Literal properties and atomic classes in NIEM 6

Rationale: Literal properties and atomic classes are a part of a technology-neutral NIEM architecture. We support XML and JSON today. I expect at some point to add YAML, Protobuf, Avro, etc. Looking ahead a few years in my crystal ball, I see most developers working with the new serializations, and relatively little XML work. So I want to avoid XML-specific weirdness in CMF and NIEM JSON. And simple content with attributes turns out to be plenty weird.

Every other serialization assumes a data model in which the value and the value's unit of measure are both properties of an object. NIEM even does this for XML. For example, if my schema has

```
<xs:element name="ThingWeight" type="nc:WeightMeasureType">
```

then my XML message will look like this:

```
<my:Thing>
  <my:ThingWeight>
    <unece:MassUnitCode>KGM</unece:MassUnitCode>
    <nc:MeasureDecimalValue>22.5</nc:MeasureDecimalValue>
  </my:ThingWeight>
</my:Thing>
```

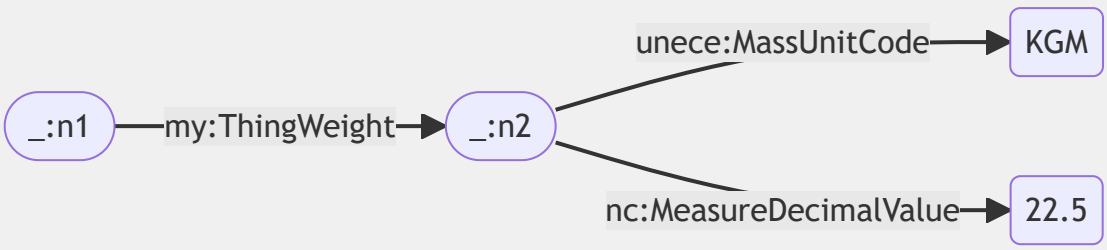
and the equivalent JSON message is

```
{
  "@context": "<i>MyContextURIHere</i>",
  "my:Thing": {
    "my:ThingWeight": {
      "unece:MassUnitCode": "KGM",
      "nc:MeasureDecimalValue": 22.5
    }
  }
}
```

and the RDF interpretation for both is

```
_:n0 my:Thing _:n1 .
_:n1 rdf:type my:ThingType .
_:n1 my:ThingWeight _:n2 .
_:n2 rdf:type nc:WeightMeasureType .
_:n2 unece:MassUnitCode "KGM" .
_:n2 nc:MeasureDecimalValue 22.5 .
```

and the RDF diagram looks like this.



According to this data model, the value of `my:ThingWeight` is an object with two properties: the weight value and the weight units of measurement. Every property has its own name and definition in the data model. Hurray!

I believe this is the data model that everyone else in the world will expect.

And then comes XML attributes. Now there is a second way to construct the data model.

```
<xs:element name="ThingWeight" type="my:ThingWeightType:/>
<xs:complexType name="ThingWeightType">
  <xs:simpleContent>
    <xs:extension base="niem-xs:decimal">
      <xs:attribute ref="my:massUnitCode"/>
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:attribute name="massUnitCode" type="unece:MassCodeSimpleType"/>
```

Now my XML message looks like this:

```
<my:Thing>
  <my:ThingWeight my:massUnitCode="KGM">22.5</my:ThingWeight>
</my:Thing>
```

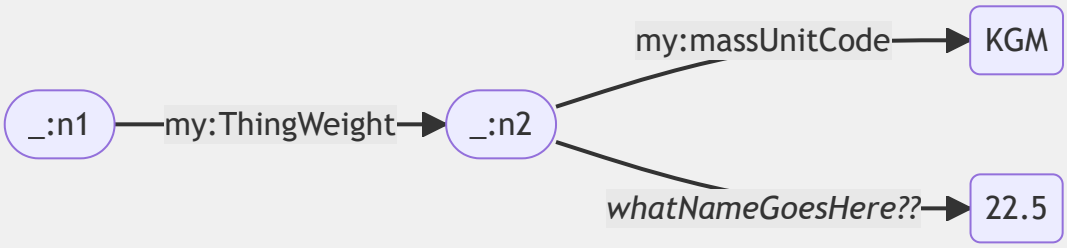
and the equivalent JSON message is

```
{
  "@context": "MyContextURIHere",
  "my:Thing": {
    "my:ThingWeight": {
      "my:massUnitCode": "KGM",
      "whatNameGoesHere??": 22.5
    }
  }
}
```

and the RDF interpretation for both is

```
_:n0 my:Thing _:n1 .
_:n1 rdf:type my:ThingType .
_:n1 my:ThingWeight _:n2 .
_:n2 rdf:type my:ThingWeightType .
_:n2 my:massUnitCode "KGM" .
_:n2 whatNameGoesHere?? 22.5
```

and the RDF diagram looks like this.



