bOWL: A Pluggable OWL Browser

Alberto Simões
2Ai, School of Technology, IPCA, Barcelos, Portugal
asimoes@ipca.pt

Ricardo Queirós
CRACS – INESC, LA, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

Abstract

The Web Ontology Language (OWL) is a World Wide Web Consortium standard, based on the Resource Description Format standard. It is used to define ontologies. While large ontologies are useful for different applications, some tools require partial ontologies, based mostly on a hierarchical relationship of classes. In this article we present bOWL, a basic OWL browser, with the main goal of being pluggable into others, more significant, web applications. The tool was tested through its integration on LeXmart, a dictionary editing tool.

2012 ACM Subject Classification Information systems → Web Ontology Language (OWL)

Keywords and phrases OWL, Web Plugin, OWL Browser, Ontology

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.18

Category Short Paper

Funding This work was partly founded by Portuguese national funds (PIDDAC), through the FCT – Fundação para a Ciência e Tecnologia and FCT/MCTES under the scope of the projects UIDB/05549/2020 and UIDB/50014/2020.

1 Introduction

With the Semantic Web, there has been a widespread of ontologies, linked data, and other resources. This is excellent, as these resources are encoded in machine readable formats, allowing their processing automatically. Nevertheless, while large data-sets are released every week, there are some specific tools that can benefit from smaller and simpler ontologies. In fact, any website that we currently use, and that is known as being Web 2.0, can use an ontology: blog software allows the use of trees of concepts, photo galleries allow tagging images with classes. All these structures could be codified as ontologies. But unfortunately, the way these sites treat classification is quite limited.

The main reason for using simple trees for classification (something that is quite near the taxonomy) is the relatively simple way they are built. While we lack proof, a reason might be the quite complicated way ontologies are presented. Indeed, ontologies are complicated structures, but at their core, they use a backbone which is (almost) a tree. In fact, the most used ontology building software (Protégé[1]) uses an interface that emphasis this specific structure.

In this paper, we explore how a simple ontology can be built using any external editor and used in a Web application, where the user can navigate it to classify individuals, and can only perform basic operations on top of the ontology directly.

[1] https://protege.stanford.edu/

Some sites already allow the navigation on an ontology, but our main contribution is to build this ontology browser (and minimal visualizer) as a plug-in, to be easily embedded in larger applications, and with a small code footprint.

2 The OWL Standard and OWL Editors

The Web Ontology Language [6] is a language designed by the World Wide Web Consortium (W3C) on top of the Resource Description Framework (RDF) specification. Its specification allows different dialects. One of the basic exportation formats by tools like Protégé [2] and its Web variant, Webprotégé [5] is a flat format, that describes the ontology classes, its annotation properties, and relations, one at a time.

2.1 The OWL Standard

In this section, we describe a few of the structures that the OWL standard allows, that are crucial for the tool being developed. Although we are aware that the OWL standard is a larger whole world, we currently are targeting a small subset, that might be expanded in the future.

- Prefix definition:

  ```
  <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
  ```

  These are single elements, that describe some IRI prefixes which can be abbreviated in the document. This allows that some elements have a `IRI` attribute, with a full IRI, and some others have an `abbreviatedIRI` attribute, that uses the prefix notation.
- Classes definition:

  ```
  <Declaration>
      <Class IRI="http://webprotege.stanford.edu/agricultura"/>
  </Declaration>
  ```

  These declarations are used to define which classes are recognized by the ontology. Each declaration includes just the class IRI.
- Data properties definition:

  ```
  <Declaration>
      <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
  </Declaration>
  ```

  Similar to the classes definition, these structures declare all data properties that can be used in the ontology. Data properties are treated as entities very similar to classes as we will see, as they can also have relations and properties.
- There are two different kind of sub classes definitions. The first one, for standard ontology classes hierarchy, OWL describe them as:

  ```
  <SubClassOf>
      <Class IRI="http://webprotege.stanford.edu/citologia"/>
      <Class IRI="http://webprotege.stanford.edu/biologia"/>
  </SubClassOf>
  ```

  Thus, in this example, the class “citologia” [citology] is a subclass of “biologia” [biology]. The relationship between classes and data properties are also described as a `SubClassOf` element, as in the next example:

```
<SubClassOf>
   <Class IRI="http://webprotege.stanford.edu/citologia"/>
   <DataHasValue>
      <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
      <Literal>citol.</Literal>
   </DataHasValue>
</SubClassOf>
```

These structures describe a relation from a class with a literal (a string) by a data property previously defined.

