Linked Data Engeneering Lab Winter 11/12

Linked Data Sensor Platform

Project Report

C. Autermann, D. Demuth, C. Kiehl, N. Winkler

January 30, 2012

1 Motivation

With the open-hardware movement the "Internet of Things"(IoT) is invading the sensor-market. Processing platforms like Arduino[1](and its clones) or nanode[2] are cheap and can be programmed to fit your own needs with basically no knowledge of microcontroller-programming. This hardware is capable of providing open standards like TCP/IP or HTTP and therefore is able to serve as a tiny-webserver. Those webservers are able to provide RESTful services as well as they can deliver all sorts of mime[3]-types and can even serve as fileserver. Limited in RAM, ROM and bandwidth they are not the best option to serve a metadata-refined xml document, but they are capable of describing the basic attributes of an attached sensor in a more lightweight text-based format known as "json".

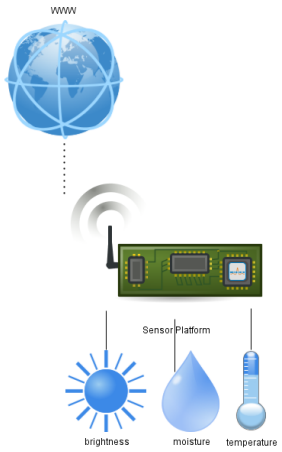

Figure 1: Sensor Platform

Demuth has build a prototypic sensor platform which is integrated into the "Web of Things" and providing its measured data with the aid of a RESTful service in a json format of O&M[4] and delivering a possibility to identify the accuracy of each sensor due to sensordescription-documents linked to the sensor data. It can be assumed that more devices will have the ability of self-annotation in the future. To deal with this, we

[1] http://www.arduino.cc
[2] http://www.nanode.eu
[3] Multipart Internet Mail Extensions
[4] Observations & Measurements

will present an alternative method to harvest the measured data. Contrary to services like pachube[5], the data is neither pushed to the service nor pulled from the sensor by the service in static periods: the data is only pulled from the sensor when new data is available.

The aim of this project is to develop a web service which is querying sensor platforms to generate "Linked-Data". A sensor platform has the capability of announcing itself to the service, but it can also be added manually to the service. After registering an announcement, the service is querying the sensor platform to discover its capabilities. The sensor platform will provide a document describing the URI of each sensor's data, as well as a URI pointing on the corresponding description of each sensor in a machine-readable format. After analyzing this input the service knows which sensors are attached to the sensor platform. The service will store this data in a triplestore. The service is able to query each sensor attached to the sensor platform by requesting the corresponding data url. This request will be responded with a json file containing the measured value, the unit of the measurement, the position of the sensor platform and corresponding timestamps. The service has to be able to query multiple sensors and sensor platforms at the same time, to generate a "linked data" view on the generated data.

2 Requirements

To fit the needs of such a use case, a web service has to be easy to use. The registration-process for a new sensor platform has to be as simple as possible. Every user should be able to add new sensor platforms to the service. This delivers the first requirement: The service must be easy to use. Sampling intervalls of sensors can differ. When a sampling intervall has passed a new measurement is available. As the sensor platform is not storing the measured data, the web service must be capable of querying the data within the sampling intervall, otherwise the data will be lost. This will result in a lot of synchronous connections which have to be handled by the service. The second and third requirement are: The server has to act fast, and must be capable of providing multiple threads.

After retrieving the data from the different sensor platforms, the web service should be able to store the data in a triplestore. After storing, the data can be accessed via a SPARQL-endpoint. The fourth requirement: Persistence of Data

3 Architecture

The web service is divided into three parts. The first part connects to sensor-platforms and tries to harvest their data, the second part converts detected SensorML and O&M documents to RDF, the third part will store the generated RDF in a triplestore. A figure describing the service can be found as figure 3 on page 4. To register a new sensor

