**Experiences Modeling Domain Content in NIEM-UML**

April 2014

**Prepared for:**

Wendy Coyle, PMPDepartment of Homeland SecurityNPPD/CS&C/NSD(703) 235-3941

**Prepared by:**

The Johns Hopkins University   
Applied Physics Laboratory

11100 Johns Hopkins Road

Laurel, MD 20723-6099

Program Contact:

Gina Marshall-Johnson (240) 228-4374   
Technical Contact:   
Jeffrey Stuckman (240) 228-9398

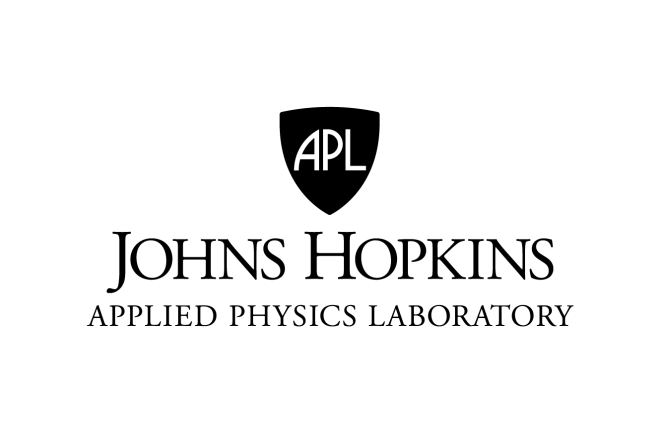


Table of Contents

[1 Executive summary 4](#_Toc385490172)

[2 Introduction 4](#_Toc385490173)

[3 Tools and procedure 5](#_Toc385490174)

[4 Specific cases when modeling XSD content 6](#_Toc385490175)

[4.1 Use of XML simple types 6](#_Toc385490176)

[4.1.1 Modeling with MagicDraw 6](#_Toc385490177)

[4.1.2 Modeling with Eclipse 6](#_Toc385490178)

[4.1.3 Importing with MagicDraw 6](#_Toc385490179)

[4.1.4 Discussion 7](#_Toc385490180)

[4.2 Sharing elements between types 7](#_Toc385490181)

[4.2.1 Modeling with MagicDraw 7](#_Toc385490182)

[4.2.2 Modeling with Eclipse 7](#_Toc385490183)

[4.2.3 Importing with MagicDraw 7](#_Toc385490184)

[4.2.4 Discussion 8](#_Toc385490185)

[4.3 Reusing types 8](#_Toc385490186)

[4.3.1 Modeling with MagicDraw 8](#_Toc385490187)

[4.3.2 Modeling with Eclipse 8](#_Toc385490188)

[4.3.3 Importing with MagicDraw 8](#_Toc385490189)

[4.3.4 Discussion 9](#_Toc385490190)

[4.4 Using and extending NIEM Core 9](#_Toc385490191)

[4.4.1 Modeling with MagicDraw 9](#_Toc385490192)

[4.4.2 Modeling with Eclipse 9](#_Toc385490193)

[4.4.3 Importing with MagicDraw 10](#_Toc385490194)

[4.4.4 Discussion 10](#_Toc385490195)

[4.5 Code types 10](#_Toc385490196)

[4.5.1 Modeling with MagicDraw 10](#_Toc385490197)

[4.5.2 Modeling with Eclipse 10](#_Toc385490198)

[4.5.3 Importing with MagicDraw 11](#_Toc385490199)

[4.6 Reusing code types 11](#_Toc385490200)

[4.6.1 Modeling with MagicDraw 11](#_Toc385490201)

[4.6.2 Modeling with Eclipse 11](#_Toc385490202)

[4.6.3 Importing with MagicDraw 11](#_Toc385490203)

[4.6.4 Discussion 11](#_Toc385490204)

[4.7 Extending domain types 12](#_Toc385490205)

[4.7.1 Modeling with MagicDraw 12](#_Toc385490206)

[4.7.2 Modeling with Eclipse 12](#_Toc385490207)

[4.7.3 Importing with MagicDraw 12](#_Toc385490208)

[4.7.4 Discussion 12](#_Toc385490209)

[4.8 Recursive inclusion 12](#_Toc385490210)

[4.8.1 Modeling with MagicDraw 12](#_Toc385490211)

[4.8.2 Modeling with Eclipse 12](#_Toc385490212)

[4.8.3 Importing with MagicDraw 12](#_Toc385490213)

[4.8.4 Discussion 13](#_Toc385490214)

[4.9 Automatic element naming 13](#_Toc385490215)

[4.9.1 Modeling with MagicDraw 13](#_Toc385490216)

[4.9.2 Modeling with Eclipse 13](#_Toc385490217)

[4.9.3 Discussion 13](#_Toc385490218)

[4.10 Automatic type naming 13](#_Toc385490219)

[4.10.1 Modeling with MagicDraw 13](#_Toc385490220)

[4.10.2 Modeling with Eclipse 14](#_Toc385490221)

[4.10.3 Discussion 14](#_Toc385490222)

[4.11 Associations 14](#_Toc385490223)

[4.11.1 Modeling with MagicDraw 14](#_Toc385490224)

[4.11.2 Modeling with Eclipse 14](#_Toc385490225)

[4.11.3 Discussion 14](#_Toc385490226)

[4.12 Substitution groups 15](#_Toc385490227)

[4.12.1 Modeling with MagicDraw 15](#_Toc385490228)

[4.12.2 Modeling with Eclipse 15](#_Toc385490229)

[4.12.3 Importing with MagicDraw 15](#_Toc385490230)

[4.12.4 Discussion 16](#_Toc385490231)

[4.13 External adapter types 16](#_Toc385490232)

[4.13.1 Modeling with MagicDraw 16](#_Toc385490233)

[4.13.2 Modeling with Eclipse 16](#_Toc385490234)

[4.13.3 Discussion 17](#_Toc385490235)

[4.14 Metadata 17](#_Toc385490236)

[4.14.1 Modeling with MagicDraw 17](#_Toc385490237)

[4.14.2 Modeling with Eclipse 17](#_Toc385490238)

[4.14.3 Discussion 17](#_Toc385490239)

[4.15 Elements and attributes 17](#_Toc385490240)

[4.15.1 Importing with MagicDraw 18](#_Toc385490241)

[4.15.2 Discussion 18](#_Toc385490242)

[5 Specific cases for general tool usage 18](#_Toc385490243)

[5.1 Creating non-default information models 18](#_Toc385490244)

[5.2 Copying content from imported non-NIEM XSD files 18](#_Toc385490245)

[5.3 Utilizing domain models 19](#_Toc385490246)

[5.4 Domain models and IEPD models 19](#_Toc385490247)

[5.5 IEPD packaging 20](#_Toc385490248)

[5.5.1 MagicDraw 20](#_Toc385490249)

[5.5.2 Eclipse 20](#_Toc385490250)

[5.6 Sharing models between multiple projects 20](#_Toc385490251)

[5.6.1 MagicDraw 20](#_Toc385490252)

[5.6.2 Eclipse 21](#_Toc385490253)

[5.7 Collaboration and version control 21](#_Toc385490254)

[5.8 Import and export 21](#_Toc385490255)

[5.9 Reliability and ease of use 22](#_Toc385490256)

[6 Discussion 22](#_Toc385490257)

[6.1 Discussion on the NIEM-UML standard 22](#_Toc385490258)

[6.2 Discussion on NIEM-UML tools and processes 23](#_Toc385490259)

[6.3 Summary 24](#_Toc385490260)

1. Executive summary

The National Information Exchange Model Unified Modeling Language (NIEM-UML) is a new standard that could potentially ease the development of NIEM content and integrate UML modeling into the NIEM development cycle. However, the standard is new and relatively untested, especially for developing NIEM domain content. Using two NIEM-UML development plug-ins—the MagicDraw NIEM-UML plug-in and the Eclipse NIEM-UML plug-in—a variety of models were created to investigate different scenarios that may arise in NIEM-UML development.

