# Introduction to NIEM Model Instances

## Consider a 1-2 paragraph “abstract” with highpoints focused on strengths/benefits/positives and key definitions (this may be repurposed and used in other materials/briefs

## Consider how this will be used: do you want readers to get the “good stuff” up front? then consider leading w/ a very brief background explanation of issue then roll into solution – then go you can return Communications scholarly research shows that if we’re attempting to introduce something new and great, it’s better to lead with the “goodness” – if you lead with what wasn’t great, that’s what sticks in readers’ minds and/or they never drift down to the important stuff. It is likely important to introduce key parts of the terms/definitions early on – my impression is that these are important – we could possibly box them in or highlight them in some way early, then come back to the complete terminology later on

## Background: How We Got Here

NIEM began as a framework for building and defining messages to facilitate exchanges of information. One part of the framework was a defined process for these message. Another prt of the framework was a model that formed the basis on which exchanges were defined. The base would be both carved down and extended to meet the needs of each exchange. This base exists as a set of XML Schema documents.

Other means of exchanging information are increasingly being used. Demand for NIEM with exchange mechanisms have expanded beyond the XML Schema. Most notably, this is JSON[1](#json_fn), which 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? We have a specification and other guidance for this translation, but the translation isn’t trivial and the results may not be satisfying

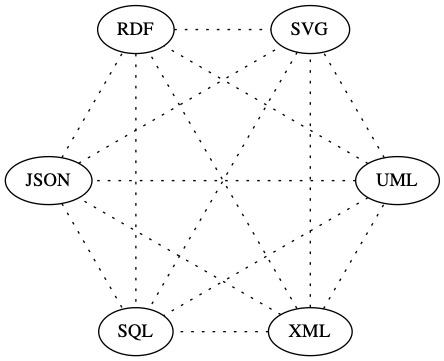
The situation is shown below, the dotted line representing the incomplete translation between these two technologies.

Two Technologies

Two Technologies

Still, the XML Schema to JSON\_LD conversion is manageable. As more technologies are added, the translations become unwieldy. Looking forward, this becomes more problematic as the community does work in 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?

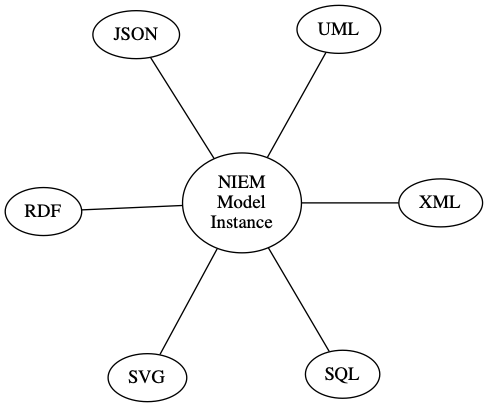
The “N-squared” diagram is familiar to anyone who has seen many presentations about NIEM. Usually it’s representing a variety of entities making an ever growing number of peer-to-peer sharing agreements. The same diagram applies here, as a multitude of technologies start requiring peer-to-peer conversions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

This same concept applies with models and technologies. There is one centralized and standardized “model instance,” rather than individual translations between technologies. Different technologies are translated from thw standard model. The lines are solid, as each translation is better able to leverage the abilities of a particular technology.



**Standardized and Centralized**

NIEM modeling concepts are currently embedded in 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 (going forward)

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

**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 conversionsm (e.g. XML Schema -> JSON**, etc,)

## How It Works

**“Metamodel” is a framework for building models**. iThe metamodel,

The metamodel itself isn’t the NIEM model.

## 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 convert this NIEM Model Instance subset to the technology you’ll be using. That can be represented in XML Schema, JSON, RDF, UML[3](#uml_fn), or whatever meets your requirements.

<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.”

If you create NIEM as a model, that’s a “NIEM Model Instance (NMI).” This is still a conceptual model

The Metamodel is a crucial tool for creating and maintaining models -- it is not typically not something an ordinary NIEM user would be concerned with.

### 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, the name for that model instance is given a prefix determined by the specific model you’ve created. When 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 is instantiated into some format.

Up until this point, the Metamodel and Model Instances are mainly behind-the-scenes tools. **Community members eventually need platform agnostic versions of NIEM that can be used to define exchanges, that in turn, can be implemented in any given technology . The NIEM Model Instance is used to define an exchange, in terms of creating subsets and new content. Itunderlies the tooling. It is the platform independent, or agnostic, version of NIEM.**

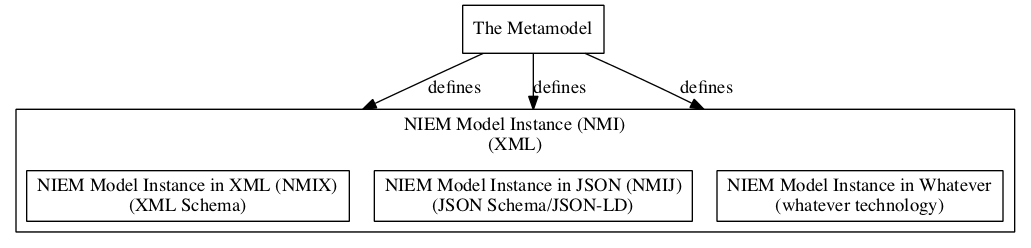
To implement an exchange, platform dependent versions are needed.

### NIEM Model Instance XML/JSON

Transforming the NIEM Model Instance to a representation in a particular technology adds the technology as a suffix. If a NIEM Model Instance is converted to XML Schema, it becomes a “NIEM Model Instance in XML (NMIX)”. If converted to JSON Schema, I is a “NIEM Model Instance in JSON (NMIJ).”

A NIEM Model Instance in XML (NMIX) is currently termed “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 are the starting points for this conversion with other targets (is this targets other technology? )following. (are you saying that the first Model Instance – should we lead with that to clarify?



**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 NIEM model instance formats and views from one “source.” The NIEM Model Instance can 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. There is one tool suite that deals with models with 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 RDF/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? (consider a positive statement rather than “why not”

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.

**Getting Started with the Metamodel**

Visit <https://github.com/webb/niem-metamodel/blob/master/docs/index.md> to view and learn more about the Metamodel.

* 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)