[5] http://www.pachube.com

platform at the web service, the URI of the sensor platform is posted to the service. This can be done by the sensor platform startup routines or manually. After the web service has recieved the URI of the platform, it tries to connect to this URI. If successfull, the platform will deliver a json-file containing an array of sensor-objects. These sensor-objects describe the sensors connected to the platform. Each sensor-object contains a URI pointing to a sensor-description document named "sensor-description" and a URI pointing to the current observation of the sensor named "currentObservation". The web service will follow this URIs. Both URI's can point to a location on this sensor platform, or to an external source. Due to limitations in bandwidth of the sensor platform the sensor-description will mostly be stored on an external source. The web service will receive a SensorML document when following the "sensor-description" and a O&M document when following the "currentObservation" URI. The process of registering a new sensor platform can be seen in figure 3. The documents sent by the sensor platform will be processed in the second part of the web services routines, where both SensorML and O&M will be converted to RDF. Finally the generated RDF will be stored in a triplestore. To test this setup, we have created a sensor platform emulator [6]. The data can be obtained by a SPARQL-endpoint

Once a sensor platform is registered, the web service is able to recognize the sampling-intervall of each sensor. Everytime a new measurement is made, the web service will connect to the sensor platform's sensor-data URI to get the latest measurement. The new measurement will be transformed to RDF and stored in the triplestore.

4 Implementation

The service is implemented as an Java Webapp running on an Apache Tomcat Application Server and uses a Sesame triplestore to save the generated Linked Data. The triplestore is accessed by its HTTP interface and could also be deployed on an external source.

The sensor platform's sensors provide their Observations as O&M 2.0 Measurements. It is encoded in a JSON format as described by the UncertWeb project [7]. To parse the observations the service uses the UncertWeb O&M 2.0 Java API [8].

A sensor platform is registerd at the service by simply posting the URL of the sensor platform to the path "/senseboxes" (see Listing 1). The service will then explore the sensor platform and discover the attached sensors. The SensorML description of the sensor will be requested, parsed, converted to RDF and saved to the triplestore.

To convert both O&M and SensorML to RDF four different ontologies were used, out of which one was created for this purpose.

To convert the SensorML data, the ontology "SensorOntology.owl" http://www.w3.org/2005/Incubator/ssn/wiki/images/4/42/SensorOntology20090320.owl.xml was

[6] http://giv-uw.uni-muenster.de:8080/sensebox/
[7] https://wiki.aston.ac.uk/foswiki/bin/view/UncertWeb/OMJson
[8] https://wiki.aston.ac.uk/foswiki/bin/view/UncertWeb/OmJavaAPI

