

# Designing Semantic Web of Things Applications with M3

## M3 API Documentation and user interface for developers

Creator	Amelie Gyrard (Eurecom)
Send Feedback	Do not hesitate to ask for help or give us feedback, advices to improve our tools or documentations, fix bugs and make them more user-friendly and convenient: <a href="mailto:amelie.gyrard@insight-centre.org">amelie.gyrard@insight-centre.org</a>
Goal	<ul style="list-style-type: none"><li>• This documentation guides developers to design Semantic Web of Things applications thanks to the Machine-to-Machine Measurement (M3) framework through APIs that we develop or user interfaces.</li><li>• Get Semantic Web of Things templates with web services or the user interface</li><li>• Use the M3 converter to semantically annotate IoT data</li><li>• Interpret IoT data and get suggestions or high level abstractions</li><li>• A tutorial is also included to design the application with the M3 and Jena framework.</li><li>• Use LOV4IoT web services</li><li>• Use M3 nomenclature and M3 ontology web services</li></ul>
Requirement	Use the Jena Framework.
Created	April 22, 2015
Last updated	February 25, 2016
Status	Work in progress
URL	<a href="http://www.sensormeasurement.appspot.com/documentation/M3APIDocumentation.pdf">http://www.sensormeasurement.appspot.com/documentation/M3APIDocumentation.pdf</a>

This documentation guides developers to design Semantic Web of Things applications thanks to the Machine-to-Machine Measurement (M3) framework.

It will assist them in:

- **Generating IoT templates.** The developers do not need to design any ontologies, datasets and rules, they are provided in the M3 templates.
- **Semantically annotating IoT data.** The developers not need to semantically annotate his IoT data. It will be automatically done by the M3 converter.
- **Interpreting IoT data.** The developers are assisted by the M3 framework to interpret IoT data. They will get high-level abstractions or even M3 suggestions according to the M3 template chosen.

In this documentation, the developers can either use web service or user interface.

## Table of Contents

I. Tutorial: Building the naturopathy application with the user interface SWoT generator and the Jena framework .....	3
1. Generating the naturopathy template with the SWoT generator.....	3
2. Explore the naturopathy template .....	4
3. Get the sensor dataset already converted with M3 .....	5
4. Be familiar with the Jena framework.....	6
5. Load the sensor dataset in your Java application with the Jena framework.....	6
6. Load the ontologies and datasets in your Java application with the Jena framework....	6
1. Load the rules and execute the Jena reasoner .....	7
2. Modify the SPARQL query .....	7
3. Execute the SPARQL query with Jena .....	8
4. Check that the naturopathy application works .....	9
II. Generating IoT templates with M3 user interface or web services .....	10
1. M3 User interface.....	10
2. M3 Web Service: looking for IoT application template .....	11
3. M3 Web Service: generating IoT application template .....	12
4. M3 Web Service: generating the SPARQL query with variables replaced .....	13
5. Code example.....	15
III. Semantically annotating IoT data with the M3 converter .....	16
1. M3 converter user interface .....	16
2. Code example to semantically annotate IoT data with M3.....	17
3. Enrich the M3 converter and adapt it to your data .....	17
IV. Interpreting IoT data and getting M3 suggestions .....	19
1. Loading M3 domain knowledge.....	19
2. Executing rules .....	20
3. Executing SPARQL query .....	20
4. Finishing the application .....	20
5. Code summary .....	21

V.	Summary: Developing Semantic Web of Things applications .....	21
VI.	Query the M3 nomenclature/ontology .....	22
1.	Web service: querying sensors .....	22
2.	Web service: querying actuators .....	23
3.	Web service: querying domains .....	23
4.	Web service: querying health devices .....	23
5.	Web service: querying transport devices.....	24
6.	Web service: querying home devices .....	24
VII.	LOV4IoT web services .....	24
1.	Web service: Get the total number of ontologies .....	24
2.	Web service: Get the number od ontologies by domains .....	25
3.	Web service: Get the number of ontology by ontology status.....	26
4.	Use case.....	27
VIII.	Citations .....	28

# I. Tutorial: Building the naturopathy application with the user interface SWoT generator and the Jena framework

## 1. Generating the naturopathy template with the SWoT generator

- Go on this web page:

<http://www.sensormeasurement.appspot.com/?p=m3api>

- Choose the sensor 'Thermometer' in the drop-down list.
- Choose the domain 'Healthcare' in the drop-down list.

- Choose the template 'Body Temperature, Symptoms and Home Remedies' in the drop-down list. In this case, we suggest only one template.
- Click on the button 'Generate ZIP file.'

## Semantic Web of Things (SWoT) Generator

The SWoT generator enables designing SWoT applications to interpret IoT data.

### STEP 1: Search M3 Template

1. Choose a sensor (e.g., Light/Illuminance Sensor)
2. Choose the domain where is deployed your sensor (e.g., Weather)
3.

### STEP 2: Choose M3 Template

- Choose an application template:

### STEP 3: Download M3 template

- 

Figure 1. Download the naturopathy template using the SWoT generator

## 2. Explore the naturopathy template

Open the naturopathy template that you just downloaded. This template is composed of the following files:

- **ruleM3Converter.txt**: a set of rules used to convert sensor data according to our M3 language implemented in the M3 ontology. For instance, we use the term temperature and not term. An essential basis for the reasoning.
- **naturopathy.owl**: the naturopathy ontology
- **naturopathy-dataset.rdf**: the naturopathy dataset
- **m3SparqlGeneric.sparql**: the SPARQL query to get smarter data or even suggestions. For instance, get home remedies when you have the fever.
- **m3.owl**: the M3 ontology essential to describe sensor data in an interoperable manner to ease the reasoning and the interlinking of domains.
- **LinkedOpenRulesHealth.txt**: This file is a dataset of interoperable rules to interpret health measurements. For instance: IF BodyTemperature > 38°C THEN **HighFever**.
- **health.owl**: the health ontology. For instance, **Symptom** is a concept defined in this ontology.
- **health-dataset.rdf**: the health dataset. For instance, **HighFever** is an instance of the **Symptom** concept in this dataset.

