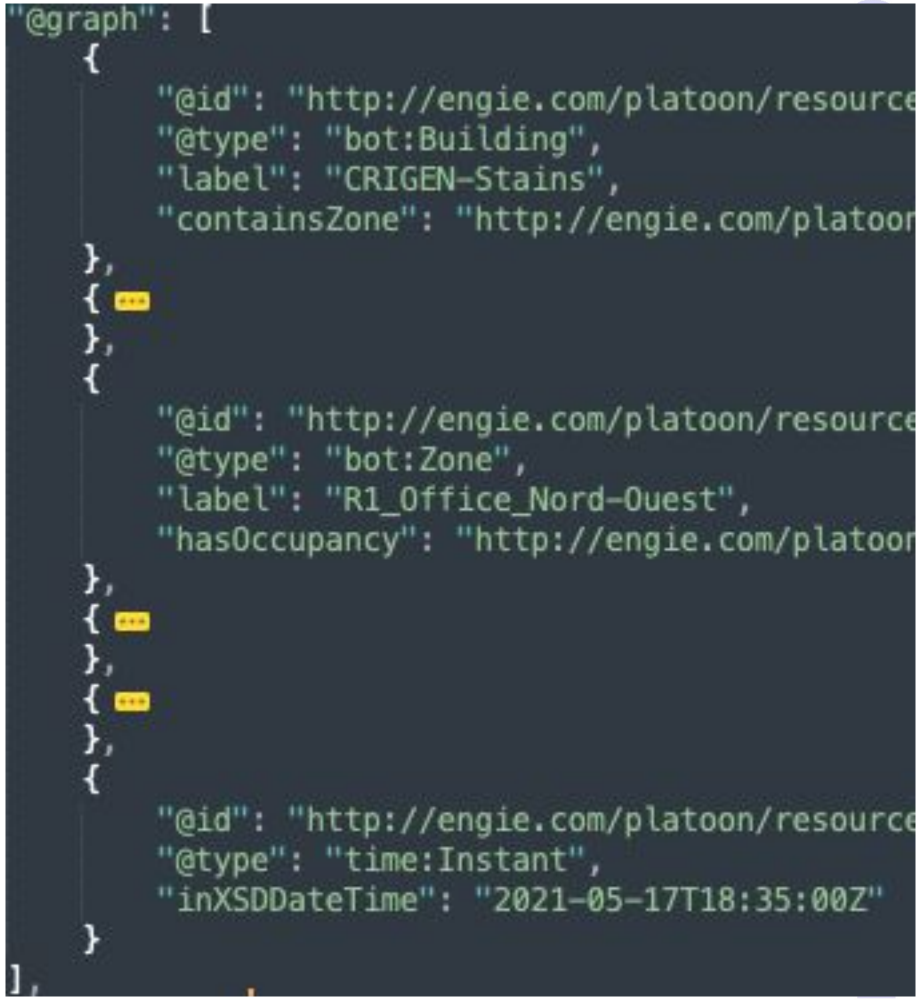
Requirement: JSON-LD Flattened Form

- JSON-LD Flat Form

```
{
"@context": {
            "containsZone": {
               "@id": "https://w3id.org/bot#containsZone",
               "@type": "@id"
            }, …,
           "owl": "http://www.w3.org/2002/07/owl#",
       },
"@graph": [
                {
                "@id":   " ",
                "@type":  " ",
                 "containsZone":    " ”
```