The NIEM-UML standard can be used to successfully build NIEM domain content, and both MagicDraw and Eclipse implement the bulk of this standard. However, unexpected or invalid output sometimes resulted unless a knowledgeable user consistently performed certain workarounds and manual steps while using the tools. In some cases, automatically importing existing eXtensible Markup Language (XML) schemas provided an effective base for a NIEM-UML model, although manual corrections to the imported XML were necessary.

Although both tools work with the same standards, the tools use different file formats that, in most cases, are not compatible. For this reason, a NIEM-UML project may become tied to a particular tool throughout its life. Developing NIEM content in NIEM-UML (as opposed to directly developing in NIEM XML) has usability benefits, because the tools that will be used are expressly customized to work with NIEM. However, a developer using NIEM-UML will need significant knowledge of both NIEM and UML. NIEM-UML does not shield the developer from the implementation details of NIEM, and using UML tools requires broad UML knowledge. This document serves to preserve and transfer UML knowledge, specifically with regard to modeling NIEM content in NIEM-UML, and includes technical descriptions for issue resolution targeting practitioners who build NIEM domains using UML.

1. Introduction

NIEM-UML is a relatively new technology that allows NIEM data models to be built and expressed as UML models. NIEM, a standard that facilitates the construction of reusable data model components, most commonly encodes models and instance documents in XML. NIEM-UML allows developers of data models to avoid directly expressing the models as XML Schema Definition (XSD) files. Because XML is still the primary representation for NIEM models and content, UML modeling tools that support NIEM-UML model development can also export NIEM-UML models as XML schemas.

The available tools for developing and exporting NIEM-UML models are also new, and documentation or support articles on the technical and practical aspects of using these tools is limited. Adopting NIEM-UML and using NIEM-UML tools would allow an implementer to avoid some of the complexity of XML schemas. However, this comes at the cost of simultaneously learning how to use a UML modeling tool, the NIEM-UML plug-in for a UML modeling tool, and the NIEM-UML standard itself. Also, the NIEM-UML tools focus on creating Information Exchange Package Documentation (IEPD) models, data models that utilize NIEM Core and domain content, whereas some data modelers develop the domain content itself.

Because of the relative newness of NIEM-UML, it is uncertain whether the standard or the standard’s tooling is fully usable for large-scale data model construction. For this reason, two NIEM-UML tools were used to construct various scenarios that may arise in real-world data modeling. Although a competitive evaluation that would allow the quality of the tools to be ranked was not performed, some shortcomings in each tool were identified, and, in some cases, workarounds were developed. This document describes what was discovered when constructing these scenarios, including information on tools and information on NIEM-UML itself.

This document assumes that the reader has a moderate level of understanding of NIEM and XML schemas. Several audiences may find this document useful:

* *Authors who are deciding whether NIEM-UML should be used for a particular project:* These readers would find the Discussion section most useful, as it discusses general aspects of NIEM-UML that may make it more or less applicable in particular situations.
* *Authors who are choosing a NIEM-UML tool:* Because NIEM-UML models must be converted to XSD files to be used, using a specialized NIEM-UML tool as part of any NIEM-UML effort is essential. Because the available NIEM-UML tools use different file formats, choosing a particular tool before undertaking major development is recommended. Although this document does not recommend one tool over the other, it highlights difficulties encountered that the reader may also encounter. The reader is encouraged to personally assess these difficulties before ruling out a particular tool because the tools are actively maintained, and issues that were encountered in this exercise may not impact every user.
* *Authors who are currently using a NIEM-UML tool for data modeling:* This document describes procedures and workarounds that were successfully used to build UML models and XSDs. Users of NIEM-UML tools can consult this document as a guide to avoid some of the trial and error encountered during this study.

1. Tools and procedure

Using several NIEM-UML modeling tools, numerous UML models were created that reflected various data modeling situations and features of NIEM. Each of these models was exported to a NIEM XSD (using the export functionality built into the modeling tools). The NIEM XSD files were then checked for compliance with XML schema modeling rules, compliance with NIEM XML schema modeling rules, and consistency with what was created in the UML model.

The following tools were used when creating these models:

* MagicDraw: MagicDraw version 17.0.2 with the NIEM-UML plug-in version 17.0.2 SP1 was used.
  + For cases that involved importing non-NIEM XSD files into UML, the Cameo XSD Import plug-in version 1.5 SP3 was used.
* Eclipse: The plug-ins were installed on Eclipse Kepler SR1. Initially, the NIEM-UML plug-in version 1.0.0 as built on 11/25/2013 was used. The NIEM-UML plug-in was upgraded to the latest version from Github, modified on 1/2/2014, and all cases with issues were repeated using this version. The NIEM-UML plug-in depended on numerous other Eclipse plug-ins, notably including the Papyrus version 0.10.1 UML diagramming tool.
  + To import non-NIEM XSD files into UML using Eclipse, the Hypermodel plug-in version 3.6.0 was used. However, a NullPointerException error message was repeatedly encountered when attempting to import portions of XML schemas, and no further attempts to import XSD into Eclipse were made for this reason.

Some cases involved importing content from STIX (Structured Threat Information Exchange) or Cybox (Cyber Observable Expression), two existing XML schemas. For the sake of consistency, both the cases that involved importing and the cases that involved UML modeling from scratch used these schemas as a test case. Before using the schemas, a subset of the schemas’ XSD files was manually created. This subset, based on STIX 1.0, included the types and elements mentioned at various places in this document. This was created so the entire XML schemas, which are substantial, did not need to be modeled or imported at once.

For each of the cases that follow, the relevant types were located in the STIX/Cybox subset XSD files. Then, in the “Modeling” cases, new UML classes were patterned after these types. In the “Importing” cases, all of the STIX/Cybox subset XSD files were imported into one UML model, and only the relevant objects were copied to another UML model.

1. Specific cases when modeling XSD content
   1. Use of XML simple types

Some basic building blocks used when constructing XML schemas are the *simple types*. Simple types store primitive values such as strings, dates, and numbers. Complex, user-defined types are ultimately created by aggregating these simple types.

