UNIVERSITÀ

DEGLI STUDI DI BARI

ALDO MORO

COMPUTER SCIENCE DEPARTMENT

MASTER'S DEGREE IN COMPUTER SCIENCE

CURRICULUM KNOWLEDGE ENGINEERING

AND MACHINE INTELLIGENCE

THESIS IN

ARTIFICIAL INTELLIGENCE

AN APPROACH TO CONNECT SEMANTIC WEB TECHNOLOGIES AND GRAPH DATABASES

Supervisor:

Eng. Prof. Stefano FERILLI

Co-supervisor:

Domenico REDAVID

Student:

Davide DI PIERRO

Academic Year 2020/2021

It is our choices, Harry, that show what we truly are, far more than our abilities.

Harry Potter and the Chamber of Secrets

Summary

Abstract

The number of databases available both in the public domain and in private keeps on increasing every day. Scientists and researchers need to analyze and make use of the data stored in different databases. One limitation is that these databases are stored in diverse formats. However, semantic web methods have introduced the Resource Description Format (RDF) to unify heterogeneous databases. In this thesis, we want to propose a solution to be able to connect semantic web technologies with graph databases.

The main reason that leads us to this work is the possibility of doing reasoning on a portion of a graph. Through connection, and importing the ontology, we can extract axioms that are not available only with the graph database. The most common reasoning operations are checking the consistency of the knowledge base and the satisfiability of a class.

This can allow us to extract new information but also to explain it, since one of the essential characteristics of symbolic AI is its explicability. Explanatory mechanisms are useful in some circumstances but essential in others. In tourism, explanations would be optional but in the medical field they are essential.

I. Introduction, Background and Related Works

The World-Wide Web Consortium's Semantic Web initiative began in recent years and has attracted either interest or skepticism. This initiative started thanks to one of its main founders, Tim Berners-Lee, of a flexible, integrated, automatic, and self-adaptive Web that provides a richer, more interactive experience for users. During the years, W3C developed a set of standards and tools to support this scope, and now these are usable and could have a real impact on the Web.

Recently, the term knowledge graph has been used frequently in research and business, usually in close association with Semantic Web technologies, linked data, large-scale data, large-scale analytics, and cloud computing.

During the years, it raised the question of what the difference between the Semantic Web and knowledge graphs is. Smaller KGs, for example, enterprise knowledge graphs, can be differentiated from the Semantic Web because of their narrow domain. The goal of large search engines is to crawl and process all the information available on the web, which leads to increased interest in the widespread implementation of semantic technology. Considering the layers of the Semantic Web, a knowledge graph, by comparison, employs either the same technology for each layer or a similar one that offers the same features. For example, companies might use proprietary identifiers instead of URIs, and JSON-LD5 as the serialization format instead of XML and RDF. However, these technologies are only examples, and particularly in the syntax level XML is often replaced by lighter and more easily readable formats such as and more easily readable formats such as Turtle, N-Triples or N-Frames in the Semantic Web community. In conclusion, the Semantic Web could be interpreted as the most comprehensive knowledge graph, or, conversely, a knowledge graph that crawls the entire Web could be interpreted as an autonomous Semantic Web.

Considering the strong correlation between the two technologies, we want to propose a strategy to connect these two and to create value in their use.
This thesis is organized as follows: Section 1 provides background knowledge and state of art. Section 2 describes the proposed solution and the Appendix shows the implementation details of our solution.

I.I Semantic Web Technologies

The term *Semantic Web* is widely used, often without much care or understanding of its origin and meaning. However, in general, there are three main views of the term which are widely used: the vision, the programme and the technology.

I.I.I The Vision

The Semantic Web is inspired by a vision of the current Web which has been in the background since its inception, and which is influenced by earlier work dating back to Vannevar Bush's idea of the "memex" machine in the 1940s (based on a universal library, complete with a searchable catalogue)[1]. Tim Berners-Lee originally envisioned the WWW as including richer descriptions of documents and links between them [2]. However, in the effort to provide a simple, usable and robust working system, which could be used by everyone "out-of-the-box", these ideas were put to one side, and the simpler, more human-mediated Web which we know today resulted.

The bigger vision found expression in an article written by Tim Berners-Lee, Jim Hendler and Ora Lassila in Scientific American in May 2001 [3]. In this article, they provide a compelling vision of a world where instead of people laboriously trawling through information on the Web and negotiating with each other directly to carry out routine tasks such as scheduling appointments, finding documents and locating services, the Web itself can do the hard work for them. This can be done by providing sufficient context about resources on the Web and also providing the tools to use the context so that machines can find the right things and make decisions. Once the Web has a mechanism for defining semantics about resources and links, then the possibility arises for automatic processing by software agents, rather than mediation by people. In the article, the Semantic Web is defined as "an extension of the current Web in which

information is given well-defined meaning, better enabling computers and people to work in cooperation."

A simple example used to motivate the Semantic Web is the need to discover documents on the Web, not only from their textual content, as conventional search engines do, but also from a description. The problem is exemplified by the frustration in finding articles written by a particular author, rather than those which include the author's name. In response to the query 'Tim Berners-Lee' a search engine will respond with all the papers including that phrase, some of which will be by Tim Berners-Lee, but most of which will cite or refer to him. The Semantic Web can allow each document on the Web to be annotated stating who its author was, when it was created, and what content it has; then only those with the appropriate author will be returned.

To add these descriptions or annotations, it is necessary to state what this additional description, sometimes known as "metadata", should be, and how it should be interpreted. How this is done is the subject of the programme.

I.I.II The Programme

It was not until the Semantic Web Roadmap [4] appeared, setting out a plan for re-engineering the Web to achieve this vision, that the Semantic Web became a programme.

The W3C was set up when it became clear that there was a danger of the Web breaking apart through the pressure of competing commercial interests, and it is now a major forum providing information infrastructure between people and organizations in the world. The W3C seeks to maintain the interoperability and universality of the Web via the setting of open standards to which Web tools should conform – independent of interests. The Semantic Web initiative was started as the Web Metadata Working Group in 1998, and subsequently became the Semantic Web Activity [5] which has taken the view that the Semantic Web

provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming.

Beyond the W3C, the programme has taken on a life of its own. A large number of researchers are now exploring how to take the best advantage of the technology. It has the attraction of combining the distributed nature of the Web with the power of semantic description, logic and reasoning. There are many projects within the UK and sponsored by the European Commission, as well as in the US and the rest of the world. The total investment in the Semantic Web worldwide has been in the tens of millions of pounds.

Also notable within the Semantic Web is how communities of individual developers and users are working together to provide tools and information collaboratively. However, to make a major impact on the IT infrastructure, major IT companies will need to take part.

I.I.III The Technologies

The third common use of the term Semantic Web is to identify a set of technologies, tools and standards which form the basic building blocks of a system that could support the vision of a Web imbued with meaning. The Semantic Web has been developing a layered architecture, which is often represented using a diagram first proposed by Tim Berners-Lee, with many variations since. Figure 1 gives a typical representation of this diagram.

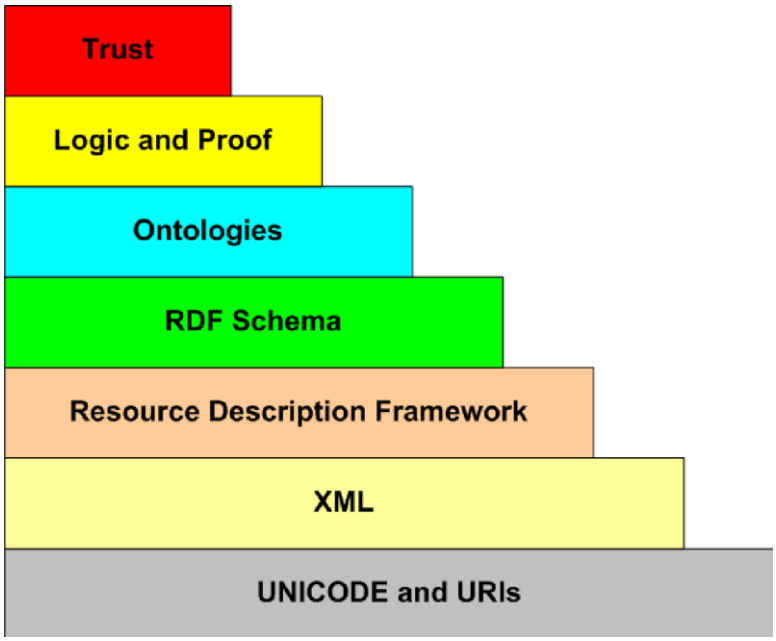


Figure 1: Semantic Web layered architecture

While necessarily a simplification which has to be used with some caution, it nevertheless gives a reasonable conceptualization of the various components of the Semantic Web. We briefly describe these layers.

Unicode and URI: Unicode, the standard for computer character representation, and URIs, the standard for identifying and locating resources (such as pages on the Web), provide a baseline for representing characters used in most of the languages in the world, and for identifying resources.

XML: XML and its related standards, such as Namespaces, and Schemas, form a common means for structuring data on the Web but without communicating the meaning of the data. These are well established within the Web already.

Resource Description Framework: RDF is the first layer of the Semantic Web proper. RDF is a simple metadata representation framework, using URIs to identify Web-based resources and a graph model for describing relationships between resources. Several syntactic representations are available, including a standard XML format. RDF format is used to model relationships between two entities using a 3-tuple, called a triple. The 3-tuple consists of subject, predicate,

and object. A predicate is a relationship between, or a fact about, the subject and the object. Subjects, predicates, and objects can be uniquely represented by uniform resource identifiers (URI). All the elements in a triple can be URIs. However, the objects can also be strings. URIs might have a repeated prefix when they are assigned to related concepts. In such cases, a namespace can be assigned to avoid redundancy and to easily identify related concepts. [6]

RDF Schema: a simple type modelling language for describing classes of resources and properties between them in the basic RDF model. It provides a simple reasoning framework for inferring types of resources.

Ontologies: a richer language for providing more complex constraints on the types of resources and their properties.

Logic and Proof: an (automatic) reasoning system provided on top of the ontology structure to make new inferences. Thus, using such a system, a software agent can make deductions as to whether a particular resource satisfies its requirements (and vice versa).

Trust: The final layer of the stack addresses issues of trust that the Semantic Web can support. This component has not progressed far beyond a vision of allowing people to ask questions of the trustworthiness of the information on the Web, to assure its quality.

The Semantic Web initiative has an ambitious programme to bring existing work on knowledge representation and reasoning to bear on the Web. These concepts were traditionally developed within the Artificial Intelligence community, and this has given the impression that the activity is of largely academic interest. A common misconception is that it is an attempt to bring AI to the Web. However, the Semantic Web has a less ambitious and more immediately realizable goal of making the Web machine-processable, making it in practice more like database and information systems management, but extended to the database of the whole Web. The application and potential of this work are enormous. [1]

I.II Knowledge Graphs

Considerable research into knowledge graphs (KGs) has been carried out in recent years, especially in the Semantic Web community, and thus a variety of partially contradicting definitions and descriptions have emerged. One definition, provided here [7], defined a KG as follows: "a knowledge graph mainly describes real-world entities and their interrelations, organized in a graph, defines possible classes and relations of entities in a schema, allows for potentially interrelating arbitrary entities with each other and covers various topical domains". It is necessary to define and differentiate KGs from other concepts to make valuable and accurate statements about the introduction and dissemination of knowledge graphs. The second problem leads to the misleading assumption that the term knowledge graph is a synonym of knowledge base, which is itself often used as a synonym for ontology. An ontology is a formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased complexity [8]. Ontological representations allow semantic modelling of knowledge and are, therefore, commonly used as knowledge bases in artificial intelligence (AI) applications, for example, in the context of knowledge-based systems. Application of an ontology as a knowledge base facilitates validation of semantic relationships and derivation of conclusions from known facts for inference (i.e., reasoning) [8]. We explicitly emphasize that an ontology does not differ from a knowledge base, although ontologies are sometimes erroneously classified as being at the same level as database schemas [9]. An ontology consists not only of classes and properties (e.g., owl:ObjectProperty and owl:DatatypeProperty), but can also hold instances (i.e., the population of the ontology). The second interpretation leads to the assumption that a knowledge graph is a knowledge-based system that contains a knowledge base and a reasoning engine. A more complete definition of KG is the following:

"A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge". This definition aligns with the assumption that a knowledge graph is somehow superior and more complex than a knowledge base (e.g., an ontology) because it applies a reasoning engine to generate new knowledge and integrates one or more information sources. [10] Knowledge Graphs can effectively organize and represent knowledge so that it can be efficiently utilized in advanced applications. Recently, reasoning over knowledge graphs has become a hot research topic, it can obtain new knowledge and conclusions from existing data. We know there exist at least three categories of reasoning: rule-based, distributed representation-based and neural network-based. [11]

One of the most known KG is DBpedia. It is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against datasets derived from Wikipedia and to link other datasets on the Web to Wikipedia data. We show a portion of the DBpedia Knowledge Graph in Figure 2.

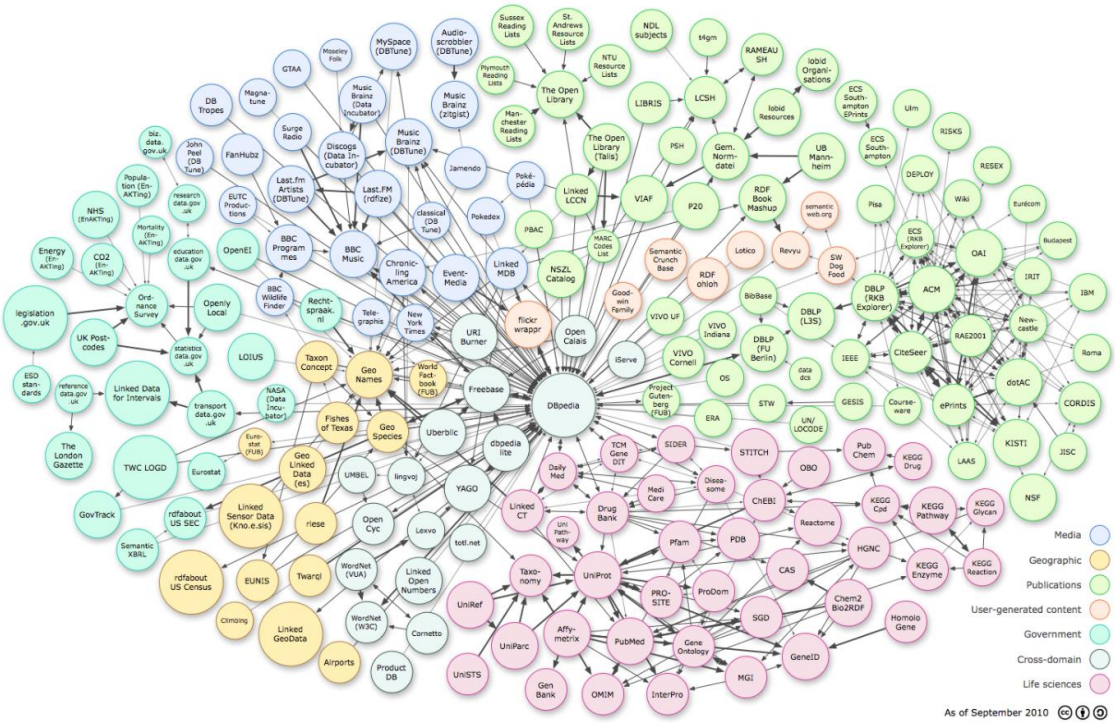


Figure 2: Portion of the DBpedia graph

I.II.I Neo4j

Neo4j is an open-source graph database software developed entirely in Java. It is a fully transactional database, which is integrated into applications allowing standalone operation and stores all data in a folder. It is purpose-built to work with highly connected data, delivers lightning-fast performance and enables powerful, actionable insights.

With Neo4j, you can map, store and traverse networks of highly connected data to reveal invisible contexts and hidden relationships. By intuitively analyzing data points and the connections between them, Neo4j powers intelligent, real-time applications that tackle today's toughest challenges. Neo4j combines native graph storage, a scalable architecture optimized for speed and ACID compliance to ensure predictability of relationship-based queries, all without compromising your data's integrity. Neo4j's index-free adjacency shortens read time, allowing you to run complex, multi-hop queries at lightning-fast speeds.

It also minimizes learning-related downtime, thanks to a mature UI with intuitive interaction and built-in learning. You'll also have access to a wealth of learning materials, thanks to the Neo4j community. There are a lot of APIs and drivers for all major languages, including Cypher, the world's most powerful and productive graph query language, or the native Java API for writing custom extensions. Loading data in Neo4j is rapid and simple, with a staggering loading speed for even huge data sets, all with a very low memory footprint.

The logical model is the physical model, and with Neo4j, you're empowered to make reflective on the ever-changing business landscape. The full documentation about Neo4j, its drivers and Cypher can be found here [12].

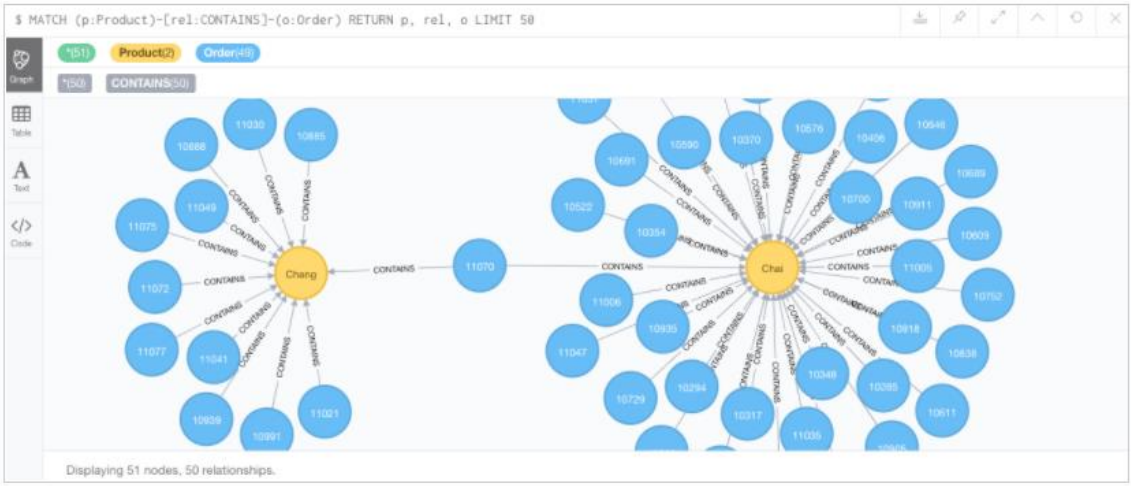


Figure 3: Part of a graph database

I.III Handling Ontologies

The first, simplest, tool is a software created by W3C for validating an RDF document. It is much more than a validator as it also builds triples and a knowledge graph. The second is a library developed in Python named RDFLib that allows you to read, manipulate and write ontologies in different formats. The last is an API written in Java named Apache Jena that provides the main

services offered by RDFLib but the two libraries differ in some operations and modes. Let us now describe the details.

I.III.I W3C RDF Validator

The validator is available here [13]. It is a parser that can read an RDF document and construct, if the document is syntactically correct, the triples related to the knowledge graph. By setting some parameters it is also possible to choose if you want to show the relative graph and its format. We report a small example.

This is the RDF code:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description rdf:about="http://www.w3.org/">
  <dc:title>World Wide Web Consortium</dc:title>
  </rdf:Description>

<rdf:Description
   rdf:about="http://www.recshop.fake/cd/Empire Burlesque"
   cd:artist="Bob Dylan" cd:country="USA"
   cd:company="Columbia" cd:price="10.90"
   cd:year="1985" />

</rdf:RDF>
```

and these are the triples and the graph:

Number	Subject	Predicate	Object
1	http://www.w3.org/	http://purl.org/dc/elements/1.1/title	"World Wide Web Consortium"
2	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#artist	"Bob Dylan"
3	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#country	"USA"
4	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#company	"Columbia"
5	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#price	"10.90"
6	http://www.recshop.fake/cd/Empire Burlesque	http://www.recshop.fake/cd#year	"1985"

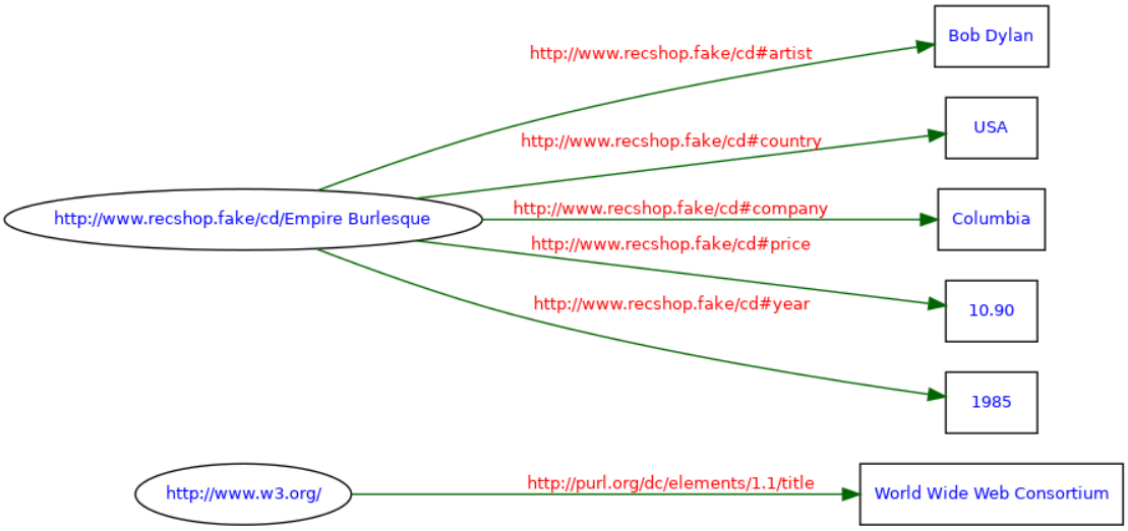


Figure 4: W3C RDF Validator export

It is very helpful since it also recognizes the line and the type of error when the parsing fails as in the example below.

FatalError: Element type "rdf:Description" must be followed by either attribute specifications, ">" or "/>".[Line = 7, Column = 4]

The only major limitations of this parser are that it can only read RDF documents and that neither triples nor graphs can be exported in any way. This ties in with the fact that this validator is only presented on the W3C site as a support tool for those entering the semantic web area. W3C also provides the source code (developed in Java) of this validator even if we don't consider it can be useful. It is difficult to integrate with other systems since the parser takes as input an HTTP request with many parameters that are not detailed in the code and that make difficult the extraction of the parsing section. Moreover, we need an RDF/XML validator that is not available. In the end, this validator can be used only for didactic purposes.

I.III.II RDFLib

RDFLib is a pure Python package for working with RDF. RDFLib contains useful APIs for working, including:

- **Parsers & Serializers:** for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata and JSON-LD, via a plugin module.
- **Store implementations:** for in-memory and persistent RDF storage - Berkeley DB.
- **Graph interface:** to a single graph or a conjunctive graph (multiple Named Graphs) or a dataset of graphs.
- **SPARQL 1.1 implementation**: supporting both Queries and Updates.

This library provides utilities for loading and saving RDFs, creating triples, navigating graphs and querying with SPARQL. The full documentation is available here [14]. The code is available here [15] and can be easily imported in a Python IDE using the command import rdflib.
RDFLib allows you to manipulate ontologies very smoothly. The Gr@phBRAIN system, however, was developed in Java and therefore using this library would first require a linking phase between the two languages.

I.III.III Apache Jena

Apache Jena is a free and open-source Java framework for building Semantic Web and Linked Data Applications. It can interact with the core API to create and read Resource Description Framework (RDF) graphs and serialize triples using popular formats such as RDF/XML or Turtle. It gives the possibility of querying your RDF data using ARQ, a SPARQL 1.1 compliant engine and reason over your data to expand and check the content of your triple store. [16] The full documentation of Apache Jena is available here. [17]
For using Apache Jena APIs, you need to download the jar available here [18] and then import the library in an Eclipse (or whatever) project.

I.III.IV OWL API

The OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL Ontologies. The latest version of the API is focused on OWL 2. The OWL API is open source and is available under either the LGPL or Apache Licenses. The OWL API includes the following components: an API for OWL 2 and an efficient in-memory reference implementation, RDF/XML parser and writer, OWL/XML parser and writer OWL functional syntax parser and writer, Turtle parser and writer, KRSS parser, OBO Flat file format parser and reasoner interfaces for working with reasoners such as FaCT++, HermiT, Pellet and Racer. OWL API provides mechanisms to combine reasoning with explanation processes. Unluckily, at the state of art, the explanations mechanism is not well documented and enriched. For this reason, explanations are not always available (e.g., in some inconsistency problems).

I.III.V Neo4j Import

Neo4j also has libraries for the automatic import of files in different formats, including OWL/RDF (the one we need). To use these libraries, you need to import the jar file *neosemantics* that allows import but also manipulation functions. The import can also be customized on some parameters such as attribute language or multivalue attribute management. The execution time is drastically reduced. This is the command for importing:

```
CALL n10s.onto.import.fetch("https://github.com/neo4j-labs/neosemantics/raw/3.5/docs/rdf/vw.owl","Turtle");
```

The result is a small ontology of cars regarding Volkswagen and SUV.

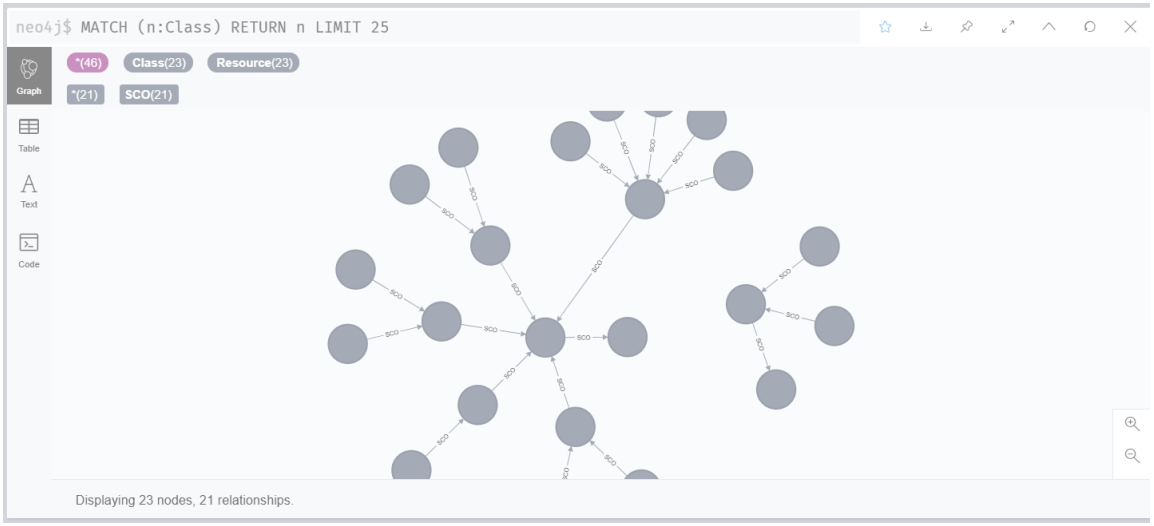


Figure 5: Graph in Neo4j

There are no drawbacks to using this call, except that you cannot apply the translations and transformations described above. For this reason, you might consider a hybrid approach where you perform the necessary manipulations directly on the file and then use the Neo4j import.

Here, we report an example of import with Neo4j.
We will use the same ontology already mentioned and published here [19]. The command for the import is the following one:

```
CALL
n10s.rdf.import.fetch("file:///C:\Users\ddipi\Desktop\Davide\Universita\Tesi\Hotel.rdf","RDF/XML")
```

These are all the class and relationships nodes.

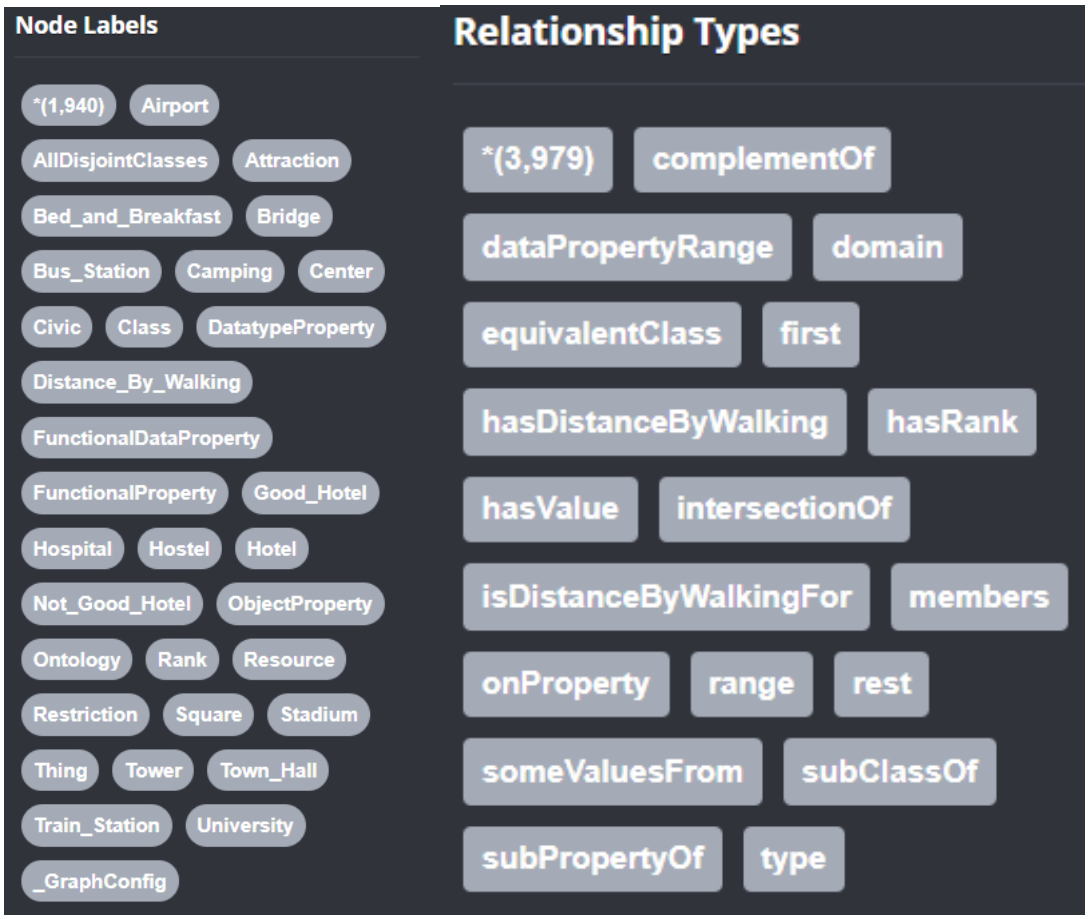


Figure 6: Nodes and relationships in Neo4j

Here, we report an example of some Hotel nodes.

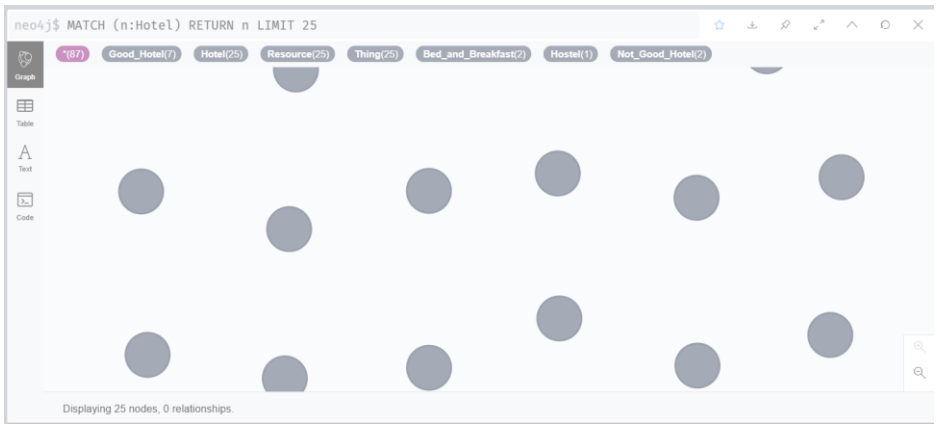


Figure 7: Hotel nodes in Neo4j

Here, we report some examples of hasDistance relationship nodes.

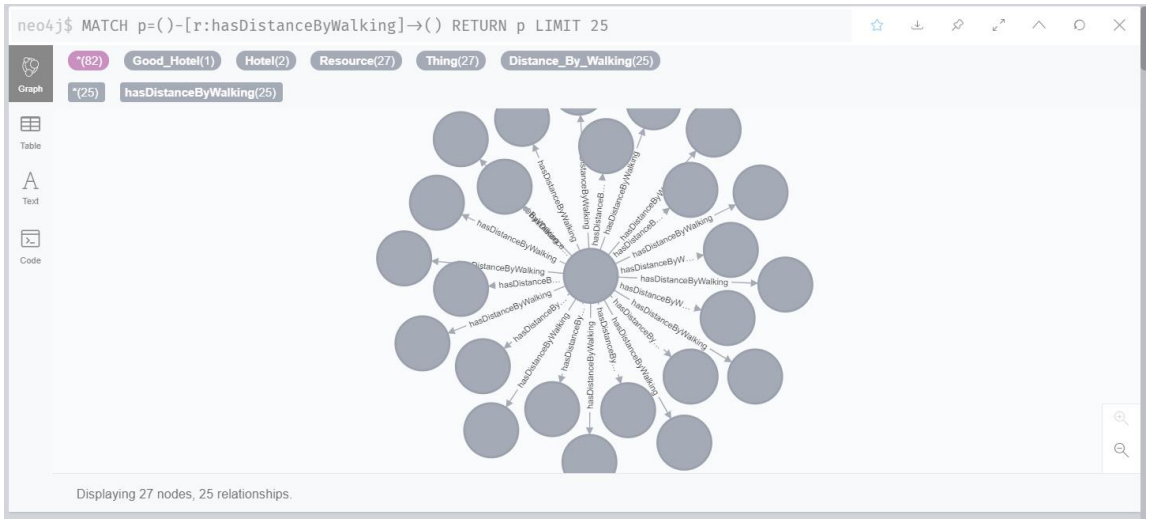


Figure 8: hasDistance nodes in Neo4j

As you can see in the figure below, when the ontology is loaded in form of a graph, there is the problem that selecting the node of a class, properties do not appear since they do not have filled. Properties of a class can be seen selecting individuals. For this reason, it is necessary to import ontologies differently.

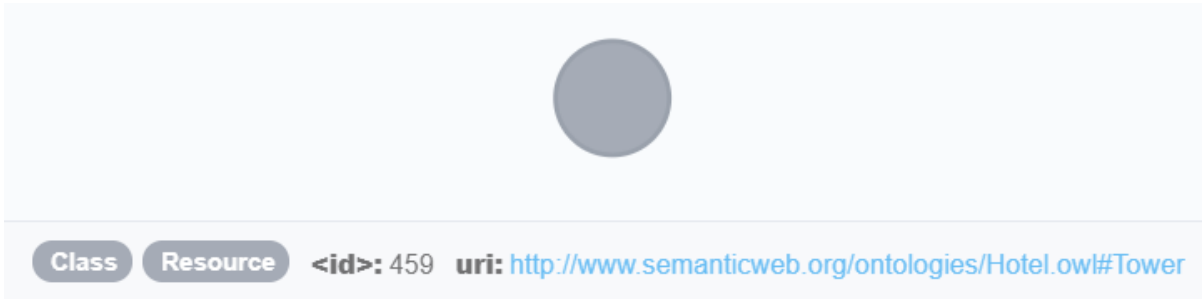


Figure 9: Node of the Hotel class

Figure 10: Node of a Hotel instance

There is also the possibility to export pieces of ontology in some formats (CSV, PNG, CSV and JSON). As an example, we will report the export of all the hotel instances in CSV format.