[No embeddings here! Strings]

```
                 }, { … },
           ]
}
```

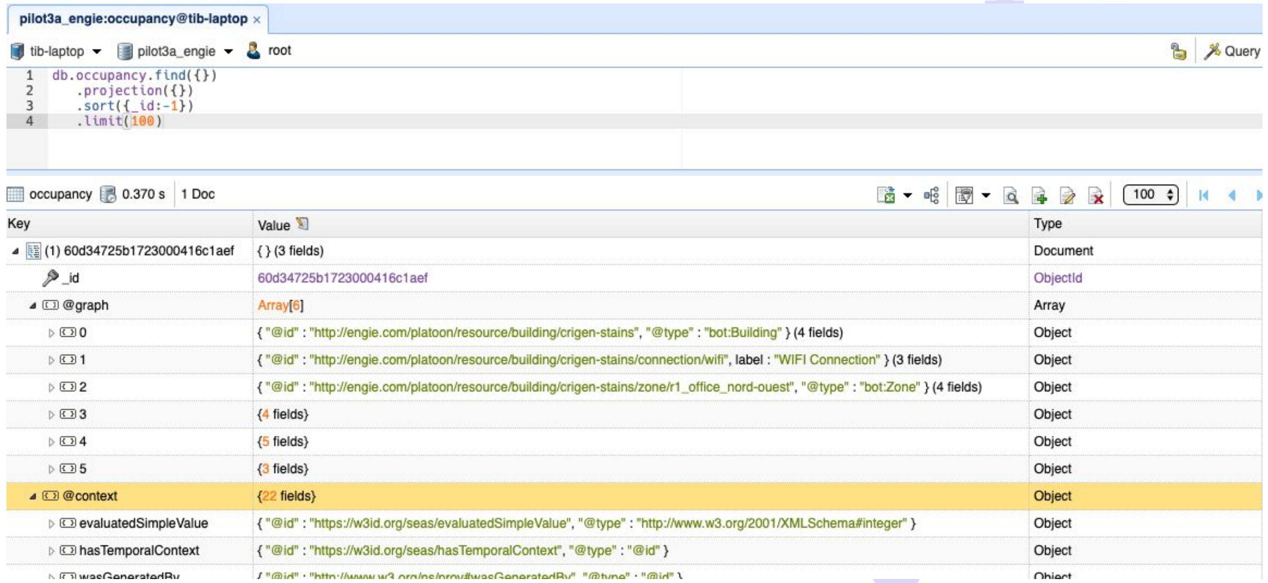


```
"@graph": [
    {
        "@id": "http://engie.com/platoon/resource
        "@type": "bot:Building",
        "label": "CRIGEN-Stains",
        "containsZone": "http://engie.com/platoon
    },
    { ...
    },
    {
        "@id": "http://engie.com/platoon/resource
        "@type": "bot:Zone",
        "label": "R1_Office_Nord-Ouest",
        "hasOccupancy": "http://engie.com/platoon
    },
    { ...
    },
    { ...
    },
    {
        "@id": "http://engie.com/platoon/resource
        "@type": "time:Instant",
        "inXSDDateTime": "2021-05-17T18:35:00Z"
    }
],
```

PLATOON

Requirement: Data Storage

- Data is stored in MongoDB store (version >4.4 for $unionWith queries)
  - A database may contain multiple collections
  - Each collection can also have multiple semantic concepts, Building, Zone, Occupancy, etc.


pilot3a_engie:occupancy@tib-laptop

tib-laptop · pilot3a_engie · root · Query

```
db.occupancy.find({})
   .projection({})
   .sort({_id:-1})
   .limit(100)
```

occupancy 0.370 s 1 Doc · 100

Key	Value	Type
(1) 60d34725b1723000416c1aef	{ } (3 fields)	Document
_id	60d34725b1723000416c1aef	ObjectId
@graph	Array[6]	Array
0	{ "@id" : "http://engie.com/platoon/resource/building/crigen-stains", "@type" : "bot:Building" } (4 fields)	Object
1	{ "@id" : "http://engie.com/platoon/resource/building/crigen-stains/connection/wifi", label : "WIFI Connection" } (3 fields)	Object
2	{ "@id" : "http://engie.com/platoon/resource/building/crigen-stains/zone/r1_office_nord-ouest", "@type" : "bot:Zone" } (4 fields)	Object
3	{4 fields}	Object
4	{5 fields}	Object
5	{3 fields}	Object
@context	{22 fields}	Object
evaluatedSimpleValue	{ "@id" : "https://w3id.org/seas/evaluatedSimpleValue", "@type" : "http://www.w3.org/2001/XMLSchema#integer" }	Object
hasTemporalContext	{ "@id" : "https://w3id.org/seas/hasTemporalContext", "@type" : "@id" }	Object
wasGeneratedBy	{ "@id" : "http://www.w3.org/ns/prov#wasGeneratedBy", "@type" : "@id" }	Object

PLATOON

Requirement: Query Form

```
db.<collection_name_above>.find(
  {$and:[
          {"@graph":{$elemMatch:{ "@id":"http://engie.com/platoon/resource/windfarm/frcve/windturbine/80499/vane"}}},
          {"@graph.inXSDDateTime":{$gte:"2016-10-14T02:50:00Z"}},
          {"@graph.inXSDDateTime":{$lt:"2016-10-15T02:50:00Z"}}
        ]
    })
```

```
PREFIX wt: < http://engie.com/platoon/resource/windfarm/frcve/windturbine/>

SELECT *
WHERE {
    wt:80499/vane seas:averagePosition      ?positionProp .
    ?positionProp seas:evaluation            ?eval .
    ?eval          plt:temporalContext        ?context .
    ?eval         seas:evaluatedSimpleValue ?avgpositionValue.
    ?context      time:inXSDDateTime         ?time .

       FILTER (?time > xsd:dateTime("2016-10-14T02:50:00Z" &&
          ?time < xsd:dateTime("2016-10-15T02:50:00Z")
}
```