When constructing a data model in NIEM, NIEM provides its own implementations of simple types (sometimes called proxy types), which augment simple types with required NIEM attributes. When modeling an XSD in NIEM, simple types must be replaced with their proxy counterparts.

Modeling with MagicDraw

In MagicDraw, the STIX Incident Time type, which references several simple types, was modeled. MagicDraw allowed primitive types to be added as attributes to classes, and when exporting to an XSD file, these primitive types were output as NIEM proxy types.

Modeling with Eclipse

In Eclipse, the STIX Incident Time type, which references several simple types, was modeled. Eclipse allowed “XML” types, which were written as NIEM proxy types when exporting an XSD file, to be added to the model.

Importing with MagicDraw

In MagicDraw, the STIX Incident Time type was imported into a UML model. The types that MagicDraw assigned to the relevant attributes were not the standard XML or UML primitive types; instead, MagicDraw created new UML classes to represent these types. The types of all of these attributes had to be manually changed to make the UML model carry these primitive types. *Note that in all of the remaining cases that involve importing with MagicDraw, it is implied that this correction was made.* In addition, all properties were imported with an aggregation setting of “none.” The aggregation of these properties was changed to “shared” so the elements in the generated XSD would not be emitted as references. *In all of the remaining cases that involve importing with MagicDraw, it is also implied that this correction is made.*

Discussion

When building a NIEM data model that incorporates XML simple types, simply using the primitive types built into the UML modeling tool is sufficient for creating valid NIEM-XML. However, if an XSD is imported into NIEM-UML with MagicDraw, all references to simple types must be individually repaired.

* 1. Sharing elements between types

XML schemas are composed of types (which define datatypes or classes) and elements (which are named tags that hold instances of these types). However, NIEM-UML models are composed of classes and relationships (or attributes) between these classes. When exporting a NIEM-UML model as an XML schema, these relationships will be exported as elements. However, if several relationships on a UML diagram have the same name, it is not possible to export them all as-is because all elements in a NIEM XSD must have unique names.

NIEM-UML defines the concept of a “Property Holder,” which allows attributes to be defined that are shared among multiple classes. This represents that an element with a given name will be shared among several types in the exported XSD.

Modeling with MagicDraw

In MagicDraw, the STIX Incident and Intended Effect types were modeled, both of which reference a confidence assessment.

When properties with identical names were defined in both of these classes, only one element was emitted in the exported XSD, as expected. This behavior was observed both with and without the use of property holders. In addition, when a property in one package was linked to a property in another package (through the use of a References relation), one schema (the one corresponding to the client) utilized another schema’s (the one corresponding to the supplier) element in the exported XSD. However, drawing a References relation between classes in different packages did *not* result in shared elements in the exported XSD, as the NIEM-UML specification describes.

Modeling with Eclipse

In Eclipse, the STIX Incident and Intended Effect types were modeled and the same cases were evaluated. As with MagicDraw, using properties with the same name resulted in shared elements in the XSD. In addition, drawing references relations between either classes or properties allowed elements to be shared between schemas in an exported XSD.

Importing with MagicDraw

The STIX Incident and Intended Effect types were imported from the XSD into NIEM-UML. Instead of representing properties with UML attributes on classes, properties were represented with association relations linking the classes. When two elements had identical names in the XSD file, two relations appeared in the imported NIEM-UML model. When exporting the model back to NIEM XSD, only one element was written, as expected.

Discussion

There is no need to ensure that properties/elements with identical types and identical names are emitted as a shared element in a NIEM XSD. This is because this situation is automatically detected and corrected in both tools. Both tools are capable of allowing a class in one package to utilize a property defined in another package; however, when using MagicDraw, relations must always be drawn between individual properties and not their classes.

Note that the case where two properties/elements have the same name but differing types was not tested. Because all elements in a NIEM XSD must be globally defined and uniquely named, converting such a model to NIEM will require the property to be manually corrected.

* 1. Reusing types

When developing XML schemas or NIEM data models, types and classes should be reusable components that can be used in multiple places in the model. It should be possible for one class to be related to another class in multiple ways, where each relationship specifies that the class is fulfilling a different role or purpose.

NIEM-UML allows relationships between classes to be specified in two ways—through the use of UML properties or by creating UML relations that have a particular role specified.

Modeling with MagicDraw

In MagicDraw, the STIX Incident and Information Source classes were modeled in UML. A STIX incident contains multiple information sources, such as the incident reporter and responder, both of which were implemented in our model.

Two information source properties were added to the incident class—one reporter and one responder. In the NIEM XSD that was exported from the UML, two information source elements were generated as expected.

Modeling with Eclipse

When modeling the STIX Incident and Information Source classes in Eclipse, the result was the same as in MagicDraw—separate elements for the reporter and responder appeared in the exported NIEM XSD.

Importing with MagicDraw

In MagicDraw, the STIX Incident and Information Source types were imported from the existing XSD into UML. The import process generated two separate relations between the incident and information source types—one for the reporter and one for the responder. The “Role of InformationSourceType” attribute of the relations disambiguated the two otherwise identical relations, specifying which one was for the reporter and which was for the responder. The exported NIEM XSD contained two elements for these relations, as expected.

Discussion

No special care needs to be taken when implementing reusable types in NIEM-UML, as the relevant cases are natively handled by the available tools.

* 1. Using and extending NIEM Core

NIEM contains a number of commonly used objects, such as date ranges and persons, that are included in the NIEM Core package. NIEM domain content must utilize NIEM Core content when available, rather than reimplementing the content.

When creating NIEM content in UML, a mechanism is needed to reference content from NIEM Core, meaning that it also must be possible to represent a subset of NIEM Core in UML. In addition, it should be possible to export a subset schema (an XML schema that contains a subset of the content available in NIEM Core). Although a NIEM Core subset schema would not be packaged with a NIEM domain, a creator of a NIEM IEPD model utilizing the domain will create a subset schema as well.

Finally, it should be possible to extend (subclass) classes from NIEM Core. Extending a NIEM Core class is similar to utilizing it, except the appropriate relationship must be drawn between the NIEM Core class and the class being extended.

Modeling with MagicDraw

In MagicDraw, the STIX Information Source type was modeled as a UML class, replacing the use of CIQ (Customer Information Quality) with the NIEM Core Person and Organization types. To use these NIEM Core types, a NIEM Core subset model was created, and selected attributes of a NIEM Core Person and Organization were added. The MagicDraw user interface allowed particular NIEM Core types and properties to be selected, and these types and properties were automatically added to the subset model. The exported subset schema XSD contained the NIEM Core subset as expected.

Next, a STIX Incident was modeled as a UML class. The NIEM Core Incident class was added to a subset schema, and a generalization relation was added so the STIX Incident extends the NIEM Core Incident. The resulting NIEM XSD contained an incident type that extended the NIEM Core Incident type.

