UGR, ETSIIT

Developement of a power grid ontology and a sample application

March 30, 2017

1 Introduction

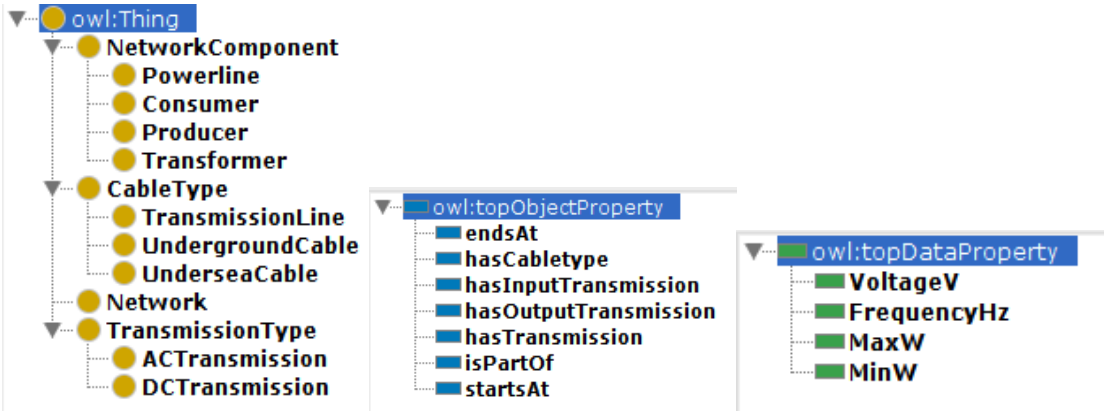
The Semantic Web extends the Web to make data between computers easier to exchange and easier to use. The term "Granada" can be supplemented, for example, in a web document with the information as to whether a ship name, family name or town name is meant here. This additional information explicates the otherwise unstructured data. Standards are used for the publication and use of machine-readable data. While people can close such information from the given context and unconsciously build such links, machines must first be brought to this context. For this purpose, the content is linked with further information and Web Ontology Language (OWL) is a way to write domains and their relationships formally. OWL is a specification of the World Wide Web Consortium (W3C) to create, publish and distribute ontologies using a formal description language.

This paper therefore creates a an power grid ontology as a general use case in Section II. Section III delineates the general challenge arising with the regard to the integration of Java Application to OWL Ontologies and presents existing technical approaches. Section IV provides an outlook and discusses the suitability for use cases with focus on integration and multi-platform approaches.

2 Ontology details

2.1 Building blocks

The power grid ontology is aimed to give a basis for representing power grids in terms of power production, consumption and transmission. For that purpose the following classes, as listed in figure 2.1a were designed.



(a) Class hierarchy (b) Object properties (c) Data properties

Figure 2.1: Building blocks of the Power Grid ontology.

The most basic building blocks are the *NetworkComponents* which model all physical entities a network may consist of and are disjoint. There are *Producers* adding power to the network, *Consumers* drawing energy from the network, *Powerlines* transporting the energy between exactly two other network entities and *Transformers* managing splits, merges and changes in the transmission type of *Powerlines*.
Those transmission types are modelled in the *TransmissionType* class. This class is divided into two disjoint subclasses *ACTransmission* for alternating current and *DCTransmission* for direct current.
Each *Powerline* furthermore has a *CableType* indicating the layout of a power line. There are 3 disjoint alternatives: regular *TransmissionLines*, *UndergroundCables* and *UnderseaCables*.
Finally every *NetworkComponent* is part of a specific *Network*. The relations between the different classes written in prose above are represented in the ontology as object properties. The domain and the range of every object property in this ontology are left blank on purpose, because they do not serve as constraint but as axioms for the reasoner. Constraints for the object properties however are represented as subclass rules in the power grid ontology. Every class relating to another one via an object property has a subclass rule of the type