```
n
"{""comment"":Via Filippo Mazzei 2 pisa,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_001,""hasPrice"":90}"
"{""comment"":Hotel Grand Bonanno,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_014,""hasPrice"":81}"
"{""comment"":AC Hotel Pisa by Marriott,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_013,""hasPrice"":115}"
"{""comment"":Hotel Abitalia Tower Plaza,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_016,""hasPrice"":95}"
"{""comment"":Hotel Golden Tulip Galilei,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_015,""hasPrice"":84}"
"{""comment"":Hotel NH Cavalieri,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_018,""hasPrice"":88}"
"{""comment"":Hotel Santa Croce Fossabanda,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_017,""hasPrice"":63}"
"{""comment"":Hotel La Pace,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_019,""hasPrice"":79}"
"{""comment"":Hotel Royal Victoria,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_021,""hasPrice"":49}"
"{""comment"":Hotel Relais dei Fiori,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_020,""hasPrice"":80}"
"{""comment"":Pisa Delfo,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_023,""hasPrice"":45}"
"{""comment"":B&B Hotel Pisa,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_022,""hasPrice"":59}"
"{""comment"":Hotel Roma,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_003,""hasPrice"":88}"
"{""comment"":Hotel Roseto,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_002,""hasPrice"":85}"
"{""comment"":Hotel Casa Doina,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_005,""hasPrice"":49}"
"{""comment"":Hotel Relais dell'Orologio,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_004,""hasPrice"":136}"
"{""comment"":Hotel Milano,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_007,""hasPrice"":62}"
"{""comment"":Hotel Bologna,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_006,""hasPrice"":99}"
"{""comment"":Hotel Repubblica Marinara,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_009,""hasPrice"":74}"
"{""comment"":Hotel Capitol,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_008,""hasPrice"":75}"
"{""comment"":Hotel Alessandro Della Spina,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_010,""hasPrice"":79}"
"{""comment"":Hotel Amalfitana,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_012,""hasPrice"":69}"
"{""comment"":Hotel Athena,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_011,""hasPrice"":63}"
"{""comment"":Residence Isola Verde,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_036,""hasPrice"":60}"
"{""comment"":Villa Kinzica,""uri"":http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_035,""hasPrice"":82}"
```

Figure 11: Neo4j export in CSV

I.III.VI Implementation

Here we show the implementation examples for each tool.

I.III.VI.I RDFLib

We report a simple example using a well-known Hotel ontology in the field of Knowledge Representation and Reasoning, mentioned here [19].

We show a simple example in which we import a rdf/xml file, transform the KB into triples, print triples and then export the KB in XML.

```
@author: ddipi
"""

import rdflib

# create a Graph
g = rdflib.Graph()
f = open("C:/Users/ddipi/Desktop/Davide/Universita/Tesi/triples.txt", "w")

# parse in an RDF file hosted on the Internet
g.parse("C:/Users/ddipi/Desktop/Davide/Universita/Tesi/exampleHotel.rdf")

# loop through each triple in the graph (subj, pred, obj)
for subj, pred, obj in g:
    f.write("Subj : " + str(subj) + ", Pred : " + str(pred) + ", Obj : " + str(obj) + "\n")
f.close()

# print the number of "triples" in the Graph
print("graph has {} statements.".format(len(g)))
# prints graph has 86 statements.

# print out the entire Graph in the XML format
g.serialize("exampleHotel2.owl", format="xml")
```

Figure 12: Example of an RDFLib application

This is an example of two triples found in the Knowledge Base.

Subj : http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_044, Pred : http://www.semanticweb.org/ontologies/Hotel.owl#hasDistanceByWalking, Obj : http://www.semanticweb.org/ontologies/Hotel.owl#d44_21

Subj : http://www.semanticweb.org/ontologies/Hotel.owl#d32_3, Pred : http://www.w3.org/1999/02/22-rdf-syntax-ns#type, Obj : http://www.semanticweb.org/ontologies/Hotel.owl#Distance_By_Walking

And this is a portion of the export.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns="http://www.semanticweb.org/ontologies/Hotel.owl#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
  <rdf:Description rdf:about="http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_044">
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_6"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_22"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_7"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_18"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_13"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_10"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_21"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_23"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_9"/>
    <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#Hotel"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_8"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_2"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_4"/>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_14"/>
    <rdfs:comment>Hotel Villa Primavera</rdfs:comment>
    <hasDistanceByWalking rdf:resource="http://www.semanticweb.org/ontologies/Hotel.owl#d44_1"/>
    <rdf:type rdf:nodeID="Nb73e277c3be9492da76f40a30fc059b1"/>
```

Figure 13: RDFLib export in RDF/XML

I.III.VI.II Apache Jena

We show a brief example of the main functions provided by these APIs.

In the example below, we imported the same Hotel ontology as before and we extracted the property hasPrice which is a data property for all the accommodations in the ontology, so every hotel has this property. We read all classes and export to a file all the classes and individuals of Hotel. In the end, we exported the ontology into triples and print the number of individuals of type Hotel which have the property hasPrice.

```java
import java.io.FileWriter;
import java.io.IOException;
import org.apache.jena.ontology.Individual;
import org.apache.jena.ontology.OntClass;
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntProperty;
import org.apache.jena.ontology.OntResource;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.util.iterator.ExtendedIterator;

public class Main {

    public static void main(String[] args) throws IOException {
        OntModel model = ModelFactory.createOntologyModel();
        model.read("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\exampleHotel.rdf", "RDFXML");

        ExtendedIterator<OntProperty> properties = model.listAllOntProperties();
        Property p = null;
        while (properties.hasNext()) {
            Property next = (Property) properties.next();
            if (next.toString().contains("hasPrice")) {
                p = next;
            }
        }
        System.out.println(p);

        ExtendedIterator<OntClass> classes = model.listClasses();

        int countHasPrice = 0;
        FileWriter myWriter = new FileWriter("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\exampleHotel.txt");
        while (classes.hasNext()) {
          OntClass thisClass = (OntClass) classes.next();
          myWriter.write("Found class: " + thisClass.toString() + "\n");
          if(thisClass.toString().contains("#Hotel")) {
              ExtendedIterator<? extends OntResource> instances = thisClass.listInstances();
              while (instances.hasNext()) {

                Individual thisInstance = (Individual) instances.next();
                if(thisInstance.hasProperty(p)) {
                    countHasPrice++;
                }

                myWriter.write("Found instance: " + thisInstance.toString() + "\n");
              }
          }
        }
        myWriter.close();
        System.out.println("Number of hotels with price : " + countHasPrice);
        model.write(System.out, "N-TRIPLES");
    }

}
```

Figure 14: Example of an Apache Jena application

This is a portion of the text file reporting all classes and Hotel instances.

Found class: http://www.semanticweb.org/ontologies/Hotel.owl#Table_Tennis

Found class: http://www.semanticweb.org/ontologies/Hotel.owl#Tower

Found class: http://www.semanticweb.org/ontologies/Hotel.owl#Hospital

Found class: http://www.semanticweb.org/ontologies/Hotel.owl#Hotel

Found instance: http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_056

Found instance: http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_014

Found instance: http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_032
Found instance: http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_028

This is a portion of triples produced for the Hotel ontology.

<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_046>
<http://www.semanticweb.org/ontologies/Hotel.owl#hasDistanceByWalking>
<http://www.semanticweb.org/ontologies/Hotel.owl#d46_18> .

<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_046>
<http://www.semanticweb.org/ontologies/Hotel.owl#hasDistanceByWalking>
<http://www.semanticweb.org/ontologies/Hotel.owl#d46_12> .

<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_046>
<http://www.semanticweb.org/ontologies/Hotel.owl#hasDistanceByWalking>
<http://www.semanticweb.org/ontologies/Hotel.owl#d46_23> .

This is the number of Hotel individuals with the hasPrice property.

Number of hotels with price : 59

I.III.VI.III OWL API

We report a small example in which we import the Pizza ontology using OWL API. The ontology is one of the most famous and it is available here [20].

```
public class ExplanationsOriginal {

    public static void main(String[] args) throws Exception {
        OWLOntologyManager manager=OWLManager.createOWLOntologyManager();
        OWLDataFactory dataFactory=manager.getOWLDataFactory();
        File inputOntologyFile = new File("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\Import RDF\\Pizza.owl");
        OWLOntology ontology=manager.loadOntologyFromOntologyDocument(inputOntologyFile);
    }
}
```

Figure 15: OWL API implementation to import ontologies

Further operations will be described later.

I.IV Reasoning

Reasoning in ontologies and knowledge bases is one of the reasons why a specification needs to be a formal one. By reasoning, we mean deriving facts that are not expressed in the ontology or the knowledge base explicitly. Description logics are created with the focus on tractable reasoning. A few examples of tasks required from the reasoner are as follows.

- Satisfiability of a concept - determine whether a description of the concept is not contradictory, i.e., whether an individual can exist that would be an instance of the concept.
- Subsumption of concepts - determine whether concept C subsumes concept D, i.e., whether the description of C is more general than the description of D.
- Consistency of ABox with respect to TBox - determine whether individuals in ABox do not violate descriptions and axioms described by TBox.
- Check an individual - check whether the individual is an instance of a concept.
- Retrieval of individuals - find all individuals that are instances of a concept.
- Realization of an individual - find all concepts to which the individual belongs, especially the most specific ones.

These tasks are not semantically very different. For example, satisfiability can be tested as subsumption of ⊥-concept is unsatisfiable if no individual can exist that would be an instance of the concept. For all tasks, it is enough to be able to check deductive consequence or derive all deductive consequences of a theory. However, there may be specially optimized algorithms for different tasks in a reasoner.

All operations are decidable. Even when the theoretical complexity seems to be intractable, there are optimized reasoners available that are usable for practical real-world cases.

I.IV.I Pellet

Pellet started as a proof-of-concept system to help meet theW3C's implementation experience requirements for the Web Ontology Language. It has since become a practical and popular tool for working with OWL. Pellet has been the first reasoner to support all of OWL-DL, i.e., the Description Logic. It offers a panoply of features including conjunctive query answering, rule support, E-Connection reasoning, and axiom pinpointing, among others. To make its reasoning capabilities easily accessible to users, Pellet provides various interfaces including a command-line interface, an interactive Web form for zero-install use, DIG server implementation, and API bindings for RDF/OWL toolkits Jena and Manchester OWL-API.

The figure below shows the main components of Pellet. At its core, it is a Description Logic reasoner based on tableaux algorithms. The tableaux reasoner checks the consistency of a knowledge base and all the other reasoning services are reduced to consistency checking. The reasoner is designed so that different tableaux algorithms can be plugged in. The default algorithm handles SROIQ(D) but there are several other tableaux algorithms implemented, e.g., for non-monotonic extensions. Pellet implements most of the state-of-the-art optimization techniques provided in the DL literature including normalization, simplification, absorption, semantic branching, backjumping, caching satisfiability status, top-bottom search for classification, and model merging. In addition, Pellet incorporates several novel optimizations to improve the reasoning performance in the presence of nominals (enumerated classes) and individuals. Reasoning with nominals is especially challenging since some of the

existing optimizations in DL reasoners are not applicable anymore. Moreover, in the presence of nominals, assertions about instances can affect the concept satisfiability and classification results. A suite of new optimizations to tackle this problem has been developed. These optimizations prove sufficient to handle even the notoriously difficult Wine ontology, and, indeed, all ontologies with nominals they encountered. Another novel optimization technique implemented in Pellet is for incremental reasoning against dynamic knowledge bases. In many contexts, the knowledge base is in constant flux. They have developed techniques to reuse the reasoning results from previous steps to process updates incrementally. Conjunctive ABox query Pellet includes a query engine that can efficiently answer conjunctive ABox queries expressed in SPARQL or RDQL. Datatype reasoning Pellet uses the type of system approach to support reasoning with datatypes. Pellet has a datatype oracle that can reason with XML Schema-based datatypes. The datatype oracle can check the consistency of conjunctions of XML Schema datatypes. Pellet supports user derived types based on numeric or date/time types so, for example, numeric or date/time intervals can be defined and used as new datatypes. [21]

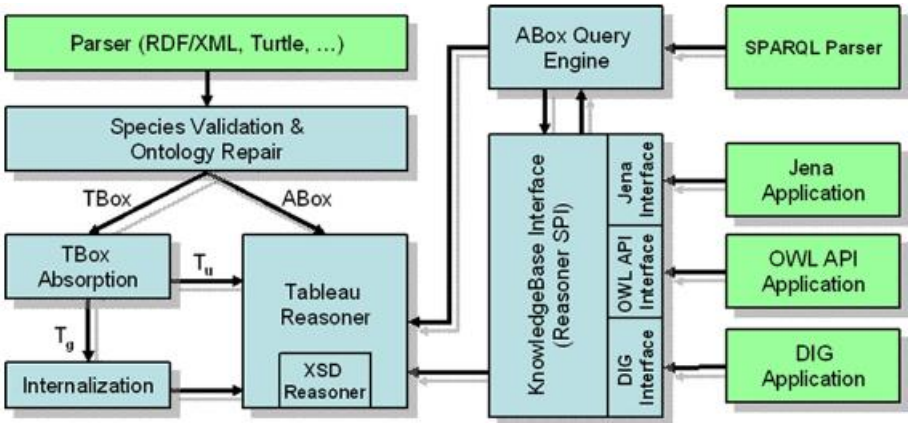


Figure 16: Pellet's architecture

I.IV.II HermiT

HermiT is a Description Logic reasoning system based on an entirely new architecture which addresses the sources of complexity. HermiT implements a "hypertableau" calculus which greatly reduces the number of possible models which must be considered (down to only a single possibility for a significant subset of ontologies). HermiT also incorporates the "anywhere blocking" strategy, which limits the sizes of models which are constructed. Finally, HermiT makes use of a novel and highly efficient approach to handling nominals in the presence of number restrictions and inverse roles; we expect that this will allow ontology authors to make much freer use of nominals than has been possible to date. This combination of fundamental algorithmic improvements also enables a range of additional optimizations. Our tests show that HermiT is as fast as other DL reasoners when classifying relatively easy-to-process ontologies, and usually much faster when classifying more difficult ontologies. In fact, HermiT can classify several ontologies which no other reasoner has previously been able to handle. The HermiT system also serves as a platform for prototypical implementations of new language features. For example, HermiT already includes support for reasoning with ontologies which include description graphs.

On OWL ontology O can be divided into three parts: the property axioms, the class axioms, and the facts. These correspond to the RBox R, TBox T , and ABox A of a Description Logic knowledge base. 2.1 Reducing Tableau Complexity To show that a knowledge base K = (R, T , A) is satisfiable, a tableau algorithm constructs a derivation—a sequence of ABoxes A0, A1, . . . , An, where A0 = A and each Ai is obtained from Ai−1 by an application of one inference rule. The inference rules make the information implicit in the axioms of R and T explicit, and thus evolve the ABox A towards a (representation of a) model of K. The algorithm terminates either if no inference rule applies to some An, in which case

An represents a model of K, or if An contains an obvious contradiction, in which case the model construction has failed. The following inference rules are commonly used in DL tableau calculi. – ⊔-rule: Given (C1 ⊔ C2)(s), derive either C1(s) or C2(s). – ⊓-rule: Given (C1 ⊓ C2)(s), derive C1(s) and C2(s). – ∃-rule: Given (∃R.C)(s), derive R(s, t) and C(t) for t a fresh individual. – ∀-rule: Given (∀R.C)(s) and R(s, t), derive C(t). – ⊑-rule: Given an axiom C ⊑ D and an individual s, derive (¬C ⊔ D)(s). The ⊔-rule is nondeterministic: if (C1 ⊔ C2)(s) is true, then C1(s) or C2(s) or both are true. Therefore, tableau calculi make a nondeterministic guess and choose either C1 or C2 If choosing C1 leads to a contradiction, the algorithm must backtrack and try C2; this procedure is known as reasoning by case. The knowledge base K is unsatisfiable if and only if all choices fail to construct a model. HermiT includes some nonstandard functionality that is currently not available in any other system. In particular, it supports reasoning with ontologies containing description graphs. As shown in [8], description graphs allow for the representation of structured objects—objects composed of many parts interconnected in arbitrary ways. These objects abound in bio-medical ontologies such as FMA and GALEN, but they cannot be faithfully represented in OWL. HermiT shows significant performance advantages over other reasoners across a wide range of real-world ontologies. In several cases, it is able to classify ontologies that no other reasoner can process and also includes support for some non-standard ontology features, such as description graphs.

I.IV.III Bossam

Bossam is an inference engine for the Semantic Web. It is basically a RETE-based rule engine with native supports for reasoning over OWL ontologies, SWRL ontologies, and RuleML rules. Additionally, Bossam includes several expressivity features including:

- URI references as symbols,

- 2nd-order logic syntax,
- disjunctions in the antecedent and conjunctions in the consequent (both via Lloyd-Topor transformation),
- URI-based java method attachment,
- support for both negation-as-failure and classical negation.

You can use Bossam for loading, inferencing, and querying over the set of documents. Also, you can call Java objects from the antecedent or consequent of rules through the URI-based java method attachment, thus enabling you to mix Java objects into the combination of rules and ontologies. [22]

I.IV.IV Reasoning in Apache Jena

Through reasoning mechanisms, we can provide value to ontologies that are imported or constructed. The goal is to produce new, non-trivial axioms that yield previously unknown results. We will go into details about how Apache Jena provides reasoning.

The Jena inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. The primary use of this mechanism is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions. However, the machinery is designed to be quite general and, in particular, it includes a generic rule engine that can be used for many RDF processing or transformation tasks.

Included in the Jena distribution are several predefined reasoners:

- **Transitive reasoner**: Provides support for storing and traversing class and property lattices. This implements just

the transitive and reflexive properties
of rdfs:subPropertyOf and rdfs:subClassOf.

- **RDFS rule reasoner**: Implements a configurable subset of the RDFS entailments.
- **OWL, OWL Mini, OWL Micro Reasoners**: A set of useful but incomplete implementation of the OWL/Lite subset of the OWL/Full language.
- **Generic rule reasoner**: A rule-based reasoner that supports user-defined rules. Forward chaining, tabled backward chaining and hybrid execution strategies are supported.

I.IV.IV.I Generic Reasoner API

For each type of reasoner, there is a factory class (which conforms to the interface ReasonerFactory), an instance of which can be used to create instances of the associated Reasoner. The behaviour of many of the reasoners can be configured. To allow arbitrary configuration information to be passed to reasoners we use RDF to encode the configuration details. Once you have an instance of a reasoner it can then be attached to a set of RDF data to create an inference model. This can either be done by putting all the RDF data into one Model or by separating it into two components - schema and instance data. For some external reasoners, a hard separation may be required. For all of the built-in reasoners, the separation is arbitrary. Finally, having created an inference model, any API operations which access RDF statements will be able to access additional statements which are entailed from the bound data through the reasoner. Depending on the reasoner these additional *virtual* statements may all be precomputed the first time the model is touched, may be dynamically recomputed each time or may be computed on-demand but cached.

The most common reasoner operation which can't be exposed through additional triples in the inference model is that of validation. Typically, the ontology languages used with the semantic web allow constraints to be expressed, the

validation interface is used to detect when such constraints are violated by some data set. A simple but typical example is that of datatype ranges in RDFS. RDFS allows us to specify the range of a property as lying within the value space of some datatypes. If an RDF statement asserts an object value for that property which lies outside the given value space, there is an inconsistency.

It is sometimes useful to be able to trace where an inferred statement was generated from. This is achieved using the InfModel.getDerivation(Statement) method. This returns an iterator over a set Derivation objects through which a brief description of the source of the derivation can be obtained. Typically understanding this involves tracing the sources for other statements which were used in this derivation and the Derivation.PrintTrace method is used to do this recursively.

I.IV.IV.II The RDFS Reasoner

Jena includes an RDFS reasoner (RDFSRuleReasoner) which supports almost all of the RDFS entailments described by the RDF Core working group.

The RDFSRuleReasoner can be configured to work at three different compliance levels:

- **Full:** This implements all of the RDFS axioms and closure rules except for bNode entailments and datatypes (rdfD 1). This is an expensive mode because all statements in the data graph need to be checked for the possible use of container membership properties. It also generates type assertions for all resources and properties mentioned in the data (rdf1, rdfs4a, rdfs4b).
- **Default:** This omits the expensive checks for container membership properties, the "everything is a resource" and "everything used as a property is one" rules (rdf1, rdfs4a, rdfs4b). The latter information is available through the Jena API and creating virtual triples to this effect has

little practical value.

This mode does include all the axiomatic rules. Thus, for example, even querying an "empty" RDFS InfModel will return triples such as [rdf:type rdfs:range rdfs:Class].

- **Simple:** This implements just the transitive closure of subPropertyOf and subClassOf relations, the domain and range entailments and the implications of subPropertyOf and subClassOf. It omits all of the axioms. This is probably the most useful mode but is not the default because it is a less complete implementation of the standard.

I.IV.IV.III The OWL Reasoner

The second major set of reasoners supplied with Jena is a rule-based implementation of the OWL/lite subset of OWL/full.

The current release includes a default OWL reasoner and two small/faster configurations. Each of the configurations is intended to be a sound implementation of a subset of OWL/full semantics but none of them is complete (in the technical sense). For complete OWL DL reasoning use an external DL reasoner such as Pellet, Racer or FaCT. Performance (especially memory use) of the fuller reasoner configuration still leaves something to be desired and will the subject of future work - time permitting.

The Jena OWL reasoners could be described as instance-based reasoners. That is, they work by using rules to propagate the if- and only-if- implications of the OWL constructs on instance data. Reasoning about classes is done indirectly - for each declared class a prototypical instance is created and elaborated. If the prototype for a class A can be deduced as being a member of class B then we conclude that A is a subClassOf B. This approach is in contrast to more sophisticated Description Logic reasoners which work with class expressions and can be less efficient when handling instance data but more efficient with complex class expressions and able to provide complete reasoning.

I.IV.IV.IV The Transitive Reasoner

The TransitiveReasoner provides support for storing and traversing class and property lattices. This implements just the transitive and symmetric properties of rdfs:subPropertyOf and rdfs:subClassOf. It is not all that exciting on its own but is one of the building blocks used for the more complex reasoners. It is a hardwired Java implementation that stores the class and property lattices as graph structures. It is slightly higher performance, and somewhat more space-efficient, than the alternative of using the pure rule engines to perform transitive closure but its main advantage is that it implements the direct/minimal version of those relations as well as the transitively closed version.

I.IV.IV.V The General-purpose Rule Engine

Jena includes a general-purpose rule-based reasoner which is used to implement both the RDFS and OWL reasoners but is also available for general use. This reasoner supports rule-based inference over RDF graphs and provides forward chaining, backward chaining, and a hybrid execution model.

The various engine configurations are all accessible through a single parameterized reasoner GenericRuleReasoner. At a minimum, a GenericRuleReasoner requires a ruleset to define its behaviour.

A GenericRuleReasoner instance with a ruleset can be used like any of the other reasoners described above - that is it can be bound to a data model and used to answer queries to the resulting inference model.

The rule reasoner can also be extended by registering new procedural primitives. The current release includes a starting set of primitives which are sufficient for the RDFS and OWL implementations but is easily extensible.

I.IV.IV.VI Forward Chaining

If the rule reasoner is configured to run in forward mode, then only the forward chaining engine will be used. The first time the inference Model is queried then all of the relevant data in the model will be submitted to the rule engine. Any rules which fire that create additional triples do so in an internal deductions graph and can in turn trigger additional rules. There is a remove primitive that can be used to remove triples and such removals can also trigger rules to fire in removal mode. This cascade of rule firings continues until no more rules can fire. It is perfectly possible, though not a good idea, to write rules that will loop infinitely at this point.

I.IV.IV.VII Backward Chaining

If the rule reasoner is run in backward chaining mode it uses a logic programming (LP) engine with a similar execution strategy to Prolog engines. When the inference Model is queried then the query is translated into a goal and the engine attempts to satisfy that goal by matching to any stored triples and by goal resolution against the backward chaining rules.

Except as noted below rules will be executed in top-to-bottom, left-to-right order with backtracking, as in SLD resolution. In fact, the rule language is essentially datalog rather than full prolog, whilst the functor syntax within rules does allow some creation of nested data structures, they are flat (not recursive) and so can be regarded a syntactic sugar for datalog.

I.IV.IV.VIII Hybrid Rule Engine

The rule reasoner has the option of employing both of the individual rule engines in conjunction. When run in this hybrid mode the data flows look something like this:

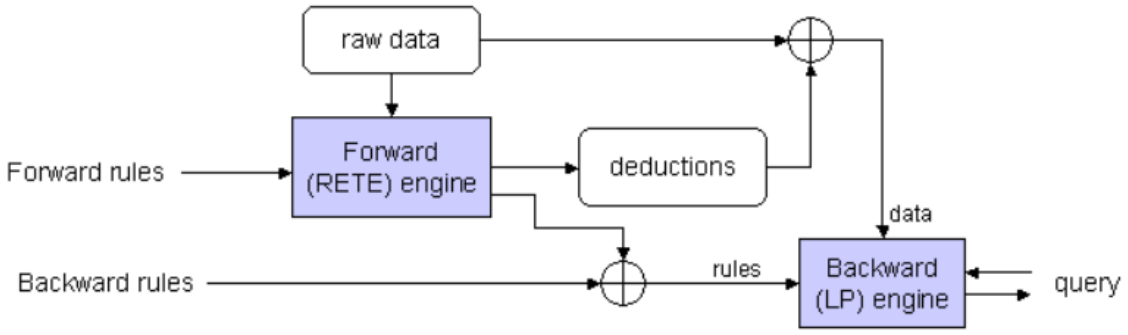


Figure 17: Hybrid rule engine's architecture

The forward engine runs as described above and maintains a set of inferred statements in the deductions store. Any forward rules which assert new backward rules will instantiate those rules according to the forward variable bindings and pass the instantiated rules on to the backward engine.
The full documentation is available here [23].

I.IV.V Reasoning in OWL API

OWL API provides reasoning mechanisms through the OWLReasoner interface.

I.IV.V.I OWLReasoner

An OWLReasoner reasons over a set of axioms (the set of reasoner axioms) that is based on the imports closure of a particular ontology - the "root" ontology. This ontology can be obtained using the getRootOntology() method. When the client responsible for creating the reasoner has finished with the reasoner instance it must call the dispose() method to free any resources that are used by the reasoner. In general, reasoners should not be instantiated directly but should be created using the appropriate OWLReasonerFactory.

At creation time, an OWLReasoner will load the axioms in the root ontology imports closure. It will attach itself as a listener to the OWLOntologyManager that manages the root ontology. The reasoner will

listen to any OWLOntologyChanges and respond appropriately to them before answering any queries. If the BufferingMode of the reasoner (the answer to getBufferingMode() is BufferingMode.NON_BUFFERING) the ontology changes are processed by the reasoner immediately so that any queries asked after the changes are answered with respect to the changed ontologies. If the BufferingMode of the reasoner is BufferingMode.BUFFERING then ontology changes are stored in a buffer and are only taken into consideration when the buffer is flushed with the flush() method.
Note that there is no guarantee that the reasoner implementation will respond to changes in an incremental (and efficient manner) manner. The set of axioms that the reasoner takes into consideration when answering queries is known as the set of reasoner axioms. This corresponds to the axioms in the imports closure of the root ontology plus the axioms returned by the getPendingAxiomRemovals() minus the axioms returned by getPendingAxiomAdditions().

I.IV.VI Implementation

We report a small example of the two frameworks for using a reasoner in a Java system.

I.IV.VI.I Reasoning in Apache Jena

In the example, let us first create a Jena model containing the statements that some property "p" is a subproperty of another property "q" and that we have a resource "a" with value "foo" for "p". This could be done by writing an RDF/XML or N3 file and reading that in, but we have chosen to use the RDF API:

```
String NS = "urn:x-hp-jena:eg/";
```

```
// Build a trivial example data set

Model rdfsExample = ModelFactory.createDefaultModel();

Property p = rdfsExample.createProperty(NS, "p");

Property q = rdfsExample.createProperty(NS, "q");

rdfsExample.add(p, RDFS.subPropertyOf, q);

rdfsExample.createResource(NS+"a").addProperty(p, "foo");
```

Now we can create an inference model which performs RDFS inference over this data by using:

```
InfModel inf = ModelFactory.createRDFSModel(rdfsExample);  // [1]
```

We can then check that resulting model shows that "a" also has property "q" of value "foo" by virtue of the subPropertyOf entailment:

```
Resource a = inf.getResource(NS+"a");

System.out.println("Statement: " + a.getProperty(q));
```

Which prints the output:

```
    Statement: [urn:x-hp-jena:eg/a, urn:x-hp-jena:eg/q, Literal<foo>]
```

I.IV.VI.II Reasoning in OWL API

We report an example with also explanations in OWL API.

```
public class ExplanationsOriginal {

    public static void main(String[] args) throws Exception {
        OWLOntologyManager manager=OWLManager.createOWLOntologyManager();
        OWLDataFactory dataFactory=manager.getOWLDataFactory();

        File inputOntologyFile = new File("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\Import RDF\\Pizza.owl");
        OWLOntology ontology=manager.loadOntologyFromOntologyDocument(inputOntologyFile);

        IRI icecreamIRI=IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream");
        OWLClass icecream=dataFactory.getOWLClass(icecreamIRI);

        ReasonerFactory factory = new ReasonerFactory();

        Configuration configuration=new Configuration();
        configuration.throwInconsistentOntologyException=false;
        OWLReasoner reasoner=factory.createReasoner(ontology, configuration);
        System.out.println("Is icecream satisfiable? "+reasoner.isSatisfiable(icecream));
        System.out.println("Computing explanations...");
        BlackBoxExplanation exp=new BlackBoxExplanation(ontology, factory, reasoner);
        HSTExplanationGenerator multExplanator=new HSTExplanationGenerator(exp);
        Set<Set<OWLAxiom>> explanations=multExplanator.getExplanations(icecream);

        for (Set<OWLAxiom> explanation : explanations) {
            System.out.println("------------------");
            System.out.println("Axioms causing the unsatisfiability: ");
            for (OWLAxiom causingAxiom : explanation) {
                System.out.println(causingAxiom);
            }
            System.out.println("------------------");
        }
```

```
        OWLAxiom ax=dataFactory.getOWLClassAssertionAxiom(icecream, dataFactory.getOWLNamedIndividual(IRI.create("http://www.co-ode.org/ontologies/pizza/pizza.owl#dum
        manager.addAxiom(ontology, ax);
        reasoner=factory.createReasoner(ontology, configuration);
        System.out.println("Is the changed ontology consistent? "+reasoner.isConsistent());
        System.out.println("Computing explanations for the inconsistency...");
        factory=new Reasoner.ReasonerFactory() {
            protected OWLReasoner createHermiTOWLReasoner(org.semanticweb.HermiT.Configuration configuration,OWLOntology ontology) {
                // don't throw an exception since otherwise we cannot compte explanations
                configuration.throwInconsistentOntologyException=false;
                return new Reasoner(configuration,ontology);
            }
        };
        exp=new BlackBoxExplanation(ontology, factory, reasoner);
        multExplanator=new HSTExplanationGenerator(exp);
        explanations=multExplanator.getExplanations(dataFactory.getOWLThing());
        System.out.println("Thing : " + dataFactory.getOWLThing());

        for (Set<OWLAxiom> explanation : explanations) {
            System.out.println("------------------");
            System.out.println("Axioms causing the inconsistency: ");
            for (OWLAxiom causingAxiom : explanation) {
                System.out.println(causingAxiom);
            }
            System.out.println("------------------");
        }
    }
}
```

This is the output produced:

Is icecream satisfiable? false

Computing explanations...

Axioms causing the unsatisfiability:

SubClassOf(<http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream>

ObjectSomeValuesFrom(<http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping>

<http://www.co-ode.org/ontologies/pizza/pizza.owl#FruitTopping>))

DisjointClasses(<http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream>

<http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza>)

ObjectPropertyDomain(<http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping>

<http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza>)

Is the changed ontology consistent? false
Computing explanations for the inconsistency...

