International Journal of Engineering, Science and Mathematics
Vol7. Issue 3, Month 2018,
ISSN: 2320-0294 Impact Factor: 6.765
Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

A BRIEF SURVEY OF CREATING SEMANTIC WEB CONTENT WITH PROTÉGÉ

M. El Asikri [*]
S. Krit [*]
H.Chaib [*]

Abstract

An ontology development tool is often the first thing that people get to see when they venture into the Semantic Web field. Ontology editors and visualization tools therefore carry a special responsibility for the success of the Semantic Web community. At the same time, the user communities around such tools serve as melting pots which can be exploited to collect feedback on the overall design of the language and associated systems.

Protégé is one of the most used development platforms for ontology-based systems. This paper report on the experiences of using Protégé with OWL. The intention of the Protégé and specially Protégé Plugin is to make Semantic Web technology available to a broad group of developers and users, and to promote best practices and design patterns. In this document we walk through a selection of these issues and suggest directions for future work and standardization efforts.

Keywords:

Semantic Web;
Ontology;
Protégé;
OWL;

Author correspondence:

Mohamed El Asikri ,
Department Mathematics, Informatics and Management,
Laboratory of Engineering Sciences and Energy.
Polydisciplinary Faculty of Ouarzazate, Ibn Zohr University
Agadir BP/638 Morocco

1. Introduction

Ontology is a conceptualization of a domain into machine readable format [1]. Ontologies are becoming increasingly popular modelling schemas for knowledge management services and applications. Focus on developing tools to graphically visualise ontologies is rising to aid their assessment and analysis. Graph visualisation helps to browse and comprehend the structure of ontologies. Protégé [2] is one of the most widely used ontology development tools that were developed at Stanford University. Protégé provides an intuitive editor for ontologies and has extensions for ontology visualization, project management, software engineering and other modelling tasks. An ontology, according to the definition in [3] is a formal explicit description of a domain, consisting of classes, which are the concepts found in the domain. Classes are organized in a specialization/generalization hierarchy through is-a (or inheritance) links, where each class is allowed to have zero, one or multiple parent classes. Each class has properties (or slots) describing various features of the modelled class. Slots are typed, and allowed types are either simple types (strings, numbers, booleans or enumerations) or instances of other classes (references); restriction on the value ranges of slots (e.g. integers from 1 to 10) may also be defined. Finally, instantiation may be applied to classes to produce items corresponding to individual objects in the domain of discourse (instances). Each instance has a concrete value for each property of the class it belongs to. Classes, together with instances are said to constitute the knowledge base. From the definition above, it is evident that the task of visualizing the full set of ontology features is not an easy one. The properties of ontology are summarized as follows:

- ***Hierarchy.*** A type of organization that, like a tree, branches into more specific units, each of which is "owned" by the higher-level unit immediately above.
- ***Properties representation.*** More than a hierarchy, as it concepts are described by using restrictions on properties.
- ***Level of detail.*** Possibility to choose till which level an ontology to be provided.
- ***History.*** The concepts that were chosen in the previous steps.
- ***Filtering.*** Ontologies could contain hundreds of properties. The user can be interested in only the subset of the ontology, based on the central concept and the properties of the user's choice.
- ***Multiple geometrical views.*** The representation of the graph in different geometrical models to better understand the structure of ontology.
- ***Zoom semantic/geometric.*** To see more or less details during ontology exploration. With the geometric zoom the visualized object is scaled when the user zooms in/out. The semantic zoom provides the possibility to see more/less details of the object by zooming in/out.

This paper introduces Protégé for creating OWL ontologies and gives a brief overview of the OWL ontology language. With the focus on building an OWL ontology and using a Description Logic Reasoner to check the consistency of the ontology and automatically compute the ontology class hierarchy and describes some plugin of Protégé which aren't directly used in the main paper.

2. Method for building ontology and OWL language

2.1 Building ontology process

We will presents, in direct chronological order, the most well known approaches for building ontologies [4] from scratch, as well as reusing ontologies that are stored in ontology libraries. First the main set of criteria used to compare different approaches of this type is presented. Then, a brief description of each approach is provided, presenting who has elaborated it and the proposed steps and activities, there is no one correct methodology for developing ontologies. Developing ontology is usually an iterative process. We can start with a rough first pass at the ontology and then revise and refine the evolving ontology. Ontology is a model of a real domain in the world and the concepts in the ontology must reflect this reality. After defining an initial version of the ontology, we can evaluate and debug it by using it in applications or problem-solving methods or by discussing it with experts in the field. As a result, we will almost certainly need to revise the initial ontology.

