SKOS TO PROV VIA OWL

3/29/2017

Combining SKOS models with RDFS within a PROV-like upper ontology

SKOS to PROV via OWL

Revision History:

Date	Version	Description	Author
3/29/2017	1.0	Initial Document	PJL
3/29/2017	2.0	Updated example queries	PJL

SKOS to PROV via OWL

Table of Contents

Figures

Tables

No table of figures entries found.

SUMMARY

SKOS, the Simple Knowledge Organization System, offers an easy to understand schema for vocabularies and taxonomies. However modeling precision is lost when skos:semanticRelation predicates are introduced.

Combining SKOS with RDFS/OWL allows both the precision of owl:ObjectProperty to be combined with the flexibility of SKOS. However clarity is then lost as the number of core concepts (aka owl:Class) grow.

Many models are not just documenting the 'state' of an entity. Instead they are often tracking the actions performed on entities by agents at locations. Thus aligning the core concepts to the Activity, Entity, Agent, and Location classes of the PROV ontology provides a generic upper-ontology within which to organize the model details.

VEHICLE MANUFACTURING EXAMPLE

This examples captures information about vehicle manufacturing. Following

1. Manufacturers: the manufacturer of models of cars in various production lines sited at plants
2. Models: the models that the manufacturer produces
3. ProductionLines: the production lines set up to produce models of vehicles on behalf of a manufacturer
4. Plants: the plants that house the production lines

In addition there are different 'styles' of manufacturing that occur for various models and various sites:

5. Manufacturing: the use of a ProductionLine for a particular Model

SKOS Modeling

If we follow a pure SKOS model we proceed as follows by creating a VehicleManufacturingScheme skos:ConceptScheme

```
s:VehicleManufacturingScheme
  rdf:type skos:ConceptScheme
.
```

Then we create skos:topConceptOf Manufacturer, Model, Plant, and Production as follows:

```
s:Manufacturer
  rdf:type owl:Class ;
  rdfs:subClassOf skos:Concept ;
  skos:topConceptOf s:VehicleManufacturingScheme
.
s:Model
  rdf:type owl:Class ;
  rdfs:subClassOf skos:Concept ;
  skos:topConceptOf s:VehicleManufacturingScheme
.
```

These top-level concepts are being created of type owl:Class and a subClassOf skos:Concept. This is the pattern recommended in (Bechhofer, et al.)

Finally we can create skos:broader concepts as follows:

```
s:Ford
  rdf:type s:Manufacturer ;
```

```
    skos:broader s:Manufacturer ;
    skos:inScheme s:VehicleManufacturingScheme
.

s:Fusion
    rdf:type s:Model ;
    skos:broader s:Model ;
    skos:inScheme s:VehicleManufacturingScheme
.
```

The resultant SKOS taxonomy of the VehicleManufacturingScheme skos:ConceptScheme then appears as follows:

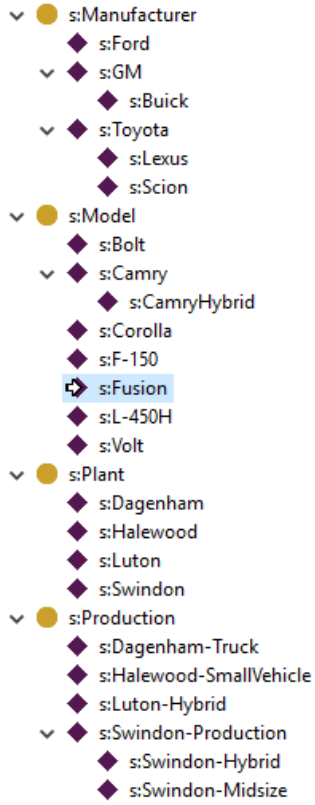

FIGURE 1: SKOS TAXONOMY

Why?

By starting with a pure SKOS model we provide access to the underling concepts in a more accessible style for the less proficient user, as illustrated by the SKOS Taxonomy above. Yet we have not sacrificed the ontological precision of owl:Classes.

Thus we can ask questions about all concepts:

```
SELECT * WHERE
{
      ?myConcepts rdfs:subClassOf+ skos:Concept .
}
```

Or we can get a list of anything broader than one of these concepts:

```
SELECT * WHERE
{
      ?myBroaderConcepts skos:broader s:Model .
```

```
}
```

SKOS+OWL Modeling

Although skos:semanticRelation allows one to link concepts together, this predicate is often too broad when trying to create an ontology that documents specific relations between specific types of concept.