- OWL treats data properties as classes. Therefore, they can have their own hierarchy. Also, as they define relationships between elements, it is possible to describe their domain and range:

```
<SubDataPropertyOf>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <DataProperty abbreviatedIRI="owl:topDataProperty"/>
</SubDataPropertyOf>
<DataPropertyDomain>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <Class abbreviatedIRI="owl:Thing"/>
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty IRI="http://webprotege.stanford.edu/abbreviation"/>
    <Datatype abbreviatedIRI="xsd:string"/>
</DataPropertyRange>
```

The first block describes the abbreviation relation as a child to a top level data property. The second block defines its domain (any class child of `owl:Thing`). Finally, the latter, describes the range or co-domain of the relation: *strings*.

- Finally, there are annotation assertions, that annotate classes and relations with external objects. These are similar to relations whose range are strings:

```
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label"/>
    <IRI>http://webprotege.stanford.edu/abbreviation</IRI>
    <Literal xml:lang="pt">abbreviation</Literal>
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label"/>
    <IRI>http://webprotege.stanford.edu/agricultura</IRI>
    <Literal xml:lang="pt">agricultura</Literal>
</AnnotationAssertion>
```

These two examples show that annotations assertions can be defined both for properties (first example) or to classes (second example). For each, a relation (property) is defined, and a target language for the literal annotating the element.

These structures comprise a small subset of the OWL standard. Nevertheless, they are the kind of structures that we expect to be useful to embed on a pluggable Web widget.

2.2 OWL Libraries and Editors for the Web

There are two main types of tools to handle OWL in the Web: libraries to be used by programmers, and online editors, to be used by end-users.

As a library, we can find some JavaScript modules to handle RDF/OWL, and perform queries on them. Their main goal is not to allow the creation of the ontology, but to reason over it.

There are some of the libraries available that are written to be run in the browser (and not as a server library, running on node.js). Most are prepared only as triple stores, to load RDF documents and allow queries using SPARQL [1]:

- `rdfstore-js`[2] is built as a triple store, supporting the SPARQL query language. While it might be useful as a library, it is over complicated for our intents. Also, given bOWL is a plugin for other applications, it is important to use to keep it with a small footprint.
- `rdflib.js`[3] has similar goals as `rdfstore-js`, allowing the loading of RDF in different formats, and performing queries using SPARQL. Its versatility makes it a huge library to be embedded in a web application.
- `SPARQL.js`[4] as expected from its name, is another triple store library. In fact, it is more devoted to the parsing of the SPARQL language than on the processing of RDF.
- `rdfquery`[5] is not maintained for more than 10 years, and its main goal is to be used as a query library for RDF documents.

Regarding editors, there is not much choice. The main used editor for OWL is Protégé [2], that includes a web version, WebProtégé [5]. But this is a full blown web application, and is not prepared to be plugged into other applications, as a simple module to manage the main ontology hierarchical structure.

Protégé is implemented in Java, and WebProtégé is a simple Web wrapper to the Protégé library. They both use the OWL Api[6] Java library to generate and parse OWL files. While there is an attempt to port this library to JavaScript[7], it does not run in a browser, neither in node.js.

3 Implementation Details

The main goal when deciding for the implementation of a new tool to browse ontologies was making it embeddable, as independent as possible from other tools, and event oriented. It should also support basic edition capabilities.

3.1 bOWL internal OWL representation

Figure 1 shows the class diagram for bOWL. As it can be observed, we are focusing on very specific constructions. While in the future there might be the option to add new features, at the moment the main goal is to have it working for basic ontologies that define, mostly, a kind-of taxonomy.

Currently bOWL stores information about prefixes – in order to be able to expand or compact abbreviated URIs –, information about classes, and information about data properties.

For each class, its IRI is stored along with its parents (using the subClass relation) and its data and annotation properties. Each annotation property has the respective literal and the used language for that literal. Regarding data properties, only the IRI for the property relation and the value are stored.

The information about data properties define their IRI along with their domain and range, and annotation properties.

[2] `https://github.com/antoniogarrote/rdfstore-js`
[3] `https://github.com/linkeddata/rdflib.js`
[4] `https://github.com/RubenVerborgh/SPARQL.js`
[5] `https://code.google.com/archive/p/rdfquery/`
[6] `https://github.com/owlcs/owlapi`
[7] `https://github.com/cmungall/owljs`

