# NIEM-UML-3

Potential Improvements & Simplifications

As of 7/3/2014 (Rev 0.1)

# Potential improvements

The NIEM-PMO has requested ideas with respect to NIEM-UML-3 based on usage and development experience with NIEM-UML-2. This document is intended to start the process of collecting these ideas and estimating the impact of resolution on the NIEM-UML-3 schedule and level of effort. Note that none of these issues are required as part of the NIEM-3 RFP but any could be addressed under that process.

Several iterations of this document are anticipated as different stakeholders contribute ideas and then the submission team can estimate impact.

## General Information Models (OMG FTF Issue No: 18174)

### Summary:

NIEM has very specific namespace types such as subset, reference and constraints. These correspond directly to NIEM-UML information model types. User experience has shown that segmenting a model in this way is complex for users and couples the PIM design with the PSM (XSD) representation. It would be preferable that, in the PIM, there was a higher level representation.

### Suggested Resolution:

Add an information model kind “General” that can subset any other model(s), add properties and include property redefinitions of cardinality and type. To support this kind of model, augment the QVT such that a General information model produces, as required, subset schema, constraint schema and extension schema that capture the semantics of the general model using legal NIEM schema.

*Note that allowing type substitution in a subset was added to NIEM-UML subset models in the FTF process.*

### Further clarification:

The intended user is the PIM modeler and stakeholders of that model. In an IEPD or EIEM the model representing “Person” could well be spread out over multiple classes: A subset from NIEM core, one or more “augmentation” types, one or more extensions, multiple substitution groups and multiple constraint schema (or schematron). This is difficult and confusing for the modeler but has even more impact on stakeholders trying to understand the model. Our goal should be that in a particular context one domain concept (e.g. Person) should be represented by exactly one class. The NIEM-UML machinery should then take care of segmenting that class according to the NIEM rules. This goal may or may not be fully realized in NIEM-UML-3, but perhaps some improvements can be made. The specific issues that follow contribute to this overall goal.

## Subset Vs. extension schema

NIEM-UML-2, like NIEM-XML, requires that a subset schema be used that is ONLY a subset of a single NIEM reference model and that any extensions to that concept be put in an extension schema. In a general information model it would be preferable to be able to subset and extend in the same model and not always use the same name as the reference model (in which case a “references” would be required to the reference model). In addition, a domain type should be able to “pull” from multiple reference models. The machinery should then be able to match those properties that came from a reference model and create the subset(s) automatically.

## Augmentation Vs. multiple inheritance

NIEM-XML uses “augmentation” to simulate a multiple inheritance capability (Per the NDR). NIEM-UML makes this easier but still requires the explicit definition of something as an augmentation class that is then inherited (or used as a property) in an extension. NIEM-3 eliminates the need for the property with the corresponding reduction in the explicit capability that the extension must be included in any particular class. Augmentation is complex and hard to understand.

### Possible solution

For NIEM-UML a simpler approach would be to not require the specific declaration of an augmentation type, but rather to allow multiple inheritance. To provide for XSD rules the “augmenting” inheritance would be marked as “augmented by”. This is instead of the current style of marking what is augmented and having an explicit augmentation type. For forward compatibility and reference models “augments” could still be supported. The result would be the generation of an augmentation type as required by the NDR. As a result ANY type could be used as an augmentation, even if not originally defined as such.

## Multiple subsets of the same type

It is common that, within the same IEPD or EIEM, but in different context, there is a need for different combination of properties from reference models. E.g. a “person” that is a “subject” may need different properties that a “person” that is an “officer”.

A NIEM subset (e.g. of person) must have the same name and namespace as the corresponding element in a reference model. This NIEM design choice effectively prevents the same element from being subset differently for different purposes. A “work around” is that a “subset” can be defined that is the unification of all possible properties from all desired uses of the reference element. A “restriction” can then be made for each different purpose and then (if needed) an extension of each of those restricted subsets defined to make the final desired element (e.g. “officer”).

The other solution is just to have the “unified subset” and ignore the unneeded properties (they would then need to be marked as optional in the subset). Ignoring the correct properties is then a matter of documentation.

So while possible, the NIEM solutions are overly complex, couple the different purposes into a unified subset and make the resulting model difficult to interpret. Note that this is an issue with NIEM-XML as well as NIEM-UML.

### Possible solution

When the modeler is creating domain specific types (e.g. “officer” or “subject”) each would be allowed to include an arbitrary subset of reference model(s) properties as well as add new properties. The NIEM-UML machinery would then create the required unified subset, restrictions and extensions. So in the domain model a single “officer” type would fully define the needs of that type.

