InfixOWL: An Idiomatic Interface for OWL

Chimezie Ogbuji, Cleveland Clinic Foundation
Heart and Vascular Institute <ogbujic@ccf.org>

Abstract

The Ontology Web Language (OWL) provides a powerful framework for describing the semantics and constraints of a particular domain described in RDF. Along with that power comes the burden of a learning curve that is often inherited by the tools used to manage OWL ontologies. The Manchester OWL syntax introduced an intuitive syntax for the presentation and editing of OWL ontologies that attempted to address some of this burden for the benefit of authors. One of the major design decisions of this syntax was the adoption of an infix notation. The primary contribution of this paper is a native, idiomatic API for constructing and manipulating OWL in Python that builds on the strengths of the Manchester OWL syntax. It is called InfixOWL.

Introduction

The Ontology Web Language is a Description Logic (DL) dialect for the Semantic Web [SW]. DLs are decidable subsets of First Order Logic that have a tree-like model [DL2OWL]. When modeling reality in the classical paradigm [SW-Modeling], the domain being modelled is abstractly represented as a set of objects with relationships between them.

Ontologies consist of sets of axioms that describe required characteristics of the domain expert's understanding of the world. An ontology corresponds to a logical theory. This requires a certain level of sophistication in the knowledge representation that can be problematic for editing tools and the readability of OWL document.

Readability of OWL

Generally, the canonical serialization of OWL (RDF/XML) suffers from the same impedance to readability that RDF graphs do. RDF/XML has been shown by many to be cumbersome to manage and read. The standard representational forms for OWL are not suitable for writing class expressions are either too verbose, or too complicated. [Man-OWL].

OWL can be considered a special-purpose language for modeling and reasoning about reality in a computationally tractable and expressive way. Given the nature of OWL as a language framework, it is very important that a reader of OWL syntax find the forms easily to understand without any disturbance from *syntax*. The concrete syntax used to capture OWL needs to make use of language idioms in order for them to be intuitive to domain experts.

Readability of Manchester OWL

The Manchester OWL syntax was designed as a simple, readable text format for expressing complex class descriptions. It was primarily intended for presenting such descriptions to (non-logician) human users. The focus was on allowing even relatively complex expressions to be readable as natural (English) language. Another major goal for this syntax was readability and a reduction in the amount of time it takes domain experts to understand class descriptions. A significant design decision for the Manchester OWL syntax

was the adoption of an infix notation for keywords used in restrictions. This was in order to directly combat the problem of non-logicians misreading class expressions [Man-OWL]. The use of infix notation with role restrictions is a powerful idiom for describing the relationships between concepts in a tree-like model.

These were the main motivations for adopting the Manchester OWL methodology in creating *InfixOWL*.

Agility of dynamic languages

A Python API for OWL has the distinct advantage of the dynamic nature of the host language. Python and other similar dynamic languages such as Ruby are very agile and have a rich set of natural keywords for operator and list management. As we will find later in this paper, this lends itself well as a framework for boolean and role restriction descriptions. There is a progressive shift towards the use of dynamic languages within the larger domain of informatics and information technology due to the abundance of scenarios where computing needs to be ubiquitous, pervasive, and highly dynamic [RPYTHON].

Python is a very expressive dynamic language. Even useful subsets of the full language such as RPYTHON are more expressive than either C# or Java. This expressive gap includes features such as: limited support for mixins, first-order function and class values, limited use of bound methods, and metaclasses [RPYTHON].

An executable exchange format

The use of natural language keywords affords the Manchester OWL syntax a certain amount of latitude with copying and pasting expressions into email and other such correspondence without requiring any special fonts for representing mathematical constructs. This makes it an ideal exchange syntax for OWL authoring tools, semantic web software agents, and OWL engineers. However, a Manchester OWL syntax parser (such as the one that comes with Protege) is needed in order to process such exchanges.

One of the primary goals of *InfixOWL* is to allow the use of Python as an interpreter for a concise, abstract OWL syntax that is both *executable* and readable. The typical use case would be one where a domain expert includes a section of *InfixOWL* expressions in some correspondence such that a recipient can re-use the excerpt directly into their application.