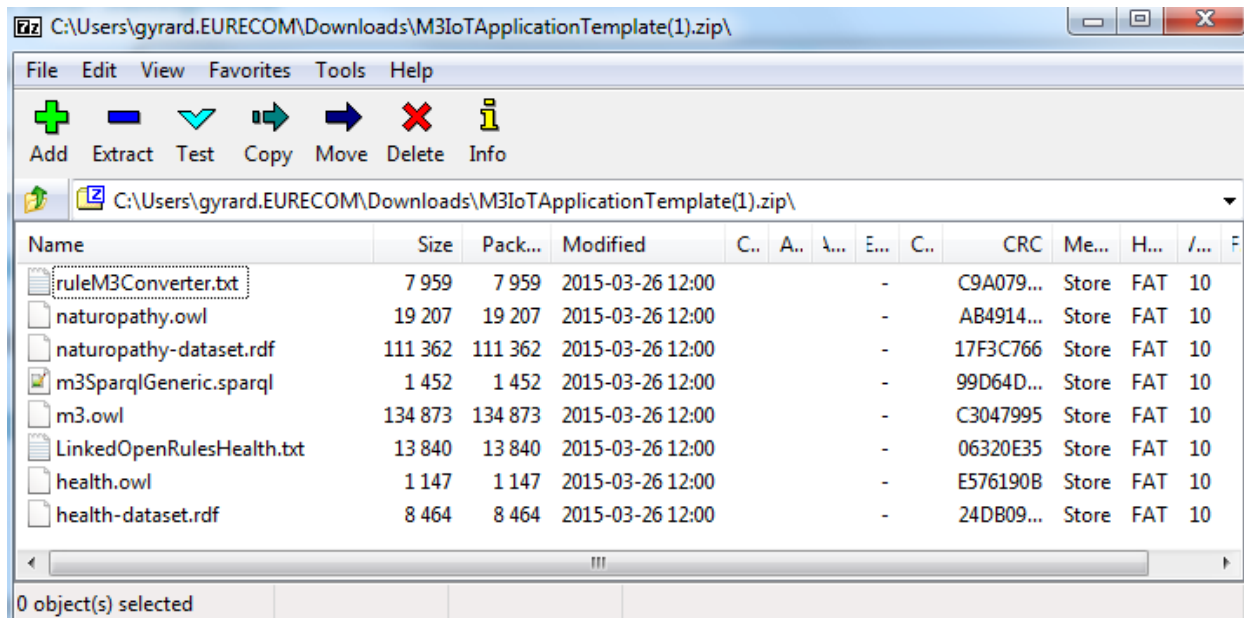


Figure 2. The naturopathy template

### 3. Get the sensor dataset already converted with M3

- Download the sensor dataset:  
[http://www.sensormeasurement.appspot.com/dataset/sensor\\_data/senml\\_m3\\_health\\_data.rdf](http://www.sensormeasurement.appspot.com/dataset/sensor_data/senml_m3_health_data.rdf)

To begin with, try with the sensor dataset that we have already converted according to the M3 ontology. In the extract below, you have the measurement 'temperature 38°C', a new type has been added 'BodyTemperature' which will be used in the reasoning process to infer high-level abstractions.

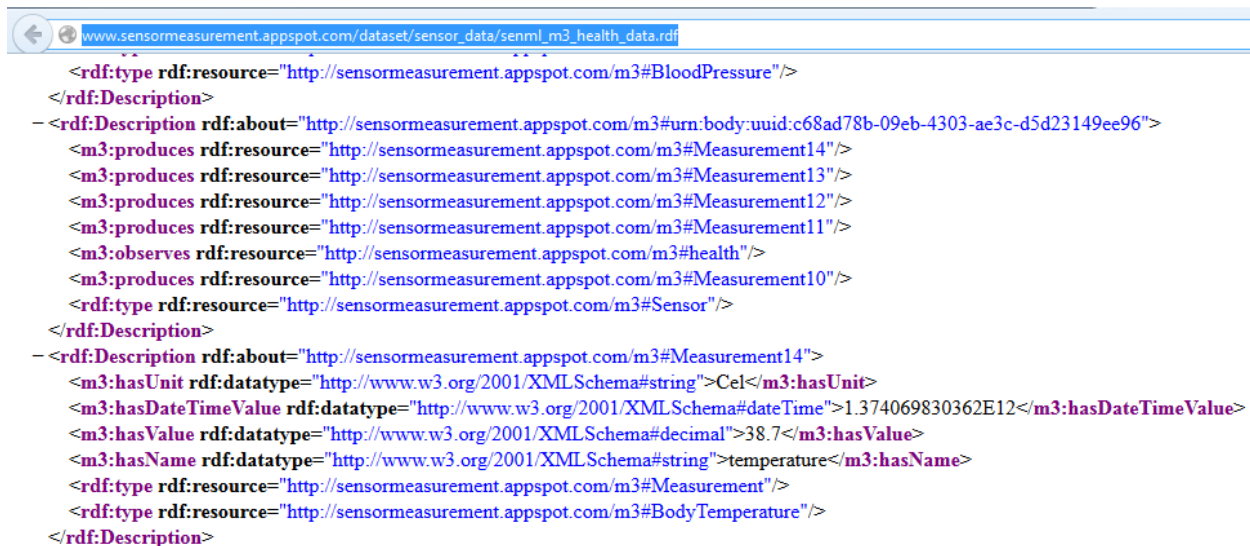


Figure 3. Extract of the sensor dataset

## 4. Be familiar with the Jena framework

Jena tutorial if you are not familiar with this framework: <https://jena.apache.org/>

## 5. Load the sensor dataset in your Java application with the Jena framework

Java code example:

```
1 public static final String HEALTH_M3_SENSOR_DATA_WAR =
2   "./dataset/sensor_data/senml_m3_health_data.rdf";
3
4 Model model = ModelFactory.createDefaultModel();
5 ReadFile.enrichJenaModelOntologyDataset(model, HEALTH_M3_SENSOR_DATA_WAR);
6
7 //check that the model is not empty, that the sensor data is loaded
8 Model.write(System.out);
9
```

**Figure 4. Load the Sensor dataset with Jena**

### ReadFile Java Class:

Java code example:

```
1  /**
2   * Read ontologies or RDF dataset,
3   * included directly from the file (a path) and add it to the jena model
4   * @param model
5   * @param file
6   */
7  public static void enrichJenaModelOntologyDataset(Model model, String file){
8      try {
9          InputStream in = new FileInputStream(file);
10         model.read( in, file );//file:"+"
11         in.close();
12     } catch (IOException e) {
13         // TODO Auto-generated catch block
14         e.printStackTrace();
15     }
16 }
```