This process of iterative design will likely continue through the entire lifecycle of the ontology.
Developing an Ontology may include:

1. Selection of Domain and Scope
2. Consider Reuse
3. Find out Important Terms

4. Defining Classes and Class Hierarchy
5. Defining Properties of Classes and Constraints
6. Create Instances of classe

2.2 OWL Ontologies :

Ontologies are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts. Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (W3C).

OWL makes it possible to describe concepts but it also provides new facilities. It has a richer set of operators - e.g. intersection, union and negation. It is based on a different logical model which makes it possible for concepts to be defined as well as described. Complex concepts can therefore be built up in definitions out of simpler concepts. Furthermore, the logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are mutually consistent and can also recognise which concepts fit under which definitions. The reasoner can therefore help to maintain the hierarchy correctly. This is particularly useful when dealing with cases where classes can have more than one parent.

An OWL ontology consists of Individuals, Properties, and Classes, which roughly correspond to Protégé frames Instances, Slots and Classes.

2.3 OWL Language:

The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema. The OWL Use Cases and Requirements Document provides more details on ontologies, motivates the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web:

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

- OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users :

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL, see the section on OWL Lite in the OWL Reference for further details.

OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Owl example:

```
<owl:Class
rdf:about="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#PlanteS
avoureuse">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#Plant
e"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty
rdf:resource="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#mangepar"/>
            <owl:someValuesFrom
rdf:resource="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#Carnivore"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty
rdf:resource="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#mangepar"/>
            <owl:someValuesFrom
rdf:resource="http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#Herbivore"/>
        </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

3 **protégé**

Protégé [5] is a very popular knowledge-modelling tool developed at Stanford University. Ontologies and knowledge-bases can be edited interactively within Protégé and accessed with a graphical user interface and Java API figure 1**.** Protégé can be extended with pluggable components to add new functionalities and services.
There exists an increasing number of plugins offering a variety of additional features, such as extra ontology management tools, multimedia support, querying and reasoning engines, problem solving

methods, etc. Protégé implements a rich set of knowledge-modelling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé gives support for building the ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). The extended version of frame based system was introduced in 2003 to support OWL with an advantage of semantic web version. There are various forms such as RDF(s), OWL and XML Schema in which protégé ontology can be exported.

