Structured Ontology Format

Rob Shearer

The University of Manchester

rshearer@cs.man.ac.uk

Abstract. This paper presents a simple data model for the representation of OWL ontologies (including the new features of OWL 1.1). The model is built from basic structures native to all common programming environments, so it can be used directly as an API for ontology analysis and manipulation. Furthermore, serialization of these structures using the widely-supported YAML standard yields a readable text format suitable for ontology authoring by average users with text editors and code-management tools.

1 Introduction and Motivation

OWL standardization solves many of the interoperability problems which affected early DL systems; however, OWL's RDF/XML exchange syntax presents two challenges for developers:

1. The syntax is difficult for human users to read and write.
2. Parsing ontologies and working with the resulting ontology data (using some proprietary internal representation) require significant engineering effort.

The first issue has been addressed primarily through development of graphical tools for working with ontologies, such as Protégé [KFNM04] and SWOOP [KPH05]. Such tools make authoring accessible to inexperienced users, but graphical interfaces are forced to presume a particular user workflow and mindset which might not be appropriate in all cases; sophisticated editors include "expert mode" interfaces in which users directly manipulate text-based formats. Taking traditional programming as an analogy, graphical programming environments are helpful for those new to a programming language, and can even play a significant role in experienced engineers' workflows, but experienced developers rely on the ability to edit source code directly—a language without an accessible text-based format would be difficult to promote. As another parallel, it has frequently been noted that one of the primary advantages of HTML over its early competitors was the ability to easily examine and modify a web page's source code using a simple text editor. The complexity of RDF/XML makes text editing of OWL ontologies in that format extremely demanding, and graphical editors are not a sufficient replacement for a manageable syntax.

The second issue has led to the development of a number of sophisticated libraries which handle RDF parsing and allow programmatic access to ontology

content. Systems such as Jena[1] offer direct access to an RDF model, while the WonderWeb OWL API [BVL03] and KAON2[2] include Java libraries with customized APIs for working with ontology structures at a higher level of abstraction than RDF graphs. Using such libraries does avoid the need for from-scratch parser implementation, but it also requires that developers learn new APIs and manage sometimes obscure library dependencies. More importantly, the primary advantage of OWL standardization has been lost: code for working with OWL ontologies is dependent not on the OWL standard, but on the particular API chosen for implementation. Most damningly, the current technology landscape suggests that working with OWL requires Java programming expertise. This puts OWL applications beyond the scripting skills of many biologists, and even out of reach of many web programmers who work mainly in Perl, Python, Ruby, and Javascript.

This paper addresses the stated problems with RDF/XML by offering a structured data model for ontologies built from primitive data types available in all major programming environments. Such a model is easily accessible to a wide range of implementors without the need for a proprietary API. Furthermore, the standard YAML [YAM] serialization of these structures provides a text ontology format appropriate for human authors. This ontology format has a standard translation to OWL 1.1 [PSH06] (a superset of OWL DL) and is interpreted using OWL 1.1 semantics, and it supports all features of OWL 1.1 with the exception of datatypes and annotations. This paper is not meant to be a complete specification for all aspects of the format; an extended specification, conversion tools, tutorial code, and sample ontologies are available at the Structure Ontology Format web site.[3]

2 Background

Description logic notations derived from structured data models are not new. The KRSS syntax [PSS] was effectively a purely structural specification realized as "symbolic expressions" (S-expressions), the fundamental datatype in the LISP programming language based on linked lists of atoms (with a canonical serialization in LISP syntax). Such a "native" format was ideally suited to development of LISP reasoners and tools (and to ontology authoring by LISP programmers). Parsing and programmatic manipulation of S-expressions in other languages, however, is not widely supported, and while the syntax is very clean, the standard prefix notation and heavy use of parentheses feel unnatural to many users.

The W3C recognizes at least three different serializations for OWL: an *abstract syntax* [PSHH03] using a function-style format, the official exchange syntax [DCv$^+$02] based on RDF graphs (and serialized as XML encodings of these graphs), and a rarely-used pure XML syntax [HEPS03] defined in a "W3C Note".

[1] `http://jena.sourceforge.net/`
[2] `http://kaon2.semanticweb.org/`
[3] `http://www.cs.man.ac.uk/~rshearer/sof/`

The Description Logic Implementors Group[4] has defined an alternative XML language for encoding description logic ontologies. (The next version of the DIG specification is expected to share a pure XML syntax with the OWL 1.1 proposal.)

