

# Modeling Properties and Associations in NIEM-UML Transcript

## NIEM Program Management Office Resource

Welcome to the NIEM-UML tutorial on modeling properties and associations. We assume you have a little background for this tutorial, that you know the basics of NIEM and UML. Perhaps you've seen the NIEM-UML high level introduction. In this tutorial, you'll learn about modeling properties and associations in NIEM-UML and how these relate to NIEM concepts. If you need more background, please see the other tutorials on the NIEM.gov website.

Properties represent information about an entity and are modeled as properties of UML classes or as UML association ends. We'll see a bit more about association ends a little later. Information from each property includes its name, the type of information in that property, and its cardinality, or how many of them you can have. In the person example, we see we have three properties: person birthdate, which is a date, person name, which also has the type person name, and person social security identification, which has the type identification. Notice the bracket in the back of each property. This indicates the cardinality. The cardinality for our three example properties is one, meaning that each property must have one value for each person. Other possibilities for cardinality are zero to one, for optional properties, zero to star, or just a star, for properties that can have any number of values, or one, or one to one, which means there must be exactly one value for each property, or one to star, which means that the property is not optional but could have multiple values.

This UML class is mapped into NIEM XML. In NIEM XML each property is represented by two elements—a top level element and an element nested inside of the enclosing class, in this case person type. Both representations are required by the NIEM Naming and Design Rules (NDR) and this supports the reuse of properties across multiple classes.

Let's look at an example. We can create a class, Automobile, and a property, Number of Seats, that must be an integer, with at most one optional value. We'll then take a quick look at the XML. We're now using the UML tool and we'll create a new type and call it Automobile. And we'll load the specification window to add an attribute. Note that the way your UML tool may be slightly different,

and the NIEM PMO doesn't endorse any particular tooling. We're going to name this Number of Seats, and select the Integer type, being careful to select the integer from XML primitive types that is used in NIEM. We're also going to select the Multiplicity, how many we can have, to zero or one. And that's all we have to do to create a new property. To look at this in XML, we're going to add automobile to our exchange types, and based on the NIEM-UML standard, produce the MPD. Now let's look at a little bit of the XML that was produced. We'll bring up our extension schema. First we'll note the element Number of Seats as a global element that's defined according to the NIEM NDR. We also note that the element is used within the automobile type to define the structure of automobile based on this property. And that's all there is to it.

Now let's consider associations. Associations connect objects and represent some logical connection between those objects. In UML, associations are defined using a line between classes. Here we see an association between service provider and providing organization. Notice that each association also has ends, in this provider and organization. This is where the name and the cardinality of the properties are defined. Note that the properties are defined on the side opposite of the class that has that property. So in this case, service provider has the property organization and providing organization has the property provider.

There are multiple kinds of associations in NIEM-UML. We'll summarize them here and then go into more detail. There are Simple Associations that are equivalent to properties. There is Reference Associations for connecting independent objects. There's Association Classes that provide more capabilities and allow associations to have properties of their own and be created and deleted independently. There's Roles, a special concept within NIEM. And each association has ends, and for each end, we want to know its name, its multiplicity or cardinality, its aggregation, whether the data is nested inside which is indicated by a black diamond, and whether its one way or two way, one way associations having an arrow to indicate the directionality. And then we may have Association Classes.

Simple Associations are equivalent to properties. Here we see person type, which has the property PersonAgeMeasure, and also has the property PersonName. Since PersonName is an aggregation using the black diamond notation and it has an arrow indicating its one direction, it's essentially equivalent to a property. We could've shown the same thing as a property person type or as an association. As with any other type, these properties have a cardinality. In this case, a person can have any number of names.

Then there's Reference Associations. Reference Associations connect independent objects. The data is not nested inside either of the objects, only a reference to the other object. In this case, Service Provider has a reference to an organization and Providing Organization has a reference to provider. Note the cardinalities, which are shown as a star, indicates any number of values for either of these properties.

Let's do another example. We'll create a Reference Association between a Service Provider and a Providing Organization. We'll assume the Service Provider and Providing Organization classes have

already been defined. We'll simply select Association, define it between the two classes, and start naming our ends. We'll then define the cardinalities and we're done.

We'll now consider Association Classes. Association Classes add an additional capability to the association so that the association can have properties in and of itself. It also becomes an independent element as a class independent of the things it connects. An Association Class is shown as a dotted line connecting the association with the class that represents the properties for that association. A different notation for association classes uses the association type stereotype on a regular class. UML associations have some particular constraints. For example, a UML association must associate at least two other classes. NIEM associations don't have this restriction. So for this reason, the association type notation is used, need to association, and shown as a one way association pointing away from that association type. Note that this association type results in the exact same XML schema elements as does the association class we saw previously.