An alternative goal of this software module could have been to allow the exchange of Manchester OWL expressions as strings within the host language. In such an arrangement, a parser could be used to interpret the string directly rather than rely on programming interfaces to construct OWL expressions. However, the primary advantage of a programmatic interface is in facilitating the dynamic composition of OWL expressions that could possibly rely on the results of other processing performed in Python as well. For example, an OWL editor (written in Python) could use InfixOWL to build the underlying OWL graph using data associated with various user interface widgets. Without a programmatic interface for constructing the OWL expressions in this fashion, the data would have to be directly substituted into sections of Manchester OWL syntax strings.

For the purpose of a consistent example throughout this paper, we will use excerpts from the Open GALEN terminology system [GALEN]. The experience gained from the development of a large Description Logic terminology system such as GALEN was a significant influence in the development of the Manchester OWL syntax.

Consider the class definitions for *CongenitalAtrialSeptalDefect and AtrialSeptalDefect*:

```
class: CongenitalAtrialSeptalDefect
EquivalentTo:
   AtrialSeptalDefect that hasAcquisitionMode some
   ( AcquisitionMode that hasAbsoluteState value congenital )
```

```
class: AtrialSeptalDefect
EquivalentTo:
   PlanarDefect that hasSpecificLocation some InteratrialSeptum

class: PlanarDefect
SubClassOf:
   NonnormalBodyCavity that
   hasTopology some ( Topology that hasAbsoluteState value tubular )
```

The expression above makes use of the more common forms of Description Logic expressions in large medical record terminology systems such as GALEN: boolean class constructors, role restrictions, and subsumption axioms. The following InfixOWL excerpt demonstrates the programmatic construction of these classes:

```
from FuXi.Syntax.InfixOWL import *
from rdflib import BNode, URIRef, Literal, Namespace

GALEN = Namespace('http://www.co-ode.org/ontologies/galen#')

#Properties
hasTopology         = Property(GALEN.hasTopology)
hasAbsoluteState    = Property(GALEN.hasAbsoluteState)
hasSpecificLocation = Propert(GALEN.hasSpecificLocation)
hasAcquisitionMode  = Property(GALEN.hasAcquisitionMode)
hasAbsoluteState    = Property(GALEN.hasAbsoluteState)

#Classes
AcquisitionMode     = Class(GALEN.AcquisitionMode)
InteratrialSeptum   = Class(GALEN.InteratrialSeptum)
PlanarDefect        = Class(GALEN.PlanarDefect)

PlanarDefect.subClassOf =
    [Class(GALEN.NonnormalBodyCavity),
     hasTopology|some|(Class(GALEN.Topology) &
          hasAbsoluteState|value|GALEN.tubular)]

ASD = PlanarDefect &
      hasSpecificLocation|some|InteratrialSeptum
ASD.identifier = GALEN.AtrialSeptalDefect

CASD = AtrialSeptalDefect &
       hasAcquisitionMode|some|
       (AcquisitionMode & (hasAbsoluteState|value|GALEN.congenital)
CASD.identifier = GALEN.CongenitalAtrialSeptalDefect
```

InfixOWL

Contrasting the *InfixOWL* excerpt with the Manchester OWL version emphasizes some interesting differences. The Manchester OWL version is much more concise and readable. The difference is primarily due to its being an independent language. *InfixOWL*, however, is a subset of Python and thus requires certain Python constructs.

Intuitively, any Manchester OWL parser would rely on knowledge of operator precedence and the controlled EBNF for a form of *duck-typing [TEACHING_PYTHON]*. Duck-typing is a dynamic-typing

style in which the type associated with an object is determined by its use rather than by an explicitly assigned type. Inferring an object type from context is a standard programming language best practice [TYPING] . As a result, the naming convention that emphasizes the difference between *AtrialSeptalDefect* and *hasSpecificLocation* (the former starts with a capitalized letter while the latter starts with a lower case letter) is redundant due to the fact that the first is an OWL class and the second is an OWL object is evident from their use with the keywords **that** and **value**.

On the other hand, InfixOWL requires that OWL classes and properties be instantiated unless they are used in expressions that automatically instantiate them. One of the reasons for this is to provide specific *bindings* to an underlying RDF graph. These bindings enforce, for example, the requirement that all OWL classes in an RDF graph have an *rdf:type* assertion with a value of owl:Class in the graph. A Manchester OWL syntax parser that was also responsible with converting the parsed text into statements in an RDF graph would need to maintain such bindings as well.

InfixOWL defines a handful of Python classes for the core components of the OWL abstract syntax: *Class*, *Property*, *Individual*, *BooleanClass*, *Restriction*, etc. Each of these classes define accessors methods, override behavior of use with python operators, and implement other functionality specific to the abstract syntax forms they represent.