PLATOON

Supported Patterns (SPARQL 1.0)

```
PREFIX wt: < http://engie.com/platoon/resource/windfarm/frcve/windturbine/>

SELECT *
WHERE {
      wt:80499/vane seas:averagePosition      ?positionProp .              [1]
      ?positionProp seas:evaluation           ?eval . .                    [2]
      ?eval          plt:temporalContext       ?context . .                [3]
      ?eval         seas:evaluatedSimpleValue ?avgpositionValue. .         [4]
      ?context      time:inXSDDateTime        ?time . .                    [5]

       FILTER (?time > xsd:dateTime("2016-10-14T02:50:00Z" &&
            ?time < xsd:dateTime("2016-10-15T02:50:00Z")
}
```

- SPARQL query support – SELECT caluses
  - BGP – Basic Graph Patterns
    - Set of Triple Patterns (from [1] to [5] above)
    - Zero or more FILTER clauses
  - Optional Patterns
    - Only BGPs and Other OPTIONAL clauses
  - Distinct
  - !No Aggregation, grouping clauses

PLATOON

Result format: SPARQL JSON

```
{ "head": {
   "vars": []
   },
    "results": {
   "bindings": []
   }
}
```

PLATOON

Getting Started: FQP for Pilot1A and Pilot 3A

- Two ways:
  - As a python library
  - As SPARQL endpoint service
- Configuration:
  - There should be at least one Federation with One Data Source

FQP as python lib

- Install
  - python3 setup.py install
- Configuration
  1. As python Object

```
from awudima import Federation, DataSource, DataSourceType
# create a federation and data source
fed = Federation(fedId="Pilot1AFed",
                 name="PLATOON Pilot 1A Data Federation"
          desc="description here …")
ds1 = DataSource(dsId="pilot1a_engie",
         name = "Pilot1A from ENGIE",
         url = "192.168.0.11:27017",
         dstype = DataSourceType.MONGODB_LD_FLAT,
         params = {'username': 'root',
                        'password': 'pass',
                  '<http://..../D2RQ/0.1#jdbcDSN>': 'pilot1a_engie'
              })
# add data source to the federation
fed.addSource(ds1)
pprint(fed.to_json())
```

  2. As json config file (next slide)

PLATOON

2. As json config file

```
{
”fedId”: ”Pilot1AFed”,
“name”: “PLATOON Pilot 1A Data Federation”,
“desc”: “description here …”,
“sources”: {
   ”dsId”: “engie_pilot1a”,
   “name”: “Pilot1A from ENGIE”,
   “desc”: “description here …”,
   “url”: “192.168.0.11:27017”,
   “dstype”: “MONGODB_LD_FLAT”,
   “params”: {
      “username”: “root”,
      ”password”: “1234”,
      “<http://.../jdbcDSN>”: “pilot1a_engie”
    }
  }
}
```

PLATOON

2. As json config file

```
from awudima import Federation, DataSource, DataSourceType


fed = Federation.config('config.json')
pprint(fed.to_json())
```

```
{

"fedId": "Pilot1AFed",
"name": "PLATOON Pilot 1A Data Federation",
"desc": "description here …",
"sources": {
   "dsId": "engie_pilot1a",
   "name": "Pilot1A from ENGIE",
   "desc": "description here …",
   "url": "192.168.0.11:27017",
   "dstype": "MONGODB_LD_FLAT",
   "params": {
      "username": "root",
      "password": "1234",
      "<http://.../jdbcDSN>": "pilot1a_engie"

  }

}}
```

config.json

PLATOON

Collect Source Description metadata

```
from awudima import Federation, DataSource, DataSourceType

# load config
fed = Federation.config('config.json')
pprint(fed.to_json())

#collect source metadata
fed.collect_molecules()

#dump it to a file
fed.dump_to_json("federation_1a.json")

#more information is now added to the Federation fed object as RDFMTs
pprint(fed.to_json())
```

PLATOON

Running Queries over the Federation

```
# import FQP driver
from awudima import AwudimaFQP, Federation

# load config
fed = Federation.config('config.json')

# create the FQP object using the federation object created above
fqp = AwudimaFQP(fed)

# SPARQL query to get a list of WindFarms in the federation of data sources
query = "SELECT * WHERE {?windfarm a <https://w3id.org/platoon/WindFarm>}"

# Execute SPARQL queries
resultset = fqp.execute(query)

if resultset:
    # print results as SPARQL JSON Result format (json object)
    pprint(resultset.results)

    # show query plan
    pprint(resultset.plan)
```

PLATOON

FQP as SPARQL Endpoint Service

- Run as Docker service
  - `docker build -t awudima-fqp:0.3 .`
  - `docker run -d --name fqp -p 8000:8000 -e CONFIG_FILE=/data/federation.json awudima-fqp:0.3`
- Configure
  - `curl --location -g --request POST ‘localhost:8000/configure?federation={jsonencoded-federation-config}’`
- Inspect Federation metadata
  - `curl --location -g --request GET ‘localhost:8000/inspect’`

FQP as SPARQL Endpoint Service

- Run query
  - `curl --location -g --request GET 'localhost:8000/sparql?query=SELECT%20*%20WHERE{….}'`

PLATOON

DEMO

PLATOON

PLATOON