Owlifier: Creating OWL-DL Ontologies from Simple Spreadsheet-Based Knowledge Descriptions☆

Shawn Bowers[a,*], Joshua S. Madin[b], Mark P. Schildhauer[c]

[a] *UC Davis Genome Center*
[b] *Dept. of Biological Sciences, Macquarie University, Australia*
[c] *National Center for Ecological Analysis and Synthesis, UC Santa Barbara*

Abstract

Formal ontlogies promise to provide many added advantages to ecological data management ... Most ontology editing frameworks and tools, however, are design for knowledge-representation experts and those with considerable understanding of the complexities of ontology languages (i.e., description logics). This paper proposes an approach for ontology creation based on ...

1. Introduction

The use of formal ontologies can significantly enhance metadata descriptions of ecological data by providing richer and more consistent terminologies [**?** **?** **?**]. Annotating data with ontology terms can both help users interpret data as well as enable advanced capabilities for data discovery and integration, e.g., by exploiting subsumption and part-of hierarchies as well as more formal constraints such as cardinality restrictions on properties and term equivalence [**?** **?**]. Many efforts are underway to engage scientists in the development of ontologies to represent common ecological and biodiversity concepts. Most of these projects leverage the W3C Web Ontology Language (OWL) as a standard XML syntax for representing and sharing ontologies. A key advantage of OWL is that it is supported by a wide range of generic tools, including editors [**?** **?**], reasoning systems [**?** **?**], query languages [**?**], and storage technologies [**?** **?**]. Most of these tools, however, are primarily targeted at experts in knowledge engineering and software development. This is especially true with ontology editors (such as Protege, OWLed, SWOOP, etc.), which allow for very detailed ontology specifications, but at the same time require a considerable amount of understanding of the underlying ontology formalisms and syntax. Thus, we see the lack of suitable ontology editing tools for scientists as one of the major barriers for more wide-scale adoption of ontologies in ecology.

Here, we present a novel approach for ontology creation that aims at being more intuitive for ecologists and that can be used to rapidly construct large

[*] Corresponding author

ontologies for describing scientific data. Our approach is to allow scientists to use common spreadsheet-based tools to describe, in an intuitive way, different aspects of an ontology, and then to take these descriptions and convert them into full-fledged OWL ontologies using a software application called Owlifier.

An Owlifier spreadsheet consists of a set of *blocks* that have a predefined template structure for users to fill in. Each non-empty row in an owlifier table constitutes a block, where blocks can be used to define ontology classes (including subclasses), class synonyms, data- and object-type properties (and constraints), etc. We also provide blocks for plain-text descriptions of concepts and properties, and for referencing one or more existing ontologies (e.g., to extend an existing ontology or to define ontology articulations). Blocks can be sparse (inheriting from previous blocks), which simplifies the creation of large ontologies.

Although not as expressive as full-fledged OWL ontologies, our approach can produce ontology structures essential for enhanced data discovery and integration, while at the same time providing a more accessible user interface for ecologists. Further, compared to many existing editors, our approach can be used to rapidly construct concept hierarchies, e.g., from long lists of keywords, within familiar spreadsheet software. We have field-tested the owlifier approach with ecologists and evolutionary biologists working with trait data, and found this application enabled them to quickly and easily comprehend and construct ontologies.

In this paper, we describe the syntax and semantics of Owlifier. We consider a set of blocks that generally follow the ...

Notes:

- we generally follow the guidelines in ontology creation of Rector et al. in [**?**].
- we also try to simplify certain aspects, e.g., disjoint classes and open-world reasoning, which can be confusing for non-experts
- we try to use more meaningful terms, when applicable (e.g., instead of datatype property, attribute, etc.)
- our goal is to not support all constructs in OWL-DL, but to provide the most commonly used ones for defining typical ontologies, but still allow experts to refine the ontology in ontology editing tools if necessary

2. Owlifier Syntax and Semantics

An Owlifier table defines an OWL-DL [**?**] ontology through a set of *blocks* representing one or more ontology definitions. Each non-empty row in an Owlifier table corresponds to a block. The type of the block is given in the first column of the row. We describe each type of block supported by Owlifier below. Here we assume that if any of the properties or concepts used in a block are defined, i.e., are not imported from another ontology, then they are added to the ontology being defined by the Owlifier table.