Axioms causing the inconsistency:
SubClassOf(<http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream>
ObjectSomeValuesFrom(<http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping>
<http://www.co-ode.org/ontologies/pizza/pizza.owl#FruitTopping>))
ObjectPropertyRange(<http://www.co-ode.org/ontologies/pizza/pizza.owl#isToppingOf>
<http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza>)
InverseObjectProperties(<http://www.co-ode.org/ontologies/pizza/pizza.owl#isToppingOf>
<http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping>)
DisjointClasses(<http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream>
<http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza>)
ClassAssertion(<http://www.co-ode.org/ontologies/pizza/pizza.owl#IceCream>
<http://www.co-ode.org/ontologies/pizza/pizza.owl#dummyIndividual>)

I.V Related Works

In the literature, we find several attempts to import, integrate, and possibly correct databases, knowledge bases, or graphs for use as a single knowledge graph. An accurate solution can be found here [6]. In this paper, the correction and integration of triples are performed very effectively. The new knowledge base that is created is also queryable by users. The approach described, however, is by no means general or usable in any field other than biology because of how it integrates the various nodes and URIs chosen. Despite this, it can provide interesting insights in cases where similar techniques can be reused. In [24] is not used a generic KG but WordNet. WordNet is a large electronic lexical database for English [25]. The approach involves scanning this knowledge base and using an RNN network to classify words. At the end of the process, in the knowledge graph, each word is tied with the label produced by the neural network. In [26] is described a great system to collect as much information as possible in the field of biodiversity. Data are taken from articles and journals. The extraction software was developed in R for this project [27]. The new graph is being made publicly

available as Linked Open Data and is specific to that field. In [28] a process is described in which transformation operations are performed on the graph.
What is relevant for us is the import phase which is done through two possible libraries: RDFLib in Python or Apache Jena in Java. These two technologies will be described later.

I.V.I Gr@phBRAIN

Gr@phBRAIN is an ontology management system to create, extend, importing, and modifying ontologies from different sources.
Ontology refers to the study of the existence of any kind of entity (concrete or abstract) that is part of our conceptualization of the world. It aims to classify things that exist through the words that describe them. The study of ontology stems from philosophy as an area of metaphysics that studies how the universe around us is made. In computer science, particularly in the area of Artificial Intelligence, the aim is to create a model of the reality of interest. From a formal point of view, an ontology is a triple (C, R, A) in which:

- C is a set of concepts.
- R is a set of relationships among concepts.
- A is a set of axioms on which the universe we want to describe is based.

R is therefore a subset of the Cartesian product C x C. An ontology can also be devoid of axioms, in which case it is called non-axiomatized.
The fact that any ontology can be described by concepts and relations has meant that graphs are often used to describe them. A graph is a pair (V, E) where V represents the set of vertices, in our case the concepts, while E is the set of arcs joining two vertices. In this case, each pair in R is represented through an arc in E from one vertex V to another.

From this model, networks are born to represent ontologies. They represent an oriented graph in which each arc is labelled, and the label is, in a very intuitive way, the relationship between the two entities. We report here a small example.

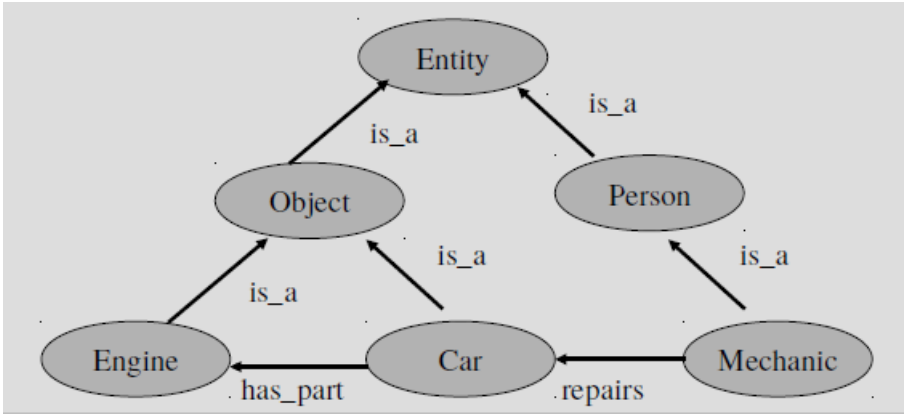


Figure 18: Ontology example

The system is available online at http://193.204.187.73:8088/GraphBRAIN/ and, at present, it is possible to choose one of the possible available domains (food, general, lam, retro-computing and tourism) and visualize entities, relations and a graph of nodes.

The goal is to encapsulate as much knowledge about the world as possible, and this is accomplished by the users themselves who can easily register and enter new information. There is also a Hall of Fame divided by domains where you can see who the major contributors to the service are; it is also designed to become a tool to store the reliability of each cooperating user.

You can learn about the different entities, see their values and insert new ones by specifying some mandatory attributes (usually at least the name). For each entity, you can also identify its relationships. The structure of the relations is very simple since they have: name, subject and object. The subject represents the orientation of the relationship or the node that starts the arc in the case of networks. Every relation has an inverse one in which subject and object are obviously reversed.

Entities and relationships can also be described by attributes that add additional information.

Gr@phBRAIN is based on a graph database that is much more efficient than a relational one, especially in the case of a large amount of data. There is also a section where you can identify in a graphical way some nodes and relations expressed through labelled nodes and arcs. In Figure 4 we report a small portion of the KG.

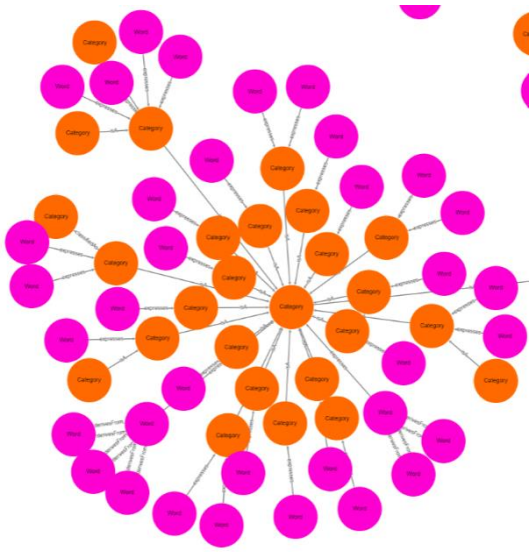


Figure 19: Portion of Gr@phBRAIN Knowledge Graph

I.V.II A Knowledge Graph from arCo

An attempt to combine a knowledge base with knowledge graphs has been made by students of the University of Bari Aldo Moro with the arCo graph. ArCo is the Knowledge Graph of the Italian Cultural Heritage: it consists of 7 vocabularies describing the cultural heritage domain and data from the General Catalogue of the Italian Ministry of Cultural Heritage and Activities (MiBAC) published as RDF. [29]

The study aims to develop a plugin able to retrieve the knowledge present in a specific triple dataset (in this case arCo) and to transfer it to a Prolog knowledge base. To reach this final point, different steps have been performed.

First, it is necessary to read and understand the input triples. Second, a matching is useful to keep a correspondence with another ontology. Third, the triples according to the new ontology are imported into a graph database (in this case Neo4j). Finally, starting from the knowledge graph, some knowledge is extracted and imported into a Prolog knowledge base useful to make inferences.

This project is based on building a very huge knowledge graph, which represents all the information present in arCo. In general, a knowledge graph is a collection of interlinked entities (real-world objects and events, or abstract concepts), where descriptions have formal semantics that allow both people and computers to process them efficiently and unambiguously. Entity descriptions contribute to one another, forming a network. Ontologies represent the backbone of the formal semantics of a knowledge graph: they can be seen as the data schema of the graph. Commonly in computer science, an ontology includes a representation, formal naming and definition of the categories, properties and relations between the concepts, data and entities that validate one, many or all domains of discourse.

A knowledge graph acquires and integrates information into an ontology and may apply a reasoner to derive new knowledge. In other words, a knowledge graph is a way to model a knowledge domain with the help of experts, data interlinking, and machine learning algorithms. A knowledge graph is typically built on top of the existing databases to link all together both structured and unstructured (articles) information.

Knowledge graphs are at the core of many of the tools that we use in our daily lives, such as voice assistants, search applications and even online store recommenders. For example, the Google Knowledge Graph is a knowledge base used by Google to enhance its search engine results with information gathered

from a variety of sources: the information is presented to users in an "infobox" next to the search results. Not only internet giants but also other companies (such as BBC and Electronic Arts) have already integrated the technology and are using knowledge graphs to exploit the power of all data they have accumulated over the years.

In this study, the programming language chosen to build the knowledge graph and to extract some interesting knowledge is Java, because it can easily interact with a Neo4j graph database and provide simple interfaces useful to connect to it.

Unlike traditional databases, which arrange data in rows, columns and tables, Neo4j has a flexible structure defined by stored relationships between data records. With Neo4j, each data record, or node, stores direct pointers to all the nodes it's connected to. Because Neo4j is designed around this simple, yet powerful optimization, it performs queries with complex connections orders of magnitude faster, and with more depth, than other databases.

Some information modelled in arCo is the following: circumstances involving cultural property, estimation of sex and age of death of an anthropological finding, biological taxonomy (genus, species, subspecies), archaeological materials, accessibility and availability of the cultural property, various types of classifications for individual types of cultural property, documentation of the intangible cultural property, etc.

The seven modules occurring in arCo are organised in this way:

- **core:** models the information considered as top-level, e.g., the relationship between the whole and its part, between an entity and its characteristics, between an entity and a generic place, etc. It is imported by all other modules of the network.

  Examples: classification, category, event, organization, etc.

- **arco:** models the information considered as central concepts in the Cultural Heritage domain (e.g., modelling of the material composing a batch of archaeological materials).

  Examples: archaeological property, material, dating, botanical heritage, etc.
- **denotative description** models information regarding the cultural property itself, with its attributes. Denotation refers to the act of indicating something through external signs.

  Examples: area, height, capacity, diameter, extension, thickness, etc.
- **context description** models the information regarding the context of cultural property during its history: each context is described in terms of "space", "time", "event", "entities involved".

  Examples: bibliography, dating, inventory, responsibility, provenance, diagnosis, etc.
- **location** models information related to the location of cultural property. This allows for example the correct identification of the location of a stratigraphic record.

  Examples: address, altitude, city, old town, district, site type, etc.
- **catalogue** models the information related to the Catalogue of Cultural Heritage, and therefore to the catalogue records.

  Examples: cataloguing level, access profile, catalogue record, version, etc.
- **cultural event** models information about events involving cultural property, e.g., recurrent events, understood as collections of events occurring with a certain periodicity.

  Examples: collection, exhibition, time, event or situation, etc.

This is an overview of the ontology that can be downloaded here [30].

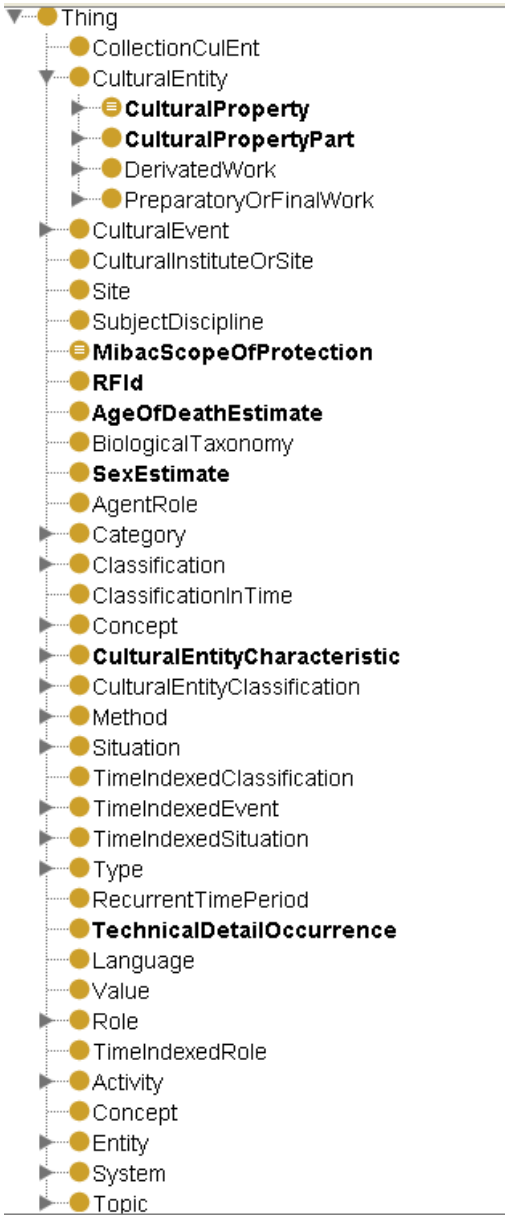


Figure 20: arCo ontology

1. Translation table

This first stage is important because it could be employed to create a mapping between information present in arCo and that present in GraphBrain. To reach this objective, a translation table has been developed. In this section, the design of the translation table to provide to the import module (explained in the next section) is reported. The table aims to reach a dual objective:

- **translating** the arCo triples expressed in the English language into new triples expressed in another language (in this case in Italian).
- **transforming**, if needed, the relationships into attributes or vice versa.

The structure of the chosen table consists of three columns in which: the first one represents the original arCo entity, relationship and attributes names; the second one represents the translation and, if needed, the transformation of these names; while the third column represents the type (class) of the new node created when an attribute becomes a relationship. The third field column is not mandatory, but it appears just in the last case.

The format used to define entities, relationships and attributes in this table is the following:

(entity: entityName) | (relationship: relationshipName) | (attribute: attributeName) | (class: className)

This format has been chosen both to keep track of the names in the two different languages and to check the possible transformation (attribute to relationship or vice versa). The prefixes that have been employed are the following: **entity**, **relationship**, **attribute**, **class**. They are useful to specify the type of elements that must be translated or, maybe, transformed. This format has been selected specially to make the transformation process simpler (attribute to relationship or vice versa). This table format can include three different possibilities:

- If the element in the first column is an entity having format (entity:entityName), it is just translated in the language stated in the second table column.
- If the element appearing in the first column is a relationship having format (relationship:relationshipName), it is translated and, if needed, it may be

also transformed into an attribute. In this case, the second table column has this format: (attribute:attributeName).

- Similarly, if the element appearing in the first column is an attribute having format (attribute:attributeName), it is translated and, if needed, it could be transformed into a relationship. In this case, the second table column has this specific format: (relationship:relationshipName), while the third column includes the class name of the new node that must be created and that is linked to the old node through the new relationship.

To better understand all the cases that can occur, a clearer example is reported here. It justifies the choice of the table format since it can catch all the possible variations explained before.

- If the element is an **entity**, the translation is made in this way:

  (entity: Agent),(entity: Agente)

- If the element is a **relationship** and it is decided to keep the relationship, just the translation is made:

  (relationship: hasAgent),(relationship: haAgente)

- If the element is an **attribute** and it is decided to keep the attribute, just the translation is made:

  (attribute: description),(attribute: descrizione)

- If the element is a **relationship** and it is decided to transform the relationship into an attribute, both the translation and the transformation must be specified in this way:

  (relationship: hasCharacteristic),(attribute: caratteristica)

- If the element is an **attribute** and it is decided to transform the attribute into a relationship, both the translation and the transformation must be specified in this way:
  (attribute: keyword),(relationship: haParolaChiave),(class: ParolaChiave)

Therefore, for each row, when the prefix (**entity**, **relationship**, **attribute**) is the same for the first two columns, it means that just the translation is performed. Otherwise, when the prefix is different in the first two columns, it means that, in addition to the translation, also the transformation has to be applied (relationship into an attribute or vice versa). The third column is useful just in case an attribute should become a relationship. When it happens, the last column includes the class name of the new node. This is also useful when there are different attributes with the same name that refer to different classes.

In this project, a basic table has been manually built, in which the translations of the original arCo entity, relationship and attribute names have been made. This table could be a starting point to define possible transformations (attribute ⊚ relationship or relationship ⊚ attribute). If someone wants to do so, it is necessary to change the prefix of the second column of the row in which they want to transform the attribute into a relationship or vice versa. This basic table involves a translation from English to Italian according to the official translations available on the arCo website. Relationships haven't been transformed into attributes (or vice versa), since this is a basic version in which the potential user can modify the rows in the second table column according to their needs. This basic table built has been stored in the attached csv file.

The table has to be provided as input to the import module (described in the next section) and exhibits the structure previously seen.

In this section, the description of the strategy used to import arCo RDF triples into a Neo4j graph database is reported. Neo4j is a graph database management system as an ACID-compliant transactional database with native graph storage

and processing, and it is the most popular graph database. Neo4j is implemented in Java and accessible from software written in other languages using the Cypher query language through a default HTTP protocol. The design proposal involves a strategy useful to:

- read the n triples of the arCo dataset (of size 48GB).
- recognize the triple types.
- translate/transform the triples according to the translation table (previously seen).
- add the triples in the graph database.

2. Read and recognize the triples

In this section, it is reported the main strategy used to read and recognize the arCo RDF triples, without considering the translation table (it will be mentioned in the next section). First, the dataset is composed of n lines and each line represents a single **triple** structured in the following way:

<subject> <predicate> <object>

In the arCo dataset, the subject, the predicate and often, also the object includes an IRI (Internationalized Resource Identifier), which is useful to univocally identify specific resources on the web. Usually, the subject and the object are nodes, and each node is identified through an IRI; but in the dataset sometimes the object could also be a string, which may contain the attribute value (of any type) of a specific subject (which is always a node). In most cases, the predicate links two nodes and, when this happens, it represents an arch between the two nodes. The next step consists of recognizing the three different types of triples stored in the arCo dataset. Specifically, we can distinguish:

1. **creating an instance** of a particular node
2. **adding an attribute** to a particular node
3. **adding a relationship** that links two nodes

In order to distinguish the three possible triple types, the dataset has been analyzed to discover a strategy to apply.

1. Creating an instance

In particular, if the triple includes a predicate with an IRI like this "**<.../rdf-syntax-ns#type>**", it means that an instance must be created. In this case, a node is added in the Neo4j graph database (if it doesn't exist yet). The newly added node adopts the following features: the **node label** is expressed through its IRI (specified in the triple subject), while the **node type** is specified in the triple object. An example of this kind of triple is shown below:

<https://w3id.org/arco/resource/CatalogueRecordRA/1200583495> □ Node label
<http://www.w3.org/1999/02/22-**rdf-syntax-ns#type**>
<https://w3id.org/arco/ontology/catalogue/**CatalogueRecordRA**> . □ Node type

This triple would become in the graph like this:

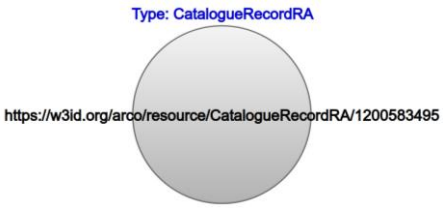


Figure 21: Node in arCo

2. Adding an attribute

If the triple includes an object as a string, it means that an attribute must be added to a node specified in the triple subject. There could be two different cases: if the node already exists, the attribute is just added to the node; while if it doesn't exist yet, the node is firstly entered in the graph and then the attribute is added.

Besides, the **attribute name** is specified in the triple predicate (the last element of the IRI path). An example of this kind of triple is shown below:

<https://w3id.org/arco/resource/CatalogueRecordRA/1200583495> □ Node label
<https://w3id.org/arco/ontology/core/**note**> □ Attribute name
"LA STIMA SI RIFERISCE A DUE FRAMMENTI; DATA DI SCAVO: 1965". □ Attribute value

This triple would become in the graph like this:

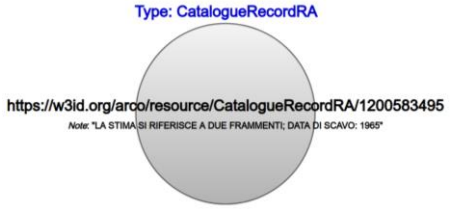


Figure 22: Attribute in arCo

3. Adding a relationship

If the triple contains a subject and an object expressed as resources (nodes including resource IRIs), then it means that the predicate necessarily corresponds to a relationship and so an arch must be added between the two nodes present on the left and the right of the triple. The arch is added if the left and right nodes already exist in the graph, otherwise, the nodes are firstly entered into the graph and then the arch linking them is generated. Furthermore, the **arch label** is expressed in the triple predicate (the last element of the IRI path). An example of this kind of triple is shown below:

<https://w3id.org/arco/resource/CatalogueRecordRA/1200583495> □ Left node label <https://w3id.org/arco/ontology/catalogue/hasCatalogueRecordVersion> □ Predicate label <https://w3id.org/arco/resource/CatalogueRecordVersion/1200583495-compilation>. □ Right node label

This triple would become in the graph like this:

Figure 23: Relationship in arCo

From this strategy, we can reuse the driver to interact with Neo4j and an idea on how to parse triples.

II. Proposed Solution

First, we need a section to create, import, and manipulate ontologies.

II.I Schema Tab

The purpose is to include a schema and ontology management section. Specifically, the Schema section will have four components: one for classes (and subclasses), one for relationships, another for importing or removing new domains, and a bottom bar (Domain section) for exporting to three formats (XML, OWL, and Prolog) and changing domains. We report here the screenshot of the Schema section. For the development of Gr@phBRAIN, the JavaServer Faces (JSF) framework has been used. It is a Java technology, based on the Model-View-Controller (MVC) architectural design pattern, whose purpose is to simplify the development of the user interface (UI) of a web application; therefore, it can be considered a framework for server-side UI components.

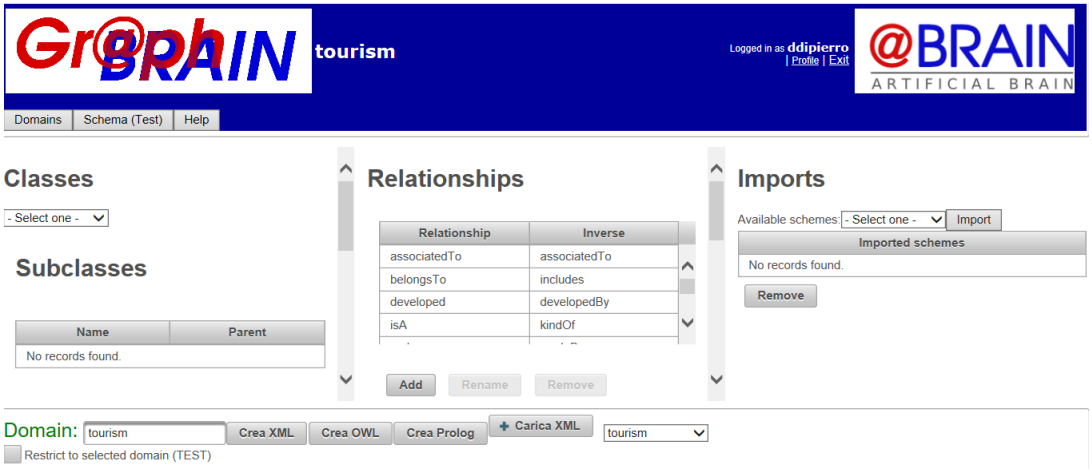


Figure 24: Schema section in Gr@phBRAIN

II.I.I Design

Classes

Starting from the left section, we have the title Classes and a selection that allows you to choose among possible classes.

Figure 25: Classes in Gr@phBRAIN

Attributes

Once the selection has been completed, the table below will show the list of attributes of that class and any subclasses. On the classes must be possible to

perform three basic operations: addition, renaming and deletion. The third one must be possible only if the selected entity is not involved in any relationship.
On attributes it must be possible to execute the same operations but, with the generic name Edit, we mean also the possibility to change not only the name of the attribute but also three further fields (mandatory, display and distinguishing). It must be possible to modify also the attribute type, and if the attribute is of entity type, its target. You can modify an attribute only if you have first selected a class and then selected an attribute from the table below the classes.

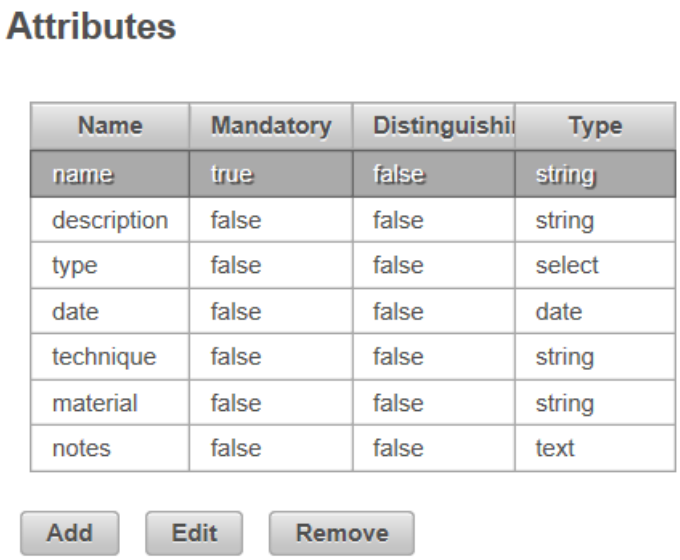


Figure 26: Attributes in Gr@phBRAIN

An attribute can be of type select. This indicates that it cannot take free values (like the string type) but a value is taken from a domain of possible values for that attribute. You must provide the ability to modify the set of values. For this purpose, under the attributes table, there is also a values table which, when selecting an attribute of type select, fills up with the values of the domain. To remove or add a value you must first select a class and then an attribute. It was

not deemed necessary to include the functionality to edit a value as you can simply remove the old one and insert the new one.

We report here a table of the values.

Figure 27: Values of classes in Gr@phBRAIN

Once a value is selected, it is possible to remove it while adding it is always possible (if it is an attribute of type select). When you edit an attribute and select a value other than select, all its values (if any) are removed.

Subclasses

In the ontology, it is possible to define the relation is_a or its equivalent that allows creating hierarchies of classes through the concept of inheritance typical of object-oriented programming. A subclass inherits all attributes of the parent class and can add more. The possible operations on subclasses are the same as for classes, including creating a subclass to create hierarchical chains. Once a class is selected, a table is shown with two columns: name and parent and the attributes of the parent class. If you select a child class, all attributes of the child class not in the parent will be shown. If you remove a class, all classes that inherit from it will be cascaded.

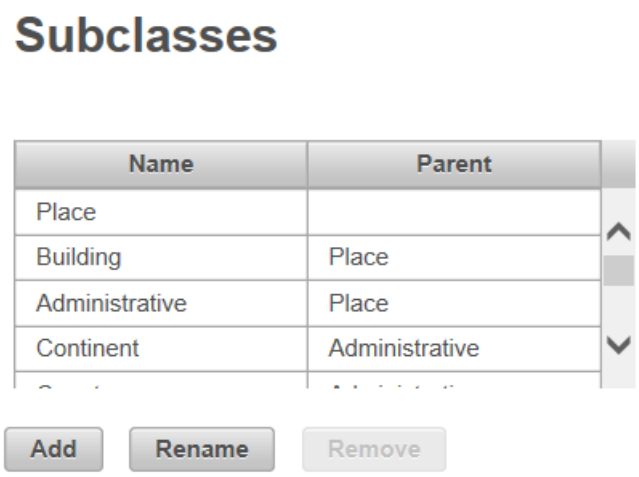


Figure 28: Subclasses in Gr@phBRAIN

As you can see, the first line contains the name of the top-level class. Clicking on that line takes you back to the main class level to see the common attributes or perform editing operations on the top-level class.

Relationships

In the middle section, we have the title Relationships and a drop-down menu that allows you to choose from the possible relationships in the domain.

Figure 29: Relationships in Gr@phBRAIN

Once the selection is complete, the list of references and attributes for that relationship should appear in the table below. The reference is a subject-object entity pair that represents the orientation of the relation.

The same operations must be possible on relations as on classes. When adding a new relation, the first reference must already be inserted to give foundation to the existence of that relation.

For relations, if you decide to remove one of them, you will not check if there are references that have not been removed, but you will proceed with the removal of these in cascade. This is because, when you select a relation, a table of references appears, and therefore it is assumed that the user recognizes that there are open references. You cannot think the same way for classes because there is no direct way to check that a class does not have references in relationships.

For each relation, it must be possible to add or remove references. As in the previous situation, we avoid changing a reference since it consists of only two fields, it will be equivalent to remove the old one and add a new one. We report a table of references.

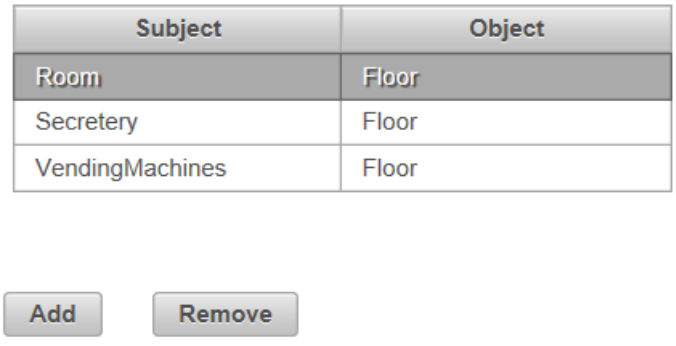


Figure 30: References in Gr@phBRAIN

Under this table, there is another one containing the attributes of the relation on which you can perform the same operations possible on the class attributes.

Below this table are also the values for the select type attributes. Operations, screens, and constraints are identical.

It is important to note that relation only makes sense if at least one reference is present. If therefore all of them are removed, automatically the relation is also removed.

Imports

In the third section, we have the title Imports and a drop-down menu that allows you to choose from the possible domains to import.

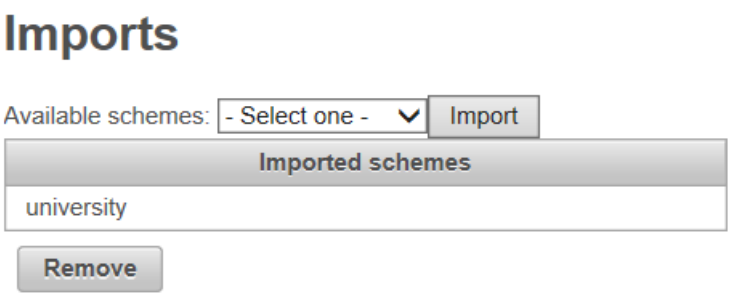


Figure 31: Import ontologies section in Gr@phBRAIN

Domains that have already been imported will be removed from the drop-down menu and the current (importing) domain will never appear. Each time you import a domain, all the classes and relationships of that domain are added. A domain can have classes or relations of the same name as the current domain, in that case, the elements of the current domain are given priority and those of the imported domain are overwritten.

If you remove a domain, similarly, all classes and relationships from that domain are removed. In this way, you can build import chains and it is essential to manage the changes because from one domain you can modify classes and relationships coming from another. To remove a domain just select it and click on Remove.

Domain

In this last section, we have the name of the current domain, three buttons that allow you to export the current ontology in three formats: XML, OWL, Prolog, a button to load XML files from a specific location on the server (not part of the goals) and a menu to change the domain you are in.

Figure 32: Export section in Gr@phBRAIN

We focus on the three buttons that allow you to export the current ontology. The three formats are not random because XML is the format used to import new domains (as will be specified later) while the OWL format is useful to describe knowledge bases, make inferences about them and integrate them with the contents of Web pages. This format contributes to the semantic web. The Prolog language, finally, is strongly used to describe ontologies and knowledge bases as it allows to easily build deduction rules based on Horn clauses.

By exporting an ontology to Prolog, a small knowledge base will be built to perform queries about the ontology. The new knowledge base will import the data exported by Gr@phBRAIN.

It is important to choose a format (for entities, relations, attributes, etc.) that is effective in answering possible queries.

To change the domain, instead, just select a different one from the drop-down menu below, all unsaved changes will be lost, and the newly selected schema will be loaded.

Before proceeding with the design phase, you want to describe a conceptual model (E-R for example) to represent the objects that the system will modify to clarify any constraints that will be described later.

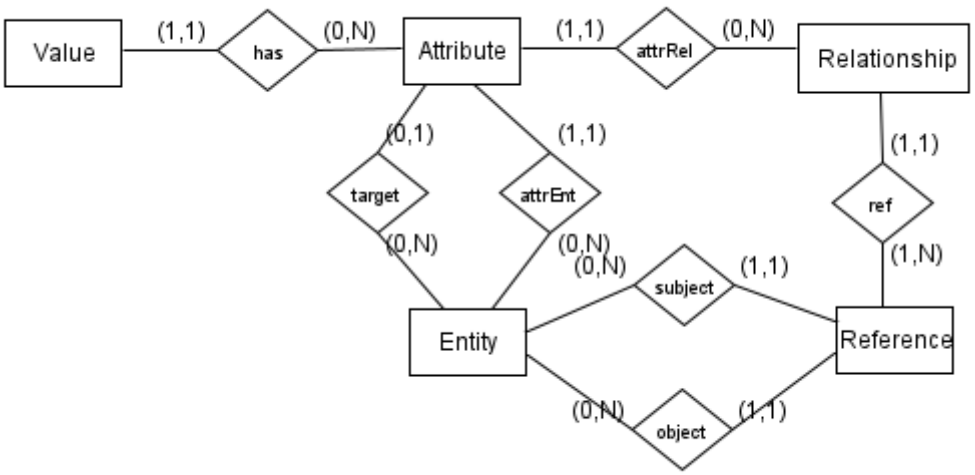


Figure 33: Relational model of the objects in Gr@phBRAIN

Before describing in detail how the various tasks are performed, it is necessary to describe the structure of the XML files that concerns both import and export. Each file is composed of a "domain" node with the domain name as an attribute and at least two child nodes: "entities" and "relationships". Optionally, it's possible to insert a third child node before "entities" named "import" used to import domains.

The children of the "import" node are called "files" and have a single attribute that indicates the name of the domain to be imported. The last child of the "import" node can also be the "deleted" node, which will be described later.

The child nodes of "entities" are named "entity" and have a class description. The only attribute of this name is the name and has two possible child nodes: "attributes" and "taxonomy". Under "attributes" there are as many "attribute" nodes as there are attributes, they have as attributes the name, the mandatory and the type of attribute. If the attribute is of select type, there are a child node "values" which contains child nodes "value" for each possible value of the attribute. Under "taxonomy" the subclasses of the source class are described. Its

only child is the "values" node which in turn contains a "value" node for each subclass. Each subclass can recursively have attributes under the "attributes" node and other subclasses under the "values" node.

