

# JOT: a Scripting Environment for Creating and Managing Ontologies

Olivier Dameron<sup>1,2</sup>

<sup>1</sup> SMI, Stanford University, USA

<sup>2</sup> 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, as well as with the OWL plugin. 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 towards 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<sup>1</sup> is a tool for building and editing ontologies [1]. Typically, it allows the user to create concepts and relationships between them, 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 and allowing reasoning with interesting computational properties [3]. Particularly, the Web Ontology Language OWL<sup>2</sup> is recognized as a standard in a semantic web context [4]. An OWL plugin<sup>3</sup> for Protégé has been developed to add description logics facilities to Protégé [5].

**Problems** When creating an ontology or maintaining one, a possible source of problems comes from the fact that the concepts are usually related to several

---

<sup>1</sup><http://protege.stanford.edu>

<sup>2</sup><http://www.w3.org/TR/owl-ref/>

<sup>3</sup><http://protege.stanford.edu/plugins/owl/>

other concepts. Moreover, there are often some semantic dependencies between concepts or relationships. For instance, the **Heart** has a wall composed of three layers: **Epicardium**, **Myocardium** and **Endocardium**. Decomposing the heart in left and right atrium and ventricle requires to create 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 delimitation is somehow arbitrary and do not rely on any structural feature. The latter has to be both a regional part of the **Epicardium**, and a layer of the **Wall of right ventricle**. These two relationships are consequences from 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 it requires to meet all the semantic dependencies. For example, lateralized concepts such as **Lung** require to create the **LeftLung** and **RightLung** subconcepts. In addition, in OWL, the user may need to provide definitions for the concepts, namely that a **Lung** is either a **LeftLung** or a **RightLung**; or that **LeftLung** is equivalent to a **Lung** located on the left side. Now, imagine creating manually 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 plugin for Protégé proposes a todo mechanism to help the curator keeping track of all the necessary tasks, but that will assist him in performing these tasks nor in checking that the semantic dependencies have been addressed.

**Solution proposed** We propose a scripting extension to Protégé called JOT. It assists the user by allowing him to create or reuse 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 as well as with its various plugins, and particularly with the OWL plugin.

This article shows how JOT have been successfully used to assist the creation of an ontology of the thorax anatomy in OWL, based on the Foundational Model of Anatomy [8]. The examples and the Python scripts can be downloaded from the JOT tutorial. The results are also applicable to a frame-based ontology. Moreover, a similar approach can be extended to other domains than anatomy.

## 2 The JOT Python scripting console

The JOT<sup>4</sup> scripting console is a Protégé tab containing a Python<sup>5</sup> console. JOT relies on SPyConsole<sup>6</sup>, which is a Java implementation of a Python con-

<sup>4</sup><http://smi-web.stanford.edu/people/dameron/jot/>

<sup>5</sup><http://www.python.org>

<sup>6</sup><http://www.uvm.edu/giee/SME3/ftp/JConsole/>

sole. SPyConsole itself relies on Jython<sup>7</sup>, which is a Java implementation of Python. The user can interact directly with the open knowledge base through the `kb` variable. This variable is the python equivalent of the 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 ever the user needs to access other tabs such as Prompt that would allow several projects to be open simultaneously.

### 3 Assisting the ontology maintenance

A typical task like the creation of a lateralised concept involves several steps that can be automatised in a macro function. First, create the "abstract" concept and its left and right subconcepts. Second, define the "abstract" concept as the reunion 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 disjoint (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 utilisation of the Python's features is for enumerated concepts such as the ribs or the vertebrae. It involves creating the generic concept (e.g. `Rib`) as a lateralized one (so that we have `left rib` and `right rib`), and then iteratively creating each of the enumerated subconcepts (e.g. `Rib1` ... `Rib12`) as other lateralised concepts. Moreover, `Rib` is defined as equivalent to the union of `Rib1` ... `Rib12` so that any rib has to be one of the twelve ribs. Eventually, all 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.

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 the creation of the concepts.

### 4 Assisting consistency checking

Consistency checking consists in making sure that the semantic dependencies are still respected after modifications of an ontology.

The decomposition of the heart into regional parts like the atria and the ventricles is somehow conventional and does not rely on any physiological feature. Therefore, we have to make sure that all their 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 the

---

<sup>7</sup><http://www.jython.org>

structural parts of the heart are also parts of at least one of the atria or of the ventricles.

## 5 Lesson learned: Capitalizing macros

With the experience acquired from modeling anatomy in OWL, we realized that some of the macros reuse some high-level primitives. Therefore, what appeared at first sight as some convenient *ad hoc* scripts turn out to involve some more generic description schemes. 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 for argument 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 disjoint. 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 a universal ( $\forall$ ) or an existential ( $\exists$ ) restriction.

Note that the previous task requires some reasoning capabilities, as subrelationships of  $R$  have to be considered. Therefore, in parallel to continuing working on the identification of the "high level", studying their combination with classifiers may be interesting. Eventually, after having identified some primitives, we could try to study to what extent they are not only domain-independent, but also language-independent (OIL or OWL), or even paradigm-independent (frame or DL).

## 6 Discussion

Explain the advantage comparing to java : perform on the fly programatically defined actions without having to compile a java class and to restart protege.

Compare to reasoners.

Compare to rule formalisms such as ruleml...

Hard to provide a quantitative evaluation of the usefulness of the tab: what should be considered: the number of concepts and relationships generated by scripts ? the time saved ? it would require to build in parallel an ontology with and without, but maintaining a knowledge base is also some kind debugging.

This emerging set of functions are 'abstract' modeling primitive, independent of any domain being modeled. Moreover, it is possible that they are also independent (at least partially) of any particular language such as OIL and OWL; or even of any modeling paradigm such as frame or 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 en-

- vironment 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, F. Baader, W. Nutt, F. M. Donini, U. Sattler, D. Calvanese, R. Mitor, G. De Giacomo, R. Ksters, F. Wolter, D. L. McGuinness, P. F. Patel-Schneider, R. Mller, V. Haarslev, I. Horrocks, A. Borgida, C. Welty, A. Rector, E. Franconi, M. Lenzerini, and R. Rosati. *The Description Logics Handbook : Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [4] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. From *SHIQ* and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 2003. In press.
- [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.