Ontologies for emotions classification

André Furlan - UNESP - Universidade Estadual Paulista "Júlio de Mesquita Filho"

***Abstract*—A short explanation on how to make a java application based on ontologies and how to model this in order to get an usable knowledge base.**

***Index Terms*—ontologies, java, application, emotions, knowledge graph, owl**

I. Introduction

Sometimes it is hard to known what is the exact intentions behind the expressions of someone. There is so many faces and body details that must be observed [Ekm01] that is almost impossible to a non-trained classify the current person intentions or feelings. This article shows how to build an ontology, export and finally integrate it with the java programming language.

All the resulting sources (and more) derived from this article are available at **https://github.com/ensismoebius/ontologies**.

II. Tools needed

First of all an ontology editor must be installed, of course you can edit the ontologies by hand as well but this is a hard and tedious task. **Protégé** (https://protege.stanford.edu/) are recommended because it is a free open-source ontology editor and have a plenty of plugins that hopefully will help. **Java Development Kit** is needed as well, along with a good **java IDE**, in this article **Eclipse IDE** was the chosen one, optionally **Maven** dependency manager may be installed.

III. Building up the ontology

Anger, contempt, fear, happiness and lying has a wide range of emotions involved [Ekm01] [EF74] so build an ontology for that will help the system to infer the correlations.

For the sake of this article **it is important to save the ontology in RDF format**.

Begin adding to ontology the chosen characteristics like in the figure 1 this is what we want to know. The these

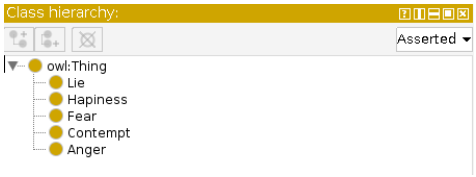

Fig. 1. Main characteristics

characteristics are composed by a number of emotions sometimes diferent classifications may have some similar emotions [Ekm01]. To tackle this difficult a new class must be created: **EmotionsSignals** (figure 2).

This class express a body and communicational signals some person may exhibit while talking. Subdividing **EmotionsSignals** using categories like **Hands**, **Head**, etc. The

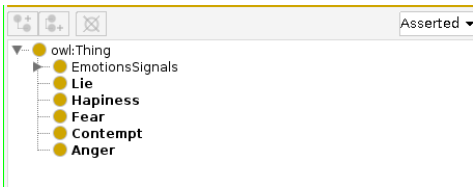

Fig. 2. Emotions signals

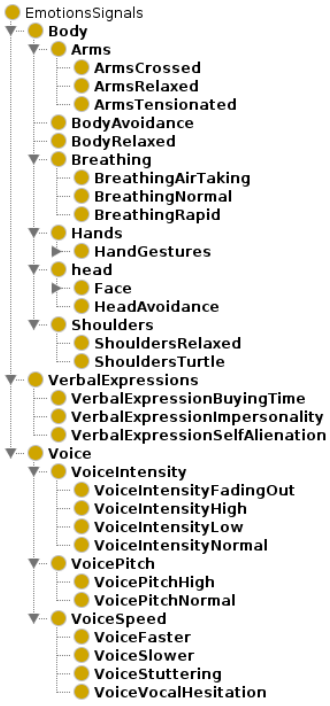

Fig. 3. Emotions signals sub classes

hierarchy ends like the figure 3 shows. Having the defined hierarchy, the next step is to bind this main characteristics to emotions signals. It is done by creating what is called **object properties**. To do that go to "object properties" tab inside "Entities" tab (figure 4) and create **hasCaracteristic** property. The **hasCaracteristic** property was chosen arbitrary. Finally, go to "classes" tab and select one by one the main characteristics (*Lie* for example) and using the **SubClass** add option bind emotions signals using the **object retriction editor** like in the figure 5.

A. Building the "Lie"

According to [EF74] and [Ekm01] a lie may be constituted by **at least by 3** of the factors shown in the figure 6.

