Semantic Web of Things (SWOT) Generator

Creator	Amelie Gyrard (Eurecom - Insight - NUIG/DERI) Designed and implemented by Amélie Gyrard, she was a PhD student at Eurecom under the supervision of Prof. Christian Bonnet and Dr. Karima Boudaoud. Currently, SWoT generator is maintained since she is a post-doc researcher at Insight within the IoT unit led by Dr. Martin Serrano. She is highly involved in the FIESTA-IoT (Federated Interoperable Semantic IoT/Cloud Testbeds and Applications) H2020 project.
Contributors	Thanks to Pankesh Patel for fruitful questions and numerous questions ☺
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: amelie.gyrard@insight-centre.org
Google Group	https://groups.google.com/d/forum/m3-semantic-web-of-things (Not really active yet)
Last updated	June 2016
Created	June 2016
Status	Work in progress
Goal	This documentation enables understanding the SWoT generator: • Generating an IoT application template with the user interface • Generating an IoT application template web services • Understanding the M3. Code related to the SWoT generator • Documentation built for the ISWC 2016 and Demos

Table of contents

I. SWoT generator Citations 6

II. SWoT generator Architecture 6

III. Using the HTML User interface 6

IV. Tutorial: Building the naturopathy application with the user interface SWoT generator and the Jena framework 8

1. Generating the naturopathy template with the SWoT generator 8
2. Understanding the naturopathy template 8
3. Getting the sensor dataset already converted with M3 9
4. Be familiar with the Jena framework 10
5. Loading the sensor dataset in your Java application with the Jena framework 10
6. Loading the ontologies and datasets in your Java application with the Jena framework 11
7. Loading the rules and execute the Jena reasoner 12
8. Modifying the SPARQL query 12

VariableSparql Java Class: 12

9. Executingthe SPARQL query with Jena 13

ExecuteSparqlGeneric Java class 13

ExecuteSparqlGeneric Java class 14

10. Checking that the naturopathy application works 14

V. Generating IoT templates with M3 user interface or web services 15

1. M3 User interface 15
2. M3 Web Service: looking for IoT application template 16
3. M3 Web Service: generating IoT application template 17
4. M3 Web Service: generating the SPARQL query with variables replaced 18
5. Code example 20

VI. Adding a new SWoT template 21

VII. SWoT generator sequence diagram 21

VIII. Iot application template RDF dataset 22

IX. Understanding M3 web services 23

1. M3WS 24

X. References 25

Table of figures

FIGURE 1. SWOT GENERATOR ARCHITECTURE 6
FIGURE 2. GENERATING SEMANTIC WEB OF THINGS TEMPLATES 7
FIGURE 3. ZIP FILE GENERATED WITH DOMAIN KNOWLEDGE FOR INTERPRETING SENSOR DATA 7
FIGURE 4. DOWNLOAD THE NATUROPATHY TEMPLATE USING THE SWOT GENERATOR 8
FIGURE 5. THE NATUROPATHY TEMPLATE 9
FIGURE 6. EXTRACT OF THE SENSOR DATASET 10
FIGURE 7.LOAD THE SENSOR DATASET WITH JENA 10
FIGURE 8.LOAD A FILE (ONTOLOGY OR RDF DATASET) IN THE JENA MODEL 11
FIGURE 9.LOAD RULES AND EXECUTE THE JENA REASONER 12
FIGURE 10.MODIFY VARIABLES IN THE SPARQL QUERY 12
FIGURE 11. THE VARIABLESPARQL JAVA CLASS EXAMPLE 13
FIGURE 12. EXECUTE THE SPARQL QUERY EXAMPLE 13
FIGURE 13. GET THE RESULT OF THE SPARQL QUERY, MORE PRECISELY THE HIGH LEVEL ABSTRACTIONS 13
FIGURE 14. EXECUTESPARQLGENERIC JAVA CLASS EXAMPLE 14
FIGURE 15. SUGGESTIONS PROVIDED BY THE SPARQL QUERY FROM THE TEMPLATE 15
FIGURE 16. GENERATING M3 TEMPLATES USING M3 USER INTERFACE 16
FIGURE 17. LOOKING FOR THE M3 TEMPLATES 17
FIGURE 18. GENERATING THE M3 SPARQL QUERY 19
FIGURE 19. GENERATING M3 TEMPLATES USING M3 WEB SERVICES 20
FIGURE 20. A SWOT TEMPLATE 21
FIGURE 21. SWOT GENERATOR SEQUENCE DIAGRAM 22
FIGURE 22. SWOT GENERATOR SEQUENCE DIAGRAM EXAMPLE 22
FIGURE 23. THE IOT APPLICATION TEMPLATE DATASET 23
FIGURE 24. INSTANCE OF TEMPLATE 23
FIGURE 25. M3 WEB SERVICES USED IN THE SWOT GENERATOR 25

Terms and acronyms

IoT	Internet of Things (IoT)
SWoT	Semantic Web of Things
M3 framework	Machine-to-Machine Measurement (M3) framework

I. SWoT generator Citations

Please do not forget to cite our SWoT generator work:

- Assisting IoT Projects and Developers in Designing Interoperable Semantic Web of Things Applications. The 8th IEEE International Conference on Internet of Things (iThings 2015), 11-13 December 2015, Sydney, Australia. Amelie Gyrard, Christian Bonnet, Karima Boudaoud, Martin Serrano
- Cross-Domain Internet of Things Application Development: M3 Framework and Evaluation 3rd International Conference on Future Internet of Things and Cloud (FiCloud 2015), 24-26 August 2015, Rome, Italy. Amelie Gyrard, Soumya Kanti Datta, Christian Bonnet, Karima Boudaoud
- Standardizing Generic Cross-Domain Applications in Internet of Things. Third Workshop on Telecommunications Standards, Part of IEEE Globecom 2014, Austin, TX, USA, 8-12 December 2014. Amelie Gyrard, Soumya Kanti Datta, Christian Bonnet and Karima Boudaoud