The OWL Plugin is a complex protégé extension that can be used to edit OWL files and databases. The OWL Plugin includes a collection of custom-tailored tabs and widgets for OWL, and provides access to OWL-related services such as classification, consistency checking, and ontology testing.

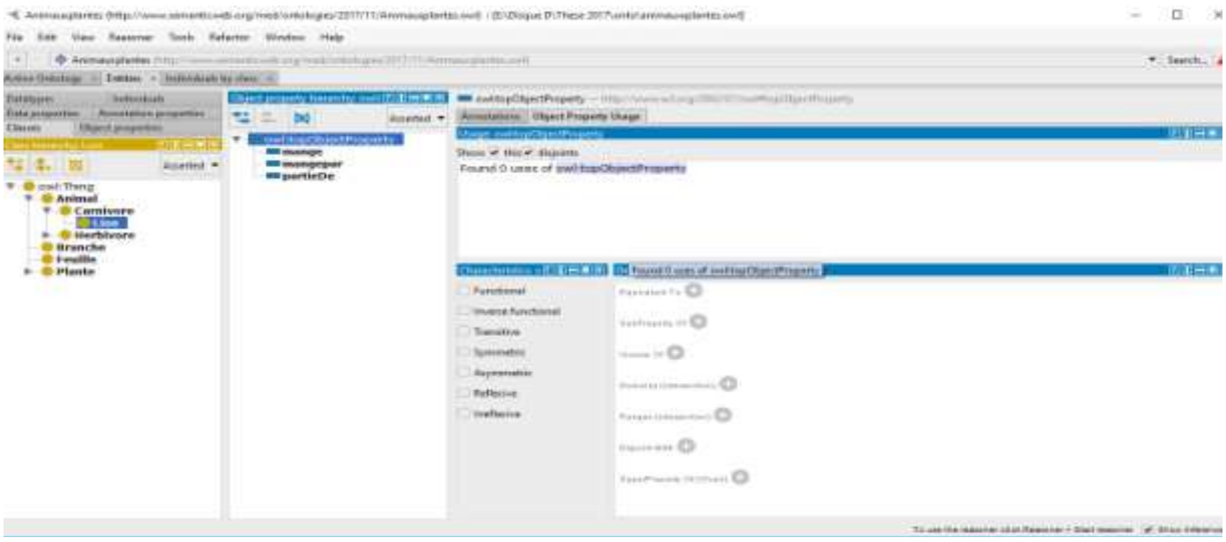


Figure 1 : Protégé interface

3.1 OWL Plugin Metamodel

The OWL Plugin extends the Protégé model and its API with classes to represent the OWL specification. The OWL Plugin supports RDF(S), OWL Lite, OWL DL (except for anonymous global class axioms, which need to be given a name by the user) and significant parts of OWL Full (including metaclasses). In order to better understand this extension mechanism, we need to look at the differences between the Protégé metamodel and OWL. OWL is an extension of RDF(S) [7].

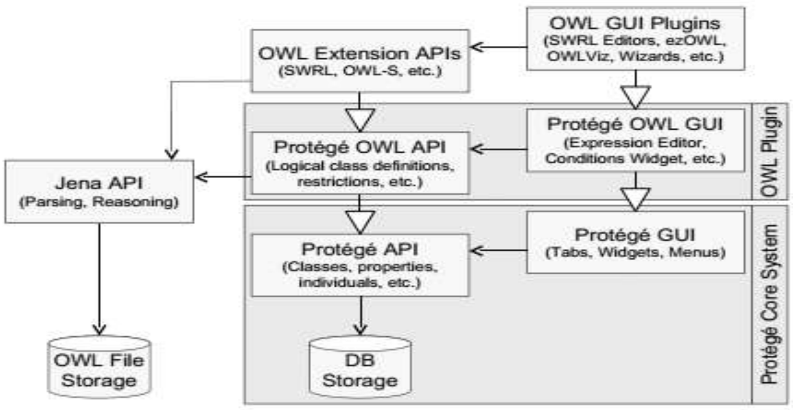


Figure 2 OWL plugins architecture

RDF has a very simple triple-based model that is often too verbose to be edited directly in a tool. Fortunately, RDF Schema extends RDF with metamodel classes and properties which can be mapped into

the Protégé metamodel. As a result, the extensions that OWL adds to RDF(S) can be reflected by extensions of the Protégé metamodel.
Although this extension has been successfully implemented for the OWL Plugin, not all aspects of the metamodels could be mapped trivially. It was straight-forward to represent those aspects of OWL that just extend the Protege metamodel. For example, in order to represent disjoint class relationships, it was sufficient to add a new property :OWL-DISJOINT-CLASSES to Proteg owl:Class metaclass. It was also relatively easy to represent OWL's complex class constructors that can build class descriptions out of logical statements. For example, OWL classes can be defined as the complement of other classes, using the owl:complementOf constructor. In the OWL Plugin, complements are represented by instances of a metaclass :OWL-COMPLEMENT-CLASS that inherits from other Protege system classes, the other types of OWL class constructors such as restrictions and enumerated classes, and the various kinds of properties are mapped into similar metaclasses. Other aspects of OWL required some work to maintain a maximum of backward compatibility with traditional Protege applications. There is a semantic difference between Protege and OWL if multiple restrictions are defined at the same time. In particular, Protégé properties with multiple classes as their range can take as values instances of all classes (union semantics), whereas OWL properties with multiple classes in their range can only take values that are instances of all classes at the same time (intersection semantics). In order to solve this mismatch, the OWL Plugin uses an internal owl:unionOf class if the user has defined more than one range class. The same applies to a property's domain. Another difference is that OWL does not have the notion of facets, which in Protege are used to store property restrictions at a class. While a maximum cardinality restriction at a class in Protege is represented by a single quadruple (class, property, facet, value), the same is stored as an anonymous superclass in OWL. OWL even supports attaching annotation property values to such anonymous classes, and therefore it would be insufficient to map OWL restrictions into facets only.

