# Introduction to NIEM Model Instances

## Background: How We Got Here

NIEM began as a framework for building and defining messages to facilitate exchanges of information. Part of that framework was a defined process. Part of the framework was a model that formed a base on which exchanges were defined. That base would be both carved down *and* extended to meet the needs of each exchange. That base existed, and still exists, as a set of XML Schema documents.

As other means of exchanging information have become popular, notably JSON[1](#json_fn), the community needs have expanded to using NIEM with exchange mechanisms other than XML Schema. JSON is the current alternate means, but many more exist.

Dealing with this issue, both now and in the future, is the rationale for the Metamodel.

## Problem/Issue

This introduces a problem. How do you translate the model from XML Schema to some form of JSON, when XML Schema and JSON have similar, but definitely different, feature sets? This isn’t impossible. We do have a specification and other guidance for this translation, but the translation isn’t trivial and the results may not be as satisfying as possible.

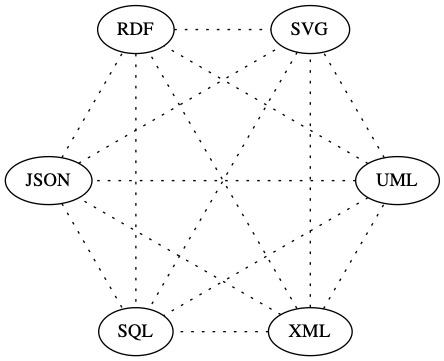
The situation is shown below, the dotted line representing the incomplete translation between these two technologies.

Two Technologies

Two Technologies

Still, this conversion is manageable. When more technologies get added, the translations get out of hand, especially as the community does work in these other technologies. How does work done in JSON get translated to RDF[2](#rdf_fn)? How accurate is that translation? Does that work get translated to XML Schema via JSON directly or via RDF?

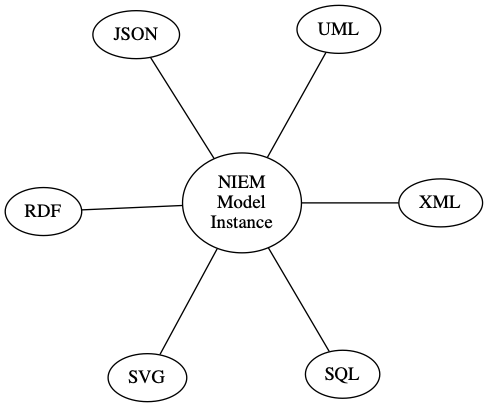
The “N-squared” diagram is familiar to anyone who has seen many presentations about NIEM. Usually it’s representing a variety of entities making an ever growing number peer-to-peer sharing agreements. The same diagram applies here, as a multitude of technologies start requiring peer-to-peer conversions between technologies.



N-Squared Technologies

At the message level, NIEM provides the means to define a single centralized and standardized format for the exchange, replacing the complexity of numerous peer-to-peer agreements. Instead of peer-to-peer agreements, everyone implements towards the standard.

The same concept applies here with models and technologies. Instead of individual translations between technologies, there’s one centralized and standardized “model instance.” Different technologies are translated from that standard model. Now the lines are solid, as each translation can better leverage the abilities of a particular technology.



Standardized and Centralized

Currently, modeling concepts are embedded in a specific technology, XML Schema. We overload XML Schema concepts to include real-world concepts. We use XML Schema to both define NIEM *and* act as the tool for validating actual messages.

## Solution

The solution is to create the modeling concepts in a conceptual format instead of embedding them in XML Schema. Instead of implying modeling concepts in XML Schema, we explicitly define them in a Model Instance.

This allows for **concept**-to-technology conversions, e.g. Model -> XML Schema and Model -> JSON. This is easier and more accurate than technology-to-technology conversions, e.g. XML Schema -> JSON.

## How It Works

“Metamodel” is a framework for building models. The metamodel itself isn’t the NIEM model. It’s a “neutral modeling formalism” for defining models. It provides the means to define a model, as compared to NIEM which is a means for defining real world objects:

* NIEM: Defines real world things
* Metamodel: Defines modeling concepts

## What It Looks Like

Here’s a snippet from NIEM, a subset of nc:PersonEmploymentAssociation and its type nc:EmploymentAssociationType. It defines how a matching XML instances document needs to look, but modeling concepts are implied.