**Figure 5. Load a file (ontology or RDF dataset) in the Jena model**

## 6. Load the ontologies and datasets in your Java application with the Jena framework

// load **m3.owl**

```

ReadFile.enrichJenaModelOntologyDataset(model, ROOT_OWL_WAR + "m3");

// load naturopathy.owl
ReadFile.enrichJenaModelOntologyDataset(model, NATUROPATHY_ONTOLOGY_PATH);

// load naturopathy-dataset.rdf
ReadFile.enrichJenaModelOntologyDataset(model, NATUROPATHY_DATASET_PATH);

// load health.owl
ReadFile.enrichJenaModelOntologyDataset(model, HEALTH_ONTOLOGY_PATH);

// load health-dataset.rdf
ReadFile.enrichJenaModelOntologyDataset(model, HEALTH_DATASET_PATH);

```

## 1. Load the rules and execute the Jena reasoner

```

// load LinkedOpenRulesHealth.txt

1 //reasoner for jena rules
2 // the reasoner will infer new triples
3 // and high level abstraction from sensor data
4
5 // read rules
6 Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL(PATH + LinkedOpenRulesHealth.txt));
7
8 //for android use Rule.parseRule
9 reasoner.setDerivationLogging(true);
10
11 //apply the reasoner
12 InfModel inf = ModelFactory.createInfModel(reasoner, model);
13 return inf;
14 }

```

**Figure 6.**Load rules and execute the Jena reasoner

## 2. Modify the SPARQL query

Java code example to modify the SPARQL query with variables:

```

1 //variable in the sparql query
2 ArrayList<VariableSparql> var = new ArrayList<VariableSparql>();
3 var.add(new VariableSparql("inferTypeUri", Var.NS_M3 + "BodyTemperature", false));
4 // we look for BodyTemperature Measurements

```

**Figure 7.**Modify variables in the SPARQL query

In this example, we are looking for BodyTemperature measurements in the dataset.

VariableSparql Java Class:

```

1  /**
2  * To change the values of some variables in sparql queries
3  */
4  public class VariableSparql {
5
6  private String variableName;
7  private String value;
8  private boolean isLiteral;
9  public String getVariableName() { return variableName; }
10 public VariableSparql(String variableName, String value, boolean isLiteral) {
11     super();
12     this.variableName = variableName;
13     this.value = value;
14     this.isLiteral = isLiteral;
15 }
16 public void setVariableName(String variableName) { this.variableName = variableName; }
17 public String getValue() { return value; }
18 public void setValue(String value) { this.value = value; }
19 public boolean isLiteral() { return isLiteral; }
20 public void setLiteral(boolean isLiteral) { this.isLiteral = isLiteral; }
21 }

```

**Figure 8. The VariableSparql Java Class example**

### 3. Execute the SPARQL query with Jena

**// load m3SparqlGeneric.sparql**

```

1  ExecuteSparqlGeneric reqSenml = new ExecuteSparqlGeneric(inf, sparqlQuery);
2  String resultSparqlsenml = reqSenml.getSelectResultAsXML(var);
3  // you should get high level abstractions in XML.
4

```

**Figure 9. Execute the SPARQL query example**

#### ExecuteSparqlGeneric Java class

```

1  public String getSelectResultAsXML(ArrayList<VariableSparql> var){
2      QueryExecution qe = replaceVariableInRequest(this.model, this.query, var);
3      //get result from sparql request
4      ResultSet results = qe.execSelect() ;
5      String res = "No results";
6      res = ResultSetFormatter.asXMLString(results);
7
8      qe.close();
9      return res;
10 }
11

```

**Figure 10. Get the result of the SPARQL query, more precisely the high level abstractions**



## ExecuteSparqlGeneric Java class

```
1 public Model model;
2 public Query query;
3
4 public ExecuteSparqlGeneric(Model model, String sparqlRequest) {
5     super();
6     this.model = model;
7
8     //load the sparql query
9     this.query = QueryFactory.create(ReadFile.readContentFile(sparqlRequest));
10 }
11
12 public static QueryExecution replaceVariableInRequest(Model model, Query query, ArrayList<VariableSparql> var){
13     QueryExecution qe = null;
14     RDFNode node = null;
15     QuerySolutionMap initialBinding = new QuerySolutionMap();
16     //replace sparql request by variables
17     if(var!=null){
18         for (VariableSparql variableSparql : var) {
19             if (variableSparql.isLiteral()){
20                 node = model.createLiteral(variableSparql.getValue());
21             }
22             else{
23                 node = model.getResource(variableSparql.getValue());
24                 //System.out.println("node: " + node);
25             }
26             initialBinding.add(variableSparql.getVariableName(), node);
27         }
28         qe = QueryExecutionFactory.create(query, model, initialBinding);
29     }
30     else{
31         qe = QueryExecutionFactory.create(query, model);
32     }
33     return qe;
34 }
```

**Figure 11. ExecuteSparqlGeneric Java class example**

## 4. Check that the naturopathy application works

You should have the results in xml, if it not empty it works!

Congratulations!

You can then design your own applications, and display the result in a user interface.

## Suggesting home remedies according to body temperature

1. This scenario is based on these [M3 RDF health data](#)
  2. M2M Aggregation Gateway (Convert Health Measurements into Semantic Data):
  3. We deduce that the temperature corresponds to the body temperature.
  4. We deduce that the person is sick.
  5. We propose all fruits/vegetables according to this disease.
  6. M2M Application: Temperature => Cold => Food: (Wait 10 seconds!)
- Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Pepper mint
  - Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Thyme
  - Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Cinnamon
  - Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Honey
  - Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Ginger
  - Name=temperature, Value = 38.7, Unit=Cel, InferType = Body Temperature, Deduce = HighFever, Suggest= Lemon

Figure 12. Suggestions provided by the SPARQL query from the template

## II. Generating IoT templates with M3 user interface or web services

### 1. M3 User interface

You can use the user interface: <http://www.sensormeasurement.appspot.com/?p=m3api>

See user guide: [www.sensormeasurement.appspot.com/documentation/UserGuide.pdf](http://www.sensormeasurement.appspot.com/documentation/UserGuide.pdf)

## STEP 1: Search IoT Application Template

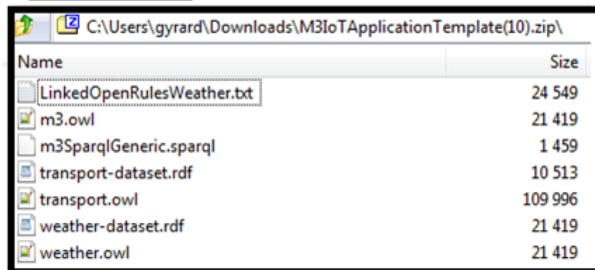
1. Choose a sensor (e.g., Light/Illuminance Sensor)  ← **M3 ontology - sensor + domain**
2. Choose the domain where is deployed your sensor (e.g., Weather)  ←
3.