Let's do an example. We'll create an Association Class between an incident and a victim. We'll then do the same thing using an Association type. As before, we'll assume our classes, incident and victim, have been created already. We'll select association class and define it between incident and victim. Give it a name. This creates the association. To make the example more interesting, let's also give this Association Class a property. And that's all there is to it.

Now let's do the same thing with Association Type. Note that these two representations of association, Association Type and Association Class, produce the same XML schema.

NIEM has a concept of Roles. Roles allow us to represent what an entity does or participates in independently of what that entity is natively. In this example, we see that victim is the role of person and subject is the role of person. That is, a person can be a subject or a victim, can be a subject and a victim at the same time, and a person can be a victim multiple times, or a subject multiple times. This is indicated by the cardinalities that are shown on the RoleOf association. RoleOf associations are shown in NIEM-UML by marking the association with the RoleOf stereotype. The other common pattern for roles is where there can be at most one instance of each role for each entity. For example, a person can be an FBI agent, but only one FBI agent at a time. The person can't be multiple FBI agents at the same time. This more restricted concept of role can be represented by UML generalization, or subclass, marked by a RolePlayedBy stereotype. The cardinality in this type of role is fixed. That is, an FBI agent must be the role of exactly one person, and a person can be an FBI agent zero or one time.

Let's do an example of a role. We'll create the role Patient, which is the role of a person. In our first scenario, we're going to assume that we want to create a patient record that will represent a person no matter how many times they've been in our hospital. We're going to reuse person type from NIEM Core and we'll assume that we've previously subset person type for what we need. There's a separate video on how to subset. We're going to select the RolePlayedBy specialization of Generalization, which says that a Patient is a role played by a person. That's all that need be done.

In our second scenario, we're going to assume that we want to create a patient record every time someone has been in our hospital, so a person could be a patient multiple times. We're still going to reuse person type, but this time we're going to create a RoleOf association. So this says that a patient can be a RoleOf a person zero or one times. Now we're going to change that to require that the patient be the RoleOf a person, and by default, it already is stating that a person can be a patient any number of times. Now this shows the two representations of roles and the different semantics each of them implies.

NIEM and XML schema have a notion of properties that can be defined independently of any class and can be reused across classes. These are called top level properties. UML has no native notion of such properties. All UML properties are defined within a class. In order to handle this, we have two special concepts within NIEM-UML, references and property holders. Property holders, as we see on the bottom for OrganizationPropertyHolder, is a way to define these top level properties. While the property holder shows up in UML it does not map to any class in the XML schema. It essentially disappears. It's just there to hold the properties. The other thing we want to represent is that in a class we can reuse a property defined in some other class. So in this example, OrganizationItemAssociation, in payload is being reused from the definition of that same property in OrganizationPropertyHolder. NIEM Core has a property Medical Condition; it's a property of person. Let's reuse that property in the role patient. Here we see the patient role of person that we already created. What we'd like to do is reuse PersonMedicalCondition in Patient. We've found a very easy way to do this is to copy and paste the property from the PersonType into Patient. What we then want to do is define a Reference between these two properties. What this does is tell the tool that these are the same property but used in two different contexts.

NIEM also defines concept of substitution groups. This allows for hierarchies of properties and properties that can be substituted for other properties. A substitution group is represented in NIEM-UML using property subsetting. Subsetting is a built in UML capability, one that is less familiar to many UML modelers. We can see subsets being used in the substitution group at the bottom of this example. The way subsetting works is you define a property, in this case ContactMeans, that can be substituted for any number of other properties, in this case ContactTelephoneNumber, ContactWebsite, and ContactEmailID. Then, whenever ContactMeans is used, you can substitute any of these other properties for ContactMeans. So ContactMeans sits in a class above, which references our substitution group, which will allow any of these other variants for contact.

Let's do an example. We'll define a property zip code which is allowed to be a number or a string. As you may know, in some countries zip codes are allowed to contain alpha-numeric characters. Of course NIEM core already includes zip code, but we'll create another one just as an example. We'll create a new property holder for zip code. And let's open its specification. We'll now create a new property for zip code. We're now going to create zip code as a string giving it the type string from the XML primitive types. And here we're going to subset zip code. Notice that zip code string subsets zip code. Now let's add zip code as a number, give it the type integer, and also set its subsets. This now defines a substitution group for zip code. If you wanted to then use zip code in

an object type, we could copy and paste zip code. We could then add a references realization from the object type to our property holder. This will establish the relationship between zip code and the object type and zip code and the property holder.

That concludes the NIEM-UML tutorial on modeling properties and associations. If you'd like more information, please see the NIEM.gov website. Thank you for your interest in NIEM-UML.