XML does offer a formal data model, so XML-based formats can be viewed as structural specifications and processed with one of the many XML toolchains available. Such tools can be integrated with most programming environments, but "native" manipulation of XML structures (and appropriate mechanisms for abstraction of irrelevant low-level details) is available in only very specialized languages (such as XSLT and XQuery). Furthermore, XML syntax is not optimized for direct human interaction.

The Open Biomedical Ontologies[5] effort has independently developed a standardized encoding for ontologies[6], with human-readable syntax and simple parsing as major design goals. The OBO syntax breaks an ontology document into labeled sections called "stanzas"; a stanza contains a set of "tags", each specified in `key: value` format on a single line. Some stanzas define "terms" (comparable to OWL classes), and some describe "instances" (comparable to OWL individuals). OBO format is very accessible to human users, and a mapping from OBO to OWL has recently been proposed[7], but the syntax requires a custom parser, there is no obvious programmatic API for OBO data, and the language does not provide the same formal expressiveness as OWL.

Graphical modeling tools frequently need to display complex class expressions to users, and this was initially done using formal logic symbols (e.g. $\exists R.(C \sqcap D)$). In order to make such descriptions more accessible to nonlogicians, the *Manchester OWL Syntax* (MOS) [HDG+06] was designed to use infix notation and read as natural language. (The above expression would be written in MOS as "R some (C and D)".) A simple frame-based syntax was described for text exchange of class definitions, but the approach was not extended to a full ontology language. The structured format presented in this paper incorporates a formalized, extended version of Manchester syntax for complex class descriptions (see Section 3.2).

3 Structured Ontology Format

3.1 Data Model

The Structured Ontology Format (SOF) data model is based on three types of structure: character strings (the only atomic type) store text; *collections* directly contain other structures; and *maps* associate keys with values. The representational details of these types are unimportant: map and collection types may be

[4] `http://dl.kr.org/dig/`
[5] `http://obo.sourceforge.net/`
[6] `http://www.godatabase.org/dev/doc/obo_format_spec.html`
[7] `http://www.cs.man.ac.uk/~horrocks/obo/`

ordered or unordered, duplicate values are allowed (but never required) in collections, and maps may contain a single binding or multiple bindings for the same key.

The ontology format is dependent upon an *expression language* used to encode complex class descriptions, properties, and individual names. Expression languages are detailed in Section 3.2; for the purposes of this discussion, classes, properties, and individuals are assumed to be identified using strings.

An ontology is a map. If the string `classes` is present as a key in the ontology, then its binding is either a collection of class identifiers, or a map whose keys are class identifiers and whose values are *frames* describing the classes to which they are bound. Such frames are maps; if a frame bound to C binds the string `subsumed by` to a collection containing class identifier D, then D subsumes C is an axiom of the ontology. Bindings within class frames for keys such as `equivalent to` and `disjoint from` are interpreted analogously.

Other keys in an ontology have similar meanings: the value bound to the `properties` key is a collection or a map from property identifiers to frames describing those properties, and that bound to `individuals` is a collection or a map to individual frames.

The bindings for the `facts`, `class axioms`, and `property axioms` keys within an ontology are not maps but collections containing axioms not directly associated with any particular class, property, or individual. Within the `class axioms` collection, a map containing a single binding from `disjoint` to a collection of class identifiers asserts that all of the specified classes are disjoint from one another.

The formal semantics for SOF ontologies are given by a correspondence with OWL 1.1 functional syntax, presented in Table 1. We use a small subset of YAML notation to represent structures: $[x_1,...,x_n]$ is a collection containing elements x_1 through x_n, and $\{x : y\}$ is a map in which the binding for key x is y. A path syntax identifies the bindings for keys within nested maps: "/foo" indicates the value associated with key `foo` in the ontology map, and "/foo/bar" identifies the binding for key `bar` within "/foo". An object x *contains* value y if x is a collection containing y as an element, or if x is a map with an assignment for key y. Finally, for expression x in a structured ontology, the term $\bar{x}$ within an OWL axiom represents the translation of x to a class, property, or individual expression in OWL functional syntax, in accordance with the chosen expression language (described in Section 3.2).

