

Extending OWL with Maximal Subproperties: An Approach to Define Qualified Cardinality Restrictions and Reflexive Properties

Stanislav Pokraev and Rogier Brussee

Telematica Instituut, P. O. Box 589,
7500 AN, Enschede, The Netherlands
{firstname.lastname}@telin.nl

Abstract. This paper proposes a simple, single extension of OWL, namely *maximalSubPropertyOf* that allows to “localize” a given property to a fixed domain and range. As an application we show that both qualified cardinality restriction and reflexive properties can be defined using this construction in conjunction with existing OWL functionality.

1 Introduction

When we model, we need a way to say that one or more individuals of a class can be related to one or more individuals of the same or some other class. For example, we might need to state that an individual of class `Component` can be part of an individual of class `Machine` or that an individual of class `Person` can be part of zero or more individuals of class `Organization`. We usually do this by defining a property `isPartOf` and then creating two subproperties, e.g. `isComponentPartOfMachine` (with domain `Component`, range `Machine` and cardinality 1) and `isPersonPartOfOrganization` (with domain `Person`, range `Organization` and no defined cardinality). In this way, whenever we state that some of the subproperties hold for two individuals of the respective classes, we can infer that the property `isPartOf` also holds for the same individuals. However, anytime we state that a `Component` `c` `isPartOf` a `Machine` `m` we implicitly mean that `c` `isComponentPartOfMachine` `m`, and likewise if `Person` `p` `isPartOf` an `Organization` `o` we implicitly mean that `p` `isPersonPartOfOrganization` `o`. However, it is impossible to express such implications in the current version of OWL.

In this paper we propose a simple, single extension of OWL, namely `maximalSubPropertyOf`, that allows expressing this. For example, it would be more convenient and more precise to say that `isComponentPartOfMachine` is the maximal subproperty of `isPartOf` with domain `Person` and range `Machine`, and `isPersonPartOfOrganization` is the maximal subproperty of `isPartOf` with domain `Person` and range `Organization`. This way, we can simply use `isPartOf` to state relations between individuals, and later on query using the more specific subproperties `isComponentPartOfMachine`

and `isPersonPartOfOrganization` or vice versa. In addition, maximal subproperties can be used to address the lack of the qualified cardinality restrictions and reflexive properties in the current version of OWL.

2 Maximal Subproperty

We propose a simple, single extension of OWL, namely `maximalSubPropertyOf`. The formal semantics of `maximalSubPropertyOf` is defined as follows:

$$\exists \text{maximalSubPropertyOf}(Q, P) \wedge \exists \text{domain}(Q, D) \wedge \exists \text{range}(Q, R) \rightarrow (\forall x.D(x) \wedge \forall y.R(y) \wedge \exists P(x, y) \rightarrow \exists Q(x, y))$$

where Q with domain D and range R is the *maximal subproperty* of the property P . The maximal subproperty of P is the unique subproperty which is maximal among all subproperties of P with the given domain D and range R , i.e. the domain and range of a `maximalSubPropertyOf` is part of its definition. A more natural name for this construction is the restriction of a property to the given domain D and range R , as the maximal subproperty of P is the same as the property P between D and R , and undefined outside. This point of view can be seen from the following example. Consider the sine function as a property on the real numbers. Then the restriction of sine to a function on $[-\pi/2, \pi/2]$ with values in $[-1, 1]$ is the maximal subproperty of sine with domain $[-\pi/2, \pi/2]$ and range $[-1, 1]$. Note that on this domain the sine function is bijective, i.e. a functional and inverse functional. However given the use of `restriction` as a *class* constructor in OWL this term would be confusing.

Lemma:

Suppose that

Q `maximalSubPropertyOf` P ;
 domain D ;
 range R .

Then the following statements are equivalent

- (1) $x Q y$.
- (2) $x \text{ a } D. y \text{ a } R. x P y$.

Proof

(1 \Rightarrow 2) trivial.

(2 \Rightarrow 1) is precisely the maximality of the `maximalSubPropertyOf`.