(object property name) exactly 1 (other classes name)

e.g. in *NetworkComponent* the following subclass rule exists:

isPartOf exactly 1 *Network*

Apart from the *isPartOf* property for *NetworkComponents* other properties exist to model the relations described above. *hasOutputTransmission* maps a *TransmissionType* to every *Producer*, *hasInputTransmission* does the mapping between *TransmissionType* and *Consumer* and finally *hasTransmission* relates a *Powerline* with its *TransmissionType*. Furthermore for expressing the *CableType* of a *Powerline* the *hasCableType* entity exists. Furthermore *Powerlines* need a starting- end an endpoint so *startsAt* allows for *Producers* or *Transformers* as starting points while *endsAt* respectively allows for *Consumers* and *Transformers* as endpoints.
Finally the ontology defines some data properties which quantify important measures in a power grid. For starters every *Producer* and *Consumer* has a maximum power consumption expressed by the *MaxW* property and a minimum power consumption expressed by the *MinW* property, which are both quantified by integers. Furthermore every *TransmissionType* uses a certain voltage, which is represented in the ontology by the integer property *VoltageV*. On top of that every *ACTransmission* also has a certain frequency of phase shift, which in the ontology is described by the integer property *FrequencyHz*.
Apart from the building blocks describing a power grid we also created entities forming a sample power grid.

2.2 A SAMPLE POWER GRID

All entities involved in the sample power grid can be seen in 2.2. The power grid uses three different types of transmission. The energy of all producers is gathered in the *HighToMediumTransformer*. While the current received from the *OffshoreWindFarm* via long range transmission line using the DCTransmission *800kVDC* is transformed to a medium energy level

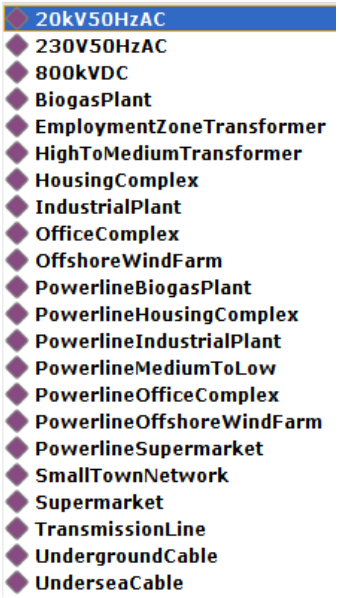


Figure 2.2: Entities of the sample power grid.

ACTransmission *20kV50HzAC*, the current from the *BiogasPlant* already is transmitted as *20kV50HzAC*. The *HighToMediumTransformer* distributes energy to an *IndustrialPlant*, a *HousingComplex* and an employment zone having its own *EmploymentZoneTransformer*. That transformer transforms the incoming current to a *230V50HzAC* ACTransmission. Connected to that transformer are a *Supermarket* and an *OfficeComplex*.
All connections between the mentioned Producers, Consumers and Transformers are Entities of the *Powerline* type and named *Powerline...*. Each of those power line have one of the three defined CableTypes: *TransmissionLine*, *UnderseaCable* or *UndergroundCable*.
Each NetworkComponent involved is part of the *SmallTownNetwork*.
Filling in every possible data and object property involved in the construction of this sample network would have been a lot of redundant work. Therefore a set of SWRL Rules was created that allows a reasoner to induce some network properties.

2.3 SWRL Rules

The power grid ontology uses four rules for inducing a *TransimssionType* for a *NetworkComponent*, as can be seen in figure 2.3.

	Na...	Body	Comment
☑	S1	startsAt(?pl, ?s) ^ hasTransmission(?pl, ?tt) ^ Powerline(?pl) -> hasOutputTransmission(?s, ?tt)	
☑	S2	hasTransmission(?pl, ?tt) ^ endsAt(?pl, ?e) ^ Powerline(?pl) -> hasInputTransmission(?e, ?tt)	
☑	S3	Producer(?power_gridp) ^ startsAt(?power_gridline, ?power_gridp) ^ hasOutputTransmission(?power_gridp, ?power_gridtt) -> hasTransmission(?power_gridline, ?power_gridtt)	
☑	S5	Consumer(?power_gridc) ^ endsAt(?power_gridline, ?power_gridc) ^ hasInputTransmission(?power_gridc, ?power_gridtt) -> hasTransmission(?power_gridline, ?power_gridtt)	

Figure 2.3: SWRL Rules used in the power grid ontology.

