Model Driven Solutions

Where Business Meets Technology

NIEM-UML

Modeling Guide

A guide to modeling for NIEM-UML with the Nomagic Cameo NIEM-UML plugin.

Table of Contents

1 Introduction

1.1 Intent of this document

The modeling guide for Cameo NIEM-UML provides modelers, information architects and developers with the information they need to develop, maintain and leverage UML for information sharing based on the National Exchange Model (NIEM). Cameo NIEM-UML is based on the NIEM-UML standard (in final stages of adoption) of the Object Management Group (OMG).

This guide is intended to support modeling for NIEM, it is not a tutorial on either NIEM or UML. Background information can be found in the following resources:

- NIEM: https://www.niem.gov/about/tech/Pages/technical-overview.aspx
- NIEM-UML: http://www.niem-uml.org
- UML Training: http://www.nomagic.com/services/training.html
- UML Resources: http://www.uml.org/
- Magicdraw: http://www.magicdraw.com

1.2 NIEM-UML Background

Grown out of a grassroots initiative, the National Information Exchange Model (NIEM) was born as a best practice developed by a handful of state and local practitioners and defined in NIEM's predecessor, the Global Justice XML Data Model (GJXDM). Today, NIEM is a national program that empowers organizations to create and maintain meaningful data connections across their stove-piped IT systems, as well as their stakeholder base. NIEM provides data components and processes needed to create exchange specifications which support mission data sharing and exchange requirements. By providing a common vocabulary and mature framework to facilitate information exchange, NIEM enables communities to "speak the same language" as they share, exchange, accept, and translate information efficiently.

Traditionaly NIEM has been defined in terms of the eXtensible Markup Language (XML), XML Schema (XSD) and the normative NIEM platform specifications which include the NIEM Naming and Design Rules (NDR) Version 1.3 and the NIEM Model Package Description (MPD) Specification Version 1.0. These platform specifications are utilized without change in NIEM-UML and the NIEM-UML specification assists information architects and developers in producing NIEM model packages conforming to these standards. More information on NIEM is available at https://www.niem.gov/.

The use of UML to represent NIEM is part of the NIEM Program Management Office's (PMO) strategy in support of the NIEM community and intended to broaden NIEM adoption and in aligning to industry standards. NIEM-UML embraces the *Model Driven Architecture* (MDA) ® standards of the Object Management Group (OMG) ® to facilitate the separation of concerns between business needs and technology implementations. More information on OMG is available at http://www.omg.org/mda/.

1.3 OMG NIEM-UML and Cameo NIEM-UML

NIEM-UML is a standard of the Object Management Group (OMG). Cameo NIEM-UML is a product of NoMagic, Inc. provided in partnership with Model Driven Solutions. Cameo NIEM-UML is a "plugin" to the popular "Magicdraw" UML tool provided by NoMagic. The combination of Cameo NIEM-UML and Magicdraw implement the NIEM-UML specification with enhanced capabilities to better facilitate:

- Easy and intuitive modeling for NIEM-UML
- Understanding and reusing the NIEM refereice vocabularies

- Utilizing other aspects of UML to better capture business requirements and facilitate the full life-cycle of development
- Integrationing NIEM with the rest of the system
- Supporting Service Oriented Archtiecture with Cameo SoaML

Information on Cameo NIEM-UML can be found here: http://www.nomagic.com/products/cameo-workbench.html

1.4 Intended Users of NIEM-UML

One of the key goals for NIEM-UML is to allow modelers and developers to apply NIEM-UML with minimal effort in order to create new models or change existing models and ultimately to produce NIEM solutions. When modeling information exchanges, there are two distinct sets of requirements that lead to two approaches to modeling. The first set of requirements represents the business requirements of an organization. This set is relatively constant and consistent over time and entails modeling the capabilities the organization has, the processes the organization employs and the information the organization leverages. The second set is related to the technical implementation of an organization's capabilities, processes and information and varies as platforms and technologies change. These approaches are defined by MDA as the *Platform Independent Model* (PIM) and the *Platform Specific Model* (PSM) approaches, respectively. The "platform" for NIEM is considered to be XML Schema structured according to the NIEM naming and design rules (NDR) for XML Schema.

The two distinct sets of requirements lead to two different approaches to modeling. The PIM is mainly a business information modeling approach while the PSM is mainly a technical modeling approach. In practice, it is important to be able to model an information exchange leveraging both the business and the technical modeling approaches. Furthermore it is critical to have an active communication and effective collaboration between business and technical modelers to assure that the model represents the business requirements correctly and implements them effectively within the means of the current platform and technology. The structure of the NIEM-UML Profile is designed to meet the requirements of the two modeling communities described above and to allow for communication and collaboration between them. NIEM-UML also contains transforms that allow a PIM to automatically produce a PSM (using standard Model Driven Architecture (MDA) tooling) while allowing the modeler to augment the PIM with PSM considerations as required.

This guide is primarily focused on the use of the PIM to design and maintain NIEM exchange specifications.

1.5 Understanding NIEM-UML and Model Driven Architecture (MDA)

1.5.1 The NIEM Platform

Inherent in the idea of a platform independent model (PIM) is that there is some kind of "platform". What constitutes the platform may, to some degree, depend on the context and purpose of a model and the platform. In the case of NIEM-UML the platform is considered to be the artifacts that make up a NIEM model package, such as an Information Exchange Package (IEP). A primary element of a NIEM package is a XML schema defined in accordance with the NIEM NDR. Other aspects of the platform include the "Master Document" and other artifacts that make up a NIEM model package. This provides a degree of separation of the technical aspects of the IEP (i.e. XML schemas) from the business aspect. While XML Schema is less platform-specific than, say, a Java class, it is a specific way to render information and therefore a platform. Other platforms of interest at this time include the Resource Description Language (RDF) and JavaScript Object Notation, JSON.

The NIEM NDR [NIEM-NDR] and MPD Specification [NIEM-MPD] describe this XML Schema centric platform as well as the principles and model behind it. The platform specifications are used by NIEM-UML without alteration.

The NIEM-XML platform is expressed using W3C XML Schema (XSD) and XML technology. There are hundreds of rules for how to use XSD. The NIEM NDR adds approximately 200 rules that define NIEM conformance by generally constraining many XSD options. This enables greater interoperability and reuse while introducing an acceptable NIEM learning curve. NIEM-UML significantly reduces the requirement to learn details of the NIEM

NDR by employing the NIEM-PIM and the NIEM-PSM with MDA. The NIEM-PIM is intended to abstract the business rules and constructs of NIEM from the details of the technology platform.

1.5.2 Intent of the PIM

While parts of the NIEM platform are specific to the technology and the way to use that technology, other parts of NIEM are derived from the business requirements for an information exchange. The most striking example of this are the controlled vocabularies represented in NIEM reference namespaces. These controlled vocabularies represent the consensus of stakeholders, within specific communities of interest, on their information requirements and are reused in information exchanges (for example, IEPDs). There are also rules and conventions for how elements are named and how they are organized in a consistent structure. The NIEM-UML PIM, PSM, and mapping between them represent and enforce NIEM rules and conventions.

The PIM profile enforces certain NIEM rules using the Object Constraint Language (OCL) and also extends UML with “Stereotypes” and “Tagged Values” to represent NIEM specific concepts including but not limited to elements of the NIEM vocabulary, the NIEM reference schema, and NIEM concepts and rules which underlie its structure and maintain its consistency. While the PIM specializes NIEM, every effort has been made to make the representation of NIEM in UML correspond to commonly accepted patterns of modeling in UML. Someone familiar with UML should be able to start modeling quickly and in many cases, with minimal modification, may be able to reuse existing models to derive NIEM PSM artifacts and ultimately NIEM artifacts.

A NIEM PIM conforms to the rules and structure of the PIM profile described in this document and, with NIEM-UML conforming tooling and is able to produce a NIEM platform specification. Production of an IEPD from a PIM model is accomplished by mapping the PIM model to an IEPD, via the NIEM-PSM, using MDA technologies. Alternatively, an existing IEPD or domain update can be mapped to a PIM model via the reverse engineering mappings.

As discussed in the NIEM platform clause above, an IEPD is specific to a particular requirement as well as to the particular constrained structure of XML Schema described by the NDR and requirements outlined in the MPD Specification. The PIM is intended to be closer to the level of abstraction that business stakeholders can deal with, separating the concerns of the business information required from the specifics and complexities of the platform.

As part of the NIEM-UML specification the mapping from a PIM to the NIEM platform via the NIEM-PSM is specified. The mapping specifications are implemented by tool builders and most users will never have to understand them, but most users will be able to use them in the form of conforming tools. NIEM-UML conforming tools can transform a PIM model to a NIEM platform IEPD by leveraging the PSM and in doing so make sure that all of the business and technology rules are correctly applied because most of those artifacts are automatically generated. A NIEM-UML modeler will not need to understand the platform specific rules, or even W3C XML Schema, to build NIEM artifacts. Of course, developers will have to understand XML to use the NIEM platform.

NIEM-UML uses the NIEM-PIM and the NIEM-PSM to separate respective concerns based on the Model Driven Architecture (MDA) standards of the Object Management Group (OMG).

1.5.3 Intent of the PSM

Another clause of this specification defines the NIEM PSM profile. A platform specific model defines a direct representation of NIEM-XML, its structure and its technical rules when leveraging W3C XML Schema. The PSM is intended to represent the technology specific requirements and structure of the NIEM platform.

This profile can be employed by users who have familiarity with NIEM and its representational concepts in XML Schema. The PSM profile allows a user to design a UML model that is closely aligned with NIEM-conforming XML schemas. It consists of a relatively small set of UML constructs and stereotypes that map to equivalent XML schema constructs in NIEM. Conforming UML tools are able to import the UML representation (XMI) of the PSM profile and subsequently provide support for creating NIEM-UML constructs and stereotypes, as well as for entering the additional data required for NIEM conformance.

1.5.4 The NIEM PIM and the NIEM PSM

Figure 6-1 shows how the components of the NIEM-UML specification are used together in a conforming NIEM-UML tool suite.

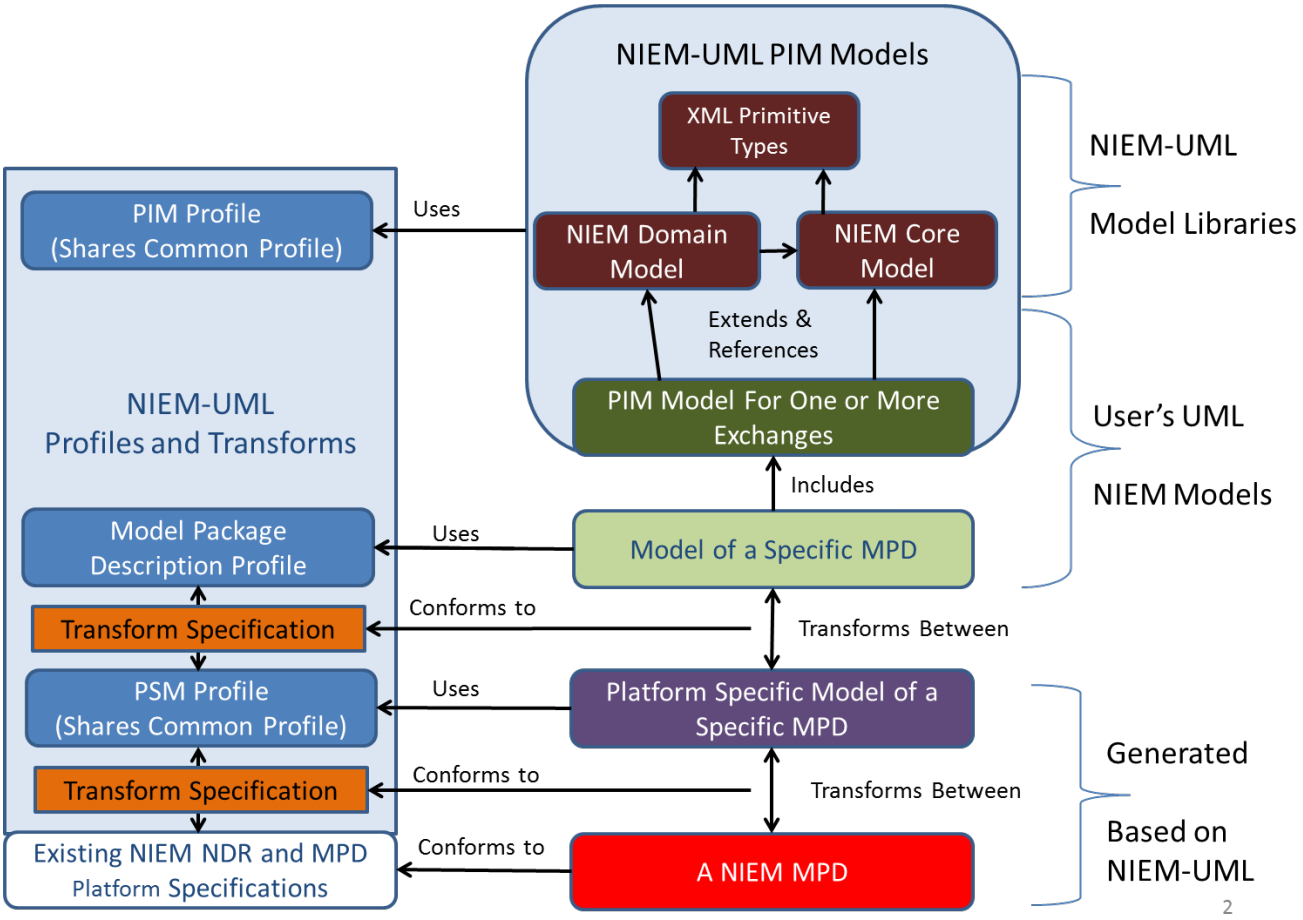


Figure 6-1 Components of the NIEM-UML specification

The component parts of the NIEM-UML specification are intended to be used together with tools to make it easy to model NIEM in UML and produce valid NIEM platform specifications. The diagram above shows the relationships between the elements of the NIEM-UML specification, a user's model and the resulting MPD, e.g. an IEPD. It is important to note that the MDA based structure and the separation of concerns between the PIM and PSM part of the NIEM-UML specification allows for representation of NIEM under a different platform if required in the future or to support integration of NIEM into legacy systems.