In our VehicleManufacturingScheme we might want to know the following:

1. isManufacturedBy: which manufacturer manufactures a particular model
2. operatedBy: which manufacturer operates a particular production facility
3. performedAt: which plant is the location of a production facility
4. wasManufacturedAt: which production facility was used to manufacture a particular model

FIGURE 2: SKOS+OWL MODEL OF RELATIONS

These predicates can be defined using RDFS as follows:

```
so:isManufacturerBy
  rdf:type owl:ObjectProperty ;
  rdfs:domain s:Model ;
  rdfs:range s:Manufacturer ;
  rdfs:subPropertyOf skos:semanticRelation
.

so:operatedBy
  rdf:type owl:ObjectProperty ;
  rdfs:domain s:Production ;
  rdfs:range s:Manufacturer ;
  rdfs:subPropertyOf skos:semanticRelation
.
```

Note that the definition of Model, Manufacturer etc. as subClassOf skos:Concept allows us to precisely define the domain and range.

```
s:Fusion
  so:isManufacturerBy s:Ford ;
  so:wasManufacturedAt s:Halewood-SmallVehicle ;
.

s:Dagenham-Truck
  so:operatedBy s:Ford ;
  so:performedAt s:Dagenham ;
.
```

Thus we have used the flexibility of SKOS with the greater modeling precision of RDFS/OWL.

Why?

By building upon the SKOS model, one can ask an expansive question such as what concepts are semantically related to, say, the concept s:Fusion with a simple query:

```
SELECT * WHERE
{
       s:Fusion  ?p ?y .
       ?p rdfs:subPropertyOf* skos:semanticRelation
}
```

Yet with the same model we can ask a specific question about a relationship of a specific instance:

```
SELECT * WHERE
{
       s:Camry so:isManufacturerBy   ?o .
}
```

SKOS+OWL+PROV Modeling

One of the attractions of SKOS is that a taxonomy can grow organically. One of the problems of SKOS is that a taxonomy can grow organically!

As the taxonomy grows it can be useful to add another layer of structure beyond a catalog of concepts. Many models are not just documenting the 'state' of an entity. Instead they are often tracking the actions performed on entities by agents at locations. Thus aligning the core concepts to the Activity, Entity, Agent, and Location classes of the PROV ontology (Lebo, et al.) provides a generic upper-ontology within which to organize the model details.

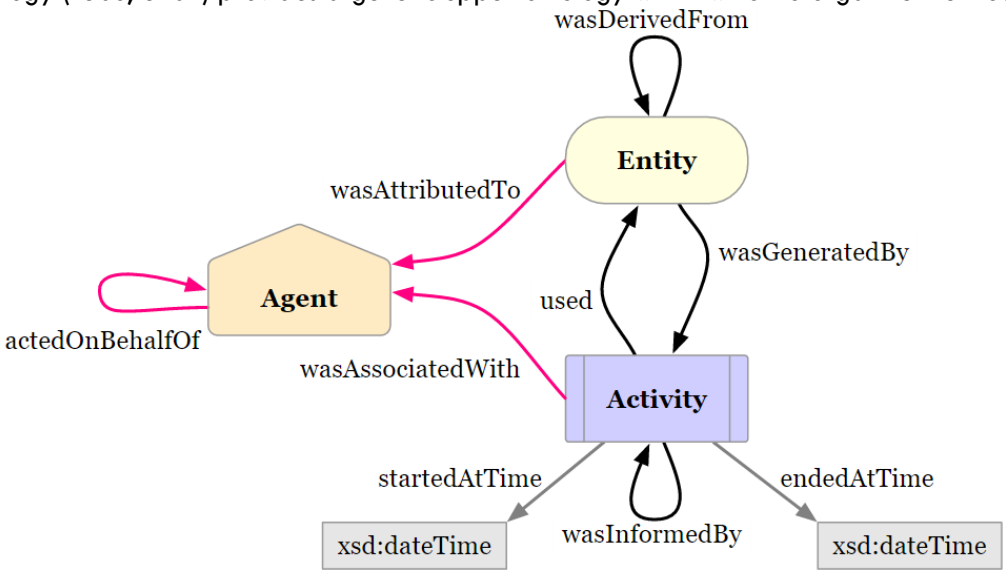

FIGURE 3: PROV MODEL

Thus our VehicleManufacturingScheme has each core PROV concept:

1. Manufacturers: the Agents who manufacture models, and operate plants
2. Models: the Entities
3. ProductionLines: the Activities that produce Models on behalf of Manufacturers.
4. Plants: the Location at which Activities take place, and Agents and Entities are located.