The child nodes of "relationships" have name "relationship" which have name attributes and inverse relationship and at least one child "references". Under "references" we have a "reference" node for each relationship reference and subject and object attributes. Under "relationships" there can also be the "attributes" node which has the same structure as the homonymous nodes in "entities".

We report a schematic example of xml file.

```
<?xml version="1.0"?>
<domain name="domain">
        <import>
                <file name="import"/>
        </import>
        <entities>
                <entity name="entity1">
                        <attributes>
                                <attribute name="name1" ... "/>
                        </attributes>
                </entity>
                <entity name="entity2">
                        <attributes>
                                <attribute name="name2" .../>
                        </attributes>
                        <taxonomy>
                                <values>
                                        <value name="value1">
                                                <attributes>
                                                ...
                                                <attributes>
                                        </value>
                                </values>
                        </taxonomy>
```

```
                </entity>
        </entities>
  <relationships>
    <relationship name="relationship" inverse="inverse">
        <references>
                        <reference subject="subject" object="object"/>
                </references>
        </relationship>
  </relationships>
</domain>
```

As you can easily spot, this is not a standard like OWL or RDF.

Classes

In the system, an entity is characterized by name, attributes, possible children's classes, possible father class and domain in which it is born. In particular, the last field is used to understand if a class has been imported from another domain. The class that contains the information about an entity is Entity.java. We report the structure.

```java
public class Entity {
    String name;
    ArrayList<Attribute> attributes = new ArrayList<>();
    ArrayList<Entity> children = new ArrayList<>();
    Entity parent;
    String domain;
```

An attribute is characterized by name, mandatory, distinction, type, possible target and a possible list of values. The class that contains the information about an attribute is Attribute.java. We show the structure of the class.

```
public class Attribute {

    public String name;
    public boolean mandatory = false;
    public boolean distinguishing = false;
    private List<Value> values = new ArrayList<>();
    public String dataType;
    private String target;
```

A relationship is so characterized: name, inverse, attributes, references (subject-object pairs) and domain in which it is born. The class that contains the information is Relationship.java of which we report the structure.

```
public class Relationship {
    private String domain;
    private String name;
    private String inverse;
    private Vector<Attribute> attributes = new Vector<>();
    private Vector<Reference> references = new Vector<>();
```

Subclasses

Subclasses are also entities, so their structure is the same as described above for entities. To retrieve all subclasses of a selected class we use variable subentities and, for display purposes, also subentitiesToString which collects only the names. They are validated through the getChildren method applied to a class.
Any method that will modify the class structure, must know if you are modifying a class or subclass as described below.

Add Class

When adding a new class, it is necessary to give it at least the name and the domain in which it is born. If there is already a class with the same name, it will be replaced by the new one because, as already described, priority is always given to the current domain. To allow the generation of subclasses, you can select the Parent class from a drop-down menu. When you select a class and want to

create a new one, the drop-down menu will be filled with all the subclasses of the selected class (plus the class itself).

There is a Domains class that contains a list of loaded domains, the class that represents a single domain is called DomainData and contains information about entities, relationships, attributes, subjects and relationship objects. We report the structure here.

```
public class DomainData {
    private ArrayList<String> importedFiles = new ArrayList<>();
    private ArrayList<String> removedEntities = new ArrayList<>();
    private ArrayList<String> removedRelationships = new ArrayList<>();
    private String domain;
    private Vector<Entity> entities = new Vector<Entity>();
    private Vector<String> subjects = new Vector<String>();
    public Vector<Relationship> relationships = new Vector<Relationship>();
```

The importedFiles variable keeps track of imported domains while removedEntities and removedRelationships list entities and relationships removed from other domains after import.

The SchemaBean.java class is the one that captures the user's values in the class entry form, fetches the right domain from the Domains class and on DomainData applies the addEntity method. We report here the interaction and the main function calls to ensure the class is added.

This is the simple pop up to insert a new entity via its name.

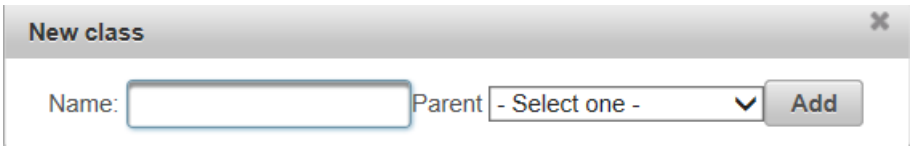


Figure 34: Creating a class in Gr@phBRAIN

The schemaBean variable represents the current domain.

The domain in which the entity is created is the current domain.

In the method, we distinguish whether we are adding a top-level class or a subclass through the selectedParent variable. In case selectedParent is validated, the class named selectedParent is retrieved through the getEntityFromSub method.

In the DomainData.java class, there is a list of entities called entities that collects all the entities in the domain.

Rename Class

When renaming a class, you must give it a new name. If there is already a class with the same name, it will be replaced in the same way as when adding it.
After selecting a class and pressing the Rename button, this is the simple pop up that appears.

Figure 35: Renaming a class in Gr@phBRAIN

The selectedEnt variable is set to the value of the selected entity thanks to a listener present in the table rows. This mechanism of connection between interface and Java classes is typical of the JSF framework which is the one we are using.
In this case, it is the variable selectedSubEntName that discriminates if the class you are renaming is a subclass or not, it will be used in the future for all other operations.
Here the domain in which the entity is created is the current one.
At this point, it is necessary to describe the mechanism by which the imported classes and relationships are removed. It was mentioned above that the last child

node of "imports" in the XML file can be the "deleted" node. The latter is needed to keep track of entities and relationships that should not be imported later because they have been removed. Each child of "deleted" is an "entity" or "relationship" node whose only attribute is the name. When renaming an entity (or relationship), it must be necessary to insert among the entities to be removed the old one and insert the new one as an entity of the current domain. It will be described later how the import function, reading the XML file, recognizes what is to be removed. This is a necessary operation to make sure that everything modified by the new domain is permanent even if you import the old domain. The old domain is not modified. In SchemaBean.java there are two lists: entitiesToRemove and relationshipsToRemove. They contain all imported entities and relationships that need to be removed and possibly rewritten. All relationships that contain as the subject or object the old entity must be replaced as the references will have to contain the new entity name.

The renaming operation is simpler in case you change the name of an entity (or relation) that does not come from an imported domain, in that case just use a set method for the name change and, as before, make the changes also to the references of the relations.

Remove Class

Removing an entity can be seen as a simpler case of a renaming since you don't need to add a new one.

After selecting a class (that is not included in relationship references) or a subclass, and pressing the Remove button, this is the simple confirmation pop up that appears.

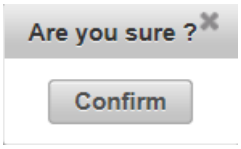


Figure 36: Removing a class in Gr@phBRAIN

Compared to renaming, here it is not necessary to rewrite the entity of that domain, as it will be enough to insert the name among the child nodes of "deleted". The entitiesToRemove list contains exactly all names of removed entities that will not be imported if the generated XML file was used as a source. The fullAttributes list contains all attributes to be shown in the table below the classes and is emptied in case of removal.

Class Attributes

Once an entity is selected, its attributes can be managed. A table is shown in which the three basic operations are possible.

If you decide to insert a new attribute a pop up will be shown in which to insert the various fields.

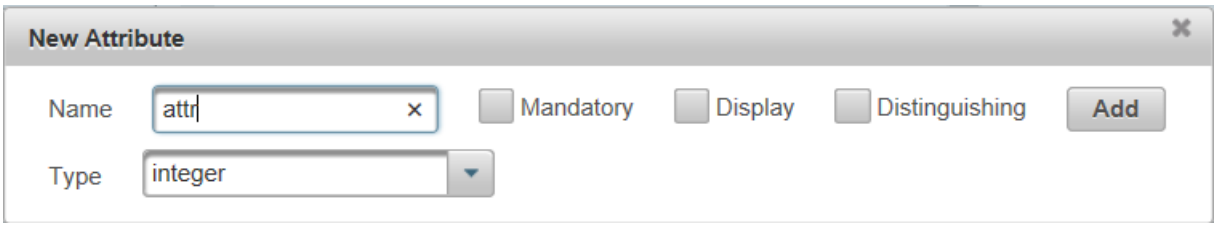


Figure 37: Creating an attribute in Gr@phBRAIN

In all the modification operations it will always be necessary to write in entitiesToRemove the entity you are modifying. There is a removeAndInsert method with the exact purpose of removing the old entity (and marking it as "deleted") and rewrite it. There is a method of the same name for relations.

In SchemaBean.java there is an attribute tempAttr which is initialized in the constructor with default values and modified appropriately by the user in Fig. 38. It is also used for editing; in that case, it will have the values of the selected attribute.

To edit an existing attribute, you will first have to select it and then click on Edit. The following pop up will be shown.

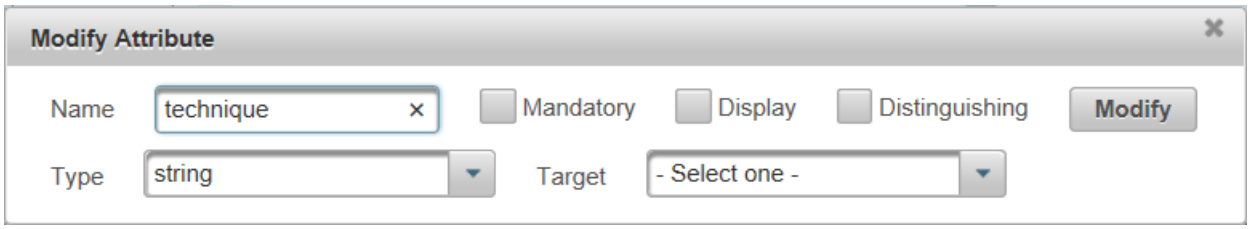


Figure 38: Modifying attribute in Gr@phBRAIN

It is possible to insert a target to the attribute, this actually should be possible only for entity type attributes. On the interface side, it was not possible to obscure that selection as it is very complex to manage the change of visibility every time a type other than an entity is chosen. However, the problem is handled in the modifyAttribute method which sets target to null in case the type is not an entity. Whenever an attribute is created that does not have type select, all values (if any) that it had been removed so that there are no inconsistencies.
In this method, tempAttr is validated with the values of the selected attribute. This happens in the attrSelected method which is called in the listener of the attribute table every time a row is selected.

SchemaBean.java contains also two variables fullValues and selectedAttr: the first one collects all values to be shown in the table under attributes, in case an attribute of select type is selected while selectedAttr contains the selected

attribute using the same mechanism described above for selectedEnt. At the end of the procedure, the attribute table will contain the new modified attribute.

To remove an existing attribute, you will first have to select it and then click Remove. In this case, in fullAttributes the selected attribute is removed and fullValues is emptied to avoid inconsistency problems. At the end of the procedure, the attribute table will no longer contain the selected attribute.

Class Values

Once an attribute of type select has been selected, it is possible to manage its possible values. A table is shown (Fig. 39) where it is possible to add/remove a value.

If you decide to insert a new value a pop up will be shown where to insert the name.

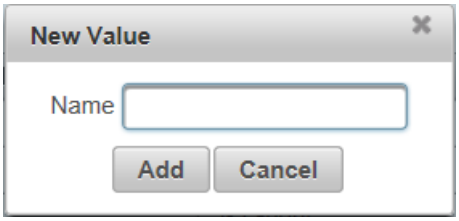


Figure 39: Creating a value in Gr@phBRAIN

In this case, tempAttr is used to refer to the attribute in which to add the value.

To remove an existing value, we will first have to select it and then click on Remove.

We report the removeValue method present in the SchemaBean.java class.

The variable selectedValue contains the selected value and is set exactly as in the listener mentioned above. At the end of the procedure, the attribute table will no longer contain the selected attribute.

Relationships

In our system, a relationship is characterized by name, attributes, inverse, references and domain in which it arises. The class that contains the information about a relationship is Relationship.java. Relationship attributes are identical to class attributes. We report the structure of the class.

Add relationship

When adding a new relation, it is necessary to attribute to it at least the name, the domain in which it is born and a reference (subject-object pair). As for the classes, a relation with a name already present will overwrite the previous one.

The SchemaBean.java class is the one that captures the user's values in the relation insertion form and applies the newRelationship method. We report here the interaction and the main function calls to ensure that a relationship is added.

This is the pop up to insert a new relationship by entering its name, inverse and at least one reference.

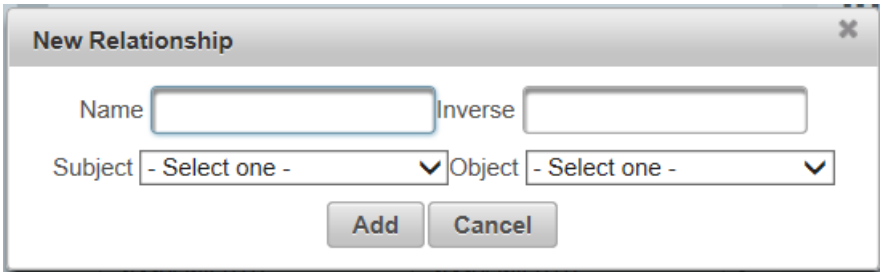


Figure 40: Creating a relationship in Gr@phBRAIN

The domain in which the relationship is created is the current domain.

In the class DomainData.java, there is a list of relationships called relationships that collects all the relationships of the domain.

Rename relationship

When you rename a relationship, you have to give it a new name. After selecting a relationship and pressing the Rename button, this is the simple pop up that appears.

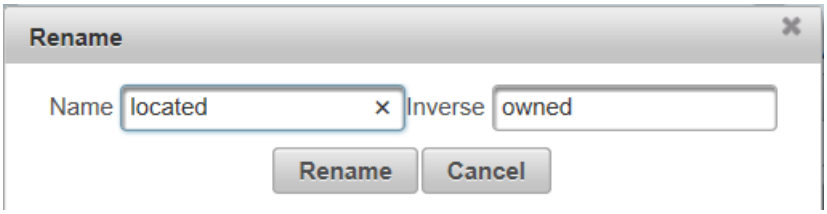


Figure 41: Renaming a relationship in Gr@phBRAIN

The selectedRel variable is set to the value of the selected relationship. Also here, the domain in which the relation is created is the current one.

Remove relationship

Removing a relation can be seen as a simpler case of renaming as there is no need to add a new one. After selecting a relation and pressing the Remove button, the relation and its references are removed.

We report the method removeRelationship present in the class SchemaBean.java. Compared to renaming, here you don't need to rewrite the relationship as you only need to insert the name among the child nodes of "deleted". The relationshipsToRemove list contains exactly all names of removed relationships that will not be imported if the generated XML file was used as a source. The fullAttributes list contains all attributes to be shown in the table below the classes and is emptied in case of removal.

Relationship References

Once a relation is selected, its references can be managed. A table is shown where you can remove or add a reference. The modification can simply be understood as removal and new deletion.

This is the pop up that is shown in the case of addition.

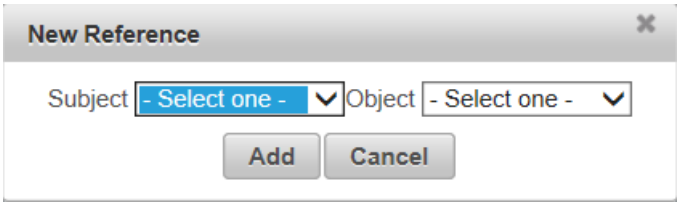


Figure 42: Creating a reference in Gr@phBRAIN

Once a reference is selected, it can be removed through the Remove button.

Relationship Attributes

Once a relationship has been selected, it is possible to manage its attributes. A table is shown in which the three basic operations are possible.

If you decide to insert a new attribute, you'll see the same pop up for the entity attributes and the XHTML code that generates it is exactly equivalent apart from the change of the ids and variables used.

In all the modification operations it will be always necessary to write in relationshipsToRemove the relationship that is being modified.

In SchemaBean.java there is an attribute tempAttrRel which has the same role as tempAttr used to modify entity attributes.

To edit an existing attribute, you will first have to select it and then click on Edit. The same pop up for entity attributes will be shown and therefore We don't report the equivalent XHTML code.

SchemaBean.java also contains the variables fullValuesRel and selectedAttrRel equivalent to fullValues and selectedAttr described above but used for relations.

To remove an existing attribute, we will have to select it first and then click on Remove.

Relationship Values

Once selected a relation attribute of select type, it is possible to manage its possible values. It is shown a table in which it is possible to add/remove a value. If you decide to insert a new value, the same pop up for entity attribute values will be shown.

In this case, tempAttrRel has the same role as tempAttr for entity attributes.

To remove an existing value, we will first have to select it and then click Remove.

The selectedValueRel variable contains the selected relationship attribute value.

Imports

Importing new files is critical for building domain hierarchies.

The importedDomains and availableDomains variables are used to keep track of the imported and importable domains and are set so that the domain appears among the imported ones and is removed from the available ones. At first, availableDomains contains all XML file names (excluding ".xml") in the project's WEB-INF directory.

After creating the newDomain, statements are executed to load entities and relationships and keep track of this import.

The WEB-INF folder is the folder that contains the XML files that can be used as sources to import new domains.

The method reads the taxonomy and, for each subclass, recursively reads any attributes and other subclasses.

Domain

In the Domain section, three buttons allow the export in three different formats: XML, OWL and Prolog and a menu that allows you to change domain.

Export in OWL

When the Create OWL button is pressed, the build method of the BeanOWL.java class shown here is activated. The BeanOwl class contains the domain name, entities and relationships. For each entity, it inserts a tag and so for each attribute and subclass recursively. The same strategy is valid also for the relations where however there is no need to execute recursive calls.

Export in Prolog

When the Create Prolog button is pressed, the createFile method of the CreatePrologBean class is activated. The format for each basic predicate is reported here.

- domain/1: describes the domain in which we are working. In the SchemaBean.java class, we have the information about the domain we are working in as already shown.
- entity/2: describes an entity. The first value represents the domain in which the entity originates and the second is the name of the entity which is unique within the domain. This predicate is also used to identify all imported domains as we will see later. We have already described the Entity class which contains a string indicating the domain it belongs to.
- attribute/4: describes an entity attribute. The four values respectively identify: domain, entity, attribute, type. Domain and entity are used to identify the attribute because the name alone may not be enough, while the type was chosen as the fourth value to make it easier to find the type of an attribute. Alternatively, we should have created a type predicate, but it would have been necessary to rewrite the domain, entity and attribute name. The information you need is all contained in the Attribute class.

Relationship attributes are handled the same way, the second value will represent the name of the relationship instead of the entity.

- values/4: describes the values of an attribute of type select. The first three values are equal to those of the predicate attribute while the last is the list of values that the attribute can take. This more compact representation of the values was preferred since it is unlikely that you would need a single domain value; you are likely to be interested in searching for all the values of an attribute. You could also have placed the list of values in the attribute predicate as well, but you preferred to avoid replicating attribute since the values predicate should in principle be used much less frequently since most attributes are not of the select type. The values are stored in the Attribute class.
- mandatory/3, distinguishing/3, display/3, target/4: describe some peculiarities of an attribute such as its compulsoriness, distinction or target in case the attribute is of an entity type. It was preferred to create separate predicates rather than reify them in the attribute predicate both to avoid predicates with many arguments and because it is believed that these predicates are much less common. This information is stored in the Attribute class.
- parent/3: describes the inheritance relationship. The first value is the domain, the second is the parent class, and the third is the inheriting class. This information can be retrieved from the Entity class in which there is a list of child entities. It will be enough to insert a predicate for each child. We preferred to treat the children individually instead of putting them in a list so that we can more easily work on the individual classes instead of extracting them from a list.
- relationship/4: describes a relation reference. The four parameters represent the domain, relationship name, subject, and object, respectively. It is handled at the reference level so that you can easily identify all

relationships in which a class is involved. The Relationship class contains the list of references in which it is possible to write a predicate for each of them.

- inverse/1: describes the inverse of a relationship. It was preferred not to reify it as it is assumed not to be so relevant and not to increase the airiness of the relationship. The information about the inverse is always contained in the Relationship class.

From the exported file, a knowledge base will be created as described later.

Log files

To allow alignment between database content and changes, it is necessary to write to a log file all changes made during the session in the correct order. The idea is that when the ontology is saved, for example when exporting the XML, the alignment of the knowledge graph must correspond.

The log file will be a text file named "log.txt" that for convenience will be saved in the same folder where the xml files are. We report the full path from the root of the workspace folder:

eclipse-workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\OntoGraph\WEB-INF\log.txt.

When the Schema section is accessed, the file is opened in write mode by writing only a first header line.

We will now report all possible types of strings that can be written to the log file.

1. **entity ([entity]) removed**

Indicates that the [entity] has been removed.

2. **entity ([entity]) created**

Indicates that the entity [entity] has been created.

3. **entity ([entity]) modify [new_entity]**

Indicates that entity [entity] has been renamed to [new_entity].

4. **entity ([entity]) add (name = [name], mandatory = [mandatory], distinguishing = [distinguishing], display = [display], type = [type])**

Indicates that a new attribute with name [name], mandatory [mandatory], distinguishing [distinguishing], display [display] and type [type] has been added to entity [entity].

5. **entity_attribute ([entity], [attribute]) modify (name = [name], mandatory = [mandatory], distinguishing = [distinguishing], display = [display], type = [type], target = [target])**

Indicates that the attribute [attribute] in the entity [entity] has been replaced by a new attribute with name [name], mandatory [mandatory], distinguishing [distinguishing], display [display], type [type], and target [target].

6. **entity [entity] remove (name = [name])**

Indicates that a [name] attribute has been removed from the entity.

7. **entity_attribute ([entity], [attribute]) add (name = [value])**

Indicates that value [value] has been added to the domain of possible values of the entity's [entity] attribute [attribute].

8. **entity_attribute ([entity], [attribute]) remove (name = [value])**

Indicates that the value [value] has been removed from the domain of possible values of the entity's [entity] attribute [attribute].

9. … child of [entity]…:

Previous operations were performed at the top-level class level. If they are performed on a subclass you specify, before the actual operation, the parent class of the class in question. For example: entity (Son) child of Parent modify Son.

10. relationship ([relationship], [inverse_relationship]) reference ([subject], [object]) created

Indicates that the relationship [relationship] with inverse [inverse_relationship], subject [subjecy] and object [object] has been created.

11. relationship ([relationship]) removed

This indicates that the [relationship] has been removed.

12. relationship ([relationship]) modify [new_relationship]:

Indicates that relationship [relationship] has been renamed to [new_relationship].

13. relation_subject ([relationship], [subject]) modify [new_subject]

Indicates that the subject of relationship [relationship] has been changed to [new_subject].

14. relation_object ([relationship], [object]) modify [new_object]

Indicates that the object of relationship [relationship] has been changed to [new_object].

15. relationship ([relationship]) add ref ([subject], [object]):

Indicates that a reference with subject [subject] and object [object] has been added in relation [relationship].

16. **relationship ([relationship]) remove ref ([subject], [object]):**

This indicates that in the relationship the reference with subject and object has been removed.

17. **relationship ([relationship]) add (name = [name], mandatory = [mandatory], distinguishing = [distinguishing], display = [display], type = [type])**

Indicates that a new attribute with name [name], mandatory [mandatory], distinguishing [distinguishing], display [display], and type [type] has been added to the [relationship] relationship.

18. **relationship_attribute ([relationship], [attribute]) modify (name = [name], mandatory = [mandatory], distinguishing = [distinguishing], display = [display], type = [type], target = [target])**

Indicates that the attribute [attribute] in relationship [relationship] has been replaced by a new attribute with name [name], mandatory [mandatory], distinction [distinguishing], display [display], type [type] and target [target].

19. **relationship ([relationship]) remove (name = [name]):**

Indicates that an attribute [name] has been removed from the [relationship] relationship.

20. **relationship_attribute ([relationship], [attribute]) add (name = [value]):**

Indicates that value [value] has been added in the domain of possible values of attribute [attribute] of relationship [relationship].

21. **relationship_attribute ([relationship], [attribute]) remove (name = [value]):**

Indicates that the value [value] has been removed from the domain of possible values of the attribute [attribute] of the relationship [relationship].

22. imported ([domain]):

Indicates that the domain [domain] has been imported.

Here is an example of a log file after performing some operations on the ontology.

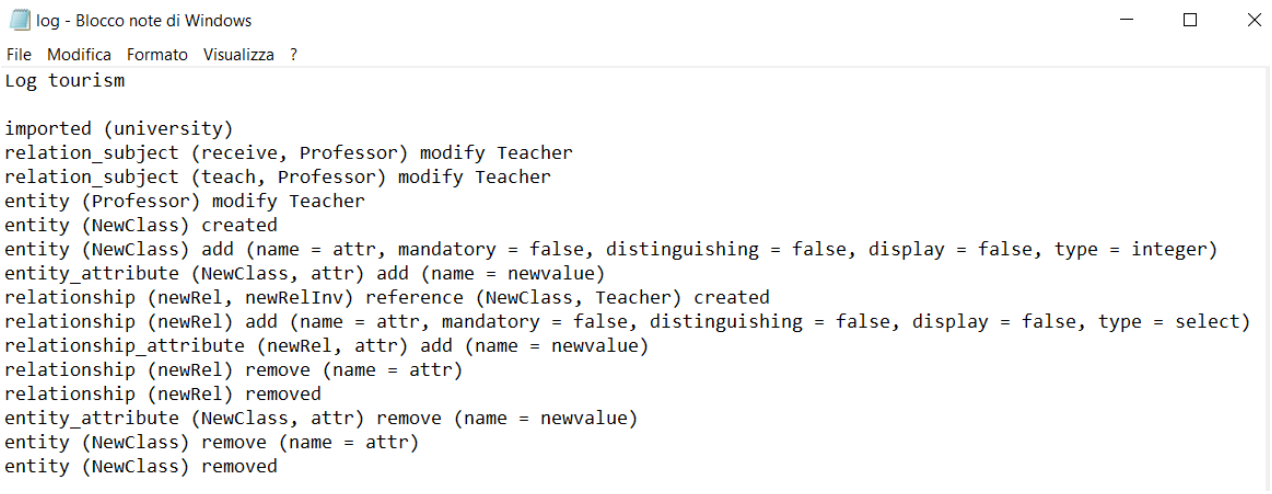


Figure 43: Example of a log export

II.I.II Test

Here we illustrate examples of tests made for the various functionalities of Gr@phBRAIN.

It is good to underline some a priori defects such as the little usability, the different page refreshes and the fact that in some cases it is necessary to perform the same action twice to execute it. The latter in particular seems to be a JSF bug that affects some interface components such as commandButton.

Despite some initial difficulties, you can still easily use the system after exploring the various sections.

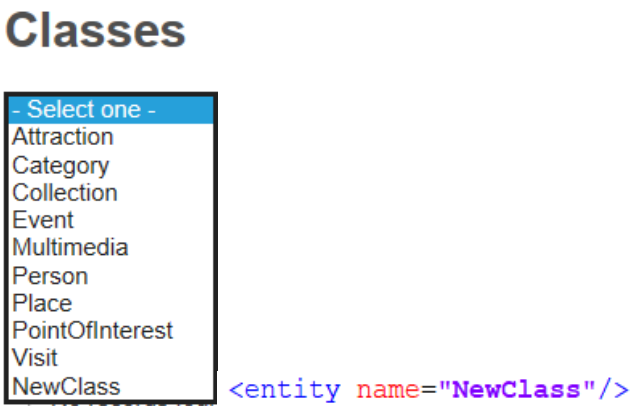
For verification, we'll use both the system's GUI and the XML export to verify the correct interpretation of the entities to be labelled as "deleted".

Add Class

To add a class there are no requirements. We will proceed with the following tests:

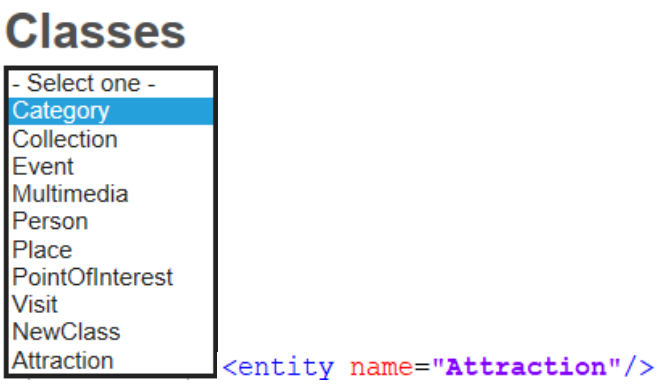
1. insert an entity name different from the present ones.
2. insert an entity name that exists only in the current domain.
3. insert an existing entity name in an imported domain.

1.



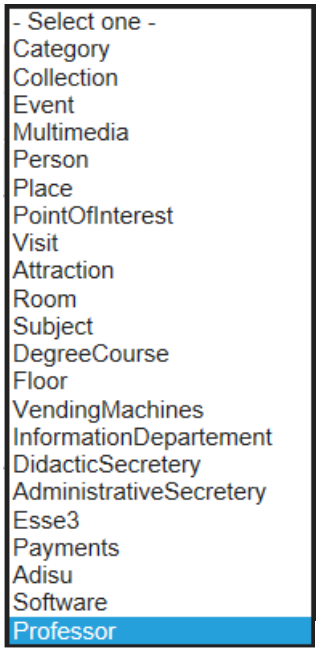
As you can see, the new class NewClass is present in the class list and exported in the XML format.

2.



A new Attraction class was created that replaced the previous one and has no relationship, like any newly created class.

3.



<entity name="Professor"/>

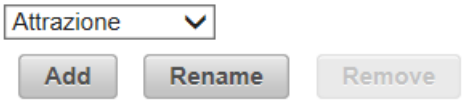
You imported the University domain and created the Professor class that replaced the previous one and removed the old relationships.

Rename Class

To rename it is necessary to select one of them. You will proceed with the following tests:

1. Rename an existing class in the current domain only.
2. Rename an existing class in an imported domain.

1.



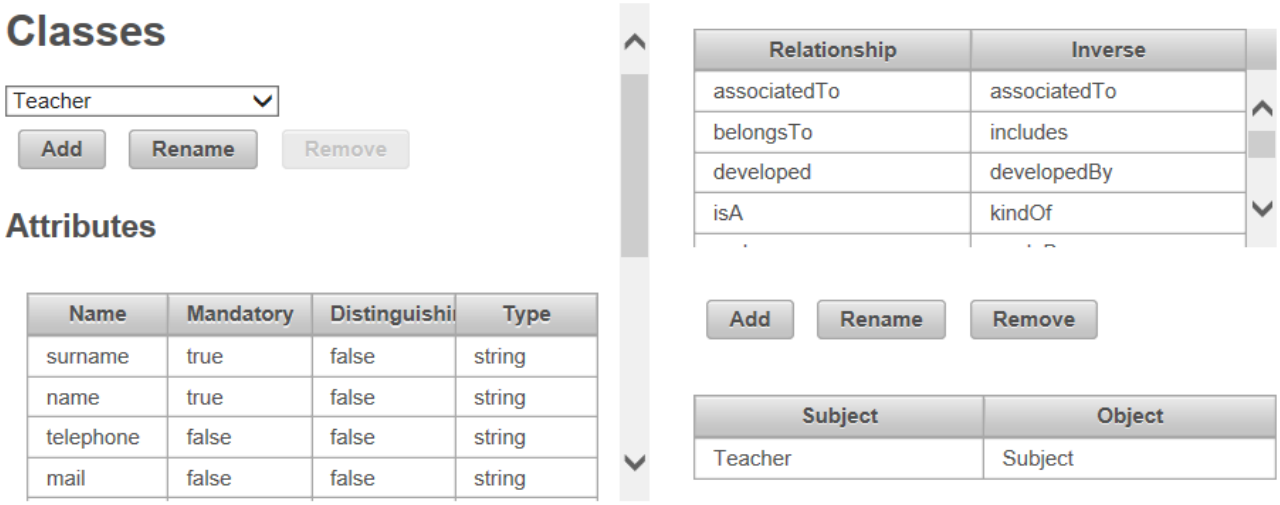
Attributes

Name	Mandatory	Distinguishi	Type
name	true	false	string
description	false	false	string
type	false	false	select
date	false	false	date
technique	false	false	string
material	false	false	string

```
<references>
    <reference object="Place" subject="Attrazione"/>
    <reference object="PointOfInterest" subject="Attrazione"/>
```

The Attraction class has been replaced by Attraction which has inherited attributes and relationships.

2.




```
<import>
    <file name="university"/>
    <deleted>
        <entity name="Professor"/>
    </deleted>
</import>
```

```
<entity name="Teacher">
    <attributes>
        <attribute datatype="string" mandatory="true" name="surname"/>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="string" mandatory="false" name="telephone"/>
        <attribute datatype="string" mandatory="false" name="mail"/>
        <attribute datatype="string" mandatory="false" name="webSite"/>
        <attribute datatype="string" mandatory="false" name="studentReception"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
    </attributes>
```

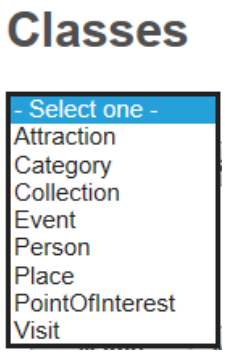
The Professor class has been replaced with Teacher; the former is labeled as "deleted" while the latter is rewritten with the same structure as the former.

