IBM

developerWorks®

Creating a NIEM IEPD, Part 1: Model your NIEM exchange

Design an XML information exchange between US government entities

Priscilla Walmsley

May 17, 2010
(First published January 19, 2010)

The National Information Exchange Model (NIEM) is rapidly becoming the most important XML exchange standard for the U.S. government and its information partners. This article, the first in a four-part series, provides an overview of the process for defining a NIEM information exchange. It then takes you through the first step—modeling your exchange using UML—with special considerations for NIEM modeling concepts. A simple case study is used to illustrate the process.

View more content in this series

18 May 2010: Added links to Part 4 of this series in Introduction, Conclusion, and Resources sections.

09 Mar 2010: Added links to Part 3 of this series in Introduction, Conclusion, and Resources sections.

09 Feb 2010: Added links to Part 2 of this series in Introduction, Conclusion, and Resources sections.

Other articles in this series

- Part 2: Map and subset NIEM
- Part 3: Extend NIEM
- Part 4: Assemble the IEPD

The National Information Exchange Model (NIEM) is a U.S. government-sponsored initiative to facilitate information sharing among public and private sector organizations. Its initial focus was on law enforcement, public safety, and emergency management, but it is continuously being expanded into other domains. New XML initiatives within the U.S. Department of Justice and Department of Homeland Security, along with other sectors of the U.S. government, use NIEM as a common base data model and methodology to promote interoperability of data and software, reduce design and development time for information exchange applications, and allow the reuse of intellectual capital and skills across multiple projects.

NIEM is described as a *framework,* because it is not just an XML vocabulary for information exchange. It has several components:

- A common XML-based data model called *NIEM core* that provides data components for describing universal objects such as people, locations, activities, and organizations
- More specialized XML data models for individual use cases, called *domains* (examples include Justice, Immigration, and Emergency Management)
- A methodology for using and extending the building blocks that come from the common and domain-specific models to turn them into a complete information exchange, known as an *information exchange package*
- Tools to help develop, validate, document, and share the information exchange packages
- A governance organization that provides training and support and oversees NIEM's evolution over time

This article, the first of a four-part series, introduces you to NIEM and shows you how to model a NIEM Information Exchange Package Documentation (IEPD) using the Unified Modeling Language (UML). The working and final UML models are available from Downloads.

How do you use NIEM?

The NIEM XML data model provides building blocks for common objects. A building block may be at a very granular level, such as "person name" or "birth date," or a much more complex component, such as "arrest" or "court case." However, the NIEM model itself doesn't define complete information exchange messages such as "Arrest Report" or "Suspicious Activity Report." It does not designate any specific message types or root elements of XML documents.

To actually use NIEM, you need to build an IEPD. The IEPD pulls the necessary components from the NIEM core and domain models and extends them to create an information exchange. An IEPD contains several artifacts:

- XML schemas that define the subset of the NIEM model used in this exchange, known as the *subset schema*
- A schema that defines the root element of the exchange, known as the *exchange schema*
- A schema that defines extensions to the NIEM model, known as the *extension schema*
- Documentation of the exchange, such as UML diagrams, narrative descriptions, and samples

Developing an IEPD

The first task in any information exchange project is to gather and analyze your requirements. NIEM does not require any particular method of defining requirements, so this article doesn't describe this process. In fact, this article assumes that before you sit down to actually create your IEPD, you have an idea of the data elements you want to exchange and the type of messages you want to structure them into.

The articles in this series will work through a simple example from start to finish, with the result being a complete IEPD. The example case study will be to implement a simple theft report that covers registered vehicles. Hypothetically, local law enforcement would use the theft report to inform interested parties, such as the Division of Motor Vehicles or the City Bicycle Registration Bureau, of thefts of motor vehicles and bicycles. During my requirements gathering phase, I have gathered general information on the data that needs to be shared and determined that only one type of message is required: the theft report. In reality, an IEPD often consists of multiple related message types.

Because a main goal of NIEM is data interoperability, it makes sense to consider reusing an existing IEPD before creating a new one from scratch. NIEM provides an IEPD clearinghouse that allows you to search for existing IEPDs submitted by other organizations. See Resources for a link to the IEPD clearinghouse.