The intent of NIEM-UML (including the PIM and the PSM) is that tools can generate NIEM artifacts directly from the model based on Model Driven Architecture (MDA) and transformations specified in this document. This capability may or may not be achievable in a "generic" UML tool; supplemental tools or plug-ins may be required.

1.5.5 NIEM-UML Transformations

NIEM-UML also contains transformations from NIEM-UML business models (NIEM PIMs) to NIEM-UML technical models (NIEM PSMs) and from NIEM-UML technical models to NIEM-compliant XML schemas and MPDs. Further, stereotypes from the NIEM PSM profile can be used to enable provisioning of the NIEM PIM as a set of NIEM MPD artifacts. Stereotypes from the NIEM PIM Profile can be added to a NIEM PSM as features to enable transforming a NIEM PSM to a NIEM PIM.

To enable reuse of existing NIEM artifacts transformations are also provided to “reverse engineer” existing MPD artifacts to NIEM-UML.

The foreward and reverse engineering of NIEM technical artifacts to and from models is fully supported by Cameo NIEM-UML.

1.5.6 NIEM-UML Libraries

A central tenet of NIEM is reuse. NIEM-UML facilitates reuse by providing the NIEM reference namespaces as NIEM-UML models. The reference namespaces represent the reusable information sharing vocabularies defined as part of the NIEM process. These vocabularies are reused in all NIEM models.

The reference vocabularies are comprised of the “NIEM-Core” with extensions for 15 domains. Altogether the reference vocabularies contain in excess of 7000 concepts ready to be reused. Cameo NIEM-UML helps users understand and reuse the reference vocabularies.

2 NIEM-UML Modeling Guide

2.1 Overview

2.1.1 Introduction

Essential to NIEM-UML is respecting and supporting the distinct perspectives or "entry points" for using the NIEM-UML profile:

- The *platform independent perspective*, optimized for a logical UML representation of NIEM using UML norms and patterns.
- The *platform specific perspective*, optimized for a direct and isomorphic UML representation of NIEM as defined in XML Schema.
- The *model packaging perspective*, optimized for representing the packaging of NIEM namespaces, modeled from either the PIM or the PSM perspective, into NIEM MPDs.

Clause 2 specifies the kinds of conforming NIEM-UML models associated with each of the above three perspectives: NIEM PIM, NIEM PSM and NIEM MPD models. The NIEM-UML profile is then structured into sub-profiles used in creating each of these three kinds of models. Further, for simplicity and consistency, the NIEM PIM and NIEM PSM profiles are based on a profile of common UML elements, constraints and stereotypes. This provides a clear, precise and concise specification of each perspective while also clearly specifying overlapping elements without redundancy.

Figure 7-1 shows the resulting structure of the NIEM-UML Profile in terms of the three perspectives.

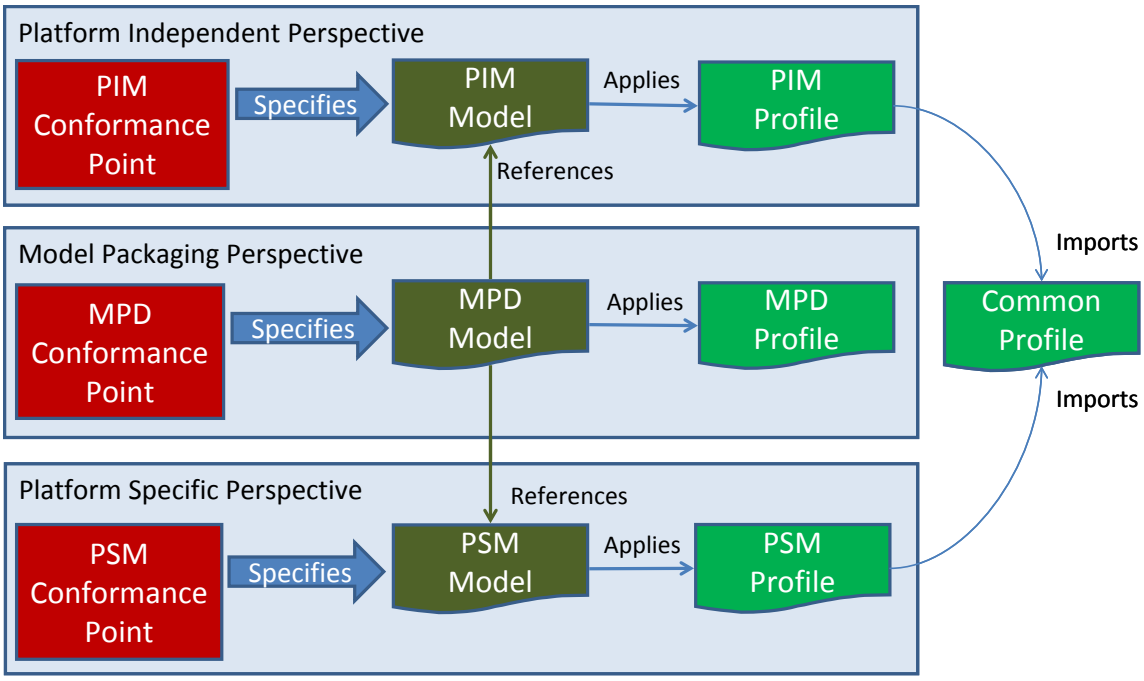


Figure 7-1 Structure of the NIEM-UML Profile

The remainder of this overview provides a summary description of each of the platform independent, platform specific and model packaging perspectives. Subsequent subclauses in this clause then discuss how to model various NIEM concepts across the three perspectives.

2.1.2 Platform Independent Perspective

A NIEM Platform Independent Model (PIM) is represented using a simplified UML class model with extensions for expressing NIEM semantics. The intent of the PIM is to capture a NIEM business vocabulary for use as a data schema within an MPD. A NIEM PIM is used in combination with a NIEM MPD model to create a complete NIEM specification.

The UML concepts shown in Table 7-1 have an interpretation in a NIEM-UML PIM and are supported by the normative mapping from a NIEM-UML model to NIEM conformant artifacts via the standard mappings. While other UML model elements may be used in a PIM for purposes of documentation or supporting other technologies, such model elements have no defined meaning with respect to NIEM and will not impact the mapping to NIEM artifacts.

Table 7-1 Platform Independent Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
Package	«Namespace»	7.2.1 Namespaces	A namespace package models a NIEM data schema
Types			
Class	None «ObjectType»	7.3.2 Object Types	Object type is the default for UML classes. A NIEM object type represents data about things with their own identity and lifespan that have some existence. An object may or may not represent a physical thing. It may represent something conceptual.
	See «RoleOf» Property and «RolePlayedBy» Generalization	7.3.3 Role Types	NIEM differentiates between an object and a role of the object. The term “role” is used here to mean a function or part played by some object. A class is interpreted as a role by means of a «RoleOf» association end or «RolePlayedBy» generalization.
	«AssociationType»	7.3.4 Association Types	A NIEM *association* is a specific relationship between NIEM objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of the relationship exist that are not attributable to the objects being related. Each end of the NIEM association is represented by a UML association end. Note that a UML association class may also be used (see below).
	«MetadataType»	7.3.5 Metadata Types	NIEM *metadata* is defined as “data about data.” This may include information such as the security of a piece of data or the source of the data. A Metadata Type models metadata.

Table 7-1 Platform Independent Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
	«AugmentationType» See also «Augments» Generalization	7.3.6 Augmentation Types	A NIEM *augmentation type* is a complex type that provides a reusable block of data that may be added to object types or association types.
	«AdapterType»	7.3.7 Adapter Types	An *adapter type* is a NIEM object type that adapts external models for use within NIEM. An adapter type creates a new class of object that embodies a single concept composed of external elements.
	«Choice»	7.5.4 Choice Groups	A *choice* is a group of properties such that when used as the type of a property exactly one of them may have a value in any instance of an enclosing type.
	«PropertyHolder»	7.5.2 Property Holders and Property References	Property holders are used to define properties that have no specific owner, or "top level" properties. These properties are generally referenced by other properties.
	isAbstract	7.3.1 Complex Types (abstract)	An abstract class may not have a direct instance, non-abstract subclasses of an abstract class may have instances.
Association Class	None	7.3.4 Association Types	A NIEM *association* is a specific relationship between NIEM objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of the relationship exist that are not attributable to the objects being related. UML association classes may be used to model NIEM associations with some limitations. Note that an «AssociationType» class may also be used, see above.
DataType *Also applies to PrimitiveType and Enumeration which are DataTypes*	«Union»	7.4.4 Unions	A *union* is a simple type whose values are the union of the values of one or more other simple types, which are the *member types* of the union.
	«List»	7.4.5 Lists	A *list* is a simple type having values each of which consists of a finite-length (possibly empty) sequence of atomic values. The values in a list are drawn from some atomic simple type (or from a union of atomic simple types), which is the *item type* of the list.

Table 7-1 Platform Independent Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
	«ValueRestriction»	7.4.1 Simple Types	The «ValueRestriction» stereotype applies to a UML data type that is a specialization of a more general data type. It defines restrictions on which values of the general data type are allowed as values of the specialized data type.
PrimitiveType	None	7.4.2 Primitive Types	A *primitive type* is a simple type defined in terms of a predefined set of atomic values such as strings and numbers.
Enumeration	None	7.4.3 Code Types	A UML enumeration represents a NIEM *code type*, which is a simple type, restricted to a specific set of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals. Each enumeration literal is a code value.
Relations			
Aggregation (Property / Association End)	None	7.5.1 Content and reference properties	A UML aggregation kind of “shared” or “composite” will result in content nested within the enclosing content. UML aggregation kind of “none” will result in a reference.
Generalization	None	7.3.1 Complex Types (type extension)	Each NIEM type may extend at most one other type due to XSD restrictions. Properties of the superclass are inherited and the subclass is substitutable for the superclass.
	«Augments»	7.3.6 Augmentation Types	Augments specifies what type an augmentation augments. Multiple augmentations may be inherited by a type or used as the types of properties. Note that a property with an «AugmentationApplication» as its type may also be used.
	«RolePlayedBy»	7.3.3 Role Types	RolePlayedBy defines the subtype as a role of the supertype. Such a role may have at most one instance per base type.
Realization	«References»	7.5.2 Property Holders and Property References	A References realization reuses class and property definitions from another class or namespace and is the basis for reusing NIEM reference namespaces.

Table 7-1 Platform Independent Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
Usage dependency	«Augmentation Application»	7.3.6 Augmentation Types	Augmentation application is a relation between a property whose type is an «AugmentationType» and a class. It restricts the classes that may have the property. Note that an «Augments» generalization may also be used, see above.
	«Metadata Application»	7.3.5 Metadata Types	Metadata application is a relation between a «MetadataType» class and any other class. It restricts the types that may have the metadata.
Properties			
Property / Association End	None	7.5.1 Properties	A *property* relates a NIEM object (the *subject*) to another object or to a value (the *object*). Property data describes an object as having a characteristic with a specific value or a particular relationship to another object.
	«RoleOf»	7.3.3 Role Types	NIEM differentiates between an object and a role of the object. The term “role” is used here to mean a function or part played by some object. Having a «RoleOf» property defines the owning class as a role. The role may have multiple occurrences for each base type.
Multiplicity (Property)	None	7.5.1 Properties	UML Multiplicity constrains how many values a NIEM property may have.
Subsets (Property)	None	7.5.3 Substitution Groups	Subset defines a property as being substitutable for another property. This expresses the NIEM substitution group concept.
Derived Union (Property)	None	7.5.3 Substitution Groups	A derived union defines a property whose values are entirely derived as the union of the values of properties that subset it. This expresses the NIEM concept of an abstract property.
General			
Name (NamedElement)	None	7.2.2 NIEM Names	NIEM PIM names are largely unconstrained as the mapping specifications will map the UML names into NIEM conformant names in the PSM and MPD artifacts. Naming conventions such that reasonable NIEM names are produced should still be practiced.