An ancillary capability would be for the single type to include subset properties from multiple reference models as the properties for a single type are often spread across multiple reference models and augmentations.

## Substitution groups as property holders

Substitution groups allow for multiple variations on a property. As with any property, most substation group “heads” are specific to a particular domain (e.g. PersonRace is clearly a property of Person). In NIEM-UML the definition of a substitution group is required to be in a separate property holder. This distributes the concepts of the domain type (Person) over these property holders and adds substantial complexity. It would be preferable to define the substitution group within the context of its domain (including the “subset” choices). Where the “head” does not have a clear domain it is still acceptable to define a property holder as was the intent for property holders originally.

This behavior was demonstrated in early NIEM-UML prototypes. Mid-stream in the NIEM-UML-2 process there was an issue raised that such a pattern would prevent extension of a substitution group. However, no such issues seems to exist – the same mechanism used now to extend a substitution group (define a subtype with subset properties) could be used with an imbedded substitution group, such an extension may or may not be a property holder based on context.

## Entities Vs data elements

There is an essential conceptual different between identifiable entities like people and organizations and “data values” like strings and phone numbers. The common thought is that the “identity” of a value is identical to its content, and that content can’t change. An entity has an independent identity and properties of an entity can change over time. In NIEM there are some clear values – all those derived from or encapsulating XML primitive types. There are also choices that could be made for other values (like address). UML contains this distinction in classes Vs. data types.

There are multiple advantages to this distinction – for one thing aggregation should always be used for data types (you can’t reference something without an identity). One thing this impacts is the generation of UML diagrams – where associations should be used between entities, but not to values. Since this information is lost the diagrams have to be hand-tuned.

### Possible solution

NIEM-UML should use data type for values and classes for entities. Values should always be shown as UML properties.

## Reducing the need for property holders

Property holders were introduced to solve an “edge case” where a property was defined in a reference model but not used as the property of class in that model. It is unlikely that a UML modeler would ever want a property holder, they would make a class to hold any property.

To solve various problems property holders have grown in usage. They essentially constitute “noise” in the UML model and should not be required except for the edge case.

## Usability of reference models

The NIEM-UML reference models are reverse-engineered from the XML schema – and this makes a great deal of sense for repeatability and reliability. However, the resulting models suffer usability problems. These include property/association choices, use of the “lower level” representation of concepts like roles and augmentations, etc. It also results in auto-generated diagrams. The result is it may be hard to make the IEPD model look like what we would want.

### Possible solution

Three things may help make the reference models more usable: 1) Tuning of the reverse engineering to better make use of the higher level concepts. 2) Manual tuning of these models. 3) Manual creation of reference model diagrams.

Note that the manual creation of reference model diagrams would be a substantial effort – but would be a real aid for users finding and reusing reference concepts.

## Aggregation commitment

NIEM-2 (and NIEM-UML-2) suffered a problem of “early commitment” to aggregation or reference – particularly in the reference models where choice was made and was frequently wrong for a particular usage. NIEM-3 inverted this and now makes no commitment to either and removes the capacity to make such a distinction. This is highly unusual for both UML and XSD modelers who are used to defining aggregation and having XML instances with a particular structure. It is interesting that RDF also does not make any such distinction and this results in the serialization of RDF instances being inconsistent and hard to parse. The result of the current policy is that users will have trouble decoding an arbitrary serialization of any NIEM instance and it will not be clear how to diagram classes.

### Possible solution

NIEM-UML-3 should allow for “no commitment” (the current NIEM policy), particularly in reference models but also allow for a commitment in a specific IEPD or EIEM. It may be acceptable to make “aggregation” an optional constraint and then allow “no commitment” to be either. If “values” are distinguished as data types (see above), values should always be aggregated properties.

The aggregation commitment should be reflected in the schema in some way, TBD.

# Impact

As an initial estimation the above issues are categorized from 1 (minimal impact on effort or schedule) to 10 (essentially a re-design of NIEM-UML).

|  |  |
| --- | --- |
| Issue | Impact Estimate |
| General Information Models (OMG FTF Issue No: 18174) | 5 |
| Subset Vs. extension schema | 4 |
| Augmentation Vs. multiple inheritance | 2 |
| Multiple subsets of the same type | 3 |
| Substitution groups as property holders | 2 |
| Entities Vs data elements | 1 |
| Reducing the need for property holders | 2 |
| Usability of reference models | 4 |
| Aggregation commitment | 2 |