If you can't find an existing IEPD that suits your needs, you will need to build one. Developing a new NIEM IEPD requires five steps:

1. Model your exchange.
2. Map your exchange to the NIEM data model.
3. Create a subset of the NIEM model for use in your exchange.
4. Create exchange and extension schemas to describe your custom components.
5. Assemble an IEPD with all of the appropriate artifacts.

This article describes step 1; the rest of the articles in this series will cover steps 2 through 5. Even if you choose to reuse an existing IEPD, this article series might still be a useful guide to help you to understand the content and structure of the IEPDs you are using.

Understanding the NIEM model

Before you create a model for your exchange, it is useful to understand how the NIEM data model is structured. NIEM defines concepts—such as types, properties, and associations—that are probably familiar to you from other data modeling paradigms.

NIEM model concepts

Types represent things, both tangible and intangible. Some of the most fundamental types in the NIEM model are `PersonType`, `ActivityType`, `ItemType`, `LocationType`, and `OrganizationType`. There are also thousands more, with varying degrees of granularity. Types might be known as *classes* or *entities* in other modeling paradigms.

Properties are attributes of types. They can themselves have complex types. For example, `PersonName` is a property of `PersonType`, but it is also has a type `PersonNameType` that has its own structure containing `PersonGivenName`, `PersonSurName`, and so on.

Types can be derived from other types and inherit their properties, which is analogous to generalizations in an object-oriented model. For example, `ItemType` is a generic type that has many types derived from it, including `VehicleType`, `JewelryType`, and `RealEstateType`.

Associations are relationships between two types. You might have an association between an `Incident` and a `Person` or a `Person` and a `Location`. Associations in NIEM are separate from the types they relate.

Roles represent temporary affiliations that a type might have in a particular context. For example, in a theft incident, a person might play the role of `Victim`, `Subject`, or `Witness`.

Augmentations are bundles of properties that you can reuse and share. These are more commonly used in the NIEM domain models than in your IEPDs.

Metadata is information about the content of a message, such as when the information was gathered and how reliable it is. NIEM makes special provisions in its model for relating data to metadata.

The NIEM model in XML

The NIEM model is implemented entirely as a set of W3C XML Schema documents. Annotations and references within the XML schemas are used to indicate whether something is a type, an association, and so on. Fortunately, you do not have to read the XML schema documents themselves to navigate the model; NIEM provides tools to search and navigate the model in a more graphical fashion.

In general, NIEM types are implemented as XML Schema complex types, and properties are elements contained within those types. Listing 1 shows how an activity is represented by a `ActivityType` complex type, with properties such as `ActivityIdentification` and `ActivityCategoryText` implemented as child elements.

Listing 1. Partial NIEM ActivityType implementation in XML Schema

```
<xsd:complexType name="ActivityType">
 <xsd:complexContent>
  <xsd:extension base="s:ComplexObjectType">
   <xsd:sequence>
    <xsd:element ref="nc:ActivityIdentification" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:ActivityCategoryText" minOccurs="0" maxOccurs="unbounded"/>
    <!-- ... -->
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

NIEM uses XML Schema extensions for type derivation. Listing 2 shows how a more specific kind of activity—the `AssessmentType` complex type—is derived from `ActivityType`.

Listing 2. NIEM type derivation in XML Schema

```
<xsd:complexType name="AssessmentType">
 <xsd:complexContent>
  <xsd:extension base="nc:ActivityType">
   <xsd:sequence>
    <xsd:element ref="nc:AssessmentScoreText" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:AssessmentFee" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:AssessmentProgram" minOccurs="0" maxOccurs="unbounded"/>
    <!-- ... -->
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Associations are special kinds of complex types that contain references to the types they associate. Listing 3 shows how an association between a person and an activity —`ActivityPersonAssociationType`—is implemented. All association types are extensions (directly or indirectly) of the NIEM core `AssociationType`.