Table 7-1 Platform Independent Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
Element	«ReferenceName»	7.2.2 NIEM Names	Reference name specifies the NIEM conformant name for an element. This may be required if the name produced by the PIM-PSM mapping does not match a reference namespace or is otherwise not as required.
Comment	None «Documentation»	7.2.1 Namespaces 7.3.1 Complex Types 7.4.1 Simple Types 7.5.1 Properties	If a UML modeling element owns only one comment, it will be used by default as the NIEM documentation for that element. Otherwise the «Documentation» stereotype must be applied to one owned comment. Documentation text will be converted to being NIEM compliant.

2.1.3 Platform Specific Perspective

A NIEM Platform Specific Model (PSM) is represented using a simplified UML class model with extensions for expressing a NIEM XML Schema (XSD). The intent of a PSM is to capture a direct representation of a NIEM XML schema in UML. A NIEM PSM is used in combination with a NIEM MPD model to create a complete NIEM specification.

The UML concepts shown in Table 7-2 have an interpretation in a NIEM-UML PSM and are supported by the normative mapping from a NIEM-UML model to NIEM conformant artifacts via the mappings specified in Clause 9. Other UML elements are not permitted in a NIEM PSM if the PSM profile is applied "strictly".

Table 7-2 Platform Specific Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
Package	«Namespace»	7.2.1 Namespaces	A «Namespace» package models a NIEM data schema
Types			
Class	«ObjectType»	7.3.2 Object Types	A NIEM *object type* represents data about things with their own identity and lifespan that have some existence. An object may or may not represent a physical thing. It may represent something conceptual.
	«ObjectType» (Used as a role)	7.3.3 Role Types	NIEM differentiates between an object and a role of the object. The term "role" is used here to mean a function or part played by some object. A class is interpreted as representing a *role type* if it has one or more properties that identify the base type(s) of the role. By the NDR naming conventions, such properties must have names beginning with "RoleOf".

Table 7-2 Platform Specific Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
	«AssociationType»	7.3.4 Association Types	A NIEM *association* is a specific relationship between NIEM objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of the relationship exist that are not attributable to the objects being related. Each end of the NIEM association is represented by a UML association end.
	«MetadataType»	7.3.5 Metadata Types	NIEM *metadata* is defined as "data about data." This may include information such as the security of a piece of data or the source of the data.
	«AugmentationType»	7.3.6 Augmentation Types	A NIEM *augmentation type* is a complex type that provides a reusable block of data that may be added to object types or association types.
	«AdapterType»	7.3.7 Adapter Types	An *adapter type* is a NIEM object type that adapts external models for use within NIEM. An adapter type creates a new class of object that embodies a single concept composed of external elements.
	«Choice»	7.5.4 Choice Groups	A *choice* is a group of properties such that when used as the type of a property exactly one of them may have a value in any instance of an enclosing type.
	«PropertyHolder»	7.5.2 Property Holders and Property References	Property holders are used to define properties that have no specific owner, or "top level" properties. These properties are generally referenced by other properties.
	isAbstract	7.3.1 Complex Types (abstract)	An abstract class may not have a direct instance, non-abstract subclasses of an abstract class may have instances.
DataType *Also applies to PrimitiveType and Enumeration which are DataTypes*	«Union»	7.4.4 Unions	A *union* is a simple type whose values are the union of the values of one or more other simple types, which are the *member types* of the union.
	«List»	7.4.5 Lists	A *list* is a simple type having values each of which consists of a finite-length (possibly empty) sequence of atomic values. The values in a list are drawn from some atomic simple type (or from a union of atomic simple types), which is the *item type* of the list.

Table 7-2 Platform Specific Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
	«ValueRestriction»	7.4.1 Simple Types	The «ValueRestriction» stereotype applies to a UML data type that is a specialization of a more general data type. It defines restrictions on which values of the general data type are allowed as values of the specialized data type.
PrimitiveType	None	7.4.2 Primitive Types	A *primitive type* is a simple type defined in terms of a predefined set of atomic values such as strings or numbers.
Enumeration	None	7.4.3 Code Types	A *code type* is a simple type that represents a list of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals. Each enumeration literal is a code value.
DataType	«XSDRepresentation Restriction»	XSD Restriction	Restricts how data is formatted in XML
Relations			
Aggregation (Property)	None	7.5.1 Content and reference properties	A UML aggregation kind of “shared” or “composite” will result in content nested within the enclosing content. UML aggregation kind of “none” will result in a reference.
Generalization	None	7.3.1 Complex Types (type extension)	Each NIEM type may extend at most one other type due to XSD restrictions. Properties of the superclass are inherited and the subclass is substitutable for the superclass.
	«XSDRestriction»	XSD Restriction	Defines a type as restrictive the possible values in another type
Realization	«References»	7.5.2 Property Holders and Property References	A References realization reuses class and property definitions from another class or namespace.

Table 7-2 Platform Specific Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
	«XSDSimple Content»	7.3.2 Object Types	The «XSDSimpleContent» stereotype represents a relationship between two type definitions: the first is a complex type definition with simple content whose content type is the second. This relationship is implemented in XML Schema through base attribute on the xsd:extension or xsd:restriction element of the first type definition, the actual value of which resolves to the second type definition.
Usage dependency	«Augmentation Application»	7.3.6 Augmentation Types	Augmentation application is a relation between a property and a class. It restricts the classes that may have the property.
	«Metadata Application»	7.3.5 Metadata Types	Augmentation application is a relation between a metadata type and any other type. Its application restricts the types that may have the metadata
Properties			
Property	«XSDProperty»	7.5.1 Properties	A *property* relates a NIEM object (the *subject*) to another object or to a value (the *object*). Property data describes an object as having a characteristic with a specific value or a particular relationship to another object.
Property	«XSDAnyProperty»	7.5.1 Properties	An «XSDAnyProperty» property may have a value of any type. It is implemented in XML schema as an xsd:any.
Multiplicity (Property)	None	7.5.1 Properties	Multiplicity constrains how many values a property or association end may have.
Subsets (Property)	None	7.5.3 Substitution Groups	Subset defines a property as being substitutable for another property. This expresses the NIEM substitution group concept.
Derived Union (Property)	None	7.5.3 Substitution Groups	Derived union defines a property whose values are entirely derived as the union of the values of properties that subset it. This expresses the NIEM concept of an abstract property.

Table 7-2 Platform Specific Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
General			
Name (NamedElement)	None	7.2.2 NIEM Names	The names of UML elements in a PSM must comply with the NDR rules for names within a NIEM XML Schema.
Comment	«Documentation»	7.2.1 Namespaces 7.3.1 Complex Types 7.4.1 Simple Types 7.5.1 Properties	Each UML model element in a PSM that represents a NIEM component that is required to have documentation must have one owned comment that has the «Documentation» stereotype applied.

2.1.4 Model Packaging Perspective

A NIEM Model Package Description specifies the NIEM artifacts that are to be produced from a NIEM-UML model and rendered into a NIEM MPD package. The ModelPackageDescription imports PIM and/or PSM namespaces and produces an MPD based on the provided metadata contained within the stereotypes.

The UML concepts shown in Table 7-3 have an interpretation in a NIEM-UML MPD model and are supported by the normative mapping from a NIEM-UML model to NIEM conformant artifacts via the mappings specified in Clause 9.

Table 7-3 Model Packaging Perspective Modeling Summary

UML Element	Stereotype	NIEM Concept Reference	Note
Component	«ModelPackage Description»	Model Package Description	A Model Package Description (MPD) describes a package of NIEM artifacts, these include IEPDs and domain updates. There are multiple kinds of MPDs as described in the Model Package Description Specification [NIEM-MPD].
Dependency	«ModelPackage Description Relationship»	Model Package Description Relationship	A Model Package Description Relationship defines the relationship between MPDs. This includes dependency and version information.
ElementImport	«ModelPackage Description File»	MPD Artifact	Model Package Description File import defines a modeled namespace or artifact that is to be included in an MPD.
Package	None	None	Packages may be used to organize MPD models but have no interpretation for a NIEM MPD.

2.2 Modeling Namespaces

2.2.1 Namespaces

2.2.1.1 Background

A *namespace* provides a means to qualify the names of a group of NIEM components. Following the conventions of [XMLNamespaces], a namespace is identified by a URI reference. All the names within a single NIEM namespace are required to be distinct, though the same name may be used across different namespaces.

NOTE. The XML Schema specification defines separate *symbol spaces* for type, attribute and element names, allowing components in different symbol spaces to have the same name, even within a single namespace. However, the NIEM naming rules imply the use distinct names across all the symbol spaces of a namespace [NDR 9].

2.2.1.2 Representation

Common

A NIEM namespace is represented as a UML package with the stereotype «Namespace» applied, with the namespace URI provided as the value of the targetNamespace attribute of the stereotype. Table 7-4 shows the kinds NIEM components whose names are included in a NIEM namespace along with how these components are represented as UML model elements within the «Namespace» package that represents the NIEM namespace.

Table 7-4 NIEM Components included in a NIEM Namespace

NIEM Component	UML Representation
Complex Type	Class (see Subclause 7.3)
Simple Type	Data Type (see Subclause 7.4)
Property Declaration	Property (see Subclause 7.1)

A «Namespace» package is used to group those model elements of the kinds shown in Table 7-4 that represent NIEM components to be placed in a single NIEM namespace. A «Namespace» package may have subpackages, and any relevant model element in any of those subpackages (or any further nested packages, to any level) is also considered to represent a member of the NIEM namespace identified for the «Namespace» package. However, a «Namespace» package may not be contained, directly or indirectly, in any other «Namespace» package.

Sometimes, a NIEM-UML model will import non-NIEM models or otherwise include modeling for non-NIEM content relevant to NIEM messages (see also Subclause 7.3.7 on Adapter Types). The «Namespace» stereotype may also be applied to packages containing models of non-NIEM conformant content, in order to specify a targetNamespace URI. However, in this case the isConformant attribute of the stereotype should be set to false.

PIM

In a PIM, the concept of a NIEM namespace is extended to encompass the representation of a platform-independent information model. The PIM «InformationModel» stereotype is a specialization of the common «Namespace» stereotype that also allows for the identification of a *default purpose* for the represented information model (such as reference, subset, extension or exchange). If no other purpose is specified when an «InformationModel» package is referenced in an MPD model, then the default purpose is used (see Subclause 7.6.2). This allows for the identification of information models in a PIM that are, e.g., specifically intended to be subsets of references models, extensions of such subsets, etc.

An «InformationModel» package provides the logical scoping for the NIEM naming of model elements representing NIEM components (see Subclause 7.2.2). This includes UML properties representing NIEM properties, even though a UML property is not a packageable element. The UML namespace for a property is the UML classifier that owns

the property. However, in NIEM every property *declaration* is considered to be "top level", and, so, the NIEM property names are included in the NIEM namespace. (This is discussed further in Subclause 7.5.2.)

Every «InformationModel» package must be documented. If the package has only one owned comment, that is considered to provide the required documentation. Otherwise, the package must have exactly one owned comment with the stereotype «Documentation» applied that provides the required documentation.

PSM

A «Namespace» package represents an XML schema. The target namespace for the schema is supplied as the value of the targetNamespace attribute of the stereotype. The elements in the package (or any subpackage, to any level of nesting) representing NIEM components (as indicated in Table 7-4) are implemented as components of the XML schema represented by package. If the isConformant attribute is true, then the represented XML schema shall be NIEM-conformant.

A «Namespace» package in a PSM must have an owned comment with the stereotype «Documentation» applied, the body of which provides the definition documentation for the represented XML schema.

2.2.1.3 Mapping Summary

PIM to PSM Mapping

- A package in a PIM shall map to a package in the PSM, with corresponding owned members mapped from the PIM. If the PIM package has the «Namespace» stereotype applied, then the PSM package also has the «Namespace» stereotype applied, with the same values for stereotype attributes. If the PIM Package has the «InformationModel» stereotype applied, then the PSM package has the «Namespace» stereotype applied, with the same values for the common stereotype attributes (see Subclause 7.6.2.3 on the handling of the «InformationModel» defaultPurpose attribute for MPD modeling).
- If a «Namespace» or «InformationModel» package in a PIM has exactly one owned comment, then the corresponding PSM package shall have an owned comment with the «Documentation» stereotype applied and the same body as the PIM package's comment. Otherwise, the PSM package shall have an owned comment with the «Documentation» stereotype applied and the same body as the «Documentation» comment owned by the PIM package. The comment body is adjusted to conform to NIEM conventions.

PSM to XML Schema Mapping

- A package in a PSM with the stereotype «Namespace» applied shall map to an XSD schema with «Namespace» stereotype attributes mapped as given in Table 7-5 and elements in the package mapped per Table 7-4.

