The OBO to OWL mapping, GO to OWL 1.1!

Christine Golbreich[1] Ian Horrocks[2]

[1]University of Versailles-Saint Quentin
55 avenue des Etats-Unis, 78035 Versailles, France
`Christine.Golbreich@uvsq.fr`

[2]Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
`horrocks@cs.man.ac.uk`

Abstract. A large library of biomedical ontologies has been developed in the OBO format, including important ontologies such as the Gene Ontology (GO). These resources have the potential to contribute to the Semantic Web in the life sciences domain. In particular, they allow the annotation of distributed experimental data in using a controlled and shared vocabulary. As the Web Ontology Language OWL is the W3C recommended standard for ontologies, converting OBO ontologies to OWL becomes an important need: it will facilitate the sharing and reuse of this important resource and make the expanding range of OWL tools available for use with OBO ontologies. In order to achieve this, we propose a formalisation of the OBO syntax in terms of a BNF style grammar, and a formalisation of the semantics of (a large part of) OBO in terms of a mapping to OWL 1.1. This demonstrates that the OWL 1.1 extension of OWL allows nearly all of the OBO language to be captured.

1 Introduction

The "OBO" acronym refers to different types of resources and efforts. First, OBO is the name of a large library of open biomedical ontologies known as Open Biomedical Ontologies (`http://www.bioontology.org/repositories.html#obo`), which is being built and hosted by the National Center for Biomedical Ontology (NCBO). The OBO repository is intended to be accessed using a Web application called BioPortal. The intention is to provide various ontology repository services, including inference, alignment, version control services etc.

On the other hand, OBO refers to a format often used for describing biomedical ontologies (see `http://www.geneontology.org/GO.format.obo-1_2.shtml`). The OBO "flat file format", originated with the Gene Ontology (GO) and is similar to the tag-value format of the GO definitions file, with a few modifications (there are also obvious similarities to the N3 syntax for RDF). There are two types of OBO format, the older GO flat file format (The OBO Flat File Format Specification, version 1.0) and the newer one (The OBO Flat File Format Specification, version 1.2). The GO flat file format is now deprecated but will continue to be provided alongside the new format. According to its authors, the OBO flat format main aims are to provide 1) human readability, 2) ease of parsing, 3) extensibility and 4) minimal redundancy. The URL `http://obo.sourceforge.net/` is an umbrella web address allowing users to view OBO ontologies in table form and to browse them. In this table, a subset of the OBO ontologies have the tag candidate "OBO Foundry". This designates the ontology as a candidate for the "OBO Foundry" project, a new paradigm for biomedical ontology development driven by a set of principles specifying best developmental practices. In the following, OBO will refer to the OBO Flat File Format Specification, version 1.2.

As OBO is much used in the life sciences and supports important ontologies, including GO, the Chemical Ontology (ChEBI), the Cell Ontology (CL) etc., a mapping from OBO to OWL [1] is of primary importance for the life science domain and for the adoption of Semantic Web technologies by the life science community. Indeed, the OWL language offers several advantages:

Interoperability. Interoperability of Web ontologies is crucial for shared use across different biomedical domains, as expected from the Open Biomedical Ontologies library. OWL fosters integration of and interoperability between Web biomedical ontologies. Once converted to OWL, OBO ontologies become easier to integrate with other OWL biomedical ontologies. The conversion of several major biomedical ontologies from other native languages to OWL is underway, including the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [2], the Foundational Model of Anatomy (FMA) [3], the Medical Subject Headings (MeSH) [4] and the National Cancer Institute Thesaurus (NCI) [5]. The conversion of ontologies to OWL has also been investigated, or is being invetigated e.g., the UMLS Metathesaurus and Semantic Network [6]; and some more recent ontologies are directly developed in OWL e.g., the Biological Pathways Exchange ontology BioPAX (see `http://www.biopax.org/`).

Expressiveness. Also of interest is the higher expressiveness of OWL and OWL 1.1 [7] which allows for the representation not only of most of the OBO primitives and the satisfaction of most of the requirements initially formulated by OBO developers, but also provides significantly increased expressive power.

Semantics. Another major advantage of OWL and OWL 1.1 is their well defined semantics (see `http://www.w3.org/Submission/owl11-semantics/`). This prevents any ambiguity and/or misunderstanding of the biomedical ontology vocabulary. Moreover, the logical semantics provides the grounding of powerful reasoning services supporting ontology engineering.

