


# M3 Framework Developer Documentation

To set up, run, understand the M3 project and exploit M3 web services



*Making smart discussions between things*

Creator	The M3 framework has been designed and implemented by Amélie Gyrard, she was a PhD student at Eurecom, Sophia Antipolis, France under the supervision of Prof. Christian Bonnet and Dr. Karima Boudaoud. Currently, Amélie Gyrard is a post-doc researcher at Insight/NUIG, Galway, Ireland.
Contact	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. <b><a href="mailto:amelie.gyrard@insight-centre.org">amelie.gyrard@insight-centre.org</a></b>
Code	<a href="https://github.com/gyrard/M3Framework">https://github.com/gyrard/M3Framework</a>
Documentation URL	<a href="http://www.sensormeasurement.appspot.com/documentation/M3DeveloperDocumentation.pdf">http://www.sensormeasurement.appspot.com/documentation/M3DeveloperDocumentation.pdf</a>
Last updated	February 25, 2016 - Update technology used in the project section + picture archi February 10, 2016 - Deploy M3 on the web – optional – section updated November 18, 2015 - Technologies used in the project - Get the M3 framework code on Github - Get the M3 required libraries and tools on Google Shared directory - Understanding the M3 converter
	<b>After three years and half of research and development, this project really needs a refactoring (even if I have done it frequently), and even maybe to be re-think from scratch. Anyway, this is the documentation to reuse, set up and understand the M3 project.</b>
Goal	<ul style="list-style-type: none"><li>• This documentation guides developers to set up and run the M3 project.</li><li>• Understanding the M3 project code (description of packages)</li><li>• Understanding the technologies used in the project</li><li>• Understanding M3 web services</li><li>• It also help understand the web services (to use it or add new ones if you have the code)</li><li>• Understanding the M3 converter</li></ul>

	<ul style="list-style-type: none"> <li>• Add a new Semantic Web of Things (SWoT) template</li> <li>• Reference a new ontology-based project in LOV4IoT (HTML web page &amp; RDF dataset)</li> <li>• Documentation to extend the STAC ontology and dataset</li> </ul>
Requirements	Java 1.7, Eclipse Kepler, App Engine SDK 1.9, Jena framework 2.11.
Technologies used	<ul style="list-style-type: none"> <li>• Java 1.7 is required to use Google App Engine</li> <li>• We used Eclipse Kepler to develop the M3 framework</li> <li>• Jena: a framework to build semantic web applications</li> <li>• Google App engine (App Engine SDK 1.9) to host the web site online (we do not need to have our own server and domain name)</li> <li>• Java Data Objects (JDO) to store data (e.g., semantic sensor data) on the google app engine datastore</li> <li>• RESTful Java, Jersey (JAX-RS) to develop Java web services</li> <li>• User interface: HTML, CSS, Bootstrap, JavaScript, AJAX</li> </ul>
System	Windows 7, 64 bits
Status	Work in progress
Time estimated	1 day to set up the project

## Table of Contents

Part I : Setting up the M3 project.....	4
I. Get the M3 framework code on Github .....	6
1. Download required libraries or tools (Eurecom users).....	7
2. Download required libraries or tools (External users).....	7
II. The NecessaryToSetUpM3 directory .....	8
III. Install Java and JRE 1.7 .....	10
IV. Install Eclipse Kepler .....	10
3. Use the default Eclipse workspace .....	12
4. Optional: Reference your own workspace.....	13
5. Result expected: Check that you can see the M3 project .....	14
V. (Optional) Install the Google Plugin for Eclipse .....	14
Part II : Running the M3 project on localhost .....	17
I. Discover the M3 project and run it.....	17
1. Fixing Google App Engine Problem .....	18
2. Java Build Path Problems .....	19
II. (Optional) Install Jena inside Eclipse.....	20
III. Run M3 on localhost .....	20

Part III :	Deploying M3 on the Web .....	21
I.	Creating an application on app engine .....	27
II.	Error App engine when deploying .....	28
Part IV :	Technologies used in M3 .....	4
I.	Jena .....	6
II.	Google App Engine.....	6
III.	RESTful implementation for JAVA: JAX-RS .....	6
Part V :	Understanding the M3 project .....	29
I.	M3 framework overview.....	29
II.	Libraries (Jars) required .....	31
III.	Descriptions of packages .....	32
1.	eurecom.common.util .....	33
2.	eurecom.constrained.devices.....	34
3.	eurecom.data.converter .....	34
4.	eurecom.generic.m2mapplication.....	34
5.	eurecom.sparql.result.....	35
6.	eurecom.store.jdo.....	35
7.	eurecom.search.knowledge.....	35
8.	eurecom.web.service.....	35
IV.	The WAR directory .....	36
1.	WAR/CSS .....	37
2.	WAR/dataset.....	37
3.	WAR/documentation .....	37
4.	WAR/html .....	37
5.	WAR/images .....	37
6.	WAR/javascript .....	37
7.	WAR/ont .....	38
8.	WAR/publication.....	38
9.	WAR/RULES.....	38
10.	WAR/SPARQL .....	38
11.	WAR/WEB-INF/ontologies .....	39
Part VI :	Understanding M3 web services.....	40

1. APIJsonWS Java class (deprecated) .....	40
2. LOV4IoTWS Java class .....	40
3. M3JsonWS (deprecated).....	42
4. M3WS.....	42
5. NaturopathyWS .....	43
6. RestaurantWS .....	44
7. STACWS.....	44
8. SWOTWS .....	45
9. TourismWS.....	46
10. TransportWS .....	47
11. Design a new web service .....	48
Part VII : Adding a new SWoT template.....	48
Part VIII : Adding a new ontology in LOV4IoT.....	49
1. HTML web page .....	49
2. LOV4IoT RDF dataset .....	49
Part IX : Understanding the M3 converter .....	50
I. Semantically annotate SenML/XML data with the M3 converter .....	50
II. Semantically annotate SenML/JSON data with the M3 converter .....	51
Part X : Extending STAC .....	52
I. Adding a new technology to the STAC ontology.....	52
II. Enriching the STAC dataset .....	53
3. Updating the STAC dataset with a new security mechanism .....	53
Part XI : Additional.....	54
I. Jena TDB and Jena Fuseki (Internship Amira) .....	54
II. Security issues to execute JavaScript with the project .....	55

## Part I : Technologies used in M3

We employed the following technologies as displayed in Figure 1:

- **Jena** 2.11 framework [2] is used to build semantic web applications. Jena includes the Jena reasoning engine to interpret IoT data and Jena/ARQ to execute SPARQL queries on the knowledge base. Jena was the easier Semantic Web framework to learn, it was well-documented and had tutorials. Integrating Jena to our application was simple thanks to the JAR file. A light version of Jena is available for constrained devices too.
- **RDF, RDFS and OWL** are used to design ontologies and datasets.
- **SPARQL** is used to query knowledge base designed with RDF, RDFS and OWL [5] .
- **Java 1.7** is used to implement the entire framework
- **Jersey** (JAX-RS) is used to design JAVA RESTful web services.
- **Google Web Toolkit (GWT)** and **Google App Engine (GAE)** are used to develop the framework and host the entire web site online. We do not have to maintain a server or taking care of security issues, it is easy to deploy and maintain applications online.
- **Java Data Object (JDO)** is used to store data in a database.
- **HTML5, CSS3, JavaScript** are used to design the Graphical User Interface (GUI)
- **AJAX** technologies are used query web services. Then, the result returned by web serviced is parsed with JavaScript.

Finally, the designing phase has been achieved by using extreme programming [1] and Scrum-like methodologies.

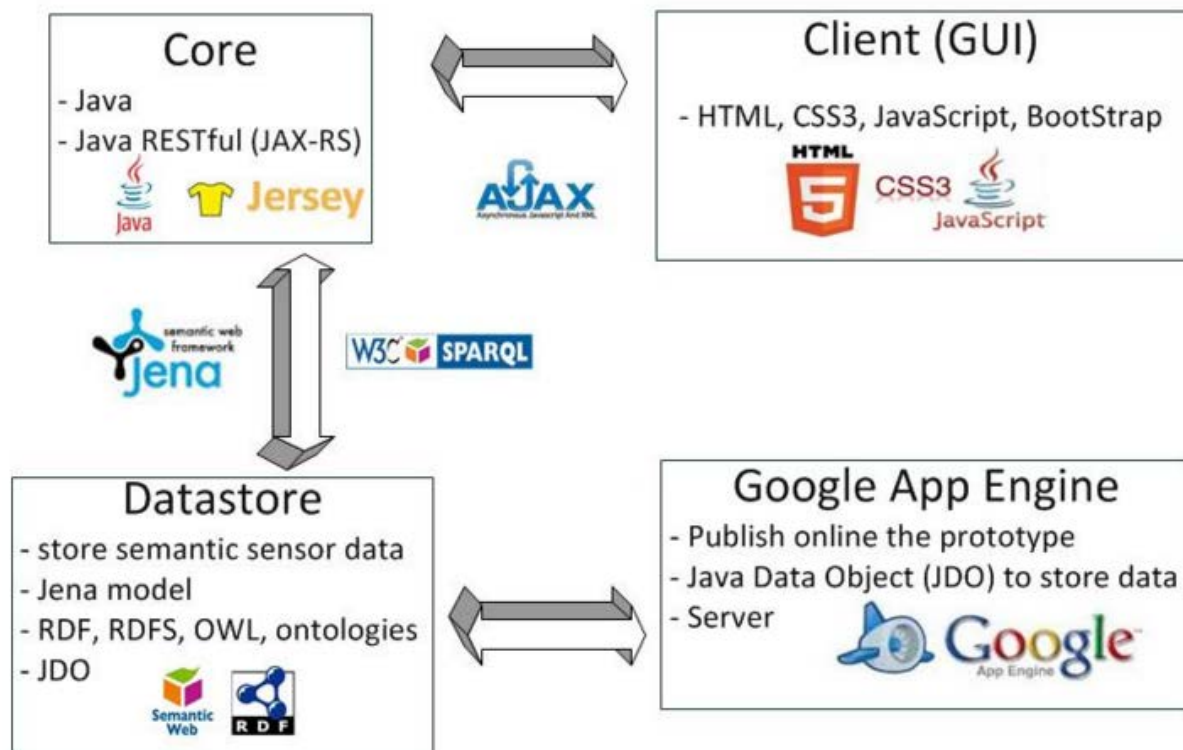


Figure 1. Technologies used is M3

## I. Jena

We used Jena<sup>1</sup>, a framework to build semantic web applications.

## II. Google App Engine

To host the M3 web site on the web, we used Google App Engine. It provides a server and the domain name (<http://sensormeasurement.appspot.com/>). “.appspot.com” means that the web site is hosted on Google, we do not need to pay for a domain name. Further, Google App Engine provides us a server, so we do not need to maintain it, and take care of security issues.

## III. RESTful implementation for JAVA: JAX-RS

To build RESTful web services in Java, we use the Jersey Java library<sup>2</sup>.

# Part II : Setting up the M3 project

## IV. Get the M3 framework code on Github

The M3 framework code is on Github: you can download the ZIP file (see Figure 2).

<https://github.com/gyrard/M3Framework>

---

<sup>1</sup> <https://jena.apache.org/>

<sup>2</sup> <https://jersey.java.net/>

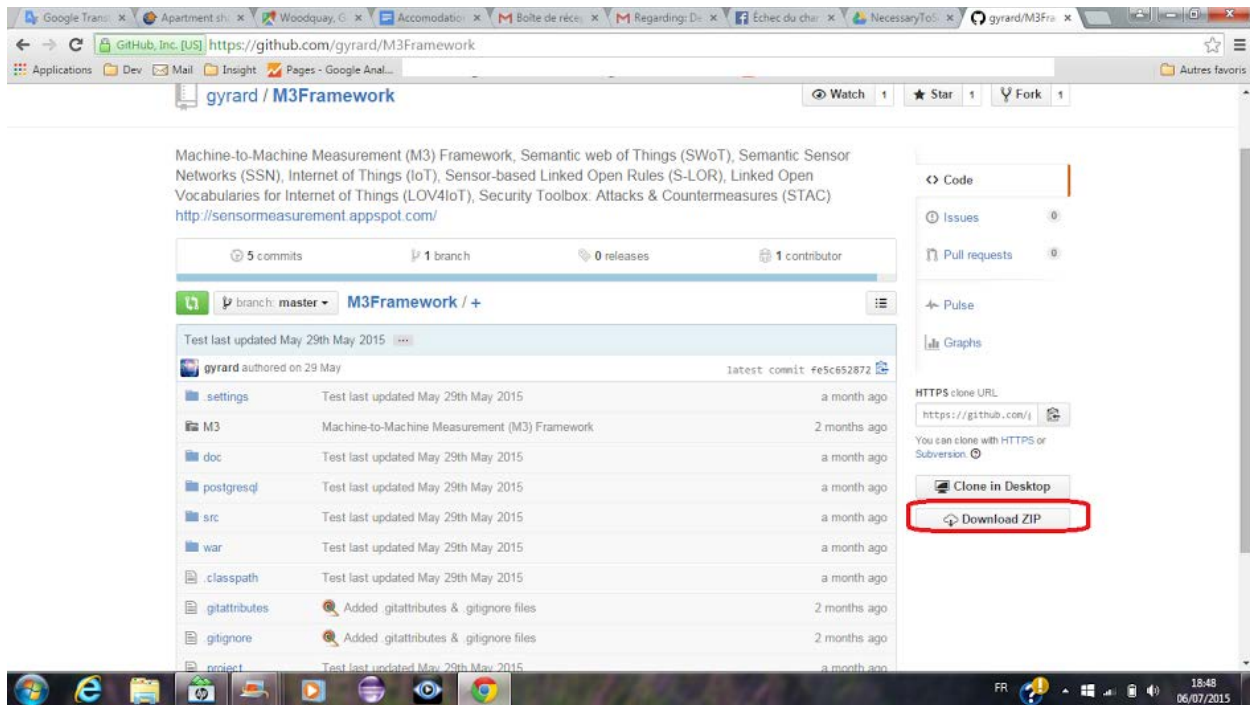


Figure 2. The M3 framework code on Github

## 1. Download required libraries or tools (Eurecom users)

On BSCW (For Eurecom users)

You have access to the directory SetUpM3 which is comprised of:

- ➔ NecessaryToSetUpM3 with eclipse Kepler, all JAR required and the JDK and JRE 1.7 (e.g., NecessaryToSetUpM3.zip)
- ➔ workspace with the M3 project (e.g., m3\_save2Avril2015.zip), see last version on Github
- ➔ Download and unzip both!

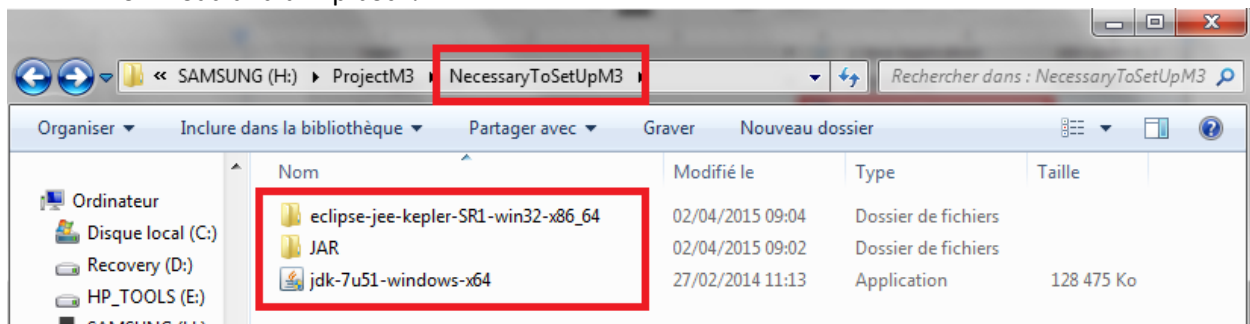


Figure 3. NecessaryToSetUpM3 directory

## 2. Download required libraries or tools (External users)

The M3 framework code is on Github: you can download the ZIP file (see Figure 1).