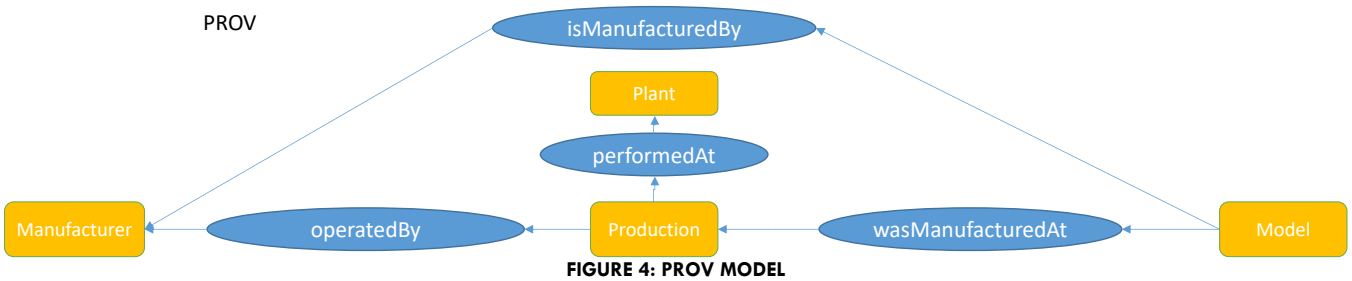

FIGURE 4: PROV MODEL

```
s:Production
  rdfs:subClassOf prov:Activity ;
```

```
.
s:Model
  rdfs:subClassOf prov:Entity ;
.
s:Manufacturer
  rdfs:subClassOf prov:Organization ;
.
s:Plant
  rdfs:subClassOf prov:Location ;
.
```

Similarly we can cast our predicates into the same PROV model as follows:

```
so:isManufacturerBy
  rdfs:subPropertyOf prov:wasAttributedTo ;
.
so:operatedBy
  rdfs:subPropertyOf prov:wasAssociatedWith ;
.
so:performedAt
  rdfs:subPropertyOf prov:atLocation ;
.
so:wasManufacturedAt
  rdfs:subPropertyOf prov:wasGeneratedBy ;
.
```

Why?

The PROV model is closer to the requirements of most enterprise models, that are trying to 'model the business', than a simple E-R model. The latter concentrates on capturing the attributes of an entity that record the current state of that entity. Often those attributes focus on documenting the process by which the entity gained its current state:

- The agent that created the entity
- The activity used to create the entity
- The location when things were performed
- The data of the activity, etc

Superimposing the PROV model formalizes this model, and thus allows a structure within which a more casual user can navigate, rather than a sea of entities.

By building upon the PROV model, one can ask an expansive question such as what entities behave as Agents and in which entities are they involved:

```
SELECT * WHERE
{
       ?organization a ?Agent .
       ?Agent rdfs:subClassOf* prov:Agent .
       ?entity ?predicate ?organization
}
```

SKOS+OWL+PROV-Qualified Modeling

Within the structure of PROV, predicates define the relationships between Activities, Entities, Agents, and Locations. However it is sometimes necessary to qualify these relationships.
For example, the so:wasManufacturedAt predicate defines that a s:Production facility was used to manufacture a s:Model. When? How was it used? Why?meenu

To extend the model, PROV adds the concept of a qualified influence, which allows the relationship to be further defined.

