# Literate, Active OWL Ontologies

Bijan Parsia

University of Manchester

**Abstract.** OWL ontologies are complex computational artifacts that are intimately connected with conceptual information and with application issues that are not easily explicable in the context of an OWL document. In this paper, drawing inspiration from literate programming and active essays, I propose a new form of narratively oriented, interactive OWL document. The basic technique has been applied to the draft version of the OWL 2 primer.

## 1 Introduction

With the rise of standardized languages intended for expressing formalizations of ontologies, the size and complexity of ontologies, both in house and publicly available, has risen dramatically. The has also been (yet another) shift in meaning of the term 'ontology' to also refer to a particular expression in a particular language: that is to a computational, rather than conceptual, artifact. The rising sense of the term places ontologies as siblings to programs, databases (and database schemes), and UML diagrams instead of conceptual models, software patterns and architectures, and algorithms.[1]

The tool and methodology infrastructure surrounding modern ontology languages (like OWL) reflect this as does the design of the languages themselves. Development environments are modeled on programming language IDEs as are new services and techniques (e.g., debugging, diffing, unit testing). This inspiration has proven quite fruitful and is likely to continue.

However, a downside of this trend is that it has become more difficult (and perhaps less common) to engage and present ontologies at a higher level. Since popular, practical ontology languages are expressively limited (in order to facilitate automated reasoning) it is not always the case that the intent of the author is clearly or correctly reflected in the expression of that intent.[2] That is,

[1] This analogy is not precise as these categories overlap in multiple ways. Also, there are systematic, if sporadic, attempts to blur them further. The spirit of the distinction is computational concreteness: if we consider the distinction between programs (which must include sufficient detail to allow execution) and algorithms (which can be more abstract), we are not far off from the distinction between the newer and the older senses of 'ontology'.

[2] This problem is exacerbated by the fact that reasoner implementations are not robustly efficient across the whole language they support. For example, inverse roles in OWL ontologies are known to cause trouble for current tableau reasoners, though recent optimization advances[hermit] promise to change that.

OWL ontologies may be, at best, *approximations* of a conceptualization. When we have another formalization to directly approximate (as with the DOLCE ontologies) that situation may not be so severe. But sometime all we have is a natural language based conceptualization.

Similarly, there are many ways to present an ontology, and for many purposes. An IDE (or standard serialization) typically forces a single, non-domain sensitive presentation (e.g., a class hierarchy with each level alphabetized). Sometimes, an ontology has a few focal classes; other times, there is a pattern of modeling used throughout; still other times, there are odd bits that need careful explanation or there are obvious alternative ways of modeling which are non-obviously ruled out. In these cases, a clear *narrative* is required to convey important aspects of the ontology. Typically, such narratives are communicated by email, verbally, or by other out of bound means (e.g., documentation).

Building documents with such narratives over a working OWL ontology is comparatively difficult — rife with tedious detail. The canonical form is prose interspersed with OWL axioms (for example, in a tutorial, an article, or a book chapter). The key difficulty is that the OWL fragments, typically, are not complete OWL ontologies thus cannot be checked even for syntactic correctness without cutting and pasting into a wrapper. Nor can highlighted entailments be verified easily, which is particularly important during composition, but it is also frustratingly common when making minor "aesthetic" tweaks.

While authoring narratives over ontologies is difficult, the audience experience is far from ideal. The syntax of the examples is fixed and not always to the taste of each reader. The fragments individually aren't workable ontologies, thus it is difficult to test or simply play with examples. All the tools a reader is used to using are not easily applicable — even if the author supplies, in an appendix, a complete version of an ontology, the *context* of the examples is lost. Thus, the reader must keep the narrative context in mind through a series of large context switches (e.g., from the narrative jump to the appendix, copy and paste to an IDE, find the relevant axioms (which may not be grouped in the IDE as in the narrative thus requiring switching back and forth inside the IDE), and *then* explore the point raised by the narrative).