OWL class accessors methods

Looking again at the first example of accessing an instantiated OWL class in the excerpt above, we notice the *subClassOf* accessors. As the name suggests, this associates a class with the parent classes that subsume it. It is meant as a proxy for an RDF assertion with rdfs:subClassOf as the predicate. When this attribute is set, as it is in the example above, the argument on the right is expected to be a list of Class instances (each of which is an OWL class description).

```
PlanarDefect.subClassOf =
    [Class(GALEN.NonnormalBodyCavity),...]
```

The effect of setting this attribute in this way is the modification of the underlying OWL RDF graph such that all *rdfs:SubClassOf* axioms for the given class are replaced with the given list of class descriptions. The **FuXi.Syntax.InfixOWL.Class** Python class defines accessors for each of the major sections of the full class description syntax for Manchester OWL: *subClassOf*, *equivalentClass*, and *disjointWith*. The corresponding axioms in the OWL abstract syntax can be used with multiple class descriptions, so each accessors is expected to take a list of Class objects when set and return an iterator of Class objects otherwise.

However, there are other accessors that take and return singular Python objects. The excerpt above demonstrates usage of the *identifier* accessor, which is used to set and retrieve the singular URI reference or Blank Node associated with an OWL class or property:

```
ASD.identifier = GALEN.AtrialSeptalDefect
```

Typically, **FuXi.Syntax.InfixOWL.Class** instances are created with the URI for the OWL class as the first argument to the constructor. Instantiating Class instances without any arguments to the constructor will create an anonymous class description. The *identifier* accessor is useful for giving identifiers to InfixOWL class descriptions that return anonymous class instances such as in the case above. Later on, we will cover the construction of boolean classes.

Below is a list of the other accessors available to Class instances:

1. *comment* (manages *rdfs:comment* assertions)

2. *type* (manages *rdf:type* assertions)

3. *label* (manages *rdf:label* assertions)

All of these accessors can be set to either single terms or a list of terms. The *comment* and *type* accessors return a generator over RDF literals that stand in the *rdfs:comment* or *rdf:type* relation (respectively) to the class object. The *type* accessors returns a generator over all the terms that stand in the *rdf:type* relation to the class object.

Boolean class constructors

The other parts of the InfixOWL excerpt demonstrate how InfixOWL can be used to construct boolean class descriptors.

```
ASD = PlanarDefect &
      hasSpecificLocation|some|InteratrialSeptum
```

The **unionOf**, **intersectionOf**, and **complementOf** OWL DL class descriptions are implemented using native Python operators. In particular, |, **&**, and ~ Python operators can be used to construct corresponding InfixOWL class descriptions that implement bindings with the underlying RDF graph such that the OWL/RDF assertions mirror the axioms in the class descriptions. In the section above, the **&** operator is used to construct an (anonymous) class descriptor for the conjunction of the *PlanarDefect* class and an existential restriction.

Overriding the 'in' Idiom in Python

OWL uses RDF lists to represent the ordered list of arguments to a boolean class description. Python has a set of natural list that can be be easily overridden. InfixOWL takes liberties with this feature in implementing its idioms. Any instance of a conjunctive or disjunctive boolean class description (unionOf or intersectionOf) can be accessed as a Python list. For example, the **in** operator can be used to iterate over the operands:

```
for booleanOperand in booleanClassInstance:
    .. perform some operation ..
```

Overriding the 'in' Idiom in Python

A Python recipe [INFIX] was incorporated in order to implement the set of infix operators that comprise the Manchester OWL syntax. The following infix operators (each of which is invoked by using the keyword in between a set of '|' operators) are defined using this recipe:

1. *some*
2. *only*
3. *max*
4. *min*
5. *exactly*
6. *value*

Role restrictions

Each of the InfixOWL operators above is associated with an anonymous method that constructs a **FuXi.Syntax.InfixOWL.Restriction** instance. This is how existential and universal restrictions are constructed. For instance, in the example above, the *AtrialSeptalDefect* class description is constructed

using the conjunction of a *PlanarDefect* and an existential restriction on the *hasSpecificLocation* role. The restriction is constructed using the |some| operator.:

```
ASD = PlanarDefect &
      hasSpecificLocation|some|InteratrialSeptum
```