Modeling with Eclipse

In Eclipse, some initial tests confirmed that NIEM Core subset schemas could be created. However, some issues were identified, making some NIEM Core subset schemas difficult or impossible to build:

1. The user interface for selecting and configuring a subset class from NIEM Core was unreliable. Many user interface elements (some needed to be manipulated and some did not) were greyed out, even if the setting corresponding to the user interface element could be edited elsewhere.
2. Although the user interface limited the user to only adding classes that actually exist in NIEM Core, the superclass was not automatically set (and could be set to arbitrary values), which would result in an invalid subset. Also, some classes (such as PersonNameTextType) were missing completely.
3. The user interface did not assist the user in adding valid properties to these NIEM Core classes. Arbitrary properties could be added to the NIEM Core classes with arbitrary types. This would allow the user to create an invalid subset of NIEM Core, which results in increased difficulty when adding properties that are valid.
4. When exporting the model to a NIEM XSD, the subset schema was not written into a “subset” directory structure, which is the standard location for a subset schema in an IEPD.

Importing with MagicDraw

In MagicDraw, the STIX Incident type was imported from the XSD into UML. The NIEM Core Activity type was then added to a subset model, and the STIX Incident type was made to inherit from the NIEM Core Activity type with a generalization relation. The resulting NIEM XSD that was exported included a properly formed subset schema of NIEM Core containing the Activity type and the selected properties, and the exported Incident class was derived from a NIEM Core Activity.

Discussion

When using MagicDraw, it was possible to create and utilize NIEM Core subset schemas without issue. The user interface allowed the user to choose the classes and elements that should be included in the schema. Although, in many cases, creating a valid subset schema in Eclipse was possible, a number of issues made it more difficult to create a NIEM Core subset schema that would ultimately be valid.

When subclassing a NIEM Core class by using a generalization relation, the way to draw the relation was not immediately obvious because the two classes resided in different packages. However, in both UML tools, UML packages are decoupled from the diagrams that depict their classes. A class from one package can be added to another package’s diagram without affecting the class’s actual package or its presence on its original diagram. By adding the existing NIEM Core class to the extension schema’s diagram, the relation between the two classes could be drawn.

* 1. Code types

NIEM allows code types, or enumerations, to be defined in IEPDs or domain schemas. Code types in a NIEM XSD are based on XSD enumerations; however, the enumeration must be created in a particular way to ensure that the required NEIM attributes are present. Code types are normally implemented by creating two NIEM artifacts—a simple XML type that contains the enumeration, and a complex XML type that adds the NIEM attributes. This is of interest when creating content in NIEM-UML, as the NIEM-UML tool should ensure that this pattern appears in the exported XSD.

Modeling with MagicDraw

The STIX Confidence Value enumeration was implemented in NIEM-UML by creating a UML enumeration with the appropriate enumeration members. When applying the “complex code type” NIEM stereotype to the UML enumeration, the code type in the exported XSD was created with the required NIEM attributes, making it a legal type to use from NIEM. The code type was not split into simple and complex artifacts (as is customary when creating a NIEM XSD); however, the NIEM type was still valid. If the complex code type stereotype was not applied, the resulting type was not a valid NIEM type.

Modeling with Eclipse

The STIX Confidence Value enumeration was implemented in NIEM-UML by creating UML enumeration-stereotyped classes with the appropriate enumeration members. It was necessary to create two enumeration-stereotype classes, one generalized from the other, with the more general class containing the enumerated values. This resulted in two artifacts being created in the NIEM XSD (one simple type and one complex type), forming a valid NIEM code type. When creating the enumeration with only one UML class, the resulting NIEM XSD did not contain a valid NIEM enumeration type. This issue becomes apparent when running the PetAdoption test case that comes with the Eclipse NIEM tool—the XSD generated in the example is not NIEM compliant, as a simple type is directly referenced by an element.

Importing with MagicDraw

The STIX Confidence Value enumeration was imported from XSD to UML using MagicDraw. The initial import did not result in a valid UML enumeration being created, as the restriction of simple content (linking the STIX vocabulary and the STIX enumeration types) was not captured in the UML model. The enumeration was repaired and incorporated into a NIEM-UML model by deleting the vocabulary type, changing the enumeration type to a complex code type, and setting the “leaf” attribute on the enumeration type, which caused MagicDraw to emit a complex rather than a simple type when exporting the NIEM XSD. References to the old vocabulary type were then updated to reference the enumeration.

* 1. Reusing code types

As in the “Reusing types” case, it should be possible for multiple classes to reference the same code type (enumeration). Only one copy of the enumeration should be written in the exported NIEM XSD, with multiple types referencing it.

Modeling with MagicDraw

In MagicDraw, a STIX incident class was modeled, which included the Incident Impact Assessment. Two properties of the Incident Impact Assessment (Brand and Market Damage and Loss of Competitive Advantage) were modeled, both referencing the Security Compromise enumeration. In the generated NIEM XSD, the enumeration was successfully shared between both elements (properties) of the impact assessment.

Modeling with Eclipse

The same procedure was followed as with MagicDraw, with the same outcome.

Importing with MagicDraw

A similar procedure as the one in the “Code Types” case was preformed, except on an imported enumeration that was used by multiple elements. The result was the same as in the other case, with the fact that the code type was used by multiple elements having no bearing on the outcome.

Discussion

When it comes to reusing types, enumerations in NIEM-UML are not treated any differently than ordinary classes. Only one instance of the enumeration is written in the exported XSD, no matter how many times it is utilized in the model.

* 1. Extending domain types

Aside from extending (subclassing) types already defined in NIEM Core, it is also useful to create custom types as part of a NIEM model and then extend those types.

Modeling with MagicDraw

In MagicDraw, a STIX Impact Assessment and a STIX Statement was modeled in UML. A generalization relation was created so the Impact Assessment subclassed the Statement. In the exported XML schema, the Impact Assessment type was generated as an extension of the Statement type as expected.

Modeling with Eclipse

In Eclipse, a STIX Intended Effect and a STIX Statement was modeled in UML. A generalization relation was created so the Intended Effect subclassed the Statement. In the exported XML schema, the Intended Effect type was generated as an extension of the Statement type as expected.

Importing with MagicDraw

In MagicDraw, the STIX Intended Effect and STIX Statement types were imported from the XSD into UML. Several changes were then made to the UML to establish the subclass relationship between the Intended Effect and Statement classes. First, the changes described in the “Code types” case were made so a complex code type would appear in the exported XSD. Then, a new class was created representing an Intended Effect Statement, subclassing the Statement class and utilizing the Intended Effect code type. As anticipated, the resulting XSD contained an Intended Effect type that extended the Statement type.

Discussion

Both tools were able to create subclasses in NIEM-UML models as expected. There was no difference between the way that NIEM Core types are extended and the way that user-defined types are extended.

* 1. Recursive inclusion

XML schemas can specify recursive relationships, where an infinite chain of elements can be created in an instance document if a circular series of references is constructed between types. It should be possible for NIEM-UML to specify such relationships.