The first rule says that if a *Powerline* starts at s and has *TransmissionType* tt, s also has to have transmission type tt.
The second rule is the same for the endpoint: If a *Powerline* ends at e and has *TransmissionType* tt, e also has to have transmission type tt.
The third rule says if a *Powerline* starts at a *Producer* having *TransmissionType* tt it also has to have transmission type tt.
Finally the fourth rule again reflects the third one only for endpoints: If a *Powerline* ends at a *Consumer* having *TransmissionType* tt it also has to have transmission type tt.

3 Processing OWL Ontologies using Java

In the following, Java code will be presented to make requests on the build ontology in section II. For this task we the *OWL API*. The *OWL API* is a *Java API* and reference implmentation for creating, manipulating and serialising *OWL Ontologies*. The latest version of the API is focused towards *OWL 2*. The challenges that are to be addressed can be illustrated with a simple scenario. We assume that an energy provider is trying to receive information over his power grid on a dashboard that is written in Java Code. In the *OWL API*, an *OWLOntology* is an interface, modeling a set logical and non-logical *OWLAxioms*, with a name (an IRI), a physical location and convenience methods to retrieve such axioms. An *IRI* allows easily identifying *OWLEntities*. Furthermore, class names, data, object properties, annotation properties and named individuals.

First of all, we create two variables to load an ontology from an existing file. There are different ways to read and work with *OWL Files*. One option is to work with the *OWLOntologyManager*. *OWLOntologies* are created by *OWLOntologyManager* and all other interfaces are built using *OWLDataFactory*.

```
public static final IRI powergrid_iri = IRI.create
        ("file:///Users/felipe-oehrwald/Documents/GitHub/WebSemantic/WebSemantic/power_grid.owl");
public static final IRI example_iri = IRI.create
        ("http://www.semanticweb.org/ontologies/ont.owl");
OWLDataFactory df = OWLManager.getOWLDataFactory();
```

Figure 3.1: Load an IRI based on the OWL File.

```
public static OWLOntologyManager create() {
    OWLOntologyManager m = OWLManager.createOWLOntologyManager();
    m.addIRIMapper(new AutoIRIMapper(
            new File("materializedOntologies"), true));
    return m;
}
```

Figure 3.2: Creates an IRI based on the OWL File.

```
System.out.println(o.getAxiomCount());
System.out.println(o.getClassesInSignature());
System.out.println(o.getObjectPropertiesInSignature());
System.out.println(o.getDataPropertiesInSignature());
```

Figure 3.3: Executing request on OWL Ontology.

```
168
[<http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerType>, <http://www.se
[<http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#endsAt>, <http://www.semar
[owl:topDataProperty, <http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#Frequ
[]
http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineBiogasPlant
[<http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineBiogasPlant>]
[]
http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineBiogasPlant
<http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineBiogasPlant>
[]
http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineOfficeComplex
[<http://www.semanticweb.org/clemens/ontologies/2017/0/power_grid#PowerlineOfficeComplex>]
[]
```

Figure 3.4: Result of the Java Request.

The output reveals that it is possible to run simple request on an OWL Ontology that have been created in Protege. With the request we received the size of the classes that are in the signature of this object (168 in total) or the names of the classes that have been build in the referenced ontology.

4 OUTLOOK

In this paper, we highlighted the enormous impact of web ontologies itself and the capabilities of web ontologies in combination with Java Application. We analyzed several existing approaches for technically supporting such integration. From the perspective of integration as well as from a broader view of a modest installation web ontologies fail to keep promise of an overall flexible environment in style of MEAN stack software bundle. Although, it shares the same implementation steps as relational database services (MySQL) in Java Application the effort is pretty high to connect OWL to Java Application. Moreover, the amount available use cases and best practice approaches is still limited. This is due to the fact, that Web Semantic is as yet less popular inside the communities and this therefore needs to be further incentivized to raise popularity. A possibility can be the use of D2R Server. D2R Server allow to publish the content of relational databases on the Semantic Web in which it maps databases into RDF format.

There is thus a clear need for developing appropriate mechanisms to connect Java Application to OWL Files andto unleash the full potential of web ontologies. For further research, it would be interesting to work with Python, as a programming language with less leverage. Furthermore, it would be intressting to do a benchmarking between Java and Python in regard to OWL Files. Nevertheless, Web Ontologies will become more and more important. With the emerge of Open Data Government policies, that allow free access to machine readable information, web semantic technologies like for instance RDF will play an important role in the future in scraping and linking data.