Listing 3. NIEM association type in XML Schema

```
<xsd:complexType name="ActivityPersonAssociationType">
 <xsd:complexContent>
  <xsd:extension base="nc:AssociationType">
   <xsd:sequence>
    <xsd:element ref="nc:ActivityReference" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:PersonReference" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Modeling your exchange

NIEM does not require that you use any particular methodology or diagram types to model your XML exchange or even that you model it at all. However, modeling is an important step in IEPD design. The modeling process fleshes out the requirements, and the final result provides documentation that is helpful for both business and technical users. The model also serves as useful input into the subsequent steps in the IEPD creation process.

Choosing a modeling paradigm

Using UML tools

If you use UML, there is an advantage to using an XMI-enabled UML tool, such as IBM® Rational® Modeler or ArgoUML, because you can use the XMI to automatically generate a mapping spreadsheet, which you can use in the next step. For this article, I used ArgoUML, an open source UML editor.

A good option is UML—in particular, UML class diagrams—because UML concepts map easily onto NIEM model concepts. Of course, you can create other UML diagrams, such as use case diagrams and sequence diagrams, to document your exchange. This article focuses on the class diagram, because that is most crucial to the IEPD development process.

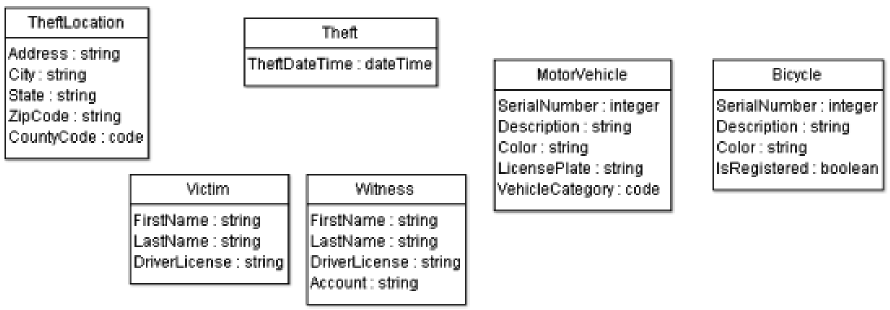
It is best to create a first draft of your model independently without trying to fit it into the NIEM model. You want to get the model right for your business needs without being unduly influenced by the NIEM way of doing things. Later in the IEPD process, it not uncommon to make small alterations to the model to better harmonize it with NIEM (in cases where it makes sense). However, there will always be differences between your model and the NIEM model.

Types and properties

This article isn't long enough to provide a complete introduction to UML modeling, so it focuses on the NIEM-specific pointers. As you might expect, NIEM types are represented by classes in a class diagram. Properties are represented by attributes of the class.

In my example case study, I determine that I have several classes that need to be exchanged—for example `Theft`, `MotorVehicle`, `Bicycle`, `Victim`, `Witness`, and `TheftLocation`. These types are depicted in Figure 1 along with their properties. (View a text-only version of Figure 1.)

Figure 1. Initial UML model with types and their properties



When specifying the data types of the properties, it is useful to use XML Schema primitive data types, because the properties will eventually be represented in an XML schema and it will be easier to determine whether the existing NIEM model fits yours if you use a common set of data types. The most commonly used XML Schema data types are listed in Table 1.