<https://github.com/gyrard/M3Framework>

For external users, ask us to share this directory:

<https://drive.google.com/drive/u/0/folders/0B5D7PRXO3e6afm1jUElVb0Z5S0dFVzJSU0VGWk01d3VtOTdiamMtUmpZOG1XM3RnRIBJSk0>

You will have access to this directory:

- ➔ NecessaryToSetUpM3 with eclipse Kepler, all JAR required and the JDK and JRE 1.7 (e.g., NecessaryToSetUpM3.zip)

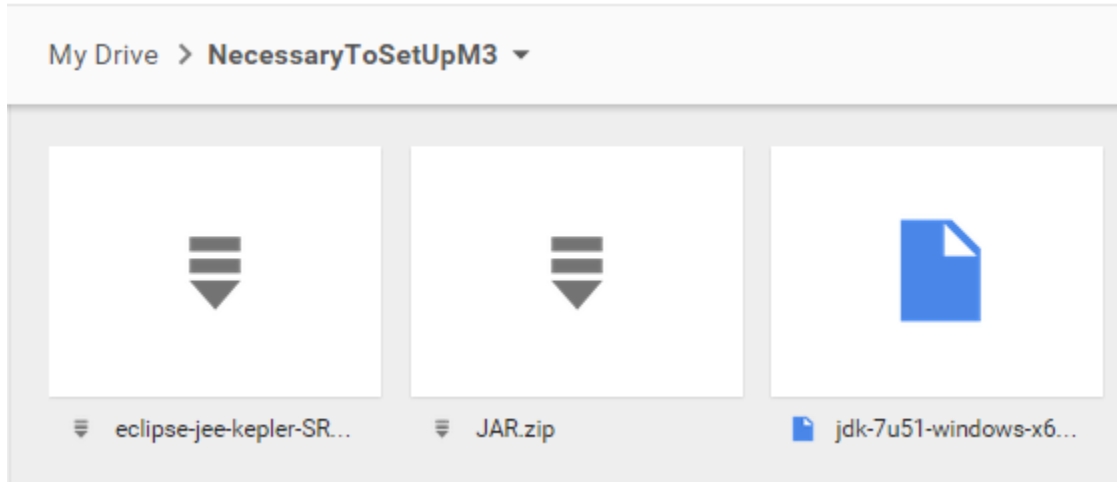


Figure 4. Download the NecessaryToSetUpM3 on Google Drive

## V. The NecessaryToSetUpM3 directory

Requirements

- Java 1.7
- Eclipse Kepler
- Plugin Google App Engine SDK App engine 1.9
- Jena
- Jars required

To avoid any incompatibility issues, we provide all tools with the good version:

The NecessaryToSetUpM3 directory is comprised of:

- ➔ Eclipse Kepler
- ➔ All jar required to run the M3 project
- ➔ JDK Java 1.7

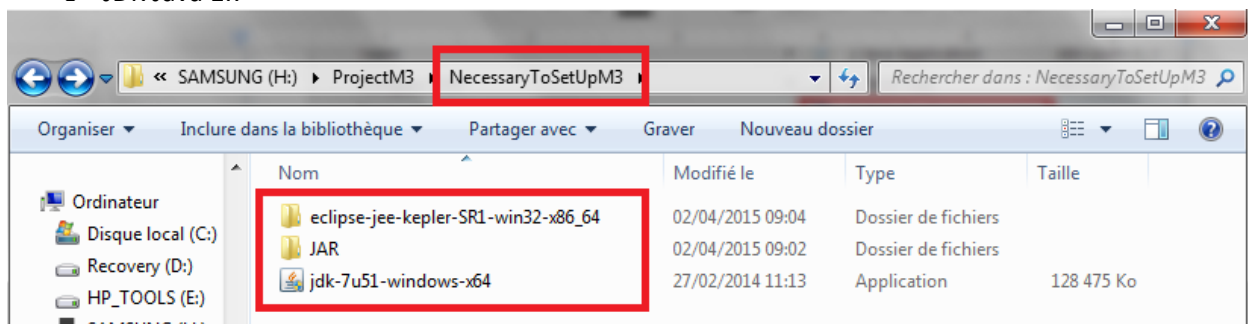
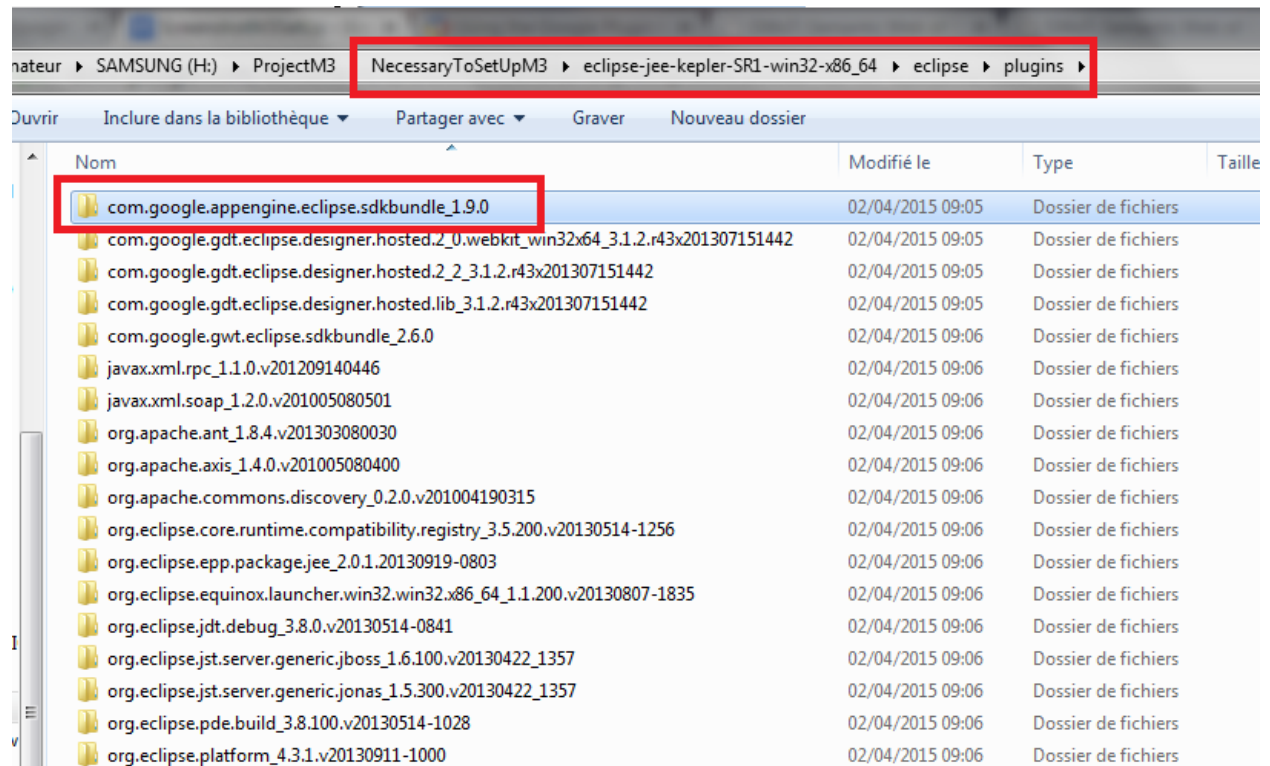


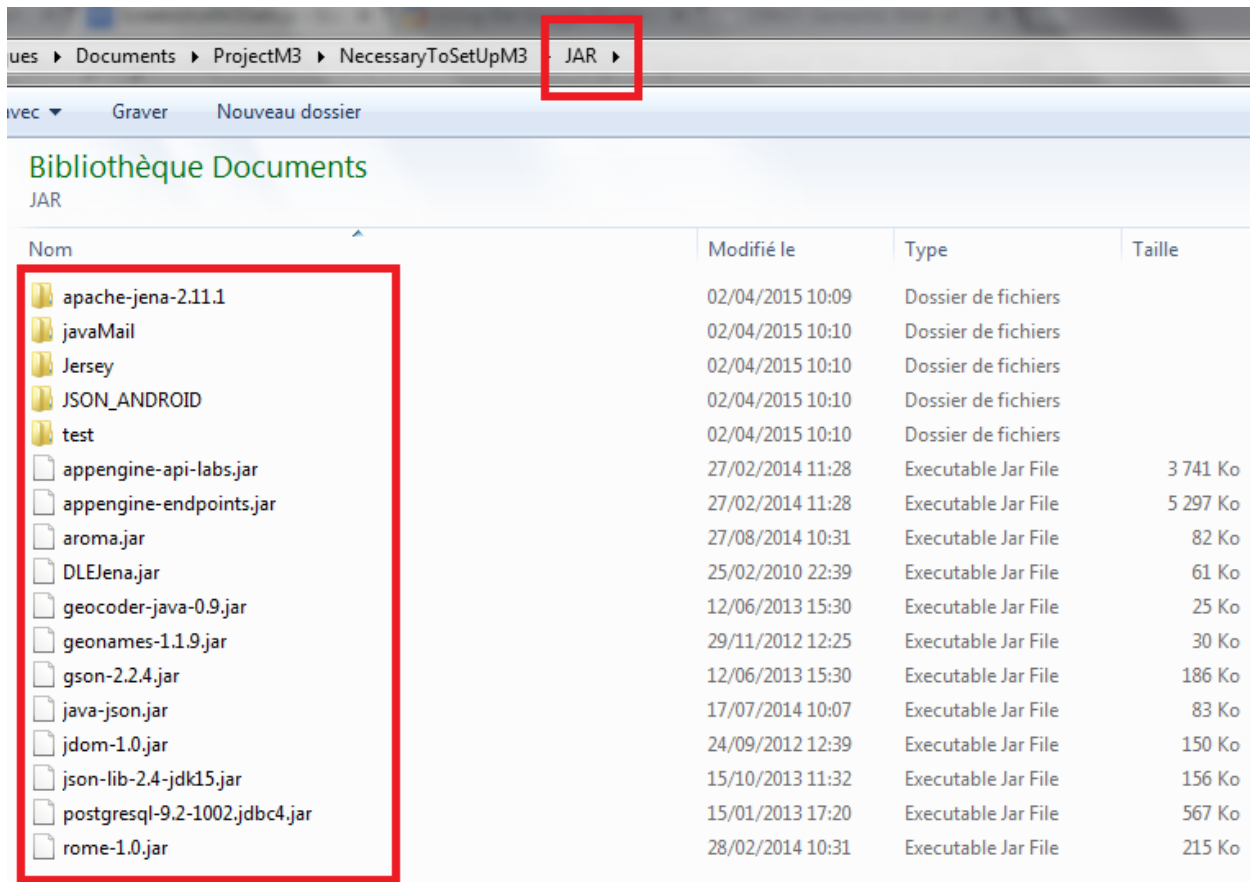
Figure 5. NecessaryToSetUpM3 directory



In the Eclipse directory you already have the plugin for App Engine SDK 1.9:



**Figure 6. App Engine SDK 1.9 already in the Eclipse that we provided**



**Figure 7. JARs required in the M3 project are provided in the JAR directory**

## VI. Install Java and JRE 1.7

In our case: jdk-7u51-windows-x64

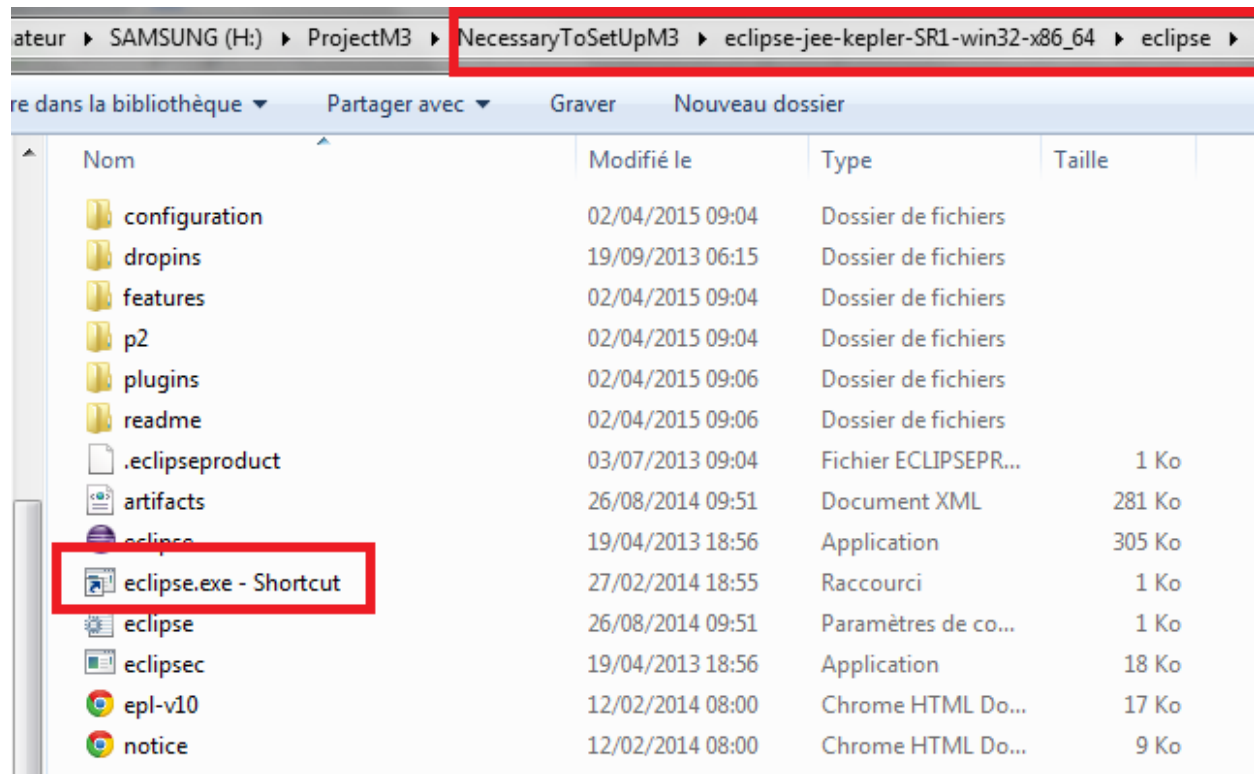
Because of Google App Engine, we cannot use anymore Java 1.6.

If you have Java 1.6, you should uninstall it, restart the machines, to avoid issues.  
(if it does not compromise the other projects running on your machine).

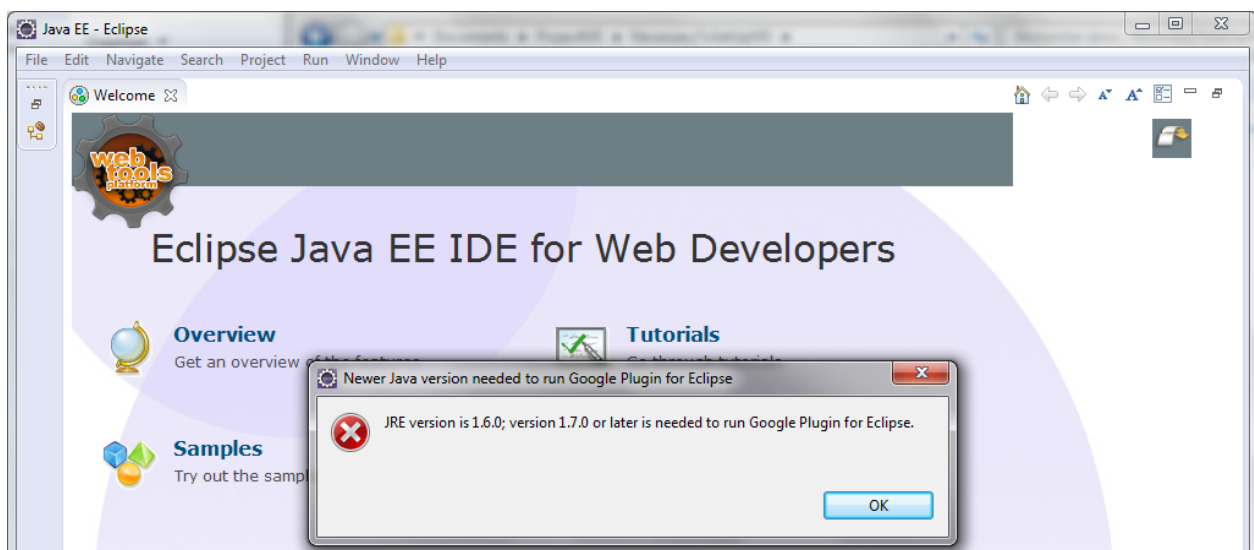
Double click on the JDK and install it.

## VII. Install Eclipse Kepler

In our case: eclipse-jee-kepler-SR1-win32-x86\_64

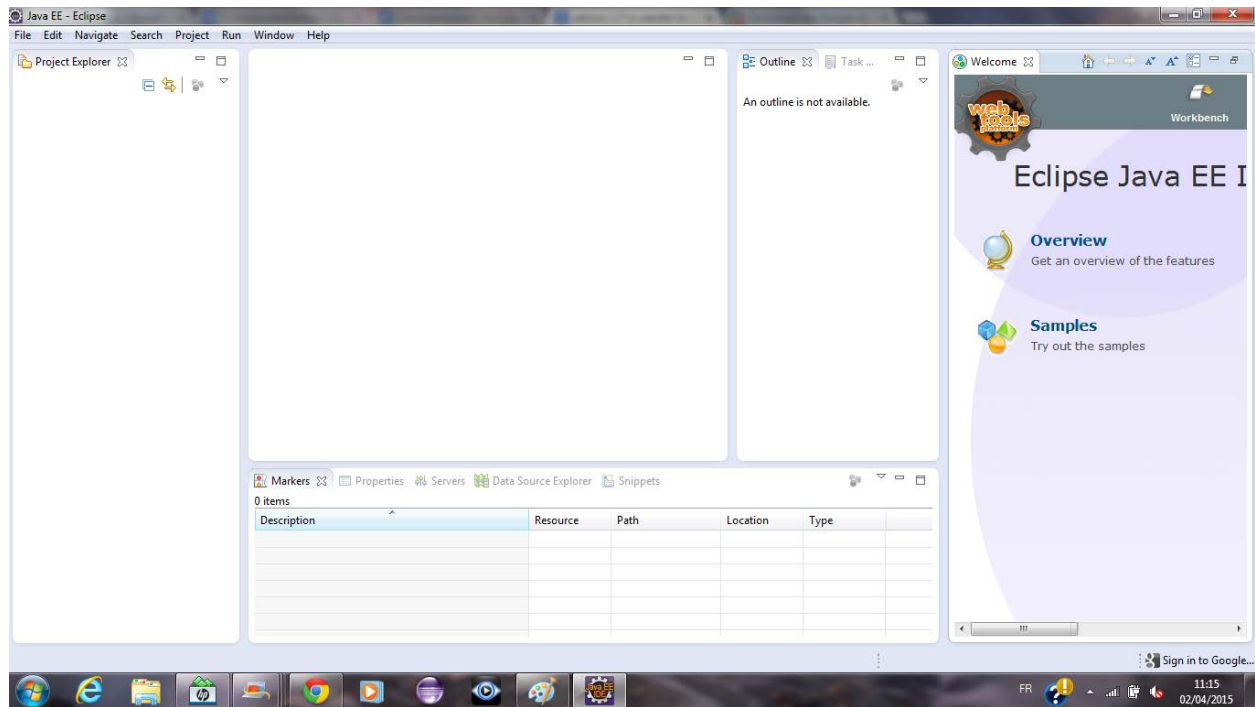


**Figure 8. Install Eclipse Kepler that we provided**



**Figure 9. Java 1.7 is required with the Google Plugin for Eclipse**

➔ Click ok (This is why we set up Java 1.7)



**Figure 10. The eclipse workspace is empty**

### **3. Use the default Eclipse workspace**

Eclipse installed by default a workspace: E:\Workspace.