<xs:complexType name="EmploymentAssociationType">  
 <xs:annotation>  
 <xs:documentation>A data type for an association between an employee and  
 an employer.</xs:documentation>  
 </xs:annotation>  
 <xs:complexContent>  
 <xs:extension base="nc:AssociationType">  
 <xs:sequence>  
 <xs:element ref="nc:Employee" minOccurs="0" maxOccurs="unbounded"/>  
 <xs:element ref="nc:Employer" minOccurs="0" maxOccurs="unbounded"/>  
 </xs:sequence>  
 </xs:extension>  
 </xs:complexContent>  
</xs:complexType>  
  
<xs:element name="PersonEmploymentAssociation" type="nc:EmploymentAssociationType" nillable="true">  
 <xs:annotation>  
 <xs:documentation>An association between an employee and  
 an employer.</xs:documentation>  
 </xs:annotation>  
</xs:element>

Here’s the matching snippet from a NIEM Model Instance subset. While it’s longer, that’s because it details the different objects in modeling terms. Note that it isn’t XML *Schema*. It’s not designed as a tool for validating exchanges. It’s plain XML and only defines the Model Instance. To use as a tool for validation, you would convert this to the technology you’ll be using, be it XML Schema, JSON, RDF, UML[3](#uml_fn), or whatever.

<ObjectProperty structures:id="nc.PersonEmploymentAssociation">  
 <Name>PersonEmploymentAssociation</Name>  
 <Namespace structures:ref="nc" xsi:nil="true"/>  
 <DefinitionText>An association between a person and employment  
 information.</DefinitionText>  
 <Class structures:id="nc.PersonEmploymentAssociationType">  
 <Name>PersonEmploymentAssociationType</Name>  
 <Namespace structures:ref="nc" xsi:nil="true"/>  
 <DefinitionText>A data type for an association between a person and an  
 employment.</DefinitionText>  
 <ExtensionOf>  
 <Class structures:ref="nc.AssociationType" xsi:nil="true"/>  
 <HasObjectProperty mm:sequenceID="1" mm:minOccursQuantity="1"  
 mm:maxOccursQuantity="1">  
 <ObjectProperty structures:id="nc.Employer">  
 <Name>Employer</Name>  
 <Namespace structures:ref="nc" xsi:nil="true"/>  
 <DefinitionText>A party/entity (organization or person) who  
 employs a person.</DefinitionText>  
 <Class structures:ref="nc.EntityType" xsi:nil="true"/>  
 </ObjectProperty>  
 </HasObjectProperty>  
 <HasObjectProperty mm:sequenceID="2" mm:minOccursQuantity="1"  
 mm:maxOccursQuantity="1">  
 <ObjectProperty structures:id="nc.Employee">  
 <Name>Employee</Name>  
 <Namespace structures:ref="nc" xsi:nil="true"/>  
 <DefinitionText>A person who works for a business or a  
 person.</DefinitionText>  
 <Class structures:ref="nc.PersonType" xsi:nil="true"/>  
 </ObjectProperty>  
 </HasObjectProperty>  
 </ExtensionOf>  
 <ContentStyleCode>HasObjectProperty</ContentStyleCode>  
 </Class>  
</ObjectProperty>

## Terminology

There are several levels to these concepts and past terminology hasn’t clearly delineated them.

### Metamodel

The framework itself is the “Metamodel.” It’s an abstraction of what makes up a model. It’s not NIEM-specific. We use it to define NIEM, but you could use it to define any number of other models. There is no “NIEM Metamodel.”

The Metamodel is a crucial tool for creating and maintaining models, but is not something an ordinary NIEM user would usually care about.

### Model Instance

Generically speaking, when you create a model from the Metamodel, you get a “model instance.” This is a conceptual model reflecting the objects and relationships in a subject area. This does *not* have to be NIEM. The Metamodel could be used to create a wide variety of different Model Instances. Of course, our main interest is in NIEM so we want to create a Model Instance reflecting NIEM.

### NIEM Model Instance

When you create a specific model, that name for that model instance gets a prefix determined by what specific model you’ve created. If you create NIEM as a model, that’s a “NIEM Model Instance (NMI).” This is still a conceptual model. To use it for validating real-world exchanges, it would need to be instantiated into some format.

Up until this point, the Metamodel and Model Instances are mainly behind-the-scenes tools. What community members eventually need are versions of NIEM they can use to define exchanges and to which they can implement. The NIEM Model Instance is what a user would use to define an exchange, in terms of creating both subsets and new content. It’s what would underlie the tooling. It’s the platform *independent* version of NIEM.

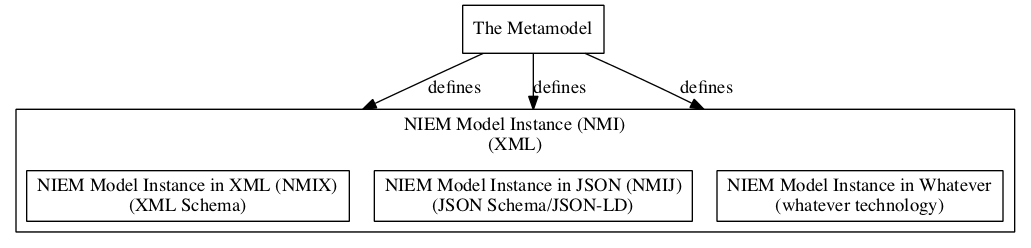
To actually implement an exchange, platform *dependent* versions are needed.