Table 1. Common XML Schema data types

Data type name	Description	Examples
`string`	Any text string	abc, this is a string
`integer`	An integer of any size	1, 2
`decimal`	A decimal number	1.2, 5.0
`date`	A date in YYYY-MM-DD format	2009-12-25
`time`	A time in HH:MM:SS format (24-hour clock)	12:05:04
`boolean`	A true/false value	true, false

Some properties have an enumerated list of valid values, also known as a *code list.* Code list values can be described in the UML model in comments or documented elsewhere in the system documentation. In my example, to keep the model clean, I simply list these properties as having a data type `code`. I will record the valid values in the mapping spreadsheet created in the next step of the IEPD process.

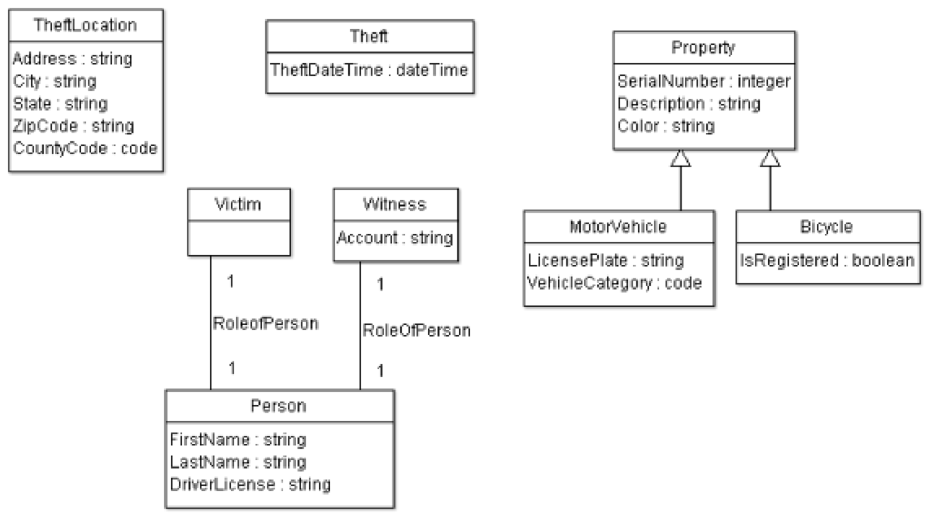
Generalizations and roles

The NIEM model uses generalizations, and when appropriate, you should use them in your model, too. In the case study, `MotorVehicle` and `Bicycle` are both specific kinds of property that might be stolen. So, I decide to add a more generic `Property` class and derive `MotorVehicle` and `Bicycle` from that. Doing so allows me to define the common properties such as `SerialNumber` only once and will also simplify associations by allowing the `Property` class to be the associated with the `Theft` class.

`Victim` and `Witness` appear to follow the same rule. After all, they are both just more specific kinds of people. However, a person's state of being a witness or a victim is temporary, so it is better represented as a role. In fact, in this case, the same person could be both a victim and a witness in a particular theft. In that case, you would only want to represent one person with two different roles. I show that in my model by adding a separate `Person` class and creating associations to the `Victim` and `Witness` classes. I label the associations *Role Of Person* to indicate that they are related through a role rather than a normal association.

Figure 2 shows the model after I have added my generalizations and roles. (View a text-only version of Figure 2.)

Figure 2. UML model with generalizations and roles added



Relationships

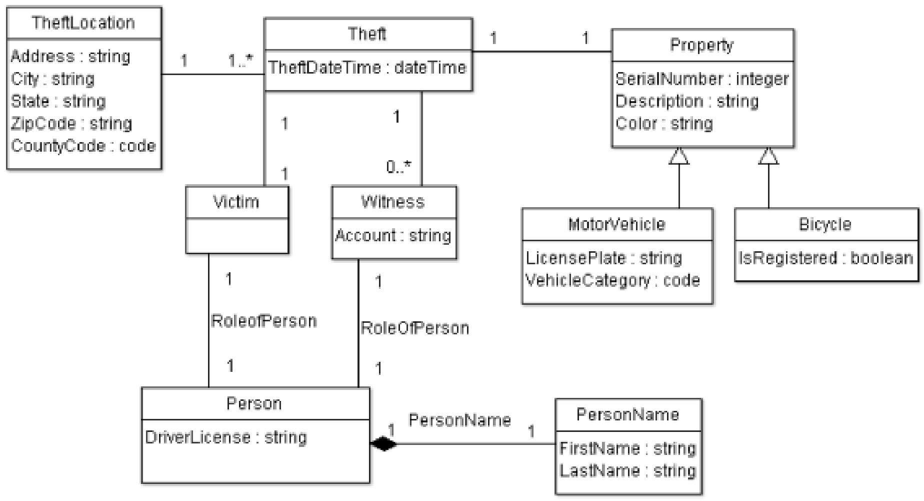
UML has three ways of representing relationships: aggregation, composition, and association.

Aggregation and composition relationships generally represent "has a" relationships, where one class is subordinate to another. In the example case study, a `Person` "has a" `PersonName`. The `PersonName` class is not useful without a person to relate it to. Aggregations and compositions are treated the same way in NIEM. In the eventual XML structure, the subordinate class will be contained in the other class. For example, there will be a `Person` element that contains a `PersonName` element.