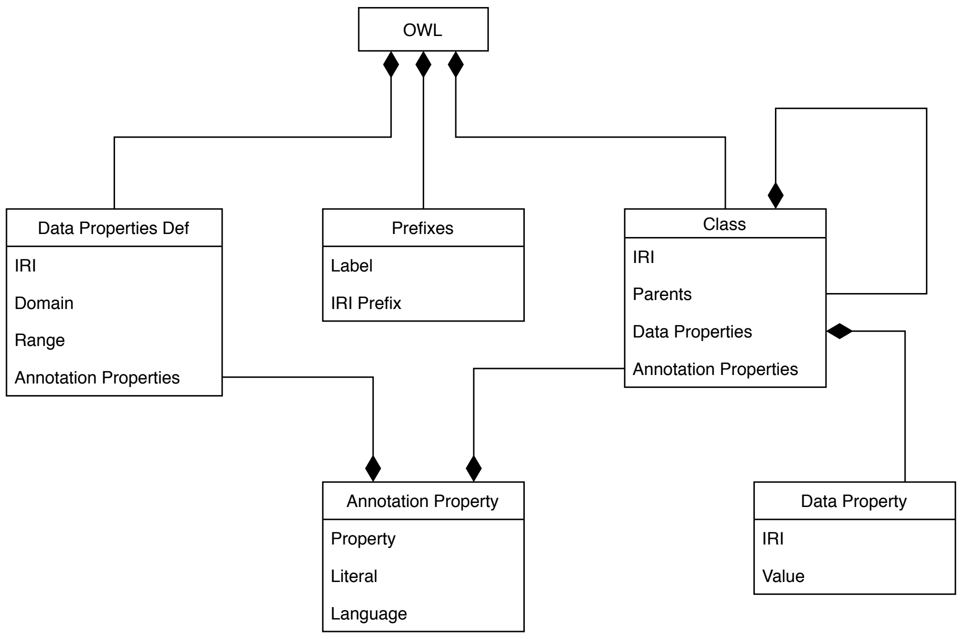

Figure 1 bOWL class diagram.

3.2 bOWL interface

bOWL is prepared to work by HTTP requests (using a GET request to fetch the ontology and a POST request to save it) or to work directly with the internal browser storage. In the future we will also implement a synchronization tool that guarantees that the ontology in the browser storage is up to date with a remote master ontology.

Given bOWL uses, internally, a JSON representation, the load and save mechanisms allow to fetch OWL directly, or to use bOWL internal JSON representation.

The communication from the world with bOWL is defined in three events/methods (see Table 1):

- The constructor, responsible for loading the ontology and showing it inside a specific element; The ontology can be load from the internal storage (in that case the constructor parameter is a string, representing the ontology name) or from a remote server (in that case, the constructor receives an URI). To make the tool more usable, the constructor can receive a second parameter with a default prefix for all created classes. This prefix will not be added in the OWL as a standard prefix: all classes will have their IRI fully qualified.
- bOWL allows simple editing of new classes (further edition capabilities are planned). Therefore, when closing the bOWL widget, the programmer might want to call a method to save the data. For saving it, the programmer might supply an URI, where the full ontology will be sent as a POST request, a string with the name of the ontology to be saved in the browser storage, or a code reference that will be run with the ontology representation as a string. A second parameter can be used to supply the saving format.
- One of the main goals is to make bOWL usable as a classification widget, allowing the user to open a popup window with the ontology and choose a class she wants to apply to some document. Thus, by default, and "apply" button is presented in the widget. When clicking this button, an event will be triggered, calling a user defined callback with that selected element data.

Table 1 bOWL programming interface.

Method	Description
`Bowl load (URI [, prefix])`	Receives an URI/Key as parameter and an optional prefix. Returns the bOWL object. Mime-type (json or xml) is automatically detected.
`Object get (event)`	Associates an event that will be triggered when the user clicks the "Apply" button. It returns an object with the selected class data.
`save (URI [, format])`	Given an URI/Key as parameter (treated like in the load method), stores the full ontology. By default, saving in local storage will use JSON while saving remotely will use OWL. This can be overridden with a second parameter. This method can also define a specific code callback to be called that should be responsible for the data save process.

3.3 Supporting technologies

bOWL is implemented using ECMAScript 8. For the XML parsing we use `js2xml`[8], that converts the XML file into a JSON data structure. Then, JSONPath Plus[9] is used to traverse the data structure and extract the required information.

For the front-end, `jstree`[10] was chosen for its versatility and easy of use. While an ontology is not necessarily a tree, it can be shown as such, repeating an element as a child of multiple parents. This is the current approach used by other tools like Protégè.

4 Proof of Concept

Our case-study is LeXmart [4, 3], a Web editor for general language dictionaries. It is developed as a Web application on top of eXist-DB[11], using Web Standards like XQuery for the application development, HTML5 and CSS for the frontend, and JavaScript for dynamic content. LeXmart features an integrated TEI editor, based on Xonomy[12].

