**Ideas for NIEM 6 NDR**

# Replace sequenceID for ordered properties

At present the attribute @structures:sequenceID is the mechanism for establishing an order for repeated properties. For example, if you want the guy’s name to be “Peter Death Bredon Wimsey” instead of “Peter Bredon Death Wimsey”, you have to do this:

<nc:PersonGivenName>Peter

<nc:PersonMiddleName s:sequenceID="1">Death

<nc:PersonMiddleName s:sequenceID="2">Bredon

<nc:PersonSurName>Wimsey

Translating those elements and attributes into the NIEM-JSON representation gives us:

"nc:PersonMiddleName": [

{

"s:sequenceID" : 1,

"rdf:value": "Death"

},

{

"s:sequenceID": 2,

"rdf:value": "Bredon"

]

And in RDF/Turtle:

\_:b1 nc:PersonMiddleName \_:b2, \_:b3 .

\_:b2 s:sequenceID 1 ;

rdf:value "Death" .

\_:b3 s:sequenceID 2 ;

rdf:value "Bredon" .

But that’s not the natural representation of ordered properties in either JSON-LD or RDF. The natural way in JSON-LD is to declare in the context that PersonMiddleName is an ordered collection:

"@context" : {

"nc:PersonMiddleName" : {

"@container": "@list"

}

},

"nc:PersonMiddleName": [ "Death", "Bredon" ]

And in RDF/Turtle there’s a simple syntax for an ordered list

\_:b1 nc:PersonMiddleName ( "Death" "Bredon" ) .

So I think the @sequenceID attribute placed into the runtime data is the wrong way to indicate that the order of repeated proprties is significant. I think appinfo in the schema is the right way, like this:

<xs:complexType name="PersonName">

<xs:complexContent>

<xs:extension base="s:ObjectType">

<xs:element ref="nc:PersonMiddleName"

minOccurs="0" maxOccurs="unbounded" appinfo:isOrdered="true"/>

We can, if we wish, retain @sequenceID for those occasions when the different element in a complex type do not appear in the desired order. The example in the NDR is Chinese personal names, where you might want to do this:

<nc:PersonName>

<nc:PersonGivenName s:sequenceID="2">Ming

<nc:PersonSurName s:sequenceID="1">Yao

But in my opinion that sort of thing should be modeled explictly when needed, perhaps like this:

<nc:PersonName>

<nc:PersonGivenName>Ming

<nc:PersonSurName>Yao

<nc:GivenNameLastIndicator>true

And so I think we could get rid of the @sequenceID attribute entirely.

# Deprecate semantic attributes

The properties of an object can be represented as an element or an attribute; for example:

<nc:ItemLengthMeasure>

<nc:MeasureDecimalValue>9.7</nc:MeasureDecimalValue>

<nc:LengthUnitCode>CMT</nc:LengthUnitCode>

<nc:PersonName nc:personNameCommentText="copied">

Attributes are a big part of the reason for the yucky rdf:value property in NIEM-JSON and NIEM-RDF. They are XML-specfic; there is no equivalent in JSON, protobuf, etc. They are one more thing to worry about for developers writing code to produce or consume NIEM messages. We would be better off without semantic attributes. (We would keep attributes for NIEM mechanics, like @id and @ref.)

There are 29 semantic attributes used in the NIEM model; 14 in CBRN, 1 in MilOps, 5 in screening, 9 in core. (Attributes in the structures namespace are all part of NIEM mechanics; they do not carry any of the model meaning.) Perhaps we could get rid of them?

# Allow elements with simple type

At present the NDR requires every element to have a complex type, either an extension of a type in the structures namespace, or an extension of a simple type with SimpleObjectAttributeGroup. The practical effect is that every datatype value can be decorated with any of the six attributes in the structures namespace: @id, @ref, @uri, @metadata, @relationshipMetadata, and @sequenceID.

* @id and @ref – I suppose you might want to bother if you had a very large data value appearing several times. Otherwise, why would you ever want a reference to a simple value? It’s a pain in the butt for consumers – you’re expecting MYSTRING, but you could get instead, so you have to code for that.
* @uri – Same as above. I suppose someone might want to assign a fixed URI to a particular string value or whatever.
* @metadata – I imagine metadata will usually be applied to an object, not a data value. But I suppose sometimes people will want to apply metadata to a specific string value or whatever.
* @relationshipMetadata – Doesn’t make sense to apply this to an element with simple content.
* @sequenceID – Sometimes useful, but I want to replace it anyway…

I think it’s unusual to apply any of those attributes to an element with simple content. And I don’t think anything important will break if we allow elements with simple type. That would allow a message designer to declare “this data property will never be a @ref pointer”, for example, and thereby make life easier for consumers.

# Allow external simple elements without an adapter type

Reusing definitions from other communities is a good thing, and we should be trying to make it easier. One way to do that would be to allow external simple types without an adapter. That might look something like this:

<xs:element ref="ext:someProperty" appinfo:conformingExternal="true"/>

A simple type with a documentation string is pretty close to conforming anyway. It does not have the six structures attributes, but those aren’t essential for a simple content element. According to the NDR, an external element in a NIEM message like

<ext:someProperty>FOO</ext:someProperty>

would entail RDF like

\_:parent <https:/the/ext/ns/uri/#someProperty> "FOO"

which seems natural; I don’t see why anyone would expect a different interpretation. However, there are problems with obtaining and subsetting the external schema document, versioning, etc. This might not be worth the bother…

# Rules for namespace URIs

Software that processes NIEM XML schemas and NIEM XML instance data needs to recognize the utility namespaces. The way I’m doing that now is to match against a URI prefix; for instance, I define

STRUCTURES\_NS\_URI\_PREFIX = "http://release.niem.gov/niem/structures/"

and then if uri.startsWith(STRUCTURES\_NS\_URI\_PREFIX), then boom! it’s the structures namespace. Of course I could create an array of all the exact URIs and check each one, but I like this better. So perhaps we need a rule that says “Don’t start an extension schema namespace URI with http://release.niem.gov/”, or something like that;

Another rule candidate: Don’t end your namespace URI with the '#' character. If you do that, we can’t determine the namespace URI from a component URI. That will cause trouble with NIEM JSON, NIEM RDF, and probably any other serialization. For example, if you have NIEM JSON like this:

{

"@context": {

"foo": "https://my/namespace/foo#"

},

"foo:MyField": "BAR"

}

the URI of the namespace for the foo:MyField property could be

https://my/namespace/foo

https://my/namespace/foo#

with no way to tell those apart. This problem goes away if we say “don’t end your namespace URI in #”.

Another rule candidate, or perhaps a convention, is to end your namespace URI with a version number. I’ve wondered if we could use semantic versioning.

## Canonical NIEM model instance

When we are able to work with NIEM model instances, will we still need the NDR? A model instance contains the information we now keep in XML schema:

* what content is required and permitted in a message
* documentation of the meaning of that content
* relationships between components of that content (IS-A, subproperty)

A lot of the NDR is to represent model relationships like IS-A in terms of XML schema. Another chunk of the NDR is to support composition of model components defined by different communities. But the metamodel already represents the required model relationships, and we intend to have tools that will support composition. So maybe all we need from XML schema in the future is validation.

An XML schema generated from a model instance for validation alone could be a lot simpler. For example, it could use xs:choice instead of substitution groups.

The $64 question is: can we do this in two years, for NIEM 6?