It follows from the lemma that *in the absence of further statements involving* Q , decidability is as difficult as classifying individuals.

We now consider some applications.

Qualified Cardinality Restriction

OWL allows the definition of cardinality restrictions which are used to constrain the number of values of a particular property. Suppose we want to define a building with at least 4 fire escape stairs:

```
Building a owl:Class .
FireEscapeStairs a owl:Class .
hasFireEscapeStairs a owl:ObjectProperty ;
  rdfs:domain Building ;
  rdfs:range FireEscapeStairs .
BuildingMin4Fire a Building ,
  [a owl:Restriction;
    owl:onProperty hasFireEscapeStairs; owl:minCardinality 4
  ] .
```

Now, suppose we want to refine our model and say that at least two of the fire escape stairs should be *external* fire escape stairs. To do that we need a mechanism to state that the property `hasFireEscapeStairs` must have `owl:minCardinality 2` when its value has type `FireEscapeStairs` and `owl:minCardinality 2` when its value has type `ExternalFireEscapeStairs`. In the OWL community this is known as *qualified cardinality restriction* and unfortunately, is unexpressable by the current version of OWL. The need for qualified cardinality restriction is motivated by Alan Rector in [3].

One possible workaround (proposed by Guus Schreiber in [1]) is to define a property `hasExternalFireEscapeStairs` with a range `ExternalFireEscapeStairs` as a subproperty of `hasFireEscapeStairs` and restrict the number of its values to at least 2:

```
Building a owl:Class .
FireEscapeStairs a owl:Class .
ExternalFireEscapeStairs rdfs:subClassOf FireEscapeStairs .
hasFireEscapeStairs a owl:ObjectProperty ;
  rdfs:domain Building ;
  rdfs:range FireEscapeStairs .
hasExternalFireEscapeStairs rdfs:subPropertyOf hasFireEscapeStairs ;
  rdfs:range ExternalFireEscapeStairs .
BuildingMin4FireMin2External a
  [a owl:Restriction;
    owl:onProperty hasFireEscapeStairs; owl:minCardinality 4],
  [a owl:Restriction;
    owl:onProperty hasExternalFireEscapeStairs; owl:minCardinality 2].
```

However, this workaround has a serious reasoning limitation. Suppose we define an instance of a `Building` only using the `hasFireEscapeStairs` property:

```
stairs1 a FireEscapeStairs ;
stairs2 a FireEscapeStairs ;
stairs3 a ExternalFireEscapeStairs ;
stairs4 a ExternalFireEscapeStairs ;

aBuilding a Building ;
  hasFireEscapeStairs stairs1 ;
  hasFireEscapeStairs stairs2 ;
```

```
hasFireEscapeStairs stairs3 ;
hasFireEscapeStairs stairs4 .
```

The instance cannot be classified as `BuildingMin4FireMin2External` because `aBuilding` does not have at least two properties `hasExternalFireEscapeStairs` with values of type `ExternalFireEscapeStairs`.

Now, let us redefine the property `hasExternalFireEscapeStairs` using `maximalSubPropertyOf`:

```
hasExternalFireEscapeStairs a
  owl:maximalSubPropertyOf hasFireEscapeStairs ;
  rdfs:domain Building ;
  rdfs:range ExternalFireEscapeStairs .
```

This way, the individual `aBuilding` can be classified as a `BuildingMin4FireMin2External` because the definition and the facts

```
stairs3 a ExternalFireEscpaeStairs ;
stairs4 a ExternalFireEscapeStairs ;
aBuilding a Building ;
  hasFireEscapeStairs stairs3 ;
  hasFireEscapeStairs stairs4 .
```

imply

```
aBuilding hasExternalFireEscapeStairs stairs3 .
aBuilding hasExternalFireEscapeStairs stairs4 .
```

3 Reflexive properties

The `maximalSubPropertyOf` allows defining *reflexive properties*. A reflexive property defined on a domain D is a property that holds between each individual of D and itself, i.e.:

$$\text{reflexive}(R) \wedge \exists \text{domain}(R, D) \wedge \exists \text{range}(R, D) \rightarrow (\forall x. D(x) \rightarrow \exists R(x, x))$$

Examples of reflexive properties are “*is equal to*”, “*is subset of*”, “*is less than or equal to*” and “*is greater than or equal to*”.