II. SWoT generator Architecture

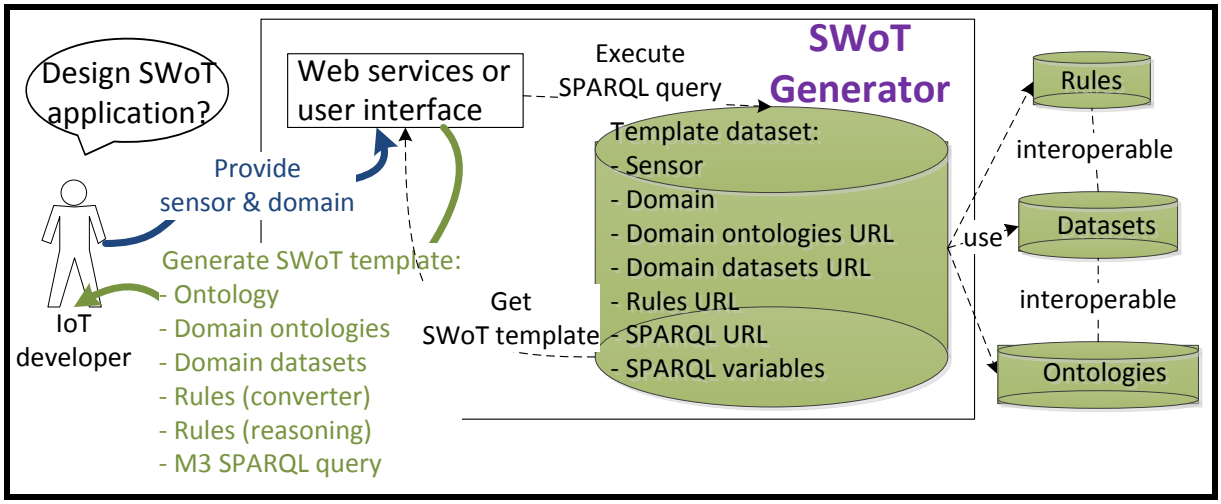


Figure 1. SWoT generator Architecture

III. Using the HTML User interface

The main purpose of the template generated is to interpret IoT data to provide suggestions.

- ➔ Go to this web page: http://www.sensormeasurement.appspot.com/?p=m3api (see Figure 2)
- ➔ Choose a sensor (e.g., Precipitation)
- ➔ Choose a domain (e.g., Weather)
- ➔ Click on the button "Search IoT application template"
- ➔ The drop-down list in STEP 2 is not empty anymore
- ➔ Choose a template (e.g., Precipitation, Transportation and Safety devices)

➔ Click on the button “Generate zip file”
➔ A zip file has been generated with interoperable M3 and domain ontologies, rules and datasets (Figure 3).