3.2 Ontology maintenance and evolution :

Ontology design is a highly evolutionary process. Ontology developers almost certainly will need to explore various iterations before an ontology can be considered to be complete. A development tool should assist in ontology evolution, and help the user to prevent or circumnavigate common design mistakes. In the OWL Plugin, some promising approaches for ontology maintenance, partly comparable to modern tools for programming languages. With programming tools, developers can get instant feedback using the compile button. Compiler errors are listed below the source code and enable the programmer to quickly navigate to the affected area. Another very efficient means of detecting programming errors is using so-called test cases, which have become popular in conjunction with agile development approaches such as Extreme Programming [6]. A test case is a small piece of code that simulates a certain scenario and then tests whether the program behaves as expected. It is a good programming style to maintain a library of test cases together with the source code, and to execute all test cases from time to time to verify that none of the recent changes has broken existing functionality. To a certain extent, the idea of test cases is related to the formal class definitions in description logics such as OWL DL. For example, by formally stating that a Parent is the intersection of Person and a minimum cardinality restriction on the hasChildren property we ensure that future statements about Parents don't contradict the original developer's intention. This is especially important in an open-world scenario such as the Semantic Web. Thus, DL reasoners can help build and maintain sharable ontologies by revealing inconsistencies, hidden dependencies, redundancies, and misclassifications . In addition to reasoners, the OWL Plugin also adopts the notions of test cases and compile buttons with an "ontology testing" feature

3.3 Reasoning based on Description Logics

One of the key features of ontologies that are described using OWL-DL is that they can be processed by a *reasoner*. One of the main services offered by a reasoner is to test whether or not one class is a subclass of another class. By performing such tests on the classes in an ontology it is possible for a reasoner to compute the *inferred* ontology class hierarchy. Another standard service that is offered by reasoners is *consistency* checking. Based on the description (conditions) of a class the reasoner can check whether or not it is possible for the class to have any instances. A class is deemed to be inconsistent if it cannot possibly have any instances.

The OWL Plugin provides direct access to DL reasoners such as Racer [7]. The current user interface supports two types of DL reasoning: Consistency checking and classification (subsumption). Support for other types of reasoning, such as instance checking, is work in progress.
Consistency checking (i.e., the test whether a class could have instances) can be invoked either for all classes with a single mouse click, or for selected classes only. Inconsistent classes are marked with a red bordered icon.