In contrast, associations are between two classes that can stand on their own. In the example case study, a `Theft` and a `TheftLocation` are two separate things; one can exist without the other. To represent these in your model, you can use generic UML associations, or, if there are additional properties relating to the association itself, add separate association classes to the model. Either way, in the NIEM XML structure, the classes will each be represented as distinct elements with a separate association element that contains references to the elements that it is relating—in this case, `Theft` and a `TheftLocation`.

In the example case study, I use composition to represent the `Person`-`PersonName` relationship and simple UML associations to relate the rest of the classes to each other. Figure 3 shows the model after I have added relationships. (View a text-only version of Figure 3.)

Figure 3. UML model with relationships added

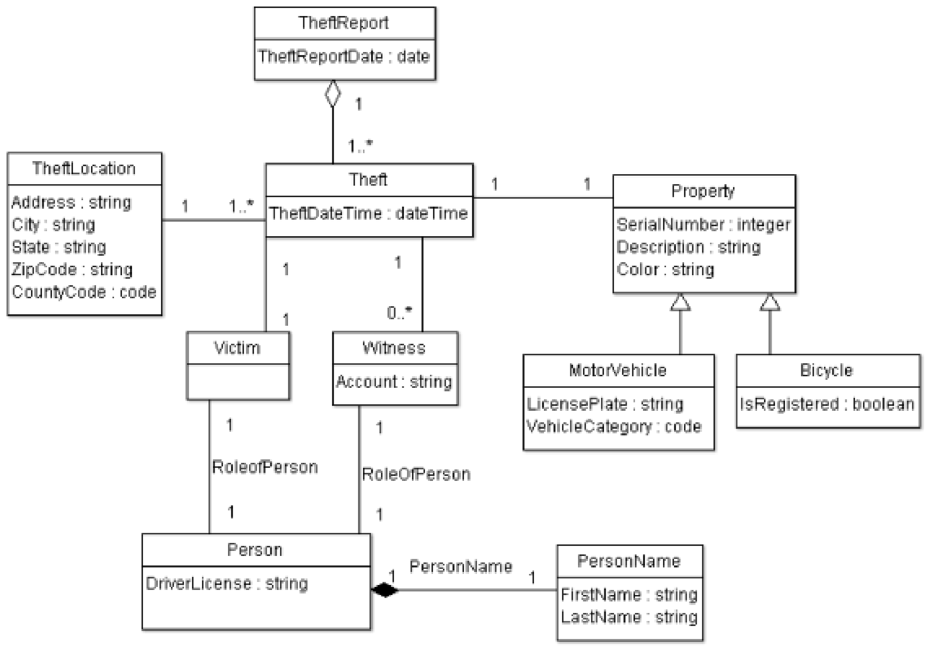


Choosing a root

Every XML message must have a single root. Generally, in an XML exchange, there is a single focal point or purpose for the message. In my case, it is the theft report itself. I add a class for `TheftReport` to my model along with a property `TheftReportDate`. I create an aggregation relationship between `TheftReport` and `Theft`, indicating that the theft report consists of a set of thefts.

The complete UML model is shown in Figure 4. This model is not yet perfect, nor does it have to be. It is common to make iterative changes to the model throughout the IEPD development process. For example, it might be useful to modify the structure or names to better fit the NIEM model, where appropriate. (View a text-only version of Figure 4.)

Figure 4. Completed UML model



Conclusion and next steps

Other articles in this series

- Part 2: Map and subset NIEM
- Part 3: Extend NIEM
- Part 4: Assemble the IEPD

This article described at a high level the steps involved in creating a NIEM IEPD and delved into detail on the first step: creating the model. The result is a working draft of a UML model that you can use to continue IEPD development. Using NIEM-targeted concepts like roles and XML Schema data types during the modeling process makes the rest of the IEPD development process easier.

The next article in this series will describe the second and third steps: mapping and subsetting. You will learn how to create a component mapping template that maps a UML model to NIEM and generate a subset of the NIEM model to match your mapping.

Downloadable resources

Description	Name	Size
Final ArgoUML model from this article	niem1.zip	12KB
Final XMI model from this article	niem1.xmi.zip	4KB