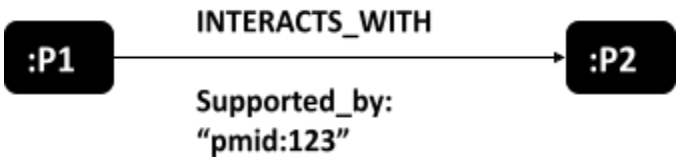
Author: Chris Mungall
Contributors: *AZ, <add names here>*
Type: Position Paper, blog
Status: Final
Projects: OBO
Audience: ontology engineers, semweb developers
Keywords: reification, singleton properties
Created: 2019
Updated: 2019
Repo:
See also:

One of the ways in which RDF differs from Property Graph (PG) models such as the datamodel in Neo4J is that there is no first-class method for making statements about statements. For example, given a triple *:P1 :interacts-with :P2*, how do we say that triple is supported by a particular publication? With a PG, we can attach a property to the edge

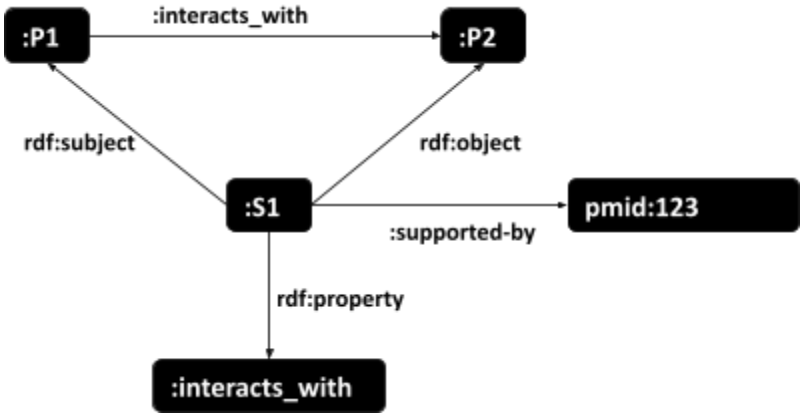


(although in some graphj database such as Neo4J there are limitations on this - the property can only indirectly refer to a publication node through an identifier). In RDF, properties cannot be directly associated with edges.

In RDF, a common approach is reification. We would create an extra node representing the statement, associate this with the original triple via three new triples, and then the statement node can be described as any other node. E.g.

- :P1 :interacts_with :P2
- :S1 a rdf:Statement
- :S1 rdf:subject :P1
- :S1 rdf:predicate :interacts_with
- :S1 rdf:object :P2
- :S1 :supported by pmid:123

This can be depicted visually as follows (note that while the first triple directly connecting P1 and P2 may seem redundant it is not formally entailed by RDF semantics and should be stated):



RDF* provides a more convenient compact syntax for writing edge properties:

- <<:P1 :interacts_with :P2>> :supported_by :pmid123 .

Interpreting this as reified triples is one of many possibilities in RDF*.

However, not everyone uses RDF reification. OWL has an analogous structure but uses a different vocabulary. Wikidata has their own n-ary relation pattern. Some prefer the use of Named Graphs.

One alternative that has been proposed is the Singleton Property Pattern (SPP), described in *Don't Like RDF Reification? Making Statements about Statements Using Singleton Property* (2015), Nguyen et al (PMC4350149). The idea here is to mint a new property URI for every statement we wish to talk about. Given a triple *S P O*, we would rewrite as *S new(P) O*, and add a triple *new(P) singlePropertyOf P*.

For example, for our protein interaction example above, we would represent as:

- :P1 :interacts_with_001 :P2
- :interacts_with_001 singletonPropertyOf :interacts_with
- :interacts_with_001 :supported_by :pmid123

To find all interacts-with associations, we could write a SPARQL query such as:

```
SELECT ?x ?y WHERE { ?x ?p ?y . ?p singletonPropertyOf :interacts_with }
```

Some of the purported advantages of this scheme are discussed in the paper.

A variant of the SPP makes the new property a subPropertyOf the original property. I will call this SPP(sub). This has the advantage that the original statement is still *entailed*.