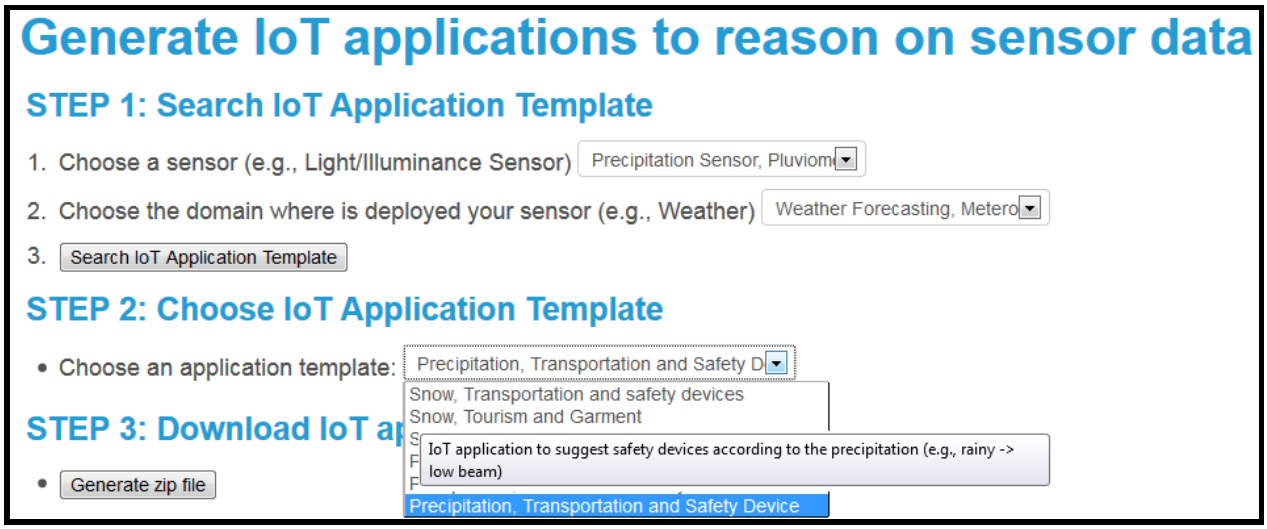


Figure 2. Generating Semantic Web of Things templates

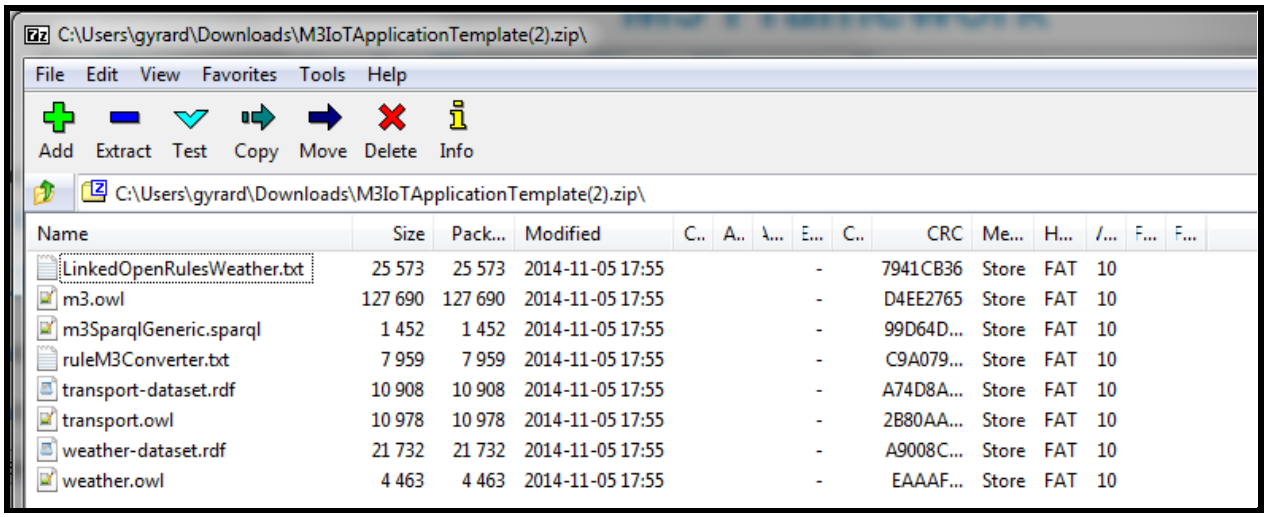


Figure 3. Zip file generated with domain knowledge for interpreting sensor data

IV. 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.'

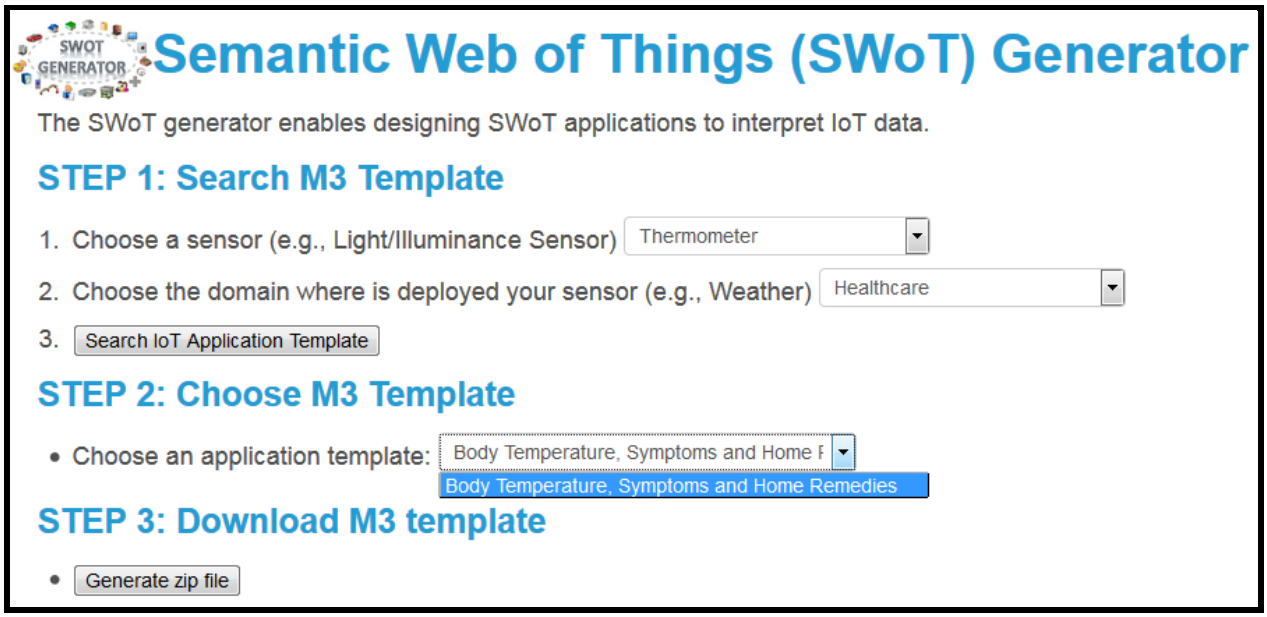


Figure 4. Download the naturopathy template using the SWoT generator

2. Understanding 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.

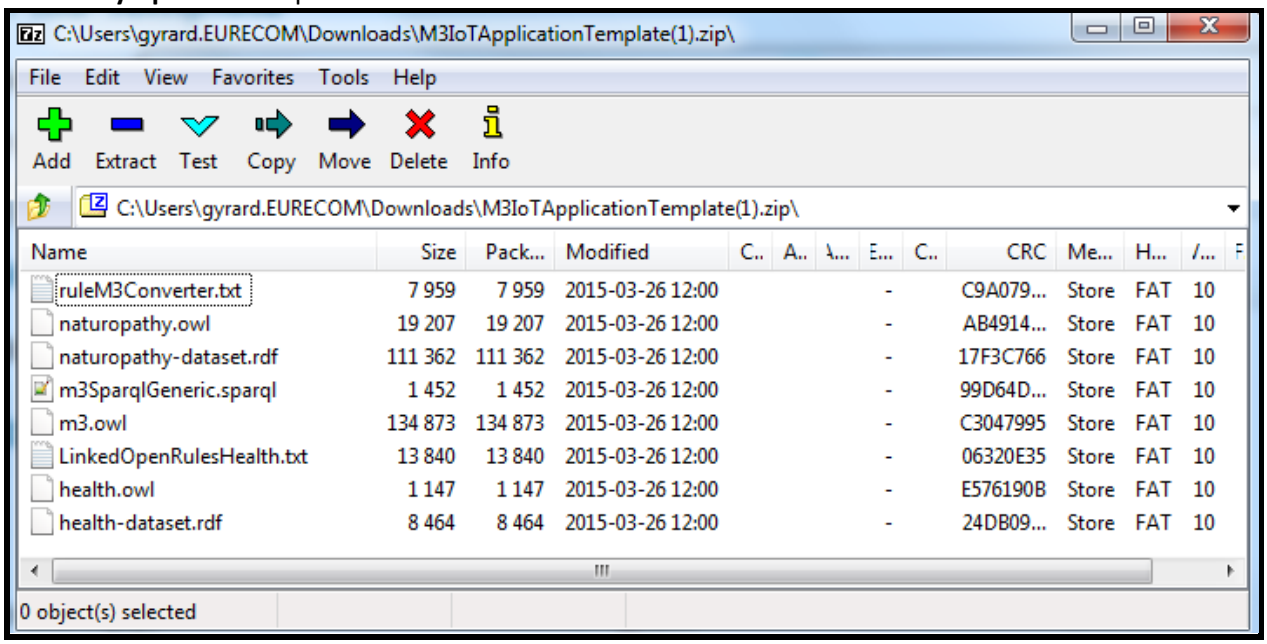


Figure 5. The naturopathy template

3. Getting the sensor dataset already converted with M3

- Download the sensor dataset: 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.