From both a reading and a writing perspective we have comparatively poor support for strongly narrative presentations of ontologies (which I shall call "narrative ontologies" throughout this paper). In this paper, I suggest a different approach to producing narrative ontologies inspired by Donald Knuth's Literate Programming methodology and tool chain. Furthermore, given the flexibility of electronic documents, I propose that the narrative ontologies we produce should be more interactive in ways partly inspired by Alan Kay's notion of an "active" essay.

## 2 Background

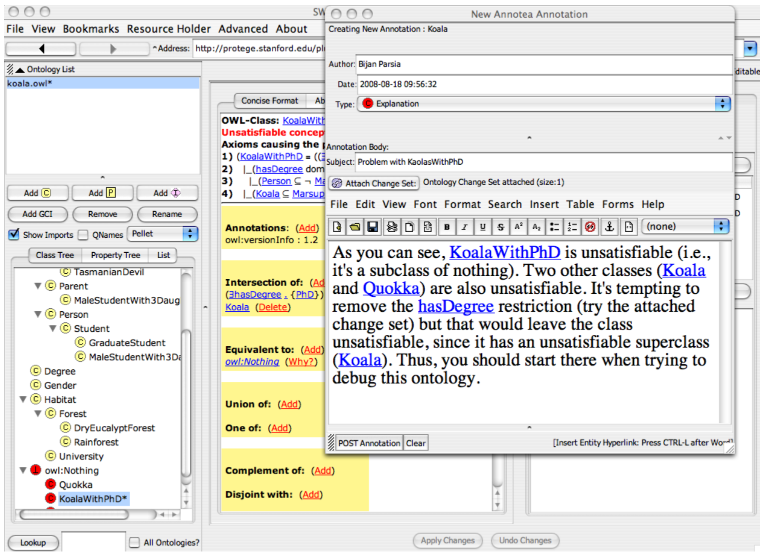
The term "ontology" has undergone considerable shift within the field of knowledge representation even once we put aside the large shift in meaning from the philosophical sense to the computer science sense. The key switch is the notion of an ontology as primarily a computational artifact expressed in a specific ontology language such as OWL whereas before the key meaning was of a *conceptualization*. Classic essays on an ontology of some domain (famously, liquids[1]) may include a formalization of some form (e.g., in first order logic), but generally fitness to application and computational issues took a back seat to fidelity to the domain and conceptual elegance. Typically, these ontologies are not developed in a standard computational notation and were not developed with the benefit of tool support. Sometimes, the formalization is fairly incomplete with axioms appearing almost more as illustrations or elucidations than as the meat of the essay.

OWL has a number of concrete syntaxes in active use, but the stability of a serialization under simple parsing and reserialization is extremely unreliable. In the OWL-S ontologies, for example, though the serialization used was RDF/XML, a great deal of documentation occurred in XML comments which are completely stripped by all RDF parsers as is the order of presentation. OWL-S is illustrative as an example for several reasons: It was intended as a high level conceptualization *and* a useable computational artifact; the canonical presentation of the computational artifact contain a substantive attempt to present explicative narrative that was lost or easily mangled by tools; additionally, there was entirely narrative material concurrently developed which had no direct connection with the computational artifact and had to be manually synched.

There are standard entity annotation techniques (e.g., using `rdf:comment` or `dc:description`) which are heavily used but restricted to documentation of a single version of a single term at a time. They also do not permit grouping of terms or focus on axioms (though OWL2 allows for axiom annotations), nor are tools sensitive to their content. Thus, they are more like tooltips than substantive narrative structuring constructs.

Swoop supports Annotea[**?**] annotations on OWL entities with the body of the annotations being generic HTML. This support has several interesting features: The annotations are out of band thus can be supplied by arbitrary parties; hyperlinks in the body linking to entities in the ontology were live so could be used to control the current Swoop display; and change sets could be attached to an annotation allowing for predefined modifications to the ontology. The last two features, when combined with Swoop's undo and rollback mechanism, work well to provide simple narratives, e.g., of proposed changes or repairs to an ontology (see Figure 1).