- :P1 :interacts_with_001 :P2
- :interacts_with_001 rdfs:subPropertyOf :interacts_with
- :interacts_with_001 :supported_by :pmid123

Then if we assume RDFS entailment, the query becomes simpler:

```
SELECT ?x ?y WHERE { ?x :interacts_with ?y }
```

This is because the direct interacts_with triple is entailed due to the subPropertyOf axiom.

In the discussion section of the original paper the authors state that the pattern is compatible with OWL, but don't provide examples or elucidation, or what is meant by compatible. There is a website that includes an OWL example.

Unfortunately, problems arise if we want to use this same pattern with other axiom types, beyond OWL Object Property Assertions; for example, SubClassOf, EquivalentClasses, SameAs. This is true for both the original SPP and the subproperty variant form.

These problems arise when working in OWL2-DL. Use of OWL2-Full may eliminate these problems, but most OWL tooling and stacks are built on OWL2-DL, so it is important to be aware of the consequences.

OWL-DL has major problems with the SPP

The reason for the incompatibility is that the pattern only works if the property used in the statement is an actual OWL property. In the official OWL2 mapping to RDF, predicates such as owl:equivalentClass, owl:sameAs, owl:subClassOf are syntax for constructing an OWL axiom of the corresponding type (respectively EquivalentClasses, SameIndividual and SubClassOf). **These predicates have no corresponding properties at the OWL level**.

For example, consider the case where we want to make an equivalence statement between two classes, and then talk about that statement using the SPP pattern.

