Macro expansion for OWL

Chris Mungall, Alan Ruttenberg, David Osumi-Sutherland, ...

Abstract. Accurate representation of complex domains such as biomedicine demands powerful and expressive ontology languages such as OWL. However, the complex nested class expressions required for modeling can be a hindrance to ontology authoring and adoption. These class expressions can appear opaque to domain experts, and even users proficient in OWL can benefit from some kind of syntactic sugar or "short-cut" strategy, especially when authoring large ontologies.
One solution is to have domain experts fill in simple templates (for example, in Excel) and translate the results into more complex axioms, but this has the disadvantage of being disconnected from full ontology authoring and reasoning environment.
We present here a method of specifying shortcut properties directly in OWL. These shortcut properties can be used in similar ways as object properties within the OWL environment, with the resulting simple axioms translated automatically to more complex axioms via macro expansion. We describe some example scenarios where this is of use in authoring existing bio-ontologies.
One of the main implications of this work is a way to simplify the translation between OBO format and OWL, and the use of RDF triple-stores with complex OWL ontologies.

1 Introduction

Accurate representation of complex domains such the biosciences demand powerful and expressive ontology languages such as OWL. However, the very expressive power of OWL can be a hindrance to widespread adoption and effective use - OWL axioms and class expression may not always mirror the internal cognitive models of domain experts.

In addition, the development of large ontologies often requires the repeated application of the same template pattern for different combinations of classes. In addition to being tedious and error-prone, the high-level pattern is not immediately apparent on viewing the ontology. Table 1 shows some example repeated patterns extracted from core bio-ontologies from the OBO Foundry registry[7].

To the seasoned ontologist accustomed to writing complex nested expressions, this may seem a trivial problem. It is not hard for a computer scientist to devise some "syntactic sugar" or intermediate representation, and have this translated behind the scenes to OWL. For example, on particular approach is to have domain experts fill in an Excel table, with columns corresponding to variables in the above, and to write a script to generate the OWL.

However, this approach can be unsatisfying for a number of reasons. A scripting approach is ad-hoc and difficult to integrate with existing OWL toolchains.

Source	Pattern	Example
CL	hasPart some ('plasma membrane' and hasPart ?Y)	Class: 'alpha-beta T cell' EquivalentTo: 'T cell' and hasPart some ('plasma membrane' and hasPart 'alpha-beta TCR complex')
CL	bearerOf some (realizedBy only ?Y)	Class: 'immune cell' EquivalentTo: 'cell' and bearerOf some (realizedBy only 'immune system process')
GO	(partOf some ?X) disjointFrom (partOf some ?Y)	(partOf some nucleus) disjointFrom (partOf some cytoplasm)
Fly	(?X partOf some ?Y) (?Y hasPart some ?X)	Class: retina SubClassOf: partOf some eye Class: eye SubClassOf: hasPart some retina
CL	hasPart exactly 0 ?Y	Class: 'enucleate cell' EquivalentTo: cell and hasPart exactly 0 nucleus
TAO	hasPart some (partOf some ?Y)	Class: 'vertebra 5' SubClassOf: hasPart some (partOf some 'V4-5 joint')
OBI	realizes some ('analyte role' and roleOf some ('scattered molecular aggregate' and hasGrain only ?Y))	Class: 'human antithrombin-III (AT-III) in blood assay' SubClassOf: realizes some ('analyte role' and (roleOf some ('scattered molecular aggregate' and (hasGrain some 'human antithrombin-III protein'))))

Table 1. Example generic patterns taken from various ontologies in the OBO library (http://obofoundry.org). CL = cell ontology, Fly = *Drosophila* anatomy, GO = Gene Ontology, OBI = Ontology for Biomedical Investigations.

In addition, the source intermediate representation is lost on translation into OWL – ideally the intermediate representation would still be visible within environments and tools such as Protege and Pellet.

Here we describe a macro expansion language that allows the representation in a high-level intermediate form directly in OWL. A generic simple macro-expansion tool can be used to translate back and forth between the intermediate form and the expanded form. This allows users to view and edit at either level, as appropriate.

We also discuss the usage of a macro expansion language versus using and extending existing OWL2 constructs such as property chains.

2 Macro Expansion Language

We propose a macro expansion language that can be represented directly within OWL-DL. The macro expansions are represented as OWL Manchester Syntax[6] with the addition of template variables. Either annotation properties or object properties are annotated with macros, and these can be used to expand individual axioms at any time during the ontology development or deployment cycle, preferably prior to reasoning. We refer to properties that are annotated in this way as "shortcut relations". Table 2 shows the two different annotation properties that can be used to define shortcut relations, and table 3 specifies how expansion occurs.