Table 7-5 Mapping of a «Namespace» package to an xsd:schema

Stereotype Attributes	Schema Property
Namespace::targetNamespace	`xsd:schema/@targetNamespace`
Namespace::version	`xsd:schema/@version`
Namespace::isConformant	`xsd:schema/xsd:annotation/xsd:appinfo/i:ConformantIndicator`

- The «Documentation» comment owned by a «Namespace» package in the PSM shall map to the documentation for the XML schema mapped from the package, with the body of the comment providing the `xsd:schema/xsd:annotation/xsd:documentation` for the schema definition.

2.2.1.4 Example

PIM and PSM Representation

Figure 7-2 shows an example of a NIEM namespace represented as a package. The package contains a class that represents a NIEM object type (see Subclause 7.3.2). The properties of this class represent both the declaration of NIEM properties and the use of those properties in the context of the object type represented by the class (see Subclause 7.5.2). Therefore, the names of the object type and all the properties are members of the identified NIEM namespace.

Figure 7-2 Representation of a NIEM-conforming XML schema as a UML package

XML Schema Representation

The package shown in Figure 7-2 represents the following XML schema (with the type definitions elided):

```
<xsd:schema
    targetNamespace="http://niem.gov/niem/domains/cbrn/2.1"
    version="1">
    <xsd:annotation>
        <xsd:documentation>
            Chemical, Biological, Radiological, and Nuclear Domain
        </xsd:documentation>
        <xsd:appinfo>
            <i:ConformantIndicator>true</i:ConformantIndicator>
        </xsd:appinfo>
    </xsd:annotation>
   ...
</xsd:schema>
```

2.2.2 NIEM Names

2.2.2.1 Background

The NIEM NDR includes extensive rules on the naming of NIEM components. A *NIEM name* is a name of a NIEM component that follows the naming rules given in [NDR 9]. A NIEM name has the form of a sequence of required object class, property and representation terms, with optional qualifiers for these terms.

2.2.2.2 Representation

Common

The uniqueness rules for NIEM component names in a NIEM namespace are based on the use of proper NIEM names, regardless of what names are used for the corresponding model elements in a PIM. Therefore, every model element in a NIEM-UML model that represents a NIEM component as listed in Table 7-4 is considered to have a corresponding NIEM name.

PIM

The names of the UML model elements representing NIEM components in a PIM are not required to comply with the NDR naming rules. In particular, the names of UML elements do not need to include the representation-term suffix, which may be entirely determined by the kind of the element.

NOTE. Part of the rationale for including a representation term in all NIEM names is: "It helps prevent name conflicts and confusion. For example, elements [properties] and types may not be given the same name." [NDR 9.11] However, UML naming rules allow model elements of different kinds (e.g., classes and data types) to have the same name within a single UML namespace, and properties are scoped in classifier namespaces rather than package namespaces. Therefore, UML models generally do not use a convention of having a representation term suffix on model element names.

Nevertheless, every model element in a PIM that represents a NIEM component has a NIEM name. The NIEM name for a PIM element may be specified explicitly by applying the «ReferenceName» stereotype to the element and setting the NIEMName attribute. If the PIM element does not have the «ReferenceName» stereotype applied, and its UML name conforms to the NDR naming rules, then this is also the NIEM name for the element. Otherwise, the NIEM name for the element is constructed from the UML name (generally by appending a representation term suffix) as specified in the default PSM mapping rules in subsequent subclauses covering each kind of item.

NOTE. The rules for constructing NIEM names are intended to produce names that are syntactically valid according to the rules for required name prefixes and/or suffixes. It is still the responsibility of the modeler to provide UML names for model elements representing NIEM components that have semantically appropriate object-class, property and qualifier terms (per [NDR 9.8., 9.9, 9.10]), so that the constructed NIEM names are fully conformant.

The name of PSM element mapped from a PIM element shall be the NIEM name of the PIM element. (Note that this name rule does not apply if the PIM element represents a member of a non-NIEM namespace – that is, if the element is contained in a «Namespace» package with isConformant = false.)

PSM

The names of UML model elements representing NIEM components in a PSM are required to comply with the NDR naming rules. Therefore, the NIEM name of such a model element in a PSM is the same as its UML name.

2.2.2.3 Mapping Summary

PIM to PSM Mapping

- If a class, data type or property in a PIM is contained (directly or indirectly) within a «Namespace» package with isConformant=true, then it shall map to a corresponding class, data type or property in the PSM whose name is the NIEM name of the PIM element. Otherwise it shall map to a corresponding PSM element with the same name as the PIM element.
- If an element in a PIM has the «ReferenceName» stereotype applied, then its NIEM name shall be the value of the NIEMName attribute of the stereotype.

2.2.2.4 Example

Figure 7-3 shows a class representing a NIEM object type (see Subclause 7.3.2) with the name PersonClass, which does not conform to the NIEM naming rules for object types. The class has the «ReferenceName» stereotype applied, giving it a NIEM name of "PersonType", which is conformant.

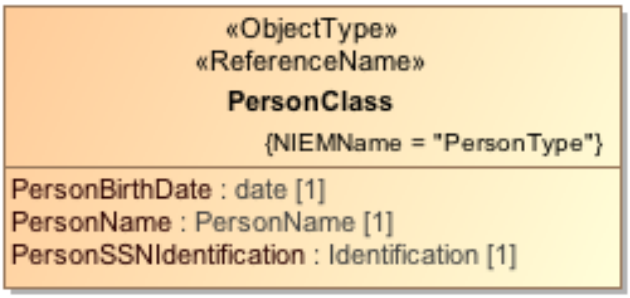


Figure 7-3 Specification of a NIEM name using the «ReferenceName» stereotype

2.3 Modeling Complex Types

2.3.1 Complex Types

2.3.1.1 Background

A *complex type* represents any structured data used for information exchange [NIEM-NDR 7.4.1]. A NIEM type shall be one of the following kinds of types:

- An object type
- A role type
- An association type
- A metadata type
- An augmentation type
- An adapter type

2.3.1.2 Representation

Common

A complex type is represented as a UML class. The different kinds of complex type are distinguished using the stereotypes summarized in Table 7-6, which are all specializations of the abstract «NIEMType» stereotype. Subsequent subclauses provide the details on how to model each kind of complex type.

Table 7-6 Complex Type Representation

Complex Type	Representation
Object Type	Apply «ObjectType» to the class. (In a PIM this is the default, so the stereotype is not required.) See Subclause 7.3.2.
Role Type	Identify one or more properties as role-of properties. (In a PIM, this may be done by applying «RoleOf» to a property or by using a «RolePlayedBy» generalization.) See Subclause 7.3.3.
Association Type	Apply «AssociationType» to the class. (In a PIM, alternatively use an association class.) See Subclause 7.3.4.
Metadata Type	Apply «MetadataType» to the class. See Subclause 7.3.5.
Augmentation Type	Apply «AugmentationType» to the class. (In a PIM, alternatively use an «Augments» generalization.) See Subclause 7.3.6.
Adapter Type	Apply «AdapterType» to the class. See Subclause 7.3.7.

In general, a «NIEMType» class may be the generalization for other «NIEMType» classes of the same type. A general class may optionally be modeled as *abstract*, meaning that there are no direct instances of that class itself, only of (non-abstract) subclasses of the class.

A «NIEMType» class may also be the client of a realization stereotyped as «Restriction», whose supplier is another «NIEMType» class of the same type, the *base type* for the restricted type. In this case, the client class may list a subset of the attributes of the supplier class. Any attributes not so listed must have a multiplicity lower bound of 0 in the supplier class. Instances of a restricted type are considered to be substitutable for instances of the base type, but any attributes not explicitly listed in the restricted type are mandated to be empty in any instance of that type.

(Note that an «AdapterType» class may not participate in generalizations or «Restriction» realizations – see Subclause 7.3.7.)

PIM

There are a number of default notations and additional representations allowed in a PIM that are not allowed in a PSM. These are indicated in Table 7-6 and discussed further in subsequent subclauses.

Every «NIEMType» class must be documented. If the class has only one owned comment, that is considered to provide the required documentation. Otherwise, the class must have exactly one owned comment with the stereotype «Documentation» applied that provides the required documentation.

In a PIM, additional notations using generalization are allowed in the modeling of role and augmentation types (see Subclauses 7.3.3 and 7.3.6). However, a «NIEMType» class may be the special class in at most one generalization that is not marked as a «RolePlayedBy» stereotype or the generalization of an augmentation type, and it may not have such a generalization if it is the client of a «Restriction» realization.

PSM

A «NIEMType» class in a PSM represents a NIEM type that is implemented as a complex type definition. The UML properties of the class represent the NIEM properties (XSD attributes and elements) of the complex type.

A «NIEMType» class in a PSM must have an owned comment with the «Documentation» stereotype applied, the body of which becomes the content of the documentation element in the complex type definition.

The class may be the special class in at most one generalization, the general class of which must also represent a NIEM type. The complex type represented by the general class is then the base type for the complex type

represented by the special class, and the complex type represented by the special class is an extension of the base type. A class marked as abstract represents an abstract complex type.

The class may be the client of at most one «Restriction» realization, and it may not be both the client of a «Restriction» realization and the special class in a generalization. The complex type represented by the supplier class is then the base type for the complex type represented by the client class, and the complex type represented by the client class is a restriction of the base type.

2.3.1.3 Mapping Summary

PIM to PSM Mapping

- A class in a PIM shall map to a corresponding class in the PSM, with corresponding properties mapped from the properties of the PIM class.
- If the class is the special classifier in a generalization, then the corresponding class in the PSM shall be the special classifier in a generalization to the class mapped from the general class in the PIM.
- If the class is the client of a realization with the «Restriction» stereotype applied, then the corresponding class in the PSM shall be the client of a «Restriction» realization whose supplier is the class mapped from the supplier class in the PIM.
- If a «NIEMType» class in a PIM has exactly one owned comment, then the corresponding PSM class shall have an owned comment with the «Documentation» stereotype applied and the same body as the PIM class comment. Otherwise, the PSM class shall have an owned comment with the «Documentation» stereotype applied and the same body as the «Documentation» comment owned by the PIM class. The comment body is adjusted to conform to NIEM conventions.

PSM to XML Schema Mapping

- A class in a PSM with a «NIEMType» stereotype (i.e., one of the stereotypes listed in Table 7-4) applied shall map to a corresponding complex type definition with the `xsd:complexType/@name` given by .the class name.
- If the class is the specific classifier in a generalization, then the corresponding complex type definition shall be an extension, with the base type definition being the type definition mapped from the general class.
- If the class is the client of a realization with the «Restriction» stereotype applied, then the corresponding complex type definition shall be a restriction, with the base type definition being the type definition mapped from the supplier class
- If the class is not the specific classifier in a generalization or the client of a «Restriction» realization, then the base type definition for the complex type definition shall be `s:ComplexObjectType` and the complex type shall be an extension.
- The «Documentation» comment owned by a «NIEMType» class in the PSM shall map to the documentation for the XML complex type definition mapped from the class, with the body of the comment providing the `xsd:complexType/xsd:annotation/xsd:documentation` for the complex type definition.

2.3.2 Object Types

2.3.2.1 Background

An *object type* is a type definition, an instance of which asserts the existence of an object. An object type represents some kind of object: a thing with its own lifespan that has some existence. The object may or may not be a physical object. It may be a conceptual object. [NIEM-NDR 7.4.1]

2.3.2.2 Representation

Common

A NIEM object type is represented as a UML class with the stereotype «ObjectType» applied. The properties of an «ObjectType» class model the structured data represented by the object Type.

NOTE. In NIEM, the term *object* is used only to refer to an instance of an object type, whereas in UML an object may be the instance of any class. In order to avoid confusion due to this difference in terminology, the qualified terms *NIEM object* and *UML object* will be used in this document.

PIM

In a PIM, a class representing an object type is not required to be stereotyped. A class with no stereotype is considered by default to be an object type.

The properties of a class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.1.

PSM

An «ObjectType» class represents a NIEM object type that is implemented in XML Schema as a complex type definition. Normally, the complex type definition for an object type will have complex content. The owned attributes of the «ObjectType» class represent the property references (attribute uses and element particles) within the complex content.

However, a PSM may also explicitly model the case of an object type with simple content. If the «ObjectType» class is the client of an «XSDSimpleContent» realization, then it represents an object type that is implemented as a complex type definition with simple content. The simple content is given by the simple type represented by the supplier of the «XSDSimpleContent» realization, which must be a UML data type (see Subclause 7.4 on modeling simple types).

2.3.2.3 Mapping Summary

PIM to PSM Mapping

- A class in a PIM with no stereotype applied that is not the special class in a «Augments» generalization (see Subclause 7.3.6) shall map to a class in the PSM with the «ObjectType» stereotype applied.
- If a class in a PIM representing an object type does *not* have the «ReferenceName» stereotype applied, then its NIEM name is determined as follows:
  - If the PIM class name ends in “Type”, then the NIEM name shall be the same as the UML name.
  - Otherwise, the NIEM name be the PIM class name with “Type” appended.