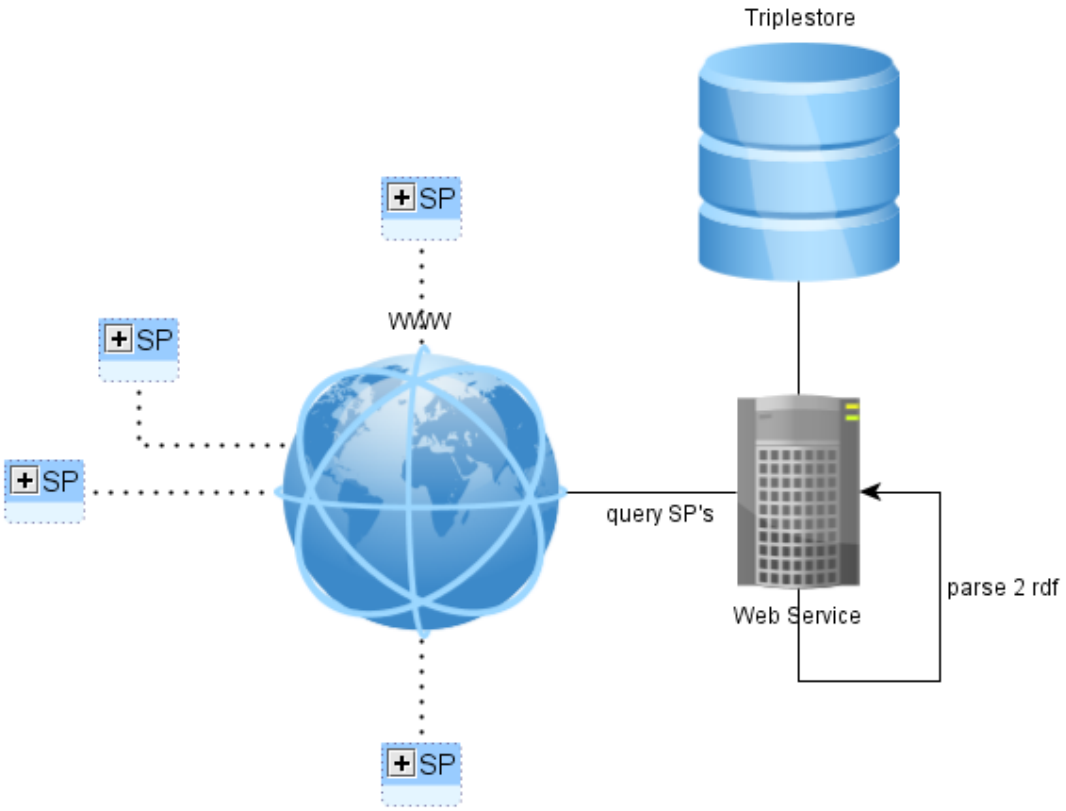

Figure 2: Service Architecture

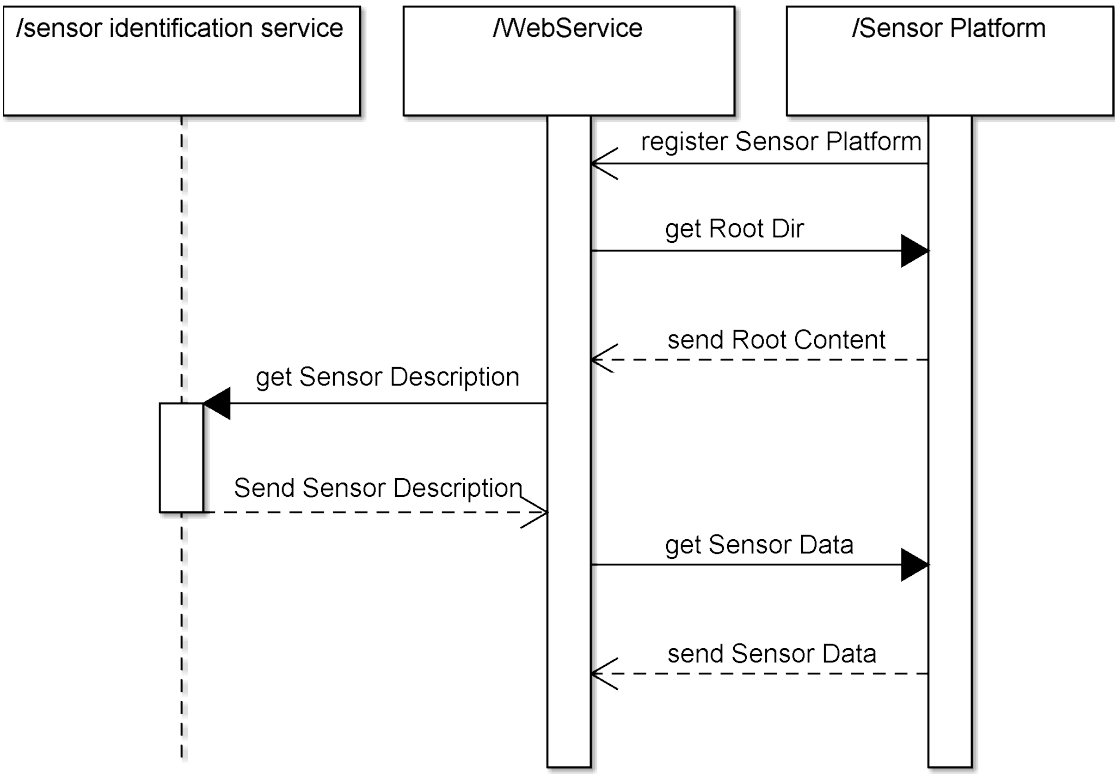

Figure 3: Sequence of registering a new sensor platform

used. From this ontology the resources “Process", “Output" and “Input" were implemented. In addition the properties “hasIdentification", “hasOutput" and “hasInput" were used. In order to convert the SensorML data, the timestamp of the data needed to be processed. Unfortunately the SensorOntology is not offering such resources or properties, so the requiered properties “hasRefreshTime" and “hasCurrentObservationUrl" were created. The properties can be only found at a local namespace.

For converting the O&M data, the ontology “OuM.owl" http://www.personal.psu.edu/kuj13/OuM/OuM.owl# were used. The resources “Observation", “ObservedProperty", “ObservationResultValue" and “Procedure" and the property “resultTime" were required in this case. In order to describe a location where a certain measurement was taken, the ontology “wgs84_pos" http://www.w3.org/2003/01/geo/wgs84_pos# was used. The class “SpatialThing" and the properties “lat" and “long" were needed in this case.

Listing 1: Request to register a sensor platform

```
POST /ldsp/sensors HTTP/1.1
Host: giv-uw.uni-muenster.de:8080
Accept: */*
Content-Type: text/plain
Content-Length: 43

http://giv-uw.uni-muenster.de:8080/SenseBox
```

The sampling rate of each sensor is determined by the first observation request. The sensor sets the HTTP-headers “Last-Modified” and “Expires” and the service will request observations only when a new observation is generated (see Listing 2 and 3).

Listing 2: Observation-Request

```
GET /sensor platform/sensors/PM10/observation HTTP/1.1
Host: giv-uw.uni-muenster.de:8080
Accept: application/json
```

Listing 3: Observation-Response

```
HTTP/1.1 200 OK
Last-Modified: Sun, 29 Jan 2012 17:28:16 GMT
Expires: Sun, 29 Jan 2012 17:28:21 GMT
