Getting the Best from Two Worlds: Converting Between OBO and OWL Formats

Vicky Dritsou[1], Elvira Mitraka[1,2], Pantelis Topalis[1], and Christos Louis[1,2]

[1] Institute of Molecular Biology and Biotechnology, Foundation for Research and Technology, Heraklion, Crete, Greece
[2] Department of Biology, University of Crete, Heraklion, Crete, Greece
{vdritsou, elvira, topalis, louis}@imbb.forth.gr

Abstract. OWL is one of the most popular ontology languages, mostly due to its expressiveness as well as its formal semantics. Yet, in the biomedical domain, OBO, a language that was expressly developed for the construction of bio-ontologies, is the format preferred by the community for this purpose. This is best exemplified by the fact that OBO-based ontologies are driving the model organism databases that use the common schema named Chado, a standard in the GMOD project. Thus, converting ontologies written in OWL to OBO and vice versa are crucial processes, in order to take full advantage of all resources that are being developed. In this report we describe first the hurdles that we faced while using the existing tools for converting ontologies between the two different formats. Furthermore, we present two scripts that we developed in order to overcome the recorded difficulties. A number of tests performed on several bio-ontologies show that these scripts are in position to successfully carry out these conversions.

1 Introduction

In the Semantic Web community, the Web Ontology Language (OWL[21]) has been the recommendation of the World Wide Web Consortium (W3C) for defining and expressing ontologies, thus serving as a very popular web standard. Its well defined semantics and the high expressiveness it provides, facilitated the increase of its popularity even more. Indeed, several tools and applications have been developed for OWL ontologies. Among others, Protégé[8] is one of the most widely known open-source OWL ontology editors. Many reasoner engines have also been developed, such as RacerPro[17] and FaCT++[27].

At the same time, researchers in the biomedical community have put a lot of effort in developing biomedical ontologies; the National Center for Biomedical Ontology BioPortal[3] currently hosts 305 ontologies. In contrast to ontologies representing other domains, the majority of these bio-ontologies is structured according to OBO Flat File Format[24]. This was first introduced in the previous decade, when the Gene Ontology[10] was developed, and is now widely adopted. OBO ontologies are expressed by defining `stanzas`, which can be either `terms`, `typedefs` or `instances`. Each `stanza` is followed by tag-value pairs, with valid tags being predefined in the OBO specification. Most of its statements can be converted to OWL statements with

[3] http://bioportal.bioontology.org/

respect to their semantics. Semantical issues can rise in few specific cases when converting these expressions, for example due to the multiple synonym elements that OBO allows to be defined. In the opposite direction, OWL is considered a more powerful ontology with strict formalization and semantics, allowing the developer to express `classes`, (different types of) `properties` and `resources`, even `rules` and `axioms`. The two latter cases can become complex sometimes and there are cases where the expressiveness of OWL can not be described in OBO. Towards the vision of the Semantic Web however, the ability to share and reuse knowledge among different ontological sources becomes a necessity. A variety of systems has been developed that can import ontologies of only one format. For example, the model organism databases that use the common schema named Chado[23], a standard in the GMOD project[4], rely on OBO ontologies only. In order to take advantage of the knowledge being expressed, regardless of the format it is structured in, we need the means to convert such ontologies, from OBO to OWL and vice-versa. These conversions may lead to minor loss of information or no loss at all. Moreover, when such conversions are applicable, OBO developers can take advantage of the various OWL reasoners to check and validate the consistency of their ontologies.

Converting ontologies between OBO and OWL has already been attempted and applied. In Section 2 we provide a description of methods and tools that have been proposed in the literature. However, our experience has shown that there are some special cases, where either the methods fail to convert successfully the ontologies or where they omit information that could be maintained. In an attempt to overcome these obstacles, we have developed two Perl scripts that convert OBO files to OWL and backwards. These are described in Section 3. Finally, in Section 4 we summarize the findings of this work and provide concluding remarks.