In this case, close Eclipse, copy the workspace with the m3 project and replace the default workspace with this workspace.

Result expected:

#### 4. Optional: Reference your own workspace

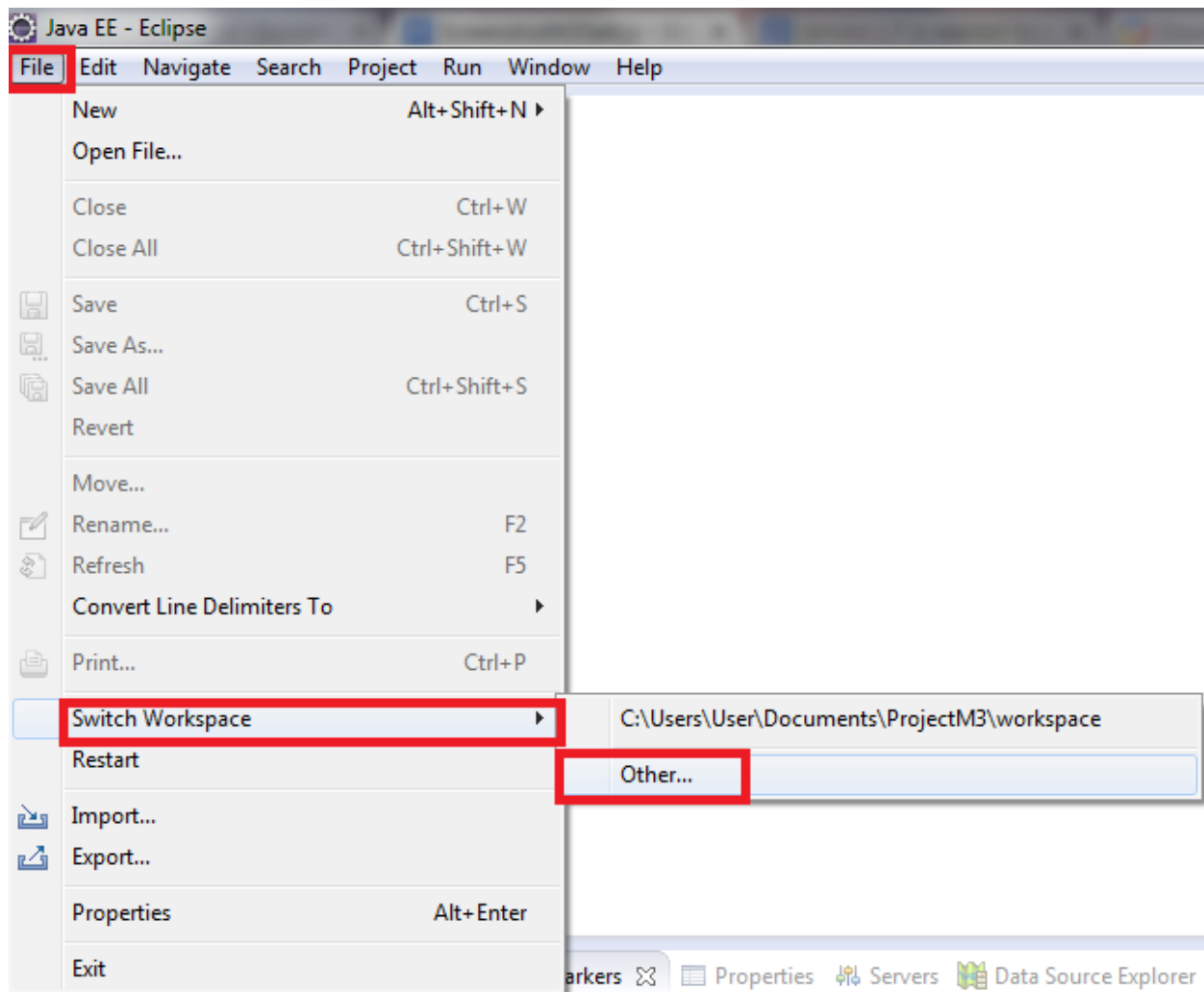


Figure 11. Switch workspace to reference the M3 project

We referenced the existing M3 workspace and not the one by default:

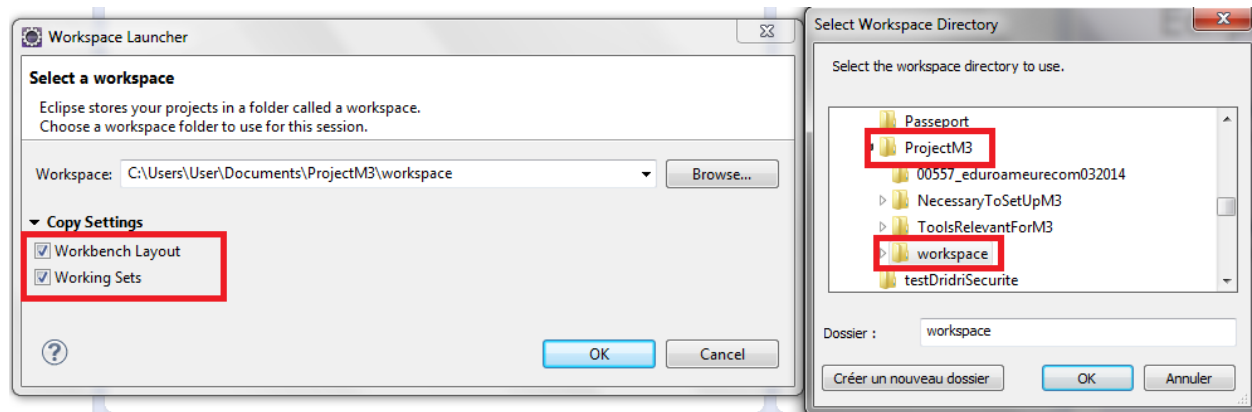


Figure 12. Reference the M3 project

## 5. Result expected: Check that you can see the M3 project

You should have the following screenshot:

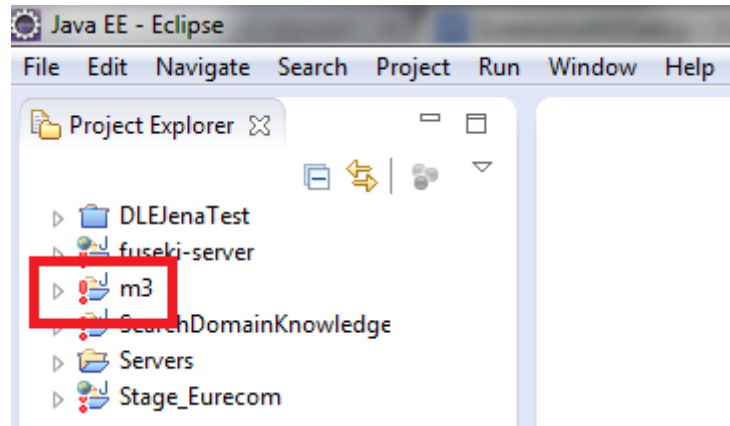


Figure 13. The M3 project in the Eclipse workspace

## VIII. (Optional) Install the Google Plugin for Eclipse

In case you encounter some issues, you can follow this tutorial:

<https://cloud.google.com/appengine/docs/java/tools/eclipse>



**We do not remember why, but we have to change the default setting of Google:  
M3 project -> Right Click -> Google -> App Engine Settings  
→ Choose DataNucleus JPO/JPA version v1**

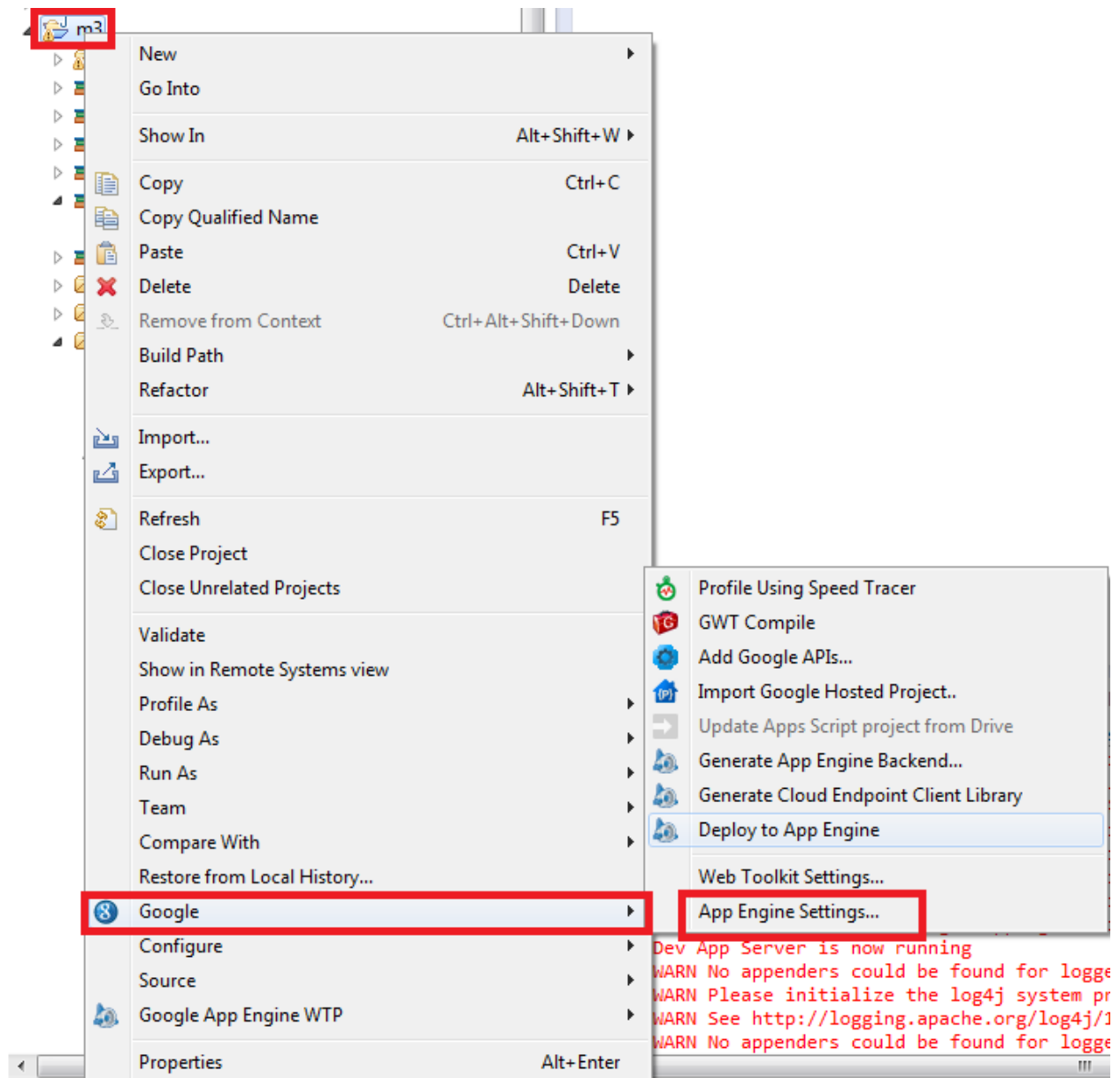


Figure 14. Google App Engine settings

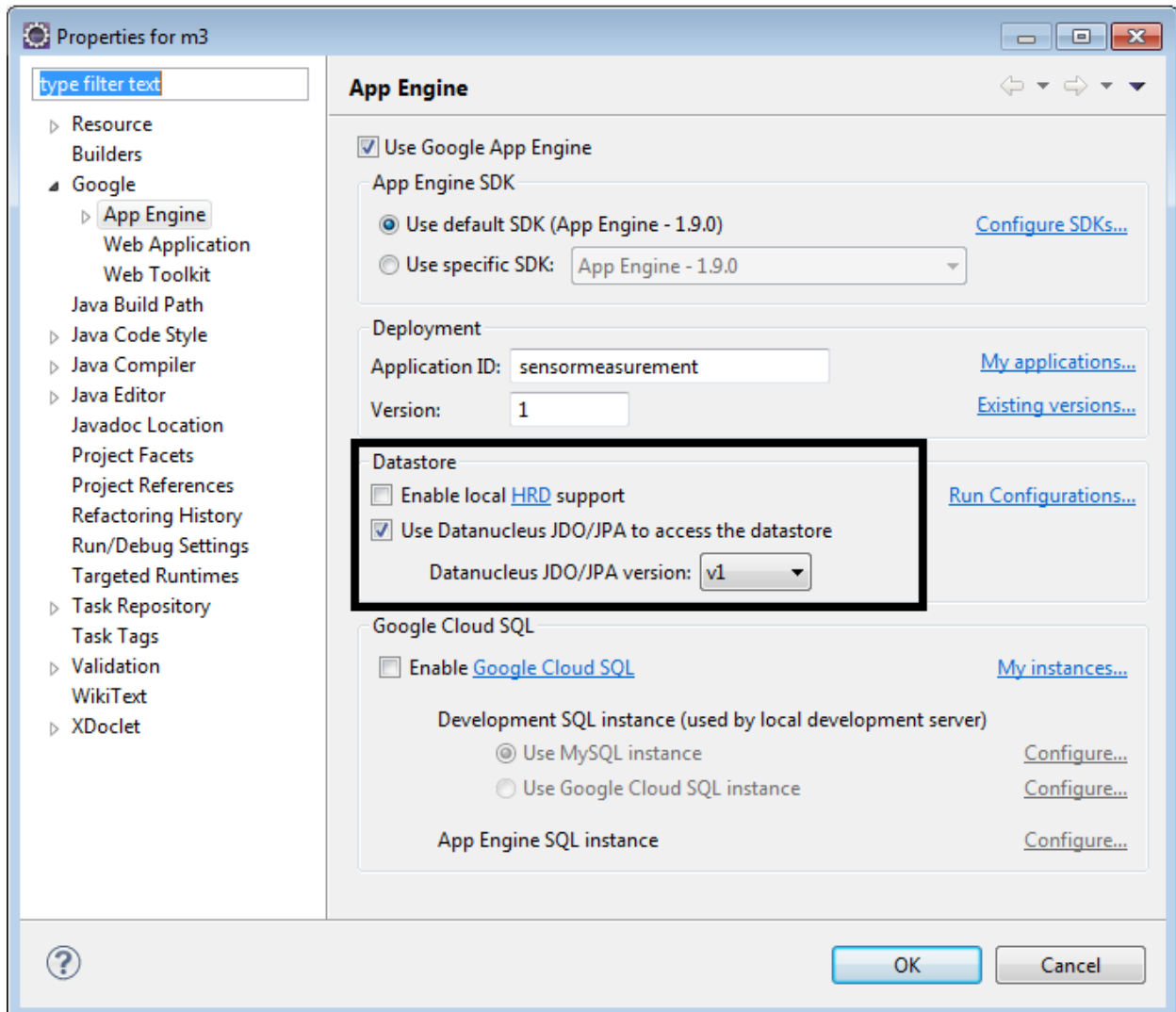
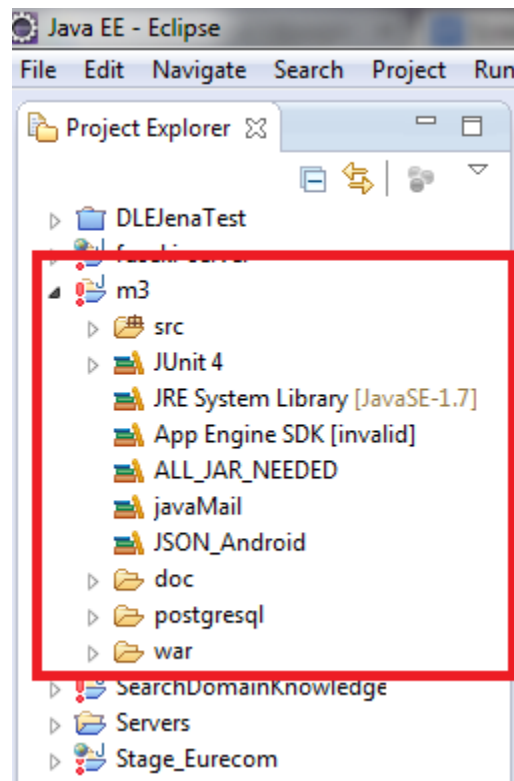


Figure 15. Check that the Datanucleus JPP/JPA is V1



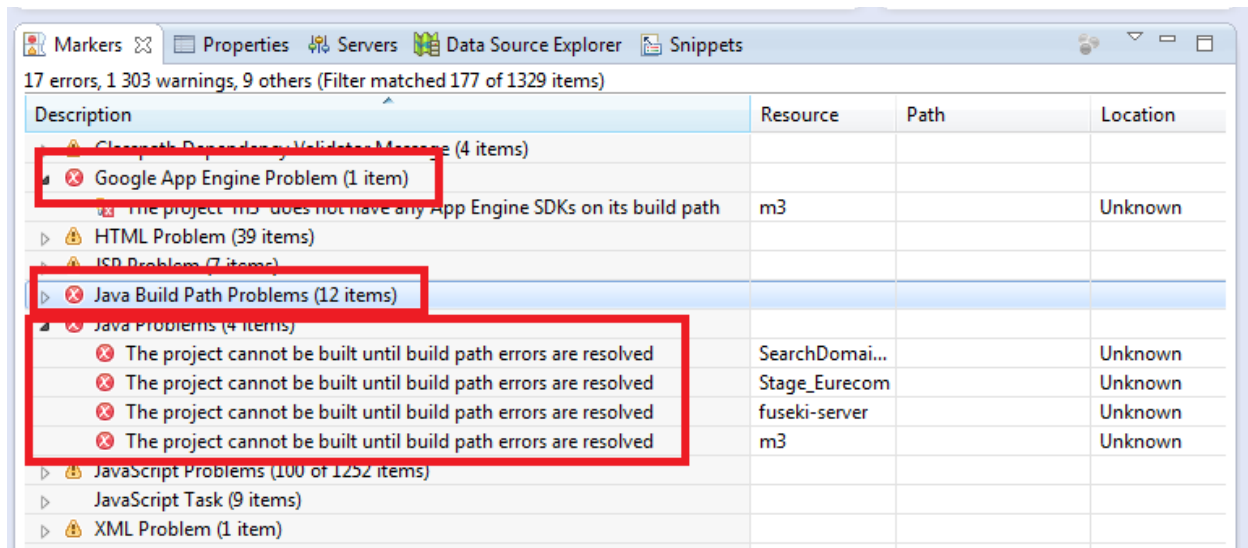
# Part III : Running the M3 project on localhost