Remove Class

To remove a class, it mustn't be present in any relationship. We will proceed with the following tests:

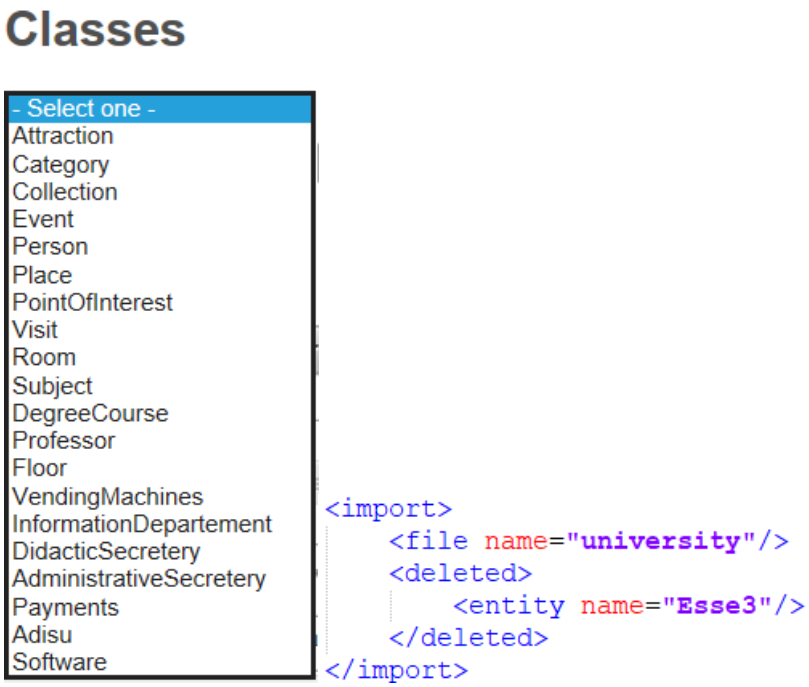
1. delete an existing class in the current domain only.
2. delete an existing class in an imported domain.

1.



After removing the associatedTo relation, the Multimedia class could be removed from the list.

2.



After importing the University domain, you can directly remove the Esse3 class since it has no relationships, and this is labelled as "deleted".

Class Attributes

To interact with attributes, it is necessary to select a class that has them. We proceed with the following tests:

1. add an attribute in an existing class in the current domain only.
2. add an attribute in an existing class in an imported domain.

3. modify name, mandatory, distinction and type of an attribute.
4. change the attribute type in select and see if you can add values.
5. change the attribute type to an entity, choose a target and verify that it is stored.

The function that allows inserting classes removed from imported domains in the deleted list is the same regardless of if the operation is insertion and modification and regardless of if you modify an attribute or a value. For this reason, only for the insertion of an attribute, a distinction is made between native classes in the domain and imported classes.

1.

```
<entity name="Attraction">
    <attributes>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="string" mandatory="false" name="description"/>
        <attribute datatype="select" mandatory="false" name="type">
            <values>
                <value name="WorkOfArt"/>
                <value name="Painting"/>
                <value name="Sculpture"/>
            </values>
        </attribute>
        <attribute datatype="date" mandatory="false" name="date"/>
        <attribute datatype="string" mandatory="false" name="technique"/>
        <attribute datatype="string" mandatory="false" name="material"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
        <attribute datatype="string" mandatory="true" name="nuovo"/>
    </attributes>
```

To the Attraction class, the new attribute "new" of type string has been inserted.

2.

```
<import>
    <file name="university"/>
    <deleted>
        <entity name="Professor"/>
    </deleted>
</import>
```

```
<entity name="Professor">
    <attributes>
        <attribute datatype="string" mandatory="true" name="surname"/>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="string" mandatory="false" name="telephone"/>
        <attribute datatype="string" mandatory="false" name="mail"/>
        <attribute datatype="string" mandatory="false" name="webSite"/>
        <attribute datatype="string" mandatory="false" name="studentReception"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
        <attribute datatype="boolean" mandatory="false" name="PhD"/>
    </attributes>
</entity>
```

The old Professor class is removed and the new one is rewritten with the additional attribute PhD.

3.

```
<entity name="Professor">
    <attributes>
        <attribute datatype="string" mandatory="true" name="surname"/>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="string" mandatory="false" name="telephone"/>
        <attribute datatype="string" mandatory="false" name="mail"/>
        <attribute datatype="string" mandatory="false" name="webSite"/>
        <attribute datatype="string" mandatory="false" name="studentReception"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
        <attribute datatype="string" mandatory="true" name="Dottorato"/>
    </attributes>
</entity>
```

The PhD attribute has been changed in the name (Dottorato), type and mandatory.

4.

Name	Mandatory	Distinguishi	Type
name	true	false	string
type	false	false	select
acronym	false	false	string
description	false	false	select
startDate	true	false	date
endDate	false	false	date
notes	false	false	text

Add Edit Remove

Values

Name
No records found.

Add Remove

In the Event class, the description type has been changed to select and the Add button of the Values section is available.

5.

After the modification of the description type in entity the Attraction class has been inserted as a target, this is saved when trying to remodify the attribute.

Class Values

To interact with the values, it is necessary to select an attribute of type select. We will proceed with the following tests:

1. add a value in an attribute.
2. remove a value in an attribute.

3. change the type of a select attribute to another and then change it back to select.

1.

```
<entity name="Event">
    <attributes>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="select" mandatory="false" name="type">
            <values>
                <value name="Conference"/>
                <value name="Exhibition"/>
                <value name="Fair"/>
                <value name="Show"/>
                <value name="Concert"/>
                <value name="Lecture"/>
                <value name="Historical_Event"/>
                <value name="Nuovo"/>
            </values>
        </attribute>
        <attribute datatype="string" mandatory="false" name="acronym"/>
        <attribute datatype="string" mandatory="false" name="description"/>
        <attribute datatype="date" mandatory="true" name="startDate"/>
        <attribute datatype="date" mandatory="false" name="endDate"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
    </attributes>
</entity>
```

The value "Nuovo" has been inserted among the possible values of the type of attribute in Event.

2.

```
<entity name="Event">
    <attributes>
        <attribute datatype="string" mandatory="true" name="name"/>
        <attribute datatype="select" mandatory="false" name="type">
            <values>
                <value name="Conference"/>
                <value name="Exhibition"/>
                <value name="Fair"/>
                <value name="Show"/>
                <value name="Concert"/>
                <value name="Lecture"/>
                <value name="Nuovo"/>
            </values>
        </attribute>
        <attribute datatype="string" mandatory="false" name="acronym"/>
        <attribute datatype="entity" mandatory="false" name="description"/>
        <attribute datatype="date" mandatory="true" name="startDate"/>
        <attribute datatype="date" mandatory="false" name="endDate"/>
        <attribute datatype="text" mandatory="false" name="notes"/>
    </attributes>
</entity>
```

The Historical_Event value has been removed from the possible type values.

3.

type	false	false	select
date	false	false	date
technique	false	false	string
material	false	false	string
notes	false	false	text

Add Edit Remove

Values

Name
No records found.

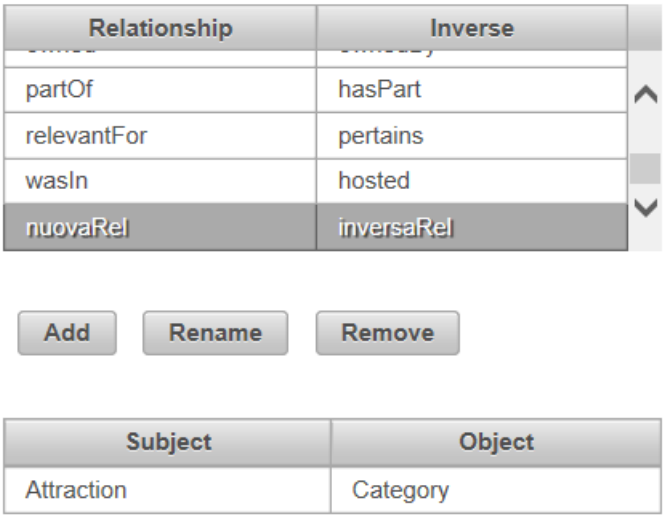
The type has been changed from select to string and vice versa and its old values are lost.

Add Relationship

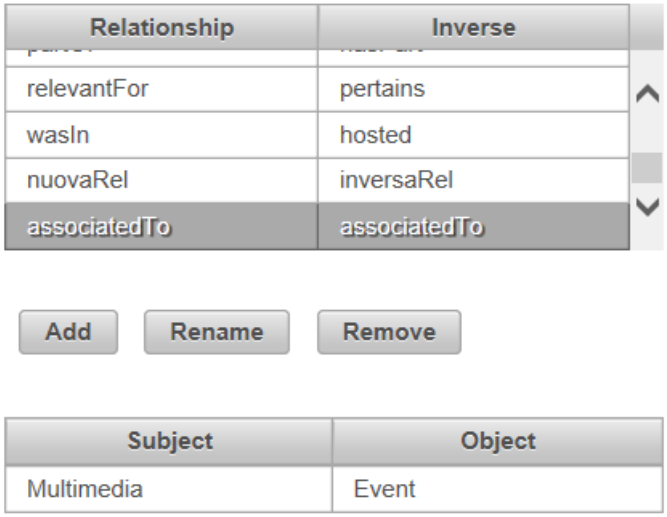
To add a relation there are no special requirements. We will proceed with the following tests:

1. enter a new report with a name that does not exist.
2. insert a relationship with an existing name only in the current domain.
3. insert a relation with an existing name in an imported domain.

1.

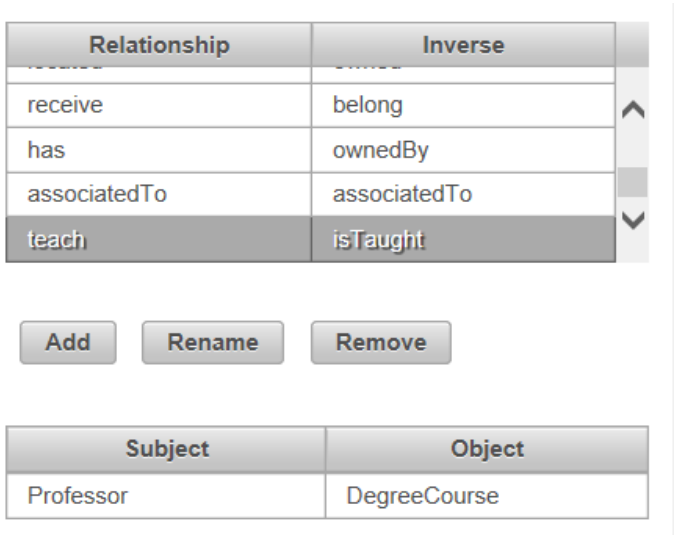


2.



A new associatedTo relationship with only one reference was created and replaced the existing one.

3.



Relationship	Inverse
receive	belong
has	ownedBy
associatedTo	associatedTo
teach	isTaught

Add Rename Remove

Subject	Object
Professor	DegreeCourse

```
<import>
    <file name="university"/>
    <deleted>
        <relationship name="teach"/>
    </deleted>
</import>
```

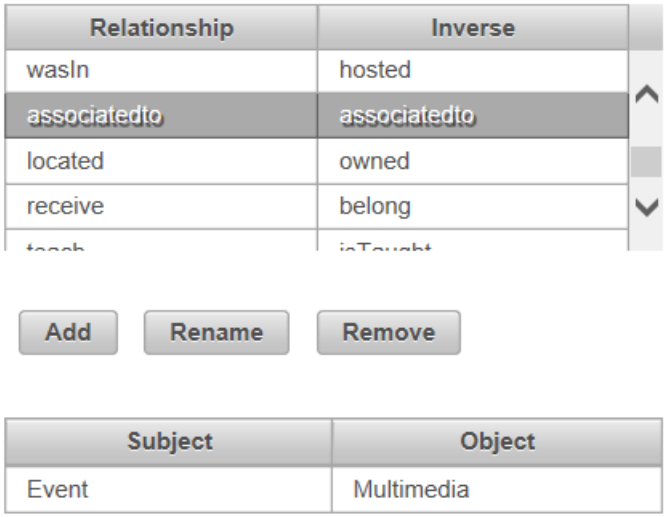
A new teach relationship has been created with a different reference than the imported one. The previous one is removed.

Rename Relationship

To rename you must select one. You will proceed with the following tests:

1. rename an existing relationship in the current domain only.
2. rename an existing relationship in an imported domain.

1.



Relationship	Inverse
wasIn	hosted
associatedto	associatedto
located	owned
receive	belong

Add Rename Remove

Subject	Object
Event	Multimedia

The associatedTo relationship has been renamed associatedto.

2.

```
<import>
    <file name="university"/>
    <deleted>
        <relationship name="teach"/>
    </deleted>
</import>

<relationship inverse="isTaught" name="teaches">
    <references>
        <reference object="Subject" subject="Professor"/>
    </references>
</relationship>
```

The teach relationship has been renamed and then removed. The new teaches relationship is rewritten by inheriting the references.

Remove Relationship

To remove a relation there are no preliminary steps. We will proceed with the following tests:

1. delete an existing relationship in the current domain only.
2. delete an existing relationship in an imported domain.

1.

Relationships

Relationship	Inverse
associatedto	associatedto
located	owned
receive	belong
has	ownedBy

The report receive was removed and labelled as "deleted"-

2.

```
<import>
    <file name="university"/>
    <deleted>
        <relationship name="teach"/>
        <relationship name="receive"/>
    </deleted>
</import>
```

La relazione receive è stata rimossa ed etichettata come “deleted”.

Relationship References

To interact with the references just select an attribute. We will proceed with the following tests:

1. add a reference.
2. remove a reference.

1.

Subject	Object
Multimedia	Place
Multimedia	Person
Multimedia	Visit
Multimedia	PointOfInterest

Add Remove

The Multimedia-PointOfInterest reference has been added to the associatedTo report.

2.

Subject	Object
Multimedia	Event
Multimedia	Place
Multimedia	Person
Multimedia	Visit

Add Remove

The Multimedia-PointOfInterest reference has been removed from the associatedTo report.

Relationship Attributes

To interact with attributes, it is necessary to select a relation that has them. We will proceed with the following tests:

1. add an attribute in an existing relationship only in the current domain.
2. add an attribute in an existing relation in an imported domain.
3. change the name, mandatory, distinction and type of an attribute.
4. change the type of an attribute in select and check if values can be added.
5. change the attribute type in an entity, choose a target and check if it is stored.

1.

```
<relationship inverse="includes" name="belongsTo">
    <references>
        <reference object="Collection" subject="Attraction"/>
        <reference object="Collection" subject="Event"/>
        <reference object="Collection" subject="Collection"/>
        <reference object="Category" subject="Person"/>
        <reference object="Category" subject="Place"/>
        <reference object="Collection" subject="Place"/>
        <reference object="Category" subject="PointOfInterest"/>
        <reference object="Collection" subject="PointOfInterest"/>
    </references>
    <attributes>
        <attribute datatype="integer" mandatory="false" name="firstAttribute"/>
    </attributes>
</relationship>
```

An integer attribute firstAttribute has been added to the belongsTo relation.

2.

```
<import>
    <file name="university"/>
    <deleted>
        <relationship name="teach"/>
    </deleted>
</import>

<relationship inverse="isTaught" name="teach">
    <references>
        <reference object="Subject" subject="Professor"/>
    </references>
    <attributes>
        <attribute datatype="integer" mandatory="false" name="year"/>
    </attributes>
</relationship>
```

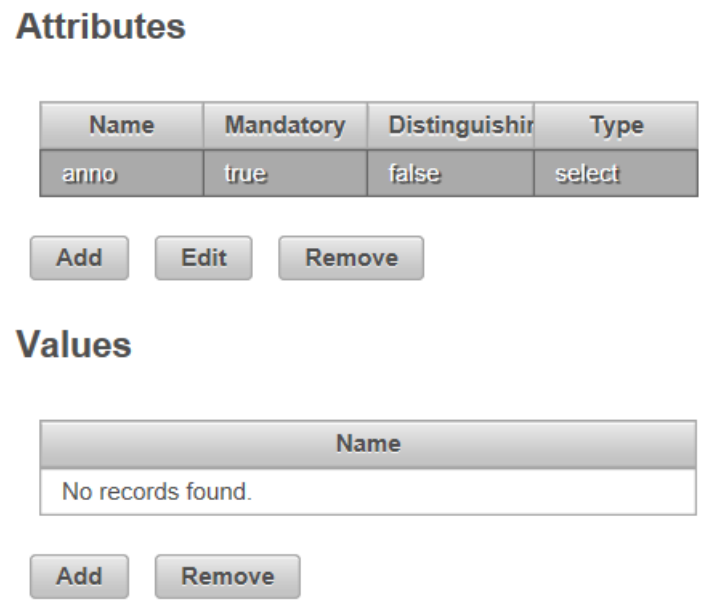
An integer attribute year has been added to the teach relation and the old teach relation is labelled as "deleted".

3.

```
<relationship inverse="isTaught" name="teach">
    <references>
        <reference object="Subject" subject="Professor"/>
    </references>
    <attributes>
        <attribute datatype="string" mandatory="true" name="anno"/>
    </attributes>
</relationship>
```

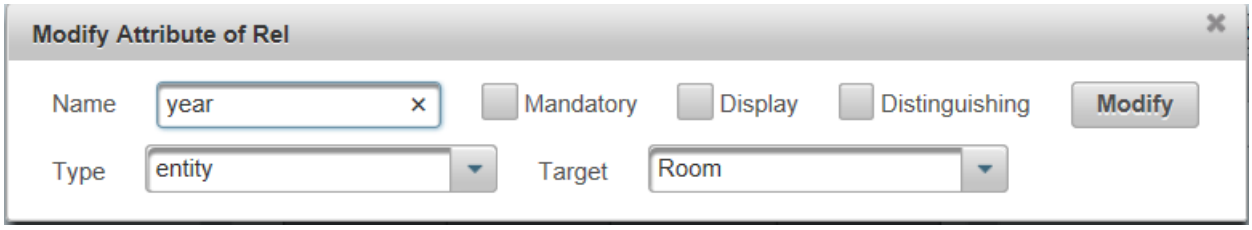
The year attribute has been changed in name (year), type and mandatory.

4.



New values can be added to the year attribute after changing its type in select.

5.



After changing the type of year in the entity you have inserted as target the class Room, this is saved when you try to remodify the attribute.

Relationships Values

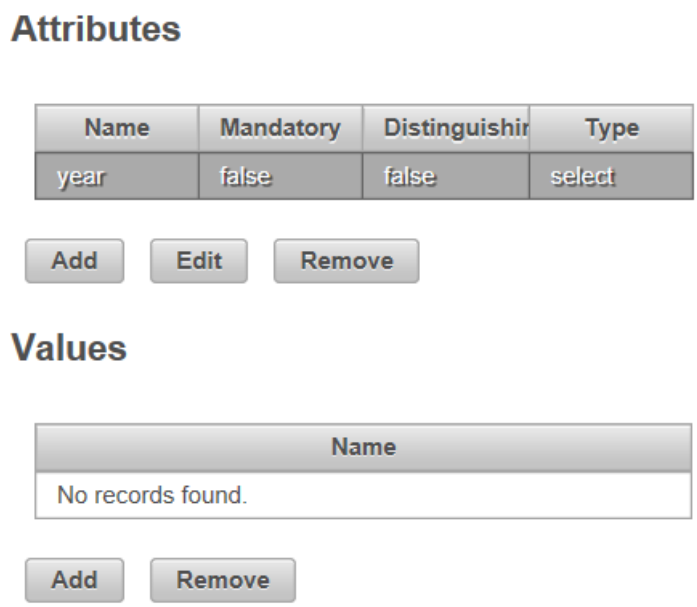
To interact with the values, it is necessary to select an attribute of type select. We will proceed with the following tests:

1. add a value in an attribute.
2. remove a value in an attribute.
3. change the type of a select attribute to another and then change it back to select.

1.

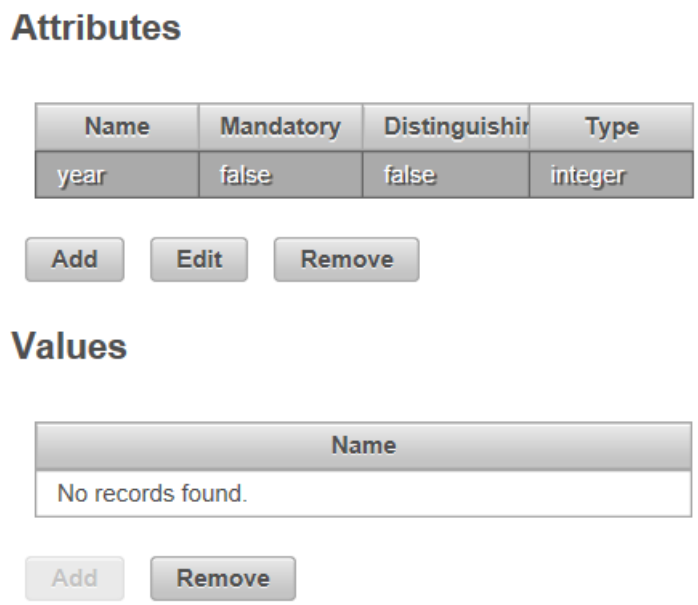
The value "first" has been inserted among the possible values of the attribute year in teach.

2.



The value "first" has been removed fromTo the values of year in teach.

3.



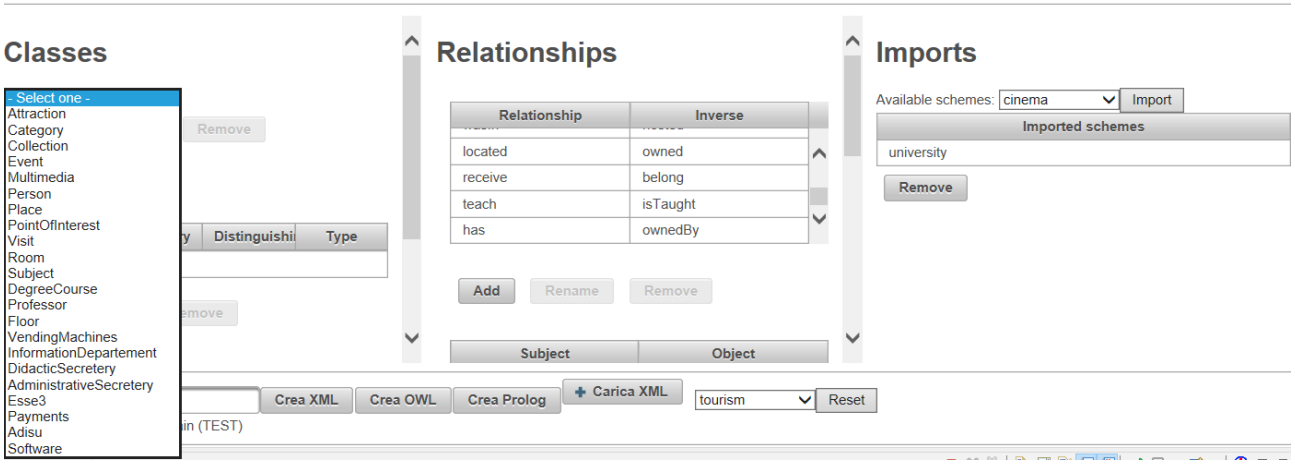
The year type was changed to select, a value was added and then it was remodified to integer causing it to lose its values.

Imports

You can import or remove domains at any time. You will run the following tests:

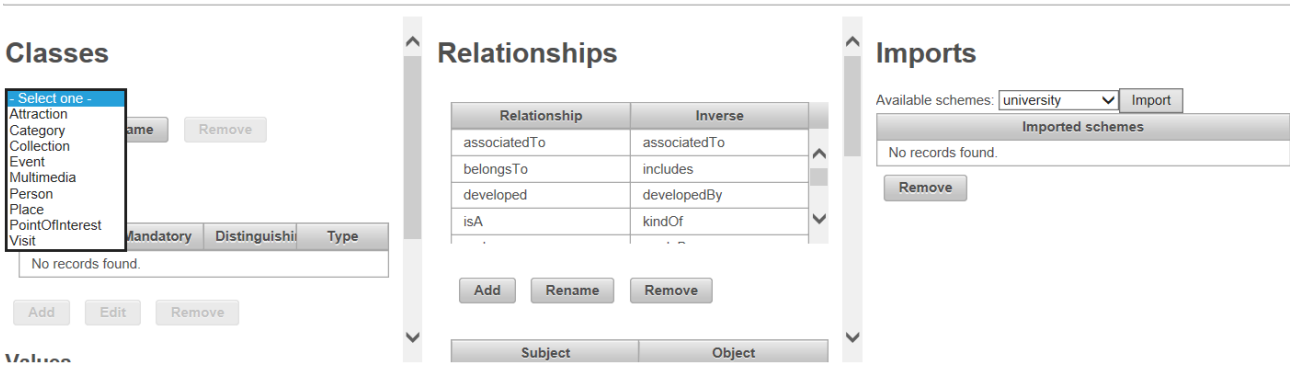
1. import a domain.
2. remove an imported domain.

1.



After importing a domain, new classes and relationships will be available and the entered domain will appear in the list.

2.



After removing a domain, its classes and relationships will be removed, and the domain will become available again among the importable.

Domain

To export and store changes made to a domain, you must provide methods to export. The following tests will be performed:

1. Export in XML.
2. Export in OWL.
3. Export in Prolog.

1.

```
<domain name="tourism">
    <import>
        <file name="university"/>
        <deleted>
            <relationship name="teach"/>
        </deleted>
    </import>
    <entities>
        <entity name="Attraction">
            <attributes>
                <attribute datatype="string" mandatory="true" name="name"/>
                <attribute datatype="string" mandatory="false" name="description"/>
                <attribute datatype="select" mandatory="false" name="type">
                    <values>
                        <value name="WorkOfArt"/>
                        <value name="Painting"/>
                        <value name="Sculpture"/>
                    </values>
                </attribute>
                <attribute datatype="date" mandatory="false" name="date"/>
                <attribute datatype="string" mandatory="false" name="technique"/>
                <attribute datatype="string" mandatory="false" name="material"/>
                <attribute datatype="text" mandatory="false" name="notes"/>
                <attribute datatype="select" mandatory="false" name="aa">
                    <values>
                        <value name="b"/>
                    </values>
                </attribute>
            </attributes>
        </entity>
```

```
        <relationship inverse="hosted" name="wasIn">
            <references>
                <reference object="Place" subject="Attraction"/>
                <reference object="PointOfInterest" subject="Attraction"/>
                <reference object="Collection" subject="Collection"/>
                <reference object="PointOfInterest" subject="Collection"/>
                <reference object="Place" subject="Event"/>
                <reference object="Event" subject="Person"/>
                <reference object="Period" subject="Person"/>
                <reference object="Collection" subject="Person"/>
                <reference object="Place" subject="Person"/>
                <reference object="PointOfInterest" subject="Person"/>
                <reference object="Collection" subject="Place"/>
                <reference object="Place" subject="Place"/>
                <reference object="Place" subject="PointOfInterest"/>
                <reference object="Event" subject="Visit"/>
                <reference object="Place" subject="Visit"/>
            </references>
            <attributes>
                <attribute datatype="string" mandatory="false" name="reason"/>
                <attribute datatype="string" mandatory="false" name="address"/>
                <attribute datatype="date" mandatory="false" name="startDate"/>
                <attribute datatype="date" mandatory="false" name="endDate"/>
                <attribute datatype="text" mandatory="false" name="notes"/>
            </attributes>
        </relationship>
        <relationship inverse="isTaught" name="teach">
            <references>
                <reference object="Subject" subject="Professor"/>
            </references>
            <attributes>
                <attribute datatype="select" mandatory="false" name="year"/>
            </attributes>
        </relationship>
    </relationships>
</domain>
```

When exporting to XML you get a file with this structure, only the initial and final fragments have been reported.

2.

```
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://owl.api.ontology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     ontologyIRI="http://owl.api.ontology">
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Declaration>
        <Class IRI="#Address"/>
    </Declaration>
    <Declaration>
        <Class IRI="#AdministrationService"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Administrative"/>
    </Declaration>
    <Declaration>
        <Class IRI="#AirTransport"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Architecture"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Archive"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Area"/>
```

```
    <DataPropertyRange>
        <DataProperty IRI="#attribute"/>
        <Datatype IRI="#integer"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#attribute"/>
        <Datatype IRI="#real"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#attribute"/>
        <Datatype IRI="#select"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#attribute"/>
        <Datatype IRI="#string"/>
    </DataPropertyRange>
    <DataPropertyRange>
        <DataProperty IRI="#attribute"/>
        <Datatype IRI="#text"/>
    </DataPropertyRange>
</Ontology>
```

Initial and final export fragments in OWL.

3.

Exporting to Prolog is critical for building an a posteriori knowledge base for performing queries on the ontology.

Stored as facts are the domain, all classes and subclasses with the domain in which the class was born, all attributes and values of select type attributes, domain native and imported relationships, with references and any attributes.

```
domain(tourism).
entity(tourism, attraction).
entity(tourism, category).
entity(tourism, collection).
entity(tourism, event).
entity(tourism, multimedia).
entity(tourism, person).
entity(tourism, place).
entity(tourism, pointofinterest).
entity(tourism, visit).
entity(university, room).
entity(university, subject).
entity(university, degreecourse).
entity(university, professor).
entity(university, floor).
entity(university, vendingmachines).
entity(university, informationdepartement).
entity(university, didacticsecretery).
entity(university, administrativesecretery).
entity(university, esse3).
```

```
relationship(tourism, wasin, collection, collection).
relationship(tourism, wasin, collection, pointofinterest).
relationship(tourism, wasin, event, place).
relationship(tourism, wasin, person, event).
relationship(tourism, wasin, person, period).
relationship(tourism, wasin, person, collection).
relationship(tourism, wasin, person, place).
relationship(tourism, wasin, person, pointofinterest).
relationship(tourism, wasin, place, collection).
relationship(tourism, wasin, place, place).
relationship(tourism, wasin, pointofinterest, place).
relationship(tourism, wasin, visit, event).
relationship(tourism, wasin, visit, place).
inverse(tourism, wasin, hosted).
relationship(tourism, located, room, floor).
relationship(tourism, located, secretery, floor).
relationship(tourism, located, vendingmachines, floor).
inverse(tourism, located, owned).
relationship(tourism, receive, professor, office).
inverse(tourism, receive, belong).
relationship(tourism, teach, professor, subject).
inverse(tourism, teach, istaught).
relationship(tourism, has, degreecourse, subject).
inverse(tourism, has, ownedby).
```

Prolog

In order to be able to perform queries on the ontology, it is necessary to extend what is exported from Gr@phBRAIN with a knowledge base that allows to easily retrieve the main information.

Rules:

- child/3: allows you to retrieve the parent of a class in a domain. The first two values are the domain and the entity name to avoid ambiguity. We can easily retrieve this information by exploiting the parent/3 predicates already present in the exported file. A class is a child of another if it is true that the other is its parent.
    - `child(D, S, F) :- parent(D, F, S).`

```
?- child(tourism, stuff, X).
X = category
```

- superclass/3: allows you to retrieve subclasses of a domain class. The definition uses parent and a recursive call to a higher level. Recursion is fundamental in a logical language, and the superclass definition can easily be made recursive by knowing that the superclass is either the parent of a class or the parent of a superclass of a class. The first value indicates the domain, as usual, the second the superclass, and the third the child. Taking advantage of parent/3 you can define the predicate.
    - `superclass(D, F, S) :- parent(D, F, S).`
    - `superclass(D, F, S) :- parent(D, F, M), superclass(D, M, S).`

```
?- superclass(tourism, category, X).
X = concept ? ;
X = period ? ;
X = stuff ? ;
X = subject ? ;
X = trend ? ;
no
```

- subclass/3: allows retrieving the subclasses of a class of a domain. The second and third values are inverted from the previous predicate. For this reason, it is easy to define subclass/3 concerning superclass/3.
    - `subclass(D, S, F) :- superclass(D, F, S).`

```
?- subclass(tourism, continent, X).
X = administrative ? ;
X = place ? ;
no
```

- attribute/4: allows you to retrieve attributes and types for each class. When generating the file in Prolog, we avoid repeating all the attributes inherited from a class; for this reason, it is necessary to also use the superclass definition to retrieve the inherited attributes as well. The attributes of the parents are also inserted.
    - `attribute(D, S, A, T) :- superclass(D, M, S), attribute(D, M, A, T).`

```
?- attribute(tourism, period, X, Y).
X = startdate,
Y = date ? ;
X = enddate,
Y = date ? ;
X = name,
Y = string ? ;
X = id,
Y = string ? ;
X = description,
Y = string ? ;
X = notes,
Y = text ? ;
```

- attribute/3: allows you to retrieve the names of attributes of a class. This is only needed to get a predicate with minor ariety.
    - `attribute(D, S, A) :- attribute(D, S, A, T).`

```
?- attribute(tourism, period, X).
X = startdate ? ;
X = enddate ? ;
X = name ? ;
X = id ? ;
X = description ? ;
X = notes ? ;
no
```

- entity/1: allows you to retrieve the classes that are present, regardless of the source domain.
    - `entity(X) :- entity(D, X).`

```
?- entity(X).
X = attraction ? ;
X = category ? ;
X = collection ? ;
X = event ? ;
X = multimedia ? ;
X = person ? ;
X = place ?
```

- imported/1: retrieves all imported domains. The sort predicate is used to remove duplicates in this case while the findall retrieves all present entity domains and puts them in a list. Sorting allows the removal of duplicates.
    - `imported(X) :- findall(D, entity(D, C), _Z), sort(_Z, X).`

```
?- imported(X).
X = [tourism,university]
```

- type/3: allows you to retrieve the type of an attribute of a class. It takes advantage of the attribute definition. The three values are class name, attribute name, and type.
    - `type(C, A, X) :- attribute(D, C, A, X).`

```
   ?- type(category, name, X).
X = string ? ;
```

- type/2: allows you to retrieve the types of attributes having the same name in the domain. Compared to the previous predicate, you don't have to specify the class because it takes all the attributes in the domain with that name. The two values are attribute name and type.
    - `type(A, X) :- attribute(D, C, A, X).`

```
   ?- type(type, X).
X = select ? ;
X = select ? ;
X = select ? ;
X = select ? ;
X = select ? ;
X = string ? ;
X = select ? ;
X = select ? ;
X = select ? ;
X = select ? ;
no
```

- topentity/1: allows you to retrieve classes that are not subclasses of other classes. Trivially, these are all entities that do not have superclasses.
    - `topentity(X) :- entity(X), not superclass(D, F, X).`

```
?- topentity(X).
X = attraction ? ;
X = category ? ;
X = collection ? ;
X = event ? ;
X = multimedia ? ;
X = person ? ;
X = place ? ;
X = pointofinterest ? ;
X = visit ? ;
X = room ?
```

- bottomentity/1: allows you to retrieve classes that have no further subclasses.
    - `bottomentity(X) :- entity(X), not subclass(D, S, X).`

```
?- bottomentity(X).
X = attraction ? ;
X = collection ? ;
X = event ? ;
X = multimedia ? ;
X = person ? ;
X = visit ? ;
X = subject ?
```

- values/3: allows you to retrieve values that can take select type attributes with the same name. It simply reduces the ariety.
    - `values(C, A, X) :- values(D, C, A, X).`

```
?- values(collection, type, X).
X = [family,group,place,series]
```

- inherited/2: allows you to retrieve attributes inherited from superclasses. It searches among the attributes of a class those that are also present in at least one superclass. The two values are class name and inherited attribute.
    - `inherited(C, X) :- attribute(C, X, Y), superclass(D, F, C), attribute(D, F, X, Y).`

```
   ?- inherited(period, X).
X = name ? ;
X = id ? ;
X = description ? ;
X = notes ? ;
no
```

- notinherited/2: allows you to retrieve the attributes of a class that have not been inherited by others. It searches among the attributes of all those not present in the superclasses using the previously defined and negated definition of inherited. If a class is topentity it cannot inherit anything. The two values are class name and attribute not inherited.
    - `notinherited(C, X) :- attribute(C, X, Y), topentity(C).`
    - `notinherited(C, X) :- attribute(C, X, Y), not inherited(C, X).`

```
   ?- notinherited(period, X).
X = startdate ? ;
X = enddate ? ;
no
```

- mandatory/2, distinguishing/2, target/3: allow you to retrieve information about the mandatory, distinct and target attributes respectively. They simplify the search by avoiding re-entering the domain.
    - `mandatory(C, X) :- mandatory(D, C, X).`
    - `distinguishing(C, X) :- distinguishing(D, C, X).`
    - `target(C, A, X) :- target(D, C, A, X).`

```
   ?- mandatory(person, name).
yes
   ?- distinguishing(person, name).
no
   ?- target(person, bornin, X).
X = place
```

- relclass/2: allows you to retrieve all relationships in which a class is involved. It searches where the class is present as the subject or object of the relationship. The two values are class name and relationship name.
    - `relclass(C, X) :- relationship(D, X, C, O).`
    - `relclass(C, X) :- relationship(D, X, S, C).`

```
?- relclass(attraction, X).
X = belongsto ? ;
X = relevantfor ? ;
X = wasin ? ;
X = wasin ? ;
X = developed ? ;
X = owned ? ;
X = owned ? ;
X = relevantfor
```

- subject/2: This allows you to retrieve all the subjects of a relationship, i.e. the third value of a relationship. The two values are relationship and subject.
    - `subject(R, X) :- relationship(D, R, X, O).`

```
?- subject(teach, X).
X = professor
```

- object/2: retrieves all objects of a relation, which is the fourth value of a relationship. The two values are relationship and object.
    - `object(R, X) :- relationship(D, R, S, X).`

```
?- object(associatedto, X).
X = event ? ;
X = visit ? ;
X = place ? ;
X = person
```

We believe that these rules are satisfactory for providing all the information in the ontology as they can provide exactly the same information present on the XML export. The exception is writing relationships and entities removed from other domains since that information is only useful in XML which can be used as a new source for importing domains.