At this point it might seem convenient to have a value as the subject of an RDF triple, like this:

Or, in natural language, "this value 22.5 is expressed in kilos". Just what we want! But that is not possible in RDF. The subject of a triple must be a resource. You can't have a predicate and value on a literal value. So that's out. What else?

Well, we long ago decided to use JSON-LD for NIEM JSON. When we say "plain JSON", we mean JSON-LD that looks like plain old JSON, with no developer understanding of JSON-LD required. But it still has a `@context` and otherwise follows JSON-LD rules.

That constrains us in several ways. JSON-LD is an RDF syntax. That means a NIEM JSON message alreadh has an RDF interpretation. We have to make sure it's the same RDF you get from the equivalent NIEM XML message.

Those constraints mean:

- We can't use `@value` for *whatNameGoesHere*. `@value` is a reserved key in JSON-LD. It means something else. (But you *can* use it in JSON-LD-star; see below.)
- We can't use `$` or similar magic tokens for *whatNameGoesHere*. That key isn't a compact IRI. In JSON-LD, if a key is not an IRI or a compact IRI, then that key/value pair simply vanishes from the RDF.

We could use `rdf:value` for *whatNameGoesHere*. But as a JSON developer, I say that sucks. What else?

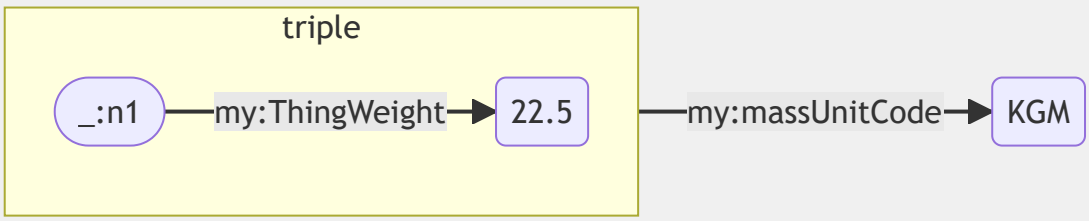
RDF-star? Turns out I looked into that two years ago. I'm still not convinced that RDF-star (soon to be RDF 1.2) and JSON-LD-star are the answer. RDF-star is a way to have a triple as the subject of a triple. It's a less-ugly form of reification. It could maybe work. If we write JSON-LD-star like this:

```
{
  "@context": "<i>MyContextURIHere</i>",
  "my:Thing": {
    "my:ThingWeight": {
      "@value": 22.5.
      "@annotation": {
        "my:massUnitCode": "KGM"
      }
    }
  }
}
```

which I believe turns into RDF-star like this:

```
_:n0 my:Thing _:n1 .
_:n1 rdf:type my:ThingType .
_:n1 my:ThingWeight 22.5
  {| my:massUnitCode "KGM" |} .
```

we get an RDF diagram like this



OK, if I squint just right I can interpret that graph as "kilogram is the units of measure of the ThingWeight relationship between `_:n1` and the value 22.5". So I suppose it *could* work. But...

- I don't know if this is a customary or acceptable interpretation in RDF-land. I searched and could not find anything one way or the other.
- RDF 1.2 and JSON-LD-star are not yet standards. (RDF-star is supported by Apache Jena; I haven't found any FOSS support for JSON-LD-star.) I don't mind using them for relationship metadata, because I suspect no one has ever used relationship metadata in the real world. We couldn't even define what it means until 2023. So if it turns out to be a mistake, well, the pain will be small.

- But now we're going to use it for freaking `nc:TextType`?? If that turns out to be a mistake, the pain will not be small.
- I am happy when the two ways to say the same thing in XML have the same sort of RDF interpretation. I am uneasy when they have very different interpretations.
- In general, I like anything that hides the element/attribute distinction from non-XML developers. Using RDF* forces everyone to care, forever.

What's left? What sucks least? Literal properties.

Author: Scott Renner
Last modified: 2024-10-15