Classification (i.e., inferring a new subsumption tree from the asserted definitions) can be invoked with the classify button on a one-shot basis. When the classify button is pressed, the system determines the OWL species, because some reasoners are unable to handle OWL Full ontologies. This is done using the validation service from the Jena

library. If the ontology is in OWL Full (e.g., because metaclasses are used) the system attempts to convert the ontology temporarily into OWL DL.

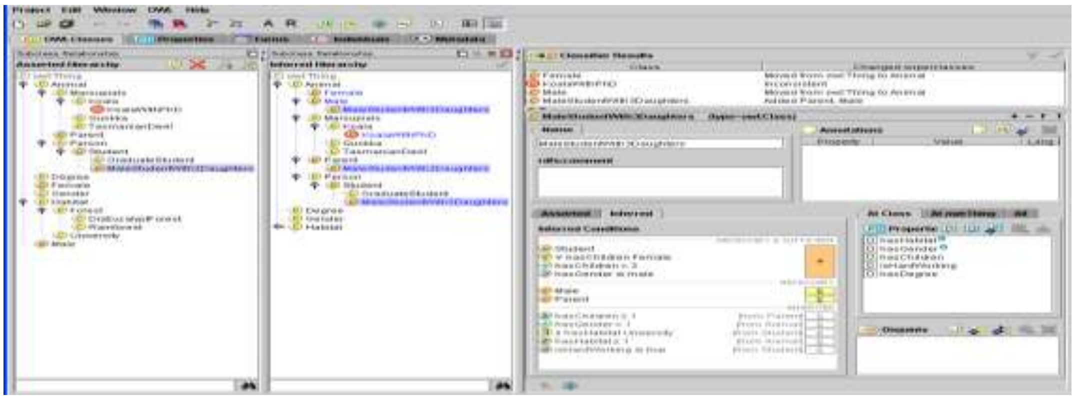


Figure 3 Racer protégé reasoned

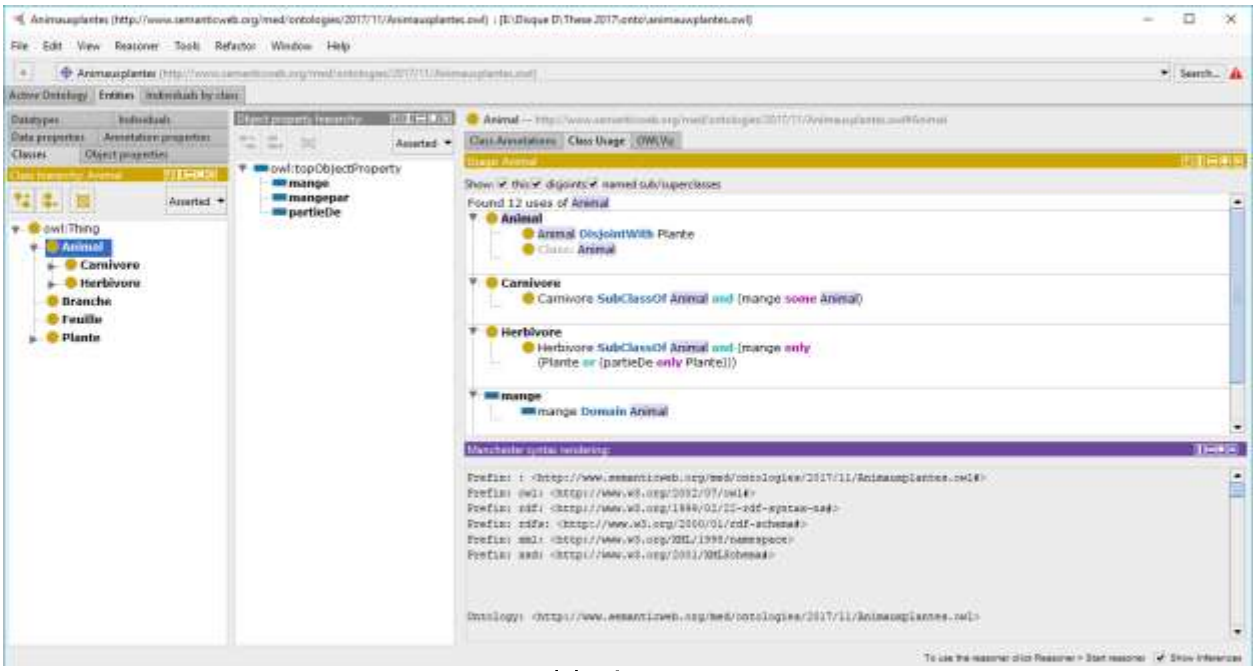
The OWL Plugin supports editing some features of OWL Full (e.g., assigning ranges to annotation properties, and creating metaclasses). These are easily detected and can be removed before the data are sent to the classifier. Once the ontology has been converted into OWL DL, a full consistency check is performed, because inconsistent classes cannot be classified correctly. Finally, the classification results are stored until the next invocation of the classifier, and can be browsed separately. Classification can be invoked either for the whole ontology, or for selected subtrees only. In the latter case, the transitive closure of all accessible classes is sent to the classifier. This may return an incomplete classification because it does not take incoming edges into account, but in many cases it provides a reasonable approximation without having to process the whole ontology. OWL files store only the subsumptions that have been asserted by the user. However, experience has shown that, in order to edit and correct their ontologies, users need to distinguish between what they have asserted and what the classifier has inferred. Many users may find it more natural to navigate the inferred hierarchy, because it displays the semantically correct position of all the classes

The OWL Plugin addresses this need by displaying both hierarchies and making available extensive information on the inferences made during classification. As illustrated in Figure 3, after classification the OWL Plugin displays an inferred classification hierarchy beside the original asserted hierarchy. The classes that have changed their superclasses are highlighted in blue, and moving the mouse over them explains the changes. Furthermore, a complete list of all changes suggested by the classifier is shown in the upper right area, similar to a list of compiler messages. A click on an entry navigates to the affected class. Also, the conditions widget can be switched between asserted and inferred conditions. All this allows the users to analyze the changes quickly

4 Ontology visualization methods in protégé :

4.1 The Indented list method

The indented list methods represent the taxonomy of the ontology following the file system explorer-tree view. These methods are intuitive and simple to implement, representing is-a inheritance relationships through the indented list paradigm, with subclasses appearing below their super classes and indented to the right. Users may navigate within the class hierarchy and expand or retract branches; when a class (or multiple classes) is (are) selected in the hierarchy pane as also confirmed by the results of a user evaluation [8]. The following figure 1 is example of the indented list method