Access-Control-Max-Age: 3628800
Access-Control-Allow-Headers: Content-Type, Origin
Access-Control-Allow-Methods: GET, OPTIONS
Access-Control-Allow-Origin: *
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 29 Jan 2012 17:28:16 GMT

{"OM_Measurement": {
    "identifier": {
```

```
        "codeSpace": "http:\/\/giv-uw.uni-muenster.de:8080\/sensor
           platform\/",
        "value": "o_1284"
    },
    "phenomenonTime": {"TimeInstant": {"timePosition": "2012-01-29T18
       :28:16.764+01:00"}},
    "resultTime": {"TimeInstant": {"timePosition": "2012-01-29T18
       :28:16.764+01:00"}},
    "observedProperty": "http:\/\/giv-genesis.uni-muenster.de:8080\/
       SOR\/REST\/phenomenon\/OGC\/Concentration\/PM10",
    "procedure": "http:\/\/giv-uw.uni-muenster.de:8080\/sensor
       platform\/sensors\/PM10",
    "featureOfInterest": {"SF_SpatialSamplingFeature": {
        "type": "http:\/\/www.opengis.net\/def\/samplingFeatureType\/
           OGC-OM\/2.0\/SF_SamplingPoint",
        "sampledFeature": "urn:ogc:def:nil:OGC:unknown",
        "shape": {
            "type": "Point",
            "coordinates": [
                7.611039595771213,
                52.12930561374704
            ],
            "crs": {
                "type": "name",
                "properties": {"name": "http:\/\/www.opengis.net\/def
                   \/crs\/EPSG\/0\/4326"}
            }
        }
    }},
    "result": {
        "uom": "ug\/m^3",
        "value": 0.6246824312417747
    }
}}
```


As the service does not use an internal database, all informations regarding the sensors are saved as RDF in the triplestore. This includes the determined sample rate and the URL's to request the SensorML file and the latest observation and enables the service to request observations of the sensors after a restart of the service. The relevant sensor information is gained by the service by the SPARQL query seen in listing 4.