2 Existing Tools and Experiences

Procedures meant to convert OBO ontologies to OWL and vice versa have already been described in the literature and several tools have been developed for this purpose. OWLTools[5] is a popular java Application Programming Interface (API) for managing and analyzing ontologies with many features, built on top of the java OWL API[20] created for manipulating OWL ontologies. One of the included applications of OWLTools is the Obo Ontology Release Tool (OORT)[7]. This tool exploits the oboformat converter[3] and performs conversions between OBO and OWL files. In order to fully describe all the expressions present in an OBO file in OWL, the converter uses the oboInOwl mapping[4]. This mapping is also used in two popular plugins, one for each of the most commonly used ontology editors for each format: (i) in Protégé ontology editor, the OBO Converter Tab plugin[22] reads and writes OBO files, and (ii) in OBO-Edit[13], which is an OBO ontology editor, the OboEdit OWL plugin[2] imports and exports ontologies in OWL format. The same mapping is used in ONTO-PERL Application Programming Interface[9], a java application that is built on a set of object-oriented Perl modules and which enables such conversions. Other mappings for the conversion of OBO files to OWL have been proposed in the literature as well. In [25] the authors describe a methodology for mapping OBO ontologies to OWL, while a grammar for OBO syntax and its derived mapping is proposed in [14] and [15]. In [19] the authors

[4] http://gmod.org/wiki/Main_Page

implement their own mapping form OBO to OWL avoiding a loss of information for roundtrip conversions (OBO to OWL to OBO). Regardless of the mapping we adopt to transform an OBO file to OWL, a common feature of such transformations is that the resulting OWL file contains statements that help us identify the initial expressions present in the OBO format. These statements are identified by the aforementioned converters in order to produce an OBO file, i.e. to perform the reverse conversion. However, there are some cases where an ontology written from scratch in OWL must be converted to the OBO format, e.g. when it is required to import an ontology in OBO-based tools like the relational database schema for biological knowledge Chado[23]. If the OWL ontology we wish to convert does not rely on any OBO to OWL mapping (i.e. is written from scratch in OWL), then the above converters will not be able to identify all the statements contained in it. For instance, imagine that we want to convert the following OWL statements to OBO

```
<owl:Class rdf:ID="alanyl_tRNA">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="tRNA"/>
  </rdfs:subClassOf>
  <rdfs:isDefinedBy rdf:datatype="http://www.w3.org/
  2001/XMLSchema#string">A tRNA sequence that has an
  alanine anticodon, and a 3′ alanine binding region.
  [SO:0000254]</rdfs:isDefinedBy>
</owl:Class>
```

and that we use one of the above converters. Then the resulting OBO statements will be

```
[Term]
id: alanyl_tRNA
is_a: tRNA !
```

which means that we have lost the definition of the concept. Consider for example that we use the Oort tool mentioned above to transform this OWL statement to OBO. Then the definition could be identified by the tool if it was embraced by the tag `<obo:IAO_0000115>` referring to the ID of the annotation property for definitions in the Information Artifact Ontology[1]. Moreover, this definition could be identified by the Protégé OboConverter Tab if the tag `<oboInOwl:hasDefinition>` was used. In both cases, the definition of the term will be lost.

In cases like the one described in the above example, we loose important information. Recall that an ontology is “formal, explicit specification of a shared conceptualization” [16] where all concepts should be accompanied by their definition. In OWL ontologies the formal way of expressing the definition of a class (or any other concept) is by using the tag `<rdfs:isDefinedBy>`. If we were able to identify this tag within the OWL files, then we would maintain the information into the generated OBO file. Of course, there is a difference between the definition expressed in OWL with the definition in OBO: while in OWL we only need to express the statement that defines the concept, in OBO this statement should also be followed by the cross reference where this definition originates from. If this cross-reference does not exist in the OWL definition, we believe that it is more preferable to retrieve the definition statement even without a cross-reference than to loose it entirely. Therefore, our goal here is to develop a tool that is able to convert such ontologies that were originally written in OWL without loosing any important information that can be also expressed in OBO.