Modeling with MagicDraw

In MagicDraw, the STIX Confidence and Confidence Chain types were modeled in UML. Through a recursive relationship, this allows the user to construct an infinitely deep confidence assessment chain. In the exported NIEM XSD files, the recursive relationship was reflected as expected.

Modeling with Eclipse

In Eclipse, the STIX Confidence and Confidence Chain types were modeled in UML. The result was identical to the result of the MagicDraw case.

Importing with MagicDraw

With MagicDraw, the STIX Confidence and Confidence Chain types were imported from the STIX XSD to NIEM-UML. In the exported NIEM XSD, the recursive relationship was reflected without having to make any manual modifications to the data model.

Discussion

There were no special considerations when creating NIEM-UML models with recursive relationships (above and beyond the ordinary considerations when creating any NIEM-UML model).

* 1. Automatic element naming

The NIEM Naming and Design Rules (NDR) mandate that NIEM content be named with a particular naming scheme. For example, properties referring to dates and times must have “DateTime” at the end of their names.

If NIEM-UML modeling tools do not automatically enforce these naming conventions (either while creating the diagram or while exporting the diagram to XSD), users of the tools must ensure that these naming conventions are satisfied.

Modeling with MagicDraw

MagicDraw was used to model the STIX Incident Time class as NIEM-UML. This class has numerous properties that refer to dates and times. Upon creating these date and time properties, the tool automatically added the DateTime suffix to the property name. However, if this suffix was later deleted, it would not automatically be recreated on the model or in the XSD.

Modeling with Eclipse

Eclipse was used to model the STIX Incident Time class as NIEM-UML. When adding properties that referred to dates and times, the tool did not automatically add a suffix to the property name to satisfy the naming rules. When exporting the model to an XSD, the suffix was not automatically added.

Discussion

When using both NIEM-UML tools, care must be taken to ensure that the naming conventions are satisfied and that any required prefixes or suffixes are not inadvertently removed. This requires the user to be familiar with the NIEM naming rules. However, when using the MagicDraw tool, the tool attempts to assist user in this process by adding a required suffix by default.

* 1. Automatic type naming

Aside from requiring the elements be named with a particular convention, the NIEM NDR also specifies a naming convention for types. For example, names of complex types must end with “Type,” and names of simple types must end with “SimpleType.”

Modeling with MagicDraw

MagicDraw was used to model a STIX Confidence class in NIEM-UML, including the enumeration used for the confidence value. When creating classes in UML, the tool automatically made class names conform with this convention by adding “CodeType” or “Type” to the ends of the names. If the “Type” suffix was manually deleted from a class name in the UML, it was automatically regenerated when exporting a NIEM XSD.

Modeling with Eclipse

Eclipse was used to model a STIX Confidence class in NIEM-UML, including the enumeration used for the confidence value. When exporting the UML model as a NIEM XSD, proper suffixes, such as “CodeType” or “CodeSimpleType,” were added to type names, even if the suffixes were not present in the UML model. In some cases (such as when adding an enumeration to the diagram), the tool also added the appropriate suffix to the class name when adding it to the diagram.

Discussion

Both NIEM-UML tools added the appropriate suffixes to the names of types that were exported in an XSD. When developing a UML model, less effort was required to ensure that types were named appropriately than to ensure that elements were named appropriately.

* 1. Associations

NIEM defines the concept of association types, which are types that specify how other objects are related to each other. Although association types resemble ordinary NIEM types, they are linked to the associated objects by reference (as opposed to the linked objects being embedded inside the associated object). This allows a single object to participate in multiple associations.

NIEM-UML provides two notations for creating associations: UML association classes and ordinary UML classes with association stereotypes.

Modeling with MagicDraw

In MagicDraw, STIX Indicator and STIX Incident classes were modeled. These STIX classes included Related Incident and Related Indicator properties, which specify relationships between indicators and incidents that are otherwise independent. Both kinds of NIEM-UML associations were used to model these relationships—UML association classes and ordinary UML classes with association stereotypes.

In both cases, the association class was exported to the NIEM XSD as expected. In particular, the references to the indicator and incident were reference properties—rather than value properties—meaning that the indicator and incident could exist independently from the association. In the MagicDraw tool itself, validation-related warning messages appeared when using UML association classes, even though issues with the UML or the exported XSD could not be identified.

Modeling with Eclipse

In Eclipse, STIX Indicator and STIX Incident classes were modeled using UML association classes and ordinary UML classes with association stereotypes. When using ordinary UML classes with association stereotypes, the association types in the exported NIEM XSD were created as expected. However, issues were encountered when exporting models with UML association classes to XSD. Whenever a UML association class was created, one of the two elements participating in the association was missing in the resulting type in the XSD.

Discussion

Association types effectively allowed independent objects to be related to each other in NIEM models. Both NIEM-UML tools were capable of generating association types in the exported XSD. However, issues were encountered in both tools when creating UML association classes, as opposed to ordinary UML classes with the association stereotype.

* 1. Substitution groups

Substitution groups in XML give schema users a choice of which element should appear in a particular location. With substitution groups, there is no need to create a single list of all the elements that can appear at a given point. Rather, each legal element declares itself to be a member of the group, allowing the group to be expanded without maintaining a single list.

Creating a substitution group in NIEM-UML is a multistep process, requiring the creation of a property holder that declares all members of a substitution group. A reference to the property holder is then created wherever the substitution group is utilized.

Modeling with MagicDraw

Using the Cybox objects as a guide, MagicDraw was used to implement a basic File and Address object in NIEM-UML. Both of these objects subclassed a base cyber object properties type, which was created. A property holder that defined the substitution group was then created. The first property in the property holder (the head of the substitution group) was created with the “derived” and “derived union” settings, and it was assigned the type of the base cyber object properties. The other properties in the property holder, referencing the address and file types, were set up so their “subsetted property” was the head of the substitution group.

Other objects were then made to utilize the substitution group by creating a property with a name matching the name of the head of the substitution group. The “derived” or “derived union” settings were not set on these properties. In the resulting NIEM XSD, an XML substitution group was created as expected. Note that, unlike in examples in the NIEM-UML specification, it was not necessary to create a References relation to the property holder.

Modeling with Eclipse

Using the Cybox objects as a guide, Eclipse was used to implement a basic File and Address object in NIEM-UML. The model was created in the same way that the MagicDraw model was created. As in the MagicDraw case, the resulting XSD contained a valid XML substitution group.

Importing with MagicDraw

Using MagicDraw, the Cybox File and Address objects were imported into NIEM-UML. As implemented in Cybox, using objects such as the File and Address objects requires the type of the object to be specified using the xsi:type mechanism. To provide each object with its own tag (which would allow different objects to be specified in XML instance documents without using xsi:type), a property holder was created, which allowed object-specific properties to subset the generic object property. When exporting a NIEM XSD, this property holder caused an XML substitution group to be created. This allowed Cybox objects to be created without using the xsi:type mechanism to specify the type of each object.