The OWL 1.1 ontology derived from an ontology in SOF includes the axiom in the third column of Table 1 for every value identified by the path in the first column which contains the structure given in the second column. Converting an ontology to SOF from OWL 1.1, however, can be done in a number of ways. Most OWL axioms can be encoded in several different ways in SOF; e.g. `SubClassOf(`c`,` d`)` can be encoded by adding c to "/classes/d/subsumes", adding d to "/classes/c/subsumed by", or adding "$\{c : d\}$" to "/class axioms". Any translation need choose only one such translation. Rows 1–33 are usually only applicable to restricted forms of the OWL 1.1 axioms (e.g. equality axioms

Table 1. OWL interpretation of Structured Ontology Format

	SOF structure	Contains	OWL 1.1 Equivalent
1	/classes	c	Declaration(OWLClass($\bar{c}$))
2	/classes/c/subsumed by	d	SubClassOf($\bar{c}$ $\bar{d}$)
3	/classes/c/subsumes	d	SubClassOf($\bar{d}$ $\bar{c}$)
4	/classes/c/equivalent to	d	EquivalentClasses($\bar{c}$ $\bar{d}$)
5	/classes/c/disjoint union of	$[d_1,...,d_n]$	DisjointUnion($\bar{c}$ $\bar{d_1}$ ... $\bar{d_n}$)
6	/classes/c/disjoint from	d	DisjointClasses($\bar{c}$ $\bar{d}$)
7	/classes/c/domain of	r	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
8	/classes/c/range of	r	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
9	/classes/c/members	i	ClassAssertion($\bar{i}$ $\bar{c}$)
10	/properties	r	Declaration(ObjectProperty($\bar{r}$))
11	/properties/r/subsumed by	s	SubObjectPropertyOf($\bar{r}$ $\bar{s}$)
12	/properties/r/subsumes	s	SubObjectPropertyOf($\bar{s}$ $\bar{r}$)
13	/properties/r/subsumes	$[s_1,...,s_n]$	SubObjectPropertyOf(SubObjectPropertyChain($\bar{s_1}$ ... $\bar{s_n}$) $\bar{r}$)
14	/properties/r/equivalent to	s	EquivalentObjectProperties($\bar{r}$ $\bar{s}$)
15	/properties/r/inverse	s	InverseObjectProperties($\bar{r}$ $\bar{s}$)
16	/properties/r/disjoint from	s	DisjointObjectProperties($\bar{r}$ $\bar{s}$)
17	/properties/r/domain	c	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
18	/properties/r/range	c	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
19	/properties/r	functional	FunctionalObjectProperty($\bar{r}$)
20	/properties/r	inverse functional	InverseFunctionalObjectProperty($\bar{r}$)
21	/properties/r	reflexive	ReflexiveObjectProperty($\bar{r}$)
22	/properties/r	irreflexive	IrreflexiveObjectProperty($\bar{r}$)
23	/properties/r	symmetric	SymmetricObjectProperty($\bar{r}$)
24	/properties/r	asymmetric	AntisymmetricObjectProperty($\bar{r}$)
25	/properties/r	transitive	TransitiveObjectProperty($\bar{r}$)
26	/properties/r/related	$\{i : j\}$	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
27	/properties/r/not related	$\{i : j\}$	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
28	/individuals	i	Declaration(Individual($\bar{i}$))
29	/individuals/i/same as	j	SameIndividual($\bar{i}$ $\bar{j}$)
30	/individuals/i/different from	j	DifferentIndividuals($\bar{i}$ $\bar{j}$)
31	/individuals/i/member of	c	ClassAssertion($\bar{i}$ $\bar{c}$)
32	/individuals/i/related/r	j	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
33	/individuals/i/not related/r	j	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
34	/facts	$\{i : c\}$	ClassAssertion($\bar{i}$ $\bar{c}$)
35	/facts	$\{\{i : j\} : r\}$	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
36	/facts	{same : $[i_1,...,i_n]$}	SameIndividual($\bar{i_1}$ ... $\bar{i_n}$)
37	/facts	{different : $[i_1,...,i_n]$}	DifferentIndividuals($\bar{i}$ $\bar{j}$)
38	/facts	{not related : $\{\{i : j\} : r\}$}	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
39	/class axioms	{disjoint : $[c_1,...,c_n]$}	DisjointClasses($\bar{c_1}$ ... $\bar{c_n}$)
40	/class axioms	{equal : $[c_1,...,c_n]$}	EquivalentClasses($\bar{c_1}$ ... $\bar{c_n}$)
41	/class axioms	$\{c : d\}$	SubClassOf($\bar{c}$ $\bar{d}$)
42	/class axioms	{disjoint union : $\{c : [d_1,...,d_n]\}$}	DisjointUnion($\bar{c}$ $\bar{d_1}$ ... $\bar{d_n}$)
43	/property axioms	{disjoint : $[r_1,...,r_n]$}	DisjointObjectProperties($\bar{r_1}$ ... $\bar{r_n}$)
44	/property axioms	{equal : $[r_1,...,r_n]$}	EquivalentObjectProperties($\bar{r_1}$ ... $\bar{r_n}$)
45	/property axioms	$\{r : s\}$	SubObjectPropertyOf($\bar{r}$ $\bar{s}$)
46	/property axioms	$\{[r_1,...,r_n] : s\}$	SubObjectPropertyOf(SubObjectPropertyChain($\bar{r_1}$ ... $\bar{r_n}$) $\bar{s}$)
47	/property axioms	{functional : r}	FunctionalObjectProperty($\bar{r}$)
48	/property axioms	{inverse functional : r}	InverseFunctionalObjectProperty($\bar{r}$)
49	/property axioms	{reflexive : r}	ReflexiveObjectProperty($\bar{r}$)
50	/property axioms	{irreflexive : r}	IrreflexiveObjectProperty($\bar{r}$)
51	/property axioms	{symmetric : r}	SymmetricObjectProperty($\bar{r}$)
52	/property axioms	{asymmetric : r}	AnitsymmetricObjectProperty($\bar{r}$)
53	/property axioms	{transitive : r}	TransitiveObjectProperty($\bar{r}$)
54	/property axioms	{domain : $\{r : c\}$}	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
55	/property axioms	{range : $\{r : c\}$}	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
56	/property axioms	{inverse : $\{r : s\}$}	InverseObjectProperties($\bar{r}$ $\bar{s}$)