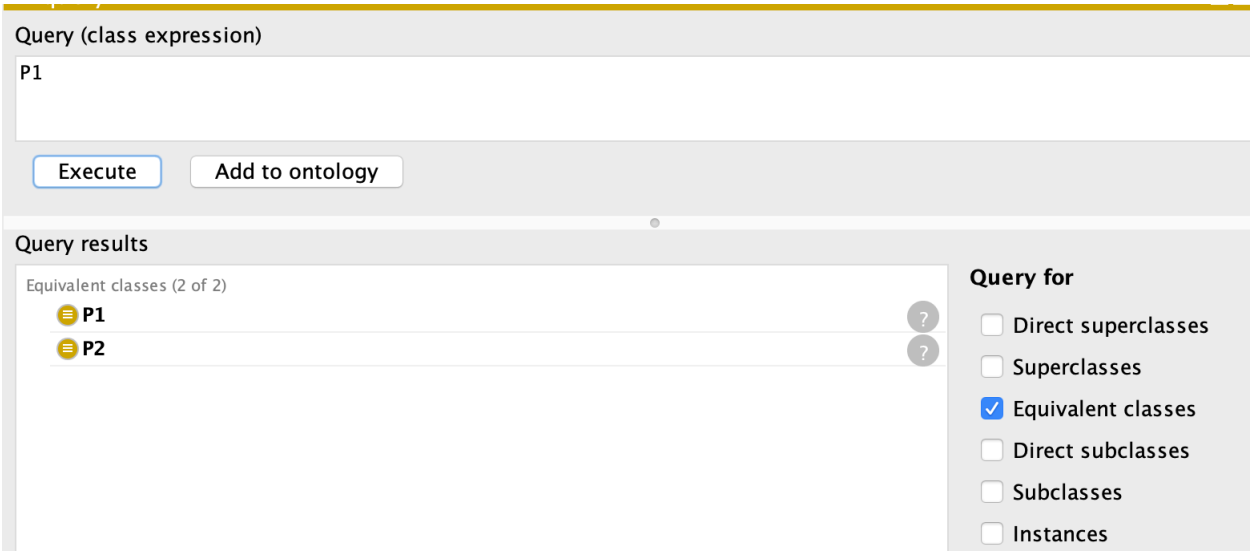
We do this by making a fresh property (here, arbitrarily called *equivalentClass-1*) that is the SPP form of owl:equivalentClass. For comparison purposes, we make a normal equivalence axiom between P1 and P2, and we also try an equivalence axiom between P3 and P4 using the SPP(sub) pattern:

```
PREFIX : <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

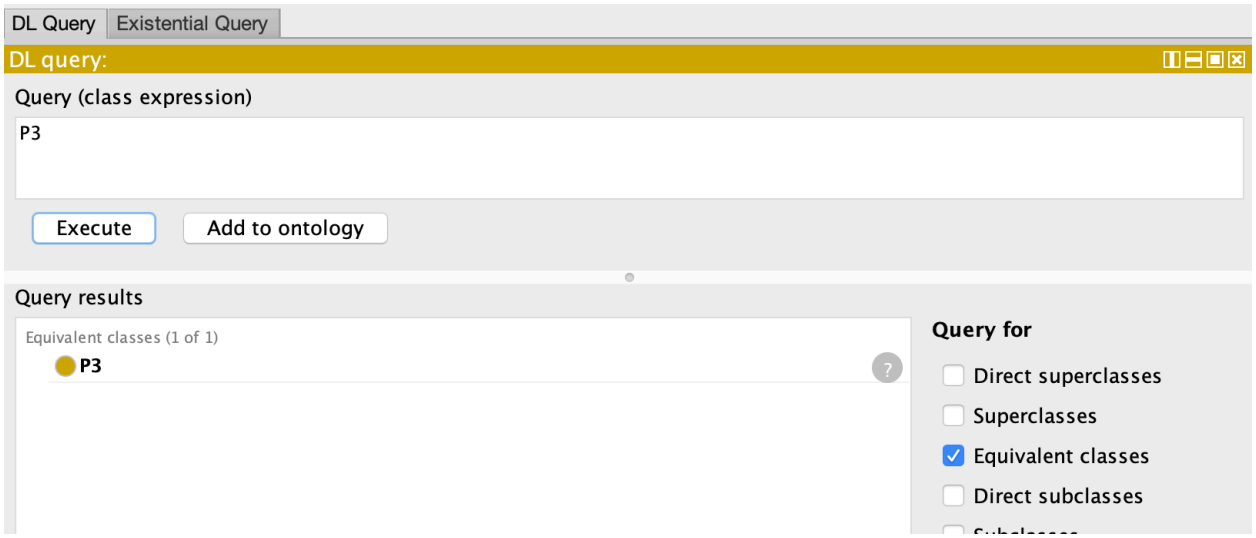
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

:P1 a owl:Class .
:P2 a owl:Class .
:P3 a owl:Class .
:P4 a owl:Class .

:P1 owl:equivalentClass :P2 .
:P3 :equivalentClass-1 :P4 .
:equivalentClass-1 rdfs:subPropertyOf owl:equivalentClass .
```

Note that this does not have the intended entailments. We can see this by doing a DL query in Protege. Here we use HermiT but other DL reasoners exhibit the same features.

For the normal equivalence axiom, we get expected entailments (namely that P2 is equivalent to P1):



However, we do not get the intended entailment that P3 is equivalent to P4



We can get more of a clue as to what is going on by converting the above turtle to OWL functional notation, which makes the DL assertions more explicit:

Prefix(:=<http://example.org/>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)

Ontology(
Declaration(Class(:P1))
Declaration(Class(:P2))
Declaration(Class(:P3))
Declaration(Class(:P4))
Declaration(AnnotationProperty(:equivalentClass-1))
Declaration(**AnnotationProperty(owl:equivalentClass)**)
##############################
Annotation Properties
##############################

Annotation Property: :equivalentClass-1 (:equivalentClass-1)

SubAnnotationPropertyOf(:equivalentClass-1 owl:equivalentClass)

##############################

Classes
#############################

Class: :P1 (:P1)

EquivalentClasses(:P1 :P2)

Class: :P3 (:P3)

AnnotationAssertion(:equivalentClass-1 :P3 :P4)

)

Note that the SPP-ized equivalence axiom has been translated into an owl annotation assertion, which has no logical semantics. However, even if we force the SPP property to be an object property, we still lack the required entailments. In both cases, we induce owl:equivalentClass to be an annotation property, when in fact it is not a property at all in OWL-DL.

