A pipeline for biomedical ontology maintenance and release: current status and future requirements

Carlo Torniai[1*], Matthew Brush[1]and Melissa Haendel[1]

[1]Library and Department of Medical Informatics of Oregon Health & Science University, 3181 S.W. Sam Jackson Park Rd. Portland, OR 97239-2499

ABSTRACT

Summary: This note describes a maintenance and release pipeline for ontologies that leverage a modular reuse of external ontologies. Here we describe our approach and identify a set of requirements for continued enhancement and integration of external tools.

1 INTRODUCTION

One hopes that when creating well-designed ontologies for a certain domain that they will be reused and extended by others. Despite the large effort to promote reuse of existing ontologies, in particular within the biomedical domain (Smith 2007), the tools and methodologies that facilitate multiple ontology integration and reuse are far from being mature. We have previously published on our ontology engineering approach (Torniai 2011) to build modular application ontologies leveraging the MIREOT principle (Courtot 2009). In this process we have recognized the advantages of maintaining synchronization with externally referenced ontologies, specifically to facilitate data currency and therefore interoperability. Here, we describe the approach we have developed for an ontology maintenance and release pipeline that takes into account these needs for ontology reuse and interoperability, and identify requirements for available tools to be integrated into this process.

2 METHODS

For the ontologies we have developed (eagle-i and CTSAconnect among others), we have used Protégé for editing and we have included references to external ontology entities using the MIREOT principle. In this context, our approach to develop a maintenance and release pipeline was driven by three requirements:

- Facilitate editing, maintenance and release of the ontology even by non-technical people
- Reuse available tools
- Automate the maintenance and release process as much as possible

The requirements above translated into the following decisions:

- Use Ontofox (Xiang 2010; http://ontofox.hegroup.org/) as an implementation of the MIREOT principle
- Use the OBO Ontology Release Tool (Oort) (http://code.google.com/p/owltools/wiki/OortIntro) to manage releases.
- Define specific organization for the files containing external referenced entities

*To whom correspondence should be addressed

Rather than having all the entities referenced from external ontologies together in one single 'external.owl' file, as originally suggested in the MIREOT specification (http://bit.ly/Mt5xoo), we create a file for each external ontology source. These files are grouped under a same directory and named according to the following convention: [ontology-prefix]-imports.owl (Fig. 1A). For instance the main ontology file for eagle-i (ero.owl) imports all these "imports" files as well as full ontologies such the basic formal ontology (BFO), the Relation Ontology (RO), and the Information Artifact Ontology (IAO) (Fig. 1B). The separation of each import file according to the source ontology was functional to implement the workflow for updating MIREOTed classes and properties represented in Figure 1C.

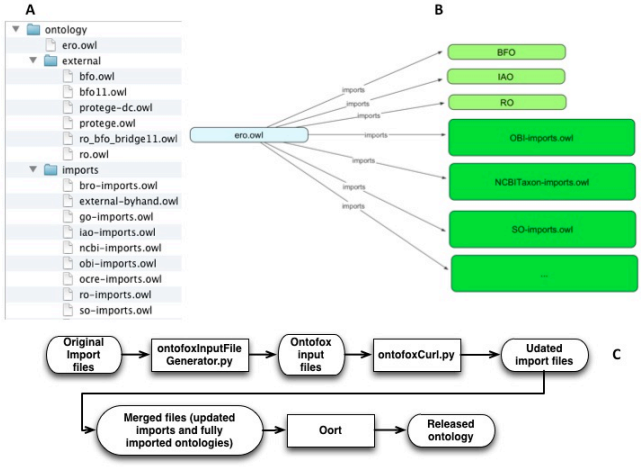

Fig. 1. A) The base directory structure for the ero ontology with all the imports files. B) The chain of imports of the main ontology .C) Release pipeline: MIREOTed entities (classes and properties) update and usage of Oort

A Python script (ontofoxInputFileGenerator.py) generates an Ontofox input file by parsing each imports file, pulling classes and their subClassOf information and defining custom preferences about how to map annotations on these classes to appropriate IAO annotation properties (See Fig. 3). The generated Ontofox input file is used by another Python script (ontofoxCurl.py), which accesses Ontofox programmatically to return an updated owl file.

The MIREOTed entities update pipeline is the first step of our release workflow: it generates all the updated files used by the

Oort tool, which generates all the ontology release files (see http://code.google.com/p/eagle-i/#Description for an example).