involving only two elements); all OWL 1.1 axioms can be transformed to SOF using rows 34–56.

3.2 Expression Language

The structures used to encode axioms in SOF are at most five levels deep, but OWL class descriptions (and property expressions) can be very complex, with arbitrarily deep nesting of subexpressions. It is difficult to define a single format for such a language which is both readable for humans and easy for machines to process.

The structured format described in Section 3.1 is independent of the language used for class descriptions, property expressions, and individual identifiers. This allows different dialects of structured format to use different expression languages.

In order to eliminate the need for text parsing, a structured expression language similar to KRSS but with OWL 1.1 expressiveness has been formalized. (Details are available on the SOF web site.[8]) For many users, however, a more lightweight, readable encoding closer to natural language is preferable. For this reason, a second dialect of structured ontology format is defined which encodes class, property, and individual expressions as simple (unicode) strings. These strings are interpreted as Extended Manchester OWL Syntax (EMOS), which we summarize here. A formal grammar and translation to OWL 1.1 are given at the Manchester OWL Syntax web site.[9]

EMOS is backwards-compatible with the original Manchester syntax presented in [HDG+06]. Manchester syntax was designed as a simple and readable text format for expressing complex class descriptions, intended primarily for presenting such descriptions to (non-logician) human users. The focus is on allowing even relatively complex expressions to be readable as natural (English) language: ObjectIntersectionOf, ObjectUnionOf, and ObjectComplementOf OWL descriptions are written with `and` (or `that`), `or`, and `not`, class expressions involving properties (such as ObjectSomeValuesFrom and ObjectHasValue) are expressed using infix notation (with keywords `some`, `only`, `min`, `max`, `exactly`, and `value`), and parentheses are needed only where rules of precedence demand them. The language also includes a number of shorthands for common modeling patterns: the `someonly` construct stands for the intersection of multiple `some` and `only` expressions, `never R` is short for `R max 0`, and `always R` means `R some owl:Thing`. A property `P`'s inverse is given by `P-`, and class, property, and individual names can be quoted to prevent conflicts with the language keywords.