Tools and services. OWL provides access to an expanding range of tools and services that facilitate both development and deployment of biomedical ontologies, namely building, maintaining and reusing ontologies. Biomedical ontologies are often huge and almost always evolving. For example, SNOMED [2] has over 308,000 active concepts, the FMA [3] has over 75,000 anatomical classes, has been under development at the University of Washington since 1994, and is constantly being improved and extended. Automated tools are essential to check the consistency of large ontologies. OWL DL (and OWL 1.1) reasoners, e.g. RacerPro [8], Pellet [9] and Fact++ [10], enable the automatic detection of inconsistencies caused by possible modelling errors, which may otherwise be difficult to identify. OWL also supports tools for ontology debugging and modularity. Debugging tools [11, 12] allow for tracing the reasons for inconsistencies identified by reasoners. Modularity tools [13, 14] allow for integrating and extracting different modules, e.g., extracting a module of *brain anatomy* from the FMA ontology. Consistency checking, debugging and modularity services are of primary importance for the OBO repository. Given a mapping to OWL, biomedical ontologists can exploit all these services while still modelling in the familiar OBO framework.

For all these reasons, converting OBO ontologies to OWL has become an important need. To this end we propose a formalisation of the OBO Flat File Format Specification version 1.2, available at `http://www.godatabase.org/dev/doc/oboa_format_spec.html`, in terms of a BNF style grammar, and a formalisation of the semantics of (a large part of) OBO in terms of a mapping to OWL (1.1). The document that describes the grammar and mapping in detail is available online at `http://www.cs.man.ac.uk/~horrocks/obo/`, and is open for discussion.

The remainder of this paper is organised as follows: Section 2 gives an overview of the syntax proposed for OBO; Section 3 presents the semantics of (part of) OBO defined via its mapping to OWL (1.1); and Section 4 discusses the main advantages of the approach.

2 OBO Syntax

This section presents a fragment of the BNF style syntax defined at `http://www.cs.man.ac.uk/~horrocks/obo/` to show the general idea. The grammar of OBO is presented in the standard BNF notation. Nonterminal symbols are denoted in bold (e.g., **stanza**), terminal symbols are written in single quotes (e.g. 'Term'), zero or more instances of a symbol is denoted with curly braces (e.g., { ***stanza*** }), alternative productions are denoted with the vertical bar (e.g., ***term-stanza*** | ***typedef-stanza***), and zero ore one instance of a symbol are denoted with square brackets (e.g., ['is_anonymous: true']).

OBO File Structure. An OBO file consists of a header followed by zero or more stanzas. Each stanza introduces and describes the properties of either a class (a Term stanza), a property (a Typedef stanza) or an individual (an Instance stanza). The syntax for OBO ontology files is defined as follows:

OBO-Doc := ***header*** **{** ***stanza*** **}**
stanza := ***term-stanza*** | ***typedef-stanza*** | ***instance-stanza***

OBO Header. The header consists of a number of tag-value pairs (several are ignored for the time being). Many of these e.g., <remark> could clearly be treated as annotations; others (e.g., <data-version>, <saved-by>) would correspond to parts of an XML document preamble or to the default namespace of an ontology (e.g., <idspace>).

The syntax for OBO header is defined as follows:

```
header :=
        <format-version>
        [ <data-version> ]
        [ <date> ]
        [ <saved-by> ]
        [ <auto-generated-by> ]
        [ <subsetdef> ]
        { import }
        { <synonymtypedef> }
        { <idspace> }
        [ <default-relationship-id> ]
        { <idmapping> }
        [ <remark> ]
import := 'import:' <URL>
```

Term Stanzas. Term stanzas introduce and define the meaning of *terms* (AKA concepts, classes and unary predicates). For example, in the OBO ontology CARO (Common Anatomy Reference Ontology)[1] being developed, the concept *anatomical structure* is specified by:

```
[Term]
id: CARO:0000003
name: anatomical structure
def: "Material anatomical entity that has inherent 3D shape and is generated by
    coordinated expression of the organism's own genome." [CARO:MAH]
is_a: CARO:0000006
```

The syntax for OBO Term is defined as follows:

```
term-stanza:=
            '[Term]'
            termid-TVP
            'name:'<string>
            [ <namespace> ]
            { <alt_id> }
            [ <def> ]
            [ <comment> ]
            { <subset> }
            { <synonym> }
            { <xref> }
            { isa-TVP }
            { intersection-TVP }
            { union-TVP }
            { disjoint-TVP }
```