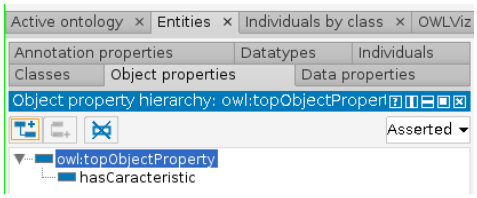

Fig. 4. Object properties tab

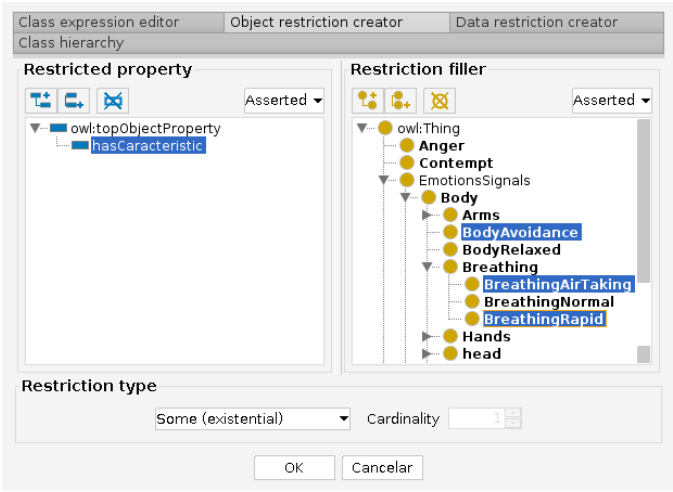

Fig. 5. Binding emotions signals

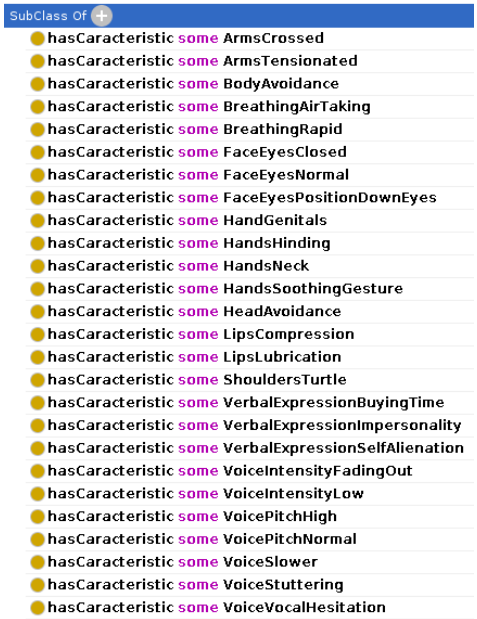

Fig. 6. Building the Lie class

B. Building the "Anger"

According to [Ekm01] the anger may be constituted by **at least by 3** of the factors shown in the figure 7.

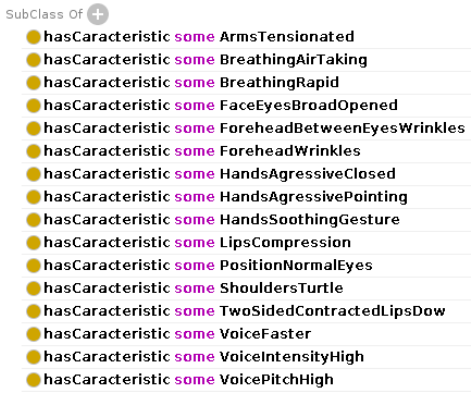

Fig. 7. Building the Anger class

C. Building the "Happiness"

According to [Ekm01] the happiness may be constituted by **at least by 3** of the factors shown in the figure 8.

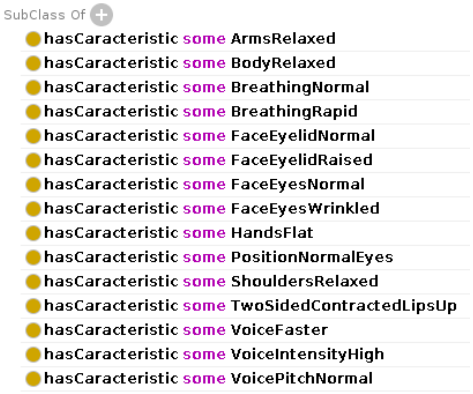

Fig. 8. Building the Happiness class

D. Building the "Fear"

[bpht] According to [Ekm01] the fear may be constituted by **at least by 3** of the factors shown in the figure 9.