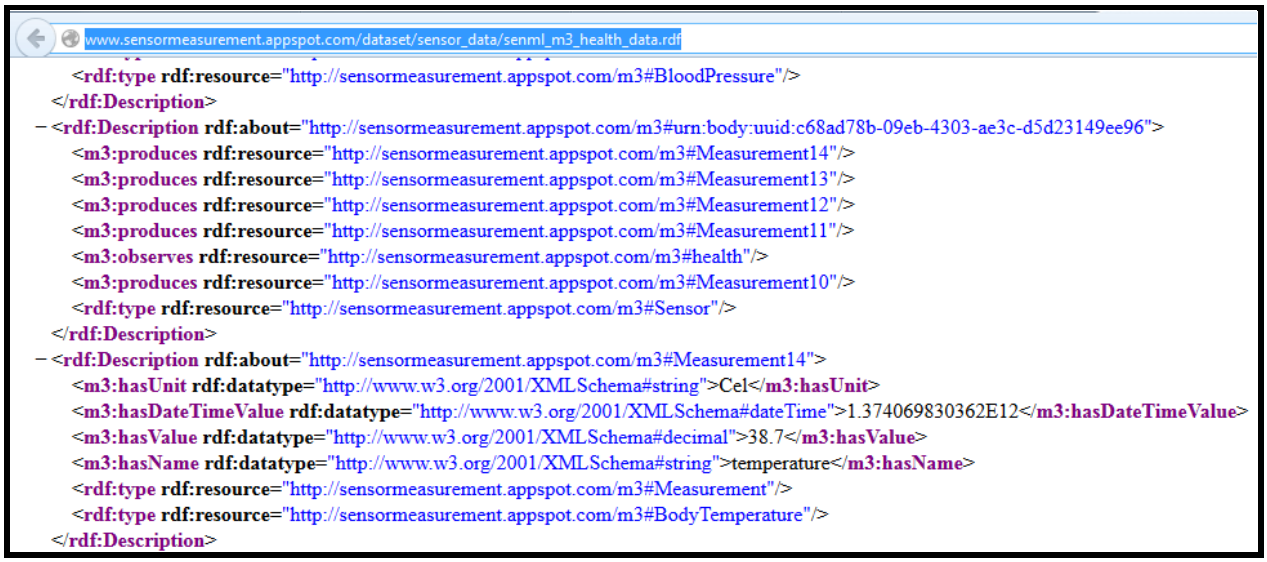


Figure 6. Extract of the sensor dataset

4. Be familiar with the Jena framework

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

5. Loading 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 7.Load the Sensor dataset with Jena

a. ReadFile Java Class:

Java code example:

```java
/**
 * Read ontologies or RDF dataset,
 * included directly from the file (a path) and add it to the jena model
 * @param model
 * @param file
 */
public static void enrichJenaModelOntologyDataset(Model model, String file){
    try {
        InputStream in = new FileInputStream(file);
        model.read( in, file );//file:"+
        in.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Figure 8.Load a file (ontology or RDF dataset) in the Jena model

6. Loading 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);

7. Loading the rules and execute the Jena reasoner

// load **LinkedOpenRulesHealth.txt**

```
//reasoner for jena rules
// the reasoner will infer new triples
// and high level abstraction from sensor data

// read rules
Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL(PATH + LinkedOpenRulesHealth.txt));

//for android use Rule.parseRule
reasoner.setDerivationLogging(true);

//apply the reasoner
InfModel inf = ModelFactory.createInfModel(reasoner, model);
return inf;
}
```

Figure 9.Load rules and execute the Jena reasoner

8. Modifying the SPARQL query

Java code example to modify the SPARQL query with variables:

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

Figure 10.Modify variables in the SPARQL query

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

VariableSparql Java Class:

```java
/**
* To change the values of some variables in sparql queries
*/
public class VariableSparql {

private String variableName;
private String value;
private boolean isLiterral;
public String getVariableName() { return variableName; }
public VariableSparql(String variableName, String value, boolean isLiterral) {
   super();
   this.variableName = variableName;
   this.value = value;
   this.isLiterral = isLiterral;
}
public void setVariableName(String variableName) { this.variableName = variableName; }
public String getValue() { return value; }
public void setValue(String value) { this.value = value; }
public boolean isLiterral() {   return isLiterral; }
public void setLiterral(boolean isLiterral) {  this.isLiterral = isLiterral; }
}
```

Figure 11. The VariableSparql Java Class example

9. Executingthe SPARQL query with Jena

// **load m3SparqlGeneric.sparql**

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

```

Figure 12. Execute the SPARQL query example

ExecuteSparqlGeneric Java class

```java
public String getSelectResultAsXML(ArrayList<VariableSparql> var){
     QueryExecution qe = replaceVariableInRequest(this.model, this.query, var);
     //get result from sparql request
     ResultSet results = qe.execSelect() ;
     String res = "No results";
     res = ResultSetFormatter.asXMLString(results);

     qe.close();
     return res;
  }

```

Figure 13. Get the result of the SPARQL query, more precisely the high level abstractions