## STEP 2: Choose IoT Application Template

- Choose an application template:

## STEP 3: Download IoT Application Template

- 



Name	Size
LinkedOpenRulesWeather.txt	24 549
m3.owl	21 419
m3SparqlGeneric.sparql	1 459
transport-dataset.rdf	10 513
transport.owl	109 996
weather-dataset.rdf	21 419
weather.owl	21 419

← **iot application template dataset**

← **M3 interoperable domain knowledge (ontologies, rules and datasets)**

Figure 13. Generating M3 templates using M3 user interface



**Be careful, the SPARQL query generated does not have SPARQL variables replaced.**

**Due to technical issues with Google Web Toolkit (cannot write in a file), please use the M3 web service to generate the SPARQL query with variables replaced.**

**If you are familiar with SPARQL, you can replace variables yourself.**

## 2. M3 Web Service: looking for IoT application template

Web service URL:

<http://www.sensormeasurement.appspot.com/m3/searchTemplate/?sensorName=LightSensor&domain=Weather&format=json>

Description: You are looking for IoT application templates with the following parameters:

- sensorName=LightSensor  
The parameter **sensorName** is the name of the sensor.

If you want to indicate another **sensorName** , see:

<http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>  
domain=Weather

The parameter **domain** is where is deployed your sensor.

If you want to indicate another domain, see:

<http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>  
format= json

The parameter **format** can be json or xml

Results:



```
{
  head: {
    vars: [
      "m2mappli",
      "m2mdevice",
      "m2mapplilabel",
      "m2mapplicomment"
    ]
  },
  results: {
    bindings: [
      {
        m2mappli: {
          type: "uri",
          value: http://sensormeasurement.appspot.com/m3#WeatherTransportationSafetyDeviceLight
        },
        m2mdevice: {
          type: "uri",
          value: http://sensormeasurement.appspot.com/m3#LightSensor
        },
        m2mapplilabel: {
          type: "literal",
          "xml:lang": "en",
          value: "Luminosity, Transportation and Safety Device"
        },
        m2mapplicomment: {
          type: "literal",
          "xml:lang": "en",
          value: "IoT application to suggest safety devices according to the luminosity (e.g., sunny -> sun visor)"
        }
      }
    ]
  }
}
```

Figure 14. Looking for the M3 templates

### 3. M3 Web Service: generating IoT application template

Web service URL:

<http://sensormeasurement.appspot.com/m3/generateTemplate/?iotAppli=WeatherTransportationSafetyDeviceLight>

Description: To generate the domain knowledge needed to build the IoT application template:

- iotAppli=WeatherTransportationSafetyDeviceLight

The parameter **ioTappli** is the end of the m2mappli URI that you can find in the result provided by the previous web service

(<http://www.sensormeasurement.appspot.com/m3/searchTemplate/?sensorName=LightSensor&domain=Weather&format=json>)

Results:

<http://sensormeasurement.appspot.com/ont/m3/transport#@http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt@http://sensormeasurement.appspot.com/SPARQL/m3SparqlGeneric.sparql@http://sensormeasurement.appspot.com/dataset/transport-dataset/@http://sensormeasurement.appspot.com/dataset/weather-dataset/@http://sensormeasurement.appspot.com/ont/m3/weather#@http://sensormeasurement.appspot.com/m3#@>

All URI files generated as separated by @.

URI finishing with # are ontologies

URI finishing with / are datasets

URI finishing with .txt are rules

URI finishing with .sparql are SPARQL queries to query data (to ignore because of google app engine we cannot automatically generate/write a new file)

To get the SPARQL query ask the web service:

<http://sensormeasurement.appspot.com/m3/getSparqlQuery/?iotAppli=WeatherTransportationSafetyDeviceLight> (see next section)

## 4. M3 Web Service: generating the SPARQL query with variables replaced

<http://sensormeasurement.appspot.com/m3/getSparqlQuery/?iotAppli=WeatherTransportationSafetyDeviceLight>

Generate the generic sparql query with variables replaced

Results:

```

sensormeasurement.appspot.com/m3/getSparqlQuery/?iotAppli=WeatherTransportationSafetyDeviceLight

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX m3: <http://sensormeasurement.appspot.com/m3#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?name ?value ?unit ?inferType ?deduce ?suggest ?suggest_comment WHERE{
  ?measurement m3:hasName ?name.
  ?measurement m3:hasValue ?value.
  ?measurement m3:hasDateValue ?time.
  ?measurement m3:hasUnit ?unit.

  ?measurement rdf:type <http://sensormeasurement.appspot.com/m3#WeatherLuminosity>.
  OPTIONAL { <http://sensormeasurement.appspot.com/m3#WeatherLuminosity> rdfs:label ?inferType. FILTER (LANGMATCHES (LANG(?inferType), "en")) }

  OPTIONAL {
    ?measurement rdf:type ?deduceUri .
    ?deduceUri rdfs:label ?deduce.
    FILTER (LANGMATCHES (LANG(?deduce), "en"))
    FILTER (str(?deduceUri) != str(m3:Measurement) )
    FILTER (str(?deduceUri) != str(<http://sensormeasurement.appspot.com/m3#WeatherLuminosity> ) )

    OPTIONAL{
      {?deduceUri m3:hasRecommendation ?resUri.} # used by home temp and noise scenario
      UNION{?resUri m3:hasRecommendation ?deduceUri . ?resUri rdf:type <http://sensormeasurement.appspot.com/ont/m3/transport#SafetyDevice>. }

      ?resUri rdfs:label ?suggest.
      FILTER (LANGMATCHES (LANG(?suggest), "en"))
      OPTIONAL{
        ?resUri dc:description ?suggest_comment.
        FILTER (LANGMATCHES (LANG(?suggest_comment), "en"))
      }
    }
  }
}