II.II Importing Ontologies in Neo4j

After adjusting the Schema section according to our needs, we need a procedure to read any file in RDF/XML format that can be imported into Gr@phBRAIN. For this purpose, we listed some attempts above and we provide also a new strategy for our purposes. The general idea is to exploit Apache Jena which is written in Java.

II.II.I Design

Our solution is based on importing an OWL/RDF file, translating it into triples and then reading the different types of triples and applying node and relation translation. There are different operations we have to deal with. Specifically, we can distinguish:

- creating an instance of a particular node.
- adding an attribute to a particular node.
- adding a relationship that links two nodes.

1. Creating an instance

In particular, if the triple includes a predicate with an IRI like this <.../*rdf-syntax-ns#type*>, it means that an instance must be created. In this case, a node is added in the Neo4j graph database (if it does not exist yet). The newly added node adopts the following features: the node label is expressed through its IRI (specified in the triple subject), while the node type is specified in the triple object. An example of this kind of triple is shown below:

<http://www.semanticweb.org/ontologies/Hotel.owl#d24_23>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.semanticweb.org/ontologies/Hotel.owl#Distance_By_Walking> .

2. Adding an attribute

If the triple includes an object as a string, it means that an attribute must be added to a node specified in the triple subject. There could be two different cases: if the node already exists, the attribute is just added to the node; while if it doesn't exist yet, the node is firstly entered in the graph and then the attribute is added. Besides, the attribute name is specified in the triple predicate (the last element of the IRI path). An example of this kind of triple is shown below:

<http://www.semanticweb.org/ontologies/Hotel.owl#d16_1>
<http://www.semanticweb.org/ontologies/Hotel.owl#hasDistance>
"4.147"^^<http://www.w3.org/2001/XMLSchema#double> .

3. Adding a relationship

If the triple contains a subject and an object expressed as resources (nodes including resource IRIs), then it means that the predicate necessarily corresponds to a relationship and so an arch must be added between the two nodes present on the left and the right of the triple. The arch is added if the left and right nodes already exist in the graph, otherwise, the nodes are first entered into the graph and then the arch linking them is generated. Furthermore, the arch label is expressed in the triple predicate (the last element of the IRI path). An example of this kind of triple is shown below:

<http://www.semanticweb.org/ontologies/Hotel.owl#d9_10>
<http://www.semanticweb.org/ontologies/Hotel.owl#isDistanceByWalkingFor>
<http://www.semanticweb.org/ontologies/Hotel.owl#attraction_1> .

Another important stage is to create a mapping between information present in the KB and that present in Gr@phBRAIN. To reach this objective, a translation table has been developed. In this section, the design of the translation is reported. The table aims to reach a dual objective:

• translating the triples expressed in the English (or whatever) language into new triples expressed in another language (in this case in Italian).

• transforming, if needed, the attributes into relationships.

The structure of the chosen table consists of three columns in which: the first one represents the original names; the second one represents the translation and while the third column represents the type (class) of the new node created when an attribute becomes a relationship. The third field column is not mandatory, but it appears just in the last case.

The format used to define entities, relationships and attributes in this table is the following:

entity: entityName | relationship: relationshipName | attribute: attributeName | class: className

This format has been chosen both to keep track of the names in the two different languages and to check the possible transformation. The prefix that has been employed is the following: entity, relationship, attribute, class. They are useful to specify the type of elements that must be translated or, maybe, transformed. This format has been selected specially to make the transformation process simpler.

To better understand all the cases that can occur, a clearer example is reported here. It justifies the choice of the particular table format since it can catch all the possible variations explained before. If the element is an entity, the translation is made in this way:

- entity: Agent, entity: Agente

If the element is a relationship just the translation is made:

- relationship: hasAgent, relationship: haAgente

If the element is an attribute and it is decided to keep the attribute, just the translation is made:

- attribute: description, attribute: descrizione

If the element is an attribute and it is decided to transform the attribute into a relationship, both the translation and the transformation must be specified in this way:

- attribute: keyword, relationship: haParolaChiave, class: ParolaChiave

II.III Translating RDF into XML

To extract parts of a graph, we need to know the ontology we want to get read so that we can identify the classes or properties to which the graph refers. We provide this procedure so that you can read any ontology, either directly written in the Gr@phBRAIN xml or by reading a generic rdf file and translate it into the xml that the following procedure will read. Also, for this procedure, Apache Jena will be used.

II.III.I Design

Apache Jena provides procedures for parsing rdf documents that are automatically imported as ontologies. After that, there are procedures to be able to scan all classes, properties and individuals. The latter will not be considered since the individuals will be extracted from the graph. Through the libraries

org.xml.sax and org.w3c.dom you can write xml files. We provide here an example of the output produced by our procedure. Apache Jena was chosen instead of OWL API because the latter does not have methods for importing rdf, only OWL. It would have been necessary to create an ad hoc procedure that could hardly compete with the performance of the methods already present in Jena.

This is an example of the output produced by the procedure:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<domain name="Hotel">
  <entities>
    <entity name="Good_Hotel"/>
    <entity name="Rank"/>
    <entity name="Site">
      <attributes>
        <attribute datatype="integer" mandatory="false" name="hasPrice"/>
      </attributes>
      <taxonomy>
        <value name="Accomodation">
          <taxonomy>
            <value name="Camping"/>
            <value name="Residence"/>
            <value name="Bed_and_Breakfast"/>
            <value name="Hostel"/>
            <value name="Hotel">
              <taxonomy>
                <value name="Hotel_5_Stars"/>
                <value name="Hotel_2_Stars"/>
                <value name="Hotel_4_Stars"/>
                <value name="Hotel_1_Star"/>
                <value name="Hotel_3_Stars"/>
              </taxonomy>
            </value>
          </taxonomy>
        </value>
        <value name="Attraction">
          <taxonomy>
            <value name="Square"/>
```

```
            <value name="Bridge"/>
            <value name="Theater"/>
            <value name="Shopping_Center"/>
            <value name="Park"/>
            <value name="Tower"/>
            <value name="Stadium"/>
            <value name="Center"/>
          </taxonomy>
        </value>
        <value name="Civic">
          <taxonomy>
            <value name="Police_Station"/>
            <value name="Town_Hall"/>
            <value name="Hospital"/>
            <value name="University"/>
            <value name="Fire_Station"/>
          </taxonomy>
        </value>
        <value name="Station">
          <taxonomy>
            <value name="Port"/>
            <value name="Bus_Station"/>
            <value name="Subway_Station"/>
            <value name="Train_Station"/>
            <value name="Airport"/>
          </taxonomy>
        </value>
      </taxonomy>
    </entity>
    <entity name="Amenity">
      <taxonomy>
        <value name="Public_facility">
          <taxonomy>
            <value name="Pet_Allowed"/>
            <value name="Swimming_Pool"/>
            <value name="Safety_box"/>
            <value name="Bar"/>
            <value name="Cradle"/>
            <value name="Conference_Hall"/>
            <value name="WI-FI"/>
```

```
            <value name="Disabled_Facilities"/>
            <value name="Parking"/>
            <value name="24h_Reception"/>
            <value name="Health_Clinic"/>
            <value name="Sport_facility">
              <taxonomy>
                <value name="Beach-Volley"/>
                <value name="Sub"/>
                <value name="Surf"/>
                <value name="Bike"/>
                <value name="Pool"/>
                <value name="Golf_Course"/>
                <value name="Bowling"/>
                <value name="Table_Tennis"/>
                <value name="Tennis_Court"/>
                <value name="Sail"/>
                <value name="Minigolf"/>
                <value name="Gym"/>
                <value name="Boat"/>
              </taxonomy>
            </value>
            <value name="Beach_Umbrella"/>
            <value name="Laundry"/>
            <value name="Wellness_Center"/>
            <value name="Booking_Service"/>
            <value name="Sunbed"/>
            <value name="Babysitting"/>
            <value name="Internet_Access_Point"/>
            <value name="Restaurant"/>
          </taxonomy>
        </value>
      </taxonomy>
    </entity>
    <entity name="Distance_By_Walking"/>
    <entity name="Not_Good_Hotel"/>
    <entity name="Place"/>
  </entities>
  <relationships>
    <relationship inverse="" name="hasRank">
      <references>
```

```
        <reference object="Rank" subject="Hotel"/>
      </references>
    </relationship>
    <relationship inverse="isDistanceByWalkingFor" name="hasDistanceByWalking">
      <references>
        <reference object="Distance_By_Walking" subject="Site"/>
      </references>
    </relationship>
    <relationship inverse="" name="hasAmenity">
      <references>
        <reference object="Amenity" subject="Hotel"/>
      </references>
    </relationship>
  </relationships>
</domain>
```

II.IV Reading XML Ontologies

Once you have obtained the XMLs in Gr@phBRAIN format, taken either through the translation described above, or from the export of Gr@phBRAIN itself, you need to create a procedure to import them. After importing the ontology, you can proceed with the extraction of a part of the graph from Neo4j. It is then necessary to accurately describe how the process of translation from XML to ontology schema takes place. For each possible node, it will be described how it will be treated.

II.IV.I Design

We now describe the components of the ontology. All classes are subclasses of Thing. We create a Relationship class in which we will put all reified relationships. We do this so that we can have the data properties on the relationships. Since each given property has a class as its domain and codomain, we need to create a class for each relationship.

When an attribute is of type "date", we need to create the classes Year, Month and Day and create the relationships with the three classes. When an attribute is of type "entity", we create the relationship with domain the starting class and codomain the class indicated with "target". For relations, each reference is a subrelation, but subrelationships are not reified. Subrelationships are used to differentiate domain and codomain and will be named relationship_domain_codomain.

We summarize the behaviour of the parser. For each node, we describe the behaviour of the parser.

<domain>: ignored.

<entities>: starting node of the entities. Ignored.

<entity>: class of the ontology. The parser will create a new class.

<attributes>: starting node of the attributes. Ignored.

<attribute>: attribute of the class. If the datatype is not “date” or “entity”, the parser will create a datatype property having domain the entity found above and range its datatype.

<attribute datatype=”select”>: attribute of the class. The parser will create a datatype property setting specific as range specific values.

<attribute datatype=”entity”>: relationship of the class. The parser will create an object property setting as domain the last-mentioned class and as range the target value.

<values>: starting node of the values. Ignored.

<value>: the possible value of the attribute of type select. The parser will add this value in the range.

<taxonomy><value>: subclass of a class. The parser will create a new class, child of the first-mentioned above.

<relationships>: starting node of the relationships. Ignored.

<relationship>: relationship between classes. The parser will create an object property. If the inverse exists, it will be created. The parser will also reify the relationship by adding a class (subclass of Relationship) named as the relationship name. This is done to have individuals of type relationships which can have data properties.
<references>: starting node of relationships children. Ignored.
<reference>: relationship between classes. The parser will create an object property with the domain the subject value and range the object value.

II.V Extract Subgraph

After reading the ontology, you need to import nodes and arcs from the graph. Several algorithms exist in the literature for extracting meaningful (or interesting) parts of the graph. The best known are based on the weight of nodes as in [31] or the similarity between nodes as in [32]. Our first approach is based on extracting nodes that are less than or equal to k away from a given node. This is not a particularly clever approach, but it is useful for starting the mapping and completing the reasoning process. Moreover, the mechanism by which nodes are exported does not affect either import or reasoning since the proposed solution is general.

II.V.I Design

For the connection between a system written in Java and Neo4j, there is a driver, as already mentioned. The latter allows you to open a connection, close it and execute queries written in Cypher. All operations on the graph will be performed by writing queries. For the extraction of nodes at most k distant from a given node, it is necessary to implement a procedure that writes the right query according to the chosen k. For k = 1 the query for retrieving nodes and arcs can be as follows:

```
START nod=NODE(2665) MATCH (nod)-[r0]-(n0) return nod, r0, n0.
```

For k = 2, it would be slightly different:

```
START nod=NODE(2665) MATCH (nod)-[r0]-(n0)-[r1]-(n1) return nod, r0, n0, r1, n1.
```

As you can easily see, what changes are only the list of node-relationships and the list of what you want to return.

We decided to have the query result returned in json format. This way, we can save the file to a path of our choosing and read the contents. From parsing the file, we have all the information about the individuals and relationships to import. We report here the query to save the result in json format:

```
call apoc.export.json.query("START nod=NODE(2665) MATCH (nod)-[r0]-(n0) return nod, r0, n0",
"subgraph.json");
```

This is an example of the content of the json file:

{"n0":{"type":"node","id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"],"properties":{"comment":"Via Filippo Mazzei 2 pisa","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_001","hasPrice":90}},"r1":{"id":"2533","type":"relationship","label":"hasDistanceByWalking","start":{"id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"]},"end":{"id":"203","labels":["Distance_By_Walking","Resource","Thing"]}},"n1":{"type":"node","id":"203","labels":["Distance_By_Walking","Resource","Thing"],"properties":{"hasDistance":5.1,"comment":"from: hotel_001\nto: square_3","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#d1_2"}},

{"n0":{"type":"node","id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"],"properties":{"comment":"Via Filippo Mazzei 2 pisa","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_001","hasPrice":90}},"r1":{"id":"2951","type":"relationship","label":"hasDistanceByWalking","start":{"id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"]},"end":{"id":"749","labels":["Distance_By_Walking","Resource","Thing"]}},"n1":{"type":"node","id":"749","labels":["Distance_By_Walking","Resource","Thing"],"properties":{"hasDistance":6.7,"comment":"from: hotel_001\nto: bus_station_2","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#d1_13"}},

{"n0":{"type":"node","id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"],"properties":{"comment":"Via Filippo Mazzei 2 pisa","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_001","hasPrice":90}},"r1":{"id":"3418","type":"relationship","label":"hasDistanceByWalking","start":{"id":"597","labels":["Good_Hotel","Hotel","Resource","Thing"]},"end":{"id":"757","labels":["Distance_By_Walking","Resource","Thing"]}},"n1":{"type":"node","id":"757","labels":["Distance_By_Walking","Resource","Thing"],"properties":{"hasDistance":6.7,"comment":"from: hotel_001\nto: station_2","uri":"http://www.semanticweb.org/ontologies/Hotel.owl#d1_23"}}.

II.VI Wrapper

Considering the ontological translation described in Section II.III, the problem emerges of importing relations (and related nodes) that result from mandatory attributes. For this reason, we have implemented a Wrapper class that, taking into account the mandatory attributes (which have become relations) goes to increase the number of imported nodes and arcs creating an augmented graph.

II.VI.I Design

Our first idea was to take advantage of the reasoner for this operation, i.e., allow it to automatically identify missing nodes and arcs and then, given the explanation, use the wrapper to import what was missing and restart the reasoner. In the state of the art, this is not possible for two reasons: the reasoner in OWL API (but also in Jena) is not smart enough to stop and restart the process from a checkpoint. If we think of having to load a part of the graph every time and restart the reasoner, it is easy to imagine that the performance would drop significantly. The second reason is probably even more relevant. Reasoners do not have a mechanism in them to be able to store attributes and/or mandatory relationships to be maintained in the graph. Given these difficulties, we have expanded the work on the wrapper, which not only has to load the missing nodes but also must verify that the new nodes do not have any new mandatory ones. It

is easy to imagine that the procedure is iterative and always arrives at convergence.

During parsing, for each class (or subclass) we store all the mandatory relationships to be imported. This way, when we encounter an instance of a class, we know what other paths to travel in the graph. We report here an example of a query in Cypher that allows you to read the relationships and associated nodes for a specific node and a given relationship:

```
START nod=NODE(2665) MATCH (nod)-[r0]-(n0) return nod, r0, n0.
```

By calling this query for each instance of a class and each mandatory relationship, we have completed the extraction of nodes of a class. This must be repeated for each class until convergence. We must also know that a node extracted from the wrapper may, in turn, require new relationships. As already done for extraction, additional nodes will be exported in json format. These files will be read with the same procedure mentioned above for import.

II.VII Reasoning

After extracting the desired nodes and importing the ontology, you can use one of the reasoners offered by OWL API. The operations possible with reasoners mainly concern hierarchies between classes. The most useful operation can be to check the consistency of the knowledge base.

II.VII.I Design

We show an example of how to exploit the reasoner to verify the consistency (before and after the extraction of the individuals), to visualize the whole chain of subclasses and, given a class, to identify all its subclasses (also derived).

We show the following results: consistency is guaranteed, a portion of the subclass chain and all subclasses of the Site class, exploiting the example of the Hotel ontology mentioned earlier.

Is the ontology consistent? True
Is it consistent? True

Subclasses of Site :
Nodeset[Node(<http://www.semanticweb.org/ontologies/Hotel.owl#Park>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_3_Stars>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Fire_Station>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Subway_Station>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Station>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Bed_and_Breakfast>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Camping>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Tower>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hospital>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#University>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Bridge>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Shopping_Center>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_5_Stars>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Port>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Square>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Airport>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Center>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Police_Station>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_2_Stars>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Stadium>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hostel>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Train_Station>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_4_Stars>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Hotel_1_Star>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Town_Hall>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Theater>), Node(owl:Nothing), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Residence>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Accomodation>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Bus_Station>), Node(

<http://www.semanticweb.org/ontologies/Hotel.owl#Attraction>), Node(
<http://www.semanticweb.org/ontologies/Hotel.owl#Civic>)]

Axiom :- SubClassOf(<http://www.semanticweb.org/ontologies/Hotel.owl#WI-FI>
<http://www.semanticweb.org/ontologies/Hotel.owl#Public_facility>)
Is axiom entailed by reasoner ? :- true
Is axiom contained in ontology ? :- true
No. of Explanations :- 1
Explanation :-
[SubClassOf(<http://www.semanticweb.org/ontologies/Hotel.owl#WI-FI>
<http://www.semanticweb.org/ontologies/Hotel.owl#Public_facility>)]

Axiom :- SubClassOf(<http://www.semanticweb.org/ontologies/Hotel.owl#Town_Hall>
<http://www.semanticweb.org/ontologies/Hotel.owl#Civic>)
Is axiom entailed by reasoner ? :- true
Is axiom contained in ontology ? :- true
No. of Explanations :- 1
Explanation :-
[SubClassOf(<http://www.semanticweb.org/ontologies/Hotel.owl#Town_Hall>
<http://www.semanticweb.org/ontologies/Hotel.owl#Civic>)]

II.VIII Integration in Gr@phBRAIN

Considering the strong correlation between Gr@phBRAIN and the node extraction procedure, we complete the experiment by integrating what has been done inside the web application. In this way, we can exploit the connection with the Gr@phBRAIN database to simulate real operations.

II.VIII.I Design

We decided to include three functionalities in the Schema section: the import of a knowledge base in the graph, the translation of a file from RDF to XML and the extraction of a subgraph given the parameters: ontology of the portion you want to extract, id of the central node, prefix of the new KB, distance from the central node and path of the export file. At the end of the operation, the export file will

contain the following information: consistency of the ontology, all subclasses of a given class, and all subclass relations present in the ontology. These are only examples of the possible operations with the reasoner.

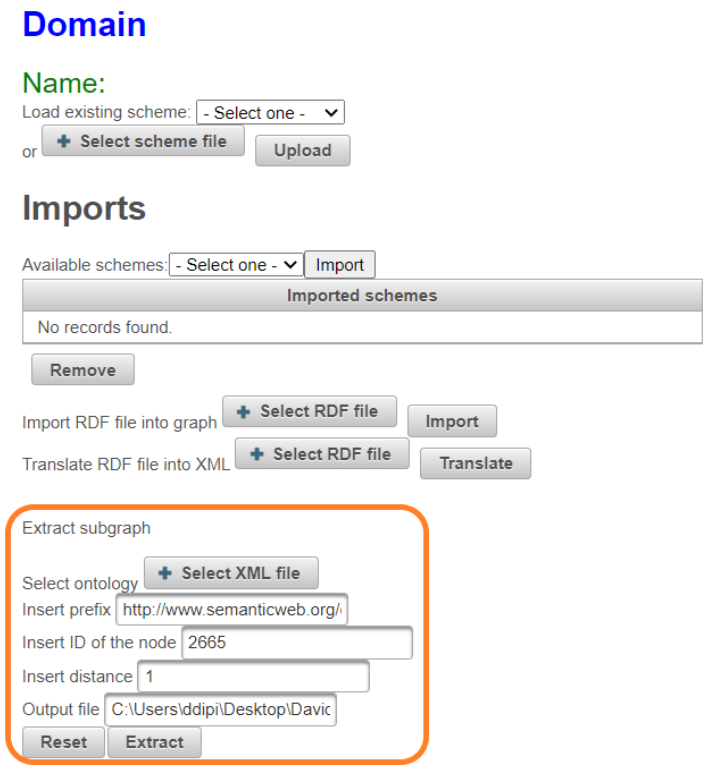


Figure 44: Integration section in Gr@phBRAIN

The section in the orange box is dedicated to the integration.

III. Conclusions and Future Works

We believe we have built a system that can effectively extract parts of a graph and perform operations not possible with Neo4j, thanks to the introduction of Semantic Web tools. This work lends itself well to numerous extensions. First, there are several ways through which significant nodes of a graph can be extracted. In the Gr@phBRAIN system, some algorithms are already available for visualizing parts of the graph, like Katz Centrality [33] and PageRank Centrality [34]. It is also possible, if you extract a large part of a graph, to use Data Mining

techniques to be able to extract new knowledge and/or create subparts of the original knowledge base. There is a further possibility of using the reasoner. As it is open source, it is possible to download the source code and modify the search for individuals, so that customizations can be made according to the intended purpose. The source code of the HermiT reasoner can be download here: http://www.hermit-reasoner.com/.

IV. Appendix

Here we show the implementation of our solution.

IV.I Schema Tab

Here we show the implementation of the different Schema Tab functions.

- getChildren

This method gets the children of an entity.

```
private void getChildren(Entity entity) {
    if(entity.getParent() != null) {
        subentities.add(entity);
        subentitiesToString.add(entity.getName());
    }
    if(entity.getChildren() != null) {
        for(Entity e : entity.getChildren()) {
            getChildren(e);
        }
    }
}
```

The method recursively calls to retrieve all children up to the leaf classes of the imaginary tree.

Once a subclass is selected, it will be stored in the selectedSubEnt variable and its name in selectedSubEntName.

- getEntityFromSub

This method retrieves the entity given its name.

```
private Entity getEntityFromSub(String selected) {
    for(Entity e : subentities) {
        if(e.getName().equalsIgnoreCase(selected)) {
            return e;
        }
    }
    return null;
}
```

- addChildNode

This method adds a children class.

```
public void addChildNode(String domain) throws IOException {
    if(domain!=null && newNodeName!=null) {
        if(selectedParent.equals("")) {
            DomainData domainData = Domains.getDomain(domain);
            if(domainData.getEntitiesToString().contains(newNodeName)) {
                if(!schema.getEntity(newNodeName).getDomain().equalsIgnoreCase(domain)) {
                    logs = "entity (" + newNodeName + ") removed\n";
                    write(logs);
                    entitiesToRemove.add(newNodeName);
                }
                Vector<Relationship> relationshipsToRemove = new Vector<>();
                for(Relationship r : domainData.getRelationships()) {
                    Vector<Reference> refsToRemove = new Vector<>();
                    for(Reference ref : r.getReferences()) {
                        if(ref.getSubject().equalsIgnoreCase(newNodeName) || ref.getObject().equalsIgnoreCase(newNodeName)) {
                            refsToRemove.add(ref);
                        }
                    }
                    r.removeAll(refsToRemove);
                    if(r.getReferences().size() == 0) {
                        relationshipsToRemove.add(r);
                    }
                }
                domainData.getRelationships().removeAll(relationshipsToRemove);
                schema.removeEntity(newNodeName);
            }
            Entity newEntity = new Entity(newNodeName, domain);
            schema.addEntity(newEntity);
            logs = "entity (" + newNodeName + ") created\n";
            write(logs);
        }else {
            Entity parent = getEntityFromSub(selectedParent);
            if(!parent.getDomain().equalsIgnoreCase(domain)) {
                entitiesToRemove.add(selectedEnt);
                schema.getEntity(selectedEnt).setDomain(domain);
            }
            if(parent.getChildrenToString().contains(newNodeName)) {
```

```
                logs = "entity (" + newNodeName + ") child of " + parent.getName() + " removed\n";
                write(logs);
                Entity toRemove = parent.getChild(newNodeName);
                parent.getChildren().remove(toRemove);
                //subentities.remove(toRemove);
                removeChildren(toRemove);
            }
            Entity newEntity = new Entity(newNodeName, domain);
            parent.addChild(newEntity);
            //System.out.println(parent.name + " " + parent.getChildren().size());
            newEntity.setParent(parent);
            subentities.add(newEntity);
            logs = "entity (" + newNodeName + ") child of " + parent + " created\n";
            write(logs);
        }
    reload();
    newNodeName = null;
    }
}
```

It is obvious that the domain in which the entity is created is the current domain. In the method we distinguish whether we are adding a top-level class or a subclass through the selectedParent variable. In case selectedParent is validated, the class named selectedParent is retrieved through the getEntityFromSub method given here.

- addEntity

This method adds a new entity.

```
public void addEntity(Entity entity) {
    Entity found = null;
    for(Entity e : entities) {
        if(e.getName().equalsIgnoreCase(entity.getName())) {
            found = e;
        }
    }
    if(found != null) {
        entities.remove(found);
    }
    entities.add(entity);
}
```

In the DomainData.java class, there is a list of entities called entities that collects all the entities in the domain.

- renameChildNode

This method renames a subclass.

```
public void renameChildNode(String domain) {
    if(domain!=null && newName!=null) {
        DomainData domainData = Domains.getDomain(domain);
        Entity old = domainData.getEntity(selectedEnt);
        System.out.println(selectedSubEntName);
        if(selectedSubEntName.equals("")) {
            if(!old.getDomain().equalsIgnoreCase(domain)) {
                Vector<Relationship> toRemove = schema.getRelationships(old);
                for(Relationship r : toRemove) {
                    Relationship newRelationship = new Relationship(domain, r.getName(), r.getInverse());
                    newRelationship.setReferences(r.getReferences());
                    newRelationship.setAttributes(r.getAttributes());
                    for(Reference ref : newRelationship.getReferences()) {
                        if(ref.getSubject().equalsIgnoreCase(old.getName())) {
                            logs = "relation_subject (" + r.getName() + ", " + ref.getSubject() + ") modify " + newName + "\n";
                            write(logs);
                            ref.setSubject(newName);
                        }
                        if(ref.getObject().equalsIgnoreCase(old.getName())) {
                            logs = "relation_object (" + r.getName() + ", " + ref.getObject() + ") modify " + newName + "\n";
                            write(logs);
                            ref.setObject(newName);
                        }
                    }
                    schema.getRelationships().add(newRelationship);
                    //relationshipsToRemove.add(r.getName());
                }
                Entity newEntity = new Entity(newName, domain);
                if(old.getAttributes() != null) {
                    newEntity.setAttributes(old.getAttributes());
                }
                if(old.getChildren() != null) {
                    newEntity.setChildren(old.getChildren());
                }
                if(old.getParent() != null) {
                    newEntity.setParent(old.getParent());
                }
                schema.addEntity(newEntity);

                    domainData.addEntity(newEntity);
                    schema.removeEntity(old.getName());
                    entitiesToRemove.add(selectedEnt);
                }else {
                    //System.out.println("setname");
                    domainData.getEntity(selectedEnt).setName(newName);
                }
                for(Relationship r : domainData.getRelationships()) {
                    for(Reference ref : r.getReferences()) {
                        if(ref.getSubject().equalsIgnoreCase(selectedEnt)) {
                            logs = "relation_subject (" + r.getName() + ", " + ref.getSubject() + ") modify " + newName + "\n";
                            write(logs);
                            ref.setSubject(newName);
                        }
                        if(ref.getObject().equalsIgnoreCase(selectedEnt)) {
                            logs = "relation_object (" + r.getName() + ", " + ref.getObject() + ") modify " + newName + "\n";
                            write(logs);
                            ref.setObject(newName);
                        }
                    }
                }
                logs = "entity (" + selectedEnt + ") modify " + newName + "\n";
                write(logs);
                newName = null;
                selectedEnt = null;
            }else {
                Entity entity = getEntityFromSub(selectedSubEntName);
                if(!old.getDomain().equalsIgnoreCase(domain)) {
                    old.setDomain(domain);
                    entitiesToRemove.add(selectedEnt);
                }
                logs = "entity (" + selectedSubEntName + ") child of " + entity.getParent().name + " modify " + newName + "\n";
                write(logs);
                Entity parent = entity.getParent();
                entity.setName(newName);
                renameChildren(selectedSubEntName, newName);
            }
        }

    reload();
}
```

- deleteChildNode

This method deletes a subclass.

```
public void deleteChildNode(String domain) {
    DomainData domainData = Domains.getDomain(domain);
    Entity old = domainData.getEntity(selectedEnt);
    if(!old.getDomain().equalsIgnoreCase(domain)) {     // entità importata da rimuovere
        entitiesToRemove.add(selectedEnt);
    }
    if(selectedSubEntName.equals(selectedEnt)) {
        domainData.getEntities().remove(old);
        schema.getEntities().remove(old);
        schema.removeEntity(old.getName());
        logs = "entity (" + selectedEnt + ") removed\n";
        write(logs);

        subentities = new ArrayList<>();
        fullAttributes = new ArrayList<>();
        newName = null;
    }else {
        Entity entity = getEntityFromSub(selectedSubEntName);
        old.setDomain(domain);
        logs = "entity (" + selectedSubEntName + ") child of " + entity.getParent().name + " removed\n";
        write(logs);
        removeChildren(entity);
        Entity parent = entity.getParent();
        parent.getChildren().remove(entity);

        fullAttributes.addAll(schema.getEntity(selectedEnt).getAttributes());
        newName = null;
    }
    reload();
}
```

- addAttribute

This method adds an attribute to a class.

```
public void addAttribute() {
    Attribute att = new Attribute(tempAttr.name, tempAttr.dataType, tempAttr.mandatory, tempAttr.distinguishing, tempAttr.display);
    fullAttributes.add(att);
    if(selectedSubEntName.equals("")) {
        schema.getEntity(selectedEnt).addAttribute(att);
        logs = "entity (" + selectedEnt + ") add (name = " + tempAttr.getName() + ", mandatory = " +
        Boolean.toString(tempAttr.getMandatory()) + ", distinguishing = " + Boolean.toString(tempAttr.isDistinguishing()) +
        ", display = " + Boolean.toString(tempAttr.display) +
                ", type = " + tempAttr.getDataType() + ")\n";
    }else {
        Entity entity = getEntityFromSub(selectedSubEntName);
        entity.addAttribute(att);
        logs = "entity (" + selectedSubEntName + ") child of " + entity.getParent().getName() + " add (name = "
        + tempAttr.getName() + ", mandatory = " + Boolean.toString(tempAttr.getMandatory()) +
                ", distinguishing = " + Boolean.toString(tempAttr.isDistinguishing()) + ", display = "
        + Boolean.toString(tempAttr.display) +
                ", type = " + tempAttr.getDataType() + ")\n";
    }
    write(logs);
    if(!schema.getEntity(selectedEnt).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getEntity(selectedEnt));
    }
    selectedAttr = null;
    reload();
}
```

- removeAndInsert

This method replaces an old class with an updated one.