PSM to XML Schema Mapping

- If a class in a PSM with the «ObjectType» stereotype applied is the client of a «XSDSimpleContent» realization, then the complex type definition mapped from the class shall have simple content with the simple type mapped from the supplier of the realization as its base.
- If a class in a PSM with the «ObjectType» stereotype applied is *not* the client of a «XSDSimpleContent» realization, then the complex type definition mapped from the class shall have complex content and:
  - The properties of the class shall map to corresponding property references (XSD attribute uses and element particles) in the complex content of the complex type definition mapped from the class.
  - If the class is not the specific classifier in a generalization, then the complex type definition mapped from the class shall be an extension with the base type being `s:ComplexObjectType`.

2.3.2.4 Examples

PIM Representation

Figure 7-4 shows an example of a Person object type represented as a class in a PIM. The identification of the class as representing an object type is implicit, since it has no stereotype.

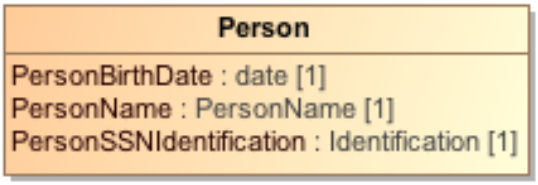


Figure 7-4 Representation of a NIEM object type as a UML class in a PIM

PSM Representation

Figure 7-5 shows the same object type represented as a class in a PSM. The class is structurally identical to the representation in the PIM, but the stereotype «ObjectType» is explicit in the PSM and the class is named PersonType, conforming to the NIEM NDR naming rules for object types [NIEM-NDR 9.12.1]. Note also the attached «Documentation» comment.

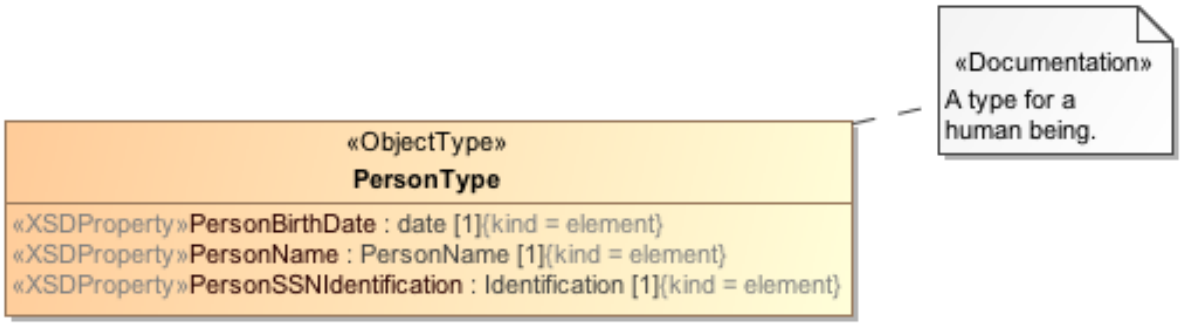


Figure 7-5 Representation of a NIEM object type as a UML class in a PSM

XML Schema Representation

The complex type definition corresponding to the PSM PersonType class is then (with the property declarations elided):

```
<xsd:complexType name="PersonType">
   <xsd:annotation>
      <xsd:documentation>A type for a human being.</xsd:documentation>
      <xsd:appinfo>
         <i:Base i:name="ComplexObjectType"/>
      </xsd:appinfo>
   </xsd:annotation>
   <xsd:complexContent>
      <xsd:extension base="s:ComplexObjectType">
      ...
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

2.3.3 Role Types

2.3.3.1 Background

A *role* is a function or part played by some NIEM object. A *role type* is a type that represents a particular function, purpose, usage, or role of a NIEM object. [NIEM-NDR 7.4.2]

2.3.3.2 Representation

Common

The simplest way to represent a role is simply to use a property, which models a function played by a NIEM object in some context, where the name of the property is the role name. In particular, a simple role such as this would most often be represented in UML as an association end. No stereotype is required. (See also Subclause 7.1 on the modeling of properties.)

However, in many cases there is a need to represent characteristics and additional information associated with a role. In this case, a role type provides a location for this additional information. A role type is modeled as an object type (see Subclause 7.3.2) with a *role-of* property. The type of this role-of property is the *base type* of the role type, and instances of the base type are said to *play* the role defined by the role type.

PIM

In a PIM, a role type may be defined either by explicitly modeling a role-of property or by modeling the role type with a generalization to the base type. If an explicit role-of property is modeled, then it is identified by applying the «RoleOf» stereotype to the UML property representing it. If a generalization is used, then this is identified by applying the «RolePlayedBy» stereotype to it.

An explicit «RoleOf» property of a role type may be the opposite end of an association between the role type and its base type (note that it is the association *end* that is stereotyped, not the association). If the «RoleOf» property is an association end, then the multiplicity of the near end of the association may be used to distinguish between two semantics interpretations of the concept of a “role”:

1. The role is repeated for each relationship that expresses the role. In this case the near end multiplicity shall be 0..*. This is also the only interpretation possible with the role-of property is not modeled as an association end.

   For example, consider a Victim role type with a Person base type. In this interpretation, there would one victim object each time a person was a victim. This means that there could be many victim objects for each person and one victim object each time the person was a victim.

2. The role occurs at most once for each base object. In this case the near end multiplicity shall be 0..1.

   In this interpretation of the Victim example, each person may play the victim role at most once – there may only be zero or one victim object for each person object. Each such victim object would need to capture information on *all* the crimes of which the person has been a victim. This interpretation corresponds more closely to a “victim data base”, with at most one entry for each person.

Modeling a role type as a specialization of the base type is an alternative representation for the second interpretation above. In this case the role type is *not* modeled with an explicit role-of property, but the generalization to the base type is instead stereotyped «RolePlayedBy». Semantically, this model represents the ability to dynamically classify instances of the base type as also being classified as being instances of the role type (UML semantics allow a UML object to have multiple types that may change over time). Since an instance of the base type can only be classified as an instance of the role type or not (corresponding to playing the role or not), the use of a «RolePlayedBy» generalization always corresponds to the second semantic interpretation of “role” above. (Note also that the specialization of a class by a role type is orthogonal to any other specializations of the base type. A base type may play multiple roles and may also be separately specialized.)

PSM

In a PSM, a role-of property is identified by having a naming beginning with “RoleOf”. Such a property must have aggregation=none. A role type is otherwise implemented exactly as for any other object type. (Note that this means the interpretation, above, can’t be explicitly represented in a PSM.)

2.3.3.3 Mapping Summary

PIM Representation Mapping

- An «ObjectType» class with a generalization that is stereotyped «RolePlayedBy» shall be considered equivalent to an otherwise identical class with the generalization replaced by a unidirectional association to the general (base) class such that:
  - The opposite (navigable) association end has the same name as the base class, multiplicity 1..1 and the stereotype «RoleOf» applied. If the «RolePlayedBy» generalization had the «ReferenceName» stereotype applied, then this end also has the «ReferenceName» stereotype applied, with the same value for the NIEMName attribute.
  - The near association end has multiplicity 0..1.

PIM to PSM Mapping

- The NIEM name of a property in a PIM with the «RoleOf» stereotype applied but not the «ReferenceName» stereotype is determined as follows:
  - If the PIM property name beings with “RoleOf”, then the PIM property name shall be the NIEM name.
  - Otherwise, the NIEM name shall be the PIM property name prefixed by “RoleOf”.

2.3.3.4 Examples

PIM Representation

Figure 7-6 shows the definition of the two role types Subject and Victim for the base type Person. This structure allows for a person to be a subject and/or a victim at the same time and for those conditions to change over time. The same person can be a subject or a victim multiple times (corresponding to the first semantic interpretation of “role”). Note that this model also allows different people to be the same victim.

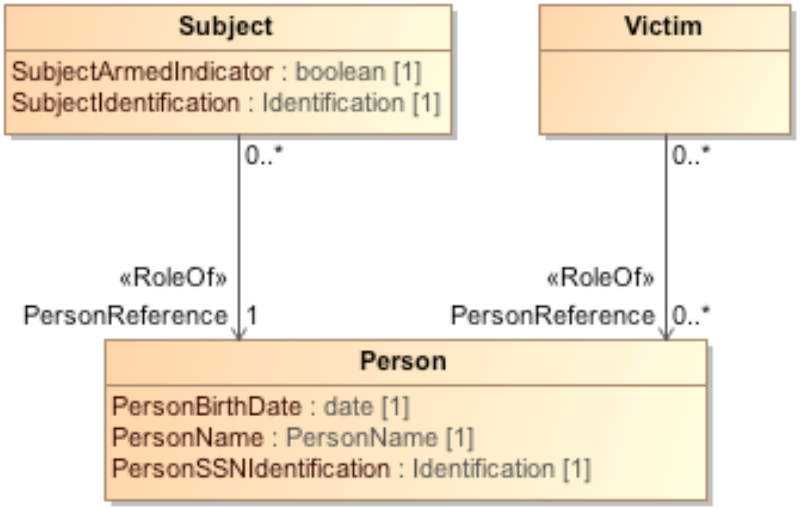


Figure 7-6 Representation of role types using role-of properties in a PIM

Figure 7-7 shows the «RolePlayedBy» representation of an FBI Agent as a role of a person which cooresponds to the second interpretation of a role. The same person could play this role as well as others at the same time but is only an FBI Agent once, at any one time.

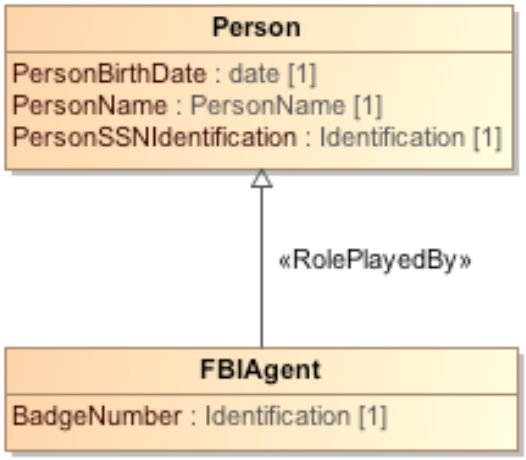


Figure 7-7 Representation of a role type using a generalization in a PIM

PSM Representation

Figure 7-8 shows the FBI Agent role type shown in Figure 7-7 as represented in the PSM. Note the required naming of the role-of properties.

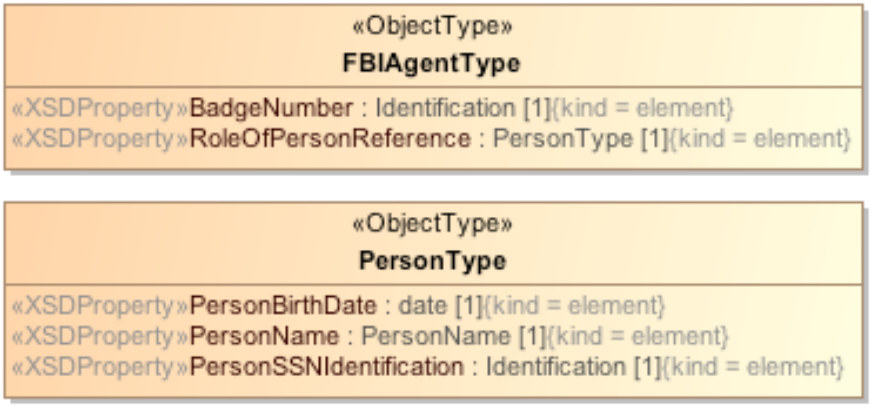


Figure 7-8 Representation of a role type in a PSM

XML Schema Representation

The SubjectType and VictimType role types shown in Figure 7-6 are represented in XML Schema as follows:

```
<xsd:complexType name="subjectType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A data type for a person who is involved or suspected of being
involved in an incident or criminal activity.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
```

```
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" ref="nc:RoleOfPersonReference"/>
        <xsd:element maxOccurs="1" minOccurs="1" ref="j:SubjectArmedIndicator"/>
        <xsd:element maxOccurs="1" minOccurs="1" ref="j:SubjectIdentification"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="VictimType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A data type for a person who suffers injury, loss, or death as
a result of an incident.</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="nc:RoleOfPersonReference"/>
      </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

2.3.4 Association Types

2.3.4.1 Background

A NIEM *association* is a specific relationship between NIEM objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of the relationship exist that are not attributable to the objects being related. A NIEM *association type* is a type that establishes a relationship between objects, along with the properties of that relationship. An association type provides a structure that does not establish existence of an object but instead specifies relationships between objects. [NIEM-NDR 7.4.3]

2.3.4.2 Representation

Common

A NIEM association is represented as a UML class with the «AssociationType» stereotype applied. The participants in an association are represented as properties of the «AssociationType» class. All the properties of an association type representing associated entities must be reference properties, which are represented by UML properties with aggregation=none (see also Subclause 7.5.1 on Properties).

NOTE. In NIEM, an association type is essentially also an object type. Therefore, an instance of an association type is a NIEM object.

PIM

Alternatively, a NIEM association may be represented as a UML association class, which is a model element that is both an association and a class in UML. The participants in the NIEM association are modeled as the ends of the association class. The association class may be used as the type of other properties.