Discussion

Both NIEM-UML modeling tools allowed substitution groups to indirectly be created, through the use of property holders. The property holder had to be carefully constructed; however, following the procedure described above resulted in valid XML substitution groups being created in both cases.

* 1. External adapter types

NIEM allows pre-existing, non-NIEM XML schemas to be included from NIEM schemas. To accomplish this, NIEM requires that external adapter types be created. External adapter types are XML types that are composed of XML elements defined in non-NIEM schemas. Although defining external adapter types in XML is straightforward, the process for defining these types in UML is less obvious because NIEM-UML models are not composed of direct references to XML namespaces or elements.

Modeling with MagicDraw

In MagicDraw, a NIEM-UML model was created that imported the STIX TTP (Tactics, Techniques, and Procedures) type, which is not a NIEM type. To do this, a UML package was created, and this acted as a surrogate for the STIX schema containing the TTP type. This UML TTP package was placed in the same folder in the model that contained the UML packages that were created by default. The InformationModel stereotype was then applied to the TTP package, the “Is Conformant” setting was set to “false,” and the “Target Namespace” setting was set to match the actual namespace of the STIX TTP schema.

This surrogate package was then integrated into the rest of the NIEM model. First, the symbol for the TTP package was added to the NIEM Overview Diagram. A Usage relationship was drawn from the Extension Model to the TTP package (although the adapter type was also observed to work even when this relationship was not added). A TTP class was added to the TTP package, representing the STIX TTP type that will be utilized (although no properties were added to this type). A TTP property holder was also added to the TTP package. A TTP property was added to the property holder, representing the TTP element to be included in the external adapter class.

The TTP property holder was then added to the Extension Model diagram. (As in the “Using and Extending NIEM Core” case, adding an object to a second diagram does not result in the actual object being copied.) A NIEM adapter type was created by using the AdapterType stereotype, and a property matching the name and type of the property in the property holder was created. Finally, a “References” relation was drawn from the property in the adapter type to the property in the property holder.

In the exported NIEM XSD, an external adapter type was created as expected. This adapter type referenced an element in the STIX namespace. The STIX namespace was implemented by the surrogate UML package, which was exported as an XML schema at the top level of the IEPD schema content.

Modeling with Eclipse

In Eclipse, a similar procedure to the one performed in MagicDraw was used to create an external adapter type and import the STIX TTP type. One difference was that, instead of adding the TTP package’s symbol and the Usage relationship to a “NIEM Overview Diagram,” the symbol and relationship were added to the extension model diagram, matching the convention used by other NIEM diagrams in Eclipse. Also, the Namespace stereotype was applied to the package, rather than the InformationModel stereotype.

The resulting NIEM XSD was similar to the one generated by MagicDraw. The adaptor type was created properly and associated with an XML schema representing the surrogate STIX package.

Discussion

In both NIEM-UML modeling tools, it was possible to create a NIEM external adapter type that referenced an element from STIX, even though the STIX schema was not implemented in UML. Creating the surrogate UML package representing the STIX TTP schema was done to work around the inability to reference types or elements that lack a UML representation.

If an external adapter type is created in NIEM-UML by following this procedure, some manual modifications will have to be made to the resulting schema. The XML schema generated for the surrogate UML package must be deleted, and imports of this schema must be changed so they import the actual STIX TTP schema.

* 1. Metadata

NIEM has a built-in metadata mechanism that allows metadata markings to be applied to any elements in a document. The technical requirements that NIEM XML schemas must follow ensure that this metadata can be applied in a consistent way to any NIEM model.

To use metadata in conjunction with a NIEM model, the model must allow instances of metadata classes to be created.

Modeling with MagicDraw

With MagicDraw, the STIX Header and STIX Incident class were implemented in NIEM-UML, including the portions of the header that would normally be used to store STIX data markings. The data marking structure was then replaced with a NIEM-UML metadata class. Finally, a property representing a data marking was added to the metadata class.

The resulting NIEM XSD allowed the metadata to be inserted into XML instance documents. Metadata references could also be applied at the level of each element in the document, allowing the metadata to be used.

Modeling with Eclipse

With Eclipse, the STIX Header and STIX Incident class were implemented in NIEM-UML. A metadata class was then created using the same procedure used in MagicDraw. As in MagicDraw, the resulting XSD allows metadata to be inserted.

Discussion

The feature that allows metadata to be attached at arbitrary points of a document is built into NIEM—no action is required on the part of NIEM content developers to make this feature available. For this reason, creating a NIEM metadata class in NIEM-UML is fairly straightforward.

* 1. Elements and attributes

Many of the situations tested involved importing XSD files into NIEM-UML models with MagicDraw using the Cameo XSD Import plug-in. Some types in the imported XSD files utilized attributes to represent data instead of only using elements. In NIEM, elements are more commonly used than attributes, and using elements provides greater flexibility to apply metadata across a document.

Importing with MagicDraw

MagicDraw was used to import the STIX RelatedIndicators type into NIEM-UML. This type contains a “scope” property, which is implemented as an XML attribute. The NIEM-UML was then exported to a NIEM XSD. The resulting scope property was implemented as an element, rather than as an attribute.

Discussion

When importing XSD files into NIEM-UML and then exporting a NIEM-compliant XSD, attributes were automatically changed into elements.

1. Specific cases for general tool usage
   1. Creating non-default information models

In both MagicDraw and Eclipse, when a NIEM-UML project is created, UML packages and diagrams are created for an exchange model, an extension model, and a NIEM Core subset. When exporting the NIEM-UML model in an XSD format, each of these models are converted to XSD and placed in their own schema files.

In some cases, such as when creating reusable content that can be shared between multiple NIEM IEPD models, it is desirable to create additional packages representing additional models that can also be converted to XSD. In both MagicDraw and Eclipse, additional packages of UML content were successfully created, and the XSD export process detected and exported this content along with the default packages created by the tool.

To create an additional information model, a UML package was created and placed in the UML model in the same place where the automatically created packages resided. (Note that this operation is performed from a UML tree view, rather than from a diagram.) An InformationModel stereotype was then applied to the package, and NIEM-UML properties were set as desired (for example, the purpose of the model—such as for storing domain content or IEPD content—is set in this manner). Next, a UML class diagram was created and placed within the package to provide an interface to create additional UML content. Finally, the package was dragged onto the project’s main diagram.

To specify how packages utilize content residing in other packages, “Use” or “Usage” relationships were drawn to and from the newly created UML package. In MagicDraw, these relationships are created from the NIEM Overview Diagram. In Eclipse, each of these relationships is placed on the diagram corresponding to the package utilizing another package’s content. This requires placing a copy of one package’s icon on another package’s diagram; however, this only copies the icon and does not result in the actual package being copied.

* 1. Copying content from imported non-NIEM XSD files

Using the Cameo XSD Import plug-in for MagicDraw, existing XML schemas (more specifically, the STIX and Cybox schemas) were imported into UML class diagrams. When performing the import process, a new UML model is automatically created, and the UML content is added to this model. However, the UML content must be copied to a separate NIEM-UML model if it is to be utilized from NIEM-UML.

