JOT: a Scripting Environment for Creating and Managing Ontologies

Olivier Dameron[1,2]
[1] SMI, Stanford University, USA
[2] INRIA, France
dameron@smi.stanford.edu

Abstract

Maintaining ontologies involves performing some generic repetitive tasks. Moreover, it requires to take semantic consistency into account. None of these points have been addressed by the existing ontology editors. We propose a Python-based scripting environment for Protégé. This extension is compatible with both the classical frame approach and with the OWL plug-in. It allows the user to define and to reuse some macros in order to make the curation of an ontology easier and less error-prone. It can also be used for consistency checking and for performing automatic corrections. This approach is a step toward the identification of some generic modeling principles that are independent of the domain represented and of the modeling language.

1 Introduction

An ontology is a representation of a shared conceptualization of a domain. Protégé[1] is a tool for building and editing ontologies [1]. Typically, it allows the user to create concepts, relationships between concepts, and to organize them in subsumption hierarchies [2].

Protégé is organized according to the Frame paradigm. Description logics are another approach based on fragments of first order logics with an explicitly defined semantics [3]. Reasoning with description logics has interesting computational properties. In particular, the Web Ontology Language OWL[2] is recognized as a standard in the semantic web context [4]. An OWL plug-in[3] for Protégé has been developed to add description logics facilities to Protégé [5].

Problems When creating or maintaining an ontology, a possible source of problems comes from the fact that the concepts are usually related to several other concepts. In addition, there are often some semantic dependencies between concepts or relationships. For example, the `Heart` has a wall composed of three layers: `Epicardium`, `Myocardium` and `Endocardium`. Decomposing the heart into the left and right atrium and ventricle requires the creation of concepts such as `Wall of left atrium` or `Epicardium of the right ventricle`. The atria and the ventricles are said to be regional parts of the heart because their delimitations are somehow arbitrary and do not rely on any structural feature. The latter concept has to be both a regional part of the `Epicardium`, and a layer of the `Wall of right ventricle`. These two relationships are consequences of the right ventricle being a regional part of the heart, and the `Epicardium` being a layer of `Wall of heart`. Such examples of semantic consistency have been developed in previous works [6, 7].

The creation of new concepts can be tedious and difficult because it is repetitive and all of the semantic dependencies to be met. For example, lateralized concepts such as `Lung` require the creation of left and right subconcepts such as `LeftLung` and `RightLung`. In addition, in OWL the user may need to provide definitions for the concepts. Thus, a `Lung` is either a `LeftLung` or a `RightLung`; or that `LeftLung` is equivalent to a `Lung` located on the left side. Now, imagine manually creating the concept `Rib` and each of its twelve subconcepts, plus the twenty four lateralized ones; and providing definitions for all of them.

To our knowledge, none of the current conceptual modeling environments handle these dependencies. The OWL plug-in for Protégé proposes a todo mechanism to help the curator keep track of all the necessary tasks. However, it only will assist him in performing these tasks. It will not perform them for the user nor check that the semantic dependencies have been addressed.

[1] `http://protege.stanford.edu`
[2] `http://www.w3.org/TR/owl-ref/`
[3] `http://protege.stanford.edu/plugins/owl/`

Proposed solution We propose a scripting extension to Protégé called JOT. It assists the user by creation or reuse of some macros. These macros can be used to make the maintenance of ontologies easier as well as to take semantic dependencies into account. This console is compatible with the core Protégé system and its various plug-ins, particularly the OWL plug-in.

This article shows how JOT has been successfully used to assist in the creation of an ontology of the thorax anatomy in OWL, based on the Foundational Model of Anatomy [8]. The goal is more to give an overview of what JOT can do rather than to concentrate on programming details. The examples and the Python scripts can be downloaded from the JOT tutorial. The results are also applicable to a frame-based ontology and a similar approach can be extended to domains domains other than that of anatomy.

2 The JOT Python scripting console

The JOT[4] scripting console is a Protégé tab containing a Python[5] console. The user can interact directly with the open knowledge base through the `kb` variable. This variable is the python equivalent of an instance of the `KnowledgeBase` class of the Protégé API, or of one of its subclasses (e.g. of `OWLKnowledgeBase` if the project is an OWL ontology). An additional variable `jotTab` representing the JOT tab is provided if the user ever needs to access other tabs such as Prompt which would allow several projects to be open simultaneously.

3 Assistance with ontology maintenance

A typical task like the creation of a lateralized concept involves several steps that can be automated in a macro function. First, create the "abstract" concept and its left and right subconcepts. Second, define the "abstract" concept as the union of its left and right subconcepts (e.g. a lung is either a left or a right one). Third, make the left and right subconcepts disjoints (e.g. a lung cannot be both a left lung and a right lung). Fourth define the left subconcept as equivalent to the intersection of the "abstract" concept and of `left anatomical concept`; repeat for the right subconcept.