Import Block. Import blocks assign namespace labels to external ontologies. Each external ontology is imported into the current ontology. We refer to the ontologies of import blocks as *imported ontologies*. Using import blocks, classes and properties of imported ontologies can be used within other blocks of an Owlifier table. Import blocks take the form

`import`	N	U

where N is a namespace label and U is an OWL ontology URI. Classes and properties from imported ontologies are referenced by prefixing the namespace label N to the corresponding class or property name in the normal way. The following block imports the "earth realm" ontology from the set of SWEET ontologies [**?**]

```
import sweet http://sweet.jpl.nasa.gov/ontology/earthrealm.owl
```

where the ontology class denoting marine ecosystems can be refered to from within the corresponding Owlifier table using the syntax, e.g., `sweet:MarineEcosystem`. Because this class refers to a class in another ontology, we call it an "imported" class.

Concept Block. Concept blocks introduce new OWL classes and specify subclass relationships. Imported classes may also be used within a concept block by prefixing the class name with a namespace label, e.g., when an ontology is being extended. We use the term 'concept' to avoid confusion with the use of the term 'class' in biological taxonomies and because 'concept' is often more intuitive for users. Concept blocks take the form

`concept`	C_1	C_2	$\ldots$	C_n

$(n \geq 1)$

where each class C_i is asserted in the current ontology to subsume a class C_{i+1}, for $1 \leq i < n$. Each C_i in a concept block induces the DL axiom

$$C_i \sqsubseteq C_{i+1}.$$

If both C_i and C_{i+1} are imported classes, we say that the block defines an "articulation" (i.e., mapping) between the two classes. The following concept block defines a simple subclass hierarchy

```
concept PhysicalFeature AquaticPhysicalFeature River
```

which states that river is a subclass of aquatic physical feature and that aquatic physical feature is a subclass of physical feature.

Synonym Block. Synonym blocks define an equivalence relationship between classes. A synonym block has the form

`synonym`	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where each class C_i is asserted as being equivalent to class C_{i+1} in the current ontology, for $1 \leq i < n$. For example, the block

```
synonym Maize Corn
```

defines maize and corn as synonym (i.e., equivalent) classes.

Overlap Block

Except in certain situations (described further below), classes are assumed to be disjoint. Overlap blocks explicitly relax this assumption for a given set of classes by stating that a set of classes may have overlapping instances. An overlap block has the form

`overlap`	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where each class C_i is allowed to share instances with each concept C_j, for $1 \leq i, j \leq n$. In particular, C_i and C_j are not defined to be disjoint classes in the current ontology.

— an example here —

Relationship Block

Relationship blocks define required *object* properties of classes. A property block has the form

`relationship`	R	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where R is an object property and each C_i is a class, for $1 \leq i \leq n$. For every class C_i, the property block induces the DL axiom

$$C_i \sqsubseteq \exists R.C_{i+1}$$

stating that each instance of C_i has, amongst possibly other things, a relationship through R to some instance of C_{i+1}. The following example

need example here

Transitive Block

Transitive blocks state that a property is transitive. That is, if P is transitive and a concept instance O_1 is related to an instance O_2 by P, and O_2 is related to an instance O_3 by P, then O_1 is also by definition related to O_3 by P. A transitive block has the form

`transitive`	R	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where P is an object property. For example, the block `transitive hasPart Body Head Eye Retina`

states that a body has at least one head, a head has at least one eye, and an eye has at least one retina.