E. Building the "Contempt"

[bpht] According to [Ekm01] the contempt may be constituted by **at least by 3** of the factors shown in the figure 10.

Do not forget to save the ontology in RDF format.

IV. SPARQL

With the ontology ready a way to retrieve the main characteristics given some emotions signals is mandatory if the target is a system that can classify if a person is angry, if its lying, happy, etc.

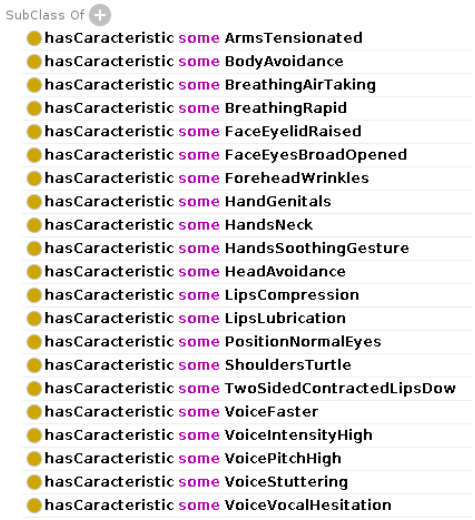

Fig. 9. Building the Fear class

Fig. 10. Building the Contempt class

That is where the SPARQL enters: Activating the SPARQL editor at Protégé (window → tabs → SPARQL Query) it is possible to run against the ontology some queries that are going to return the desired data.

At the current application the needed data are:

- Emotions signals available using listing 1.
- The main characteristics given some emotions signals using listing 2.

Note that at the listing 2 the **:ArmsRelaxed** and **:BodyAvoidance** statements are just examples for a query, it is possible to put more or fewer parameters as the necessity appears.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.semanticweb.org/ensismoebius/ontologies/2019/9/tellingLies#>

SELECT DISTINCT ?name WHERE {
   ?directSub rdfs:subClassOf ?super ;
   rdfs:subClassOf [
        rdf:type owl:Restriction ;
             owl:onProperty :hasCaracteristic;
             owl:someValuesFrom ?name
   ] .
   FILTER( ?name != :EmotionsSignals )
}
ORDER BY (?name)
```

Listing 1. Getting Emotions signals available

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://www.semanticweb.org/ensismoebius/ontologies/2019/9/tellingLies#>

SELECT DISTINCT ?directSub
WHERE {
  ?directSub rdfs:subClassOf ?super ;
    rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasCaracteristic ;
    owl:someValuesFrom :ArmsRelaxed
  ] .
  ?directSub rdfs:subClassOf ?super ;
    rdfs:subClassOf [
    rdf:type owl:Restriction ;
    owl:onProperty :hasCaracteristic ;
    owl:someValuesFrom :BodyAvoidance
  ] .
}
```

Listing 2. Getting the main characteristics given some emotions signals

V. Programming the application

The ontology will be read using the following libraries:

- jena-arq, version 3.13.0
- jena-core, version 3.13.0
- owlapi-distribution, version 3.5.1
- org.semanticweb.hermit, version 1.3.8.4

It is possible to download these libraries manually but **Maven** are recommended because there is **a lot** of dependencies.

```
<dependency>
  <groupId>net.sourceforge.owlapi</groupId>
  <artifactId>owlapi-distribution</artifactId>
  <version>3.5.1</version>
</dependency>

<dependency>
  <groupId>com.hermit-reasoner</groupId>
  <artifactId>org.semanticweb.hermit</artifactId>
  <version>1.3.8.4</version>
</dependency>

<dependency>
  <groupId>org.apache.jena</groupId>
  <artifactId>jena-core</artifactId>
  <version>3.13.0</version>
</dependency>

<dependency>
  <groupId>org.apache.jena</groupId>
  <artifactId>jena-arq</artifactId>
  <version>3.13.0</version>
</dependency>
```

Listing 3. Needed libraries

After all that the programming task is trivial: Given a **set of emotions** signals return the **respective main characteristics** if any.