An instance of a UML association class always has exactly one object participating in each end of the association. Thus, an association class models a NIEM association type whose properties all have multiplicity 1..1. A NIEM association type whose associated objects have multiplicities other than 1..1 cannot be modeled as a UML association class.

The ends of a UML association class have multiplicity. However, this multiplicity constrains the instantiation of the association class, not the number of objects that participate in each instance. For example, if an IncidentVictimAssociation is represented as an association class with multiplicity 0..* on both of its Incident and Victim association ends, then this means that there may be multiple instances of IncidentVictimAssociation with the same Incident but different Victims, and there may also be multiple instances with the same Victim but different Incidents. However, each individual instance of IncidentVictimAssociation is still between exactly one Incident and one Victim.

NOTE. A NIEM association type is always represented as a *class* in NIEM-UML, as either a regular class stereotyped as «AssociationType» or as an association class. It is never represented as a plain UML association. Instead, a UML association may be used to model a NIEM property (see Subclause 7.5.1).

PSM

An «AssociationType» class represents a NIEM association type that is implemented in XML Schema as a complex type definition with complex content. The owned attributes of the «AssociationType» class represent the element references within the complex content or properties of the association type.

2.3.4.3 Mapping Summary

PIM Representation Mapping

- An association type represented as an association class shall be considered equivalent to a class with the «AssociationType» stereotype applied and a unidirectional UML association corresponding to each end of the association class, such that:
  - The multiplicity of the opposite (navigable) end of the association is 1..1 and its name is the same as the name of the end of the association class.
  - The multiplicity of the near end of the association is the same as the multiplicity of the association class.

PIM to PSM Mapping

- A class in a PIM with the «AssociationType» stereotype applied shall map to a corresponding class in the PSM with the «AssociationType» stereotype applied.
- If a class in a PIM has the «AssociationType» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM class name ends in “AssociationType”, then the NIEM name shall be the same as the PIM class name.
  - If the PIM class name ends in “Association”, then the NIEM name shall be the PIM class name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM class name with “AssociationType” appended.

PSM to XML Schema Mapping

- A class in a PSM with the «AssociationType» stereotype applied shall map to a complex type definition mapped with complex content and:
  - The properties of the class shall map to corresponding element references in the complex content of the complex type definition mapped from the class.
  - If the class is not the specific classifier in a generalization, then the complex type definition mapped from the class shall be an extension with the base type being `s:ComplexObjectType`.

2.3.4.4 Example

PIM Representation

Figure 7-9 represents a NIEM association between incidents and victims. Each association is a relationship between exactly one incident and one victim. Since the properties of the IncidentVictimAssociation association type are modeled as UML associations, multiplicities may be shown on the near ends of the associations. This explicitly models that a victim can be a victim in any number of incidents and an incident may have any number of victims. (More restricted multiplicities may also be used, modeling additional constraints in the PIM, even though these cannot be carried forward to the PSM – see Subclause 7.5.1.)

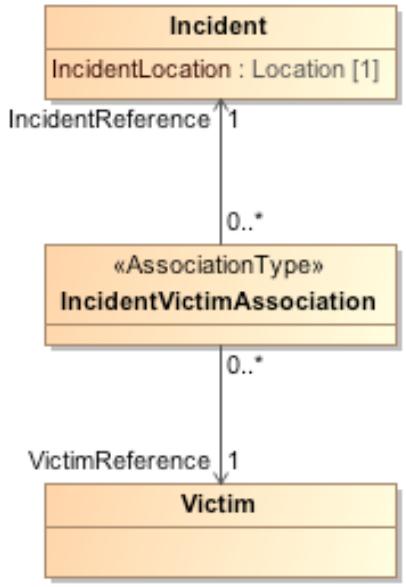


Figure 7-9 Representation of a NIEM association type as a UML class

Figure 7-10 represents the same NIEM association between incidents and victims using a UML association class. (This representation also shows that the suffix “Reference” on the names of reference properties is optional in a NIEM PIM – see Subclause 7.5.1.) The multiplicities of the association ends that a victim can be a victim in any number of incidents and an incident may have any number of victims.

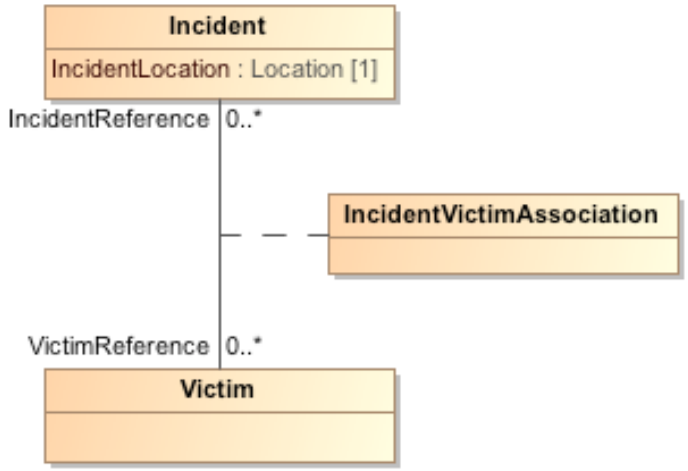


Figure 7-10 Representation of a NIEM association type as a UML association class

PSM Representation

Figure 7-11 shows the PSM representation of the IncidentVictim association type.

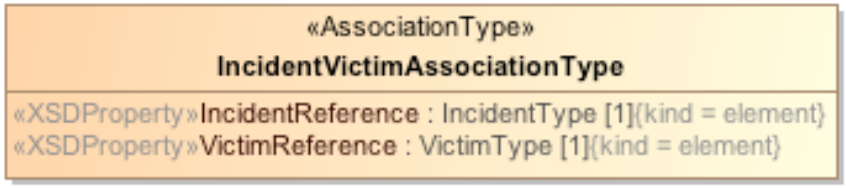


Figure 7-11 Representation of a NIEM association type in a NIEM PSM

XML Schema Representation

The IncidentVictim association type is represented in XML Schema as follows:

```
<xsd:complexType name="IncidentVictimAssociationType">
    <xsd:annotation>
      <xsd:appinfo>
        <i:Base i:name="AssociationType"
              i:namespace="http://niem.gov/niem/niem-core/2.0"/>
      </xsd:appinfo>
      <xsd:documentation>A relationship A data type for a relationship between an
incident and a person who is a victim as a result of the incident.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="nc:AssociationType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="1" ref="nc:IncidentReference"/>
          <xsd:element maxOccurs="1" minOccurs="1" ref="j:VictimReference"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

```
 <xsd:element name="VictimReference" nillable="false" type="s:ReferenceType">
    <xsd:annotation>
      <xsd:appinfo>
        <i:ReferenceTarget i:name="VictimType"
             i:namespace="http://niem.gov/niem/domains/jxdm/4.1"/>
      </xsd:appinfo>
      <xsd:documentation>A Details about a person, organization, or other entity who
suffers injury, loss, or death as a result of an incident.</xsd:documentation>
    </xsd:annotation>
  </xsd:element>

  <xsd:element name="IncidentReference" nillable="false" type="s:ReferenceType">
    <xsd:annotation>
      <xsd:appinfo>
        <i:ReferenceTarget i:name="IncidentType"
             i:namespace="http://niem.gov/niem/niem-core/2.0"/>
      </xsd:appinfo>
      <xsd:documentation>An occurrence or an event that may require a
response.</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
```

2.3.5 Metadata Types

2.3.5.1 Background

Within NIEM, *metadata* is defined as “data about data.” This may include information such as the security of a piece of data or the source of the data. These pieces of metadata may be composed into a metadata type. The types of data to which metadata may be applied may be constrained. A *metadata type* describes data about data, that is, information that is not descriptive of objects and their relationships, but is descriptive of the data itself. It is useful to provide a general mechanism for data about data. This provides required flexibility to precisely represent information. [NIEM-NDR 7.4.4]

2.3.5.2 Representation

Common

A metadata type is represented as a UML class with the «MetadataType» stereotype applied. A «MetadataType» class may be the client of a usage dependency stereotyped as «MetadataApplication» whose supplier is another class. This models the restriction of the application of the metadata to NIEM objects represented as instances of the supplier class. A «MetadataType» class with no «MetadataApplication» dependency represents metadata that may be applied to any NIEM object.

PIM

As for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of a «MetadataType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.1.

PSM

A «MetadataType» class represents a NIEM metadata type implemented in XML schema as a complex type definition with complex content. If the «MetadataType» class is the client of a «MetadataApplication» usage dependency, this is implemented in XML Schema as application information.

2.3.5.3 Mapping Summary

PIM to PSM Mapping

- A class in a PIM with the «MetadataType» stereotype applied shall map to a corresponding class in the PSM with the «MetadataType» stereotype applied.
- If a class in a PIM has the «MetadataType» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM class name ends in "MetadataType", then the NIEM name shall be the same as the PIM class name.
  - If the PIM class name ends in "Metadata", then the NIEM name shall be the PIM class name with "Type" appended.
  - Otherwise, the NIEM name shall be the PIM class name with "MetadataType" appended.
- A usage dependency in a PIM with the «MetadataApplication» stereotype applied shall map to a corresponding usage dependency in the PSM with the «MetadataApplication» stereotype applied, with corresponding client and supplier classes mapped from the PIM.

PSM to XML Schema Mapping

- A class in a PSM with the «MetadataType» stereotype applied shall map to a complex type definition mapped with complex content and:
  - The properties of the class shall map to corresponding property references (XSD attribute uses and element particles) in the complex content of the complex type definition mapped from the class.
  - If the class is not the specific classifier in a generalization, then the complex type definition mapped from the class shall be an extension with the base type being `s:MetadataType`.
- If a «MetadataType» class in a PSM is the client of a «MetadataApplication» usage dependency, then the complex type mapped from the supplier of the dependency shall be referenced in the `xsd:complexType/xsd:annotation/ xsd:appInfo/i:AppliesTo` element for the complex type definition mapped from the «MetadataType» class.

2.3.5.4 Examples

PIM Representation

Figure 7-12 shows a class that represents a metadata type. Since the class has no «MetadataApplication» dependency, the metadata modeled by the class can be applied to any NIEM object. The only difference in the PSM would be the stereotypes on property.

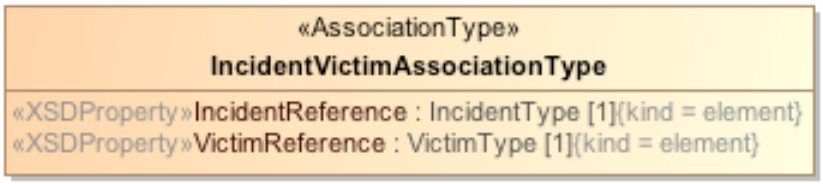


Figure 7-12 Representation of a metadata type as a UML class in a PIM

Figure 7-13 shows a «MetadataType» class with a «MetadataApplication» dependency. In this case the metadata modeled by the class only applies to NIEM objects that are instances of the type identified by the dependency.

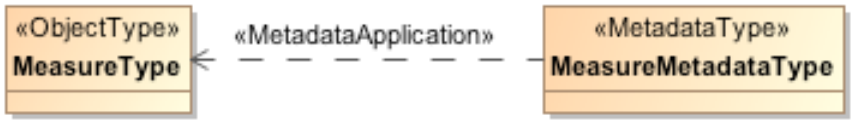


Figure 7-13 Representation of a metadata application constraint as a UML dependency in a PSM or PIM

XML Schema Representation

The MeasureMetadataType modeled in Figure 7-13 is represented in XML Schema as follows:

```
<xsd:complexType name="MeasureMetadataType">
    <xsd:annotation>
        <xsd:documentation>
            A data type for metadata about a measurement.
        </xsd:documentation>
        <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
                    i:name="MetadataType"/>
            <i:AppliesTo i:name="MeasureType"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="s:MetadataType">
            <xsd:sequence>
                ...
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

2.3.6 Augmentation Types

2.3.6.1 Background

An *augmentation type* is a complex type that provides a reusable block of data that may be added to object types or association types. [NIEM-NDR 7.4.5]

2.3.6.2 Representation

Common

An augmentation type is represented as a UML class with the «AugmentationType» stereotype applied.

A UML property with an «AugmentationType» class as its type models an *augmentation* by the data represented by the augmentation type. Such an augmentation may be the client of one or more usage dependencies stereotyped as «AugmentationApplication» whose supplier is an «ObjectType» or «AssociationType» class, known as an *applicable* type. This restricts the NIEM objects that may include the augmentation property to instances of the applicable type. Properties without an «AugmentationApplication» dependency represent augmentations that may be applied to any NIEM object.

NOTE. If an augmentation property with an «AugmentationApplication» dependency is included directly in an «ObjectType» or «AssociationType» class, then the augmented class must be a direct or indirect subclass of the supplier of the dependency. This is not the case if the augmentation property is in a «PropertyHolder» class, however (see Subclause 7.5.2). In this case, any class with a property defined by reference to the augmentation property declaration has a similar subclass restriction.

