A Python Utility for Working with OBO Foundry Terms

Jonathan P. BONA[a,1]
[a] *Department of Biomedical Informatics, University of Arkansas for Medical Sciences*

Keywords. Python, OBO Foundry, Tools

1. Introduction

This poster describes a simple utility that facilitates working with Open Biomedical Ontologies Foundry resources within programs implemented in the Python language. This utility allows a programmer to import a representation of an OBO Foundry ontology as a Python class, and then use that Python class within a program to refer to terms in the ontology using their labels rather than managing strings representing term URIs.

Term identifier conventions in the OBO Principles[1] require ontology terms to use numeric local term identifiers, and forbid local identifiers that "consist of labels or mnemonics meaningful to humans." This requirement has the unfortunate side effect of making it difficult for humans to work directly with the identifiers. In order to write by hand (or in code) an RDF assertion that a certain individual is an instance of the UBERON[2] term 'lung', one must know that the URI is `http://purl.obolibrary.org/obo/UBERON_0002048`. A reusable and human-readable way of associating that URI with its label within the code is preferrable.

In Python software that we have written to transform instance data into semantic representations using OBO Foundry ontologies, including as part of the PRISM[3] project, we started managing the association between ontology labels and URIs with ad-hoc mappings involving only terms that were of immediate interest. For example, we might maintain a Python dictionary object for all the UBERON terms used in our project and then use the Python expression `uberons['lung']` to retrieve the URI for that term. Among the disadvantages of this approach is the need to look up individual terms using external resources and then copy their labels and URIs into source code. This manual step is clunky, error-prone, and difficult to reuse from one project to the next.

The solution provided by this utility is simple but useful: rather than managing term label/URI mappings for oneself in a body of Python code, we provide the ability to import each ontology as a Python class that contains the labels and URIs for terms defined in that ontology.

A draft implementation is available in the public GitHub repository at https://github.com/jonathanbona/obof-py.

[1] Jonathan P. Bona, Corresponding author, Department of Biomedical Informatics, University of Arkansas for Medical Sciences. 4301 W. Markham St., #782 Little Rock, AR 72205-7199; E-mail: jpbona@uams.edu

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Methods

This tool and the capability described above is implemented by generating a Python class for each ontology. For each term defined in an ontology, the ontology's Python class has a class attribute named using the term's label, with underscores substituted for spaces where applicable. The value of a term's attribute is a string representation of its URI.

Following the running example, one uses the line `from obo import UBERON` at the top of a Python source file that will use UBERON terms, and then uses the Python class `UBERON` to get the URI for any term in the UBERON ontology. The following lines show an interactive Python session getting the URIs for two terms.

```
>>> UBERON.lung
'http://purl.obolibrary.org/obo/UBERON_0002048'
>>> UBERON.lobe_of_liver
'http://purl.obolibrary.org/obo/UBERON_0001113'
```

Any Python development environment with code completion will assist in locating the desired term. For instance, typing `"UBERON.lung"<TAB>` within python-mode in the emacs editor brings up a list of 17 term labels beginning with the substring "lung."

This tool includes the ability to download an OBO Foundry ontology from its URI and generate the Python class representation of that ontology. We do not intend users to be required to initiate the download of ontologies and conversion to Python classes. Rather, the information used by this tool (at present only URIs and their labels) can be extracted and cached in a simpler format for distribution.

UBERON, for example, contains over 15,000 classes, and its OWL file is over 65MB in size. Rather than either (1) downloading this OWL file at the time the UBERON Python class is imported, or (2) distributing entire OWL files for all OBO ontologies as part of this tool, we can instead generate the Python classes and serialize those as compressed pickle files to distribute with this package. These compressed files take up very little space – 4KB for the UBERON Python class.

3. Discussion

We have built a simple tool that vastly simplifies working with terms in OBO Foundry ontologies from within Python programs. This has been tested and used so far with a handful of ontologies, but it is still under active development and has remaining issues to be addressed. One is its handling of imports within ontology files. Another issue will be the release schedule for versions of this tool: since it consists mainly of Python classes built automatically from OBO Foundry ontologies, it will need to be updated regularly as those ontologies are updated. We will target an automated monthly update. Planned extensions to the basic functionality of this tool include the ability to search term labels from within code, as well as possibly the option to search additional attributes.

References

[1] The OBO Technical Working Group. http://www.obofoundry.org/principles/fp-003-uris.html

[2] Mungall CJ, Torniai C, Gkoutos GV, Lewis SE, Haendel MA. Uberon, an integrative multi-species anatomy ontology. Genome biology. 2012 Jan 1;13(1):R5.

[3] Sharma A, Tarbox L, Kurc T, Bona J, Smith K, Kathiravelu P, Bremer E, Saltz JH, Prior F. PRISM: A Platform for Imaging in Precision Medicine. JCO Clinical Cancer Informatics. 2020 Jun;4:491-9.