[1] http://purl.org/obo/owl/CARO

{ ***relationship-TVP*** }
[<is_obsolete>]
[<replaced_by>]
{ <consider>

termid-TVP :=
'id:' ***term-id***
['is_anonymous: true']
term-id := <string>

isa-TVP :=
'is_a:' ***term-id***
['namespace=' <namespace-id>]
['derived=true' | 'derived=false']

intersection-**TVP** :=
'intersection_of:' **termOrRestr**
['namespace=' <namespace-id>]
termOrRestr := ***term-id*** | **restriction**
restriction := ***relationship-id term-id***
relationship-id := <string>

union-TVP :=
'union_of:' ***termOrRestr***
['namespace=' <namespace-id>]

disjoint-TVP :=
'disjoint_from:' ***term-id***
['namespace=' <namespace-id>]
['derived=true' | 'derived=false']

relationship-TVP :=
'relationship:' ***restriction***
['not_necessary=true' | 'not_necessary=false']
['inverse_necessary=true' | 'inverse_necessary=false']
['cardinality=' <non-neg-int>]
['maxCardinality=' <non-neg-int>]
['minCardinality=' <non-neg-int>]

Typedef Stanzas and **Instances stanzas**. Typedef stanzas introduce and define the meaning of *relations* (AKA roles, properties and binary predicates). For example, the relation *part_of* is specified in the OBO ontology CARO by:

[Typedef]
id: part_of
name: part_of
is_transitive:true

An extract of the syntax for OBO Typedef stanzas, introducing the main tags that have been mapped to OWL, is given below (for details about Typedef and Instance Stanzas, see [15]):

typedef-stanza:=

    '[Typedef]'
    typedef-TVP
    ...
    [***domain-TVP***]
    [***range-TVP***]
    { ***meta-property-TVP*** }
    { ***r-isa-TVP*** }
    [***inverse-TVP***]
    [***transover-TVP***]
    ...

Instance stanzas introduce and define the meaning of *instances* (AKA individuals, individual names, constants); as we are mainly interested in "schema-level" ontologies, we won't discuss them further in this paper.

3 OBO semantics and mapping to OWL(1.1)

The semantics for (part of) the OBO language, given via its translation to the OWL DL abstract syntax, is presented in [15]. This section presents the semantics of an even larger subset of OBO via a mapping to OWL 1.1, and shows that the OWL 1.1 extension allows nearly all of the OBO language to be captured.

Mapping. The translation is defined using a translation function T which translates (a fragment of) OBO into OWL 1.1, according to the mapping described in Table 1. The definition of T is often recursive, but it will eventually "ground out" in (a fragment of) OWL 1.1. A number of simplifying assumptions are made in order to improve readability:

- Syntax related declarations (namespace, etc.) are largely ignored.
- OBO identifiers/names are translated "as is".
- Annotations to entities are added using the OWL 1.1 *Entity Annotation* facility.[2]
- The "false" cases of relation qualifying tags are ignored, although they could be translated into OWL "facts" that assert counter-examples, if this is deemed to be appropriate. E.g., given a relation R with "is_transitive: false" in its Typedef stanza, we could introduce new individuals x, y and z, and assert $R(x, y)$, $R(y, z)$ and $\neg R(x, z)$.
- According to the translation, OBO relations are translated as OWL object properties; it is assumed that, in practice, OBO relations would be translated

[2] `http://www.w3.org/Submission/2006/SUBM-owl11-owl_specification-20061219/#4.4`

into OWL datatype properties if either the Typedef stanza specifies a range that is an XML datatype or if some super relation was translated into a datatype property.

– Relation qualifying tags, reflexive, irreflexive, transitive_over etc., are handled using the extra features offered by OWL 1.1 to assert reflexive, irreflexive, asymmetric properties etc., and in particular the new possibility of expressing property chain inclusion axioms.
– Relationships in Typedef stanzas are ignored due to uncertainty as to the intended semantics.