3 DISCUSSION

While implementing this pipeline, we identified a set of feature requests for Ontofox (e.g. the need to provide within the output file a list of classes that are no longer in use in the referenced source ontology) and Oort (e.g. the need for referencing local ontology files through the catalog xml file generated by Protégé) that were promptly implemented by the respective developer teams.

We could have used a similar workflow while maintaining a single file for the MIREOTed entities. However, having a single file for each source ontology was advantageous because it is easier to manage and synchronize external entities with each ontology source and it facilitates parallel editing. Use of a single, large owl file to hold all the external entities posed an issue for editors because it was difficult to manage changes from the sources that are all in different locations and follow different release schedules. Moreover there are cases in which there is the need to apply MIREOT differently according to source ontology. For instance, each source uses a different set of annotation properties to record class metadata, and specification of mappings for non-IAO properties needs to reflect these differences. There are also cases where developers will want to specify different "levels" of MIREOT for different source ontologies, for example where class axioms need to be imported along with annotation properties. We have found this flexibility important in our reuse of classes from the Ontology of biomedical Investigations. Using separate import files for each source ontology has been essential for facilitating such needs for customizing MIREOT principle for specific source ontologies.

An interesting consideration about our design pattern is that the minimal information about the imported entities (the content of the external.owl file defined in the original MIREOT specification), is now specified in the Ontofox input file generated in our pipeline. It would be extremely useful if Ontofox would support as input the actual owl import files, or at least an owl file with the following information: entity to import, entity superclass in the target ontology, annotations to specify the "level" of MIREOT desired (simple annotation, full closure, and so on).

While it is true that different ontologies have different development strategies and different teams (and skills) involved in the development, there are some common requirements for maintenance and release tools that we have identified across several ontology efforts. One such issue is that not all the ontologies are available in Ontofox or from a single SPARQL endpoint. For instance, we use the VIVO ontology (http: http://vivoweb.org/) and the Ontology for Clinical Research (OCRe; http://code.google.com/p/ontology-of-clinical-research/) for which we need to maintain a separate updating process. Therefore, another important feature for maintenance and release tools would be to be able to configure different SPARQL endpoints (or other access means) for different ontology sources.

Another important issue is the need to have reporting about what has been updated and what has been changed during the update process of MIREOTed entities. Currently, we run our tests (comparing the original and the updated imports file) through a custom set of Java scripts and enhancements to this end have been requested to Oort developers.

Finally, we believe that in addition to Oort and Ontofox, Protégé (http: http://protege.stanford.edu/) should play a key role in an integrated pipeline of ontology maintenance and release. It would be great if Protégé generated the proper file structure for imported entities and performed the updates (through a plug-in that could, for instance, programmatically access Ontofox). To this end, we are providing requirements to the Protégé team in a coordinated initiative together with other efforts (GO (http://www.geneontology.org/), the OBI consortium (http://obi-ontology.org/page/Main_Page), VirtualFlyBrain (http://www.virtualflybrain.org/) and others).

4 CONCLUSION

The design pattern and the maintenance and release pipeline we have described has been adopted successfully for the Reagent Ontology (ReO, http://code.google.com/p/reagent-ontology/) and for the new version of Common Anatomy Reference Ontology (CARO, http://code.google.com/p/caro2/).

Longer term, such a pipeline should be integrated into existing tools and we will continue to feed our requirements to Oort, Ontofox and to the Protégé team as well. If these tools will provide a flexible level of customization the biomedical ontology community will be able to have more automated, standardized reliable and widely usable maintenance and release pipeline for building modular ontologies using the MIREOT principle.

ACKNOWLEDGEMENTS

The authors wish to thank, in addition to the the eagle-i consortium, the Ontofox and Oort developer teams - in particular Allen Xiang, He Yongqun, Chris Mungall, and Heiko Dietze for their support and prompt responsiveness in addressing our requests.

Funding: This work was supported by the National Institutes of Health and the American Recovery & Reinvestment Act [grant number 1U24RR029825-01].

REFERENCES

Smith B. *et al*. (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration, *Nature Biotechnology* 25, 1251 - 1255. http://www.nature.com/nbt/journal/v25/n11/full/nbt1346.html

Torniai C. *et al*. (2011) Developing an application ontology for biomedical resource annotation and retrieval: challenges and lessons learned, Proceedings: International Conference on Biomedical Ontology, Buffalo, NY.

Courtot, *et al*. (2009) MIREOT: the Minimum Information to Reference an External Ontology Term. ICBO, Buffalo

Xiang Z *et al*. (2010). OntoFox: web-based support for ontology reuse. *BMC Research Notes*., **3:**175. [PMID: 20569493]