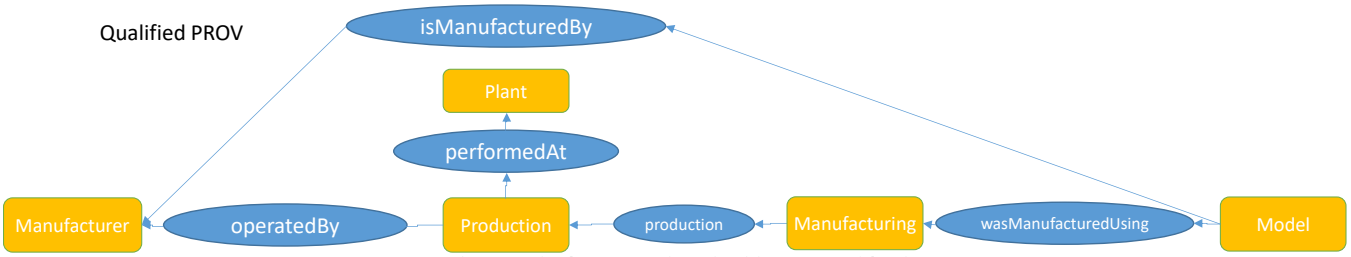

FIGURE 5: QUALIFIED PROV FOR SOME PREDICATES

We do this first of all by creating sopq:Manufacturing:

```
sopq:Manufacturing
  rdf:type owl:Class ;
  rdfs:subClassOf skos:Concept ;
  rdfs:subClassOf prov:Generation ;
  skos:topConceptOf s:VehicleManufacturingScheme ;
.
```

Note that this is a rdfs:subClassOf prov:Generation, the reification of the predicate prov:wasGeneratedBy

We then add two predicates, one (sopq:wasManufacturedUsing) from the prov:Entity to the prov:Generation, and one (sopq:production) from the prov:Generation to the prov:Activity as follows:

```
sopq:wasManufacturedUsing
  rdf:type owl:ObjectProperty ;
  rdfs:domain s:Model ;
  rdfs:range sopq:Manufacturing ;
  rdfs:subPropertyOf skos:semanticRelation ;
  rdfs:subPropertyOf prov:qualifiedGeneration ;
.

sopq:production
  rdf:type owl:ObjectProperty ;
  rdfs:subPropertyOf skos:semanticRelation ;
  rdfs:subPropertyOf prov:activity ;
.
```

Finally we can create a Manufacturing qualified generation concept as follows:

```
sopq:L-450H_at_Swindon-Hybrid
  rdf:type sopq:Manufacturing ;
  sopq:production s:Swindon-Hybrid ;
  skos:broader sopq:Manufacturing ;
.
s:L-450H
  sopq:wasManufacturedUsing sopq:L-450H_at_Swindon-Hybrid ;
.
```

In the figure below we can see that these qualified actions simply extend the SKPOS taxonomy:

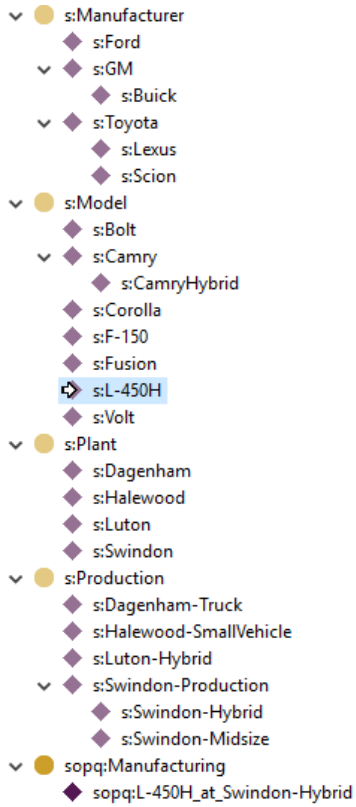

FIGURE 6: TAXONOMY EXTENDED WITH QUALIFIED ACTIONS

Why?

Using qualifiedActions provides a systematic, rather than ad-hoc, way to provide more precision to a model.

FURTHER WORK

1. The PROV structure does not manifest itself within the taxonomy. Should Activity, Entity, Agent, and Location therefore be ConceptSchemes?

MODEL

The model files used in this example are included here:

- skos.ttl
- skos+owl.ttl
- skos+owl+prov.ttl
- skos+owl+provqualified.ttl

REFERENCES

Bechhofer, Sean and Miles, Alistair. Using OWL and SKOS. [Online] W3C. https://www.w3.org/2006/07/SWD/SKOS/skos-and-owl/master.html.
Lebo, Timothy, Satya, Sahoo and McGuinness, Deborah. PROV-O: The PROV Ontology. W3C. [Online] https://www.w3.org/TR/prov-o/.