Highly Literate Ontologies

Phillip Lord* and Jennifer D. Warrender

School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK

ABSTRACT

There is still a lot of discussion about exactly what ontologies should represent, but what is generally agreed is that they formalise and relate to some relatively complex areas of knowledge. While ontology environments allow rich descriptions of the relationship between the entities inside the ontology (because this is what an ontology is), they often do not provide the same rich environment to describe the knowledge that they represent.

OWL does, for instance, supports annotations which allows an ontology developer to add comments to many parts of the ontology. But these comments, do not contain markup, sectioning or any of the standard facilities authors use when writing documents.

Our solution to this builds on Tawny-OWL, our highly-programmatic environment for ontology development. This provides a rich environment, which allows abstraction, automation and extension, while still being entirely textual. As a result, it is possible to integrate this form of ontology with similar textual environments for documentation such as LaTeX, or AsciiDoc. We call the result a literate ontology, in reference to literate programming. The result can be "tangled" to produce either a document or ontology.

However, manipulating mixed syntax formats is difficult. Generally, the text editor either supports the literate form or programmatic (ontology) form best. To address this, we have developed what we call "lenticular views" – essentially, the source code can be presented either in an ontology-centric or a document-centric view. Either form can be changed, giving the author a powerful and unique environment for creating literate ontologies. Or alternatively, semantic documents where the ontology formalises the document. We demonstrate this with our literate amino-acid ontology which is also a part of the developing manual for Tawny-OWL.

1 INTRODUCTION

Ontologies have been used extensively to describe many parts of biology. Ontologies have two key features which make their usage attractive. First, they provide a mechanism for standardising and sharing the terms used in descriptions, making comparison easier and, secondly, they provide a computationally amenable semantics to these descriptions, making it possible to draw conclusions about the relationships between descriptions even when they share no terms in common.

Despite these advantages, the oldest and most common form of description in biology is free text, or a semi-structured representation through the use of a standardised fill-in form. Free text has numerous advantages compared to ontologies: it is richly expressive, is widely supported by tooling, and while the form of language used in science ("Bad English" (Wood *et al.*, 2001)) may not be easy to use, understand or learn, it is widely taught and most scientists are familiar with it.

*To whom correspondence should be addressed: phillip.lord@newcastle.ac.uk

The two forms of description have largely been used independently. Ontology terms are sometimes used in semi-structured formats such as a UniProt record, or minimum information documents. While these use ontologies in some parts of the document, in general, ontology terms and the free text are in different parts of the record. In this paper, we show how can we integrate ontological and textual knowledge in a single authoring environment and describe how we are applying this to describing amino acids.

2 DEVELOPING KNOWLEDGE

First, we ask the question, why is it difficult to relate ontological and textual descriptions. One possible explanation is that the two forms have very different "development environments". The main documentation environment used within science is Word, followed by LaTeX, common in more mathematical environments. More recently, there has also been interest in various light-weight markup languages, such as markdown, and their associated tool-chains.

Ontology development environments also come in many different forms. Early versions of the Gene Ontology, for instance, used a bespoke text file format and a text editor – an approach rather similar to the light-weight markup languages of today. This had the significant advantage of a low-technological barrier to entry. More modern environments provide a much more graphical interface. These generally provide a much richer way of interacting with an ontology.

While these environments add a lot of value, they do not necessarily integrate well with text. Both Protégé and OBO-Edit have a class-centric view and are biased toward showing the various logical entities in the ontology, as opposed to the textual aspects. Indeed, this bias is shown even at the level of OWL. For example, annotations on an entity (or rather an axiom) are a *set* rather than a list, while ordering is generally considered to be essential for most documents.

With this divergence of development environments, it seems hard to understand how we could square the circle of combining text and ontology development. Next, we describe the amino-acid ontology and how the novel development methodology we used for this ontology allows us to achieve this.

3 TAWNY-OWL

Tawny-OWL (Lord, 2013) provides a fully programmatic environment for development. Simple ontological statements can be written with a syntax inspired by Manchester OWL notation (Horridge and Patel-Schneider, 2012); repetitive statements can be built automatically by writing functions which encapsulate and abstract over the simpler statements, a process we call "patternisation" (Warrender and Lord, 2013).