Swoop annotations are a limited form of literate active ontologies. Composition of essaylets or even longer essays is done in Swoop in the context of a live ontology. There is a significant authoring short cut — words that terminate a URI for some term in the ontology can be hyperlinked to that term with an accelerator key which makes it very easy and natural to link to terms from all over the ontology. The ontology is active both navigationally and by means of attached change sets which can be used to (speculatively) modify the ontology. Issues include the fact that overall navigation is still Swoop-centered, multiple



Creating New Annotation : Koala

Author: Bijan Parsia
Date: 2008-08-18 09:56:32
Type: Explanation

Annotation Body:
Subject: Problem with KaolasWithPhD
Attach Change Set: Ontology Change Set attached (size:1)

As you can see, KoalaWithPhD is unsatisfiable (i.e., it's a subclass of nothing). Two other classes (Koala and Quokka) are also unsatisfiable. It's tempting to remove the hasDegree restriction (try the attached change set) but that would leave the class unsatisfiable, since it has an unsatisfiable superclass (Koala). Thus, you should start there when trying to debug this ontology.

**Fig. 1.** An example of a Annotea annotation. Clicking on a hyperlink in the annotation body will shift the Swoop interface to display the linked to term, thus a reader can follow directions given in the narrative. Also, a specific change is attached to this annotation which the reader can apply to see the effect, then revert.

distinct changes are not possible in a single annotation, annotations are not themselves effectively linkable, there is no support for presenting axioms, and there is no support for checking specific entailments in the narrative. In the end, Swoop annotations are annotations and work best as an auxiliary to the standard Swoop presentation rather than a narrative driven alternative.

There are several web based, Javadoc-esque systems for presenting OWL ontologies (e.g., OWLDoc[3] or Ldontospec[4]). These have the advantage of being familiar to developers and hosted in a browser. However they do not support interaction and are not narrative based at all. With the development of browser based IDEs such as OWLSight[5] its hard to see the advantages of these forms.

Interactive proof assistants (such as [2] and their associated languages have always supported the interactive development and reading of proofs (hence their name) and have in recent years moved toward presentation modes that are closer to traditional math papers while retaining their interactive capabilities. However, they are focused on complex proofs of specific theorems rather than ontologies *per se*.

[3] presented a radically new form of programming methodology, literate programming. The fundamental point of literate programming is the program

[3] http://www.co-ode.org/downloads/owldoc/
[4] http://code.google.com/p/ldontospec/
[5] http://pellet.owldl.com/owlsight

are meant for communication between people as well as communication between a person and a system. The forms of presentation "best" for people and those best for systems are distinct. Instead of maintaining two separate forms of the program (i.e., the program and its documentation) the author would develop a single artifact, a literate program, that could generate both people oriented documents (i.e., essays) and working programs. The source of a literate program was TEX with special support macros for various programming languages, such as Pascal or C. Authors would work with program fragments which contained indicators for what other fragments they were associated with. Two support programs (WEAVE and TANGLE) could consume this source and generate a working program or a typeset essay, respectively.

Knuth was very optimistic about the benefits of this methodology. He believed that writing programs this way improved the quality of the program (for comparable effort) and that the resulting essays were superior documentation. The TEX system itself was generated from a literate program as was the companion TEXBook. However, while there are enthusiasts, literate programming has not caught on as a general used programming methodology.

Alan Kay[4] champions the notion of an *active essay*[5]. An active essay contains small embedded programs which illustrate key ideas. The embedded programs have two aspects: illustration and experimentation. As illustration they are typically animations of some idea and so make an active essay straightforwardly a multimedia document. Whatever benefits embedded alternative media can bring are thus available in active essays. The innovation is that these programs are supposed to be modifiable by the reader in order to explore the ideas presented by the program. This modification can be more or less canned, i.e., by providing controls which allow the user/reader to modify parameters to the embedded program. Or the modification can be arbitrary which requires a suitably accessible programming language and environment. As with literate programming, active essays are not hugely popular, in part, it is clear, due to the difficulty of producing and consuming them.

What I propose is somewhat less ambitious that active essays or literate programming: The context is restricted to OWL ontologies and I target existing forms of document, seeking to enhance and smooth current practice rather than produce a radical change.

## 3 The Source Language

This section gives a brief, example driven overview of the Litont language.