```

**Figure 15. Generating the M3 SPARQL query**

## 5. Code example

```
1 String URL_M3_API = "http://www.sensormeasurement.appspot.com/m3/";
2
3 // STEP 1: Searching the M3 template fitting your needs
4 String m3_sensor = "LightSensor";
5 // parameter sensorName according to the M3 nomenclature
6 String m3_domain = "Weather";
7 // parameter domain according to the M3 nomenclature
8 String format = "xml"; // or json
9 String search_M3_template = queryWebService(URL_M3_API + "searchTemplate/?" +
10     "sensorName=" + m3_sensor +
11     "&domain=" + m3_domain +
12     "&format="+ format);
13
14 // STEP 2: Choosing the M3 template
15 String m3_iotAppli = parse(search_M3_template);
16 // e.g.: = "WeatherTransportationSafetyDeviceLight";
17
18 // STEP 3: Generating the M3 template
19 String m3_template = queryWebService(URL_M3_API + "generateTemplate/?" +
20     "iotAppli="+ m3_iotAppli);
21 // parameter m3_iotAppli found in STEP 2
22
23 // STEP 4: Getting M3 ontologies, datasets and rules
24 String[] url_file = parse(m3_template);
25 for each url_file
26     String[] url_M3_ontology = download(url_file);
27     String[] url_M3_dataset = download(url_file);
28     String[] url_M3_rule = download(url_file);
29
30 // STEP 5: Getting the SPARQL Query (with variables replaced)
31 String m3_sparql = queryWebService(URL_M3_API + "getSparqlQuery/? +
32     "iotAppli="+ m3_iotAppli);
```

**Figure 16. Generating M3 templates using M3 web services**

### III. Semantically annotating IoT data with the M3 converter

#### 1. M3 converter user interface

The developer can use the M3 converter user interface:

[http://www.sensormeasurement.appspot.com/?p=senml\\_converter](http://www.sensormeasurement.appspot.com/?p=senml_converter)

See user guide: [www.sensormeasurement.appspot.com/documentation/UserGuide.pdf](http://www.sensormeasurement.appspot.com/documentation/UserGuide.pdf)



Use Chrome to get the data in a text format, with Firefox you only have the JavaScript alert popup.

#### SenML to RDF Converter

Copy/paste your SenML/XML sensor data here :

```
-09eb-4303-  
ae3c-d5d23149ee96">  
<e n="blood pressure" t="0"  
u="Pa" v="56">  
<e n="heartbeat" t="0"  
u="beet/m" v="155"></e>  
<e n="temperature"  
t="1374069830362" u="Cel"  
v="40"></e>  
</senml>
```

SenML/XML to RDF Converter

#### M3 interoperable IoT data

```
<rdf:Description rdf:about="http://sensormeasurement.appspot.com/m3#Measurement4">  
<m3:hasUnit rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Cel</m3:hasUnit>  
<m3:hasDateTimeValue rdf:datatype="http://www.w3.org/  
/2001/XMLSchema#dateTime">1.374069830362E12</m3:hasDateTimeValue>  
<m3:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">40.0</m3:hasValue>  
<m3:hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">temperature</m3:hasName>  
<rdf:type rdf:resource="http://sensormeasurement.appspot.com/m3#Measurement"/>  
<rdf:type rdf:resource="http://sensormeasurement.appspot.com/m3#BodyTemperature"/>  
</rdf:Description>  
</rdf:RDF>
```

M3 inferType

Figure 17. Semantically annotating IoT data with the M3 converter user interface



## 2. Code example to semantically annotate IoT data with M3

```
1 // Converting your IoT data using SenML to RDF converter
2
3 // String URL_M3_CONVERTER = "http://www.sensormeasurement.appspot.com/swot/";
4 String format = "xml"; // or json
5 String iot_data = getSenMLData();
6
7 String m3_data = queryWebService(URL_M3_CONVERTER + "convert_senml_to_rdf/?data=" + iot_data +
8                               "&format="+ format);
9
10 store(m3_data);
```

**Figure 18. Semantically annotating IoT data with the M3 converter web service**

## 3. Enrich the M3 converter and adapt it to your data

When you download a template with the SWoT generator<sup>1</sup> you also get the rules to semantically annotate data, the file is called 'ruleM3Converter.txt'.



**We did not have time to implement all the M3 nomenclature. Further, we frequently update the M3 nomenclature<sup>2</sup>.**



**But you can still improve and add more rules to semantically annotate your sensor data.**

---

<sup>1</sup> <http://www.sensormeasurement.appspot.com/?p=m3api>

<sup>2</sup> <http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>

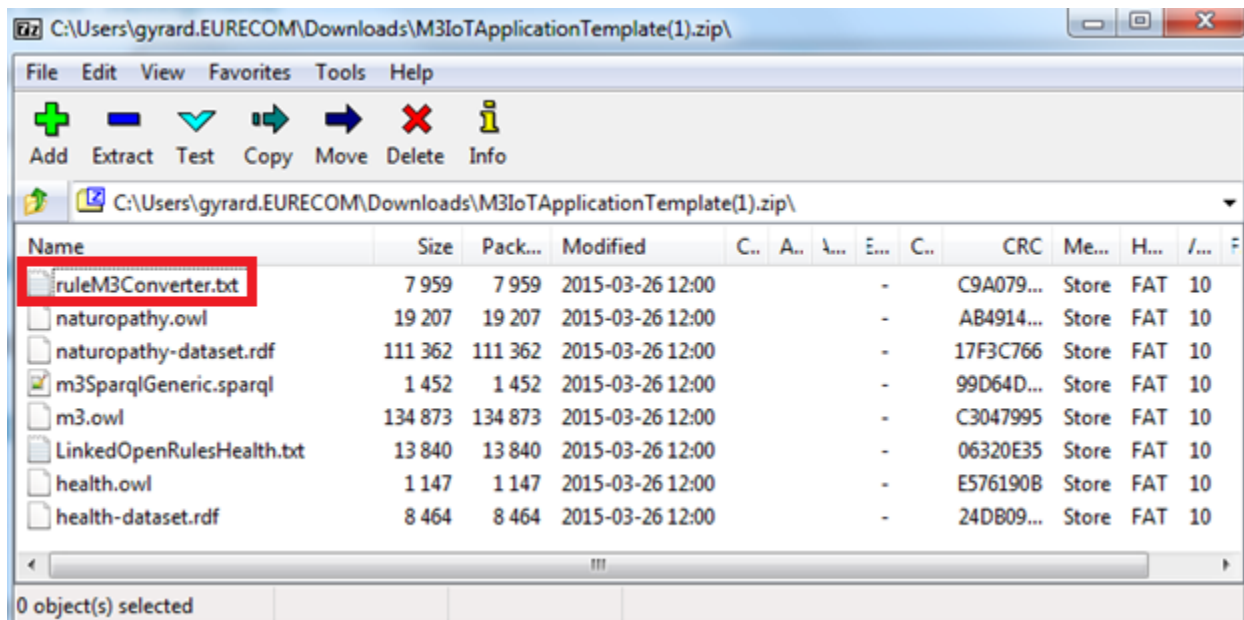


Figure 19. Rules provided in the template to semantically annotate sensor data

The following rule means that we explicitly add the context:

If you get a temperature from health domain (subclassOf m3:FeatureOfInterest), we will explicitly add that it corresponds to a body temperature.

```
ruleM3Converter - Notepad
File Edit Format View Help
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix m3: http://sensormeasurement.appspot.com/m3#
[BodyTemperature: (?measurementUri rdf:type m3:BodyTemperature) => compliant with
    <- the M3 ontology
    (?measurementUri m3:hasName "temperature")
    (?sensor m3:produces ?measurementUri)
    (?sensor m3:observes m3:health)
]
```

=> domain referenced in M3 ontology

Figure 20. Add new rules to semantically annotate sensor data according to the M3 ontology.