```
Show: ✓ this ✓ disjoints ✓ named sub/superclasses
Found 12 uses of Animal
▼ Animal
      Animal DisjointWith Plante
      Class: Animal
▼ Carnivore
      Carnivore SubClassOf Animal and (mange some Animal)
▼ Herbivore
      Herbivore SubClassOf Animal and (mange only
      (Plante or (partieDe only Plante)))
▼ mange
      mange Domain Animal
```

```
Prefix: : <http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>


Ontology: <http://www.semanticweb.org/med/ontologies/2017/11/Animauxplantes.owl>
```

Figure 4 Protégé explorer-tree view

4.2 Protégé Class browser

It is a tool developed by protégé using *indented list* method. The protégé class browser consists in its simplicity of representation, and also familiarity to the user. Secondly it offers a clear view of all the class names and their hierarchy. Thirdly, its retraction and expand of nodes in useful features specific for focusing on specific parts of the hierarchy, especially for large hierarchies.
Also the open source software is readily available of this. Figure 4 sums up its main characteristics. However, the protégé class browser has certain limitations. Its technique is that it can represent a tree and not a graph. Consequently it can display inheritance (*is-a*) relations, and not role relations. This does not support visual representation of the role relations. It cannot expand all and, not retract all buttons in protégé class browser. Above all, there will be zoomable view and no overview window display.

4.3 OntoSphere

The Protégé OntoSphere is another tool using the *node-link and tree* method with 3D view. The Protégé OntoSphere has one important merit. It makes three different views available for the user viz. Root focus scene, tree focus scene and concept focus scene. However there are three limitations. Overview taxonomy structure cannot be viewed. Properties cannot be visualized and also there is no overview window display.
The figure 5 represents the OntoSphere visualization tab.

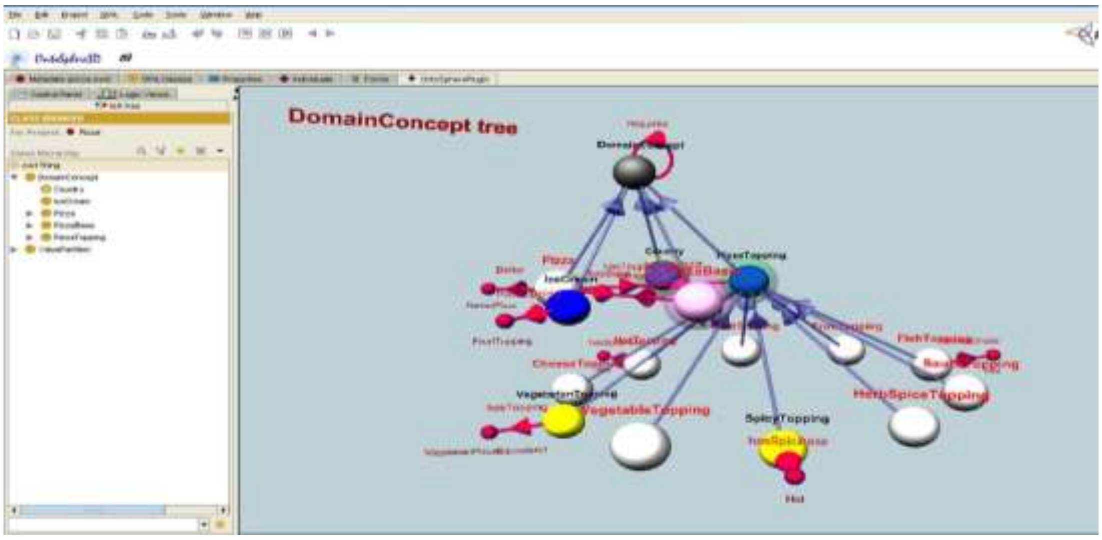


Figure 5 : The Protégé OntoSphere

4.4 Ontograph

OntoGraf gives support for interactively navigating the relationships of the OWL ontologies. Various layouts are supported for automatically organizing the structure of the ontology. Different relationships are supported: subclass, individual, domain/range object properties, and equivalence. Relationships and node types can be filtered to help you create the view you desire.

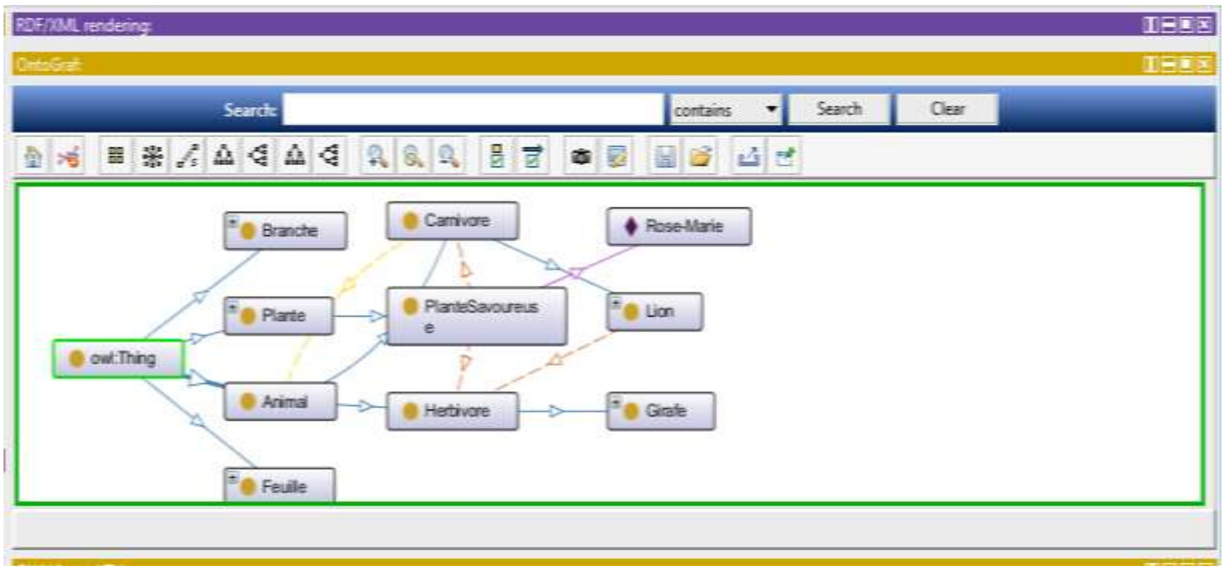


Figure 6 Protégé OntoGra

5 Discussion