```
private void removeAndInsert(Entity entity) {
    Entity newEntity = new Entity(entity.getName(), domain);
    if(entity.getAttributes() != null) {
        newEntity.setAttributes(entity.getAttributes());
    }
    if(entity.getChildren() != null) {
        newEntity.setChildren(entity.getChildren());
    }
    if(entity.getParent() != null) {
        newEntity.setParent(entity.getParent());
    }
    entitiesToRemove.add(entity.getName());
    schema.removeEntity(entity.getName());
    schema.addEntity(newEntity);
}
```

- modifyAttribute

This method modifies an attribute of a class.

```
public void modifyAttribute() {
    Entity entity = null;
    Attribute att;
    if(selectedSubEntName.equals("")) {
        att = schema.getEntity(selectedEnt).getAttribute(selectedAttr.name);
    }else {
        entity = getEntityFromSub(selectedSubEntName);
        att = entity.getAttribute(selectedAttr.name);
    }
    att.setName(tempAttr.name);
    att.setMandatory(tempAttr.getMandatory());
    att.setDistinguishing(tempAttr.isDistinguishing());
    att.setDisplay(tempAttr.isDisplay());
    att.setDataType(tempAttr.getDataType());
    if(tempAttr.getDataType().equalsIgnoreCase("entity")) {
        att.setTarget(tempAttr.getTarget());
    }else {
        att.setTarget("");
        tempAttr.setTarget(null);
    }
    if(!att.getDataType().equalsIgnoreCase("select")) {
        att.removeValues();
        tempAttr.setValues(new ArrayList<>());
        fullValues.removeAll(fullValues);
    }
    if(selectedSubEntName.equals("")) {
        logs = "entity_attribute (" + selectedEnt + ", " + selectedAttr.name + ") modify (name = " + att.getName() +
                ", mandatory = " + Boolean.toString(att.getMandatory()) + ", distinguishing = " +
                Boolean.toString(att.isDistinguishing()) + ", display = " + Boolean.toString(att.display) +
                ", type = " + att.getDataType() + ", target = " + att.getTarget() + ")\n";
    }else {
        logs = "entity_attribute (" + selectedSubEntName + ", " + selectedAttr.name + ") child of "
                + entity.getParent().name + " modify (name = " + att.getName() +
                ", mandatory = " + Boolean.toString(att.getMandatory()) + ", distinguishing = " +
                Boolean.toString(att.isDistinguishing()) + ", display = " + Boolean.toString(att.display) +
                ", type = " + att.getDataType() + ", target = " + att.getTarget() + ")\n";
    }
    write(logs);

   if(!schema.getEntity(selectedEnt).getDomain().equalsIgnoreCase(domain)) {
      removeAndInsert(schema.getEntity(selectedEnt));
   }
}
```

- removeAttribute

This method removes an attribute of a class.

```
public void removeAttribute() {
    if(selectedSubEntName.equals("")) {
        logs = "entity (" + selectedEnt + ") remove (name = " + selectedAttr.getName() + ")\n";
        schema.getEntity(selectedEnt).removeAttribute(selectedAttr.getName());
    }else {
        Entity entity = getEntityFromSub(selectedSubEntName);
        logs = "entity (" + selectedSubEntName + ") child of " + entity.getParent().getName() +
                " remove (name = " + selectedAttr.getName() + ")\n";
        entity.removeAttribute(selectedAttr.getName());
    }
    write(logs);
    fullAttributes.remove(selectedAttr);
    selectedAttr = null;
    fullValues = new ArrayList<>();
    if(!schema.getEntity(selectedEnt).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getEntity(selectedEnt));
    }
    reload();
}
```

- addValue

This method adds a value in the range of possible values of an attribute.

```
public void addValue() {
    Value newValue = new Value(newNodeName);
    if(selectedSubEntName.equals("")) {
        logs = "entity_attribute (" + selectedEnt + ", " + tempAttr.name + ") add (name = " + newNodeName + ")\n";
        schema.getEntity(selectedEnt).getAttribute(tempAttr.name).addValue(newValue);
    }else {
        Entity entity = getEntityFromSub(selectedSubEntName);
        logs = "entity_attribute (" + selectedSubEntName + ", " + tempAttr.name + ") child of "
                + entity.getParent().name +" add (name = " + newNodeName + ")\n";
        entity.getAttribute(tempAttr.name).addValue(newValue);
    }
    write(logs);
    if(!schema.getEntity(selectedEnt).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getEntity(selectedEnt));
    }
    reload();
}
```

- removeValue

This method removes a value in the possible values of an attribute.

```
public void removeValue() {
    if(selectedSubEntName.equals("")) {
        schema.getEntity(selectedEnt).getAttribute(tempAttr.name).removeValue(selectedValue);
    }else {
        Entity entity = getEntityFromSub(selectedSubEntName);
        logs = "entity_attribute (" + selectedSubEntName + ", " + tempAttr.name + ") child of "
                + entity.getParent().name + " remove (name = " + newNodeName + ")\n";
        entity.getAttribute(tempAttr.name).removeValue(selectedValue);
    }
    fullValues.remove(selectedValue);
    write(logs);
    if(!schema.getEntity(selectedEnt).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getEntity(selectedEnt));
    }
    reload();
}
```

- newRelationship

This method creates a new relationship.

```
public void newRelationship() {
    logs = "relationship (" + tempRelName + ", " + tempRelInv + ") reference (" + tempSubject + ", " + tempObject + ") created\n";
    write(logs);
    Relationship relationship = new Relationship(schema.getDomain(), tempRelName, tempRelInv);
    relationship.addReference(new Reference(tempSubject, tempObject));
    if(schema.getRelationshipsToString().contains(tempRelName)) {
        if(!schema.getRelationship(tempRelName).getDomain().equalsIgnoreCase(domain)) {
            relationshipsToRemove.add(tempRelName);
        }
        schema.removeRelationship(tempRelName);
    }
    schema.getRelationships().add(relationship);
    resetRelationship();
    reload();
}
```

- renameRelationship

This method renames a relationship.

```
public void renameRelationship() {
    if(tempRelName!=null && tempRelInv!=null) {
        logs = "relationship (" + selectedRel + ") modify " + tempRelName + "\n";
        write(logs);
        Relationship old = schema.getRelationship(selectedRel);
        if(!old.getDomain().equalsIgnoreCase(domain)) {
            Relationship newRelationship = new Relationship(domain, tempRelName, tempRelInv);
            newRelationship.setReferences(old.getReferences());
            newRelationship.setAttributes(old.getAttributes());
            schema.getRelationships().add(newRelationship);
            schema.removeRelationship(old.getName());
            relationshipsToRemove.add(selectedRel);
        }else {
            schema.getRelationship(selectedRel).set(tempRelName, tempRelInv);
        }
    }
    reload();
}
```

- removeRelationship

This method removes a relationship.

```
public void removeRelationship() {
    DomainData domainData = Domains.getDomain(domain);
    Relationship old = schema.getRelationship(selectedRel);
    logs = "relationship (" + tempRelName + ") removed\n";
    write(logs);
    if(!old.getDomain().equalsIgnoreCase(domain)) {
        relationshipsToRemove.add(selectedRel);
    }
    schema.getRelationships().remove(old);
    domainData.getRelationships().remove(old);
    schema.removeRelationship(old.getName());
    fullSubjectsObjects.removeAll(fullSubjectsObjects);
    fullAttributesRel.removeAll(fullAttributesRel);
    canDelete = schema.getRelationshipsToString(selectedEnt).size() == 0;
    selectedRel = null;
    reload();
}
```

- newReference

This method adds a new reference (subrelationship) to a relationship.

```
public void newReference() { //(ValueChangeEvent e) {
    logs = "relationship (" + selectedRel + ") add ref (" + tempSubject + ", " + tempObject + ") \n";
    write(logs);
    Reference reference = new Reference(tempSubject, tempObject);
    schema.getRelationship(selectedRel).addReference(reference);
    fullSubjectsObjects.add(reference);
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    resetReference();
    reload();
}
```

- removeReference

This method removes a reference (subrelationship) from a relationship.

```
public void removeReference() {
    logs = "relationship (" + selectedRel + ") remove ref (" + selectedRef.getSubject() + ", " + selectedRef.getObject() + ") \n";
    write(logs);
    Reference ref = schema.getRelationship(selectedRel).getReference(selectedRef.getSubject(), selectedRef.getObject());
    schema.getRelationship(selectedRel).getReferences().remove(ref);
    fullSubjectsObjects.remove(ref);
    if(schema.getRelationship(selectedRel).getReferences().size()==0) {
        removeRelationship();
    }
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    selectedRef = null;
    reload();
}
```

- addAttributeRel

This method adds a new attribute to a relationship.

```
public void addAttributeRel() {
    logs = "relationship (" + selectedRel + ") add (name = " + tempAttrRel.getName() + ", mandatory = " +
            Boolean.toString(tempAttrRel.getMandatory()) + ", distinguishing = " +
            Boolean.toString(tempAttrRel.isDistinguishing()) + ", display = " + Boolean.toString(tempAttrRel.display) +
            ", type = " + tempAttrRel.getDataType() + ")\n";
    write(logs);
    Attribute att = new Attribute(tempAttrRel.name, tempAttrRel.dataType, tempAttrRel.mandatory,
                                  tempAttrRel.distinguishing, tempAttrRel.display);
    tempAttrRel.name = "";
    schema.getRelationship(selectedRel).addAttribute(att);
    fullAttributesRel.add(att);
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    selectedAttrRel = null;
    reload();
}
```

- modifyAttributeRel

This method modifies an attribute of a relationship.

```
public void modifyAttributeRel() {
    Attribute att = schema.getRelationship(selectedRel).getAttribute(selectedAttrRel.name);
    att.setName(tempAttrRel.name);
    att.setMandatory(tempAttrRel.getMandatory());
    att.setDistinguishing(tempAttrRel.isDistinguishing());
    att.setDisplay(tempAttrRel.isDisplay());
    att.setDataType(tempAttrRel.getDataType());
    if(tempAttrRel.getDataType().equalsIgnoreCase("entity")) {
        att.setTarget(tempAttrRel.getTarget());
    }else {
        att.setTarget("");
        tempAttrRel.setTarget(null);
    }
    if(!att.getDataType().equalsIgnoreCase("select")) {
        att.removeValues();
        tempAttrRel.setValues(new ArrayList<>());
        fullValuesRel = new ArrayList<>();
    }
    logs = "relationship_attribute (" + selectedRel + ", " + selectedAttr.name + ") modify (name = " + att.getName() +
            ", mandatory = " + Boolean.toString(att.getMandatory()) +
            ", distinguishing = " + Boolean.toString(att.isDistinguishing()) + ", display = " + Boolean.toString(att.display) +
            ", type = " + att.getDataType() + ", target = " + att.getTarget() + ")\n";
    write(logs);
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
}
```

- removeAttributeRel

This method removes an attribute of a relationship.

```
public void removeAttributeRel() {
    logs = "relationship (" + selectedRel + ") remove (name = " + selectedAttrRel.getName() + ")\n";
    write(logs);
    fullAttributesRel.remove(selectedAttrRel);
    fullValuesRel.removeAll(fullValuesRel);
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    selectedAttrRel = null;
    reload();
}
```

- addValueRel

This method adds a value in the range of possible values of an attribute of a relationship.

```
public void addValueRel() {
    logs = "relationship_attribute (" + selectedRel + ", " + tempAttrRel.name + ") add (name = " + newNodeName + ")\n";
    write(logs);
    schema.getRelationship(selectedRel).getAttribute(tempAttrRel.name).addValue(new Value(newNodeName));
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    reload();
}
```

- removeValueRel

This method removes a value in the range of possible values of an attribute of a relationship.

```
public void removeValueRel() {
    logs = "relationship_attribute (" + selectedRel + ", " + tempAttrRel.name + ") remove (name = " + newNodeName + ")\n";
    write(logs);
    schema.getRelationship(selectedRel).getAttribute(tempAttrRel.name).removeValue(selectedValueRel);
    if(!schema.getRelationship(selectedRel).getDomain().equalsIgnoreCase(domain)) {
        removeAndInsert(schema.getRelationship(selectedRel));
    }
    reload();
}
```

- loadFile

This method imports a new domain using an XML file.

```
public void loadFile(File file, String domainName) throws ParserConfigurationException, SAXException, IOException {

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.parse(file);

    int entitiesPlace = 1;
    int relationshipsPlace = 3;

    if(doc.getDocumentElement().getChildNodes().getLength()==7) {
        NodeList imports = doc.getDocumentElement().getChildNodes().item(1).getChildNodes();

        for(int eni = 1; eni < imports.getLength(); eni+=2) {
            org.w3c.dom.Node en = imports.item(eni);
            if(eni != imports.getLength()-2) {

                if(en.getNodeType() == Node.ELEMENT_NODE) {
                    String imported = en.getAttributes().getNamedItem("name").getNodeValue();
                    importedFiles.add(imported);
                    if(!imported.equals("")) {
                        ExternalContext servletContext = FacesContext.getCurrentInstance().getExternalContext();
                        File webInf = new File(servletContext.getRealPath("/"), "WEB-INF");
                            loadFile(new File(webInf, imported + ".xml"), imported);
                    }
                }
            }else {
                if(en.getNodeName().equalsIgnoreCase("deleted")) {
                    NodeList deleted = en.getChildNodes();
                    for(int i=1; i<deleted.getLength(); i+=2) {
                        org.w3c.dom.Node del = deleted.item(i);
                        String type = del.getNodeName();
```

```
                if(type.equals("entity")) {
                    remove_entity(del.getAttributes().getNamedItem("name").getNodeValue());
                    removedEntities.add(del.getAttributes().getNamedItem("name").getNodeValue());
                }else {
                    remove_relationship(del.getAttributes().getNamedItem("name").getNodeValue());
                    removedRelationships.add(del.getAttributes().getNamedItem("name").getNodeValue());
                }
            }
        }else {
            String imported = en.getAttributes().getNamedItem("name").getNodeValue();
            importedFiles.add(imported);
            if(!imported.equals("")) {
                ExternalContext servletContext = FacesContext.getCurrentInstance().getExternalContext();
                File webInf = new File(servletContext.getRealPath("/"), "WEB-INF");
                    loadFile(new File(webInf, imported + ".xml"), imported);
            }
        }
    }
}
entitiesPlace += 2;
relationshipsPlace += 2;
}

NodeList entityNodes = doc.getDocumentElement().getChildNodes().item(entitiesPlace).getChildNodes();
for(int eni = 0; eni < entityNodes.getLength(); eni++) { // per ogni entita'
    org.w3c.dom.Node en = entityNodes.item(eni);
    if(en.getNodeType() == Node.ELEMENT_NODE) {
        String entityName = en.getAttributes().getNamedItem("name").getNodeValue();
        Entity entity = new Entity(entityName, domainName);
        Entity foundE = null;
        for(Entity e : entities) {
            if(e.getName().equalsIgnoreCase(entityName)) {
                foundE = e;
            }
        }
        if(foundE != null) {
            entities.remove(foundE);
        }
        entities.add(entity);
        System.out.println(entity + " " + entity.getDomain());
        NodeList entityFeatureNodes = en.getChildNodes();
        for(int efni = 0; efni < entityFeatureNodes.getLength(); efni++) {
            org.w3c.dom.Node efn = entityFeatureNodes.item(efni);
            if(efn.getNodeType() == Node.ELEMENT_NODE) {
                if(efn.getNodeName().equals("attributes")) {
                    readAttributes(efn, entity);
                }else {
                    readTaxonomy(efn.getChildNodes().item(1), entity, domainName);
                }
            }
        }
        //attrs.put(entityName, fullAttrs);
        entity.addAttribute(new Attribute("notes","text"));
    }
}
```

```
NodeList relationNodes = doc.getDocumentElement().getChildNodes().item(relationshipsPlace).getChildNodes();
for(int rni = 1; rni < relationNodes.getLength(); rni+=2) {
    org.w3c.dom.Node rn = relationNodes.item(rni);

    if(rn.getNodeType() == Node.ELEMENT_NODE) {
        String relationName = rn.getAttributes().getNamedItem("name").getNodeValue();
        String relationInverse = rn.getAttributes().getNamedItem("inverse").getNodeValue();
        Relationship relation = new Relationship(domainName, relationName, relationInverse);

        Relationship foundR = null;
        for(Relationship r : relationships) {
            if(r.getName().equalsIgnoreCase(relationName)) {
                foundR = r;
            }
        }
        if(foundR != null) {
            relationships.remove(foundR);
        }
        relationships.add(relation);
        inverseRels.put(relationName, relationInverse);
        inverseRels.put(relationInverse, relationName);
        Vector<Attribute> fullAttrs = new Vector<Attribute>();
        NodeList relationFeatureNodes = rn.getChildNodes();
        for(int rfni = 1; rfni < relationFeatureNodes.getLength(); rfni+=2) {
            org.w3c.dom.Node rfn = relationFeatureNodes.item(rfni);
            if(rfn.getNodeType() == Node.ELEMENT_NODE) {
                if(rfn.getNodeName().equals("references")) {
                    NodeList referenceNodes = rfn.getChildNodes();
                    for(int rrn = 1; rrn < referenceNodes.getLength(); rrn+=2) {
                        if(referenceNodes.item(rrn).getNodeType() == Node.ELEMENT_NODE) {
                            String subject = referenceNodes.item(rrn).getAttributes().getNamedItem("subject").getNodeValue();
                            if(!subjects.contains(subject))
                                subjects.add(subject);

                              String object = referenceNodes.item(rrn).getAttributes().getNamedItem("object").getNodeValue();
                              if(!objects.contains(object))
                                  objects.add(object);
                              Reference ref = new Reference(subject, object);
                              relation.addReference(ref);
                          }
                      }
                  } else if(rfn.getNodeName().equals("attributes")) {

                      NodeList attributeNodes = rfn.getChildNodes();
                      for(int rrn = 1; rrn < attributeNodes.getLength(); rrn+=2) {
                          if(attributeNodes.item(rrn).getNodeType() == Node.ELEMENT_NODE) {
                              String name = attributeNodes.item(rrn).getAttributes().getNamedItem("name").getNodeValue();
                              String mandatory = attributeNodes.item(rrn).getAttributes().getNamedItem("mandatory").getNodeValue();
                              String datatype = attributeNodes.item(rrn).getAttributes().getNamedItem("datatype").getNodeValue();
                              Attribute attr = new Attribute(name, datatype, mandatory);
                              relation.addAttribute(attr);
                          }
                      }
                      fullAttrs.add(new Attribute("notes","text"));
                      relation.addAttributes(fullAttrs);

                  }
              }
          }
      }
  }
}
```

There are a few features to note. Each import is done by a recursive call passing the new domain to be imported.

After reading the " deleted " node, the removal of imported entities and relationships occurs at the beginning, after reading the "deleted" node.

The reading of attributes and subclasses is done in the readAttributes and readTaxonomy methods here, respectively.

- readAttributes

This method reads the attribute of a class or relationship, it is recursive.

```
private void readAttributes(org.w3c.dom.Node parentNode, Entity entity) {
    Vector<Attribute> fullAttrs = new Vector<>();
    NodeList attributeNodes = parentNode.getChildNodes();
    for(int ani = 0; ani < attributeNodes.getLength(); ani++) {
        org.w3c.dom.Node an = attributeNodes.item(ani);
        if(an.getNodeType() == Node.ELEMENT_NODE) {
            NamedNodeMap n = an.getAttributes();
            String attribute = n.getNamedItem("name").getNodeValue();
            String datatype = n.getNamedItem("datatype").getNodeValue();
            Attribute currentAttr = new Attribute(attribute,datatype);

            setOptionalAttributes(n, currentAttr);
            if(datatype.equals("select")) {
                org.w3c.dom.Node valuesNode = an.getChildNodes().item(1);
                NodeList valueNodes = valuesNode.getChildNodes();
                for(int afni = 1; afni < valueNodes.getLength(); afni+=2) {
                    org.w3c.dom.Node afn = valueNodes.item(afni);
                    if(afn.getNodeType() == Node.ELEMENT_NODE) {
                        if(afn.getNodeName().equals("value")) {
                            Value value = new Value(afn.getAttributes().getNamedItem("name").getNodeValue());
                            currentAttr.addValue(value);
                        }
                    }
                }
            } else if(datatype.equals("entity")) {
                currentAttr.setTarget(n.getNamedItem("target").getNodeValue());
            }
            fullAttrs.add(currentAttr);
        }
    }
    entity.addAttributes(fullAttrs);
}
```

- readTaxonomy

This method reads the taxonomy of a class, it is recursive.

```
private void readTaxonomy(org.w3c.dom.Node parentNode, Entity root, String domainName) {
    NodeList valueNodes = parentNode.getChildNodes();
    for(int ani = 0; ani < valueNodes.getLength(); ani++) {
        org.w3c.dom.Node an = valueNodes.item(ani);
        if(an.getNodeType() == Node.ELEMENT_NODE) {
            NamedNodeMap n = an.getAttributes();
            String entityName = n.getNamedItem("name").getNodeValue();

            Entity currentEntity = new Entity(entityName, domainName);
            currentEntity.setParent(root);
            root.addChild(currentEntity);

            NodeList attributeFeatureNodes = an.getChildNodes();
            for(int afni = 1; afni < attributeFeatureNodes.getLength(); afni++) {
                org.w3c.dom.Node afn = attributeFeatureNodes.item(afni);
                if(afn.getNodeType() == Node.ELEMENT_NODE) {
                    if(afn.getNodeName().equals("values")) {
                        NodeList childNodes = afn.getChildNodes();
                        for (int cni=0; cni<childNodes.getLength(); cni++) {
                            org.w3c.dom.Node cn = childNodes.item(cni);
                            if(cn.getNodeName().equals("value")) {
                                entityName = cn.getAttributes().getNamedItem("name").getNodeValue();
                                //System.out.println(entityName);
                                Entity newEntity = new Entity(entityName, domainName);
                                newEntity.setParent(currentEntity);
                                currentEntity.addChild(newEntity);
                                NodeList recNodes = cn.getChildNodes();
                                if(recNodes != null) {
                                    for(int rci=1; rci<recNodes.getLength(); rci+=2) {
                                        if(recNodes.item(rci).getNodeName().equals("attributes")) {
                                            readAttributes(recNodes.item(rci), newEntity);
                                        }else {

                                            readTaxonomy(recNodes.item(rci), newEntity, domainName);
                                        }
                                    }
                                }
                            }
                        }
                    }else {
                        readAttributes(afn, currentEntity);
                    }
                }
            }
        }
    }
}
```

This method reads the taxonomy and, for each subclass, recursively reads any attributes and other subclasses. Each recursive call to readTaxonomy connects the current (parent) node to its children.

- build

This method exports the ontology in XML.

```
public void build(DomainData schema, ArrayList<String> entitiesToRemove, ArrayList<String> relationshipsToRemove)
        throws ParserConfigurationException, TransformerException, IOException, SAXException {
    entitiesList = new ArrayList<>();
    relationshipsList = new ArrayList<>();
    DomainData domainData = schema;
    for (int i=0; i<domainData.getEntities().size(); i++) {
        //imports = domainData.getImportedFiles();
        Entity entity = domainData.getEntities().get(i);
        System.out.println(entity.getName() + " " + entity.getDomain());
        if(entity.getDomain().equalsIgnoreCase(domainName)) {
            if(!entitiesList.contains(entity)) {
                entitiesList.add(entity);
            }
        }
    }
    for (int i=0; i<domainData.getRelationships().size(); i++) {
        Relationship relationship = domainData.getRelationships().get(i);
        if(relationship.getDomain().equalsIgnoreCase(domainName)) {
            if(!relationshipsList.contains(relationship)) {
                relationshipsList.add(relationship);
            }
        }
    }
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document document = db.newDocument();
    Element domain = document.createElement("domain");
    Attr domainAttr = document.createAttribute("name");
    domainAttr.setValue(domainName);
    domain.setAttributeNode(domainAttr);
    document.appendChild(domain);
    if(domainData.getImportedFiles().size() > 0) {
        Element imports = document.createElement("import");
        for(String fileName : domainData.getImportedFiles()) {
            Element file = document.createElement("file");
            Attr file_name = document.createAttribute("name");
            file_name.setValue(fileName);
            file.setAttributeNode(file_name);
            imports.appendChild(file);
        }
        domain.appendChild(imports);
        Element deleted = null;
        if(domainData.getRemovedEntities().size() + domainData.getRemovedRelationships().size() > 0) {
            deleted = document.createElement("deleted");
            for(String s : domainData.getRemovedEntities()) {
                Element entity = document.createElement("entity");
                Attr entity_name = document.createAttribute("name");
                entity_name.setValue(s);
                entity.setAttributeNode(entity_name);
                deleted.appendChild(entity);
            }
            for(String s : domainData.getRemovedRelationships()) {
                Element relationship = document.createElement("relationship");
                Attr relationship_name = document.createAttribute("name");
                relationship_name.setValue(s);
                relationship.setAttributeNode(relationship_name);
                deleted.appendChild(relationship);
            }
        }
        if(entitiesToRemove.size() > 0) {
            if(deleted == null) {
                deleted = document.createElement("deleted");
            }
            for(String e : entitiesToRemove) {
                Element entity = document.createElement("entity");
                Attr entity_name = document.createAttribute("name");
                entity_name.setValue(e);
                entity.setAttributeNode(entity_name);
                deleted.appendChild(entity);
            }
        }
        if(relationshipsToRemove.size() > 0) {
            if(deleted == null) {
                deleted = document.createElement("deleted");
            }
```

```
            for(String e : relationshipsToRemove) {
                Element relationship = document.createElement("relationship");
                Attr relationship_name = document.createAttribute("name");
                relationship_name.setValue(e);
                relationship.setAttributeNode(relationship_name);
                deleted.appendChild(relationship);
            }
        }
        if(deleted != null) {
            imports.appendChild(deleted);
        }
    }
    Element entities = document.createElement("entities");
    domain.appendChild(entities);
    Element relationships = document.createElement("relationships");
    domain.appendChild(relationships);
    for (Entity entity : entitiesList) {
        ArrayList<Attribute> attributesList = new ArrayList<>();
        Element entita = document.createElement("entity");
        entities.appendChild(entita);
        Attr entity_name = document.createAttribute("name");
        entity_name.setValue(entity.getName());
        entita.setAttributeNode(entity_name);
        attributesList = entity.getAttributes();
        if (!attributesList.isEmpty()) {
            Element attributes = document.createElement("attributes");
            entita.appendChild(attributes);
            for (Attribute a : attributesList) {
                Element attribute = document.createElement("attribute");
                attributes.appendChild(attribute);
                Attr att_name = document.createAttribute("name");
                att_name.setValue(a.getName());
                attribute.setAttributeNode(att_name);

                Attr datatype = document.createAttribute("datatype");
                datatype.setValue(a.getDataType());
                attribute.setAttributeNode(datatype);
                Attr mandatory = document.createAttribute("mandatory");

                mandatory.setValue(String.valueOf(a.getMandatory()));
                attribute.setAttributeNode(mandatory);

                if (!a.getValues().isEmpty()) {
                    Element values = document.createElement("values");
                    attribute.appendChild(values);
                    for(Value v : a.getValues()) {
                        Element value = document.createElement("value");
                        Attr value_name = document.createAttribute("name");
                        value_name.setValue(v.name);
                        value.setAttributeNode(value_name);
                        values.appendChild(value);
                    }
                }
            }
        }
        if(entity.getChildren().size() > 0) {
            Element taxonomy = document.createElement("taxonomy");
            entita.appendChild(taxonomy);
            Element values = document.createElement("values");
            taxonomy.appendChild(values);
            readValues(entity, values, document);
        }
    }
    for (Relationship rel : relationshipsList) {
        Element relationship = document.createElement("relationship");
        relationships.appendChild(relationship);
        Attr name = document.createAttribute("name");
        name.setValue(rel.getName());
        relationship.setAttributeNode(name);
        Attr inverse = document.createAttribute("inverse");
        inverse.setValue(rel.getInverse());
        relationship.setAttributeNode(inverse);
        Element references = document.createElement("references");
        relationship.appendChild(references);
        for (Reference ref :rel.getReferences()) {
            Element reference = document.createElement("reference");
```

```
                references.appendChild(reference);
                Attr object = document.createAttribute("object");
                Attr subject = document.createAttribute("subject");
                subject.setValue(ref.getSubject());
                object.setValue(ref.getObject());
                reference.setAttributeNode(subject);
                reference.setAttributeNode(object);
                references.appendChild(reference);
            }
            if (!rel.getAttributes().isEmpty()) {
                Element attributes = document.createElement("attributes");
                relationship.appendChild(attributes);
                for (Attribute attr : rel.getAttributes()) {
                    Element attribute = document.createElement("attribute");
                    attributes.appendChild(attribute);
                    Attr att_name_rel = document.createAttribute("name");
                    Attr att_mandatory_rel = document.createAttribute("mandatory");
                    Attr att_datatype_rel = document.createAttribute("datatype");
                    att_name_rel.setValue(attr.getName());
                    att_mandatory_rel.setValue(Boolean.toString(attr.getMandatory()));
                    att_datatype_rel.setValue(attr.getDataType());
                    attribute.setAttributeNode(att_name_rel);
                    attribute.setAttributeNode(att_mandatory_rel);
                    attribute.setAttributeNode(att_datatype_rel);
                    attributes.appendChild(attribute);
                    if (!attr.getValues().isEmpty()) {
                        Element values = document.createElement("values");
                        attribute.appendChild(values);
                        for(Value v : attr.getValues()) {
                            Element value = document.createElement("value");
                            Attr value_name = document.createAttribute("name");
                            value_name.setValue(v.name);
                            value.setAttributeNode(value_name);
                            values.appendChild(value);
                        }
                    }
                }
            }
        }

        TransformerFactory tf = TransformerFactory.newInstance();
        Transformer transformer = tf.newTransformer();
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        DOMSource source = new DOMSource(document);

        FacesContext fc = FacesContext.getCurrentInstance();
        ExternalContext ec = fc.getExternalContext();

        ec.responseReset();
        ec.setResponseContentType("text/xml");
        ec.setResponseHeader("Content-Disposition", "attachment; filename=\"" + domainData.getDomain() + ".xml" + "\"");

        OutputStream output = ec.getResponseOutputStream();
        fc.responseComplete();
        transformer.transform(source, new StreamResult(output));

    }
```

The method calls readValues which is a recursive method for reading subclasses of a class.

- readValues

This method reads the taxonomy for the export in XML.

```
private void readValues(Entity rootEntity, Element root, Document document) {
    if(rootEntity.getChildren().size() > 0) {
        for (Entity e : rootEntity.getChildren()) {
            //System.out.print(e.getName() + " ");
            Element entityElement = document.createElement("value");
            Attr entity_name = document.createAttribute("name");
            entity_name.setNodeValue(e.getName());
            entityElement.setAttributeNode(entity_name);
            root.appendChild(entityElement);
            ArrayList<Attribute> newAttributes = e.getNewAttributes();
            if(newAttributes.size()>0) {
                Element attributesElement = document.createElement("attributes");
                entityElement.appendChild(attributesElement);
                for(Attribute a : newAttributes) {
                    Element attributeElement = document.createElement("attribute");
                    Attr att_name = document.createAttribute("name");
                    Attr att_mandatory = document.createAttribute("mandatory");
                    Attr att_datatype = document.createAttribute("datatype");
                    att_name.setValue(a.getName());
                    att_mandatory.setValue(Boolean.toString(a.getMandatory()));
                    att_datatype.setValue(a.getDataType());
                    attributeElement.setAttributeNode(att_name);
                    attributeElement.setAttributeNode(att_mandatory);
                    attributeElement.setAttributeNode(att_datatype);
                    attributesElement.appendChild(attributeElement);
                    if(a.getValues().size() > 0) {
                        Element valuesElement = document.createElement("values");
                        attributeElement.appendChild(valuesElement);
                        for(Value v : a.getValues()) {
                            Element valueElement = document.createElement("value");
                            Attr value_attr_name = document.createAttribute("name");
                            value_attr_name.setValue(v.name);
                            valueElement.setAttributeNode(value_attr_name);
                            valuesElement.appendChild(valueElement);
                        }
                    }
                }
            }

            if(e.getChildren().size() > 0) {
                Element valuesElement = document.createElement("values");
                entityElement.appendChild(valuesElement);
                readValues(e, valuesElement, document);
            }
        }
    }
}
```

- build

This method exports the ontology in the OWL format.