This is important because after, you have the rules adapted to this kind of measurement.

```

LinkedOpenRulesHealth - Notepad
File Edit Format View Help
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
@prefix m3: http://sensormeasurement.appspot.com/m3#
@prefix naturopathy: http://sensormeasurement.appspot.com/naturopathy#
@prefix nat: http://sensormeasurement.appspot.com/naturopathy-dataset/
@prefix emotion-dataset: http://sensormeasurement.appspot.com/dataset/emotion-dataset/
@prefix health-dataset: http://sensormeasurement.appspot.com/dataset/health-dataset/
@prefix transport: http://sensormeasurement.appspot.com/ont/m3/transport#

##### LINKED OPEN RULES - HEALTH #####

##### BODY TEMPERATURE RULES #####
#Paper: An ontology based system for social networking for health application support [obaïd et al. 2013]
#IF m3:BodyTemperature GREATER_THAN 39 m3:DegreeCelsius THEN CriticallyHighFever
#tested
# completeness + correctness OK
[CriticallyHighFever:
  (?measurement rdf:type m3:BodyTemperature)
  (?measurement m3:hasValue ?v)
  greaterThan(?v,39.0)
  ->
  (?measurement rdf:type health-dataset:CriticallyHighFever)
]

```

Figure 21. Explicit M3 measurement type is reused in the reasoning process

## IV. Interpreting IoT data and getting M3 suggestions

Several steps need to be achieved to interpret IoT data (see Figure 22):

- Loading M3 ontologies, datasets which have been generated in the M3 template.
- Loading M3 data.= which has been generated by the M3 converter.
- Interpreting IoT data using the Jena reasoned
- Executing the M3 SPARQL query which has been generated in the M3 template
- Parse the result and build the user interface , control actuators or send notification, etc.

### 1. Loading M3 domain knowledge

Jena tutorial:

[http://jena.apache.org/tutorials/rdf\\_api.html](http://jena.apache.org/tutorials/rdf_api.html)

Code example:

```

// STEP 1: Loading M3 domain knowledge and m3_data
Model model = ModelFactory.createDefaultModel();
InputStream in = new FileInputStream(PATH_FILE + m3_data);
// m3_data has been generated with the M3 converter
model.read( in, fileURL );//read all ontologies generated in the M3 template (.owl)
model.read( in, fileURL );//read all datasets generated in the M3 template (.rdf)

```

```
in.close();
```

## 2. Executing rules

Jena tutorial:

<http://jena.apache.org/documentation/inference/>

Code example:

```
// STEP 2: Interpreting M3 data
Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL(PATH_FILE +
LinkedOpenRules*.txt));
// LinkedOpenRules*.txt: rules generated in the M3 template
reasoner.setDerivationLogging(true);
InfModel infModel = ModelFactory.createInfModel(reasoner, model); //apply the reasoner
// infModel has been updated with high-level abstraction
```

## 3. Executing SPARQL query

Jena tutorial:

[http://jena.apache.org/tutorials/rdf\\_api.html](http://jena.apache.org/tutorials/rdf_api.html)

Code example:

```
// STEP 3: Getting M3 suggestions
//      Executing the SPARQL query:
Query query = QueryFactory(m3_sparql); // m3_sparql has been generated in the M3 template
ResultSet results = QueryExecutionFactory.create(m3_sparql, model)
String m3_suggestions = ResultSetFormatter.asXMLString(results)
```

## 4. Finishing the application

The main task of the develop is to design a user-friendly interface or control actuators, etc. according to the high-level abstractions deduce by M3 or the M3 suggestions provided by M3.

Code example:

```
// STEP 4: Parsing and displaying m3_suggestions to build the IoT application
// or control actuators, alerting, etc.
```

## 5. Code summary

```
1 // STEP 1: Loading M3 domain knowledge and m3_data
2 Model model = ModelFactory.createDefaultModel();
3 InputStream in = new FileInputStream(PATH_FILE + m3_data);
4 // m3_data has been generated with the M3 converter
5 model.read( in, fileURL );//read all ontologies generated in the M3 template (.owl)
6 model.read( in, fileURL );//read all datasets generated in the M3 template (.rdf)
7 in.close();
8
9 // STEP 2: Interpreting M3 data
10 Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL(PATH_FILE + LinkedOpenRules*.txt));
11 // LinkedOpenRules*.txt: rules generated in the M3 template
12 reasoner.setDerivationLogging(true);
13 InfModel infModel = ModelFactory.createInfModel(reasoner, model); //apply the reasoner
14 // infModel has been updated with high-level abstraction
15
16 // STEP 3: Getting M3 suggestions
17 // Executing the SPARQL query:
18 Query query = QueryFactory(m3_sparql); // m3_sparql has been generated in the M3 template
19 ResultSet results = QueryExecutionFactory.create(m3_sparql, model)
20 String m3_suggestions = ResultSetFormatter.asXMLString(results)
21
22 // STEP 4: Parsing and displaying m3_suggestions to build the IoT application
23 // or control actuators, alerting, etc.
```