In this work excluding the graphical interface the important part of the programming are shown at listing 4 and 5.

```
public ArrayList<String> getSignals() throws
  OWLOntologyCreationException {

String queryString = "PREFIX rdf: <http://www.w3.
  org/1999/02/22-rdf-syntax-ns#>\n"
+ "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n"
+ "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
  schema#>\n"
+ "PREFIX xsd: <http://www.w3.org/2001/XMLSchema
  #>\n"
+ "PREFIX : <http://www.semanticweb.org/
  ensismoebius/ontologies/2019/9/tellingLies#>\n"
+ "SELECT DISTINCT ?name WHERE { ?directSub rdfs
  :subClassOf ?super ;"
+ "rdfs:subClassOf [ rdf:type owl:Restriction ;"
+ "owl:onProperty :hasCaracteristic; owl:
  someValuesFrom ?name] ."
+ " FILTER( ?name != :EmotionsSignals )} order by
  (?name)";

Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.
  create(query, model);

ArrayList<String> resultsList = new ArrayList<
  String>();
ResultSet results = qexec.execSelect();
while (results.hasNext()) {
  QuerySolution qsol = results.nextSolution();
  resultsList.add(qsol.get("name").toString());
}

qexec.close();

return resultsList;
}
```

Listing 4. Retrieve all available signals method

```
public ArrayList<String> getFelling(ArrayList<
  String> signals) throws
  OWLOntologyCreationException {

String queryString = "PREFIX rdf: <http://www.w3.
  org/1999/02/22-rdf-syntax-ns#>\n"
+ "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n"
+ "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
  schema#>\n"
+ "PREFIX xsd: <http://www.w3.org/2001/XMLSchema
  #>\n"
+ "PREFIX : <http://www.semanticweb.org/
  ensismoebius/ontologies/2019/9/tellingLies#>\n"
+ "SELECT DISTINCT ?directSub " + "WHERE { ";

  for (String signal : signals) {
    queryString += "?directSub rdfs:subClassOf ?
  super ;" + "rdfs:subClassOf [" + "rdf:type owl:
  Restriction ;"
    + "owl:onProperty :hasCaracteristic ;" + "owl:
  someValuesFrom <" + signal + ">] .";
  }

  queryString += "}";

Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.
  create(query, model);

ArrayList<String> resultsList = new ArrayList<
  String>();
ResultSet results = qexec.execSelect();
while (results.hasNext()) {
  try {
    QuerySolution qsol = results.nextSolution();
    resultsList.add(qsol.get("directSub").toString
  ());
  } catch (NullPointerException e) {
    continue;
  }
}

qexec.close();
return resultsList;
}
```

Listing 5. Retrieve characteristics

The final application looks like the figure 11.

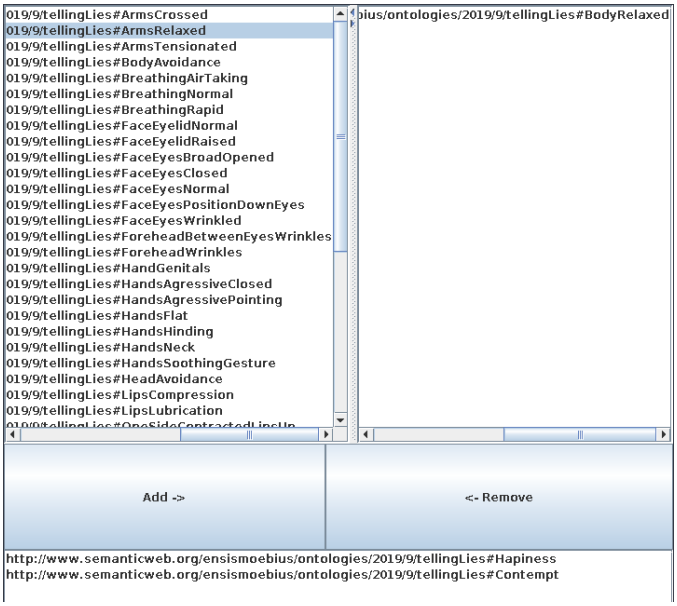

Fig. 11. Final application screenshot

REFERENCES

[EF74] Paul Ekman and Wallace V Friesen. Detecting deception from the body or face. *Journal of personality and Social Psychology*, 29(3):288, 1974.

[Ekm01] P. Ekman. *Telling Lies: Clues to Deceit in the Marketplace, Politics, and Marriage*. W.W. Norton, 2001.