Moreover, while attempting to make conversions in both directions (i.e. both from OWL to OBO and from OBO to OWL), we have experienced some issues regarding the scalability of the existing tools when dealing with large, but not huge, ontology files. More precisely, none of the above tools was able to successfully complete (due to memory issues) the conversion of either the GAZ[5] ontology or the NCBI[6] taxonomy, to which 134MB and 76,3MB of disk space are allocated, respectively. To be more precise, our tests were performed using an Intel Core i3 Processor clocked at 2.53GHz with 4GB RAM running Ubuntu 11.10 operating system. More efficient tools could thus be developed, so that they would scale when using such file sizes and enable the conversions of larger ontologies.

3 Conversion Scripts

In an attempt to overcome the issues faced with the existing tools, we have developed two individual scripts in Perl to transform ontologies between the two different formats, OBO and OWL. The scripts are available at `code.google.com/p/obowl`.

3.1 OWL to OBO Script

Using OBO to OWL mappings for the conversion between these two formats is important to fully express and maintain all the information in OBO ontologies. However, when relying on such mappings for the conversion of arbitrary OWL files to OBO we cannot maintain all the information, even in cases where the lost information can be expressed in OBO. By the term arbitrary OWL files we mean here files that have not been written by considering any OBO to OWL mapping, but instead have been written from scratch in OWL and use its predefined tags.

In our work we have developed a Perl script that converts such OWL files, written in RDF/XML syntax, to OBO format. Alongside arbitrary OWL files, this script also successfully converts to OBO the ontologies that rely on the oboInOwl mapping for expressing the OWL statements. The script takes as input an OWL file and gives in the output the OBO file (together with a supplementary file explained later). The process followed for the conversion is described in Algorithm 1. First, the script opens the files and starts processing the OWL statements. In order to be able to parse large OWL files (i.e. make the script scale), we chose not to load the contents into memory but instead we make use of the Perl Tie::File function for this processing. The Tie::File function represents a file as an array, where each element corresponds to the contents of a line in the file. It does not load the file into memory, and as a consequence it can deal with very large files. We then process the first element of the tied array, i.e. the header block, line by line. For each line we try to match the contained statement to one of the predefined patterns, this way detecting the corresponding OBO statement. After completing the header block, we continue with a similar process for all remaining elements. When entering each block, we first detect whether it concerns a class or a property. Then, we make use of two data structures, two hashes of arrays, one for the detected `terms` and one for the detected `typedefs`. Within each hash we store the information that will be afterwards printed to OBO file regarding each identified `stanza`. The keys of

[5] http://bioportal.bioontology.org/ontologies/1397
[6] http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/

Algorithm 1 OwlToObo

```
Require $owlFile$
Open $owlFile$, $oboFile$ or else die with error message
tie each block of statements of $oboFile$ to $temp$
for all $line$ in the first element of $temp$ do
    match $line$ with one of the predefined expressions
    print to $oboFile$ the corresponding owl statement
end for
for all $element$ of $temp$ (besides the first) do
    match its $statement$ with predefined expression
    if $statement$ can not be matched then
        print $statement$ to $logFile$
    else
        add to hash the tag-value pair
    end if
end for
sort the two hashes alphabetically based on their IDs
for all $term$ contained in hash do
    print to $oboFile$ all the stored information
end for
for all $typedef$ contained in hash do
    print to $oboFile$ all the stored information
end for
if $logFile$ has contents then
    prompt the user for unmatched statements
end if
```