Unlike most programming languages, OWL already is very liberal, in most serializations, about order of axioms (indeed, in RDF based serialization, even *parts* of axioms may be widely separated), thus the challenge is not to support out of order presentation, but to ensure that the the fragments cohere. In this paper, I consider two host languages: LATEX and HTML by way of MediaWiki syntax. These choices are narrowly pragmatic: They are my current most heavily used authoring environments.

Critically, authors should be able to work in their favored notation. There is no need to force an author to make a choice of notation based on publication target and to work in an unfamiliar notation. For example, it is common when targeting an OWL paper to an AI, KR, or description logic audience to use standard DL notation but when presenting to an OWLED, WWW, or ISWC audience to use functional syntax, RDF/XML, or Turtle. Given the existence of tools such as the OWL API which convert between all these formats, the originating source should be to author's taste.

Fragments of an OWL ontology (considered as a set of axioms) are often missing critical boilerplate from an OWL point of view. For example, consider the following axiom in Turtle syntax:

```
b:C rdfs:subClassOf b:D.
b:C rdf:type owl:Class.
b:D rdf:type owl:Class.
```

This simple axiom requires 4 namespace declarations (for the prefixes `b:`, `rdfs:`, `rdf:`, and `owl:`. Obviously, including these inline is wretched for readability and tedious for authors. (In this case, there would be four lines of illegible boilerplate...more than half the fragment.) In litont, the author may define named ontologies with relevant boilerplate. For example (in MediaWiki syntax):

```
{{OwlOnt
   |label=o1
   |format=turtle
   |template=
     @prefix b: <http://ex.org/anExample>.
     @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
     @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
     @prefix owl: <http://www.w3.org/2002/07/owl#>.
 }}
```

In the case of Turtle, the system knows where to insert axioms (after the boilerplate). Fragments are tied to an declared ontology in the following way:

```
{{OwlAxioms
   |ont=o1
   |label=example1
   |axioms=
       b:C rdfs:subClassOf b:D.
       b:C rdf:type owl:Class.
       b:D rdf:type owl:Class.
 }}
```

This `OwlOnt` construct serves two purposes. First, it collects all the appropriately tagged fragments in the documents in the ontology into a single ontology and inlines that ontology into the document. Second, it is use to syntactically check and transform individual fragments. Thus, an author can write in their preferred notation and render the fragments (and the overall ontology) in another.

`OwlOnt` and `OwlAxioms` comprise the basic functionality of a literate ontology and syntax translation underpins the most basic form of interaction in an active ontology.[6] This is sufficient for a number of cases and sufficiently helpful to be worthwhile. An addition construct to indicate entailments is also helpful:

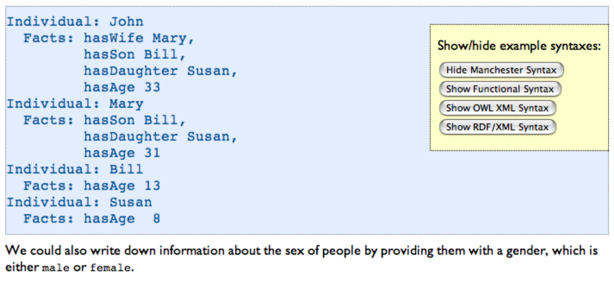
```
{{OwlEntailment
   |from=example1
   |label=entailment1
   |entailed=
       b:C rdfs:subClassOf b:D.
}}
```

(Obviously, this case is trivial.) When the source is processed, each entailment is checked to see if it holds from the specified fragments or from the whole ontology. It is also possible to make the entailment "implicit", that is, not displayed but connected with a stretch of text.

Additional features planned are addition and retraction (with diff display) and thus versions , approximation (i.e., taking an axiom and replacing it with a simpler version), and more fragment and display types.