Another interesting utilization of Python's features is the enumeration of concepts such as ribs or vertebrae. It involves creating the generic concept (e.g. `Rib`) as lateralized (so that we have `left rib` and `right rib`), and then iteratively creating each of the enumerated subconcepts (e.g. `Rib1` ...`Rib12`) as other lateralized concepts. Additionally, `Rib` is defined as the equivalent to the union of `Rib1` ...`Rib12` so that any rib has to be one of the twelve ribs. Eventually, all of the enumerated concepts are stated as disjoint so that a rib cannot be both a `Rib3` and a `Rib4`. This approach can be adapted to describe the muscles holding the ribs together and those associating the ribs with the vertebrae.

By using such functions, the user is closer to a high-level description language ("there are twelve ribs and they are lateralized"), and is less likely to forget some constraint or definition during concept creation.

4 Assistance with consistency checking

Consistency checking consists of making sure that semantic dependencies are still met after having modified an ontology.

Decomposing an organ like the heart into regional parts like the atria and the ventricles requires making sure that all structural parts like `Myocardium of left atrium` are themselves regional parts of actual structural parts of the heart like `Left atrium`. Conversely, we also have to make sure that all structural parts of the heart are also parts of at least one of the atria or ventricles.

JOT allows the user to run some predefined functions in order to detect possible inconsistencies. If necessary, the user can also devise a small script to fix the inconsistencies.

5 Lesson learned: capitalizing on macros

With the experience acquired from modeling anatomy in OWL, we realized that some of the macros reuse some higher-level primitives. Therefore, what first can appeared as some convenient *ad hoc* scripts turned out to involve

[4] `http://smi-web.stanford.edu/people/dameron/jot/`
[5] `http://www.python.org`

a more generic description scheme. These primitives emerged as implementation of a modeling approach that is independent of the domain.

For example, the creation of lateralized concepts refers to a function having an argument of a concept and a list of concepts, that makes the concept equivalent to the union of the list elements, and that makes the elements pairwise disjoints. Similarly, the functions used for consistency checking refer to the possibility of retrieving all the concepts that are related to a certain concept by a relationship `R`, be it by an universal ($\forall$) or an existential ($\exists$) restriction.

6 Discussion

Providing a quantitative evaluation of the JOT tab is rather difficult: should we consider the number of concepts and relationships generated by the scripts, or also take into account the time saved? How do we assess the number of mistakes that could be avoided?

Having a scripting environment for interacting with an ontology allows the user to perform some use-once on-the-fly process. As they consist in calls to the Protégé API, these treatments could have been implemented as Java classes. However, this would involve a larger overhead and would require restarting Protégé every time new interactions have to be added. Such a solution would be particularly inefficient for tasks that only need to be performed once.

Some of the functions used for creating concepts and relationships or for enforcing consistency require some reasoning capabilities. For example, in order to retrieve all the concepts related to a certain concept by a relationship `R`, we may also need to consider the subrelationships of `R`. This point is not yet taken into account. It is important to make clear that in a description logic context, JOT is not a substitute to reasoners such as Racer, but a complement.

Future work will consist of identifying one or several sets of reusable functions. Among them, the domain-independent functions of particular interest. We would also like to asses to what extent such functions are language-independent (i.e. provide uniform resources for OWL, OIL...) and perhaps modeling paradigm-independent (i.e. are applicable to frames and description logics).

References

[1] J.H. Gennari, M.A. Musen, R.W. Fergerson, Grosso W.E., Crubézy M., H Eriksson, N.F. Noy, and S.D. Tu. The evolution of Protégé-2000: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[2] N.F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Medical Informatics, 2001.

[3] D. Nardi, R.J. Brachman, and Baader F. *et al. The Description Logics Handbook : Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[5] H. Knublauch, O. Dameron, and M. Musen. Weaving the biomedical semantic web with the protege owl plugin. In *First International Workshop on Formal Biomedical Knowledge Representation KRMED04*, 2004. In Press.

[6] O. Dameron, A. Burgun, X. Morandi, and B. Gibaud. Modelling dependencies between relations to insure consistency of a cerebral cortex anatomy knowledge base. In *MIE'03 Proceedings*, pages 403–408, 2003.

[7] O. Dameron, B. Gibaud, and M. Musen. Using semantic dependencies for consistency management of an ontology of brain-cortex anatomy. In *First International Workshop on Formal Biomedical Knowledge Representation KRMED04*, 2004. In Press.

[8] C. Rosse and J.L.V Mejino. A reference ontology for bioinformatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36:478–500, 2003.