OBO syntax (fragment) — S	Translation — T(S)
header stanza_1 ... stanza_n	T(header) T(stanza_1) ... T(stanza_n)
[Term]	Declaration(OWLClass(term-id))
id:term-id	
name:name-string	EntityAnnotation(OWLClass(term-id) Label(name-string))
isa:isa-term_1	SubClassOf(term-id isa-term_1)
...	...
isa:isa-term_i	SubClassOf(term-id isa-term_i)
intersection_of:int-termOrRestr_1	EquivalentClasses(term-id
...	ObjectIntersectionOf(T(intersection_of:int-termOrRestr_1)
intersection_of:int-termOrRestr_j	... T(intersection_of:int-termOrRestr_j)))
union-of:union-termOrRestr_1	EquivalentClasses(term-id
...	ObjectUnionOf(T(union-of:union-termOrRestr_1)
union-of:union-termOrRestr_k	... T(union-of:union-termOrRestr_k)))
disjoint_from:disjoint-term_1	DisjointClasses(term-id disjoint-term_1)
...	...
disjoint_from:disjoint-term_m	DisjointClasses(term-id disjoint-term_m)
relationship-TVP_1	SubClassOf(term-id T(relationship-TVP_1))
...	...
relationship-TVP_n	SubClassOf(term-id T(relationship-TVP_n))
intersection_of:term-id	term-id
intersection_of:relationship-id term-id	ObjectSomeValuesFrom(relationship-id term-id)
union_of:term-id	term-id
union_of:relationship-id term-id	ObjectSomeValuesFrom(relationship-id term-id)
relationship:relationship-id term-id	ObjectSomeValuesFrom(relationship-id term-id)
relationship:relationship-id term-id cardinality=card	ObjectExactCardinality(card relationship-id term-id)
relationship:relationship-id term-id maxCardinality=max	ObjectMaxCardinality(max relationship-id term-id)
relationship:relationship-id term-id minCardinality=min	ObjectMinCardinality(min relationship-id term-id)
[Typedef]	Declaration(ObjectProperty(relationship-id))
id:relationship-id	
name:name-string	EntityAnnotation(ObjectProperty(relationship-id) Label(name-string))
domain:domain-id	ObjectPropertyDomain(relationship-id domain-id)
range:range-id	ObjectPropertyRange(relationship-id range-id)
meta-property-TVP_1	T(relationship-id meta-property-TVP_1)
...	...
meta-property-TVP_k	T(relationship-id meta-property-TVP_k)
r-isa:isa-id_1	SubObjectPropertyOf(relationship-id isa-id_1)
...	...
r-isa:isa-id_m	SubObjectPropertyOf(relationship-id isa-id_m)
inverse:inv-id	InverseObjectProperties(relationship-id inv-id)
transitive_over:tr-over-id	SubObjectPropertyOf(SubObjectPropertyChain(relationship-id tr-over-id) relationship-id)
relationship-id is_anti_symmetric:true	AntisymmetricObjectProperty(relationship-id)
relationship-id is_anti_symmetric:false	Declaration(ObjectProperty(relationship-id) comment(is_anti_symmetric:false))
relationship-id is_cyclic:true	Declaration(ObjectProperty(relationship-id) comment(is_cyclic:true))
relationship-id is_cyclic:false	Declaration(ObjectProperty(relationship-id) comment(is_cyclic:false))
relationship-id is_reflexive:true	ReflexiveObjectProperty(relationship-id)
relationship-id is_reflexive:false	Declaration(ObjectProperty(relationship-id) comment(is_reflexive:false))
relationship-id is_symmetric:true	SymmetricObjectProperty(relationship-id)
relationship-id is_symmetric:false	Declaration(ObjectProperty(relationship-id) comment(is_symmetric:false))
relationship-id is_transitive:true	TransitiveObjectProperty(relationship-id)
relationship-id is_transitive:false	Declaration(ObjectProperty(relationship-id) comment(is_transitive:false))

Fig. 1. Translation to OWL 1.1

Example. According to the OWL 1.1 mapping described, the OWL 1.1 Class for the CARO:0000003 Term given section 2 is:

Declaration(***OWLClass***(CARO:0000003))
EntityAnnotation(***OWLClass***(CARO:0000003)
  Label("anatomical structure"))
EntityAnnotation(***OWLClass***(CARO:0000003)
  Annotation(OBO:def "Material anatomical entity that has inherent 3D shape and
    is generated by coordinated expression of the organisms own genome.
    [CARO:MAH]"))
SubClassOf(CARO:0000003 CARO:0000006)

Because of space limitation, it is not possible to show examples of Typedef, though OWL 1.1 features are essential for representing the various aspects of relations, e.g., ***SubObjectPropertyOf(SubObjectPropertyChain***(located_in part_of) located_in) (see section 4).

