A method for semi-automatic extension of a middle-layer ontology

Ulf Schwarz [1]*, Holger Stenzhorn [2], Nikolina Koleva [1], Luc Schneider [1] and Emilio M. Sanfilippo [1]

[1]Institute for Formal Ontology and Medical Information Science, Saarland University, Germany
[2]Saarland University Hospital, Dep. for Pediatric Oncology and Hematology, Homburg, Germany

ABSTRACT

We present a semi-automatic method for the integration of semantic concepts under a middle-layer ontology in the biomedical domain. For specific purposes a users might want to extend this ontology with concepts or classes necessary for a task at hand and so the middle-layer ontology must be specialized in simple dedicated modules. Our strategy for performing the extension is first to search candidate concepts in existing sources. The retrieved candidate concepts are then ranked. The ranking is realized on the one hand on the ontology level and on the other hand on the concept level. We look for a class from the middle-layer ontology that matches a super class of a candidate concept. If there is such match we generate recommendation for the integration of the candidate concept. We subscribe to the MIREOT[1] principles and apply them in our strategy.

We developed an Ontology Aggregator Tool (OAT) that implements our strategy. The tool allows for the re-use of (parts) pre-existing resources and enables a user to build a custom made semantic resource.

1 INTRODUCTION

Users and designers of biomedical ontologies are currently dealing with a proliferation of heterogeneous semantic resources. The plethora of ontologies contained in libraries such as NCBO's BioPortal (Musen *et al.* (2012), Whetzel *et al.* (2011)) illustrate this issue, with more than five millions terms gathered and grouped in more than three hundred often overlapping, yet mostly unrelated resources.

A natural question that arises at this point is how are those heterogenous classes related. Our attempt for answering this question is the development of Health Data Ontology Trunk[2] (HDOT). HDOT integrates parts of semantic resources for a larger domain (in our case: the overall biomedical domain) under a middle-layer ontology. It has available a set of separate extendable modules. The modules represent distinct parts of the envisaged domain in a way that optimally equilibrates expressivity and scalability. At the same time it ensures ontological consistency in the process of extending the umbrella toward more specialized classes. Since HDOT is a middle-layer ontology it does not contain specific concepts. However, this might be of interest to its user and at this point the OAT comes into play. OAT is used for the semi-automatic extension of HDOT. Thereby parts of pre-existing semantic resources are re-used. We aim at integrating not only previously established and standardized terms but also well established class identifiers, i.e. URIs, whenever possible. The OAT is implemented in Java and thus is platform independent tool.

*To whom correspondence should be addressed: Ulf Schwarz ulf.schwarz@ifomis.uni-saarland.de

[1] http://obi-ontology.org/page/MIREOT

[2] https://code.google.com/p/hdot/

2 THE ALGORITHM

2.1 Sorting the resources

There are many portals and a huge amount of biomedical ontologies. For the first phase of the development, we decide to use NCBO BioPortal for searching an appropriate candidate. We specify a list of *prima facie* suitable ontologies to be the source of classes proposed for the integration under HDOT. In order to be able to consider new ontologies that could be included in the BioPortal repository, we define criteria for sorting all available resources in BioPortal. Consequently, the search results are sorted with respect to the semantic resource they are retrieved from. In addition, we apply eight more criteria for comparison on the ontology level, namely:

1. contained in pre-defined list of acceptable ontologies;
2. contained in OBO Foundry[3];
3. date release after 2008-12-31;
4. the resource is not flat;
5. the resource is not only metadata;
6. author of classes is specified;
7. classes are documented;
8. depth of the hierarchy;
9. no classes with one subclass.

This is a way to ensure that the quality of HDOT will not be diluted. In further stages of the development we would like to add more criteria to refine the sorting of the resources.

2.2 Finding candidate-classes for integration

As a backbone of our tool we use OntoCat (Adamusiak *et al.* (2011)). OntoCat provides a programming interface with high level of abstraction. The project can be used to query public ontology repositories via REST web services.

As soon as OAT is called the search in BioPortal starts. The search primarily, but not exclusively, uses the term given by the user and delivers a list with hits. By **hit** we mean here a search result, i.e. a class, whose label or URI matches the searched term. The received results are then restricted with respect to the similarity[4] of the query and the searched term.

2.3 Rootpath extraction

The next step is the root path(s) extraction, i.e. the path from the found class to the root class in a given ontology. We extract the root path(s) of the n-best hits contained in the restricted list of

[3] http://www.obofoundry.org/

[4] We experimented with different thresholds for the similarity and observed that 90% gives good filtering. The similarity measure used is the Levenshtein distance.

results. Currently we consider the five best hits. For the root path(s) extraction we use the BioPortal's SPARQL endpoint (Salvadores *et al.* (2012)) because the REST services often threw server errors. There is no direct way for getting the entire path with a single SPARQL query, as it was offered by the REST services of BioPortal. Thus the path from the matched class to its root is collected iteratively by querying for the direct parent and its label.
According to our observations, the reliability of the SPARQL is better than the one of the REST services. SPARQL directly queries an ontology and do not access database entries that might not be updated. The paths are then further processed when generating a recommendation for the integration of a found hit.