The SPP(e) pattern may work provide the intended entailments with OWL-Full, but this remains to be tested. Even if this is the case, for pragmatic purposes most of the reasoners in use in the life science ontologies realm are OWL2-DL or a sub-profile like EL++ or RL.

But RDF reification is also incompatible with OWL2! (sort-of)

At this stage it's worth pointing out OWL2 introduced it's own reification vocabulary (analogous to RDF reification) for RDF serialization.

What happens if we try to use RDF serialization with OWL axioms? It simply doesn't work. Triples are silently lost.

This is rather frustrating from a pragmatic engineering point of view, since there are now at least two reification standards to choose from, which complicates writing queries an simple data access code. While there is a semantic justification for this, I think this is a case where engineering concerns were ignored by the ontologists. It is also very frustrating that the statement/axiom in OWL is **forced** to be a blank node. Why? Why can't I give a URI to my axiom? This is a dumb imposition.

On the difficulty of mentally reasoning over W3C specifications

This blog post arose out of an enjoyable discussion during beers at the NCATS Data Translator hackathon in Seattle in September 2019 between myself, Michel Dumontier, and Vincent Emonet. We were discussing which reification pattern to use, and whether or not SPP(e) was compatible with OWL.

We were pulling up different W3 specs on our phones to support or our respective positions. All of us are seasoned semantic web developers, well-versed on OWL and RDF semantics. However, we neither side could provide definitive proof that their position was justified. The mapping to RDF spec (which is fairly complex) states in section 3.2.5 that triples such as *?x rdfs:subClassOf ?y* are removed after it has been translated to an OWL axiom. However, following the RDF-based semantics seems to imply that SPP(e) should work since everything is layered on RDF. I think this means that it's incompatible with OWL2-DL, yet compatible with OWL2-Full, but I lack full confidence.

We ended up pulling out a laptop and coding some turtle (see above) that showed that at least for existing OWL reasoners, SPP-izing what was an owl:equivalentClass triple at the RDF level rendered it invisible to Protege and OWL reasoning. So from a pragmatic point of view, this is a disadvantage of SPP for those of us that rely on OWL (of course, there are workarounds such as doing an pre-processing step of re-rendering the original axioms…). Still, this is a bit unsatisfying as we would like the specs to answer us clearly and unambiguously.

Sometimes W3C specs are criticized for being difficult for non-formal CS people to understand, but perhaps even harder is the challenge of mentally reasoning over how they compose together. I'm not sure what the answer is, given that at least in the life science world we don't want to throw out richer semantics such as OWL, as it buys us so much. Perhaps an alternate layering of semantics on RDF* could help here (see my post https://douroucouli.wordpress.com/2019/07/11/proposed-strategy-for-semantics-in-rdf-and-property-graphs/).

Other benefits and disadvantages of the different reification schemes

I only intend to elucidate OWL aspects of the singleton pattern here, as I don't believe this has been done before. The OWL incompatibility may not be an issue for your use case, and you may like other aspects of the SPP, if so, go for it. There may be other gotchas though. The wikidata paper suggests performance degradation in some triplestores. This is perhaps not unexpected, as an implicit unwritten assumption in RDF is that the number of properties will be relatively low, reflecting 'natural' predicates, and it's not expected that the number of properties will grow to the size of the number of triples, and indexing schemes may have made this assumption.

Lessons learned?

I consider it hugely unfortunate that something as simple as being able to make statements about a statement is such a complex minefield using the semantic web stack. There are a myriad of schemes to choose from, the implications of choosing one over another are generally not well understood by practictioners, and the semantics are not even clear to experts, with multiple hidden traps. It's no wonder that many developers avoid the W3 stack and opt for something simpler to use such as a property graph implementation e.g neo4j.

Fortunately, I think this is something that is fixable by rethinking how we do things, treat property graphs as first class citizens, and have a more natural and intuitive layering of semantics on top of property graphs. I think RDF* represents an excellent basis for merging these two worlds.