Attribute Block.. Attribute blocks are used to define the required *datatype* properties of concepts. An attribute block has the form

`attribute`	P	D	C_1	C_2	$\ldots$	C_n

$(n \geq 1)$

where P is a datatype property, each C_i is a concept for $1 \leq i \leq n$, and D is a datatype (`anyValueType`, `string`, `int`, etc.). For every concept C_i, the property block induces the DL axiom

$$C_i \sqsubseteq (\exists P.D)$$

stating that each instance of C_i has, amongst possibly other things, a relationship through P to a data value of type D.

Value Block.. Value blocks define required datatype property *constant values* for concepts. A value block has the form

`value`	P	V	C_1	C_2	$\ldots$	C_n

$(n \geq 1)$

where P is a datatype property, C_i is a concept for $1 \leq i \leq n$, and V is a datatype value. For each concept C_i, the value block induces the DL axiom

$$C_i \sqsubseteq (V \in P)$$

stating that each instance of C_i has a value V for property P. The value restrictions stated by value blocks are often used for defining so-called *value partitions* [**?**].

Inverse Block.. Inverse blocks state that two object properties are inverses of each other. That is, for inverse properties P_1 and P_2 and concept instances O_1 and O_2, if $P_1(O_1) = O_2$, then $P_2(O_2) = O_1$. An inverse block has the form

`inverse`	P_1	P_2

where P_1 and P_2 are object properties.

Minimum Block.. Minimum blocks state the minimum number of properties P an instance of a concept may have. Minimum blocks have the form

`minimum`	P	N	C_1	C_2	$\ldots$	C_m

where N is the minimum number of properties P that instances of concept C_1 may have to instances of concept C_2, C_2 to C_3, and so on. A cardinality block induces the DL axiom

$$C_i \sqsubseteq (\leq NP.C_{i+1})$$

stating that each instance of C_i must be related to at least N unique instances of C_{i+1} via P. For example, the blocks

```
minimum hasPart 1 Body Head
minimum hasPart 2 Head Eye
```

states that a body has at least one head and at least two eyes.

Maximum Block.. Maximum blocks state the maximum number of properties P an instance of a concept may have. Maximum blocks have the form

`maximum`	P	N	C_1	C_2	$\ldots$	C_m

where N is the maximum number of properties P that instances of concept C_1 may have to instances of concept C_2, C_2 to C_3, and so on. A cardinality block induces the DL axiom

$$C_i \sqsubseteq (\geq N P.C_{i+1})$$

stating that each instance of C_i may be related to at most N unique instances of C_{i+1} via P. For example, the blocks

```
maximum hasPart 1 Body Head
maximum hasPart 2 Head Eye
```

states that a body has at least one head and at least two eyes.

Sufficient Block.. Sufficient blocks state that any instance having a property P to an instance of a concept C_2 is a sufficient condition for being an instance of a concept C_1. A sufficient block has the form

`sufficient`	C_1	P	C_2

where C_1 is the target concept (i.e., denoting the concept definition), P is the sufficient property, and C_2 is the sufficient concept. A sufficient block induces the DL axiom

$$C_1 \equiv \exists P.C_2$$

Sufficient blocks provide a mechansism to construct simple class definitions (i.e., classes defined precisely by other classes), primarily for use with value partitions. [NOTE: these should be anded together?]

Description Block.. Description blocks assign plain-text definitions to concepts and properties. A description block has the form

`description`	T	S

where T is either a property or a concept and S is a description string.

Note Block.. Note blocks add comments to the current ontology, and are ignored by Owlifier. A note block has the form

`note`	S

where S is a comment string.

*** Say something about relaxing block syntax ... to make it easier to specify ontologies. Also, allow blocks to be given in any order.

3. Owlifier Properties and Reasoning

Some desirable properties:

- non-ambiguous (no block leads to ambiguous DL axioms)
- “reasonable” (not everything has to be explicitly stated)
- ???

4. Owlifier Implementation

Flags:

- delimeter characters
- perform classification
- warnings
- owlifier to owl and owl to owlifier
- ???

5. Conclusion

References