## 4 The Presentation

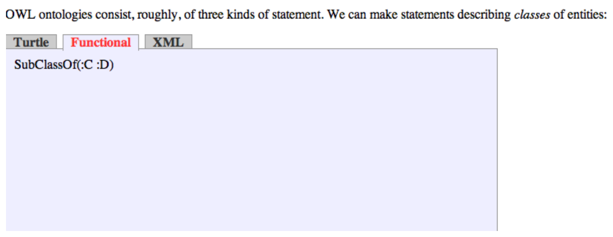
Aside from verifying that the fragments are syntactically correct, converting them to the target syntax, verifying that entailments indeed follow, and collecting fragments into a traditional OWL document, the current system has two basic interaction mechanism: First, the fragment display syntax is configurable, that is, users can select their preferred format for display, or even display more than one for comparison (see figures 2 and 3).



```
Individual: John
  Facts: hasWife Mary,
         hasSon Bill,
         hasDaughter Susan,
         hasAge 33
Individual: Mary
  Facts: hasSon Bill,
         hasDaughter Susan,
         hasAge 31
Individual: Bill
  Facts: hasAge 13
Individual: Susan
  Facts: hasAge  8
```

We could also write down information about the sex of people by providing them with a gender, which is either `male` or `female`.

**Fig. 2.** The OWL2 primer current uses a floating control panel and can display several syntaxes inline at once.


OWL ontologies consist, roughly, of three kinds of statement. We can make statements describing *classes* of entities:

```
SubClassOf(:C :D)
```

**Fig. 3.** An alternative, tab based presentation of multiple syntaxes.

The other mechanism is a copy and paste mode which presents the fragments as syntactically complete ontologies so that the user can test them in alternative tools. This mode can also supply a link which opens the fragment in OWLSight.

Future interaction mechanism include "turing off" axioms and rechecking whether an entailment holds, speculatively adding additional or altered axioms, getting explanations, adding entailments to be checked, and applying systematic transforms.

## 5 Implementation

Currently, the implementation is very hacky. The "tangle" and "weave" scripts are simple regex based preprocessing scripts and are fairly fragile. The hypermedia support is currently quite partial without support for manipulation beyond syntax customization and fragment extraction. These are being developed to support the writing of the OWL 2 Primer. After completion of that I intend to release a robust version of the framework.

Aside from helping with the generation of stand alone documents or ontologies, I anticipate that this general idea will be very helpful for Wiki based ontology development. In particular, support for simple "graphical" axiom modification will make editing an ontology in a Wiki much easier. Furthermore, the ability to target fragments for different ontologies helps distinguish the Wiki *of* the ontology from the ontology itself. An OWL ontology Wiki should be a literate active ontology.

Similarly, I intend to support ontology centric narrative development (as well as narrative centric ontology development). I plan to add Swoop like annotation support to Protégé 4. Crucially, I plan to have support for extracting a narrative from the annotations in an ontology. Thus, one can build post-facto documentation of an ontology starting from a perspective best suited for gathering "notes" about the ontology.

[6] In LaTeX, there are similarly named commands.

## 6 Conclusion

Initial feedback from readers of the syntax switching features of the OWL 2 Primer have been very positive. In addition to allowing people to read the notation they are most comfortable with, it is helpful both for learning new notations and for giving insight into their "home" notation. For example, people familiar with first order logic gain a very clear picture of the semantics of OWL by looking at translations into FOL.

From an authoring perspective, it is a considerable relief to not have to maintain all those syntaxes in parallel. Similarly, it is very nice to be able to load up the document and check that the syntax is correct. Checking such by hand is so tedious that I have tended to avoid it for long periods of time, whereas when the checking is automatic I check on almost every save and certainly on every commit.

Thus far, the main use has been for writing a tutorial, not for developing ontologies from scratch. It is unclear whether there are significant benefits to be had by a literate programming switch in style. I do believe that partial literate active ontologies will be useful as documentation and for facilitating communication about ontologies.

## References

1. Hayes, P.J.: Naive physics I: ontology for liquids. (1990) 484–502
2. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)
3. Knuth, D.E.: Literate programming. The Computer Journal **27**(2) (1984) 97–111
4. Kay, A.: Inventing the future. IEEE Software **15**(2) (1998) 22–24
5. Lincke, J., Hirschfeld, R., Rüger, M., Masuch, M.: Sophiescript - active content in multimedia documents. Creating, Connecting and Collaborating through Computing, 2008. C5 2008. Sixth International Conference on (14-16 Jan. 2008) 21–28