Using the copy and paste functions of MagicDraw, content was successfully copied from an imported XSD model to a NIEM-UML model. The copy and paste operation was performed from the trees of UML objects (rather than from the diagrams), and the objects were pasted into the Extension package of the NIEM-UML model. When copying multiple objects, all of the content was selected and copied at once, rather one at a time. This caused relations between classes (as well as the classes themselves) to be copied.

After copying, the copied objects were dragged from the UML tree to the diagram in the NIEM-UML model. This is because objects are not automatically added to the diagram when they are added to the model.

* 1. Utilizing domain models

In the previous section, the procedure for utilizing a subset of NIEM Core classes from a NIEM-UML model was described. This procedure was successfully performed with the MagicDraw tool. However, utilizing subsets from other NIEM reference models (domains) was not successful when using either tool.

When creating a NIEM-UML model with MagicDraw, the user is given the choice between including NIEM Core and including all NIEM reference models. However, when including all NIEM reference models, the tool locked up for about an hour and then crashed with an error message.

In the Eclipse tool, a feature that allowed subsets of reference models (other than NIEM Core) to be included in a NIEM-UML model could not be found.

* 1. Domain models and IEPD models

In NIEM, several types of models can be created. Two types of models are domain (reference) models, which contain content that may officially become part of NIEM, and IEPD models, which are developed in support of a particular exchange. Although the XML schemas for both types of models are created in a similar way, there are differing packaging requirements for each.

Both the MagicDraw and Eclipse NIEM-UML tools appear to have primarily been developed to generate IEPD models. MagicDraw is also capable of generating EIEM (Enterprise Information Exchange Model) models, which are slightly different from IEPD models. However, neither tool creates models or XML artifacts that are directly usable as a domain.

Even though the NIEM-UML tools are geared toward producing IEPD models, content ultimately destined for a domain could still be modeled with them. XML content that is packaged in IEPD form can easily be recopied and repackaged into another schema. Also, as discussed previously, information models can be freely created within a UML project, potentially allowing the user to put the models in a form that more closely resembles what is needed to construct domain content. Domain content would ultimately be included in IEPDs through a subsetting mechanism. As described earlier, subset schemas for domains already part of NIEM could not be created; therefore, subset schemas for custom domain content could not be evaluated.

In summary, the existing NIEM-UML development tools could be used to develop domain content (as opposed to just developing IEPD content); however, manual refactoring or repackaging may be required to put the content in a form that could be released.

* 1. IEPD packaging

Both the MagicDraw and Eclipse NIEM-UML tools attempt to package exported XSD content in a standardized directory structure. NIEM requires that IEPD content be packaged and distributed in this structure.

MagicDraw

When exporting an IEPD from MagicDraw, the full directory structure for extension, exchange, and subset XSD files is created as expected. An MPD catalog file is also created, as well as an empty placeholder for a sample XML instance document.

Eclipse

When exporting an IEPD from Eclipse, some NIEM schemas are put in the full directory structure. However, the main extension, exchange, and NIEM Core subset schemas are placed at the top level of the IEPD, rather than being put in a directory structure. The exported IEPD also contains an MPD catalog file, a placeholder changelog file, and a template sample XML instance document containing empty elements.

* 1. Sharing models between multiple projects

NIEM encourages the reuse of model content whenever possible. Content that has officially been made part of NIEM can be reused by creating subset schemas of NIEM Core or NIEM domain schemas.

It is also desirable for locally created NIEM-UML content to be reused when creating other local NIEM-UML models. This could be done to internally split apart a complex schema development effort, or to utilize a proposed reference model in an IEPD model, when the reference model has not yet become an official NIEM model. NIEM formalized the concept of an EIEM, which can be used to package NIEM content that is intended for local reuse.

MagicDraw

MagicDraw provides templates to create NIEM-UML EIEM and IEPD models. To test the ability to share models between projects, an EIEM was created and saved as UML, without exporting it to an XSD.

The MagicDraw sharing functionality was used to “share” the UML extension package of the EIEM model. This shared content was then “mounted” into the IEPD model (which resided in a different MagicDraw project) at the same level as the other UML packages in the IEPD model. This shared content was successfully utilized in the same way that any other non-default information model content could be utilized.

Caution was required when utilizing content from NIEM Core in an EIEM. When creating a NIEM Core subset in an EIEM, the content was not automatically transferred to the NIEM Core subset schema of the IEPD. If matching classes were not added to the IEPD subset schema, the resulting XSD files did not contain all of the required content.

Eclipse

Although the Eclipse NIEM-UML plug-in does not provide the ability to create an EIEM, the base UML modeling functionality allows content from other projects to be referenced by importing packages. This would potentially allow content sharing to be implemented by sharing content defined in one IEPD model with another. However, when generating an IEPD from a model that references imported packages, the imported packages are not included in the PSM or the resulting XSD. This causes the generation of XSD files that refer to nonexistent types.

* 1. Collaboration and version control

When working on collaborative projects, using a version control system allows multiple developers to simultaneously edit an artifact, reducing the chance of accidental data loss. Using a text-based file format (rather than a binary format) often makes it easier to implement version control, as conflicting changes can be analyzed (and sometimes reconciled) more readily.

MagicDraw stores UML models in a binary, compressed file format. However, MagicDraw has other features to facilitate collaboration. There is built-in support to store projects in the CVS version control system (although this system is obsolete and less widely used). There is also built-in functionality to compare two projects, which is beneficial when resolving conflicts. MagicDraw also has its own team server for collaborative model development, although this is a separate add-on.

Eclipse stores UML models in a text-based file format, which is more suited for storage in general-purpose version control systems. The Eclipse UML tooling has no special functionality for version control or collaboration. However, version control plug-ins that already exist for Eclipse may be usable in conjunction with the UML modeling features, although this feature was not tested.

* 1. Import and export

Because MagicDraw and Eclipse use different file formats, there is the danger that developers of NIEM content could become “locked in” to using one tool or another. To a degree, this could be alleviated by a reliable import and export feature.

MagicDraw could not import or load any UML content that was saved by Eclipse. However, the MagicDraw NIEM-UML plug-in has the ability to reverse-engineer an XSD NIEM IEPD file, turning NIEM content in XML format back into a model. This function was tested on NIEM XSD content generated by NIEM, and it functioned reliably, even when the content included NIEM Core subset schemas. Although some information (such as the layouts of UML diagrams) is lost when converting UML content to an XSD and back, this is one potential solution if there is a need to migrate content from Eclipse to MagicDraw.

To transfer content from MagicDraw to Eclipse, MagicDraw contains a function to export models in Eclipse format. The UML model was imported into the Eclipse workspace successfully; however, the NIEM-UML tooling in Eclipse failed to process this imported content in order to export it to an XSD. The NIEM-UML tooling in Eclipse also has the ability to reverse-engineer a NIEM XSD into a UML model. When reverse-engineering an IEPD that was originally generated by the Eclipse tool, the model was created successfully. However, Eclipse did not successfully import an IEPD that was originally generated by MagicDraw. The nature of the difference between the IEPDs that caused one import to succeed and the other to fail was not determined. In the end, NEIM-UML content created into MagicDraw could not be imported into Eclipse by any means.