PIM

An augmentation type is represented as a UML class with the «AugmentationType» stereotype applied or as UML class owning a generalization marked with the «Augments» stereotype applied.

NOTE. As for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of an «AugmentationType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.1.

In a PIM, augmentation may also be modeled using a generalization, with the Augmentation class (optionally stereotyped as «AugmentationType») as the general class and the augmented class as the special class. UML allows a class to have multiple generalizations. In NIEM-UML, a class must have at most one generalization (excluding «RolePlayedBy» generalizations) that is *not* to an «AugmentationType» class; otherwise all generalizations must be to «AugmentationType» classes. The specialized class is considered to be augmented by data corresponding to the inherited properties from each of the «AugmentationType» classes.

An augmentation application restriction may also be alternatively represented in a PIM using a generalization with the «Augments» stereotype applied, where the «AugmentationType» class is the special class and the applicable type is represented by the general class. An «AugmentationType» class shall have at most one «Augments» generalization. Typing a property by a class with an «Augments» generalization is equivalent to modeling an «AugmentationApplication» Usage to the class representing the relevant applicable type.

PSM

An «AugmentationType» class represents a NIEM augmentation type that is implemented in XML Schema as a complex type definition with complex content.

NOTE. XML Schema does not allow a type to extend more than one other type, so an approach to augmentation equivalent to using multiple generalizations in UML is not possible. This is why the representation of augmentation using generalization is not allowed in a NIEM-PSM and why multiple generalization other than as an alternative notation for augmentation (and roles) is not allowed in a NIEM-PIM.

2.3.6.3 Mapping Summary

PIM Representation Mapping

- A class with one or more generalizations to classes stereotyped «AugmentationType» shall be considered equivalent to an otherwise identical class with each generalization replaced by a property such that:
  - The name of the property is the same as the name of the «AugmentationType» class.
  - The type of the property is the «AugmentationType» class.
  - The multiplicity of the property is 1..1.
  - If the generalization had the «ReferenceName» stereotype applied, then the property has the «ReferenceName» stereotype applied, with the same value for the NIEMName attribute.
- A class in a PIM with an «Augments» generalization to a class representing an applicable type shall be considered equivalent to an otherwise identical class without the generalization but with the stereotype «AugmentationType» applied, such that any property with the class as its type (including properties defined implicitly per the alternative representation equivalence above) has an «AugmentationApplication» usage to the applicable type.

PIM to PSM Mapping

- A class in a PIM with the stereotype «AugmentationType» applied shall map to a corresponding class in the PSM with the stereotype «AugmentationType» applied.
- If a class in a PIM has the «AugmentationType» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM class name ends in “AugmentationType”, then the NIEM name shall be the same as the PIM class name.
  - If the PIM class name ends in “Augmentation”, then the NIEM name shall be the PIM class name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM class name with “AugmentationType” appended.
- A usage dependency in a PIM with the stereotype «AugmentationApplication» applied shall map to a corresponding usage dependency in the PSM with the stereotype «AugmenrationApplication» applied, with corresponding client and supplier elements mapped from the PIM.

PSM to XML Schema Mapping

- A class in a PSM with the «AugmentationType» stereotype applied shall map to a complex type definition mapped with complex content and:
  - The properties of the class shall map to corresponding property references (XSD attribute uses and element particles) in the complex content of the complex type definition mapped from the class.
  - If the class is not the specific classifier in a generalization, then the complex type definition mapped from the class shall be an extension with the base type being `s:AugmentationType`.
- If a property in a PSM is the client of a «AugmentationApplication» usage dependency, then the complex type mapped from the supplier of the dependency shall be referenced in the `xsd:element/xsd:annotation/xsd:appInfo/i:AppliesTo` element for the element declaration mapped from the «MetadataType» class (see Subclause 7.3.5.2).

2.3.6.4 Examples

PIM Representation

Figure 7-14 shows an augmentation type represented as a UML class with the «AugmentationType» stereotype.

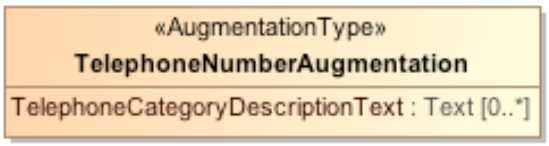


Figure 7-14 Representation of an augmentation type as a UML class in a PIM

Figure 7-15 shows the definition of an augmentation property using the «AugmentationType» class shown in Figure 7-14. It also models that this augmentation is restricted to apply only to instances of the TelephoneNumber class by using an «AugmentationApplication» dependency (Note that PSM property stereotypes are not shown).

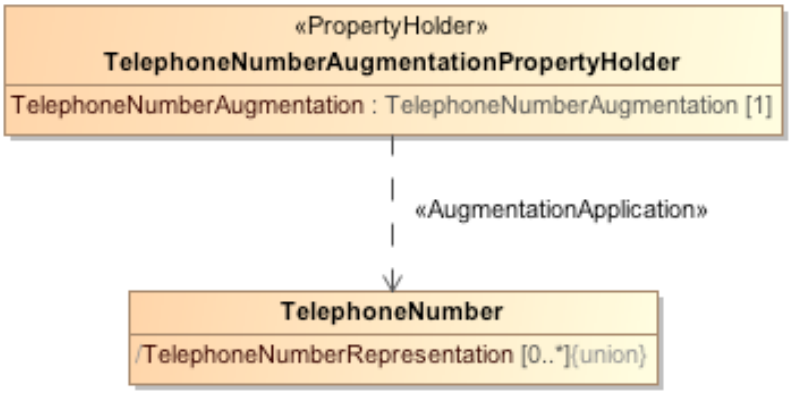


Figure 7-15 Representation of an augmentation property in a PIM or PSM

Figure 7-16 shows an alternative representation of augmentation by the class shown in Figure 7-14 using generalization. It also models the general restriction of application of the augmentation type to instances of the Telephone class by using an «Augments» generalization.

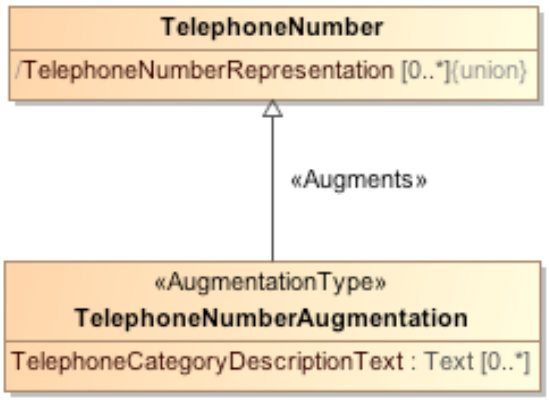


Figure 7-16 Representation of augmentation using generalization in a PIM

XML Schema Representation

The definition of the TelephoneNumberAugmentation type shown in Figure 7-15 is represented in XML schema as follows:

```
<xsd:complexType name="TelephoneNumberAugmentationType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="AugmentationType"
          i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>Supplements telephone numbers</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:AugmentationType">
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
            ref="tns:TelephoneCategoryDescriptionText"/>
      </xsd:sequence>
```

```
    </xsd:extension>
  </xsd:complexContent>
  ...
</xsd:complexType>
```

Its use, with an application restriction, is represented as follows:

```
<xsd:element name="TelephoneNumberAugmentation" nillable="false"
   substitutionGroup="s:Augmentation" type="tns:TelephoneNumberAugmentationType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:AppliesTo i:name="TelephoneNumberType"
          i:namespace="http://niem.gov/niem/niem-core/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

2.3.7 Adapter Types

2.3.7.1 Background

An *adapter type* is a NIEM object type that adapts external models for use within NIEM. An adapter type creates a new class of object that embodies a single concept composed of external elements. [NIEM-NDR 7.7]

2.3.7.2 Representation

Common

A NIEM model may reference other *external* models that are not defined using NIEM-UML. However, reference to external model elements is restricted to adapter types within NIEM. An adapter type is represented as a UML class with the «AdapterType» stereotype applied. All properties of such a class shall be defined *only* in terms of external model elements. The class shall not be a generalization of any other class. Within a PIM, an «AdapterType» class may be used in the same way as any other class representing a NIEM complex type.

Unlike any other NIEM type, an «AdapterType» class may have properties with a type that is defined outside of a «Namespace» package marked with isConformant=true and may have properties which have «Reference» realizations to elements defined outside of a «Namespace» package marked as isConformant=true.

PIM

As for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of an «AdapterType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.1.

PSM

An «AdapterType» class represents a NIEM adapter type that is implemented in XML Schema as a complex type definition with complex content. References to external model elements in the definition of the properties of an «AdapterType» class are implemented as references to external schema components from the content of the complex type definition represented by the class.

NOTE. In order for the PSM to be properly mapped to an XML schema, any external model referenced by an adapter type in the PIM must have a corresponding XML schema representation.

2.3.7.3 Mapping Summary

PIM to PSM Mapping

- A class in a PIM with the «AdapterType» stereotype applied shall map to a corresponding class in the PSM with the «AdapterType» stereotype applied.
- If a class in a PIM has the «AdapterType» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM class name ends in “AdapterType”, then the NIEM name shall be the same as the PIM class name.
  - If the PIM class name ends in “Adapter”, then the NIEM name shall be the PIM class name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM class name with “AdapterType” appended.

PSM to XML Schema Mapping

- A class in a PSM with the «AdapterType» stereotype applied shall be mapped the same way as for an «ObjectType» class (see Subclause 7.3.2.3), except that the complex type definition mapped from the class has a `xsd:complexType/xsd:annotation/xsd:appInfo/i:ExternalAdapterTypeIndicator` element with value `true`.

2.3.7.4 Example

PSM Representation

Figure 7-17 shows the PSM representation of the AlertAdapterType class.

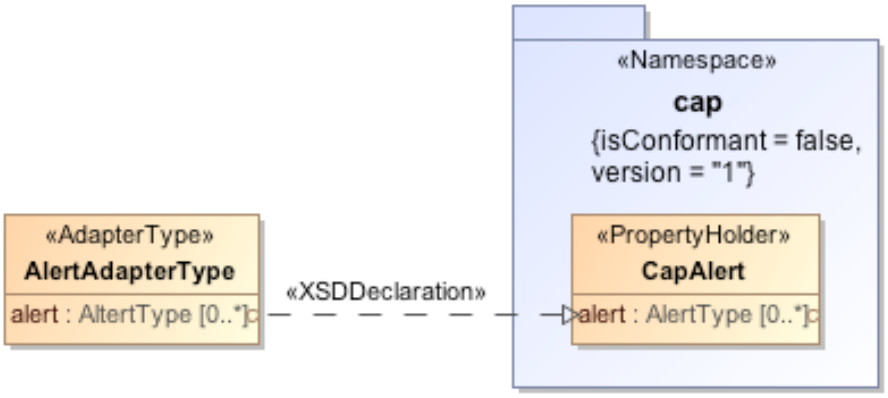


Figure 7-17 Representation of an adapter type as a UML class

XML Schema Representation

The AlertAdapterType modeled in Figure 7-17 is represented in XML schema as follows:

```
<xsd:complexType name="AlertAdapterType">
    <xsd:annotation>
        <xsd:documentation>
            A data type for a simple but general format for exchanging
            effective warning messages based on best practices identified
            in academic research and real-world experience.
        </xsd:documentation>
```

```
        <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
                    i:name="Object"/>
            <i:ExternalAdapterTypeIndicator>
                true
            </i:ExternalAdapterTypeIndicator>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
            <xsd:sequence>
                <xsd:element ref="cap:alert"
                             minOccurs="0"
                             maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

2.4 Modeling Simple Types

2.4.1 Simple Types

2.4.1.1 Background

A *simple type* defines a set of values (its *value space*) and a set of literals used to denote those. (Adapted from the definition of *datatype* in [XMLSchemaDatatypes 2.1].)

2.4.1.2 Representation

Common

A simple type is represented as a UML data type. There are two basic kinds of simple type, represented as primitive types and code types (Enumerations). Simple types can also be combined in a limited fashion into two kinds of structures: unions and lists. Table 7-7 summarizes the representation of the various kinds of simple types, as detailed in subsequent subclauses.

Table 7-7 Simple Type Representation

Simple Type	Representation
Primitive Type	Primitive Type (See Subclause 7.4.2)
Code Type	Enumeration (See Subclause 7.4.3)
Union	Data type with «Union» stereotype (See Subclause 7.4.4)
List	Data type with «List» stereotype (See Subclause 7.4.5)

A simple type may also be defined as having a value space that is a restriction of the value space of another simple type. This is represented by a UML data type that is a client of a «Restriction» realization to another UML data type representing the simple type being restricted. The restricted type may then have the «ValueRestriction» applied, the

attributes of which may be used to specify various restriction *facets* (as described in Subclause 8.2.18). Note that not all facets are applicable to all kinds of simple type.

PIM

Data types representing simple types are generally used in a PIM as the types of properties of classes representing complex types.