Identifers in EMOS can be explicitly typed as full RFC 3987 IRIs (surrounded by "`<`" and "`>`" tokens), or they can be interpreted subject to namespace expansion similar to that described in [BHLT06]. Crucially, interpretation of EMOS expressions is dependent upon a set of *namespace bindings* mapping namespaces prefixes to their expansions. In Structured Ontology Format, these bindings are

[8] `http://www.cs.man.ac.uk/~rshearer/sof/sel`
[9] `http://www.cs.man.ac.uk/~rshearer/mos/`

given by a map stored as the value of the `namespaces` key in the ontology object (with the default namespace bound to a null or empty prefix). Unlike in standard XML, a namespace prefix used in a legal EMOS identifier need not be declared; the interpretation of such identifiers is application-dependent.

Although this expression language is relatively straightforward to parse, such string processing is far more complex than manipulation of an SOF ontology's axiom structure. Ontology authors and authoring tools should be mindful of the fact that some SOF processors may view class descriptions, property expressions, and individual identifiers as opaque strings and might be unable to perform even namespace expansion. When the same expression occurs multiple times within an ontology, it is recommended that all encodings of that expression be lexically identical, particularly when the expression is a simple identifier.

3.3 Serialization

YAML[10] is a standard serialization format for simple data structures, including maps, ordered collections (called *sequences*), and string values. The syntax is extremely flexible, with a number of different styles of encoding for each data type. Sequences, for example, can be written as comma-separated lists of values within square brackets (as in Table 1), or elements can be listed on separate lines, with each element preceded by a dash. Nested sequences are distinguished by indentation level. Bindings within maps are written as `key : value`, and maps can be enclosed in curly braces with comma-separated bindings, or they can be written in a line-oriented style similar to sequences. The syntax also allows comments (introduced by `#`, and continuing to the end of the line). The full YAML specification is defined in [YAM]. A small example of a complete SOF ontology serialized as YAML is shown in Figure 1.

While YAML is the canonical (and most human-friendly) serialization format for SOF ontologies, in some cases more restrictive encodings are useful. The JSON[11] fragment of YAML allows only the braced/bracketed forms of maps and sequences (among other restrictions), and as a result JSON is extremely simple to parse. SOF ontologies using Structured Expression Language and serialized as JSON are ideally suited to efficient machine-to-machine exchange of OWL ontology data.

3.4 Special Forms

Most OWL axioms can be encoded in two or even three different ways in structured format—for example an ObjectPropertyDomain axiom in an OWL ontology can be represented in SOF according to rows 7, 17, or 54 of Table 1 (modulo class and property declarations). This provides a great deal of flexibility for ontology authors, but it is problematic for tools (or humans) scanning an ontology

[10] `http://yaml.org/`
[11] `http://www.json.org/`

```
namespaces:
    "" : http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#
    food : http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#

classes:
    Wine:
        subsumed by:
            - food:PotableLiquid   # note use of namespace
            - hasMaker exactly 1   # class descriptions use EMOS
            - locatedIn some Region
    TableWine:
        equivalent to:
            - Wine that hasSugar value Dry
properties:
    hasMaker:
        inverses: [ producesWine ] # bracketed syntax for sequences
        functional:                # some keys don't need values
    locatedIn:
        range: [ Region ]
        transitive:
individuals:
    StonleighSauvignonBlanc:
        member of:
            - Wine
        related:
            hasSugar: [ Dry ]      # sequences can be bracketed...
            hasMaker:
                - Stonleigh        # ...or use a line-oriented style.
            locatedIn: [ NewZealandRegion ]
```

Fig. 1. A small portion of the Wine ontology in SOF, using Manchester syntax and YAML serialization

for a particular type of axiom. We thus define a number of *normal forms* which add requirements that certain axioms be encoded in particular ways.

For a given row r of Table 1, OWL ontology K, and SOF representation of K S, if every axiom of K which could be represented in structured format in accordance with row r is represented in that way in S, then we say that S is *normalized* with respect to r. Such normalization does not require reasoning about semantic entailments between axioms: normal forms deal only with different ways of encoding the same structural/syntactic content.

An ontology is said to be *class-frame normalized* if it is normalized with respect to rows 1–6 of Table 1. Note that class-frame normalization does not require the encoding of domain, range, or membership axioms within the class frame.