## I. Discover the M3 project and run it



**Figure 16. Open the M3 project**

We have to fix the following issues:



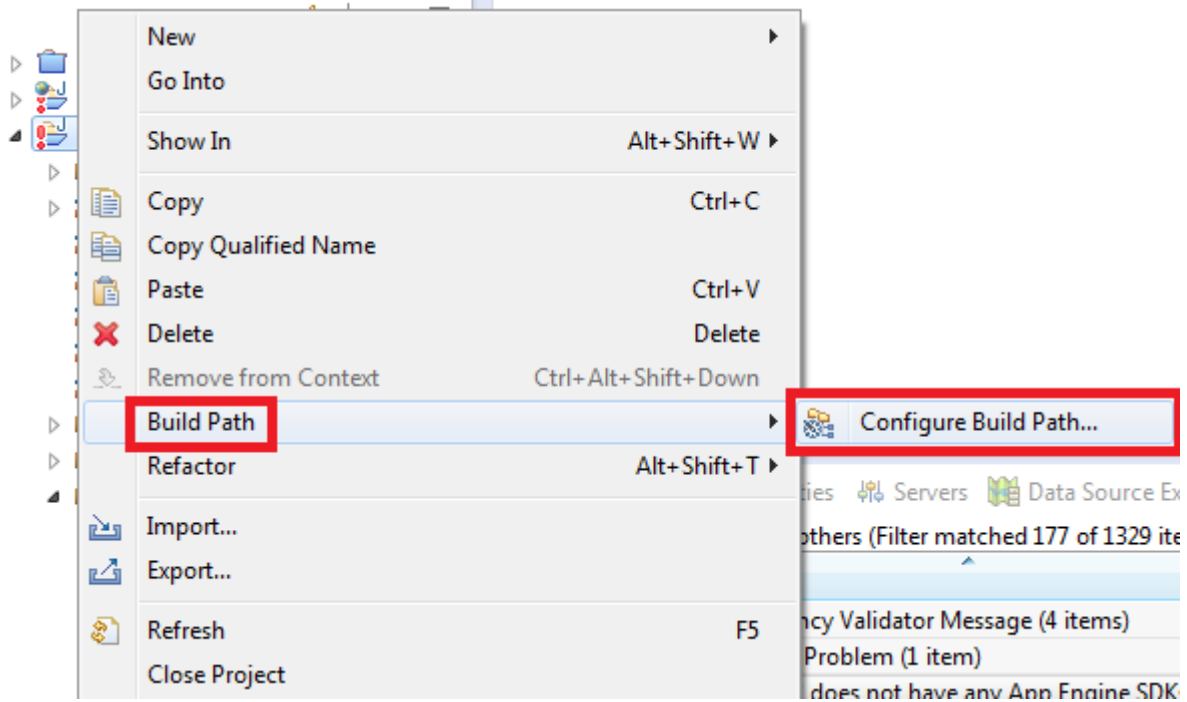
**Figure 17. Fix issues to run the M3 project**

## 1. Fixing Google App Engine Problem

Two solutions:

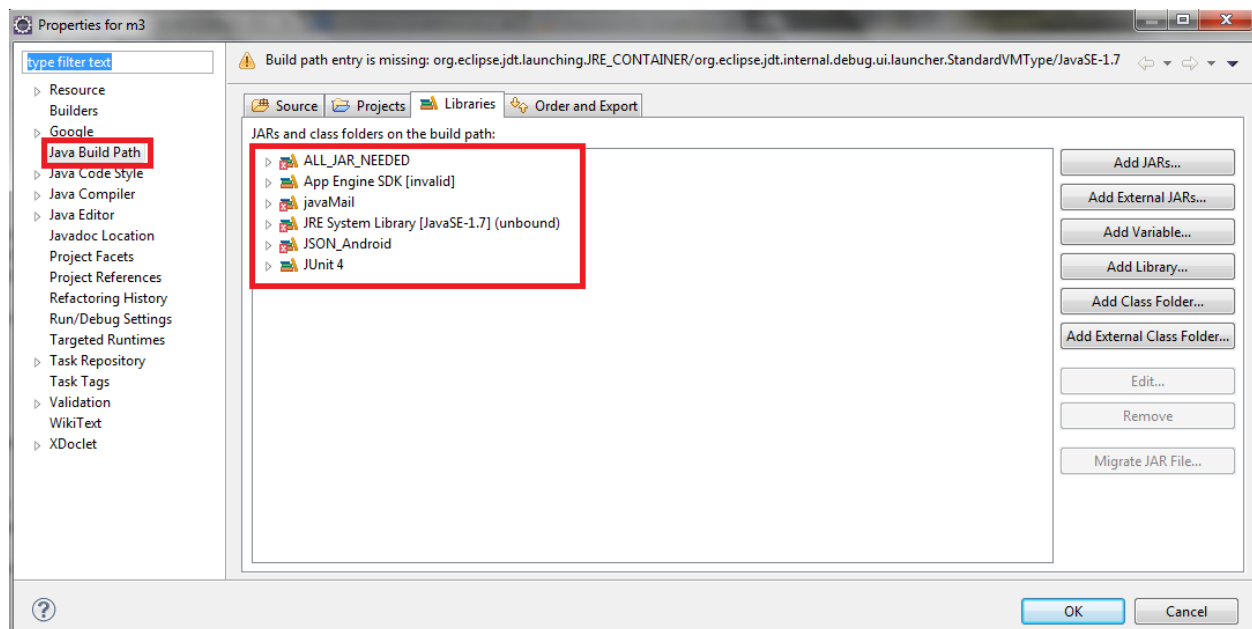
- 1) Copy/Paste the JAR directory that we provided in the M3 project
- 2) or Modify the Build Path

Modify the build path:



**Figure 18. Modify the build paths to reference the JARs required**

Modify the build path of all jar libraries:



**Figure 19. Add the JARs required**

## 2. Java Build Path Problems

Add all jars required, that are available in the JAR directory.

## II. (Optional) Install Jena inside Eclipse

In case you encounter some issues, you can follow the following tutorial:

<http://www.iandickinson.me.uk/articles/jena-eclipse-helloworld/>

## III. Run M3 on localhost

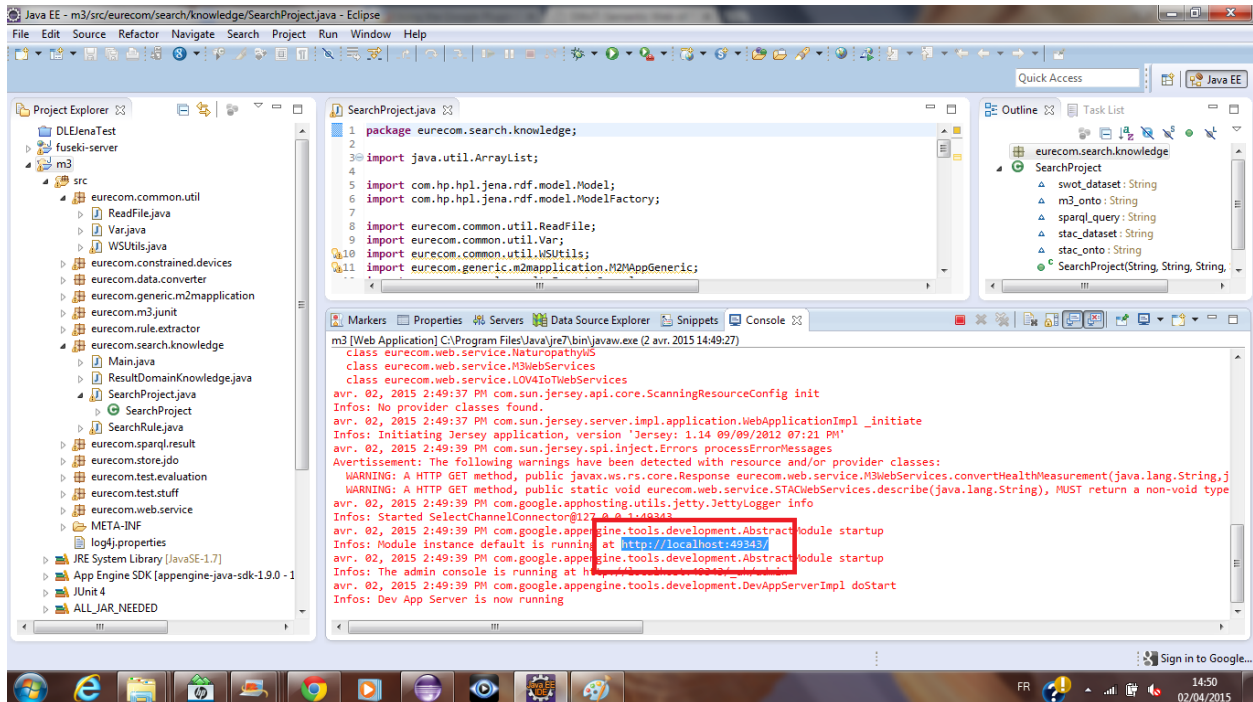
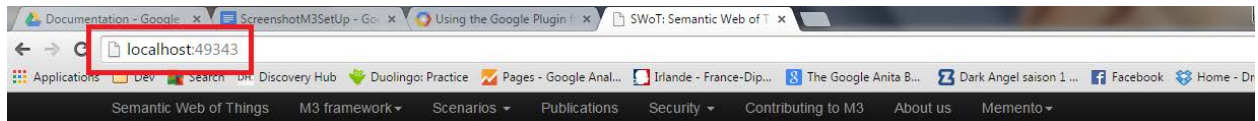


Figure 20. Get the localhost address

Copy paste the URL in red with the localhost address on the Browser (e.g., Chrome).



*Making smart discussions between things*

Machine-to-Machine Measurement (M3) is a framework to semantically annotate and easily interpret [Internet of Things \(IoT\)](#) data. M3 enables designing interoperable domain-specific or cross-domain [Semantic Web of Things \(SWoT\)](#) applications. M3 is composed of the following components:

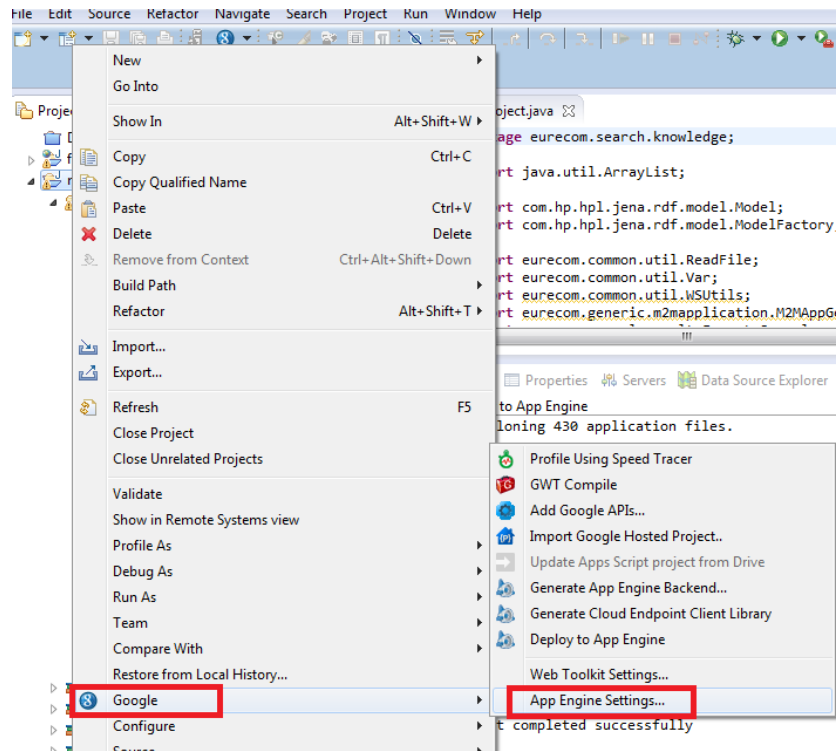


**Figure 21. M3 is running on localhost, it works!**

## Part IV : Deploying M3 on the Web (Optional)

Right click on the M3 project on Eclipse:

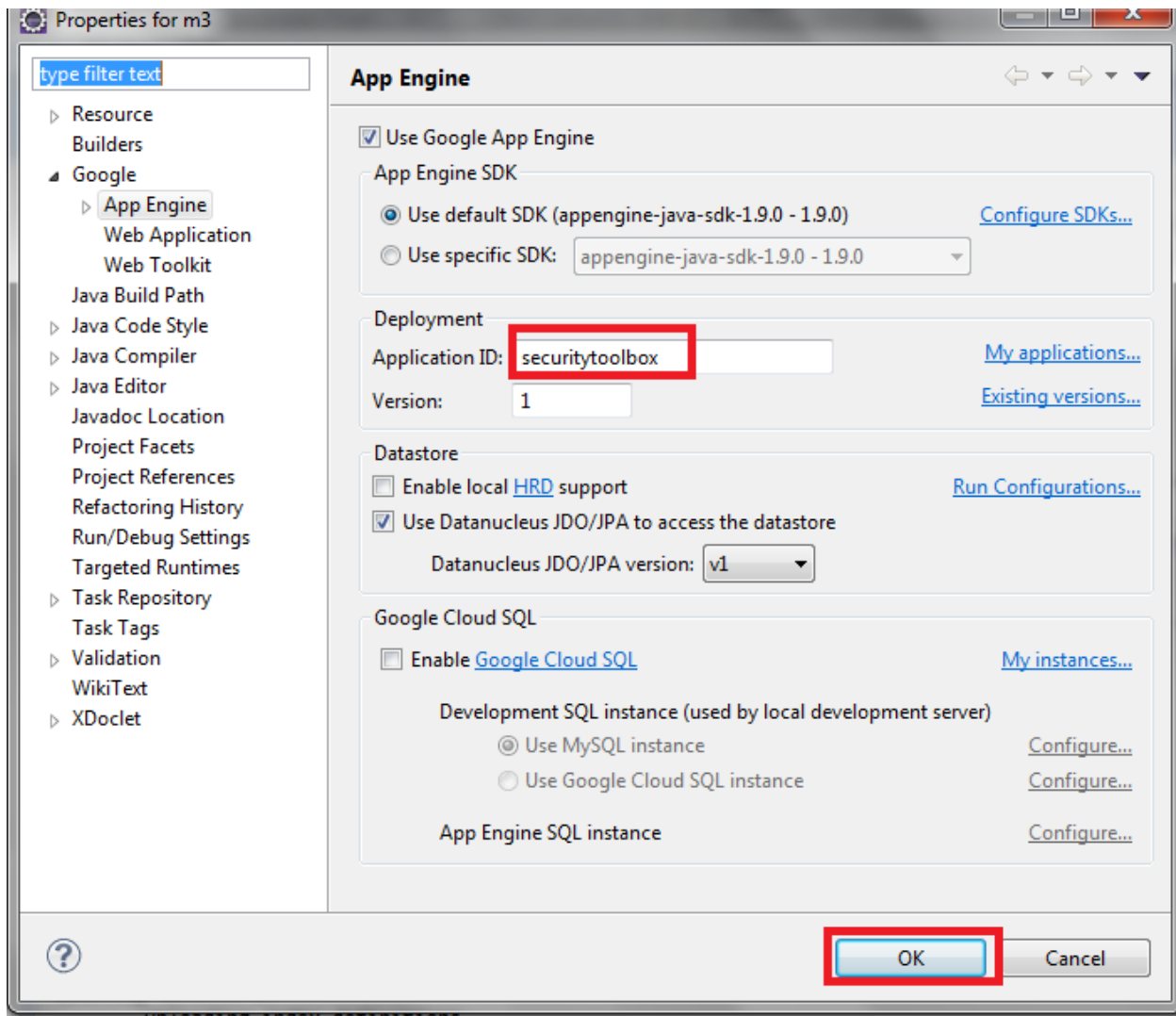
- ➔ Select Google
- ➔ Select App Engine Settings



**Figure 22. Check Google App Settings**

- ➔ You will have the following screen. In the Application ID enter **securitytoolbox**, this is the web site to try it. For this, we need to give you access rights in case you are collaborating on this project.
- ➔ Otherwise, you can create your own hostname. Please follow the documentation to deploy you app engine with Google<sup>3</sup>.
- ➔ Click OK

<sup>3</sup> [https://developers.google.com/eclipse/docs/appengine\\_deploy](https://developers.google.com/eclipse/docs/appengine_deploy)



**Figure 23. Indicate the name of the application where it will be deployed on the Web**

Then right click on M3 again:

- ➔ Select Google
- ➔ Select Deploy to App Engine

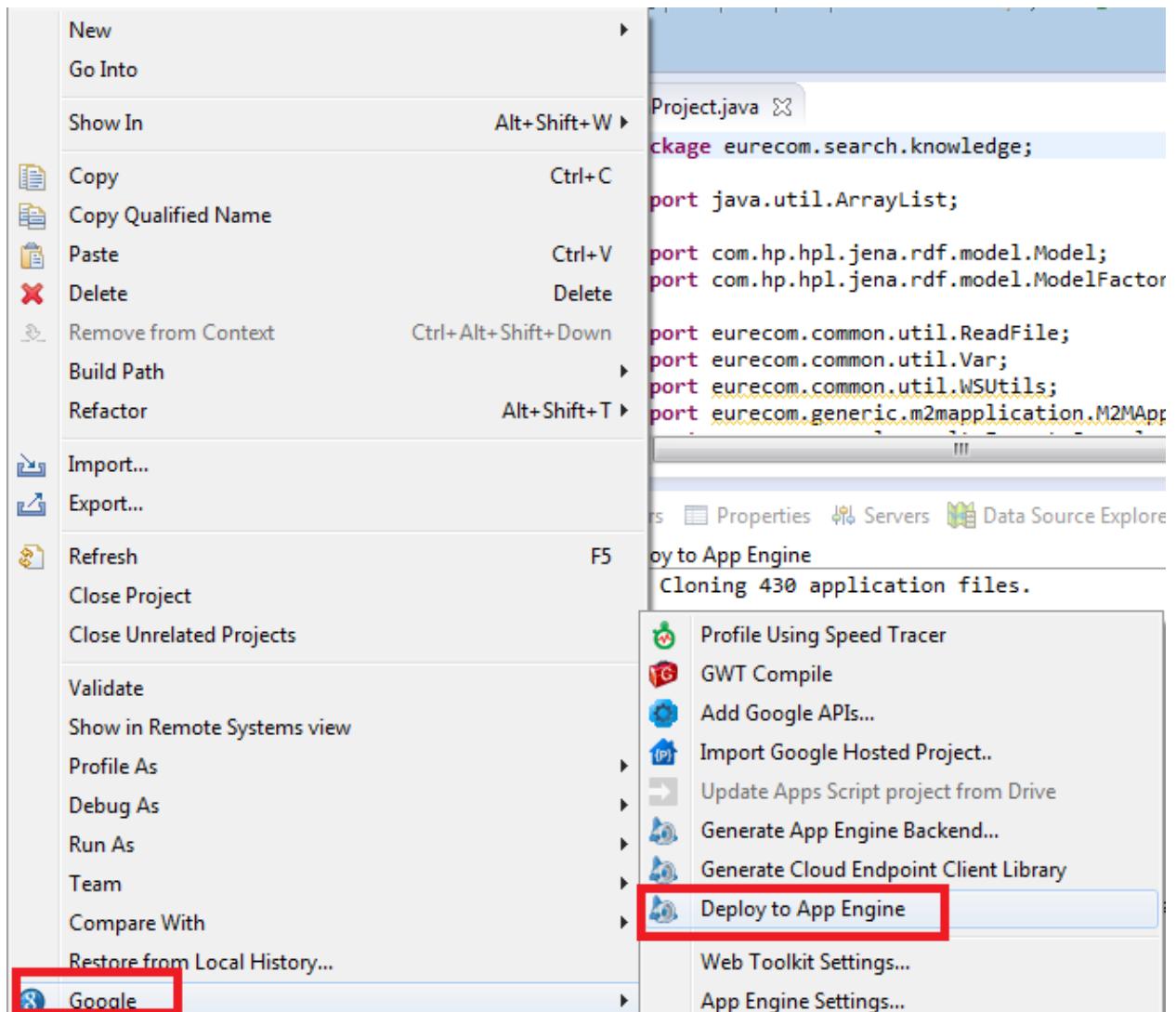


Figure 24. Deploy the M3 project on the web



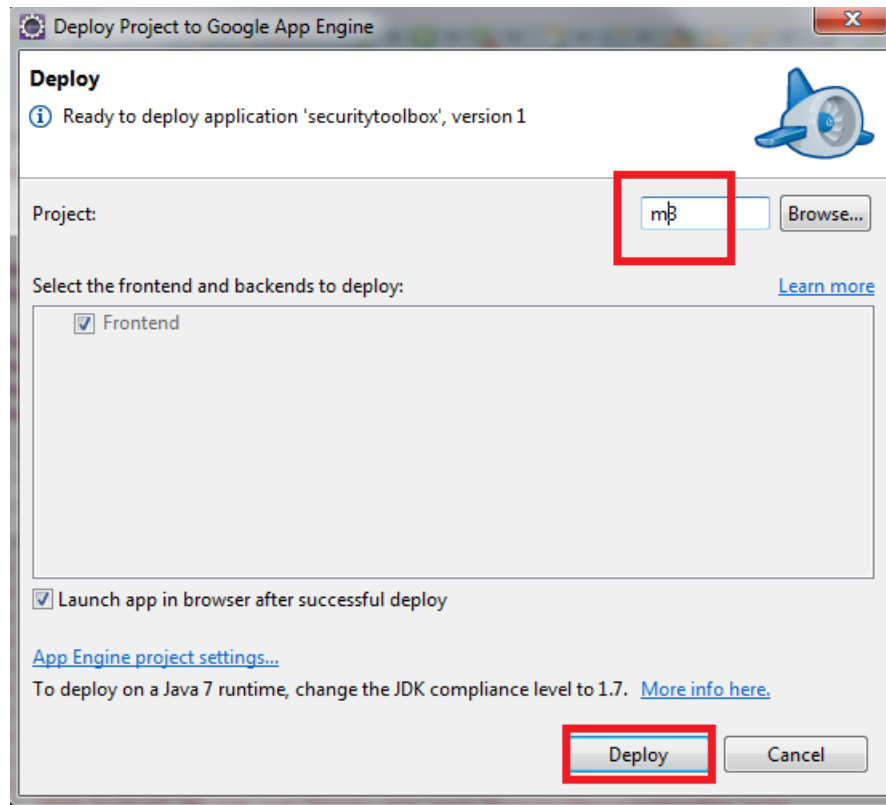


Figure 25. Confirm that you deploy the M3 project on the web

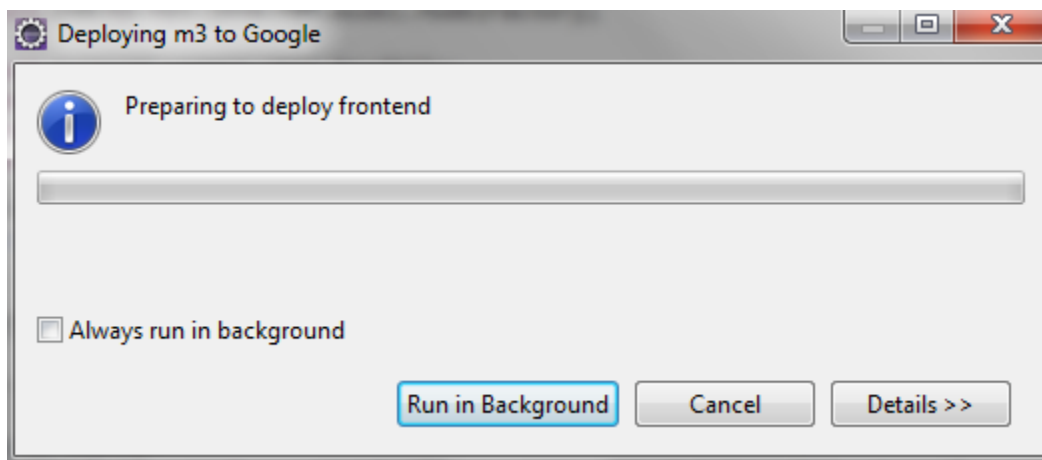


Figure 26. Wait, M3 is deploying on the web



Google will ask you to authenticate, you need to be authenticated on the project.

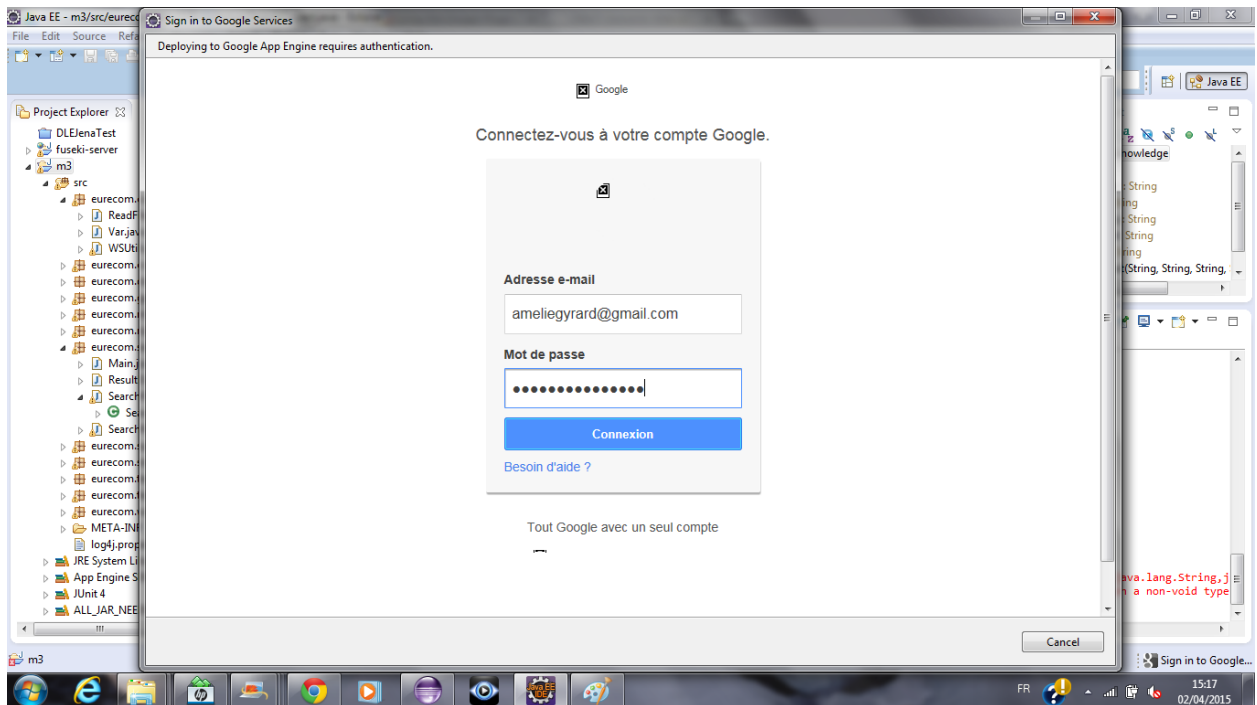


Figure 27. Authenticate yourself with Google App Engine

➔ Click on the button Accept:

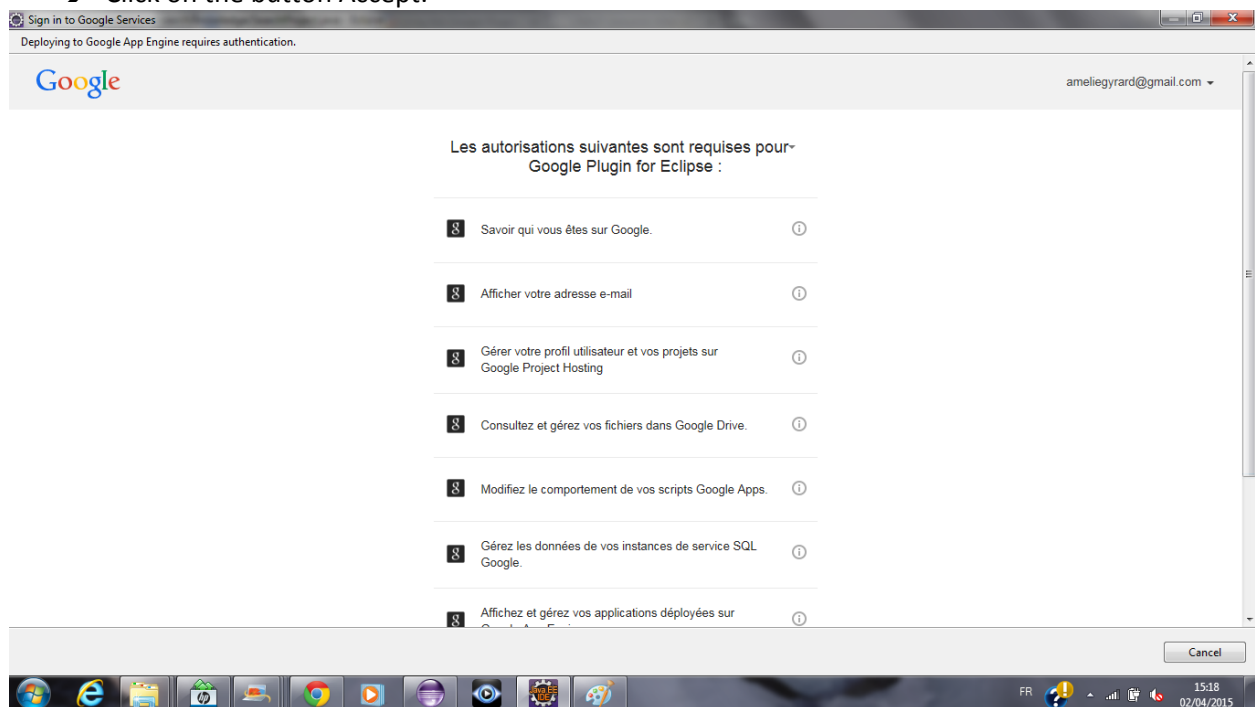


Figure 28. Accept the authentication on Google App Engine

Check that it works:



Machine-to-Machine Measurement (M3) is a framework to semantically annotate and easily interpret [Internet of Things \(IoT\)](#) data. M3 enables designing interoperable domain-specific or cross-domain [Semantic Web of Things \(SWoT\)](#) applications. M3 is composed of the following components:

				
Generating SWoT applications	Reusing domain knowledge	Interpreting IoT data	M3 Interoperable IoT Data & Domain Knowledge	Securing IoT applications

Figure 29. M3 deployed on the web

## I. Creating an application on app engine

We can deploy on the semantic-web-of-things.appspot.com web site.

You will have to find another name (check availability button)

← <https://appengine.google.com>

Useful NewsInfo SWE SSN Conf Accueil - Events Project Job SW Dev Sensor SWOT Latex dataset PaperNotAvailable

Google app engine | [My Account](#) | [Help](#) | [Sign out](#)

## My Applications

« Prev 20 1-5 of 5 Next 20 »

Application	Title	Storage Scheme	Status
<a href="#">linkedopenrules</a>	linkedopenrules	High Replication	None Deployed
<a href="#">securitytoolbox</a>	securitytoolbox	High Replication	Running
<a href="#">semantic-web-of-things</a>	sensormeasurement	High Replication	Running
<a href="#">semanticdream</a>	semanticdream	High Replication	None Deployed
<a href="#">sensormeasurement</a>	sensormeasurement	High Replication	Running
<a href="#">Create Application</a>			

You have 20 applications remaining. « Prev 20 1-5 of 5 Next 20 »

© 2015 Google | [Terms of Service](#) | [Privacy Policy](#) | [Blog](#) | [Discussion Forums](#) | [Project](#) | [Docs](#)

← <https://appengine.google.com/start/createapp>

Google app engine | ameliogyrd@gmail.com | [My Account](#) | [Help](#) | [Sign out](#)

## Create an Application

You have 20 applications remaining.

**Application Identifier:**

semantic-web-of-things .appspot.com

[Check Availability](#)

All Google account names and certain offensive or trademarked names may not be used as Application Identifiers.

You can map this application to your own domain later. [Learn more](#)

**Application Title:**

Semantic Web of Things

Displayed when users access your application.

**Authentication Options (Advanced):** [Learn more](#)

Google App Engine provides an API for authenticating your users, including Google Accounts, Google Apps, and OpenID. If you choose to use this feature for some parts of your site, you'll need to specify now what type of users can sign in to your application:

☒ **Open to all Google Accounts users (default)**

If your application uses authentication, anyone with a valid Google Account may sign in.

☐ **Restricted to the following Google Apps domain:**

e.g. foo.com

If your application uses authentication, only members of this Google Apps domain may sign in. If your organization uses Google Apps, use this option to create an application (e.g. an HR tracking tool) that is only accessible to accounts on your Google Apps domain. This option cannot be changed once it has been set.

☐ **(Experimental) Open to all users with an OpenID Provider**

If your application uses authentication, anyone who has an account with an OpenID Provider may sign in.