ExecuteSparqlGeneric Java class

```
public Model model;
public Query query;

public ExecuteSparqlGeneric(Model model, String sparqlRequest) {
    super();
    this.model = model;

    //load the sparql query
    this.query = QueryFactory.create(ReadFile.readContentFile(sparqlRequest));
}

public static QueryExecution replaceVariableInRequest(Model model, Query query, ArrayList<VariableSparql> var){
    QueryExecution qe = null;
    RDFNode node = null;
    QuerySolutionMap initialBinding = new QuerySolutionMap();
    //replace sparql request by variables
    if(var!=null){
        for (VariableSparql variableSparql : var) {
            if (variableSparql.isLiterral()){
                node = model.createLiteral(variableSparql.getValue());
            }
            else{
                node = model.getResource(variableSparql.getValue());
                //System.out.println("node: " + node);
            }
            initialBinding.add(variableSparql.getVariableName(), node);
        }
        qe = QueryExecutionFactory.create(query, model, initialBinding);
    }
    else{
        qe = QueryExecutionFactory.create(query, model);
    }
    return qe;
 }
```

Figure 14. ExecuteSparqlGeneric Java class example

10. Checking 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.

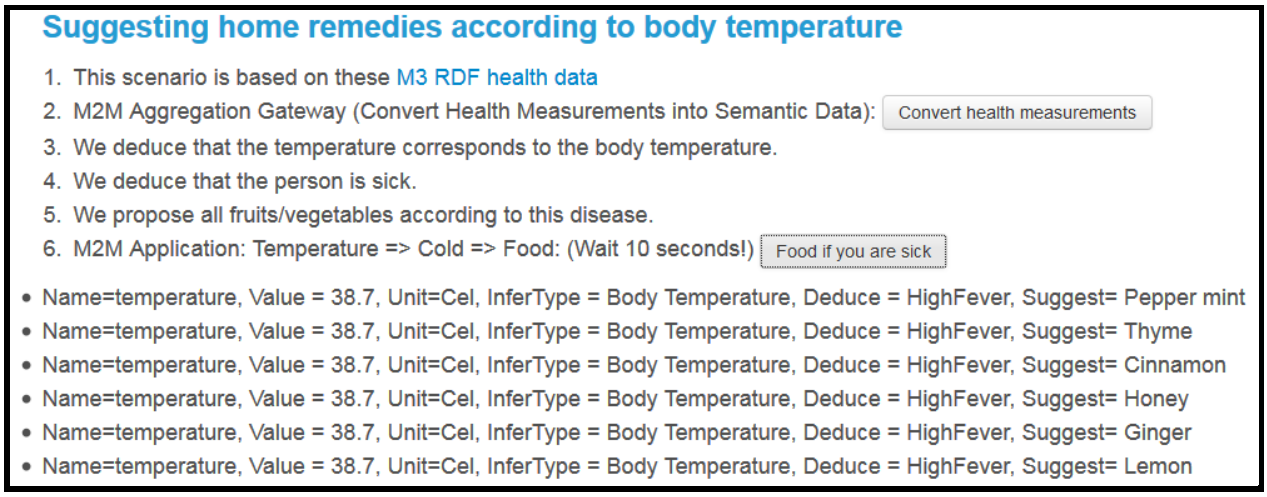


Figure 15. Suggestions provided by the SPARQL query from the template

V. 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

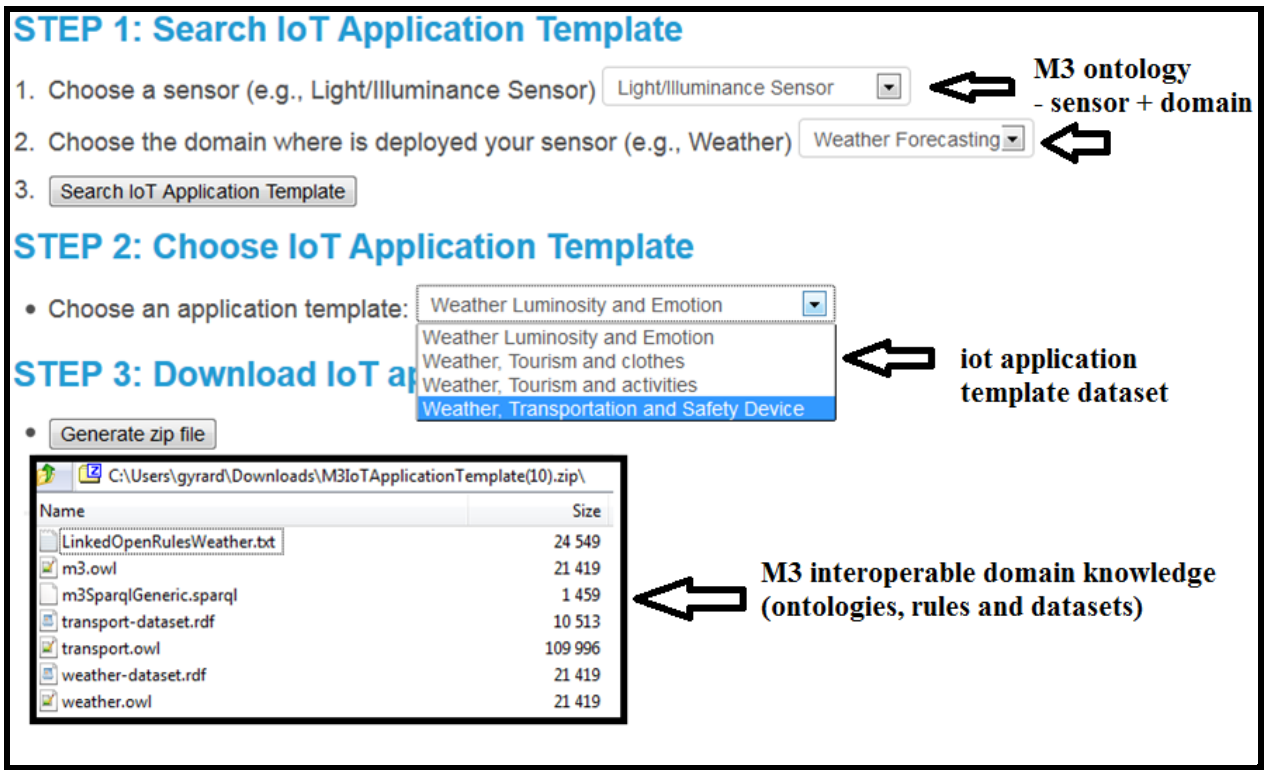


Figure 16. Generating M3 templates using M3 user interface

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:

Figure 17. 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 wa 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:hasDateTimeValue ?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>. }
scenario
                        ?resUri rdfs:label ?suggest.
                        FILTER(LANGMATCHES(LANG(?suggest), "en"))
                        OPTIONAL{
                                ?resUri dc:description ?suggest_comment.
                                FILTER(LANGMATCHES(LANG(?suggest_comment), "en"))
                        }


                }
        }
}
```

Figure 18. Generating the M3 SPARQL query

5. Code example

```
String URL_M3_API = "http://www.sensormeasurement.appspot.com/m3/";

// STEP 1: Searching the M3 template fitting your needs
String m3_sensor = "LightSensor";
// paramameter sensorName according to the M3 nomenclature
String m3_domain = "Weather";
// paramameter domain according to the M3 nomenclature
String format = "xml"; // or json
String search_M3_template = queryWebService(URL_M3_API + "searchTemplate/?" +
                                            "sensorName=" + m3_sensor +
                                            "&domain=" + m3_domain +
                                            "&format="+ format);

// STEP 2: Choosing the M3 template
String m3_iotAppli = parse(search_M3_template);
// e.g.: = "WeatherTransportationSafetyDeviceLight";

// STEP 3: Generating the M3 template
String m3_template = queryWebService(URL_M3_API + "generateTemplate/?" +
                                                  "iotAppli="+ m3_iotAppli);
// paramameter m3_iotAppli found in STEP 2

// STEP 4: Getting M3 ontologies, datasets and rules
String[] url_file = parse(m3_template);
for each url_file
    String[] url_M3_ontology = download(url_file);
    String[] url_M3_dataset = download(url_file);
    String[] url_M3_rule = download(url_file);

// STEP 5: Getting the SPARQL Query (with variables replaced)
String m3_sparql = queryWebService(URL_M3_API + "getSparqlQuery/? +
                                                "iotAppli="+ m3_iotAppli);
```

Figure 19. Generating M3 templates using M3 web services

VI. Adding a new SWoT template

Add a new template in the template dataset[1]:

- M3 is the prefix of the ontology.
- <m3:hasM2MDevice rdf:resource="&m3;LightSensor"/> means that the template is related to the light sensor which is already referenced in the M3 ontology
- <m3:hasContext rdf:resource="&m3;Weather"/> means that the template is related to the weather domain.
- <m3:hasUrlOntology rdf:resource="&weather;"/> the URL of the domain ontology required to build the Semantic Web of Things (SWoT) application
- <m3:hasUrlDataset rdf:resource="&transport-dataset;"/> the URL of the domain dataset required to build the Semantic Web of Things (SWoT) application
- <m3:hasUrlSparql rdf:resource="&sparql;m3SparqlGeneric.sparql"/> The URL of the SPARQL query
- <m3:hasSparqlVariableinferTypeUri rdf:resource="&m3;WeatherLuminosity"/> to replace variable in generic sparql queries (optionnal)
- <m3:hasSparqlVariabletypeRecommendedUri rdf:resource="&transport;SafetyDevice"/> to replace variable in generic sparql queries (optionnal)
- <m3:hasUrlRule rdf:resource="&lorWeather;"/> the URL of the Linked Open Rules dataset to get high level abstractions
- <m3:hasUrlRule rdf:resource="&ruleM3Converter;"/>the URL of the rule dataset to semantically annotate IoT data according to the M3 nomenclature and M3 ontology.