```
public void build() throws OWLOntologyStorageException, OWLOntologyCreationException, IOException {
    IRI IOR = IRI.create("http://owl.api.ontology");
    OWLOntologyManager man = OWLManager.createOWLOntologyManager();
    OWLOntology o = man.createOntology(IOR);
    OWLDataFactory df = o.getOWLOntologyManager().getOWLDataFactory();
    OWLDataProperty attributo = df.getOWLDataProperty(IOR + "#attribute");
    OWLDataProperty nome = df.getOWLDataProperty(IOR + "#name");
    OWLDataProperty distinguish = df.getOWLDataProperty(IOR + "#distinguishing");
    OWLDataProperty mandatory = df.getOWLDataProperty(IOR + "#mandatory");
    OWLDataProperty display = df.getOWLDataProperty(IOR + "#display");
    OWLDataProperty value = df.getOWLDataProperty(IOR + "#value");
    OWLSubPropertyAxiom<?> s_nome = df.getOWLSubDataPropertyOfAxiom(nome, attributo);
    OWLSubPropertyAxiom<?> s_dist = df.getOWLSubDataPropertyOfAxiom(distinguish, attributo);
    OWLSubPropertyAxiom<?> s_mand = df.getOWLSubDataPropertyOfAxiom(mandatory, attributo);
    OWLSubPropertyAxiom<?> s_disp = df.getOWLSubDataPropertyOfAxiom(display, attributo);
    o.add(s_nome);
    o.add(s_dist);
    o.add(s_mand);
    o.add(s_disp);
    for (Entity entity : entities) {
        OWLClass classe = df.getOWLClass(IOR + "#" + entity.getName());
        OWLDeclarationAxiom da = df.getOWLDeclarationAxiom(classe);
        o.add(da);
        for (Attribute a : entity.getAttributes()) {
            OWLIndividual individual = df.getOWLNamedIndividual(IOR + "#" + a.getName());
            OWLDataPropertyAssertionAxiom n = df.getOWLDataPropertyAssertionAxiom(nome, individual, a.getName());
            OWLDataPropertyAssertionAxiom dist = df.getOWLDataPropertyAssertionAxiom(distinguish, individual,
                    String.valueOf(a.isDistinguishing()));
            OWLDataPropertyAssertionAxiom mand = df.getOWLDataPropertyAssertionAxiom(mandatory, individual, String.valueOf(a.isMandatory()));
            OWLDataPropertyAssertionAxiom disp = df.getOWLDataPropertyAssertionAxiom(display, individual, String.valueOf(a.isDisplay()));
            o.add(n);
            o.add(dist);
            o.add(mand);
            o.add(disp);
            for (Value v : a.getValues()) {
                OWLDataPropertyAssertionAxiom val = df.getOWLDataPropertyAssertionAxiom(value, individual, v.name);
                o.add(val);
            }

            OWLDatatype dt = df.getOWLDatatype(IOR + "#" + a.getDataType());
            OWLDataPropertyRangeAxiom ran = df.getOWLDataPropertyRangeAxiom(attributo, dt);
            OWLDataPropertyDomainAxiom dom = df.getOWLDataPropertyDomainAxiom(attributo, classe);
            o.add(dom);
            o.add(ran);
        }
        for(Entity e : entity.getChildren()) {
            recursiveEntity(e, IOR, df, o);
        }
    }
    for (Relationship relation : relationships) {
        OWLObjectProperty relazione = df.getOWLObjectProperty(IOR + "#" + relation.getName());
        OWLDeclarationAxiom da = df.getOWLDeclarationAxiom(relazione);
        o.add(da);
        for (Attribute a : relation.getAttributes()) {
            OWLIndividual individual = df.getOWLNamedIndividual(IOR + "#" + a.getName());
            OWLDataPropertyAssertionAxiom n = df.getOWLDataPropertyAssertionAxiom(nome, individual, a.getName());
            OWLDataPropertyAssertionAxiom dist = df.getOWLDataPropertyAssertionAxiom(distinguish, individual,
                    String.valueOf(a.isDistinguishing()));
            OWLDataPropertyAssertionAxiom mand = df.getOWLDataPropertyAssertionAxiom(mandatory, individual, String.valueOf(a.isMandatory()));
            OWLDataPropertyAssertionAxiom disp = df.getOWLDataPropertyAssertionAxiom(display, individual, String.valueOf(a.isDisplay()));
            OWLDatatype dt = df.getOWLDatatype(IOR + "#" + a.getDataType());
            OWLDataPropertyRangeAxiom ran = df.getOWLDataPropertyRangeAxiom(attributo, dt);
            o.add(n);
            o.add(dist);
            o.add(mand);
            o.add(disp);
        }
        OWLClassExpression ce = df.getOWLClass(IOR + "#" + relation.getName());
        OWLDataPropertyDomainAxiom dom = df.getOWLDataPropertyDomainAxiom(attributo, ce);
        for(int i=0; i<relation.getSubjects().size(); i++) {
            OWLClassExpression c = df.getOWLClass(IOR + "#" + relation.getSubjects().get(i));
            OWLObjectPropertyDomainAxiom sub = df.getOWLObjectPropertyDomainAxiom(relazione, c);
            o.add(sub);
        }
        for(int i=0; i<relation.getObjects().size(); i++) {
            OWLClassExpression c = df.getOWLClass(IOR + "#" + relation.getSubjects().get(i));

            OWLObjectPropertyDomainAxiom obj = df.getOWLObjectPropertyDomainAxiom(relazione, c);
            o.add(obj);
        }
        OWLObjectProperty inversa = df.getOWLObjectProperty(IOR + "#" + relation.getInverse());
        OWLInverseObjectPropertiesAxiom inverse = df.getOWLInverseObjectPropertiesAxiom(relazione, inversa);
        o.add(dom);
        o.add(inverse);
    }
    FacesContext fc = FacesContext.getCurrentInstance();
    ExternalContext ec = fc.getExternalContext();
    ec.responseReset();
    ec.setResponseHeader("Content-Disposition", "attachment; filename=\"" + domain.getDomain() + ".owl" + "\"");
    OutputStream output = ec.getResponseOutputStream();
    fc.responseComplete();
    man.saveOntology(o, new OWLXMLDocumentFormat(), output);
}
```

also, bring back the recursiveEntity method to read subclasses as well.

- recursiveEntity

This method retrieves the taxonomy of the entities for the export in OWL.

```
private void recursiveEntity(Entity child, IRI IOR, OWLDataFactory df, OWLOntology o) {
    OWLDataProperty nome = df.getOWLDataProperty(IOR + "#name");
    OWLDataProperty distinguish = df.getOWLDataProperty(IOR + "#distinguishing");
    OWLDataProperty mandatory = df.getOWLDataProperty(IOR + "#mandatory");
    OWLDataProperty display = df.getOWLDataProperty(IOR + "#display");
    OWLDataProperty value = df.getOWLDataProperty(IOR + "#value");
    OWLDataProperty attributo = df.getOWLDataProperty(IOR + "#attribute");
    OWLClass classe = df.getOWLClass(IOR + "#" + child.getName());
    OWLDeclarationAxiom da = df.getOWLDeclarationAxiom(classe);
    o.add(da);
    for (Attribute a : child.getAttributes()) {
        OWLIndividual individual = df.getOWLNamedIndividual(IOR + "#" + a.getName());
        OWLDataPropertyAssertionAxiom n = df.getOWLDataPropertyAssertionAxiom(nome, individual, a.getName());
        OWLDataPropertyAssertionAxiom dist = df.getOWLDataPropertyAssertionAxiom(distinguish, individual,
                String.valueOf(a.isDistinguishing()));
        OWLDataPropertyAssertionAxiom mand = df.getOWLDataPropertyAssertionAxiom(mandatory, individual, String.valueOf(a.isMandatory()));
        OWLDataPropertyAssertionAxiom disp = df.getOWLDataPropertyAssertionAxiom(display, individual, String.valueOf(a.isDisplay()));
        o.add(n);
        o.add(dist);
        o.add(mand);
        o.add(disp);
        for (Value v : a.getValues()) {
            OWLDataPropertyAssertionAxiom val = df.getOWLDataPropertyAssertionAxiom(value, individual, v.name);
            o.add(val);
        }
        OWLDatatype dt = df.getOWLDatatype(IOR + "#" + a.getDataType());
        OWLDataPropertyRangeAxiom ran = df.getOWLDataPropertyRangeAxiom(attributo, dt);
        OWLDataPropertyDomainAxiom dom = df.getOWLDataPropertyDomainAxiom(attributo, classe);
        o.add(dom);
        o.add(ran);
    }
    for(Entity e : child.getChildren()) {
        recursiveEntity(e, IOR, df, o);
    }
}
```

- createFile

This method exports the ontology in Prolog.

```
public void createFile() throws IOException {
    String content = "domain(" + domain.getDomain().toLowerCase() + ").\n";
    content += createEntities();
    content += createRelationships();
    FacesContext fc = FacesContext.getCurrentInstance();
    ExternalContext ec = fc.getExternalContext();
    ec.responseReset();
    ec.setResponseHeader("Content-Disposition", "attachment; filename=\"" + domain.getDomain() + ".pl" + "\"");
    OutputStream output = ec.getResponseOutputStream();
    output.write(content.getBytes());
    fc.responseComplete();
    output.close();
}
```

- createEntities

This method creates the entity for the export in Prolog.

```
private String createEntities() {
    String values = "";
    String att = "";
    for (Entity entity : entities) {
        values += "entity(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ").\n";
        for (Attribute attr : entity.getAttributes()) {
            att += "attribute(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ", " + attr.getDataType().toLowerCase() + ").\n";
            if(!attr.getValues().isEmpty()) {
                att += "values(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ", " + attr.getValuesToStringToLower() + ").\n";
            }
            if (attr.getMandatory() == true) {
                att += "mandatory(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.isDisplay() == true) {
                att += "display(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.isDistinguishing() == true) {
                att += "distinguishing(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.getTarget() != null) {
                att += "target(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ", " + attr.getTarget().toLowerCase() +").\n";
            }
        }
        if(entity.getChildren().size() > 0) {
            for(Entity e : entity.getChildren()) {
                att += "parent(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                        + e.getName().toLowerCase() + ").\n";
                att += writeEntity(e);
            }
        }
    }
    values += att;
    return values;
}
```

- writeEntity

This method exports the entities in Prolog.

```
private String writeEntity(Entity entity) {
    String att = "";
    att += "entity(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ").\n";
    for (Attribute attr : entity.getNewAttributes()) {
        att += "attribute(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                + attr.getName().toLowerCase() + ", " + attr.getDataType().toLowerCase() + ").\n";
        if(!attr.getValues().isEmpty()) {
            att += "values(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ", " + attr.getValuesToStringToLower() + ").\n";
        }
        if (attr.getMandatory() == true) {
            att += "mandatory(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ").\n";
        }
        if (attr.isDisplay() == true) {
            att += "display(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ").\n";
        }
        if (attr.isDistinguishing() == true) {
            att += "distinguishing(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ").\n";
        }
        if (attr.getTarget() != null) {
            att += "target(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ", " + attr.getTarget().toLowerCase() +").\n";
        }
    }
    if(entity.getChildren().size() > 0) {
        for(Entity e : entity.getChildren()) {
            att += "parent(" + entity.getDomain().toLowerCase() + ", " + entity.getName().toLowerCase() + ", "
                    + e.getName().toLowerCase() + ").\n";
            att += writeEntity(e);
        }
    }
    return att;
}
```

- createRelationships

This method creates the relationships for the export in Prolog.

```
private String createRelationships() {
    String att = "";
    for (Relationship relation : relationships) {
        for(Reference ref : relation.getReferences()) {
            att += "relationship(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                    + ref.getSubject().toLowerCase() + ", " + ref.getObject().toLowerCase() + ").\n";
        }
        att += "inverse(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                + relation.getInverse().toLowerCase() + ").\n";
        for (Attribute attr : relation.getAttributes()) {
            att += "attribute(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                    + attr.getName().toLowerCase() + ", " + attr.getDataType().toLowerCase() + ").\n";
            if(!attr.getValues().isEmpty()) {
                att += "values(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ", " + attr.getValuesToStringToLower() + ").\n";
            }
            if (attr.getMandatory() == true) {
                att += "mandatory(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.isDisplay() == true) {
                att += "display(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.isDistinguishing() == true) {
                att += "distinguishing(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase()
                            + ", " + attr.getName().toLowerCase() + ").\n";
            }
            if (attr.getTarget() != null) {
                att += "target(" + relation.getDomain().toLowerCase() + ", " + relation.getName().toLowerCase() + ", "
                        + attr.getName().toLowerCase() + ", " + attr.getTarget().toLowerCase() +").\n";
            }
        }
    }
    return att;
}
```

IV.II Importing Ontologies in Neo4j

We report just the three pieces of codes for the three main operations. As you can see, we execute three Cypher queries.

- add_node

```
public void add_node(String iri, String type) {
    try ( Session session = driver.session() ) {
        session.writeTransaction( new TransactionWork<String>() {
            @Override
            public String execute( Transaction tx ) {
                Result result = tx.run("MERGE (n:" + type + "{ iri: \"" + iri + "\" }) RETURN n.iri");
                return result.single().get( 0 ).asString();
            }
        } );
        System.out.println("node " + iri + " added");
    }
}
```

- add_attribute

```
public void add_attribute(String iri, String type, String attributeName, String attributeValue) {
    try ( Session session = driver.session() )
    {
        session.writeTransaction( new TransactionWork<String>()
        {
            @Override
            public String execute( Transaction tx )
            {
                Result result = tx.run( "MERGE (n:" + type + " {iri: \"" + iri + "\" }) " +
                                        "SET n." + attributeName + " = \"" + attributeValue + "\" " +
                                        "RETURN n."+attributeName);
                return result.single().get( 0 ).asString();
            }
        } );
        System.out.println("attribute " + attributeName + " added on node " + iri);
    }
}
```

- add_relationship

```
public void add_relationship(String iriLeft, String typeLeft, String iriRight, String typeRight, String iriRel, String typeRel) {
    try ( Session session = driver.session() ) {
        session.writeTransaction( new TransactionWork<String>() {
            @Override
            public String execute( Transaction tx ) {
                String str = "MERGE (l:" + typeLeft +" { iri: \"" + iriLeft +"\" })" +
                        " MERGE (r: " + typeRight + "{ iri: \"" + iriRight +"\" })" +
                        " MERGE (l)-[rel: " + typeRel + " { iri: \" " + iriRel + "\" }]->(r)" +
                        " RETURN rel.iri";
                Result result = tx.run( "MERGE (l:" + typeLeft +" { iri: \"" + iriLeft +"\" })" +
                                        " MERGE (r: " + typeRight + "{ iri: \"" + iriRight +"\" })" +
                                        " MERGE (l)-[rel: " + typeRel + " { iri: \" " + iriRel + "\" }]->(r)" +
                                        " RETURN rel.iri");
                return result.single().get( 0 ).asString();
            }
        } );
        System.out.println("relationship " + iriLeft + " added");
    }
}
```

IV.III Translating RDF into XML

We report, for example, the code for importing the ontology and iterating over the classes.

```
String export = path.replace("rdf", "xml");
String domain = path.substring(path.lastIndexOf("\\")+1, path.lastIndexOf("."));
OntModel model = ModelFactory.createOntologyModel();
model.read(path, "RDFXML");

DomainData domainData = new DomainData(domain);
ExtendedIterator<OntClass> itClass = model.listClasses();

int count = 0;
Entity entity;
while(itClass.hasNext()) {
    OntClass ontclass = itClass.next();
    if(ontclass.toString().contains("#") && !ontclass.toString().contains("Thing")) {
        count++;
        entity = new Entity(domain, ontclass.toString().substring(ontclass.toString().lastIndexOf("#")+1));
        domainData.getEntities().add(entity);
    }
}
```

IV.IV Reading XML Ontologies

We report a small portion of the parser.

```
public static Manager getModelFromXML(String path, String prefix, HashMap<String, ArrayList<String>> mandatories)
        throws SAXException, IOException, ParserConfigurationException {
    Manager m = new Manager(prefix);
    File file = new File(path);
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.parse(file);
    doc.getDocumentElement().normalize();
    Element root = doc.getDocumentElement();
    Node entities = root.getFirstChild();
    Node relationships = root.getLastChild();
    NodeList nodes = root.getChildNodes();
    NodeList nodeList = doc.getElementsByTagName("entity");
    m.addClass("Year");
    m.addClass("Month");
    m.addClass("Day");
    for (int i=0; i<nodeList.getLength(); i++) {
        Node entity = nodeList.item(i);
        String newclass = entity.getAttributes().getNamedItem("name").getNodeValue();
        m.addClass(newclass);
        readAttributes(m, entity, mandatories);
        readSubClasses(m, entity, mandatories);
    }
    nodeList = doc.getElementsByTagName("relationship");
    m.addClass("relationship");
    for (int i=0; i<nodeList.getLength(); i++) {
        Node relationship = nodeList.item(i);
        String newrelationship = relationship.getAttributes().getNamedItem("name").getNodeValue();
        String inverse = relationship.getAttributes().getNamedItem("inverse").getNodeValue();
        Node references = relationship.getChildNodes().item(1);
```

```
        NodeList refsList = references.getChildNodes();
        m.addSubClass(newrelationship, "relationship");
        ArrayList<String> childrenSubjs = new ArrayList<>();
        ArrayList<String> childrenObjs = new ArrayList<>();
        for (int j=1; j<refsList.getLength(); j+=2) {
            String subject = refsList.item(j).getAttributes().getNamedItem("subject").getNodeValue();
            String object = refsList.item(j).getAttributes().getNamedItem("object").getNodeValue();
            m.addObjectProperty(newrelationship + "_" + subject + "_" + object, subject, object);
            childrenSubjs.add(subject);
            childrenObjs.add(object);
            if(!inverse.equals("")) {
                m.addObjectProperty(inverse + "_" + object + "_" + subject, object, subject);
            }
        }
        m.addParentObjectProperty(newrelationship, inverse, childrenSubjs, childrenObjs);
        if(relationship.getChildNodes().getLength() > 3) {
            readRelationshipAttributes(m, relationship, mandatories);
        }
    }

    return m;
}
```

IV.V Extract Subgraph

We report a small part of the parsing procedure that reads the json file and load all the nodes and arcs in it.

```
public static Manager readJson(Manager m, String path, int k, HashMap<String, ArrayList<String>> indTypes) {
    FileReader fileReader;
    try {
        fileReader = new FileReader(path);
        Scanner sc = new Scanner(fileReader);

        JsonParser parse = new JsonParser();
        int b = 0;
        while(sc.hasNext()){
            JsonObject jsonline = (JsonObject) parse.parse(sc.nextLine());
            for(int ind=0; ind<=k; ind++) {
                JsonObject node = (JsonObject) jsonline.get("n" + ind);
                m = createNode(m, node, indTypes);
            }
            for(int ind=1; ind<=k; ind++) {
                JsonObject node1 = (JsonObject) jsonline.get("n" + (ind-1));
                JsonObject node2 = (JsonObject) jsonline.get("n" + ind);
                JsonObject rel = (JsonObject) jsonline.get("r" + ind);
                m = createRelationship(m, node1, node2, rel, indTypes);
            }
        }

    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return m;
}
```

```
private static Manager createNode(Manager m, JsonObject node, HashMap<String, ArrayList<String>> indTypes) {
    HashMap<String, String> properties = new HashMap<>();
    JsonObject propertiesObj;
    try {
        propertiesObj = (JsonObject) node.get("properties");
        String propertiesString = propertiesObj.toString();
        String[] propertiesList = propertiesString.split(",");
        for(String p : propertiesList) {
            p = p.replace("{", "").replace("}", "").replace("\"", "").replace("\\", "");
            String propertyName = p.split(":")[0];
            String propertyValue = p.replace(propertyName, "").substring(1);
            properties.put(propertyName, propertyValue);
        }
        JsonArray labels;
        labels = ((JsonArray) node.get("labels"));
        if(!properties.get("uri").contains("node") && !m.containsClass(properties.get("uri"))) {
            ArrayList<String> types = new ArrayList<>();
            ArrayList<String> types2 = new ArrayList<>();
            for (int j=0; j<labels.size(); j++) {
                String label = labels.get(j).toString().replace("\"", "");
                if(!label.equalsIgnoreCase("Resource") && !label.equalsIgnoreCase("Thing")
                        && !label.equalsIgnoreCase("Class") && !label.equalsIgnoreCase("Restriction")
                        && !label.equalsIgnoreCase("ObjectProperty")) {
                    types.add(m.getIOR() + "#" + label);
                    types2.add(label);
                }
            }
            //String individual = properties.get("uri").substring(properties.get("uri").lastIndexOf("#")+1);
            String individual = node.get("id").getAsString();
            if(!m.containsIndividual("<" + m.getIOR() + "#" + individual + ">")) {
                if(types.size() > 0) {
                    m.addIndividual(individual, types2);
                    indTypes.put(m.getIOR() + "#" + individual, types);
                }
            }
            Iterator it = properties.entrySet().iterator();

           while (it.hasNext()) {
               Map.Entry pair = (Map.Entry)it.next();
               m.addDatatypePropertyAxiom(m.getIOR() + "#" + pair.getKey(), pair.getValue().toString(), individual);
               it.remove(); // avoids a ConcurrentModificationException
           }
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    //m.saveOWLOntology("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\Import RDF\\Test.owl");
    return m;
}
```

```
private static Manager createRelationship(Manager m, JsonObject node1, JsonObject node2, JsonObject rel, HashMap<String, ArrayList<String>> indTypes) {
    try {
        String uri1 = getUriFromNode(node1);
        String uri2 = getUriFromNode(node2);
        String id1 = m.getIOR() + "#" + getIdFromNode(node1);
        String id2 = m.getIOR() + "#" + getIdFromNode(node2);
        ArrayList<String> typeRel = new ArrayList<>();
        typeRel.add(rel.get("label").toString().replace("\"", ""));
        // ALTRI ATTRIBUTI
        if(!uri1.contains("node") && !uri2.contains("node")) {
            String propertyName = getPropertyName(m, id1, id2, rel.get("label").toString().replace("\"", ""), indTypes);
            if(propertyName.contains("INV.")) {
                propertyName = propertyName.replace("INV.", "");
                String temp = id1;
                id1 = id2;
                id2 = temp;
            }
            m.addIndividual(rel.get("label").toString().replace("\"", "") + "_" + rel.get("id").toString().replace("\"", ""), typeRel);
            m.addObjectPropertyAxiom(id1.substring(id1.lastIndexOf("#")+1),
                    propertyName.substring(propertyName.lastIndexOf("#")+1), id2.substring(id2.lastIndexOf("#")+1));
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return m;
}
```

IV.VI Wrapper

We report the Wrapper class.

```
public class Wrapper {
    private Manager m;
    private HashMap<String, ArrayList<String>> mandatories;
    private HashMap<String, ArrayList<String>> indTypes;

    public Wrapper(Manager m, HashMap<String, ArrayList<String>> mandatories, HashMap<String, ArrayList<String>> indTypes) {
        this.m = m;
        this.mandatories = mandatories;
        this.indTypes = indTypes;

        HashMap<String, ArrayList<String>> indRel = map(indTypes);
        while(indRel.size() > 0) {
            HashMap<String, ArrayList<String>> indTypes2 = load(indRel);
            indRel = map(indTypes2);
        }
    }

    private HashMap<String, ArrayList<String>> load(HashMap<String, ArrayList<String>> indRel) {
        HashMap<String, ArrayList<String>> oldIndTypes = new HashMap<>();
        oldIndTypes.putAll(indTypes);
        HashMap<String, ArrayList<String>> newIndTypes = new HashMap<>();
        try {
            Iterator itIndividual = indRel.keySet().iterator();
            while(itIndividual.hasNext()) {
                String individual = itIndividual.next().toString();
                Iterator itRel = indRel.get(individual).iterator();
                while(itRel.hasNext()) {
                    String rel = itRel.next().toString();
                    GraphDB.extractByNodeRel(individual, rel, "wrapper.json");
                    Extraction.readJson(m, "C:\\Users\\ddipi\\Documents\\neo4j-community-3.5.28-windows\\import\\wrapper.json", 1, indTypes);
                }
            }
            m.saveOWLOntology("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\ImportRDF\\output.xml");
            Iterator itIndTypes = indTypes.keySet().iterator();

            while(itIndTypes.hasNext()) {
                String individual = itIndTypes.next().toString();
                if(!oldIndTypes.containsKey(individual)) {
                    newIndTypes.put(individual, indTypes.get(individual));
                }
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return newIndTypes;
    }

    private HashMap<String, ArrayList<String>> map(HashMap<String, ArrayList<String>> indTypes) {
        HashMap<String, ArrayList<String>> indRel = new HashMap<>();
        Iterator itIndTypes = indTypes.keySet().iterator();
        while(itIndTypes.hasNext()) {
            String individual = itIndTypes.next().toString();
            ArrayList<String> rels = new ArrayList<>();
            Iterator itClasses = indTypes.get(individual).iterator();
            while(itClasses.hasNext()) {
                String c = itClasses.next().toString();
                c = c.substring(c.lastIndexOf("#")+1);
                if(mandatories.containsKey(c)) {
                    rels.addAll(mandatories.get(c));
                }
            }
            if(!rels.isEmpty()) {
                indRel.put(individual, rels);
            }
        }
        return indRel;
    }
}
```

As you can see, there is an iteration in the constructor which reads whether a new import is necessary. The indRel variable stores the map between individuals and relationships to retrieve. When relationships starting from an individual have been imported, the individual is removed from this map. New individuals can be added or not, depending on their class. The convergence is guaranteed.

IV.VII Reasoning

We report the implementation for exploiting the reasoner capabilities and obtaining the results.

```
public static void extractIndividuals(Manager m, String prefix, String id, int k, String outputFile,
        HashMap<String, ArrayList<String>> mandatories, HashMap<String, ArrayList<String>> indTypes) throws IOException {

    // REASONER ON ONTOLOGY
    FileWriter reasonerFile = new FileWriter(outputFile);
    ReasonerFactory factory = new ReasonerFactory();
    Configuration configuration = new Configuration();
    configuration.throwInconsistentOntologyException = false;

    OWLReasoner owlreasoner = factory.createReasoner(m.getOntology(), configuration);
    reasonerFile.write("Is the ontology consistent? " + owlreasoner.isConsistent() + "\n");

    // EXTRACTION
    m = subgraph(m, id, k, indTypes);

    m.saveOWLOntology("C:\\Users\\ddipi\\Desktop\\Davide\\Universita\\Tesi\\ImportRDF\\output.xml");    // COPY
    Wrapper wrapper = new Wrapper(m, mandatories, indTypes);

    owlreasoner = factory.createReasoner(m.getOntology(), configuration);
    reasonerFile.write("Are the ontology and the individuals consistent? " + owlreasoner.isConsistent() + "\n");
    reasonerFile.write("Subclasses of Site : ");
    owlreasoner.precomputeInferences(InferenceType.CLASS_HIERARCHY);
    NodeSet<OWLClass> set = owlreasoner.getSubClasses(m.getDf().getOWLClass(prefix + "#Site"), false);
    for(org.semanticweb.owlapi.reasoner.Node<OWLClass> subclass : set.getNodes()) {
        reasonerFile.write(subclass.toString() + "\n");
    }

    Reasoner reasoner = new Reasoner(configuration, m.getOntology());
    BlackBoxExplanation explain = new BlackBoxExplanation(m.getOntology(), factory, reasoner);
    HSTExplanationGenerator multiEx = new HSTExplanationGenerator(explain);
    InferredSubClassAxiomGenerator gen = new InferredSubClassAxiomGenerator();
    Set<OWLSubClassOfAxiom> subClass = gen.createAxioms(m.getDf(), reasoner);
    SatisfiabilityConverter converter = new SatisfiabilityConverter(m.getDf());

      for (OWLSubClassOfAxiom ax : subClass) {
          reasonerFile.write("\nAxiom :- " + ax + "\n");
          reasonerFile.write("Is axiom entailed by reasoner ? :- " + reasoner.isEntailed(ax) + "\n");
          reasonerFile.write("Is axiom contained in ontology ? :- " + m.getOntology().containsAxiom(ax) + "\n");
          Set<Set<OWLAxiom>> expl = multiEx.getExplanations(converter.convert(ax));
          reasonerFile.write("No. of Explanations :- " + expl.size() + "\n");
          reasonerFile.write("Explanation :- ");
          for (Set<OWLAxiom> a : expl) {
              reasonerFile.write(a.toString());
          }
          reasonerFile.write("\n");
      }
      reasonerFile.close();
  }
```

IV.VIII Integration in Gr@phBRAIN

For integration, there was no need to modify the source of the function. Therefore, they will not be carried over again. Please, refer to the previous paragraphs.

What is needed instead, is to introduce the integration section in the schema.xhtml file. This can be done by inserting graphical elements such as buttons, text fields and file uploaders already present in the different screens shown above.

For all the operations described above, the import of the following Java libraries is needed:

- automaton;
- collection;
- commons-cli;
- commons-codec;
- commons-compress;
- commons-csv;
- commons-io;
- commons-lang;
- gson;
- http-client;
- http-cache;
- httpcore;
- jackson-annotations;
- jackson-core;
- jackson-databind;
- javax.annotation-api;
- jcl-over-slf4j;
- jena-arq;
- jena-base;
- jena-cmds;
- jena-core;
- jena-dboe-base;
- jena-dboe-index;
- jena-dboe-storage;
- jena-dboe-transaction;
- jena-dboe-data;
- jene-iri;

- jena-rdfconnection;
- jena-shacl;
- jena-shaded-guava;
- jena-tdb2;
- jena-tdb;
- jsonld-java;
- libthrift;
- log4j;
- log4j-core;
- log4j-slf4j-impl;
- org.semantic.web.hermit;
- slf4j-api.

V. References

[1] B. Matthews, D. Brickley, and L. Dodds, "Semantic Web Technologies World Wide Web Consortium (W3C)," *JISC Technol. Stand. Watch*, no. January 2005, 2014.

[2] T. Berners-Lee, "Information Management: A Proposal. Internal Project Proposal," *Cern*, no. May, p. 20, 1989.

[3] T. B. Lee, J. Hendler, and O. Lassila, "The Semantic Web," vol. 284, no. 5, pp. 34–43, 2001.

[4] T. Berners-Lee, "Semantic web road map," no. September, 1998.

[5] "Semantic Web Activity." https://www.w3.org/2001/sw/.

[6] N. Zaki, C. Tennakoon, and H. Al Ashwal, "Knowledge graph construction and search for biological databases," *Int. Conf. Res. Innov. Inf. Syst. ICRIIS*, pp. 0–5, 2017, doi: 10.1109/ICRIIS.2017.8002465.

[7] P. Cimiano, "Knowledge Graph Refinement: A Survey of Approaches and

Evaluation Methods," *Semant. Web*, vol. 8, no. 3, pp. 489–508, 2017, [Online]. Available: http://www.semantic-web-journal.net/content/knowledge-graph-refinement-survey-approaches-and-evaluation-methods.

[8] C. Feilmayr and W. Wöß, "An analysis of ontologies and their success factors for application to business," *Data Knowl. Eng.*, vol. 101, pp. 1–23, 2016, doi: 10.1016/j.datak.2015.11.003.

[9] J. Euzenat and P. Shvaiko, *Ontology matching, 2nd Edition*. 2013.

[10] T. N. K. Raju *et al.*, "Towards a Definition of Knowledge Graphs," *Lancet*, vol. 329, no. 8534, pp. 651–656, 1987, doi: 10.1016/S0140-6736(87)90414-4.

[11] X. Chen, S. Jia, and Y. Xiang, "A review: Knowledge reasoning over knowledge graph," *Expert Syst. Appl.*, vol. 141, 2020, doi: 10.1016/j.eswa.2019.112948.

[12] "Neo4j documentation." https://neo4j.com/docs/.

[13] "W3C RDF Validator." https://www.w3.org/RDF/Validator/.

[14] "RDFLib." https://rdflib.readthedocs.io/en/stable/.

[15] "RDFLib code." https://github.com/RDFLib/rdflib.

[16] "Apache Jena." https://jena.apache.org/.

[17] "Apache Jena documentation." https://jena.apache.org/documentation/.

[18] "Apache Jena jar." https://jena.apache.org/download/.

[19] D. Yoo, G. Kim, and Y. Suh, "Hotel-Domain Ontology for a Semantic Hotel Search System."

[20] "Pizza Ontology." https://protege.stanford.edu/ontologies/pizza/pizza.owl.

[21] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semant.*, vol. 5, no. 2, pp. 51–53, 2007, doi: 10.1016/j.websem.2007.03.004.

[22] "Bossam." https://bossam.wordpress.com/about-bossam/.

[23] "Reasoners and rule engine: Jena inference support."

https://jena.apache.org/documentation/inference/.

[24] V. S. Silva, A. Freitas, and S. Handschuh, "Building a knowledge graph from natural language definitions for interpretable text entailment recognition," *Lr. 2018 - 11th Int. Conf. Lang. Resour. Eval.*, pp. 3438–3442, 2019.

[25] R. Poli, M. Healy, and A. Kameas, *Theory and Applications of Ontology: Computer Applications*. .

[26] L. Penev *et al.*, "OpenBiodiv: A knowledge graph for literature-extracted linked open data in biodiversity science," *Publications*, vol. 7, no. 2, pp. 1–16, 2019, doi: 10.3390/publications7020038.

[27] "R Library for Working in RDF." https://github.com/pensoft/rdf4r.

[28] S. Purohit, N. Van, and G. Chin, "Semantic property graph for scalable knowledge graph analytics," *arXiv*, 2020.

[29] "arCo." http://wit.istc.cnr.it/arco/?lang=en.

[30] "arCo ontology." http://wit.istc.cnr.it/arco/lode/extract?lang=en&url=https://raw.githubusercontent.com/ICCD-MiBACT/ArCo/master/ArCo-release/ontologie/arco/arco.owl.

[31] S. K. Biswas, M. Bordoloi, and J. Shreya, "A graph based keyword extraction model using collective node weight," *Expert Syst. Appl.*, vol. 97, pp. 51–59, 2018, doi: 10.1016/j.eswa.2017.12.025.

[32] K. Henderson *et al.*, "RolX: Structural role extraction & mining in large graphs," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 1231–1239, 2012, doi: 10.1145/2339530.2339723.

[33] J. Zhan, S. Gurung, and S. P. K. Parsa, "Identification of top-K nodes in large networks using Katz centrality," *J. Big Data*, vol. 4, no. 1, 2017, doi: 10.1186/s40537-017-0076-5.

[34] L. Lv, K. Zhang, T. Zhang, D. Bardou, J. Zhang, and Y. Cai, "PageRank centrality for temporal networks," *Phys. Lett. Sect. A Gen. At. Solid State*

Phys., vol. 383, no. 12, pp. 1215–1222, 2019, doi: 10.1016/j.physleta.2019.01.041.

VI. Thanksgivings

I would like to thank Professor Ferilli and Redavid for their contribution to this work, and I would also like to thank my supervisor for encouraging me to pursue a career in this area. Hoping it could be just the beginning of more subsequent work to come.