The cardinality restriction operator expects rdflib.Literal objects. These can be easily constructed by passing them appropriate Python native objects (such as numbers) to the constructor:

```
from rdflib import Literal
ASD = PlanarDefect &
      hasSpecificLocation|min|Literal(1)
```

FuXi.Syntax.InfixOwl.Restriction classes define the following convenient accessors for the attributes associated with the role restrictions in the abstract OWL syntax:

1. *onProperty*
2. *allValuesFrom*
3. *someValuesFrom*
4. *hasValue*
5. *cardinality*
6. *maxCardinality*
7. *minCardinality*

Subsumption

All class objects can use the `+=` Python operator to capture axioms that represent subsumption. In particular, any instance that is invoked with that operator will assert that the right operand (also a class object) is subsumed by the first class (i.e.: *rightOperand* rdfs:subClassOf *leftOperand*)

So, could have been defined in this way instead:

```
Class(GALEN.NonnormalBodyCavity) += PlanarDefect
(hasTopology|some|(Class(GALEN.Topology) &
 hasAbsoluteState|value|GALEN.tubular)) += PlanarDefect
```

Also, every Class instance has a *subSumpteeIds* method which returns a generator of all the identifiers for the OWL classes that stand in the *rdfs:subClassOf* RDF relation to the instance. Specifically, only the identifiers of the classes (explicitly) subsumed by the class description are returned.

Creating and managing OWL properties

InfixOWL also includes a Python class called **FuXi.Syntax.InfixOwl.Property** for creating and modifying OWL properties in a similar fashion as Class objects. Objects instantiated from this Python class will have the following set of additional accessors available to them:

1. *subPropertyOf (owl:subPropertyOf)*
2. *inverseOf (owl:inverseOf)*
3. *domain (rdfs:domain)*

4. *range (rdfs:range)*

Intuitively, the accessors above can be used to specify an RDF relationship between the property on the left hand side and another property or class (depending on the accessor). The QNames in parenthesis indicate the predicate of the corresponding relationship.

The common accessors introduced in section 6.1 (*type*, *label*, and *comment*) are also available to Property objects. The type accessors can be used to specify whether an OWL property is transitive, functional, symmetric, etc. This can be achieved by setting this accessors to one or more of the corresponding OWL terms on Property objects.

Round-tripping OWL axioms in Python

Finally, Python allows classes to override their **__repr__** method in order to customize how they are serialized when printed. During the development of InfixOWL and the construction of tests using the *doctest* module, this was an incredibly useful feature. Browsing the InfixOWL.py module, the reader will find executable, document strings such as the one below that serve as test cases:

```
>>> c =  a | b | Class(exNs.Work,graph=g)
>>> c
( ex:Opera or ex:CreativeWork or ex:Work )

.. snip ..

>>> del c[c.index(Class(exNs.Work,graph=g))]
>>> c
( ex:Opera or ex:CreativeWork )

.. snip ..

>>> contList = [Class(exNs.Africa,graph=g),Class(exNs.NorthAmerica,graph=g)]
>>> EnumeratedClass(members=contList,graph=g)
{ ex:Africa ex:NorthAmerica }
```

This is also very useful when building modules that manipulate OWL RDF graphs. Often, an author of such a module would like to quickly serialize a class description for debugging purposes. InfixOWL adopts the Manchester OWL style as a framework for a Python API for the construction of OWL class descriptions. However, it also overrides the serialization of human-readable string representations of InfixOWL Class objects. In particular, Manchester OWL syntax is used as the official string representation of InfixOWL Class objects.

Discussion

We have shown how InfixOWL builds on the strengths of the Manchester OWL syntax in implementing a powerful Python API for managing abstract OWL RDF graphs using the *RDFLib* open-source Python library. The Manchester OWL style is adopted faithfully by leveraging Python's support for overriding operator behavior. The combination of the agility of the host language and the concise, readable style of this popular OWL syntax results in a very idiomatic Python library for OWL ontology engineering.

This powerful combination also resulted in an intuitive way to document and test Python modules (like InfixOWL) that manipulate or manage an OWL RDF graph. Using Manchester OWL as the official string representation of InfixOWL class instances can also be useful for the quick generation of static documentation for one or more OWL documents.