If we consider R as a subset of $D \times D$ (i.e. we consider the interpretation), then R is reflexive if and only if the diagonal in $D \times D$ is contained in R . This geometric interpretation shows a natural way to define a reflexive property using `maximalSubPropertyOf`. The diagonal of `owl:Thing × owl:Thing` corresponds to `owl:sameAs`. To get the diagonal in $D \times D$, we merely have to restrict the property `owl:sameAs` to D :

```
sameAsOnD owl:maximalSubPropertyOf owl:sameAs ;
  rdfs:domain D ;
  rdfs:range D .
```

```

aReflexiveProperty a owl:ObjectProperty;
  rdfs:domain D ;
  rdfs:range D .
sameAsOnD rdfs:subPropertyOf aReflexiveProperty .

```

A reflexive and transitive property P on class C is a property such that the composition of *zero or more* properties P is a subproperty of P , where a composition of *zero* properties P equals the restriction of `owl:sameAs` to the domain C (i.e., the composition identity on C). This point of view is very convenient when dealing with part-whole relations. This was recognized by the W3C best practices workgroup and described in an extensive workaround[2]. However, this is merely an approximation of reflexivity as they point out themselves. We present an example to illustrate the problem:

Suppose we want to describe that a printer is located in the corner of a room:

```

Space a owl:Class .
Building rdfs:subClassOf Space
Room rdfs:subClassOf Space .
Corner rdfs:subClassOf Space .
Printer a owl:Class .

isLocatedIn a owl:ObjectProperty ;
  rdfs:range Space .
isPartOf a owl:TransitiveProperty ;
  rdfs:domain Space ;
  rdfs:range Space .

```

We create instances of rooms, corners and printers.

```

aBuilding a Building.
aRoom a Room .
aCorner a Corner .
aPrinter a Printer .

aRoom isPartOf aBuilding.
aCorner isPartOf aRoom .
aPrinter isLocatedIn aCorner .

```

Now suppose we want to know whether `aPrinter` is in `aRoom` and if yes, where exactly in the room it is located. Therefore, we pose a query

```
isLocatedIn(aPrinter, ?x) AND isPartOf(?x, aRoom)
```

The result we obtain is

```
?x = aCorner
```

Now, suppose that we were *less accurate* in describing the location of our printer and instead of

```
aPrinter isLocatedIn aCorner .
```

we stated

```
aPrinter isLocatedIn aRoom .
```

The same query now returns *no* results because there is no statement (neither explicit nor inferred)

```
aRoom isPartOf aRoom .
```

i.e. the `isPartOf(?x, aRoom)` will never evaluate *true*.

We can add `isPartOf` property for all individuals of type `Space`, but clearly a more elegant solution is to make `isPartOf` a reflexive property.

4 Acknowledgements

This research is a part of the Freeband Communication projects A-Muse (<http://a-muse.freeband.nl>) and awareness (<http://awareness.freeband.nl>). Freeband Communication (<http://www.freeband.nl>) is sponsored by the Dutch government under contract BSIK 03025.

5 References

- [1] Guus Schreiber. Qualified cardinality restrictions (QCRs): constraining the number of property values of a particular type, first draft, 25 May 2004. Available at <http://www.cs.vu.nl/~guus/public/qcr.html>
- [2] Simple part-whole relations in OWL Ontologies, W3C Editor's Draft 11 Aug 2005. Available at <http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/>
- [3] Alan Rector. Case for Reinstatement of Qualified Cardinality Restrictions. Available at: <http://lists.w3.org/Archives/Public/public-webont-comments/2003Apr/0040.html>