One of the features under work, allowing the lexicographer to link dictionary senses with ontology concepts, required the ability to both show a basic ontology structure (specially its bare-bone taxonomy) and to navigate and edit it. This way, bOWL was projected as a simple JavaScript library to be integrated into LeXmart, but to be as independent as possible from it, allowing its integration with other tools.

Figure 2 shows an example of the widget in use. There is the main tree, where classes are presented in hierarchy accordingly with their OWL structure, and their `rdfs:label` annotations. When selecting an element it can be used as a parent to create a new subclass, as shown in the modal window. There, the user needs to add the IRI and the label. The parent class is automatically selected. Finally, the "Apply" button is used to send the control back to the hosting application, together with the information of the selected class.

[8] `https://github.com/x2js/x2js`
[9] `https://github.com/s3u/JSONPath`
[10] `https://www.jstree.com/`
[11] `https://exist-db.org/`
[12] `https://github.com/michmech/xonomy`

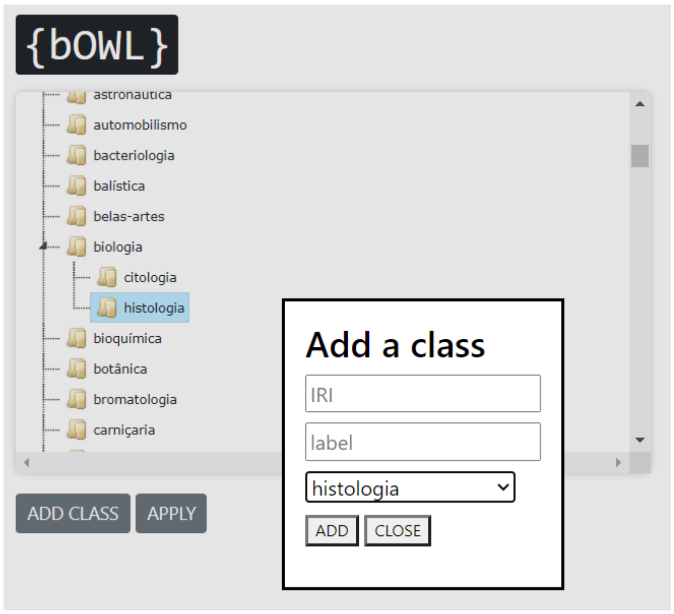

Figure 2 bOWL widget showing a (still flat) ontology from a dictionary.

5 Conclusions

At the moment bOWL is just a prototype. Nevertheless, we think it is useful for very different Web applications, and therefore, it should not be built tighten up with the underlying code. The library should be as versatile and independent as possible, in order to be pluggable in different scenarios.

As future work, we plan to improve the functionalities, but also OWL support. While some properties from the OWL data will be ignored (as they will not have a direct impact with the bOWL interface), we intend that bOWL should be able to store everything in order that, loading an ontology and saving it back is, always, the identify function.

References

1 Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, 2013. URL: `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

2 Natalya Fridman Noy, Monica Crubézy, Ray W Fergerson, Holger Knublauch, Samson W Tu, Jennifer Vendetti, and Mark A Musen. Protégé-2000: an open-source ontology-development and knowledge-acquisition environment. In *AMIA Annual Symposium Proceedings*, volume 2003, pages 953–953, 2003.

3 Alberto Simões, José João Almeida, and Ana Salgado. Building a Dictionary using XML Technology. In Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira, editors, *5th Symposium on Languages, Applications and Technologies (SLATE)*, volume 51 of *OASIcs*, pages 14:1–14:8, Germany, 2016. Schloss Dagstuhl. `doi:10.4230/OASIcs.SLATE.2016.14`.

4 Alberto Simões, Ana Salgado, Rute Costa, and José João Almeida. LeXmart: A smart tool for lexicographers. In I. Kosem, T. Zingano Kuhn, M. Correia, J. P. Ferreira, M. Jansen, I. Pereira, J. Kallas, M. Jakubíček, S. Krek, and C. Tiberius, editors, *Electronic lexicography in the 21st century. Proceedings of the eLex 2019 conference*, pages 453–466, 2019.

5 Tania Tudorache, Jennifer Vendetti, and Natalya Fridman Noy. Web-protege: A lightweight owl ontology editor for the web. In *OWLED*, volume 432, page 2009, 2008.

6 W3C OWL Working Group. OWL 2 web ontology language document overview (2nd edition). W3C recommendation, World Wide Web Consortium, December 2012. URL: `http://www.w3.org/TR/2012/REC-owl2-overview-20121211/`.