2.4 Recommendation generation

The core component of the OAT is the `Recommendation Generator` (cf. Algorithm 1). In order to generate a recommendation that suggests which of the hits to be integrated under HDOT, a class in HDOT that matches a parent class of the hit is needed. The core of HDOT is never modified and thus when looking for a match we consider certain conditions, e.g. whether the current HDOT-module is in the core of HDOT and if so whether the matched class is a leaf node. We compare the URIs and the labels of both classes. In case the labels match but the URIs do not, this is an indication that the meaning of the matched terms could be different. So we generate a new HDOT URI but integrate the label of the class[5]. If a match is detected a recommendation is generated.

Algorithm 1 *Recommendation generator*

```
for hit, path in list_of_hits_with_paths do
    for parent in path do
        for hdot_module in hdot_modules do
            if FINDMATCH(hdot_module, parent) then
                generate_recommendation

function FINDMATCH(hdot_module, parent)
    match_found = false
    for class in hdot_module do
        if URIs_match(class, parent) then
            match_found = true
        if labels_match(class, parent) then
            match_found = true
        if extension conditions are not satisfied then
            match_found = false
    return match_found
```

Possibly there may be more than one recommendations among the five best candidates. Therefore we sort the recommendations according to the following criteria:

1. hit source is in predefined list of ontologies;
2. hit has definition;
3. the matched class is deeper in the HDOT hierarchy;
4. which parent of the hit matched (lower number preferred);

[5] There is a dedicated `URI manger` for this task which allows for further analysis so that we can avoid the unnecessary generation of new URIs. We generate new URIs since we do not trust all available sources.

The sorting gives a list with ranked recommendations. If no recommendation has been generated among the five best candidates then we check if a recommendation out of the next five candidates can be generated.

2.5 Generation of new HDOT-module

In order not to damage the quality of HDOT, the integration is not performed fully automatically. The user is asked to decide if the generated recommendation is suitable or not.
In the simple case of immediate acceptance, a subsequent question is asked, namely if the user wants to integrate the subclasses of the recommended class if there are any[6]. Then a new HDOT-module is created with the OWL API (Horridge and Bechhofer (2011)). The new HDOT-module contains the recommended class and if applicable the subclasses and so the task of the OAT is fulfilled.
The other situation is that the user disagrees with the recommendation and then another class should be recommended. If there are other recommendations in the ranked list, we present the next one to the user. Otherwise we try with the next five best hits. We go at most to the best ten hits since we believe that the candidates sorted further down would not fulfil the quality criteria. In case that the user rejects all generated recommendations or no suitable recommendation is found a curator is appealed to integrate the missing class.

3 CONCLUSION

We present a strategy for semi-automatic extension of the middle-layer ontology HDOT and its implementation with the OAT. The OAT has a generic design, so that it can be applied to repositories of semantic resources other than NCBO BioPortal, e.g. EBI's Ontology Lookup Service[7] or OntoBee[8]. The suitability of the generated recommendations are crucial for the maintenance of the good quality of HDOT. We observed that the generated recommendations for some example terms were appropriate. Thus, we are optimistic that our strategy can be successfully used in practice. However, we do not allow fully automatic integration and therefore ask the user confirm the integration of a recommended class and potentially its sub classes.

REFERENCES

Adamusiak, T., Burdett, T., Kurbatova, N., van der Velde, K. J., Abeygunawardena, N., Antonakaki, D., Kapushesky, M., Parkinson, H., and Swertz, M. A. (2011). Ontocat - simple ontology search and integration in java, r and rest/javascript. *BMC Bioinformatics*, **12**(1), 218.

Horridge, M. and Bechhofer, S. (2011). The owl api: A java api for owl ontologies. *Semant. web*, **2**(1), 11–21.

Musen, M. A., Noy, N. F., Shah, N. H., Whetzel, P. L., Chute, C. G., Storey, M.-A. D., and Smith, B. (2012). The national center for biomedical ontology. *JAMIA*, **19**(2), 190–195.

Salvadores, M., Horridge, M., Alexander, P. R., Fergerson, R. W., Musen, M. A., and Noy, N. F. (2012). Using sparql to query bioportal ontologies and metadata. In *Proceedings of the 11th international conference on The Semantic Web - Volume Part II*, ISWC'12, pages 180–195, Berlin, Heidelberg. Springer-Verlag.

Whetzel, P. L., Noy, N. F., Shah, N. H., Alexander, P. R., Nyulas, C., Tudorache, T., and Musen, M. A. (2011). Bioportal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic Acids Research*, **39**(Web-Server-Issue), 541–545.

[6] This question is only asked if there are subclasses in order not to make the system too interactive

[7] http://www.ebi.ac.uk/ontology-lookup/

[8] http://www.ontobee.org