* 1. Reliability and ease of use

Both NIEM-UML tools allowed content to be built through a graphical model editor. This allowed diagrams to be laid out in an intuitive manner, rather than using tree-based or list-based user interfaces. However, a thorough understanding of UML was still required. In both tools, the content displayed on the diagram could become desynchronized with the content in the underlying model, with the underlying model containing classes or relations that had no symbols on the screen. This frequently resulted in spurious content appearing in the XML instance documents—content that originated from UML nodes that were deleted from the diagram but still existed elsewhere.

Some issues related to reliability and error messages existed in both tools. When illegal UML content, or UML content that would result in an illegal NIEM instance document, was added to a model, there were frequently no error messages (or there were non-descriptive error messages that did not help identify the problem). Also, the Eclipse tool frequently ran out of memory with continuous use and froze or crashed. This could be mitigated by periodically restarting Eclipse, although the freezes occurred without much advance warning.

1. Discussion
   1. Discussion on the NIEM-UML standard

The NIEM-UML standard has been proposed as a way to allow data modelers to focus on the business model being developed, rather than on the technical aspects of the NIEM NDR and XML Schema. Although the NIEM-UML tools do take some focus off some technical aspects of NIEM, many of these technical concerns “leak into” NIEM-UML in the end. For example, users of NIEM-UML must take special care to build complex enumerations (rather than simple enumerations) to avoid producing noncompliant XML Schema Documents. Other such aspects include having to specify the aggregation of each UML property (deciding whether content should be included by value or by reference), having to use property holders (a consequence of global element definitions in XML Schema), and the substitution group mechanism in NIEM-UML (which mirrors the corresponding mechanism in XML Schema). Although the data modeling process could be started in UML without much knowledge of NIEM, the UML model will still need much attention by someone with a thorough understanding of NIEM, XML Schema, and UML in order for the end result to be satisfactory.

Although NIEM-UML might not end up reducing the level of NIEM and XML Schema knowledge required to create a data model, using NIEM-UML can still bring significant efficiency gains due to the availability of customized tools. When modeling NIEM content in XML Schemas, the bulk of the work is done in generic XML tools that lack NIEM-specific features. However, when using NIEM-UML, tools designed for NIEM are used from the very beginning, providing a benefit to the developer. Although such a benefit could have theoretically been realized without NIEM-UML, that is, by developing better NIEM tools that work directly in XSD, the current tool landscape could leave NIEM-UML users with a smoother development process.

If pre-existing XML Schemas are to be used as a starting point for a NIEM model, using NIEM-UML is somewhat faster than using XML Schemas directly. As mentioned in the experiences related in this document, some manual repairs on XML Schemas converted to UML are required before they can function in NIEM. However, if NIEM-UML had not been used, an even greater number of repairs would have been required, as significant changes must be made to every XML type and element to comply with the NIEM NDR.

The NIEM-UML standard was found to be fairly comprehensive, and features of NIEM had counterparts in NIEM-XML. This completeness leads to the NIEM-UML standard being large, which has the potential to overwhelm users of the standard, who must learn the many UML stereotypes and the proper way to utilize them.

As stated in the NIEM-UML specification, the primary use case that is supported in NIEM-UML is modeling individual exchanges (IEPDs). For modeling domain (reference) content, the standard does contain the necessary constructs to define reference models. However, the examples and tooling for NIEM-UML only encompass the creating of IEPD and EIEM models. Modeling content of a NIEM domain is possible in NIEM-UML. However, given the state of the available tooling, the UML and/or XML schemas will need to be refactored and repackaged whenever the domain content is to be put into a usable form.

* 1. Discussion on NIEM-UML tools and processes

When using NIEM-UML tools, in general, the state of the tooling is mature enough that developing large models in NIEM-UML is viable. With some exceptions, the tools were capable of performing all NIEM-UML functions. When selecting a tool, the workarounds and bugs described in this document should be carefully considered to ensure that the chosen tool is capable of doing what is required (or to ensure that the tool can be fixed before work commences).

It had been hoped that using NIEM-UML tools would shield users from the complexity of the underlying standards. However, the NIEM-UML tools that are available were built as extensions (or profiles) on top of existing, general-purpose UML tools. Effectively, all of the basic diagramming and UML features of the tools are present, with the NIEM-specific features adding another layer of complexity. The result is that new users could potentially become overwhelmed with the array of options and functions available when using the tools. (Note that a similar issue occurs when modeling NIEM domains outside of NIEM—using XML Schema—as all of the features of XML Schema remain available to the user, even though many are irrelevant or disallowed in NIEM.)

Because NIEM-UML uses many of the same objects used in standard UML class models, it was hoped that the NIEM-UML standard would allow non-NIEM content to directly be copied to NIEM models. For example, the process of importing existing XML Schema content into NIEM-UML involves creating a generic UML class model. However, a significant amount of manual correction was required before any model could be imported into NIEM-UML and then exported as an XML Schema. This correction required significant understanding of how various choices would affect the XML Schema output that would ultimately be written by the tool. For example, the aggregation of every UML property had to be manually reset to prevent unnecessary reference properties from appearing in the XML Schema. In short, NIEM-UML had too many NIEM-specific features for any direct translation to be accomplished perfectly.

One requirement that steepens the learning curve for NIEM developers who use XML Schema is the need to follow the NIEM NDR. The NDR is large and requires a good understanding of XML Schema to be usable. Because NIEM-UML tools automatically generate XML Schemas as output, the need to understand most of the NDR is eliminated. However, cases were identified where the NDR was not always followed because of shortcomings or bugs in the tool. For example, it was possible to create enumerations lacking complex types, complex types using the “xsd:any” construct, or multiple global elements with the same name. In other cases, automatically ensuring NDR compliance would be inherently difficult or impossible, such as when ensuring that concepts are appropriate for a NIEM model or when choosing appropriate class and property names. In the end, users of NIEM-UML must understand at least some of the NDR. Because the current NDR is more tailored for NIEM developers who use XML Schema, NIEM-UML developers cannot escape the need to have a thorough understanding of NIEM and XML Schema before undertaking a NIEM-UML development task.

* 1. Summary

In summary, one desired feature of NIEM-UML is the ability to build a “platform-independent” UML model and convert it into NIEM content, sidestepping the complex rules and structures that are part of NIEM. This concept was partially realized, although expertise in NIEM is still required to avoid building UML models that will be converted into invalid or nonsensical NIEM output. There were overall usability benefits when using NIEM-UML tools to build NIEM content once the circumstances that would result in undesired NIEM output were identified and avoided. However, this must be balanced with the fact that developers must have expertise in both UML and NIEM to use NIEM-UML tools properly.