[Create Application](#)

[Cancel](#)

## II. Error App engine when deploying

Error when you want to deploy your application on app engine:

Another transaction by user ... is already in progress for this app and major version

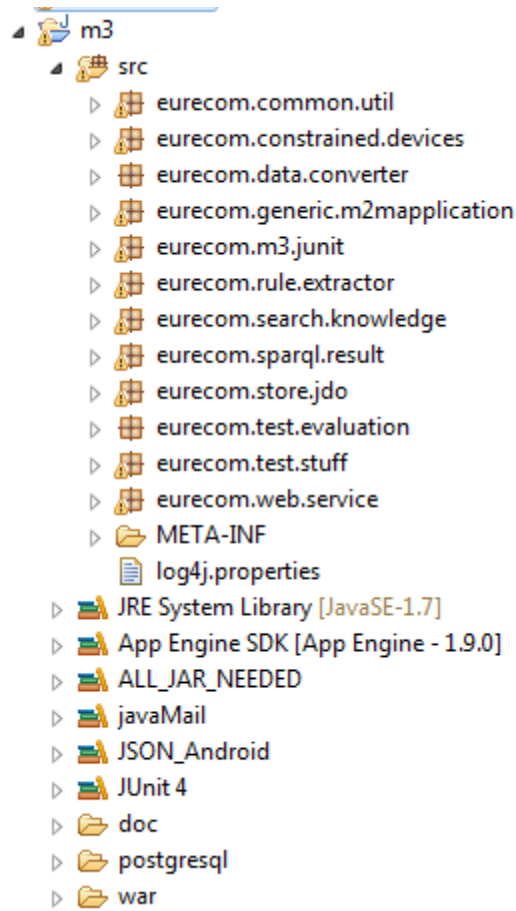
```
E:\workspace\m3> "E:\eclipse-jee-kepler-SR1-win32-x86_64\eclipse\plugins\com.google.appengine.eclipse.sdkbundle_1.9.0\appengine-java-sdk-1.9.0\bin\appcfg" rollback war
Reading application configuration data...
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.AppEngineWebXmlReader readAppEngineWebXml
INFO: Successfully processed war\WEB-INF\appengine-web.xml
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.AbstractConfigXmlReader readConfigXml
INFO: Successfully processed war\WEB-INF\web.xml
Sep 29, 2014 9:23:57 AM com.google.apphosting.utils.config.IndexesXmlReader readConfigXml
INFO: Successfully processed war\WEB-INF\appengine-generated\datastore-indexes-auto.xml

Beginning interaction for module default...
0% Rolling back the update.
Email: ameliiegyrard@gmail.com
Password for ameliiegyrard@gmail.com:
Success.
Cleaning up temporary files for module default...

E:\workspace\m3> "E:\eclipse-jee-kepler-SR1-win32-x86_64\eclipse\plugins\com.google.appengine.eclipse.sdkbundle_1.9.0\appengine-java-sdk-1.9.0\bin\appcfg" rollback war
wa./appcfg.sh rollback /home/workspace/vchat/war
```

## Part V : Understanding the M3 project

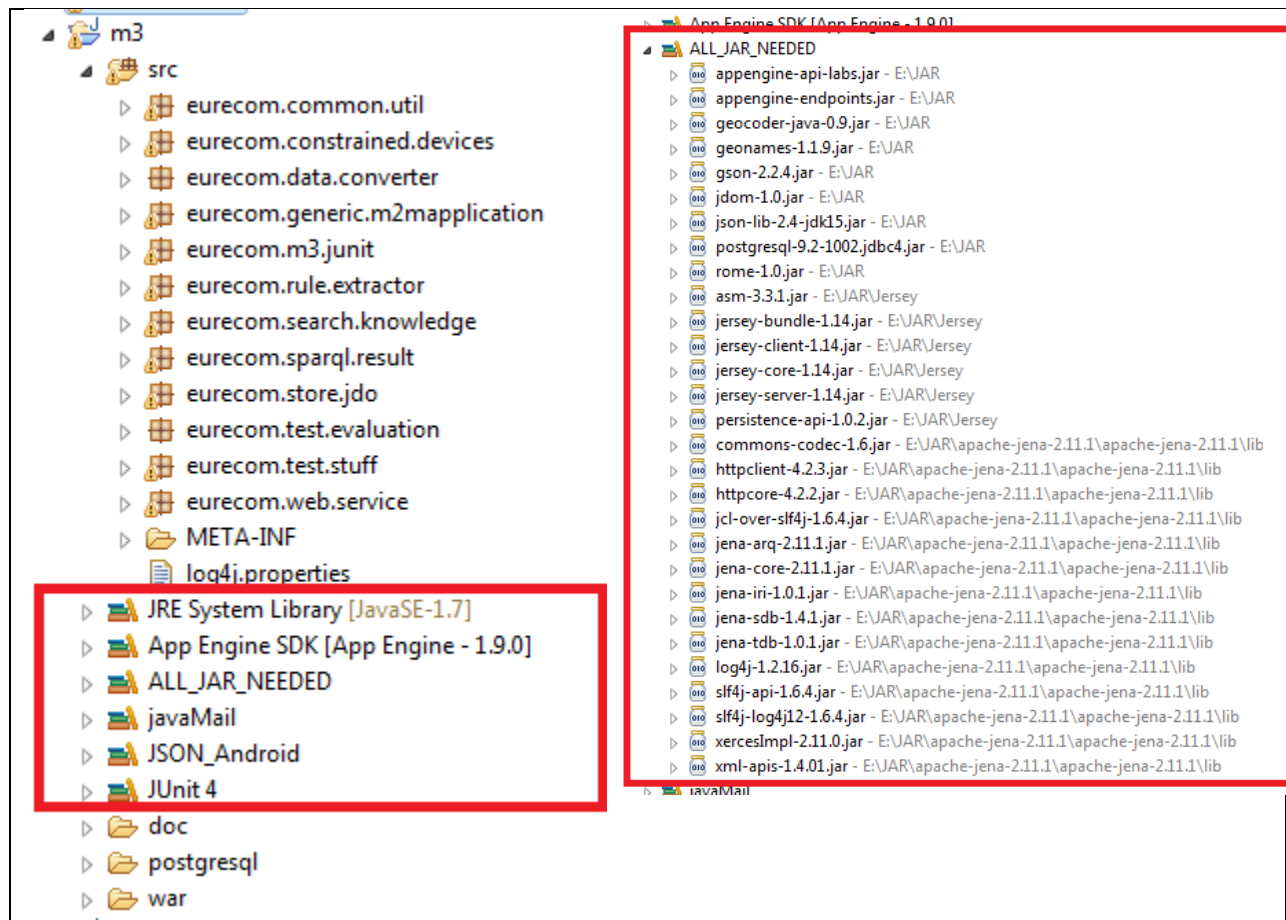
### I. M3 framework overview



- WAR/CSS
- WAR/dataset
- WAR/html
- WAR/images
- WAR/javascript
- WAR/ont: all ontologies
- WAR/publication
- WAR/RULES: ruleM3NewType.txt is used in the M3 Converter to add the right sensor, measurement, unit or domain type
  - LinkedOpenRules\*.txt: M3 rules implemented according to the Jena syntax compliant with the Jena reasoned, they are classified by domains (e.g., weather,, environment, health, Home)
  - ruleM3NewType.txt used by the M3 converter to convert SenML<sup>4</sup> sensor data in RDF m3 sensor data
  - WAR/RULES/OTHER: files are from domain experts and just shared in the ontology.html web page
- WAR/SPARQL all SPARQL queries
- WAR/WEB-INF

<sup>4</sup> <http://www.ietf.org/archive/id/draft-jennings-senml-10.txt>

## II. Libraries (Jars) required



To run the M3 project, you will need the following libraries:

- **JRE System library**, should be the Java 1.7
- **App engine SDK** [App engine - 1.9.0] used to host our project online.
- **ALL\_JAR\_NEEDED** is comprised of
  - **Jersey 1.14** to develop RESTful web services in JAVA. These web services are then called in HTML web pages through AJAX and Javascript.
  - **Jena 2.11**, a framework uses to build semantic web applications. In the M3 project, more specifically, semantic web of things applications. We use the jena reasoner, the sparql engine, etc.
  - **Geonames**<sup>5</sup> used in the restaurant scenario. From a location (longitude and latitude), we can get additional information such as city, country, etc.
- **javaMail** to send emails, used for the LOV4IoT tool
- **JSON\_Android** used to convert sensor data, semantically annotate them and return it in JSON
- **JUnit4** to develop java unit tests

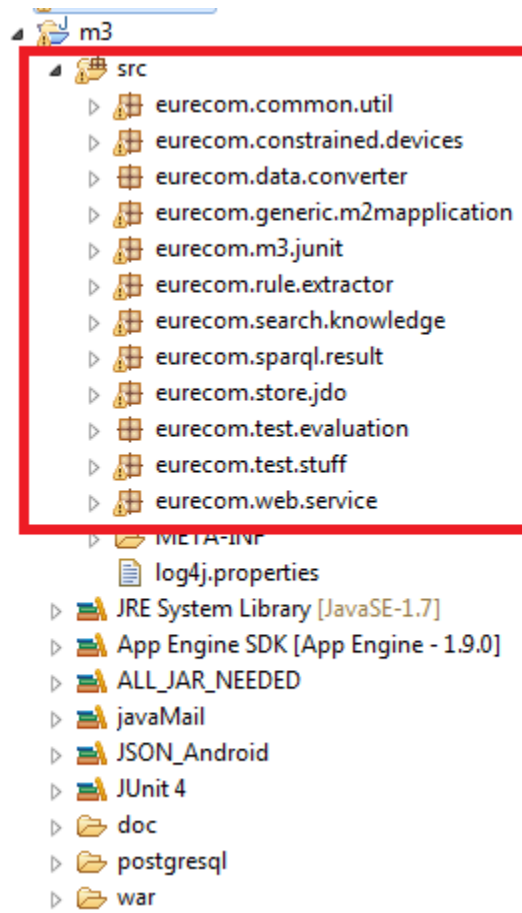
<sup>5</sup> <http://www.geonames.org/>

### III. Descriptions of packages

The M3 project is comprised of 12 packages:

- **eurecom.common.util**: some useful functions such as read a file, query a web service, etc.
- **eurecom.constrained.devices**: code tested to adapt the M3 framework to Android powered devices. For instance, we used JSON instead of XML, etc.
- **eurecom.data.converter**: to semantically annotate SenML data with the M3 nomenclature
- **eurecom.generic.m2mapplication**: to try to find a way to develop generic applications using the same reasoning process.
- **eurecom.m3.junit**: some Junit tests to test the M3 framework. Otherwise, the M3 framework has been tested manually with demos.
- **eurecom.rule.extractor** (work in progress): some tests to extract rules from the ontology catalogue called LOV4IoT.
- **eurecom.search.knowledge**: used to look for all projects using a specific sensor
- **eurecom.sparql.result**: to execute the SPARQL query and get the results in different formats.
- **eurecom.store.jdo**: to store data in the Google datastore
- **eurecom.test.evaluation**: used to evaluate the software performances
- **eurecom.test.stuff**: to test new technologies, new tools to integrate, etc.
- **eurecom.web.service**: all web services implemented in the M3 framework





## 1. eurecom.common.util

Var.java: Java class where all the static variable are set. Add to this there are sorted by what they are used for. Furthermore, all the URL are saved here.

WSUtils.java: Class behaviour to read ontology, data measurement, and SPARQL results. Moreover it implements the behaviour to execute a web service remotely. This class contains noticeable and useful functions:

- `convertXmlToJson(<string>)` and `convertJsonToXml(<string>)` : do like their name, before to use this object there is a pre-processing, the JSON (or the Xml) have to be send to the function in a String type.
- `queryWebService(<string> url, enum)` : execute remotely a web service, the first argument is the URL of the web service, the second is an enumeration (JSON or Xml), it depends if you want to use an Xml or JSON web service. Return the result of the request (<string>)
- `readXMLFile(<string> filename)` : read an xml file and transform the data in to RDF data, return the data object of type Measurements (see package `eurecom.data.convert`).
- `readFile(optional <model> (from jena library), <string> filename)` : if model is send to the function, read ontologies otherwise read a file).

## 2. eurecom.constrained.devices

To query sensor data provided by the raspberry pi

Java-json.jar used in eurecom.constrained.devices package

java json android parser for sensor data provided by the raspberry pi

## 3. eurecom.data.converter

- Domain.java: Class which transforms XML data to RDF data. Have a public ArrayList containing the measurements, getter and setter of the nameZone.
- ConvertSensorDataToM3.java: Behaviour of the conversion of java object (Measurements) into RDF data and of the other way. Can create also some instance object into RDF (type of object: Measurement, FeatureOfInterest, and Sensor).
- Measurement.java && Measurements.java: Objects measurements, describe the object that is used to transform Xml data into RDF data. Mostly composed by variables and their getter/setter.
- *Note:* A Domain has an ArrayList of Measurements (with an S) and Measurements has an ArrayList of Measurement. Where Measurement is just a data (30 °C for instance), Measurements is a bunch of data (all the data from one sensors) and FeatureOfInterest could be the domain (Temperature).

## 4. eurecom.generic.m2mapplication

Coordinates.java: Object to describe a coordinate on earth. Composed by a longitude and a latitude.

LOVClient.java: One function executeSparql(<string> sparql) return InputStream.

VariableSparql.java: Object that describe variable used in a sparql request.

M2MAppGeneric.java: Main behaviour of the application. Can load the dataset of the ontologies provide by the application. Execute the SPARQL queries and linked open rules.

Main function implemented:

- load<ontologyName>Dataset(): add the ontology in the model. There is one load for each kind of ontology.
- executeLinkedOpenRulesAndSparqlQuery(<string> SparqlQuery, ArrayList of variables of the query): prepare and call the function that can execute the SPARQL query.

M2MAppResto.java / M2MAppNaturopathy.java / M2MAppRecipe.java / M2MAppStac.java: These are an extension of M2MAppGeneric which have the same behaviour but with additional specific function that execute almost implemented SPARQL queries. A future goal is to make a real generic class which will make these classes depreciated.

Place.java: Same as Coordinates, object that describe a place in the world. However there is no getter or setter.

WlboxApi.java: Class that allows to get the hierarchy classes of a certain model. Could be depreciated.

## 5. eurecom.sparql.result

ExecuteSparqlGeneric.java: Java class that implements all type of SPARQL queries (delete or find some labels, replace/update). Concerning queries, you can specify (using a function) how to store the value of the query.

ExecuteSparql.java: Same as ExecuteSparqlGeneric.java but it uses to define one and only one SPARQL request, the object define must not execute other SPARQL request.

ExecuteSparqlHealth.java / ExecuteSparqlRecipe.java / ExecuteSparqlRestaurant.java /

ExecuteSparqlRecipeNaturopathy.java: Implement the SPARQL request function associated to an ontology, should be deleted and use a more generic function in ExecuteSparqlGeneric.java instead.

SparqlResultGeneric.java, SparqlResultRecipe.java, SparqlResultRecipeNaturopathy.java,

SparqlResultRestaurant.java: Describe the object result of a query, composed of variables, getters and setters.

VariableSparql.java: Object that describe variable used in a SPARQL request.

## 6. eurecom.store.jdo

GenericJDO: This class manipulates the measurements object get, create or delete it.

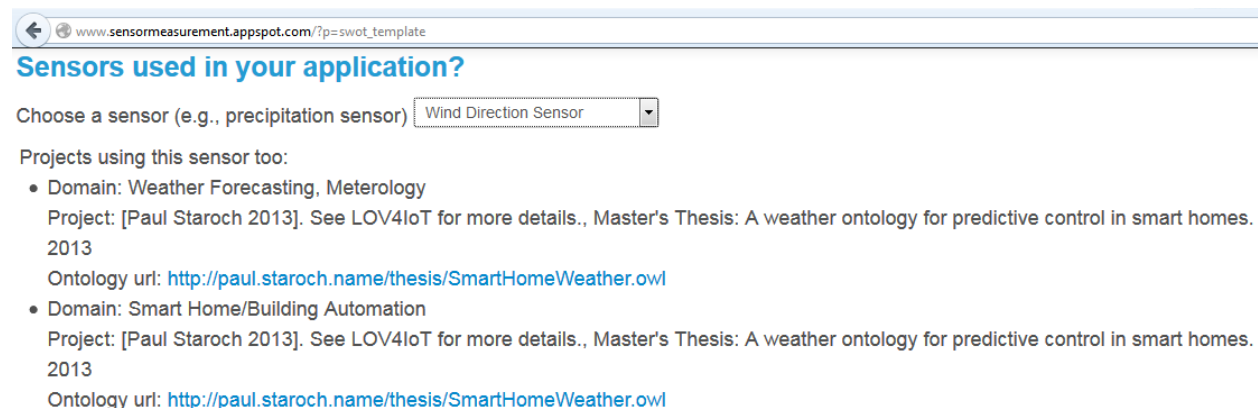
Util.java: Composed of useful function to manipulate entities and especially to get JSON strings.

JDO Java Data object, datastore compatible with google to store data (e.g., M3 rdf sensor data)

## 7. eurecom.search.knowledge

The code is used in this web page<sup>6</sup> through web services.