Listing 4: Observation-Response

```
prefix ldsp:<http://localhost:8080/onto/ldsp#>
prefix rdfs:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix sml:<http://www.csiro.au/Ontologies/2009/SensorOntology.owl#>

SELECT DISTINCT ?desc ?time ?curr
WHERE {
    ?desc ldsp:hasRefreshTime ?time.
```

```
    ?desc ldsp:hasCurrentObservationUrl ?curr.
    ?desc rdfs:type sml:Process.
}
```

4.1 Accessing observations

The access to observations is provided by the SPARQL interface of the Sesame triple-store. It allows extensive quering of the saved values (see Listing 6, 7, 8 and 5).

Listing 5: Query for all ozone values and their spatio-temporal coordinates in a bounding box at a specific day.

```
PREFIX sml:         <http://www.csiro.au/Ontologies/2009/
    SensorOntology.owl#>
PREFIX rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX om:         <http://www.semanticweb.org/ontologies/2010/3/
    Ontology1270658429740.owl#>
PREFIX sor:         <http://giv-genesis.uni-muenster.de:8080/SOR/REST/
    phenomenon/OGC/Concentration/>
PREFIX ldsp:<http://localhost:8080/onto/ldsp#>
PREFIX pos:         <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd:         <http://www.w3.org/2001/XMLSchema#>

SELECT ?lon ?lat ?time ?value WHERE {
        ?o om:ObservedProperty sor:O3 .
        ?o om:ObservationResultValue ?value .

        ?o om:resultTime ?time .
        FILTER ( ?time >= "2012-01-23T00:00:00+01:00"^^xsd:dateTime)
        FILTER ( ?time < "2012-01-24T00:00:00+01:00"^^xsd:dateTime)

        ?o pos:lat ?lat .
        FILTER ( ?lon >= 47.2708 && ?lon <= 55.0591 )
        ?o pos:lon ?lon .
        FILTER ( ?lat >= 5.8669 && ?lat <= 15.0436 )
}
```

Listing 6: Query all Sensors.

```
PREFIX sml:         <http://www.csiro.au/Ontologies/2009/
    SensorOntology.owl#>
PREFIX rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?s WHERE {
        ?s rdf:type sml:Process.
}
```

Listing 7: Query for all Sensor measuring ozone concentration.

```
PREFIX sml:         <http://www.csiro.au/Ontologies/2009/
    SensorOntology.owl#>
```

```
PREFIX rdf:          <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sor:          <http://giv-genesis.uni-muenster.de:8080/SOR/REST/
    phenomenon/OGC/Concentration/>

SELECT ?s WHERE {
        ?s rdf:type sml:Process.
        ?s sml:hasOutput sor:O3.
}
```

Listing 8: Query for all Observation by the Sensor http://giv-uw.uni-muenster.de:8080/SenseBox/sensors/O3.

```
PREFIX om:          <http://www.semanticweb.org/ontologies/2010/3/
    Ontology1270658429740.owl#>

SELECT ?o WHERE {
        ?o om:Procedure <http://giv-uw.uni-muenster.de:8080/SenseBox/
            sensors/O3>.
}
```

5 Conclusion

In this project report we presented a solution to harvest sensor platforms and store the harvested data as linked data. The proposed solution is is independend of directory structures as it only needs the URI of a sensor platform to start harvesting the platforms data. Differing directory structures on other platforms do not influence this service in its ability to harvest the data, as the service is always following the given URI and then proccesing the resulting data. By using the presented webservice we have shown, that querying sensor platforms and generating linked data from the gathered data is possible in a fast and flexible manner.

Future Work: Up to now the service is only capable of parsing SensorML documents. Unfortunately this documents need a unique key to identify a certain sensor, meaning that every sensor platform would need its own set of sensor descriptions resulting in the impossibility of sharing sensor descriptions amongst sensor platforms. This might change with the sensor description language StarFL which allows to share sensor-characteristics between sensor platforms. It would be reasonable to make the service independent from one specific sensor-description format.