```
<m3:M2MApplication rdf:about="&m3;WeatherTransportationSafetyDeviceLight">
   <m3:hasContext rdf:resource="&m3;Weather"/>
   <m3:hasContext rdf:resource="&m3;Transportation"/>
   <rdfs:label xml:lang="en">Luminosity, Transportation and Safety Device</rdfs:label>
   <rdfs:comment xml:lang="en">IoT application to suggest safety devices according to the luminosity
   <m3:hasM2MDevice rdf:resource="&m3;LightSensor"/>                                    </rdfs:comment>
   <m3:hasUrlOntology rdf:resource="&m3;"/>
   <m3:hasUrlOntology rdf:resource="&weather;"/>
   <m3:hasUrlDataset rdf:resource="&weather-dataset;"/>
   <m3:hasUrlOntology rdf:resource="&transport;"/>
   <m3:hasUrlDataset rdf:resource="&transport-dataset;"/>
   <m3:hasUrlSparql rdf:resource="&sparql;m3SparqlGeneric.sparql"/>
   <m3:hasSparqlVariableinferTypeUri rdf:resource="&m3;WeatherLuminosity"/>
   <m3:hasSparqlVariabletypeRecommendedUri rdf:resource="&transport;SafetyDevice"/>
   <m3:hasUrlRule rdf:resource="&lorWeather;"/>
   <m3:hasUrlRule rdf:resource="&ruleM3Converter;"/>
</m3:M2MApplication>
```

Figure 20. A SWoT template

VII. SWoT generator sequence diagram

TO DO

[1] www.sensormeasurement.appspot.com/dataset/iot-application-template-dataset

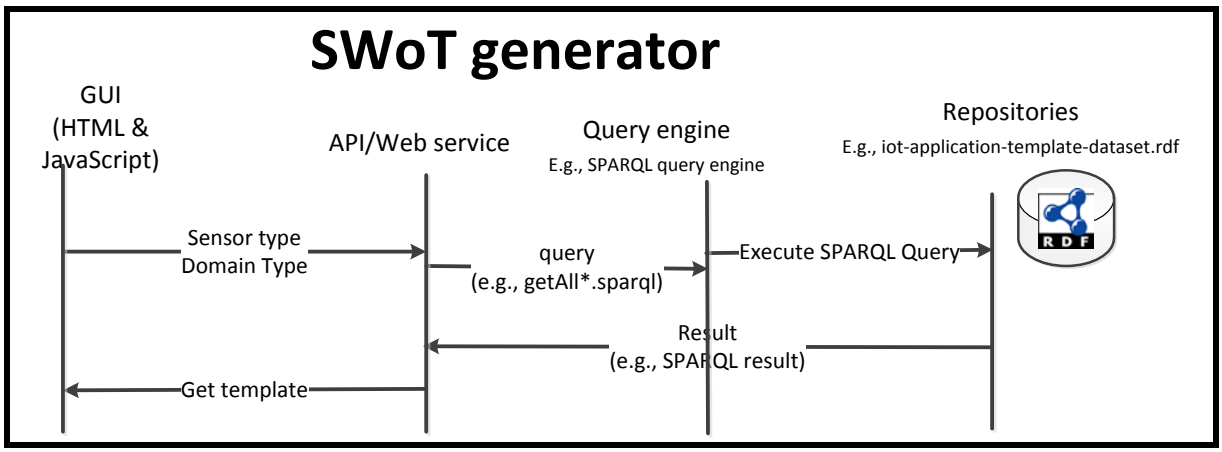


Figure 21. SWoT generator sequence diagram

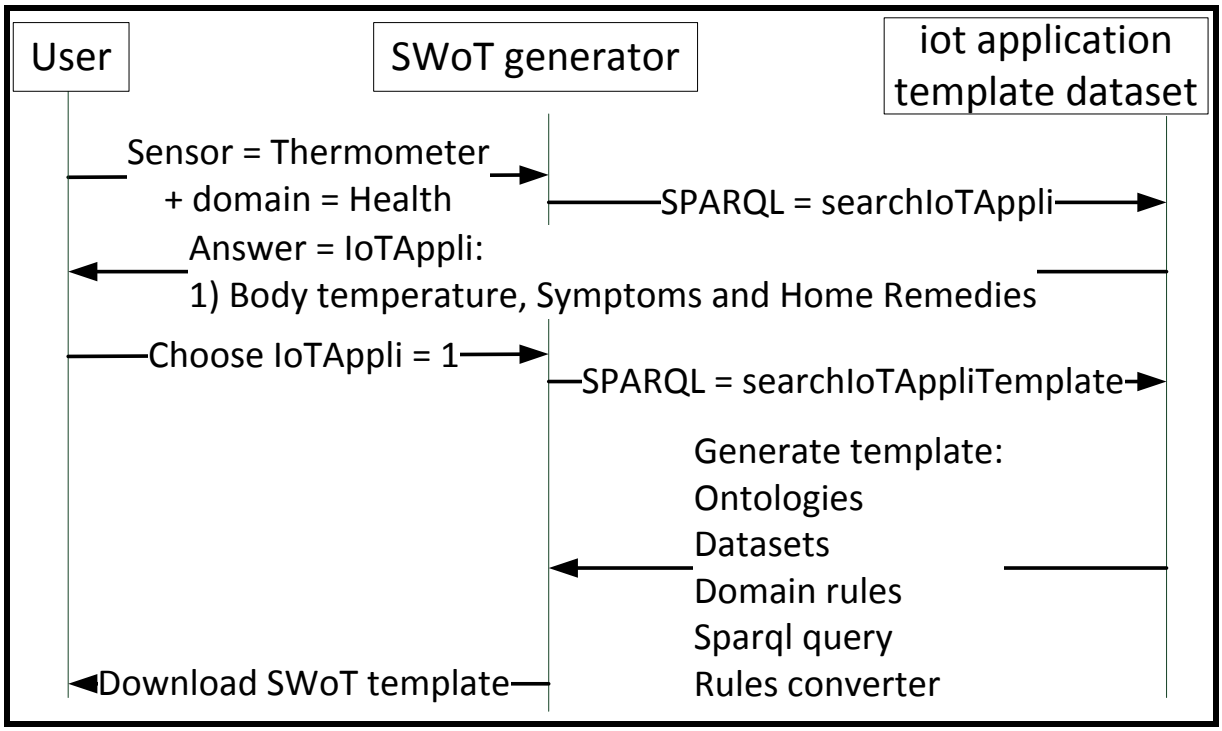


Figure 22. SWoT generator sequence diagram example

VIII. Iot application template RDF dataset

A dataset of pre-defined IoT application templates.

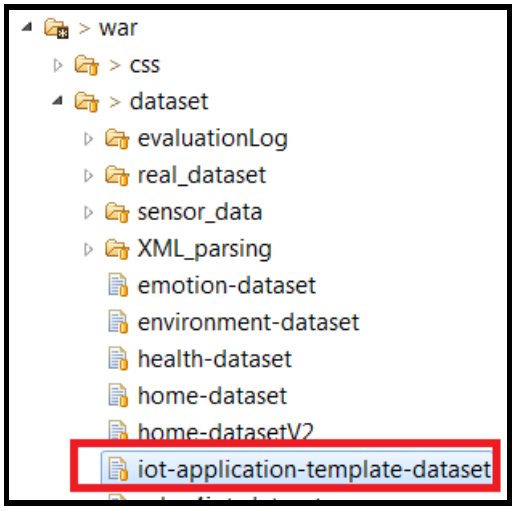


Figure 23. The IoT application template dataset