Algorithm 2 OboToOwl

```
Require $oboFile$
Open $oboFile$, $owlFile$ or else die and print error message
tie each block of statements of $oboFile$ to $temp$
for all $line$ in the first element of $temp$ do
    match $line$ with one of the mapping expressions
    print to $owlFile$ the corresponding owl statement
end for
print to $owlFile$ all required annotation properties
for all $line$ of $temp$ (besides the first) do
    match $line$ with one of the predefined expressions
    if $line$ can not be matched then
        print $line$ to $logFile$
    else
        print to $string$ the corresponding owl statement
        if then$string$ contains special characters
            replace any special characters by numeric reference
        end if
        append $string$ to $owlFile$
    end if
end for
if $logFile$ has contents then
    prompt the user for unmatched statements
end if
```

the hashes are the IDs of the concepts and within the arrays of each key we keep the tag-value pairs of the concept. For each block, the algorithm examines the statements line by line. When the end of the statement has been found, it attempts to match the predefined strings to its contents. If the statement fails to get matched, then it is printed in a log file together with the line it appears in. Otherwise, the statement is matched and the corresponding tag-value pair is stored as an array in the hash it refers to. According to the specification of OBO, `stanzas` should appear in a specific order in the file. We thus sort the hashes of `terms` and `typedefs` in alphabetical order and finally print to the output file the OBO statements starting with the `terms` in alphabetical order followed by the `typedefs` also in alphabetical order. While exiting, if the supplementary log file has contents, i.e. if there are statements that failed to get matched, the user is prompted that there exists information that is missing.

Special attention is paid whenever we store the definition of a `term` or `typedef`. As we have mentioned in the previous section, definitions in OBO are followed by a cross-reference to the source they originate from. Therefore, whenever detecting an OWL definition, we parse the statement and try to identify whether it follows this structure. If it does follow this structure, we do not ignore it, but instead we convert this expression into OBO. In order to be fully compatible with the latter and avoid parsing errors, we add after the definition the "invalid" cross-reference `[fromOwl:fromOwl]`.

Backward compatibility to previous OBO specifications has also been considered in our work. More precisely, version 1.4 of OBO specification declares the `name` tag as optional. However, in the previous versions (1.0, 1.2) this tag was required. Even though the most recent specification considers it optional, some existing tools can be OBO1.2-dependent, like Chado for example. In order to provide backward compatibility, we give to the user the option to choose whether the `name` tag would be optional or required in each conversion. Consider now that the user chooses names to be required and that

From OWL to OBO	From OBO to OWL
(converted) GAZ	GAZ
Information Artifact Ontology (IAO)	Gene Ontology[10]
Single Nucleotide Polymorphism Ontology[12] (SNPO)	NCBI Taxonomy
RNA Ontology[18] (RNAO)	Sequence Ontology[6]
Gene Regulation Ontology[11] (GRO)	Malaria Ontology[26] (IDOMAL)

Table 1: Ontologies Tested by Proposed Scripts

the script identifies in the OWL file a `class` with no defined `label`. Then it will be converted to a `term` having a `name` identical with its ID.

To check whether the script scales, we have tested it with the OWL version of GAZ (generated by the reverse script we implemented) which has a size of 370MB. Using the same machine mentioned earlier (Intel Core i3, 4GB RAM, Ubuntu 11.10) the script completes successfully the conversion in 949sec (15min), whereas other existing tools failed to complete the conversion: OORT freezes due to lack of memory, OWLDEF fails to handle such large file, while Protégé runs out of memory while loading the OWL file, so that a conversion cannot be performed. Besides GAZ, the script has also been tested with a number of other (smaller) OWL ontologies that do not rely on any OBO to OWL mapping. A list of these tested ontologies is presented in the leftmost column of Table 1. Regarding all mentioned ontologies, the script terminated with success requiring only a few seconds to convert them to OBO.