### NIEM Model Instance XML/JSON

Transforming the NIEM Model Instance to a representation in a particular technology adds the technology as a suffix. If you’ve converted the NIEM Model Instance to XML Schema, then you have a NIEM Model Instance in XML (NMIX). If you’ve converted it to JSON Schema, it’s a “NIEM Model Instance in JSON (NMIJ).”

Note that a NIEM Model Instance in XML (NMIX) is what we currently call “NIEM.” The NIEM Model Instance abstracts NIEM up a level, in order to separate the modeling concepts from the specific technology of XML Schema.

XML and JSON aren’t the only targets for this conversion/rendering, but are the starting point for the effort.



Terminology

While creating platform dependent versions of NIEM for validation purposes is a major outcome of the Metamodel and NIEM Model Instance, some instances may have entirely different purposes, often as a means of viewing a model.

## Benefits

The major benefit is enabling the use of multiple model instance formats and views from one “source.” The NIEM Model Instance could be transformed into any of these example formats:

* XML Schema
* JSON/JSON-LD[4](#json-ld_fn)
* SQL[5](#sql_fn)
* UML (via XMI[6](#xmi_fn))
* RDF/OWL[7](#owl_fn)
* OpenAPI[8](#openapi_fn)
* Protobuf[9](#protobuf_fn)
* Human readable documentation
  + Text (HTML[10](#html_fn), Markdown[11](#markdown_fn), DOCX[12](#docx_fn), RTF[13](#rtf_fn), PDF[14](#pdf_fn), CSV[15](#csv_fn), etc.)
  + Diagrams (Graphviz/DOT[16](#dot_fn))

Using a NIEM Model Instance, you no longer need separate tool suites for each format, e.g. SSGT and Movement. Instead, you can have one tool suite that deals with models, and converters for different technologies. Converters are easier to write than tool suites. Development of those converters can be spread out over the community, leveraging expertise in each technology.

## Why Not Just Use RFD/RDFS[17](#rdfs_fn)?

NIEM models have details that aren’t easily captured in RDF. Concepts like cardinality and field typing are crucial to information exchanges yet are not easily represented in RDF.

A key benefit of the Metamodel is that the NIEM Model Instance can be readily converted to RDF.

## Why Not Just Use UML?

Prior efforts at defining NIEM in UML were complex and costly. While free and open source tools for using UML exist, those that explicitly supported NIEM were expensive and proprietary.

Additionally, UML tools use XMI as a format for exchanging diagrams, but implementation of XMI across tools and versions isn’t as stable and reliable as needed.

A key benefit of the Metamodel is that the NIEM Model Instance can be readily converted to UML/XMI.

* 1. [JavaScript Object Notation (JSON)](https://en.wikipedia.org/wiki/JSON)
* 2. [Resource Description Framework (RDF)](https://en.wikipedia.org/wiki/Resource_Description_Framework)
* 3. [Unified Modeling Language (UML)](https://en.wikipedia.org/wiki/Unified_Modeling_Language)
* 4. [JavaScript Object Notation for Linked Data (JSON-LD)](https://en.wikipedia.org/wiki/JSON-LD)
* 5. [Structured Query Language (SQL)](https://en.wikipedia.org/wiki/SQL)
* 6. [XML Metadata Interchange (XMI)](https://en.wikipedia.org/wiki/XML_Metadata_Interchange)
* 7. [Web Ontology Language (OWL)](https://en.wikipedia.org/wiki/Web_Ontology_Language)
* 8. [OpenAPI](https://en.wikipedia.org/wiki/OpenAPI_Specification)
* 9. [Protocol Buffer](https://en.wikipedia.org/wiki/Protocol_Buffers)
* 10. [HyperText Markup Language (HTML)](https://en.wikipedia.org/wiki/HTML)
* 11. [Markdown](https://en.wikipedia.org/wiki/Markdown)
* 12. [Office Open XML (DOCX)](https://en.wikipedia.org/wiki/Office_Open_XML)
* 13. [Rich Text Format (RTF)](https://en.wikipedia.org/wiki/Rich_Text_Format)
* 14. [Portable Document Format (PDF)](https://en.wikipedia.org/wiki/PDF)
* 15. [Comma-Separated Values (CSV)](https://en.wikipedia.org/wiki/Comma-separated_values)
* 16. [DOT (graph description language)](https://en.wikipedia.org/wiki/DOT_(graph_description_language))
* 17. [RDF Schema (Resource Description Framework Schema)](https://en.wikipedia.org/wiki/RDF_Schema)