4 Discussion

Other groups have proposed a mapping between the OBO 1.2 file format and OWL. A summary of various conversion efforts from OBO to OWL can be found as an online Google spreadsheet available at `http://spreadsheets.google.com/ccc?key=pWN_4sBrd9l1Umn1LN8WuQQ`. Among them a mapping developped by Chris Mungall (Common Mapping) is available as a XSLT file at `http://www.godatabase.org/dev/xml/xsl/oboxml_to_owl.xsl`. A Protégé plugin based on it has recently been developed at Stanford. It allows to convert OBO files to OWL, from the OBO Converter Protégé tab. Another proposal for a "minimal OWL Full Ontology for OBO and the Gene Ontology" is available at `http://www.aiai.ed.ac.uk/resources/go`. According to the authors, this proposal aims at defining a very simple ontology of Objects, Events and part-of relations, and at axiomatising their semantics.

Providing OBO with a BNF syntax and a semantics via a mapping to OWL(1.1) has several advantages. On the one hand, the BNF style grammar for OBO helps to make the structure more precise and to clarify some ambiguities and/or misunderstandings of the OBO Flat File Format Specification, version 1.2. Indeed, the original OBO specification was not completely clear with respect to some details of syntax and semantics. Some of these have been resolved by communications from the OBO developers (in particular Chris Mungall), for example:

– An OBO file may include zero stanzas (e.g., a file of standard idspaces in obo format).
– In OBO, name is purely documentary (like RDF labels), and the id is what is used to refer to a term in other stanzas (although it is strongly encouraged, and may even be enforced, that names uniquely identify a class, relation or instance *within* an ontology or OBO namespace).

– The sets of Typedef (relationship) IDs, Term IDs and Instance IDs must be pairwise disjoint.
– The minimum and maximum number of occurrences of each tag type in a given stanza has now been explicated. For example, an OBO Typedef stanza may contain at most one of each of the following tags: range, domain, inverse, transitive_over and comment.
– OBO may adopt an OWL style import semantics in order to avoid potential problems with the existing "append document at parse time" semantics in case of cyclical imports statements.
– The not_necessary and inverse_necessary modifiers are likely to be deprecated, with their meaning being encoded in the semantics of relevant relations. Future versions of OBO may allow existential and universally quantified relationships to be distinguished.
– The relationship tag may be banned in Typedef stanzas as its meaning there is not clear.

However, OBO still needs a more precise specification, and as noticed in [15] a few points remain to be clarified:

– The derived modifier could lead to some strange situations, e.g., there could be two facts such that either can be derived, but not both. It isn't clear what the status of an OBO ontology would be if derived modifiers are "inconsistent". (This will be clarified in future versions of OBO.)
– It isn't clear if there is any restriction on mixing datatypes and classes, e.g., in intersection and union.
– It isn't clear if cardinality tags are meant to signify a qualified or unqualified restriction.
– It isn't clear if OBO built-in objects are semantically meaningful. This does not seem obviously useful, and might be potentially problematical (requiring a higher order language).
– It isn't clear what the "false" cases of the relation qualifiers (is_symmetric etc.) mean: for example, does "is_symmetric false" in the Typedef stanza signify that it is possible to have for some individuals x, y both $R(x, y)$ and $\neg R(y, x)$, or that when $R(x, y)$ is true then $R(y, x)$ is necessarily false? At the moment, they are translated by OWL 1.1 annotations.

Defining the syntax using a BNF also means that the grammar may be used as an input to parser generator tools such as JavaCC. Using this approach, a parser based on the previously described BNF and semantic mapping has been developed at Manchester as a module to the OWL API [16]. The parser is capable of parsing all well formed ontologies of the sourceforge repository (`http://obo.sourceforge.net/`), including GO, into OWL 1.1 data structures. It uses a pluggable tag-value-pair handler in order to provide semantic aware translations of the tags which have an OWL 1.1 mapping and to also be extensible in case of future evolutions of OBO format or OWL. On the other hand, a precise and rigourous mapping from OBO to OWL(1.1), defined as suggested, gives the OBO language a well-defined semantics. The other very important point to underline, is that OWL 1.1 extensions for properties are really needed for representing the various aspects of Typedef, e.g.; *transitive_over*. In particular, Property chain inclusion axioms is of primary importance for life sciences ontologies in general. It is precisely why OBO and other biomedical ontologies, for example SNOMED, include such features. Our next step will be to refine and complement the OBO language and its mapping to OWL 1.1., and to investigate the benefits of translating GO and other OBO ontologies to OWL1.1.

