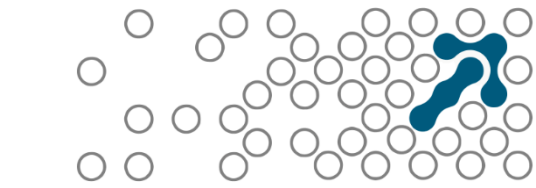
AVOIDING PROPERTY PROLIFERATION

Part 1

By Dan Carey



semantic arts

Logical Necessity Meets Elegance

Screwdrivers generally have only a small set of head configurations (flat, Phillips, hex) because the intention is to make accessing contents or securing parts easy (or at least uniform).

Now, imagine how frustrating it would be if every screw and bolt in your house or car required a unique screwdriver head. They might be grouped together (for example, a bunch of different sized hex heads), but each one was slightly different. Any maintenance task would take much longer and the amount of time spent just organizing the screwdrivers would be inordinate.

Domain and range for ontological properties are not about data integrity, but logical necessity. Misusing them leads to an inelegant (and unnecessary) proliferation of properties.

Yet that is precisely the approach that most OWL modelers take when they over-specify their ontology's properties.

On our blog, we once briefly discussed the concept of elegance in ontologies. A key criterion was, *"An ontology is elegant if it has the fewest possible concepts to cover the required scope with minimal redundancy and complexity."*

Let's take a deeper look at object properties in that light. First, a quick review of some of the basics.

1. An ontology describes some subject matter in terms of the meaning of the concepts and relationships within that ontology's domain.
2. Object properties are responsible for describing the relationships between things.
3. In the RDFS and OWL modeling languages, a developer can declare a property's domain and/or its range (the class to which the Subject and/or Object, respectively, must belong).

Break the Habit

In our many years' experience teaching our classes on designing and building ontologies, we find that most new ontology modelers have a background in relational databases or Object-Oriented modelling/development. Their prior experience habitually leads them to strongly tie properties to classes via specific domains and ranges. Usually, this pattern comes from a desire to curate the triplestore's data by controlling what is getting into it.

But specifying a property's domain and range will not (necessarily) do that.

For example, let's take the following assertions:

- The domain of the property :hasManager is class :Organization.
- The individual entity :_Jane is of type class :Employee.
- :_Jane :hasManager :_George.

Many newcomers to semantic technology (especially those with a SQL background) expect that the ontology will prevent the third statement from being entered into the triplestore because :_Jane is not declared to be of the correct class. But that's not what happens in OWL. The domain says that :_Jane must be an :Organization, which presumably is not the intended meaning.

Because of OWL's Open World paradigm, the only real constraints are those that prevent us from making statements that are logically inconsistent. Since in our example we have not declared the :Organization and :Employee classes to be disjoint, there is no logical reason that :_Jane cannot belong to both of those classes. A reasoning engine will simply infer that :_Jane is also a member of the :Organization class. No errors will be raised; the assertion will not be rejected. (That said, we almost certainly do want to declare those classes to be disjoint.)

> *"An ontology is elegant if it has the fewest possible concepts to cover the required scope with minimal redundancy and complexity."*

Keep it Useful

Quite apart from the ineffectiveness of relying on the domain and range to curate the data, there is also the detrimental effect on the ontology's usefulness and elegance if that pattern is used repeatedly to create properties that differ only in domain and/or range.

Consider this set of properties:

- :hasVendorAddress has domain :Organization.
- :hasCustomerAddress has domain :Person.
- :hasEmployeeAddress has domain :Employee.

In this case, a triplestore user wanting to query the addresses must be aware of all the properties in this :hasAddress pattern and either:

a) explicitly know in advance for which class they want results, or
b) union different sets together in the WHERE clause to get all the desired results back.

And what happens when there's a need to record your company's facility addresses? Yet another property must be created, if this pattern is to be followed. Now we are getting into a different screwdriver for every small variation.

Looking again at the criterion for elegance given above, we see that it would be far less redundant and complex to have a single property :hasAddress with no declared domain. Then, the query writer only needs to know and remember one property and can get the results back for all Subject classes more simply. Or they can add a single line to the WHERE clause explicitly stating which Subject classes or categories they want. It also means no lag time getting a change through governance when the new facilities address data needs to be loaded.

This is a much more intuitive approach for those writing queries and greatly reduces the maintenance burden for the ontologist.

For system- and application-building purposes, effective data integrity mechanisms or "hints" might well be needed in your systems. The need to curate data, controlling what goes into the triplestore, is quite legitimate. But generally, the ontology is not the appropriate place to do that. Fortunately, the W3C has created the SHACL language to be used with OWL ontologies for things like defining data integrity constraints that software developers can use in GUIs, ETL scripts, and APIs.

At this point, the astute observer might say, "But if those domain-bound properties were made sub-properties of :hasAddress, then all the results would show up in a query using the parent property." This will be true if the reasoner is running and the query engine is one that returns inferred results. (Not all of them do.) It is also the case that if this super-property/sub-property pattern is used extensively in the ontology, the reasoner will explode the number of assertions in memory to be searched through, negatively affecting performance. And this case still begs the question: what is the sub-property truly for?

What to Remember

So, to recapitulate the main points:

- Domains and ranges are for describing logical necessity, not for curating data.
    - SHACL can be used to document the curatorial data constraints for the benefit of software developers.

Having too many properties makes using and maintaining the ontology more difficult.

- Relying on super-property/sub-property inferencing can be dicey and will affect query performance.

In our experience, it is almost always better to avoid proliferation and to opt for the elegance of the unconstrained object property.

(We will follow up with another white paper explaining a different set of reasons and methods for avoiding object property proliferation.)

11 Old Town Square
Suite 200
Fort Collins, CO 80524

970-490-2224
305-425-2224
info@semanticarts.com