In this way, we have managed to combine the advantages of text-based environments for editing ontologies i.e. the use of a standard

```
First, to explain the domain. Proteins are polymers
made up from amino-acid monomers. They consist of a
central carbon atom, attached to a carboxyl group (the
``acid'' amino) and amine group (the ``amino'' group)
a hydrogen and an R group. The R group defines the
different amino acids. The different R groups have
different phyiscal or chemical properties, such as
their degree of hydrophobicity. We call these different
characteristics |RefiningFeatures|.

\begin{tawny}
(defclass AminoAcid)

(defclass RefiningFeature)
(defclass PhysicoChemicalProperty :super RefiningFeature)
\end{tawny}
```

Fig. 1. The document-centric view

```
;; First, to explain the domain. Proteins are polymers
;; made up from amino-acid monomers. They consist of a
;; central carbon atom, attached to a carboxyl group (the
;; ``acid'' amino) and amine group (the ``amino'' group)
;; a hydrogen and an R group. The R group defines the
;; different amino acids. The different R groups have
;; different phyiscal or chemical properties, such as
;; their degree of hydrophobicity. We call these different
;; characteristics |RefiningFeatures|.

;; \begin{tawny}
(defclass AminoAcid)

(defclass RefiningFeature)
(defclass PhysicoChemicalProperty :super RefiningFeature)
;; \end{tawny}
```

Fig. 2. The ontology-centric view

editing environment and integration with version control, while maintaining (and in some ways surpassing) the power of tools like Protégé.

Tawny-OWL can be used to generate any ontology, but we demonstrate it here with the amino-acid ontology: a highly patternised ontology with over 430 classes generated from one pattern. We consider next the implications that this has for the ability to integrate ontological and textual descriptions.

4 LITERATE ONTOLOGY

As Tawny-OWL is based on a full programming language, it supports a feature which at first seems quite inconsequential: comments. As with almost every programming language, it is possible to add free, unstructured text to the same source code that defines the ontology. While opinions vary on the role of comments in programmatic code, perhaps the most extreme is that of literate programming (Knuth, 1984) which suggests that code should be usable both as a program capable of execution and as a document capable of reading and that neither view should have primacy.

Literate programming can be difficult, however, partly because the editing environment offers few facilities for it: fundamentally, supporting mixed-syntax text in a tool is a difficult task. Our solution uses a multi-view approach to editing, which allows the author to see her source code in either a *document-centric* or an *ontology-centric* view. We call this approach *lenticular* text, named after lenticular printing which produces images which change depending on your angle of viewing. This is an entirely novel solution to literate programming as it effectively performs the tangling operation for the author as they type. A representation of the two views are shown in Figures 2 and 4. The two views, it should be noted, contain the same **text** but are syntactically different, such that the document-centric view is entirely valid LaTeX code, while the ontology-centric view is valid Tawny-OWL code.

We have now implemented lenticular text for the editor, Emacs[1], in a package called "lentic"[2]. A key feature of this implementation is that both views exist simultaneously in Emacs, and provide all the features of the appropriate development environment; for example, "tab-completion" works in both the document-centric view (completing LaTeX macros) and in the ontology-centric view (completing ontology identifiers). We can launch a compilation of the document-centric view (producing a PDF), or evaluate our ontology, perhaps reasoning over it, in the code-centric view. Therefore, we have achieved a key aim of literate programming: neither view holds primacy and the author can edit either.

5 DISCUSSION

In this paper, we have described our methodology for integration of text and ontological statements at authoring time, using lenticular text to enable literate ontology development. Indeed, we have fully documented the whole of the amino-acid ontology into literate form[3].

The combination of Tawny-OWL and lenticular text is an extremely rich environment. We are aware, however, that it is a specialist environment. To make full use of Tawny-OWL, the author needs to use a Clojure based-development environment, document authoring in LaTeX, and the lentic package which is Emacs-based. In reality, though, the tools are not tightly coupled: we have alternatives beyond LaTeX, Emacs, or even Tawny-OWL. At the same time, one output form of a literate ontology is a readable PDF document, something far more familiar to biologists or medics than Protégé or any ontology development environment.

ACKNOWLEDGEMENTS

This work was supported by Newcastle University.

REFERENCES

Horridge, M. and Patel-Schneider, P. F. (2012). Owl 2 web ontology language manchester syntax (second edition). Technical report.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, **27**, 97–111.

Lord, P. (2013). The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. *OWLED 2013*.

Warrender, J. D. and Lord, P. (2013). A pattern-driven approach to biomedical ontology engineering. *SWAT4LS 2013*.

Wood, A., Flowerdew, J., and Peacock, M. (2001). International scientific english: The language of research scientists around the world. *Research Perspectives on English for Academic Purposes*, pages 71–83.

[1] https://www.gnu.org/software/emacs/

[2] https://github.com/phillord/lentic

[3] https://github.com/phillord/tawny-tutorial