Similarly to OBO, "semi-formal" languages have often been defined in industry, where ISO standards are widely used. Using a BNF grammar for clarifying the syntax of such (ISO) languages and using a mapping to OWL(1.1) for defining their semantics, seem to be a promising approach for technical and engineering domains. In particular, the same "mapping to OWL" approach might be applied to the Parts Library (PLIB)—ISO 13584 Standard—initially dedicated to electronic and mechanic components specification and catalogues storage and exchange [17]. In each particular engineering domain, e.g., electronic components and process instruments (IEC 61360-4), measuring instruments (ISO 13584-501), machining tools (ISO 13399), mechanics (ISO/TC 10) NWI), optics and photonics (ISO/TC 172 NWI) etc., PLIB ontologies of Products are defined and are used in particular to create "e-business" catalogues. A mapping from PLIB to OWL 1.1 would allow the PLIB library to benefit of OWL tools and services to build and maintain the Products ontologies and for querying the catalogues supported by PLIB standard.

5 Conclusion

This paper has presented a precise and comprehensive mapping from OBO to OWL (1.1). The approach has several advantages. On the one hand, the BNF syntax and the semantics of OBO defined via a mapping to OWL (1.1) allow precision and coverage. Indeed, the BNF style grammar helps to make the structure more precise and to clarify some ambiguities and/or misunderstandings of the OBO Flat File Format Specification, version 1.2. It also allows the grammar to be automatically checked and to create a parser. Moreover, using OWL 1.1 allowed nearly all of OBO to be captured. As a result, it gives a well-defined semantics to a (large) part of OBO. On the other hand, this mapping of OBO to OWL 1.1 will provide biomedical users with all the available reasoning services supported by OWL, in particular to design, construct, maintain or reuse their ontologies, while still using their preferred language framework.

References

1. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
2. SNOMED Clinical Terms. Northfield, IL: College of American Pathologists, 2007.

3. Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, 4(3), 2006.
4. Lina Fatima Soualmia, Christine Golbreich, and Stéfan Jacques Darmoni. Representing the mesh in owl: Towards a semi-automatic migration. In *KR-MED*, pages 81–87, 2004.
5. Sherri de Coronado, Margaret W. Haber, Nicholas Sioutos, Mark S. Tuttle, and Lawrence W. Wright. NCI thesaurus: Using science-based terminology to integrate cancer research results. In *Proc. of MEDINFO 2004*. IOS Press, 2004.
6. V. Kashyap and A. Borgida. Representing the umls semantic network using owl: Or "what's in a semantic web link?". In Mylopoulos J Fensel D, Sycara K, editor, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Lecture Notes in Computer Science, pages 1–16. Springer, 2003.
7. Peter Patel-Schneider and Ian Horrocks. OWL 1.1 Web Ontology Language overview. W3C Member Submission, 19 December 2006. Available at `http://www.w3.org/Submission/owl11-overview/`.
8. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.
9. Evren Sirin and Bijan Parsia. Pellet: An OWL DL reasoner. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, 2004.
10. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.
11. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca-Grau, and James Hendler. SWOOP: a web ontology editing browser. *J. of Web Semantics*, 4(2), 2005.
12. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in owl ontologies. *J. of Web Semantics*, 3(4):243–366, 2005.
13. Bernardo Cuenca Grau, Yevgeny Kazakov, Ian Horrocks, and Ulrike Sattler. A logical framework for modular integration of ontologies. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 2007.
14. Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. of the Sixteenth International World Wide Web Conference (WWW 2007)*, 2007.
15. Ian Horrocks. Obo flat file format syntax and semantics and mapping to OWL Web Ontology Language. Editor's draft, 4 March 2007. Available at `http://www.cs.man.ac.uk/~horrocks/obo/`.
16. Matthew Horridge, Sean Bechhofer, and Olaf Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. OWLED 2007 (submitted), 2007.
17. ISO 13584-42. Industrial automation systems and intregration - Parts Library - part 42: Description methodology: Methodology for structuring parts families, iso, 1998.