According to a specific sensor given in the drop-down list, we can retrieve all projects using this sensor.



www.sensormeasurement.appspot.com/?p=swot\_template

### Sensors used in your application?

Choose a sensor (e.g., precipitation sensor)

Projects using this sensor too:

- Domain: Weather Forecasting, Meterology  
Project: [Paul Staroch 2013]. See LOV4IoT for more details., Master's Thesis: A weather ontology for predictive control in smart homes. 2013  
Ontology url: <http://paul.staroch.name/thesis/SmartHomeWeather.owl>
- Domain: Smart Home/Building Automation  
Project: [Paul Staroch 2013]. See LOV4IoT for more details., Master's Thesis: A weather ontology for predictive control in smart homes. 2013  
Ontology url: <http://paul.staroch.name/thesis/SmartHomeWeather.owl>

## 8. eurecom.web.service

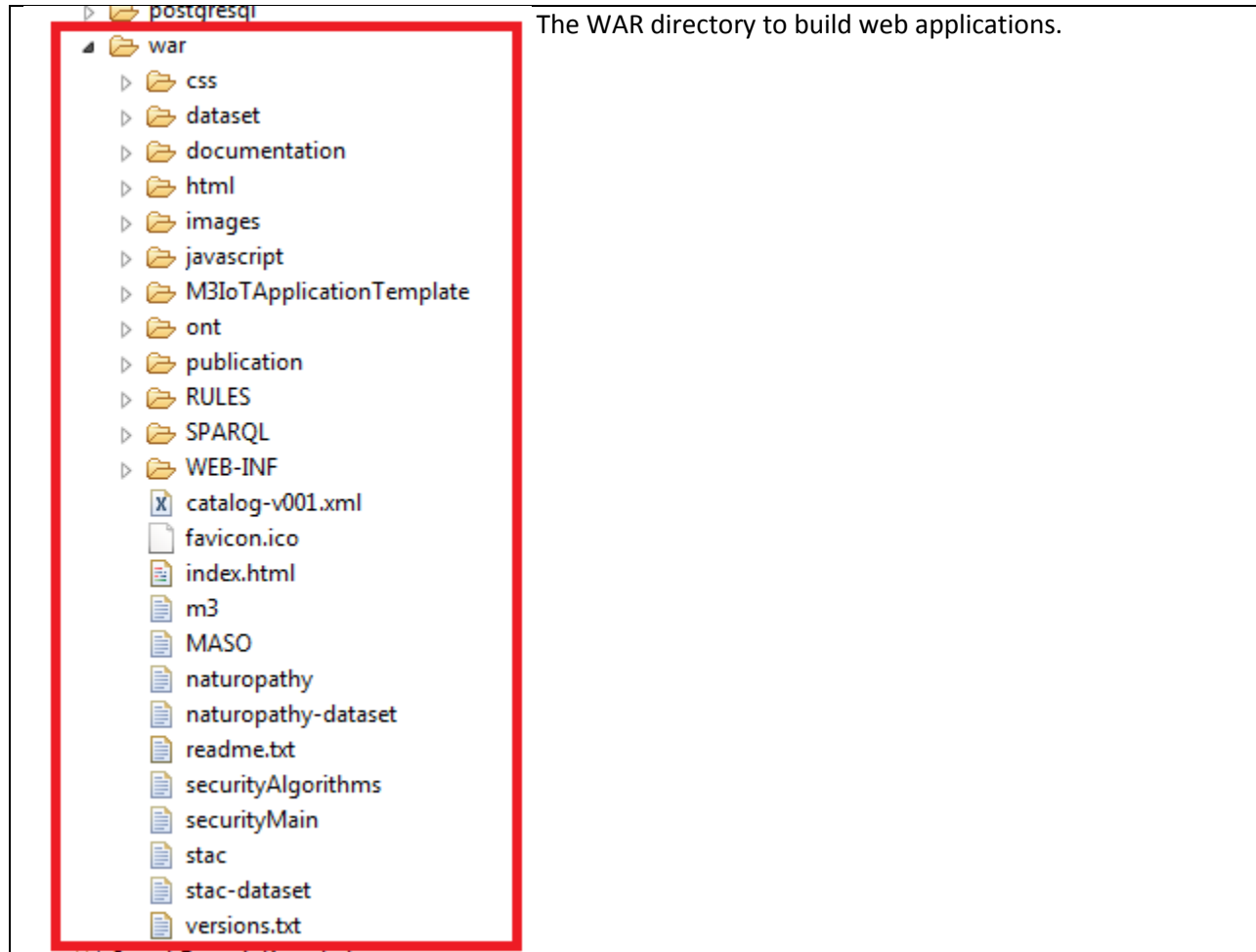
All the classes here are Web Services. There is one class for each type of ontology. Before every method, there are some important parameters:

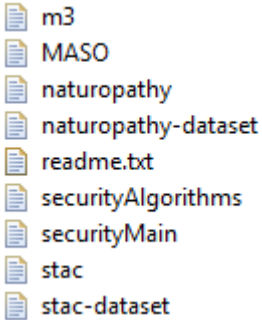


- @GET: Indicate the kind of protocol used (here is Get by opposition of POST for instance).
- @Path("String"): This is the URL where the method is called. This URL is relative.

<sup>6</sup> [http://www.sensormeasurement.appspot.com/?p=swot\\_template](http://www.sensormeasurement.appspot.com/?p=swot_template)

- @Consumes(MediaType): What kind of string this method takes.
- @Produces(MediaType): What kind of string this methods produces (It is often a JSON string or Xml).

## IV. The WAR directory



	 <p><b>At the beginning of the project, we had the following ontologies and datasets directly in the WAR file. We keep these ontologies and datasets since they have been referenced in research articles. We do not remove them to avoid 'error 404: page not found'.</b></p> <p><b>But now, all modifications of ontologies and datasets are in the corresponding directory.</b></p>  <p><b>TO DO: A solution would be to use the PURL tool<sup>7</sup> to avoid such issues!</b></p>
---	---

## 1. WAR/CSS

All CSS file for the web application user interface.

## 2. WAR/dataset

All datasets such as sensor data, sensor data semantically annotated with M3, domain knowledge bases, etc.

## 3. WAR/documentation

All documentation to understand the project or use the tools.

## 4. WAR/html

All HTML web pages used for the user interface.

## 5. WAR/images

All images used in the projects.

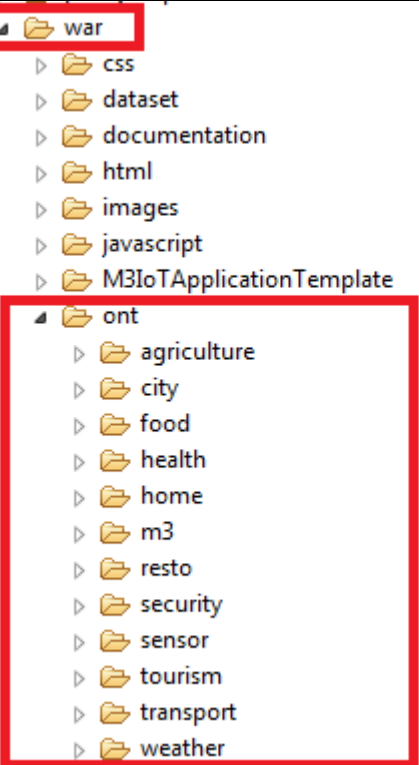
## 6. WAR/javascript

The HTML web pages will send AJAX queries to web services and get the results.  
The result is parsed in JavaScript and displayed on HTML web pages.

---

<sup>7</sup> <https://purl.oclc.org/docs/index.html>.

## 7. WAR/ont

 <ul style="list-style-type: none"><li>war<ul style="list-style-type: none"><li>css</li><li>dataset</li><li>documentation</li><li>html</li><li>images</li><li>javascript</li><li>M3IoTApplicationTemplate</li><li>ont<ul style="list-style-type: none"><li>agriculture</li><li>city</li><li>food</li><li>health</li><li>home</li><li>m3</li><li>resto</li><li>security</li><li>sensor</li><li>tourism</li><li>transport</li><li>weather</li></ul></li></ul></li></ul>	<p>All new ontologies should be in this directory. They are classified by domains.</p> <p>Some old ontologies are still in WAR/WEB-INF/ontologies. The less interesting ones, that we did not move to the new directory.</p> <p>In WAR/ont/m3: you will find the M3 interoperable domain ontologies.</p>
---	--

## 8. WAR/publication

All publications related to this thesis to understand better the M3 project. You will find slides, research articles, participation to standards, etc.

## 9. WAR/RULES

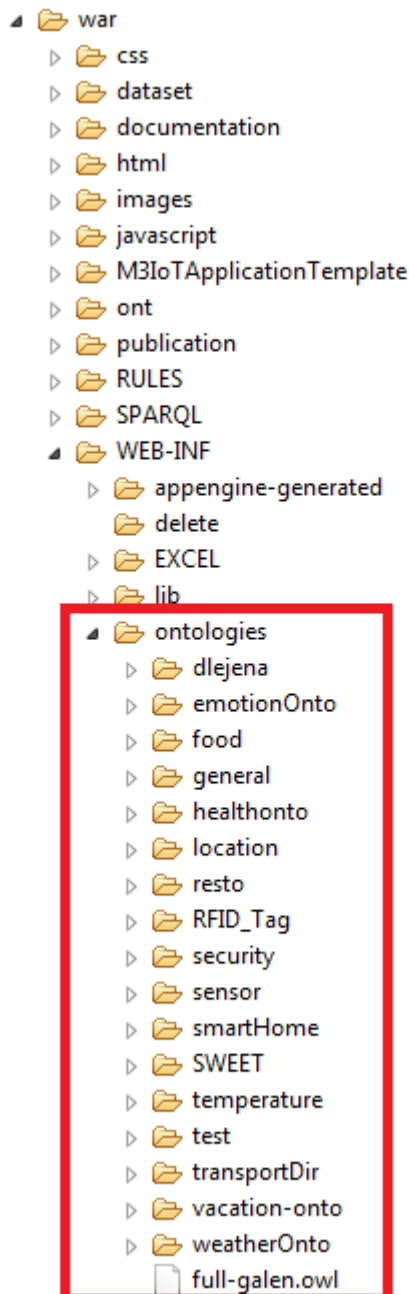
You will find in this directory all the interoperable rules that we designed.

This is the Sensor-based Linked Open Rules (S-LOR) tool. The SWoT generator has predefined-templates to build semantic-based IoT application. The templates will referenced these pre-defined set of rules classified by domains.

## 10. WAR/SPARQL

You will find in this directory all SPARQL queries.

## 11. WAR/WEB-INF/ontologies



Why do we have several directeroy related to ontologies?

Be careful: Ontologies and datasets duplicated  
We are agree this is a total mess! Some ontologies required to be shared online, but if we move all ontologies and datasets in the same place, we will have issues with path and namespace already defined in previous ontologies and referenced in the LOV4IoT dataset!

WAR/ont: we added this directory to host the ontology that we found online

WEB-INF/ontologies/: old repository with all ontologies that we found but not necessarily use them.



**Solve this issue with PURL<sup>8</sup> or URL redirection?**  
If if we migrate the ontologies in the other directory, we will have error page not found in some web

pages.

**TO DO:** delete WEB-INF/ ontology and merge it with WAR/ontologies and WAR/datasets.

**BIG ISSUE:** change namespaces everywhere (ontologies, datasets, rules, sparql) to be accessible online (URI deferencable)

**URL** already used in publications links.

<sup>8</sup> <https://purl.oclc.org/docs/index.html>

# Part VI : Understanding M3 web services

There is also the documentation to use the web services if required<sup>9</sup>.

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<sup>10</sup> implementation.

	All web services names ended by WS in Java class
---	--

## 1. APIJsonWS Java class (deprecated)

The web services provided by this Java class were focused on returning the result in JSON.

After, I found the solution to return the result either as JSON or as XML by adding:

```
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public static Response nameFunction( @QueryParam(value = "format") String format) {
    if (format.equals("xml")){
        resultSparql = req.selectResultAsXML(var);
    }
    else if (format.equals("json")){
        resultSparql = req.selectResultAsJson(var);
    }
    return Response
        .ok(resultSparql)
        .header(HttpHeaders.CONTENT_TYPE, "json".equals(format) ? MediaType.APPLICATION_JSON : MediaType.APPLICATION_XML)
        .build();
}
```

Figure 30. Code example to return the result either as JSON or XML

## 2. LOV4IoTWS Java class

All web services related to the Linked Open Vocabularies for Internet of Things (LOV4IoT) dataset<sup>11</sup> to automatically count the number of ontologies in this dataset (e.g., by domains, by ontology status, etc.):

<sup>9</sup> <http://www.sensormeasurement.appspot.com/documentation/M3APIDocumentation.pdf>

<sup>10</sup> <https://jersey.java.net/>

<sup>11</sup> <http://www.sensormeasurement.appspot.com/?p=ontologies>



- lov4iot/totalOnto/ which executes a SPARQL query to count the total number of ontology-based project referenced in the LOV4IoT RDF dataset.  
E.g., <http://sensormeasurement.appspot.com/lov4iot/totalOnto/>
- /lov4iot/ontoStatus/{status} which executes a SPARQL query to count the different status of ontologies
  - Status can be: Online, Confidential, OngoingProcessOnline, WaitForAnswer, Online, OnlineLOV, AlreadyLOV
 E.g., <http://sensormeasurement.appspot.com/lov4iot/ontoStatus/?status=Online>
- /lov4iot/nbOntoDomain/{domain} which executes a SPARQL query to count the different ontologies in all domains
  - Domain can be: BuildingAutomation, Weather, Emotion, Agriculture, Health, Tourism, Transportation, City, EnergyFOI, Environment, TrackingFood, Activity, Fire, TrackingCD, TrackingDVD, SensorNetworks, IoT, Security

E.g.,

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

- /lov4iot/sendEmail/{recipient,paper} which sends email to encourage people to share their domain knowledge (ontologies, datasets, and rules)

## Linked Open Vocabularies for Internet of Things (LOV4IoT) Reusing domain knowledge expertise

LO4IoT web service:  
/lov4iot/totalOnto/

Before to reinvent the wheel, maybe you can reuse the following existing ontologies with minor modifications.

We have referenced in this web page **269** ontology-based projects relevant for IoT. These domain ontologies, although very interesting, are not referenced on the [Linked Open Vocabularies \(LOV\)](#) since the [Semantic Web best practices](#) are not followed.

LO4IoT web service:  
/lov4iot/nbOntoDomain/

										
Nb onto: <b>45</b>	Nb onto: <b>8</b>	Nb onto: <b>10</b>	Nb onto: <b>30</b>	Nb onto: <b>28</b>	Nb onto: <b>17</b>	Nb onto: <b>14</b>	Nb onto: <b>3</b>	Nb onto: <b>17</b>		
										
Nb onto: <b>52</b>	Nb onto: <b>29</b>	Nb onto: <b>6</b>	Nb onto: <b>6</b>	Nb onto: <b>8</b>	Nb onto: <b>7</b>	Nb onto: <b>27</b>				

LO4IoT web service:  
/lov4iot/ontoStatus/

Ontologies have been colored as following:

The ontology will never be available (lost, confidential, etc.)	We are waiting the response from the authors if they can publish the ontology online	Authors are publishing online the ontology (ongoing work)	Ontology published online but the Semantic Web best practices are not followed.	Ontology published online and referenced by LOV since Semantic Web best practices are adopted! :-)))	Already on LOV - No email sent
Nb onto: <b>25</b>	Nb onto: <b>113</b>	Nb onto: <b>24</b>	Nb onto: <b>87</b>	Nb onto: <b>13</b>	Nb onto: <b>7</b>

Figure 31. LOV4IoT web services

```

@GET
@Path("/totalOnto/")
@Produces(MediaType.APPLICATION_XML)
public Response getTotalNumberOntology() {
    //load the LOV4IoT dataset into the model
    Model model = ModelFactory.createDefaultModel();
    ReadFile.enrichJenaModelOntologyDataset(model, Var.LOV4IOT_DATASET_PATH);
    M2MAppGeneric m2mappli = new M2MAppGeneric(model);

    //SPARQL query
    ExecuteSparql sparqlQuery = new ExecuteSparql(model, Var.ROOT_SPARQL_LOV4IoT + "countTotalOntology.sparql");

    //no variable to replace in the SPARQL query
    ArrayList<VariableSparql> var = new ArrayList<VariableSparql>();
    String resultSparqlsenml = sparqlQuery.getResultAsXML(var);

    return Response.status(200).entity(resultSparqlsenml).build();
}

```

**Figure 32. Example of the lov4iot/totalOnto: web service**

### 3. M3JsonWS (deprecated)

The web services provided by this Java class were focused on returning the result in JSON.

After, I found the solution to return the result either as JSON or as XML by adding:

These web services enable to query the M3 ontology to get:

- Get all M3 sensors (/json/m3/sensor)
- Get all M3 domains (/json/m3/featureOfInterest)
- Get all M3 measurement type (/json/m3/measurement)

### 4. 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<sup>12</sup>:

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

<sup>12</sup> <http://www.sensormeasurement.appspot.com/?p=m3api>



## Semantic Web of Things (SWoT) Generator

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

### STEP 1: Search M3 Template

=> call web service: (/m3/subclassOf/Sensor)

1. Choose a sensor (e.g., Light/Illuminance Sensor)
2. Choose the domain where is deployed your sensor (e.g., Weather)
3.  => call web service: (/m3/searchTemplate) => call web service: (/m3/subclassOf/FeatureOfInterest)

### STEP 2: Choose M3 Template

- Choose an application template:

### STEP 3: Download M3 template

- => call web service: (/m3/generateTemplate)

Figure 33. M3 web services used in the SWoT generator

## 5. NaturopathyWS

All web services used for the restaurant scenario<sup>13</sup>:

- Suggest home remedies according to the body temperature (/naturopathy/sick)
- Deducing mood according to the external luminosity (/naturopathy/emotion\_luminosity/)
- Suggesting food according to the outside temperature (/naturopathy/seasonTemperatureFoodRecipe/)
- Deducing mood or diseases from:
  - heart beat (/naturopathy/emotion\_disease/HeartBeat)
  - skin conductance (/naturopathy/emotion\_disease/SkinConductance)
  - blood pressure (/naturopathy/emotion\_disease/BloodPressure)
- See the comments in the Java class for more web services

<sup>13</sup> <http://www.sensormeasurement.appspot.com/?p=naturopathy>

www.sensormeasurement.appspot.com/?p=naturopathy

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Scenario Naturopathy: Suggest ingredients or recipes according to the weather, diseases, diets and emotions

### Suggesting home remedies according to body temperature

1. This scenario is based on: [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!)  => call web service /naturopathy/sick

Figure 34. Naturopathy web services used in the naturopathy scenario

## 6. RestaurantWS

All web services used for the restaurant scenario<sup>14</sup>.

See Java Class comments.

This scenario does not work exactly in the same way that last scenarios (naturopathy, transport, tourism). It was our second scenario who helps us to design the semantic engine, to reuse the domain knowledge, etc.