While real Semantic Web applications are still in their infancy, there is a clear demand for tools that assist in application development. The intention of the Protege OWL Plugin is to make Semantic Web technology available to a broad group of developers and users, and to promote best practices and design patterns. One of the major benefits of using Protege is its open architecture. The system provides various mechanisms to hook custom-tailored extensions into it, so that external components like reasoners and Web services can be integrated easily. Since the source code is open and freely available as well, existing base components can be used as templates for customized solutions. Projects don't need to spend time developing their own base infrastructure with standard features such as loading and managing OWL ontologies. Instead, they can start with the OWL Plugin as it comes out-of-the-box and then gradually adapt or remove the features that don't completely match their requirements. One of the major benefits of OWL is that it is an official standard, it would be counterproductive to the Semantic Web if tools are "forced" into defining proprietary extensions.

6 Conclusion

The visualization of ontologies using protégé tool is a particular sub problem of this area with many implications due to the various features that an ontology visualization should present. The current work is an attempt to summarize the research that has been done so far in this area, providing an overview of the protégé tools and their main advantages and limitations. There is no one specific method that seems to be the most appropriate for all applications and, consequently, a viable solution is providing the user with several visualizations, so as to be able to choose the one that is the most appropriate for one's current needs.
Semantic web research has, however, already had a major impact on the development and deployment of ontology languages and tools now often called semantic web technologies. These technologies have rapidly become a de facto standard for ontology development, and are seeing increasing use not only in research labs but in large scale IT projects, particularly those where the schema plays an important role, where information has high value and where information may be incomplete

References

[1]Guarino, N., Giaretta, P., (1995). Ontologies and Knowledge bases: towards a terminological clarification. Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing,
IOS, 25-32

[2]Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., & Musen, M.A., (2001). Creating Semantic Web Contents with Protege-2000.IEEE Intelligent Systems, 16, 60--71.

[3]Noy, N. F., McGuiness D. L., (2001). Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880

[4] C.W. Holsapple and K.D. Joshi, "A collaborative approach to ontology design". Commun. ACM 45 (2002) pp 42–47

[5] http://protege.stanford.edu

[6] . K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999

[7]V. Haarslev and R. Moeller. Racer: A core inference engine for the Semantic Web. In 2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003), Sanibel Island, FL, 2003

[8] A. Katifori, E. Torou, C. Halatsis, G. Lepouras, C. Vassilakis, (2006). A Comparative Study of Four Ontology Visualization Techniques in Protégé: Experiment Setup and Preliminary Results,