3.2 OBO to OWL Script

Recall that in Section 2 we report on the scalability issues faced when trying to convert large OBO files to OWL. To overcome these issues, we have implemented a Perl script that automatically performs this conversion, taking in the input the OBO file and giving the corresponding OWL file in the output. A decision we had to take before implementing the script was whether we would use in the generated OWL file one of the available mappings or not. First of all, we want to give the opportunity to users that start from an arbitrary OWL file to perform a round-trip of conversions and get in the output the initial arbitrary OWL file. However, we also wish to be compatible with existing tools that consider mappings in their converters. Considering these facts, we have decided to implement both options and let the user decide if a mapping will be included or not. Since our reverse script takes into account both versions of OWL files, OBO files can be successfully converted by the proposed script to OWL with or without mapping. In case users choose the “with mapping” option, then the oboInOwl mapping is used to express the OWL statements. Otherwise, we express the required `Annotation Properties` within the generated OWL file and make use of tags defined in the OWL specification.

The incremental steps of the algorithm are presented in Algorithm 2. First, the script tries to open the two required files. If an error occurs while opening the files, the algorithm dies printing the appropriate message. Following, we once again use the Tie::File function of Perl to link the file contents to an array. As we have mentioned earlier, this function gives us the ability not to load the contents into memory but, instead, load only one element each time. This makes the script much less memory consuming with the ability to deal with large files (even with files taking up a few gigabytes in disk space). Moreover, this time we do not need to create any other data structures to store additional information, since there is no restriction on the ordering of statements in OWL. We therefore `tie` each block of statements found in the OBO file to an element of an array and process the file sequentially. First we consider the header (first element of array), and match each line with one of the expressions that implement the correspondences between tags. We then print to the output file all the `Annotation Properties` required, either by using the mapping or not. For the remaining elements of the array, we parse them sequentially and for each line of the block, we match its contents with one of the known OBO tags. If for any reason the script fails to identify a tag, then the statement is written to a log file accompanied by the number of the line it appears at. Each identified tag is converted to the corresponding statement in OWL, all special characters it possibly contains are replaced and the final statement is printed to the output file. The algorithm terminates when all elements have been parsed and prompts the user while exiting if non-identified statements have been detected.

The scalability of the script has been tested with some ontologies that are relatively big showing successful results, whereas it has been proved to be efficient in process time as well. When converting the GAZ ontology with the same machine mentioned above, the script took $663sec$ ($11min$), while OORT fails to run this conversion (freezes due to lack of memory), Protégé plugin also fails due to the special characters contained in the ontology that are not allowed in OWL and OWLDEF fails to open such a large file. Besides this big ontology, others have also been tested with successful results. The list of these ontologies is presented on the rightmost column of Table 1.

4 Conclusions

In the last decade a lot of effort has been made to express domain knowledge with the aid of ontologies. In order to take full advantage of knowledge sharing and reuse, we need to develop the appropriate tools so that these ontologies can "understand" each other. Therefore the goal of successfully converting OBO ontologies to OWL and conversely becomes a necessity. This problem has received a lot of attention during the past years, thus resulting in the development of a variety of tools. However, to the best of our knowledge, there is no tool that deals with "arbitrary" OWL files and converts them to OBO. At the same time, scalability issues rise when the size of the ontology becomes relatively large. In this work, two Perl scripts we have developed for such conversions are described. Our goal was to build the necessary tools to overcome the hurdles we have come across. Indeed, a variety of tests we performed, using different ontologies, show that our scripts manage to successfully convert them between the two formats, even in cases when other tools fail to do so.

Acknowledgements

We would like to thank Emmanuel Dialynas for his help throughout this work. The work was supported by the NIAID (contract HHSN272200900039C to the core VectorBase project) and the Hellenic General Secretariat for Research and Technology (MicroRNA project, 09SYN-13-1055).