```
<m3:M2MApplication rdf:about="&m3;WeatherTransportationSafetyDeviceLight">
    <m3:hasContext rdf:resource="&m3;Weather"/>
    <m3:hasContext rdf:resource="&m3;Transportation"/>
    <rdfs:label xml:lang="en">Luminosity, Transportation and Safety Device</rdfs:label>
    <rdfs:comment xml:lang="en">IoT application to suggest safety devices according to the luminosity
    <m3:hasM2MDevice rdf:resource="&m3;LightSensor"/> => Sensor used                    </rdfs:comment>
    <m3:hasUrlOntology rdf:resource="&m3;"/>  => Ontology to annotate data
    <m3:hasUrlOntology rdf:resource="&weather;"/>
    <m3:hasUrlDataset rdf:resource="&weather-dataset;"/>          => Weather ontology and dataset
    <m3:hasUrlOntology rdf:resource="&transport;"/>
    <m3:hasUrlDataset rdf:resource="&transport-dataset;"/>        => Transport ontology and dataset
    <m3:hasUrlSparql rdf:resource="&sparql;m3SparqlGeneric.sparql"/>  => SPARQL query to get suggestions
    <m3:hasSparqlVariableinferTypeUri rdf:resource="&m3;WeatherLuminosity"/>
    <m3:hasSparqlVariabletypeRecommendedUri rdf:resource="&transport;SafetyDevice"/>
    <m3:hasUrlRule rdf:resource="&lorWeather;"/> => Rules to get high level abstractions
    <m3:hasUrlRule rdf:resource="&ruleM3Converter;"/>
</m3:M2MApplication>
```

Figure 24. Instance of template

IX. Understanding M3 web services

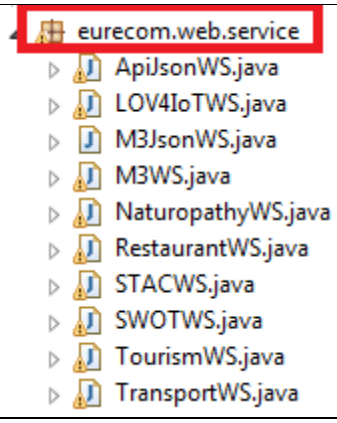
There is also the documentation to use the web services if required[2].

Root path web service: http://www.sensormeasurement.appspot.com/

In the package eurecom.web.service, you will find all web services, implemented in Java using the Jersey[3] implementation.

[2] http://www.sensormeasurement.appspot.com/documentation/M3APIDocumentation.pdf

[3] https://jersey.java.net/

	All web services names ended by WS in Java class

1. M3WS

All web services related to the M3 nomenclature implemented in the ontology.

Support new web services handling both XML and JSON format.

Should replace M3JsonWS and APIJsonWS Java classes:

- To semantically annotate sensor, IoT, M2M data (/m3/convert)
- Get all M3 sensors (/m3/subclassOf/sensor). This web service replaced M3JsonWS.
- Get all M3 domains (/m3/subclassOf/featureOfInterest). This web service replaced M3JsonWS.
- Get all M3 measurement type (/m3/subclassOf/measurement). This web service replaced M3JsonWS.

All web services related to the SWoT generator[4]:

- To look for templates (/m3/searchTemplate)
- To get the template (/m3/generateTemplate)
- To replace variables in the SPARQL query (/m3/getSparqlQuery)

[4] http://www.sensormeasurement.appspot.com/?p=m3api

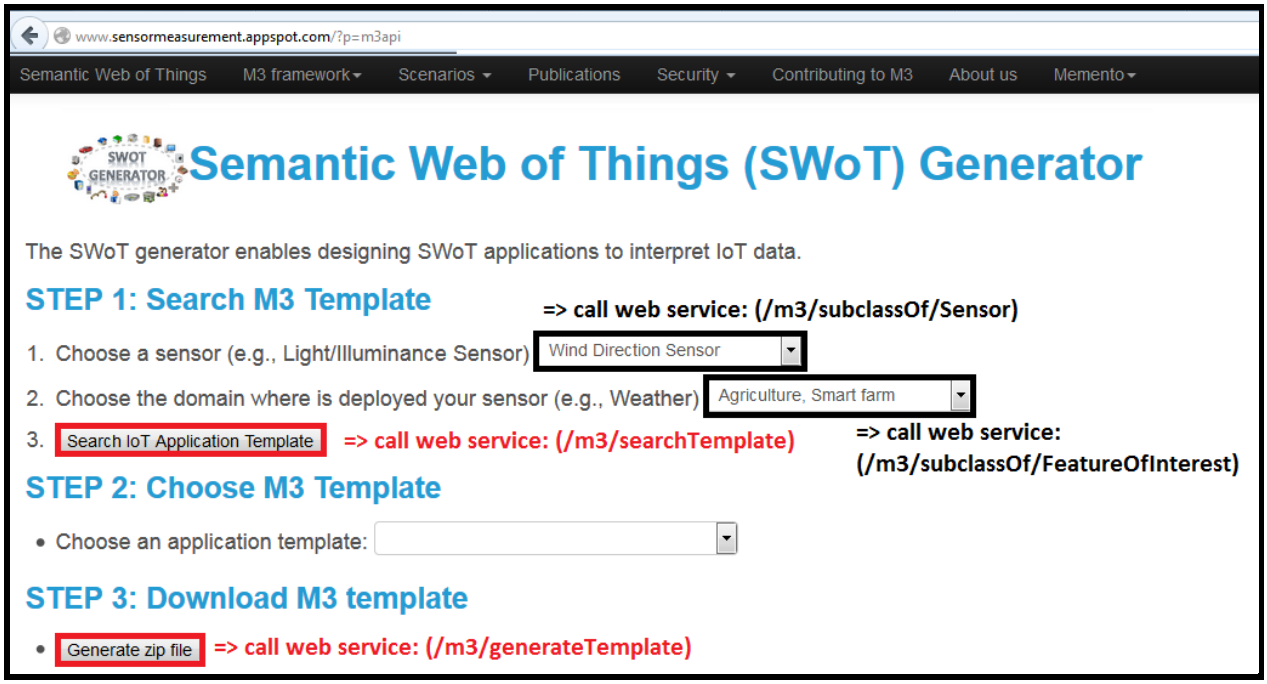


Figure 25. M3 web services used in the SWOT generator

X. References