Additional duck-typing

As mentioned previously, InfixOWL currently requires the explicit instantiation of at least the primitive OWL class objects in an ontology as it is being constructed. However, this impedance can be partially addressed by additional functionality such as extending the *RDFLib* Namespace class with a specialization that constructs Class objects instead. These would essentially be used as a convenient method for generating InfixOWL class descriptions that share a common base URI:

```
GALEN      = ClassFactory('http://www.co-ode.org/ontologies/galen#')
GALEN_ABOX =    Namespace('http://www.co-ode.org/ontologies/galen#')

GALEN.PlanarDefect.subClassOf =
    [GALEN.NonnormalBodyCavity,
     hasTopology|some|(GALEN.Topology &
              hasAbsoluteState|value|GALEN_ABOX.tubular)]
... snip ...
```

Supporting additional infix operators

Finally, the incorporated Python recipe can also be used to implement additional infix operators. This could be as simple as adopting the **that** operator from Manchester OWL or adopting operators for the design patterns [Man-OWL] that are included in the Manchester OWL syntax: *onlySome* and *ValuePartition*,

Using InfixOWL for other logic forms

Another motivation for InfixOWL was the use of Python to generate definite Logic Programs from OWL RDF graphs using the Description Logic Program knowledge representation (DLP) [DLP]. InfixOWL is part of a larger software project called *python-dlp* [PYTHON-DLP] that includes a forward-chaining production system, an implementation of the DLP algorithm, and the InfixOWL module.

The DLP algorithm provides a direct mapping from a subset of Description Logic into an equivalent Logic Program. InfixOWL can be used as a framework for composing OWL RDF graphs and converting them into a customized ruleset using the DLP implementation. The ruleset can then be used to calculate entailments of an RDF graph with respect to the OWL expressions. In some cases, it is more intuitive to build a ruleset in this way.

Conclusion

InfixOWL is distributed as a module and is implemented as a companion to *RDFLib* – which it requires for its various RDF processing. It can be downloaded from http://code.google.com/p/python-dlp/ or installed using setuptools:

```
easy_install FuXi
```

This paper describes some of the design considerations that motivated the creation and development of InfixOWL. The result was an expressive, agile syntax for constructing, manipulating, and serializing OWL RDF axioms in an *RDFLib* graph.

Bibliography

[PYTHON-DLP] http://code.google.com/p/python-dlp/

[N3Logic] Berners-Lee, T - Connolly, D - Kagal, L - Scharf, Y - Hendler, J: N3Logic: A Logical Framework For the World Wide Web. 2007. To appear in Theory and Practice of Logic Programming http://arxiv.org/abs/0711.1533v1

[SW-Modeling] Patel-Schneider, Peter Horrocks, I: Position Paper: A Comparison of Two Modelling Paradigms in the Semantic Web . 2006. http://www2006.org/programme/item.php?id=4015

[DLP] Grosof, B – Volz, R - Horrocks, I - Decker, S: Description Logic Programs: Combining Logic Programs with Description Logic. 2003. http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/p117-grosof.pdf

[GALEN] Rector, A. - Solomon, W. - Nowlan, W. - Rush, T.: A terminology server for medical language and medical information systems. Methods of Information In Medicine 34 (1994) 147–157

[Man-OWL] Horridge, M. - Drummond, N. - Goodwin, J. - Rector, A. Stevens, R. - Wang, H.: The Manchester OWL Syntax. 2007. Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions

[RPYTHON] Meijer, E. - Drayton, P.: RPython: a Step Towards Reconciling Dynamically and Statically Typed OO Languages. 2007. Dynamic Languages Symposium. Proceedings of the 2007 symposium on Dynamic languages.

[TYPING] E. Meijer - P. Drayton.: Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. 2004. Proceedngs of OOPSLA'04 Workshop on Revival of Dynamic Languages.

[TEACHING_PYTHON] Goldwasser, M. - Letscher, D: Teaching an Object-Oriented CS1 — with Python. 2007. Journal of Computing Sciences in Colleges. Volume 22 , Issue 4. Pages: 62 - 64.

[SOF] Shearer, R : Structured Ontology Format. 2007. Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions.

[DL2OWL] Horrocks, I - Patel-Schneider, P - Harmelen, F: From SHIQ and RDF to OWL: The Making of a Web Ontology Language., 2003. Journal of Web Semantics. http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPH03a.pdf

[SW] Berners-Lee T- Hendler J - Lassila O: The Semantic Web, May, 2001. Scientific American.

[INFIX] Jamitzky, F.: Infix operators, February 2005. http://code.activestate.com/recipes/384122/ [http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPH03a.pdf].