Annotation Property	**Usage**	**Domain**
expandAssertionTo	Expanding annotation assertion axioms connecting two classes	Annotation Property
expandExpressionTo	Expanding class expressions	Object Property

Table 2. Two annotation properties for describing shortcut relations. The first is intended to annotate annotation properties that are used to make property assertions about classes, and the second is to annotate object properties used in class expressions.

Annotation Property	**Matches**	**Expansion**
expandAssertionTo	PropertyAssertion(?R,?X,?Y)	substitute(?R,?X,?Y)
expandExpressionTo	?R some ?Y	substitute(?R,?Y)
expandExpressionTo	?R value ?Y	substitute(?R,?Y)

Table 3. Macro expansion of shortcut relations. If a match is found for $?R$, then the matching axiom or expression is substituted using the value of the annotation property for $?R$.

Table **??** shows some examples. We now describe these in more detail.

2.1 Macro Expansion of Annotation Properties

We propose an annotation property expandAssertionTo that specifies how to expand a single OWL PropertyAssertion (i.e. rdf triple) into a more complex axiom by substituting two variables $?X$ and $?Y$ with the subject and target of the assertion respectively.

The rule can be specified as:

```
PropertyAssertion(?R,?X,?Y) ==> substitute(?R,?X,?Y)
```

For example:

```
AnnotationProperty: spatially_disconnected_from
Annotations: expandAssertionTo
  "DisjointClasses: (part_of some ?X) disjointFrom part_of some ?Y)"
```

The following triple:

```
Class: nucleus
Annotations: spatially_disconnected_from cytoplasm
```

would expand to a general class inclusion (GCI) axiom:

```
DisjointClasses: (part_of some nucleus) disjointFrom (part_of some cytoplasm)
```

Note that this pattern does not work in all cases - for example, for defining classes using equivalence axioms. Here we want a generic way of expanding class expressions using shortcut relations, such that they can be used in different axiom types.

2.2 Macro Expansion of Object Properties

As an alternative that works for both SubClass and EquivalentTo axioms, we propose a different expansion rule for a new annotation property expandExpressionTo. Here, the macro describes how to expand a class expression, not a property assertion, and only uses a single variable.

The macro object property should be used in existential restrictions, and the translation rule is:

```
?R some ?Y ==> substitute(?R,?Y)
```

For example, the cell ontology defines a shortcut relation used for defining cell types based on the proteins expressed on the plasma membrane (i.e. the boundary of the cell)[5].

```
ObjectProperty: has_plasma_membrane_part
Annotations: expandExpressionTo
  "has_part some ('plasma membrane' and has_part some ?Y)"
```

This allows us to specify axioms using existential restrictions as follows:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and has_plasma_membrane_part some ('alpha-beta TCR complex')
```

This intermediate level representation would expand to:

```
Class: 'alpha-beta T cell'
EquivalentTo: 'T cell' and
  has_part some ('plasma membrane' and
                 has_part some 'alpha-beta TCR complex')
```

Note that the intermediate level representation can still be used by a reasoner. Whilst it may fail to make all the correct inferences, in this particular case it will not produce any incorrect inferences. We regard this semantically correct intermediate representation as a desirable trait, even if the intention is to ultimately reason over the expanded form.

However, if existential restrictions are used in the intermediate level representation, then it is possible to have an intermediate representation that is semantically incompatible with the expanded form.

Consider the following shortcut relation:

```
ObjectProperty: lacks_part
Annotations: expandExpressionTo "has_part exactly 0 ?Y"
```

The author would write in the intermediate form:

```
Class: 'enucleate cell'
EquivalentTo: cell and lacks_part some nucleus
```

Which would expand to:

```
Class: 'enucleate cell'
EquivalentTo: cell and has_part exactly 0 nucleus
```

Note that the intermediate form would produce *different* inferences from the expanded form. For example, if nucleus is a subclass of organelle, then the intermediate form entails that a enucleate cell lacks part some organelle. When expanded, this becomes too strong.

Notice that table 3 has an additional rule for hasValue restrictions:

```
?R value ?Y ==> substitute(?R,?Y)
```

Here we are treating the intermediate property as a relation between an individual and a class.

Now we can write an intermediate representation:

```
Class: 'enucleate cell'
EquivalentTo: cell and lacks_part value nucleus
```

This expands to the correct form. Whilst the intermediate form does not produce all the same inferences as the intermediate form, the intermediate form does not produce any incorrect inferences.

The meaning of the intermediate form may appear curious. Here we are treating the class *nucleus* as part of the domain of discourse, which is allowed in OWL2. The lacksPart shortcut relation holds between instances and classes.

We can also safely introduce an expansion rule for individuals:

```
PropertyValue(?R,?X,?Y) ==> ?X Types: substitute(?R,?Y)
```

We can write property assertions:

```
Individual: imaged-cell00001234
Properties: lacks_part nucleus
```

Which expand to:

```
Individual: imaged-cell00001234
Types: has_part exactly 0 nucleus
```

3 Uses

4 Discussion

4.1 Do we need shortcut relations?

Some ontology authors may have no objection to explicitly writing the fully expanded forms of axioms. But this is not really practical for the development of large ontologies by domain experts with intermediate-level OWL skills. In these cases it is tremendously beneficial to be able to use shortcut relations in axioms, especially where these shortcut relations are used repeatedly. Ideally these shortcut relations would be both usable within the standard OWL-centric ontology authoring environment.

This also works well in terms of division of labor - a smaller number of OWL experts can define a limited set of shortcut relations for a particular domain, and a larger number of domain experts can apply these.

4.2 Comparison with property chains

Property chains are a powerful feature introduced into OWL2. They are even more powerful when combined with self-restrictions, and can be used to recapitulate some of the features of the macro-expansion rules described above. For example, the *has plasma membrane part* relation could be specified as follows:

```
ObjectProperty: has_plasma_membrane_part
SubPropertyChain: has_part o is_plasma_membrane o has_part

ObjectProperty: is_plasma_membrane

Class: 'plasma membrane'
SubClassOf: is_plasma_membrane Self
```

A minor disadvantage here is that opaque axioms and properties are inserted into the ontology – but these could in principle be hidden given appropriate annotation properties and tooling.

A more serious disadvantage is that property chains cannot be used to *define* a property. The direction of implication is one-way only.

To see this, consider a standard axiom in mereology defining an overlaps relation in terms of parthood:

```
overlaps(x,y) <-> exists z: hasPart(x,z), partOf(z,y)
```

The overlaps relation is useful in bio-ontologies[2], so it can be tempting to declare an object property for it, and provide a property chain:

```
ObjectProperty: overlaps
SubPropertyChain: has_part o part_of
```

But this axiom is weaker, and there is a danger if this property is used in assertions thinking it has the same semantics. If an ontology asserts that a overlaps some b, then this is *not* the same as asserting that a hasPart some partOf some b. If a user queries for *hasPart some partOf some b*, then classes asserted to be subclasses of *overlaps some b* will *not* be returned.

It may be useful to have ontology "spell-checker" detect when properties with property chains are used in assertions, and ask the author if they really intended to write out the fully expanded form. As a matter of naming convention, it may be wise to call the property chain above something different from "overlaps".

Overall, the temptation to define shortcut relations using property chains should be avoided. Property chains should only be used when the unidirectional implication is truly intended (and in these situations they are of course very useful).

4.3 Practical usage and implementation: ontology views

The scheme we have proposed is relatively simple and can be implemented easily.

However, to be maximally useful, the translator could be integrated into ontology editing environments such as Protege4[**?**]. Expanded axioms could be marked using axiom annotations. Users would have the option of viewing the ontology at the intermediate level, the expanded level, or both. Changes at one level would automatically expand/contract to the other level, and feed in to incremental reasoning.

However, even without this tight integration we believe that a translation tool implementing the macro language specified here would be useful.

There is a possible concern about reasoning over the intermediate view, prior to expansion. However, our strategy here insulates us from making incorrect inferences at the intermediate level. Users will still have to be aware of the differences between the two levels to know why they get more inferences in the expanded view.

4.4 Applications for OBO to OWL translation

OBO-Edit is an ontology editing environment popular amongst biologists[3]. The native model for this tool is the OBO file format, which can be translated to a subset of OWL[4][8]. A characteristic feature of the OBO format is that an ontology is treated as a labeled graph, much like RDF. Edges in the graph are by default translated to existential restrictions. Whilst it is possible to write nested class expressions in OBO format by using RDF b-node style anonymous classes, this is not well-supported and best avoided. In addition, there is no standard way to write other OWL constructs such as negation and universal restriction, which are becoming increasingly necessary for bio-ontologies.

Many OBO-Edit authored ontologies have started informally adopting shortcut relations with a view to translating these to expanded OWL axioms further down the line. One approach is to do this as part of the obo to owl translation, but we believe there are advantages to doing the expansion in a separate layer. For one thing, it simplifies the already complicated translation, which has some problems in the existing translation. It also has the advantage of maintaining the shortcut relations in the OWL environment.

Currently a piece of obo-format may look like this:

```
[Typedef]
id: ro:has_part
is_transitive: true
inverse_of: ro:part_of

[Typedef]
id: lacks_part
expandExpressionTo: "ro:has_part 0 ?Y" []

[Term]
id: cl:enucleate_cell
intersection_of: cl:cell
intersection_of: lacks_part go:nucleus
```

Using a simple extension of the standard obo-to-owl specification, this would translate to:

```
ObjectProperty: lacks_part
```

```
Annotations:
  expandExpressionTo "ro:has_part exactly 0 ?Y"

Class: cl:enucleate_cell
EquivalentTo: cl:cell and lacks_part value go:nucleus
```

Applying the expansions rules we would get:

```
ObjectProperty: ro:has_part
Characteristics: Transitive

ObjectProperty: lacks_part
Annotations:
  expandExpressionTo "ro:has_part exactly 0 ?Y"

Class: cl:enucleate_cell
EquivalentTo: cl:cell and ro:has_part exactly 0 go:nucleus
```

Which has the intended semantics.

This has the advantage of allowing the users of OBO format tools to access the more expressive constructs in OWL, yet retain a simple graph-like model. This has positive implications for many downstream consumers of OBO format ontologies, many of which are tools or databases that also have a simple graph-like model of ontologies.

4.5 Applications for RDF and semantic web applications

If an OWL ontology contains complex axioms involving nested class expressions this can result in RDF triples that are difficult to work with at the RDF level. This difficulty is ameliorated by languages such as SPARQL-DL, but there is still problem for many triple stores and RDF libraries that are not OWL-aware.

The shortcut relation strategy provided here could provide a means of providing a "triple-friendly" view over complex OWL ontologies.

4.6 N-ary relations

The expansion strategy specified here is intended for binary relations, but could in principle be extended to n-ary relations.

4.7 Comparison with Quick Term Templates

Quick Term Templates (QTTs)[1] are a way of translating from simple tables that can be generated in Excel into complex OWL axioms. The macro expansion strategy specified here provides a useful extension to QTTs, ...

5 Conclusions

We have defined two annotation properties that can be used to specify shortcut relations that can be used in intermediate OWL-DL representations that provide a simplified view over a complex ontology.

6 References

References

1.
2. WM Dahdul, JG Lundberg, PE Midford, JP Balhoff, H. Lapp, TJ Vision, MA Haendel, M. Westerfield, and PM Mabee. The Teleost Anatomy Ontology: Anatomical representation for the genomics age. *Systematic Biology*, 2009.
3. John Day-Richter, Midori A Harris, Melissa Haendel, Gene Ontology OBO-Edit Working Group, and Suzanna Lewis. OBO-Edit–an ontology editor for biologists. *Bioinformatics*, 23(16):2198–2200, Aug 2007.
4. C. Golbreich and I. Horrocks. The obo to owl mapping, go to owl 1.1. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions: June 6-7 2007; Innsbruck, Austria*. Citeseer.
5. Anna Maria Masci, Cecilia Arighi, Alexander Diehl, Anne Lieberman, Chris Mungall, Richard Scheuermann, Barry Smith, and Lindsay Cowell. An improved ontological representation of dendritic cells as a paradigm for all cell types. *BMC Bioinformatics*, 10(1):70, 2009.
6. M.Horridge, N.Drummond, J.Goodwin, A.Rector, R.Stevens, and H.Wan. The Manchester OWL Syntax. *OWL: Experience and Directions 2006*, 2006.
7. Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J Mungall, The OBI Consortium, Neocles Leontis, Philippe Rocca-Serra, Alan Ruttenberg, Susanna-Assunta Sansone, Richard H Scheuermann, Nigam Shah, Patricia L Whetzel, and Suzanna Lewis. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat Biotechnol*, 25(11):1251–1255, Nov 2007.
8. S.H. Tirmizi, S. Aitken, D.A. Moreira, C. Mungall, J. Sequeda, N.H. Shah, and D.P. Miranker. OBO & OWL: Roundtrip Ontology Transformations. In *Semantic Web Applications and Tools for Life Sciences*, 2009.