References

1. Information Artifact Ontology, http://code.google.com/p/information-artifact-ontology/
2. OBOEdit OWL Plugin, http://smi-protege.stanford.edu/ nigam/OboEditPlugin.zip
3. Oboformat: Parser and OWLAPI mapping for OBO Format, http://code.google.com/p/oboformat/
4. oboInOwl Mapping, http://www.bioontology.org/wiki/index.php/OboInOwl:Main_Page
5. OWLTools Java API, http://code.google.com/p/owltools/
6. Sequence Ontology, http://www.sequenceontology.org/
7. The OBO Ontology Release Tool, http://code.google.com/p/owltools/wiki/Oort
8. The Protégé Ontology Editor and Knowledge Acquisition System, http://protege.stanford.edu
9. Antezana, E., Egaña, M., Baets, B.D., Kuiper, M., Mironov, V.: ONTO-PERL: An API for supporting the development and analysis of bio-ontologies. Bioinformatics 24(6) (2008)
10. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene Ontology: Tool for the unification of biology. Nature Genetics 25, 25–29 (2000)
11. Beisswanger, E., Lee, V., jae Kim, J., Rebholz-Schuhmann, D., Splendiani, A., Dameron, O., Schulz, S., Hahn, U.: Gene Regulation Ontology (GRO): Design Principles and Use Cases. In: MIE. pp. 9–14 (2008)
12. Coulet, A., Smail-Tabbone, M., Benlian, P., Napoli, A., Devignes, M.D.: SNP-Ontology for semantic integration of genomic variation data (2006)
13. Day-Richter, J., Harris, M.A., Haendel, M., Lewis, S.: Obo-edit–an ontology editor for biologists. Bioinformatics 23, 2198–2200 (August 2007)
14. Golbreich, C., Horridge, M., Horrocks, I., Motik, B., Shearer, R.: OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences. In: ISWC/ASWC (2007)
15. Golbreich, C., Horrocks, I.: The OBO to OWL mapping, GO to OWL 1.1! In: OWLED (2007)
16. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition 5, 199–220 (June 1993)
17. Haarslev, V., Möller, R.: Racer system description. In: IJCAR. pp. 701–706 (2001)
18. Hoehndorf, R., Batchelor, C., Bittner, T., Dumontier, M., Eilbeck, K., Knight, R., Mungall, C.J., Richardson, J.S., Stombaugh, J., Westhof, E., Zirbel, C.L., Leontis, N.B.: The RNA Ontology (RNAO): An ontology for integrating RNA sequence and structure data. Applied Ontology 6(1), 53–89 (2011)
19. Hoehndorf, R., Oellrich, A., Dumontier, M., Kelso, J., Rebholz-Schuhmann, D., Herre, H.: Relations as patterns: bridging the gap between OBO and OWL. BMC Bioinf. 11, 441 (2010)
20. Horridge, M., Bechhofer, S.: The OWL API: A Java API for Working with OWL 2 Ontologies. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2008)
21. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language, W3C Recommendation, http://www.w3.org/TR/owl-features/
22. Moreira, D.A., Musen, M.A.: OBO to OWL: a protégé OWL tab to read/save OBO ontologies. Bioinformatics 23(14), 1868–1870 (2007)
23. Mungall, C., Emmert, D.B.: A Chado case study: an ontology-based modular schema for representing genome-associated biological information. In: Bioinformatics. pp. 337–346 (2007)
24. Mungall, C., Ireland, A.: The OBO Flat File Format Guide, version 1.4, http://www.geneontology.org/GO.format.obo-1_4.shtml
25. Tirmizi, S.H., Aitken, S., Moreira, D.A., Mungall, C., Sequeda, J., Shah, N.H., Miranker, D.P.: Mapping between the OBO and OWL ontology languages. J Biomed Semantics 2 Suppl 1, S3 (2011)
26. Topalis, P., Mitraka, E., Bujila, I., Deligianni, E., Dialynas, E., Siden-Kiamos, I., Troye-Blomberg, M., Louis, C.: IDOMAL: an ontology for malaria. Malar Journal 9 (2010)
27. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: In Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006. pp. 292–297 (2006)