Figure 22. Code example to interpret IoT data and get M3 suggestions

## V. Summary: Developing Semantic Web of Things applications

- ➔ STEP 1: Getting the domain knowledge to interpret sensor data
  - Use the web service (see M3 Web Service: generating IoT application template)
  - Generating ontologies (.owl)
  - Generating datasets (.rdf)
  - Generating rules (LinkedOpenRules\*.txt)
- ➔ STEP 2: Getting the SPARQL query
  - Use the web service (see M3 Web Service: generating the SPARQL query with variables replaced)
  - Generating the SPARQL query: m3\_sparql (.sparql)
- ➔ STEP 3: Converting your data using SenML to RDF converter
  - [http://www.sensormeasurement.appspot.com/?p=senml\\_converter](http://www.sensormeasurement.appspot.com/?p=senml_converter)
  - See documentation:  
[www.sensormeasurement.appspot.com/documentation/UserGuide.pdf](http://www.sensormeasurement.appspot.com/documentation/UserGuide.pdf)
  - Storing the RDF M3 sensor data in a file "m3\_data.rdf"
- ➔ STEP 4: Building the cross-domain IoT application:
  - Tutorial: [An Introduction to RDF and the Jena RDF API](#)
  - Storing RDF sensor data "m3\_data.rdf" file in a Jena Model:  
Model model = ModelFactory.createDefaultModel();  
InputStream in = new FileInputStream(PATH\_FILE + "m3\_data.rdf");  
model.read( in, fileURL );//read all ontologies generated in STEP 1 (.owl)

```
model.read( in, fileURL );//read all datasets generated in STEP 1 (.rdf)
in.close();
```

- Tutorial: [Reasoners and rule engines: Jena inference support](#)
- Download the files with rules:

[Sensor-based Linked Open Rules \(S-LOR\)](#) or generated in see M3 Web Service: generating IoT application template

Syntax: Jena rules, fileName = "LinkedOpenRules\*.txt"

- Reasoning on sensor data:  
Reasoner `reasoner` = new GenericRuleReasoner(Rule.rulesFromURL(PATH\_FILE +  
LinkedOpenRules\*.txt));// read rules  
reasoner.setDerivationLogging(true);  
InfModel infModel = ModelFactory.createInfModel(`reasoner`, `model`); //apply the  
reasoner  
// infModel model updated with sensor data inferred

- Executing the SPARQL query:

```
Query query = QueryFactory(m3_sparql)
```

```
ResultSet results = QueryExecutionFactory.create(m3_sparql, model)
```

```
Return ResultSetFormatter.asXMLString(results)
```

- Parse and display the results to build the IoT application

## VI. Query the M3 nomenclature/ontology

### 1. Web service: querying sensors

Search for all M3 sensors:

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=Sensor&format=json>

Results:

 [www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=Sensor&format=json](http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=Sensor&format=json)

```
{
  "head": {
    "vars": [
      "subject",
      "object",
      "label",
      "comment",
      "imageUrl"
    ]
  },
  "results": {
    "bindings": [
      {
        "subject": {
          "type": "uri",
          "value": "http://sensormeasurement.appspot.com/m3/WindDirectionSensor"
        },
        "object": {
          "type": "uri",
          "value": "http://sensormeasurement.appspot.com/m3#Sensor"
        },
        "label": {
          "type": "literal",
          "xml:lang": "en",
          "value": "Wind Direction Sensor"
        },
        "comment": {
          "type": "literal",
          "xml:lang": "en",
          "value": "WindDirectionSensor, unit in Degree"
        },
        "imageUrl": {
          "type": "uri",
          "value": "http://sensormeasurement.appspot.com/images/sensor/windDirection.png"
        }
      }
    ]
  }
}
```

## 2. Web service: querying actuators

Search for all M3 actuators:

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=Actuator&format=json>

## 3. Web service: querying domains

Search for all M3 domains (=FeatureOfInterest):

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=FeatureOfInterest&format=json>

## 4. Web service: querying health devices

Search for all M3 health devices:

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=HealthM2MDevice&format=json>

## 5. Web service: querying transport devices

Search for all M3 transport devices:

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=TransportM2MDevice&format=json>

## 6. Web service: querying home devices

Search for all M3 home devices:

<http://www.sensormeasurement.appspot.com/m3/subclassOf/?nameClass=HomeM2MDevice&format=json>

# VII. LOV4IoT web services

Yan can download the LOV4IoT RDF dataset<sup>3</sup>.

Otherwise, we design some web services:

## 1. Web service: Get the total number of ontologies

Query:

<http://www.sensormeasurement.appspot.com/lov4iot/totalOnto/>

---

<sup>3</sup> <http://www.sensormeasurement.appspot.com/dataset/lov4iot-dataset>



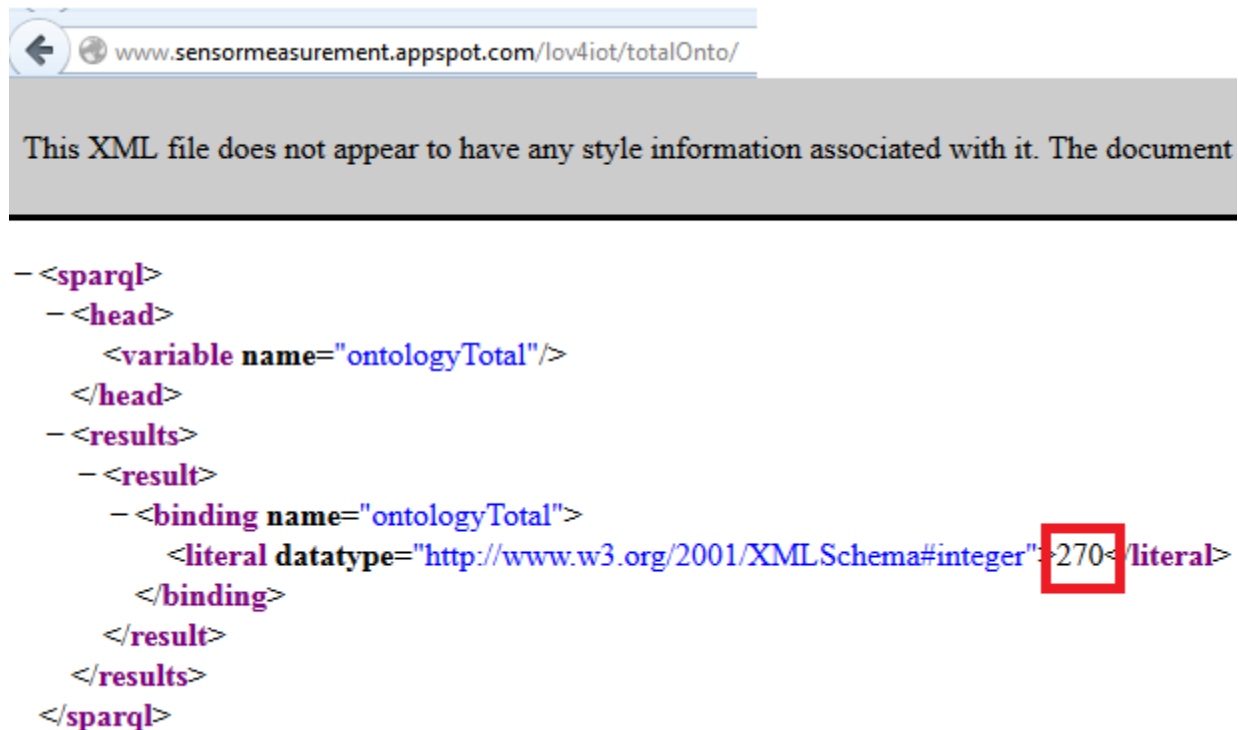


Figure 23. LOV4IoT Web service to count the total number of ontologies

In the picture, 270 is the total number of ontologies referenced in the LOV4IoT RDF dataset.