Analogously, *property-frame normalized* ontologies are normalized with respect to rows 10–25, and *individual-frame normalized* ontologies are normalized with respect to rows 28–33. A *fully frame-normalized* ontology is normalized with respect to rows 1–33 (and thus must be individual-frame, property-frame, and class-frame normalized), and an *axiom-normalized* ontology is normalized with respect to rows 34–56. A *fully normalized* ontology is normalized with respect to all rows of Table 1.

In addition to these simple syntactic conditions on an ontology's SOF representation, three criteria on the underlying ontology data are very useful for ontology authoring and processing tools. First, if all namespace prefixes used in all class, property, and individual expressions throughout the ontology are declared in the `namespaces` mapping, then the ontology is *namespace consistent*. Without this property, there is no guarantee that two different translations of the same ontology into OWL will entail each other. Second, an ontology with declarations for all class, property, and individual names is *structurally consistent*. (This notion is from the OWL 1.1 specification.) An ontology which is both namespace and structurally consistent can benefit from extensive authoring support (highlighting of typos, tab completion, etc.). Finally, given some definition of equivalence between class descriptions (lexical equality is sufficient), an ontology is *taxonomy optimized* if no class subsumption axioms are redundant with respect to the transitive-reflexive closure of all subsumption axioms in the ontology.

4 Discussion and Future Work

Structured format has already proven to be extremely useful as a language for creating test suites for new reasoner implementations: the syntax makes it possible to author simple tests (with dozens of axioms) by hand, and more complex tests are easy to construct and serialize using standard scripting languages. Furthermore, implementation of a parser for OWL's RDF/XML syntax can be substantially more expensive than construction of naïve reasoners for restricted logic fragments. The use of a Java tool based on one of the existing OWL parsing libraries to "preprocess" ontologies into structured format has allowed realistic knowledge bases to be used as tests for new reasoning algorithms prototyped in Perl and Python.

Conversion of ontologies to SOF and examination with text tools such as `grep` has replaced ontology exploration using graphical tools in some workflows. Version control of ontologies maintained in structured format has proven much easier than with other formats: text difference tools allow the same conflict-resolution strategies as are common with traditional programming languages.

Straightforward programmatic access to ontology data using modern dynamic languages has decreased the engineering cost for developing new OWL tools by several orders of magnitude. Routines to convert ontologies between the special forms described in Section 3.4 can be implemented in under a dozen lines of Python code (and only a few minutes' work), and a from-scratch ontology ex-

ploration interface intended for visually-impaired users weighs in at under one hundred lines (and roughly an hour of implementation time).

The current specification, however, offers only partial coverage of OWL 1.1: neither datatypes, datatype properties, nor annotations are currently supported. Further, there is no "import" functionality for structured ontologies. Networks of OWL imports are converted to SOF as a single monolithic text file, which can be difficult to manage. An extension of structured format which includes these features is currently being developed.

References

BHLT06. Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (second edition). Technical Report http://www.w3.org/TR/2006/REC-xml-names-20060816/, W3C, August 2006. `http://www.w3.org/TR/2006/REC-xml-names-20060816/`.

BVL03. Sean Bechhofer, Raphael Volz, and Phillip W. Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer, 2003.

DCv+02. Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language 1.0 reference, July 2002. `http://www.w3.org/TR/owl-ref/`.

HDG+06. Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang. Manchester OWL syntax. In *Proc. of the 2006 OWL: Experiences and Directions Workshop (OWLED 2006)*, 2006.

HEPS03. Masahiro Hori, Jérôme Euzenat, and Peter F. Patel-Schneider. OWL web ontology language XML presentation syntax. W3C Note, 11 June 2003. `http://www.w3.org/TR/owl-xmlsyntax/`.

KFNM04. Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.

KPH05. Aditya Kalyanpur, Bijan Parsia, and James A. Hendler. A tool for working with web ontologies. *Int. J. Semantic Web Inf. Syst.*, 1(1):36–49, 2005.

PSH06. Peter F. Patel-Schneider and Ian Horrocks. Owl 1.1 web ontology language overview. W3C Member Submission, 19 December 2006. `http://www.w3.org/Submission/2006/10/`.

PSHH03. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. W3C Candidate Recommendation, 18 August 2003. Available at `http://www.w3.org/TR/owl-semantics/`.

PSS. Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. `http://www.cs.bell-labs.com/cm/cs/who/pfps/publications/krss-spec.pdf`.

YAM. YAML specification. `http://www.yaml.org/spec/`.