## 7. STACWS

All web services used for security application<sup>15</sup>, called STAC (Security Toolbox: Attacks & Countermeasures):

- Get all technologies referenced in the STAC dataset (stac/subclassOf/Technology)
- Get all attacks related to a specific technology (e.g., stac/attack/SensorTechnology)
- Get all features related to a specific security mechanism (e.g., stac/hasFeature/SSH)

<sup>14</sup> <http://www.sensormeasurement.appspot.com/?p=restaurant>

<sup>15</sup> <http://www.sensormeasurement.appspot.com/?p=stac>

- Get all security mechanisms related to a specific technology (e.g., stac/techno/SensorTechnology)
- See Java class for all web services and their comments

The name of the technology should be referenced in the STAC ontology<sup>16</sup>!

www.sensormeasurement.appspot.com/?p=stac

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## STAC (Security Toolbox: Attack & Countermeasure)

### Technologies used in your application?

1. Choose a technology (e.g., WiFi Technology)  => call web service: /stac/subclassOf/Technology
2. Attacks related to this technology:  => call web service: /stac/attack/SensorTechnology
3. Wait (10 seconds!)
4. A tooltip is displayed for more information about a specific security mechanism
5. Click on a security mechanism (e.g., WPA2): => call web service: /stac/techno/SensorTechnology
6. Advantages and weaknesses are displayed (Feature):
7. Security properties satisfied are displayed:

Figure 35. STAC web services

## 8. SWOTWS

- Web service for the M3 converter (/swot/convert\_senml\_to\_rdf)
- Web service to get all rules associated to a specific sensor (/swot/convert\_senml\_to\_rdf)

www.sensormeasurement.appspot.com/?p=senml\_converter

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## M3 Converter (SenML to RDF)

### Copy/paste your SenML/XML data

Copy/paste your SenML/XML sensor data here (need to be surrounded by zone balise):

```
<zone name="health">
<senml
bn="urn:body:uuid:c68ad78b-09eb-4303-ae3c-d5d23149ee96">
<e n="blood pressure" t="0"
u="Pa" v="56"></e>
<e n="cholesterol" t="0"
u="g/L" v="5"></e>
<e n="heartbeat" t="0"
u="beet/m" v="155"></e>

```

=> call web service: /swot/convert\_senml\_to\_rdf

Figure 36. Web service to convert senML data to RDF according to the M3 nomenclature

<sup>16</sup> securitytoolbox.appspot.com/stac#

## Sensor-based Linked Open Rules (S-LOR)

### Sensors used in your application?

=> /m3/subclassOf/?nameClass=Sensor&format=xml

Choose a sensor (e.g., precipitation sensor)

Rules using this sensor: => /swot/rule/WindDirectionSensor

- Rule: WestWind, IF m3:WindDirection greaterThan 225 m3:DegreeAngle AND lessThan 315 m3:DegreeAngle THEN WestWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: SouthWind, IF m3:WindDirection greaterThan 135 m3:DegreeAngle AND lessThan 225 m3:DegreeAngle THEN SouthWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: EastWind, IF m3:WindDirection greaterThan 45 m3:DegreeAngle AND lessThan 135 m3:DegreeAngle THEN EastWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>
- Rule: NorthWind, IF m3:WindDirection (greaterThan 0 m3:DegreeAngle AND lessThan 45 m3:DegreeAngle) OR (greaterThan 315 m3:DegreeAngle AND lessThan 360 m3:DegreeAngle) THEN NorthWind  
Linked Open Rules URL: <http://sensormeasurement.appspot.com/RULES/LinkedOpenRulesWeather.txt>

**Figure 37. Web service to get rules related to a specific sensor**



**This Java class should be merged with M3WS in future versions or create new Java file SLORSW and M3ConverterWS and then update the user interface in HTML and JavaScript.**

## 9. TourismWS

All web services used for the tourism scenario<sup>17</sup>:

- tourism/clothes\_weather/{nameMeasurement}: to suggest activities according to weather measurements
- tourism/activity\_weather/{nameMeasurement}: to suggest activities according to weather measurements
- /tourism/snowGarment/: to deduce snow, a more scenario involving two sensors at the same time

nameMeasurement is described in the M3 nomenclature<sup>18</sup>. For instance, it can be WeatherLuminosity, Precipitation, WeatherTemperature

See comments in the Java Class

<sup>17</sup> <http://www.sensormeasurement.appspot.com/?p=tourism>

<sup>18</sup> <http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>

www.sensormeasurement.appspot.com/?p=tourism

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Tourism (Weather & Emotion & Activity & Transport)

### Suggesting activities according to the weather

1. This scenario is based on: [M3 RDF sensor data](#)
2. We deduce the weather outside.
3. We propose activities according to the weather. => [/tourism/activity\\_weather/Precipitation](#)
4. M2M Application (Temperature => weather => Activity): Activity & Temperature
5. M2M Application (Luminosity => weather => Activity): Activity & Luminosity
6. M2M Application (Precipitation => weather => Activity): **Activity & Precipitation**
7. M2M Application (Wind speed => weather => Activity): Activity & Wind Speed

Figure 38. Tourism web services

## 10. TransportWS

All web services used for the transportation scenario<sup>19</sup>:

- `tourism/safety_device_weather/{nameMeasurement}`: to suggest safety devices in the smart car according to weather measurements
- `/tourism/snow/`: to deduce snow, a more scenario involving two sensors at the same time

`nameMeasurement` is described in the M3 nomenclature<sup>20</sup>. For instance, it can be `WeatherLuminosity`, `Precipitation`, `WeatherTemperature`

See comments in the Java Class

www.sensormeasurement.appspot.com/?p=transport

Semantic Web of Things M3 framework Scenarios Publications Security Contributing to M3 About us Memento

## Transport (Weather & Safety devices & Driver's state)

### Suggesting safety devices in the smart car according to the weather

1. This scenario is based on: [M3 RDF sensor data](#)
2. We deduce the weather outside.
3. We propose safety devices according to the weather. => [/transport/safety\\_device\\_weather/WeatherLuminosity](#)
4. M2M Application (Temperature => weather => Safety devices): Safety devices & Temperature
5. M2M Application (Luminosity => weather => Safety devices): **Safety devices & Luminosity**
6. M2M Application (Precipitation => weather => Safety devices): Safety devices & Precipitation

- Name=luminosity, Value = 5000.0, Unit=lx, InferType = Weather Luminosity, Deduce = Cloudy
- Name=luminosity, Value = 50000.0, Unit=lx, InferType = Weather Luminosity, Deduce = Sunny, Suggest= Switch on the sun visor

Figure 39. Transport web services

<sup>19</sup> <http://www.sensormeasurement.appspot.com/?p=transport>

<sup>20</sup> <http://www.sensormeasurement.appspot.com/documentation/NomenclatureSensorData.pdf>



## 11. Design a new web service

To design a new web service, take inspiration from all of these web services.

# Part VII : Adding a new SWoT template

Add a new template in the template dataset<sup>21</sup>:

- 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"/>
  <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 40. A SWoT template**

<sup>21</sup> [www.sensormeasurement.appspot.com/dataset/iot-application-template-dataset](http://www.sensormeasurement.appspot.com/dataset/iot-application-template-dataset)



# Part VIII : Adding a new ontology in LOV4IoT



In the future, we will automatically build the HTML web page according to the LOV4IoT RDF dataset. This work is ongoing. Currently, we have to update the HTML web page and the RDF dataset when we want to reference a new ontology-based project.

## 1. HTML web page

Go to [m3/WAR/html/lov4iot.html](http://m3/WAR/html/lov4iot.html)

Look for the table related to the domain, add a new line with all columns required.

## 2. LOV4IoT RDF dataset

In the LOV4IoT RDF dataset<sup>22</sup>, add a new ontology-based project.

---

<sup>22</sup> <http://www.sensormeasurement.appspot.com/dataset/lov4iot-dataset>

```

<m3:M2MApplication rdf:about="PaulStaroch">
  <m3:hasContext rdf:resource="&m3;Weather"/>
  <m3:hasContext rdf:resource="&m3;BuildingAutomation"/>
  <rdfs:label xml:lang="en">[Paul Staroch 2013]. See LOV4IoT for more details.</rdfs:label>
  <rdfs:comment xml:lang="en">Master's Thesis: A weather ontology for predictive control
  <m3:hasM2MDevice rdf:resource="&m3;Thermometer"/> in smart homes. 2013</rdfs:comment>
  <m3:hasM2MDevice rdf:resource="&m3;PrecipitationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;HumiditySensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;AtmosphericPressureSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SolarRadiationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;WindDirectionSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;WindSpeedSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SunPositionDirectionSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;SunPositionElevationSensor"/>
  <m3:hasM2MDevice rdf:resource="&m3;CloudCoverSensor"/>
  <m3:hasUrlOntology rdf:resource="http://paul.staroch.name/thesis/SmartHomeWeather.owl"/>
  <m3:hasUrlRule rdf:resource="http://paul.staroch.name/thesis/SmartHomeWeather.owl"/>
  <lov4iot:hasOntologyStatus rdf:resource="&lov4iot;OnlineLOV"/>
  <dcterms:creator>
    <foaf:Person rdf:about="mailto:paul@staroch.name">
      <foaf:name>Paul Staroch</foaf:name>
    </foaf:Person>
  </dcterms:creator>
</m3:M2MApplication>

```

**Figure 41. An ontology-based project referenced in the LOV4IoT RDF dataset**

## Part IX : Understanding the M3 converter

### I. Semantically annotate SenML/XML data with the M3 converter

We stored the semantic sensor data in the Google App Engine Datastore called Java Data Objects (JDO).

When you see this code or params, it means that I use Google datastore:

- Var.KIND\_JDO\_HEALTH\_V2
- Var.KEY\_NAME\_JDO\_HEALTH\_V2
- String kindJDO
- String keyNameJDO

In the file ConvertSensorDataToM3.java, you should delete all variables with the name kindJDO and keyNameJDO. the data will be stored in a jena model:

```
Model model = ModelFactory.createDefaultModel();
```

So you have to replace the params kindJDO and keyNameJDO by the model

```

public Model convertJavaObjectsToM3(Domain senmls, String kindJDO, String keyNameJDO) throws
IOException{
//HERE YOU HAVE DATA SEMANTICALLY ANNOTATED

```

```

// Where will be the resulting RDF dataset be saved if I want to perform the conversion on local disk?
infModel.write(System.out);
//then you can store your jena model as you want
// YOU CAN DELETE THIS
//GenericJDO.createOrUpdateMeasurementsSaved(kindJDO, keyNameJDO, infModel);
System.out.println("kindJDO:" + kindJDO + " keyNameJDO:" + keyNameJDO);

return infModel;
}

```

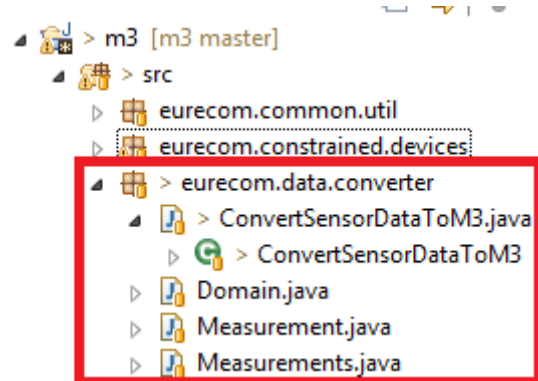


Figure 42. The M3 converter code to semantically annotate SenML/XML data

## II. Semantically annotate SenML/JSON data with the M3 converter

We build another Java package called “eurom.constrained.devices”, since we had to convertSenML/JSON sensor data.

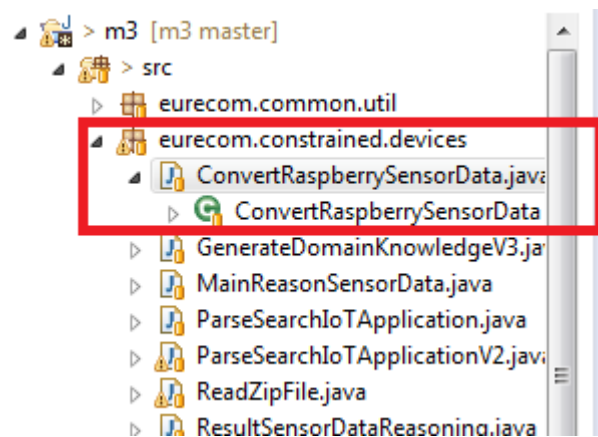


Figure 43. The M3 converter code to semantically annotate SenML/JSON data

# Part X : Extending STAC

You can either manually update the STAC ontology and dataset or use ontology editor tools such as Protégé. In this section, we display screenshot of the code.

## I. Adding a new technology to the STAC ontology

STAC ontology: <http://securitytoolbox.appspot.com/stac#>

To add a new technology to the STAC ontology is simple.

1. You have to add a new class (e.g., BluetoothTechnology), which is a subclass of stac:Technology (see Figure 43). Then, add the restriction regarding Attack. The technology is vulnerable to some attacks (e.g., BluetoothAttack) (see Figure 43).
2. Create a new subclass of stac:Attack (e.g., BluetoothAttack) (see Figure 44).
3. Create a new subclass of stac:SecurityMechanism (e.g., BluetoothSecurityMechanism) and the related restriction (see Figure 45).

```
<owl:Class rdf:ID="BluetoothTechnology">
  <rdfs:label xml:lang="en">Bluetooth Technology</rdfs:label>
  <rdfs:comment xml:lang="en">A protocol for short-range (up to 100 meters) wireless networks.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Technology"/>
  <rdfs:seeAlso rdf:resource="#&serviceContext;Bluetooth"/>
  <rdfs:seeAlso rdf:resource="#&ct;BluetoothConnectivity"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVulnerability"/>
      <owl:someValuesFrom rdf:resource="#BluetoothAttack"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**Figure 44. Creating a new STAC technology**

```
<owl:Class rdf:ID="BluetoothAttack">
  <rdfs:label xml:lang="en">Bluetooth Attack</rdfs:label>
  <rdfs:comment xml:lang="en"></rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Attack"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSecurityMechanism"/>
      <owl:someValuesFrom rdf:resource="#BluetoothSecurityMechanism"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**Figure 45. Creating a new STAC attack**

```

<owl:Class rdf:ID="BluetoothSecurityMechanism">
  <rdfs:label xml:lang="en">Bluetooth SecurityMechanism</rdfs:label>
  <rdfs:comment xml:lang="en"></rdfs:comment>
  <rdfs:subClassOf rdf:resource="#SecurityMechanism"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#protects"/>
      <owl:someValuesFrom rdf:resource="#BluetoothTechnology"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

**Figure 46. Creating a new STAC security mechanism**

## II. Enriching the STAC dataset

STAC dataset: <http://securitytoolbox.appspot.com/stac-dataset>

### 3. Updating the STAC dataset with a new security mechanism

Figure 46 shows the way to add a new instance of the BluetoothSecurityMechanism in the STAC dataset. This new instance is called JSR82. Do not forget to add the label and the comment to describe this new instance. In this example, we also explain that this instance is a type of stac:ProgrammingLanguageSecurityMechanism.

```

<stac:BluetoothSecurityMechanism rdf:about="JSR82">
  <rdfs:label xml:lang="en">JSR-82</rdfs:label>
  <rdof:type rdf:resource="#stac:ProgrammingLanguageSecurityMechanism"/>
  <rdfs:comment xml:lang="en">JSR-82 is the official Java Bluetooth Application
</stac:BluetoothSecurityMechanism> Programming Interface (API)</rdfs:comment>

```

**Figure 47. Add a new instance of BluetoothSecurityMechanism**

Figure 46 shows the way to add a new instance of the BluetoothAttack in the STAC dataset. In this example, the new Bluetooth attack is called "Bluejacking".

```

<stac:BluetoothAttack rdf:about="Bluejacking">
  <rdfs:label xml:lang="en">Bluejacking</rdfs:label>
  <rdfs:comment xml:lang="en">Bluejacking is the sending of either a picture or a message
</stac:BluetoothAttack> from one user through Bluetooth.</rdfs:comment>

```

**Figure 48. Add a new instance of BluetoothAttack**

You can add more explanations to the security mechanisms instance, for instance the stac:satisfies property. In the example depicted in Figure 48, we describe that the SSH satisfies three security properties called Authentication, Integrity and Confidentiality.

```

<stac:WebSecurityMechanism rdf:about="SSH">
  <rdfs:label xml:lang="en">Secure Shell (SSH)</rdfs:label>
  <rdfs:comment xml:lang="en">SSH is the now ubiquitous program for logging into or executing
  <rdf:type rdf:resource="&stac;Protocol"/> commands on a remote machine.</rdfs:comment>
  <stac:protectsInLayer rdf:resource="ApplicationLayer"/>
  <stac:satisfies rdf:resource="Authentication"/>
  <stac:satisfies rdf:resource="Integrity"/>
  <stac:satisfies rdf:resource="Confidentiality"/>
  <stac:hasFeature rdf:resource="Popular"/>
  <dcterms:hasPart rdf:resource="AsymmetricAlgorithm"/>
</stac:WebSecurityMechanism>

```

**Figure 49. Add the security property to the instance**

Figure 49 show the creation of a new security property instance called Confidentiality. We add links to the existing security ontologies (owl:sameAs) describing the same instance.

```

<stac:SecurityProperty rdf:about="Confidentiality">
  <rdfs:label xml:lang="en">Confidentiality/Privacy </rdfs:label>
  <rdfs:comment xml:lang="en">Confidentiality means that only destined user must
  <owl:sameAs rdf:resource="&herzog;Confidentiality"/> be able to read data. </rdfs:comment>
  <owl:sameAs rdf:resource="&denker;Confidentiality"/>
  <owl:sameAs rdf:resource="&kim;Confidentiality"/>
  <owl:sameAs rdf:resource="&maso;Confidentialite"/>
  <owl:sameAs rdf:resource="&securitysw;Confidentiality"/>
</stac:SecurityProperty>

```

**Figure 50. Add a new security property instance.**

## Part XI : Additional

### I. Jena TDB and Jena Fuseki (Internship Amira)

The project is available in the same BSCW directory

Install Jena TDB

Does not work with App Engine but the server tomcat

See documentation Internship

Install Jena Fuseki

Does not work with App Engine but the server tomcat

See documentation Internship

## II. Security issues to execute JavaScript with the project

to disable the browser security (javascript problem)

command name\_browser -disable-web-security

chrome -disable-web-security