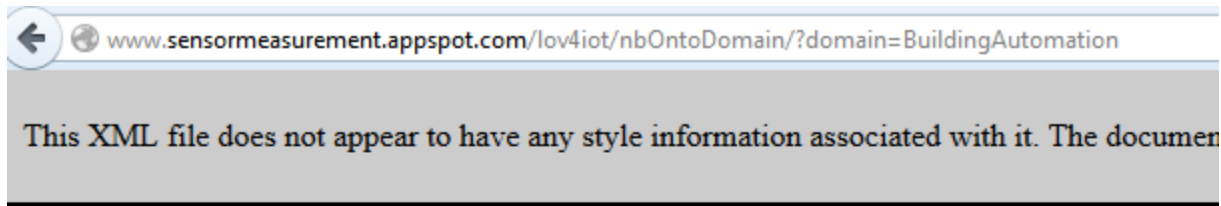
## 2. Web service: Get the number of ontologies by domains

Query:

<http://www.sensormeasurement.appspot.com/lov4iot/nbOntoDomain/?domain=BuildingAutomation>

For instance domain is: BuildingAutomation, Weather, Emotion, Agriculture, Health, Tourism, Transportation, City, Energy, Environment, TrackingFood, Activity, Fire, TrackingCD, TrackingDVD, SensorNetworks, Security.

The domain is referenced in the M3 nomenclature which is implemented in the M3 ontology (subclassOf FeatureOfInterest).



```
- <sparql>
  - <head>
    <variable name="nbOntoDomain"/>
  </head>
  - <results>
    - <result>
      - <binding name="nbOntoDomain">
        <literal datatype="http://www.w3.org/2001/XMLSchema#integer">45</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Figure 24. LOV4IoT Web service to count the number of ontologies by domain

### 3. Web service: Get the number of ontology by ontology status

Query:

<http://www.sensormeasurement.appspot.com/lov4iot/ontoStatus/?status=Online>

For instance, status is: Confidential, OngoingProcessOnline, WaitForAnswer, Online, OnelinLOV, AlreadyLOV.



```
-<sparql>
  -<head>
    <variable name="ontologyTotal"/>
  </head>
  -<results>
    -<result>
      -<binding name="ontologyTotal">
        <literal datatype="http://www.w3.org/2001/XMLSchema#integer">87</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Figure 25. LOV4IoT Web service to count the number of ontologies by ontology status

The web service returns that 87 ontologies referenced in the LOV4IoT RDF dataset are online.

## 4. Use case

All of these web services have been used in the HTML LOV4IoT web page<sup>4</sup> to automatically count the number of ontologies in the dataset (e.g., by domains, by ontology status, etc.)

---

<sup>4</sup> <http://www.sensormeasurement.appspot.com/?p=ontologies>

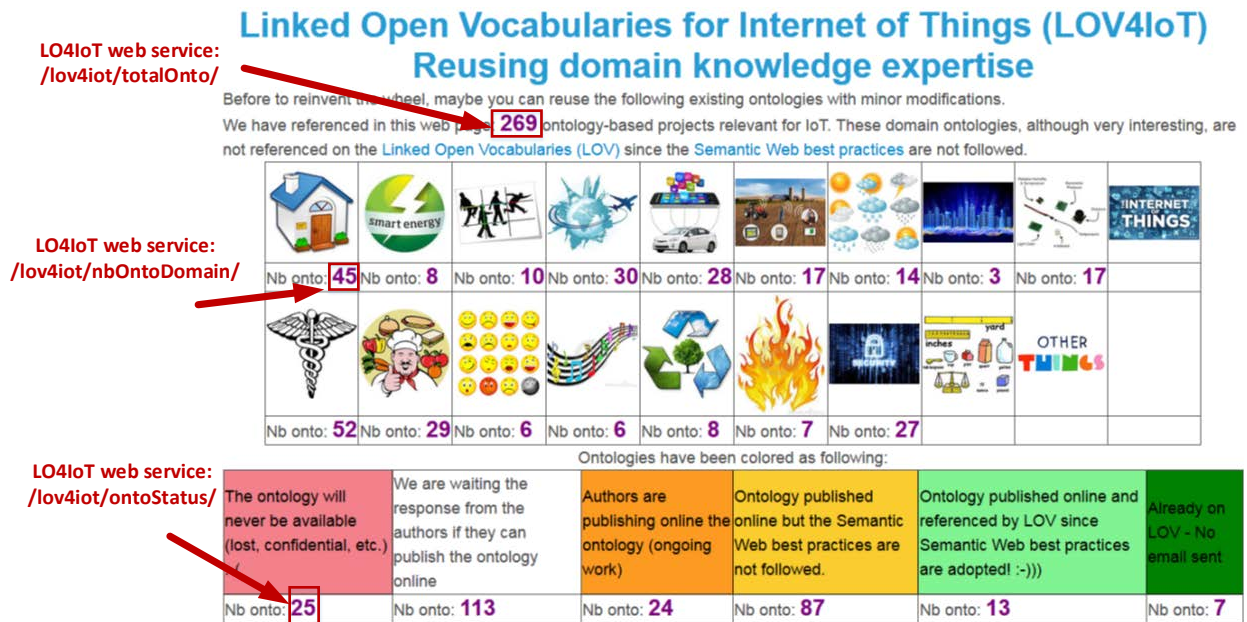


Figure 26. LOV4IoT web services

## VIII. Citations

If you use our work, please do not forget to cite us:

- [Standardizing generic cross-domain applications in Internet of Things \[Gyrard et al. 2014\]](#)
- [Helping IoT application developers with sensor-based linked open rules \[Gyrard et al., ISWC SSN 2014\]](#)
- [Enrich machine-to-machine data with semantic web technologies for cross-domain applications \[Gyrard et al., WF-IOT 2014\]](#)