In a PIM, rather than using a «Restriction» realization, a data type that has the «ValueRestriction» stereotype applied may, equivalently, have a generalization relationship to the UML data type representing the simple type being restricted. A data type in a PIM that is *not* stereotyped as a «ValueRestriction» may still be the special type in a generalization. However, this is actually mapped to the PSM as a complex type. If the general type is a pre-defined primitive type or a «ValueRestriction» data type, then this complex type has simple content (see Subclause 7.3.2.2), otherwise it has complex content. Such a specialized data type may not be the general type for any «ValueRestriction» data type.

Every data type must be documented. If the data type has only one owned comment, that is considered to provide the required documentation. Otherwise, the data type must have exactly one owned comment with the stereotype «Documentation» applied that provides the required documentation.

PSM

A data type in a PSM is implemented in XML Schema as a simple type definition. The variety of the simple type definition may be atomic, union or list, depending on whether the data type represents a primitive type, code type, union or list.

Generalization is not used with data types in a PSM.

A data type in a PSM that is the client of a «Restriction» realization may also have the «XSDRepresentationRestriction» stereotype applied. This models the restriction on the representation of the literals denoting values of the data type in an XML schema. Specifically, the whiteSpace attribute of «XSDRepresentationRestriction» is implemented as the xsd:whiteSpace element in the simple type definition, with possible values “collapse”, “preserve” and “replace”.

A data type in a PSM must have an owned comment with the «Documentation» stereotype applied, the body of which becomes the content of the documentation element in the simple type definition.

2.4.1.3 Mapping Summary

PIM to PSM Mapping

- A data type in a PIM shall map to a corresponding data type in the PSM (except in the case of a primitive type that is the special type in an generalization – see Subclause 7.4.2.3).
- A data type in a PIM that is the client of a «Restriction» realization shall map to a data type of the same kind in the PSM, with a «Restriction» realization to the data type mapped from the supplier data type in the PIM.
- A specialized data type in a PIM with the «ValueRestriction» stereotype applied shall map to a data type of the same kind in the PSM with the «ValueRestriction» stereotype applied, with the same values for the stereotype attributes, and a «Restriction» realization to the type mapped from the general data type in the PIM.
- A specialized data type in a PIM without the «ValueRestriction» stereotype applied shall map to a class in the PSM with the «ObjectType» stereotype applied.
    - If the general data type in the PIM is itself a specialization that is not a «ValueRestriction», then the «ObjectType» class shall be the special type in a generalization whose general type is the data type mapped from the general type in the PIM.
    - Otherwise, the «ObjectType» class shall be the client of a realization stereotyped «XSDSimpleContent» for which the supplier is the type mapped from the general data type in the PIM.

- If a data type in a PIM has exactly one owned comment, then the corresponding PSM data type shall have an owned comment with the «Documentation» stereotype applied and the same body as the PIM data type comment. Otherwise, the PSM data type shall have an owned comment with the «Documentation» stereotype applied and the same body as the «Documentation» comment owned by the PIM data type. The comment body is adjusted to conform to NIEM conventions.

PSM to XML Schema Mapping

- A data type in a PSM shall map to a corresponding simple type definition with the `xsd:simpleType/@name` given by the data type name.
- If a data type in a PSM is the client of a realization stereotyped as «Restriction», then it shall map to a simple type definition that is a restriction whose base type is the supplier type of the realization. If the data type has the «ValueRestriction» stereotype applied, then the attribute values of the stereotype shall map to corresponding restriction facets.
- If a data type in a PSM has the «XSDRepresentationRestriction» stereotype applied, then the simple type definition mapped from the data type shall include a xsd:restriction/xsd:whiteSpace element with a value given by the value of the whiteSpace attribute of the «XSDRepresentationRestriction» stereotype.
- The «Documentation» comment owned by a data type in the PSM shall map to the documentation for the XML simple type definition mapped from the class, with the body of the comment providing the `xsd:simpleType/xsd:annotation/xsd:documentation` for the simple type definition.

2.4.2 Primitive Types

2.4.2.1 Background

A *primitive type* is a simple type defined in terms of a predefined set of atomic values. An *atomic value* is an elementary value, not constructed from simpler values by any user-accessible means defined by this specification. (Adapted from [XMLSchemaDatatypes].)

2.4.2.2 Representation

Common

The NIEM Primitive Type Library (see Annex C) defines a predefined set of UML primitive types to be used in NIEM-UML models. To insure integrity and consistency of the type system used at the PIM level with the generation of NIEM compliant schema, the primitive types in this library are based on XML schema primitive types [XMLSchemaDatatypes].

A NIEM-UML model may also define new primitive types by specializing the predefined primitive types from the Primitive Type Library (the NIEM Core model provides a set of such specialized primitive types ready-made – see Annex C). All primitive types used in a NIEM-UML model shall be either a predefined primitive type from the Primitive Type Library or a primitive type that is a direct or indirect specialization of a predefined primitive type.

PIM

A specialized UML primitive type in a PIM to which the «ValueRestriction» stereotype is applied defines a new primitive type. However, a specialized UML primitive type without the stereotype application is actually mapped to the PSM as a complex type (as specified for data types in general in Subclause 7.4.1.2).

PSM

A primitive type in a PSM (other than a predefined primitive type from the Primitive Type Library) must be the client in a «Restriction» realization with another primitive type. It is implemented in XML schema as an atomic simple type definition with a base type given by the type represented by its generalization. If the primitive type has the «ValueRestriction« stereotype applied, the attributes of the stereotype are implemented as restriction facets.

2.4.2.3 Mapping Summary

PIM to PSM Mapping

- A reference to a primitive type from the Primitive Type Library in a PIM shall map to a reference to the same primitive type in the PSM.
- If a primitive type in a PIM does not have the «ReferenceName» stereotype applied, then its NIEM name is determined as follows:
  - If the PIM primitive type name ends in “SimpleType”, then the NIEM name shall be the PIM primitive type name.
  - If the PIM primitive type name ends in “Simple”, then the NIEM name shall be the PIM primitive type name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM primitive type name with “SimpleType” appended.

PSM to XML Schema Mapping

- A primitive type in a PSM shall map to an atomic simple type definition with a base type given by the simple type mapped from the supplier type of the «Restriction» realization in which the primitive type is the client type.

2.4.2.4 Examples

PIM Representation

Figure 7-18 shows a Text primitive type defined as a specialization of the String primitive type from the Primitive Type Library, which is then further specialized by the ProperNameText type.

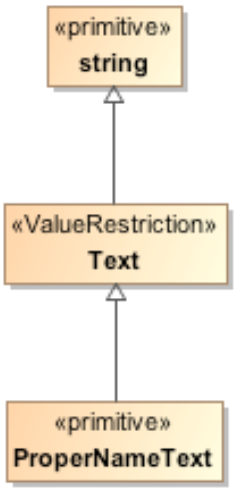


Figure 7-18 Representation of primitive types in a PIM

The Text data type in Figure 7-18 is stereotyped as a «ValueRestriction», but it does not have any restriction facets specified. Figure 7-19 shows an example of a primitive type defined as a «ValueRestriction» with restriction facets.

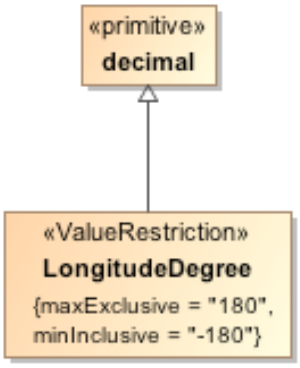


Figure 7-19 Representation of a primitive type with a value restriction in a PIM

PSM Representation

Figure 7-20 shows the PSM representation of the primitive types modeled in Figure 7-18. A primitive types in a PSM must be stereotyped as a «ValueRestriction», so the ProperNameText type becomes an «ObjectType» class in the PSM.

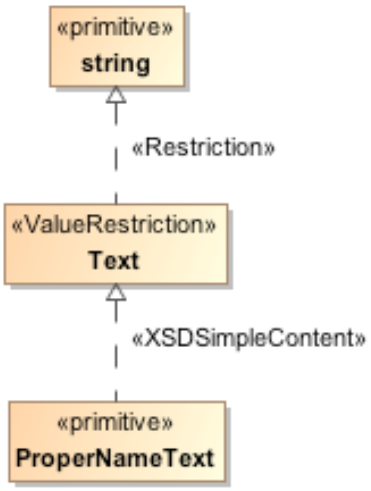


Figure 7-20 Representation of primitive types in a PSM

Figure 7-21shows the PSM representation of the primitive type shown in Figure 7-19, which uses a «Restriction» realization instead of a generalization.

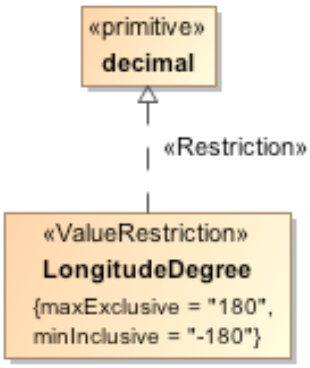


Figure 7-21 Representation of a primitive type with a value restriction in a PSM

XML Schema Representation

The primitive types shown in Figure 7-20 are represented in XML schema as follows:

```
<xsd:simpleType name="TextSimpleType">
    <xsd:annotation>
      <xsd:appinfo>
        <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
      </xsd:appinfo>
      <xsd:documentation>A data type for text</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base=" xsd:string"/>
</xsd:simpleType>
<xsd:complexType name="ProperNameTextSimpleType">
    <xsd:annotation>
      <xsd:appinfo>
        <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
      </xsd:appinfo>
      <xsd:documentation>A data type for proper name text</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:extension base="tns:TextSimpleType">
            <xsd:attributeGroup ref="s:SimpleObjectAttributeGroup"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

The primitive type shown in Figure 7-21 is represented in XML schema as:

```
<xsd:simpleType name="LongitudeDegreeSimpleType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A data type for longitude degrees</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="-180"/>
    <xsd:maxExclusive value="180"/>
  </xsd:restriction>
</xsd:simpleType>
```

2.4.3 Code Types

2.4.3.1 Background

A *code type* is a simple type that represents a list of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals. [NIEM-NDR 9.12.3]

2.4.3.2 Representation

Common

A code type is represented as a UML enumeration. Each code value is one enumeration literal of the enumeration.

The code values are considered to be a restriction of the value space of the *base type* of the enumeration. The base type may be explicitly modeled as the supplier of a «Restriction» realization in which the numeration is the client.

PIM

An enumeration in a PIM need not be the client of a «Restriction» realization. By default, the base type of the enumeration is taken to be the XSD token primitive type.

A specialized enumeration to which the «ValueRestriction» stereotype is applied also defines a new code type as a restriction of the code type defined by the general enumeration. However, a specialized enumeration without the stereotype application is actually mapped to the PSM as a complex type (as specified for data types in general in Subclause 7.4.1.2).

PSM

The base type of an enumeration in a PSM is must be explicitly identified using a «Restriction» realization from the enumeration to the base type. Such an enumeration represents a NIEM code type that is implemented in XML schema as atomic simple type definition that is a restriction of the identified base type using multiple `xsd:enumeration` facets.

2.4.3.3 Mapping Summary

PIM Representation Mapping

- An enumeration in a PIM that is neither a specialization nor the client of a «Restriction» realization shall be considered equivalent to an enumeration with a «Restriction» realization to the token primitive type from the XML Primitive Type Library.

PIM to PSM Mapping

- An enumeration in a PIM shall map to a corresponding enumeration in the PSM, with corresponding enumeration literals.
- If an enumeration in a PIM does not have the «ReferenceName» stereotype applied, then its NIEM name is determined as follows:
  - If the PIM enumeration name ends in “CodeSimpleType” or “CodeType”, then the NIEM name shall be the PIM enumeration name.
  - If the PIM enumeration name ends in “CodeSimple”, then the NIEM name shall be the PIM enumeration name with “Type” appended.
  - If the PIM enumeration name ends in “Code”, then the NIEM name shall be the PIM enumeration name with “SimpleType” appended.
  - Otherwise, the NIEM name shall be the PIM enumeration name with “CodeSimpleType” appended.

PSM to XML Schema Mapping

- An enumeration in a PSM shall map to an atomic simple type definition. The base type of the simple type definition is the type mapped from the supplier data type of the «Restriction» realization in which the enumeration is the client.

- Each enumeration literal of the enumeration shall map to an enumeration facet of the simple type definition mapped from the enumeration, whose value is given by the enumeration literal name.

2.4.3.4 Example

PIM Representation

Figure 7-22 shows the definition of the SupervisionLevelCode type as a UML enumeration.

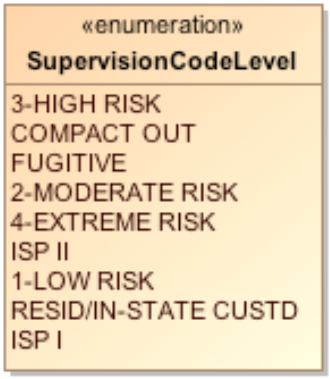


Figure 7-22 A code type represented as a UML enumeration in a PIM

PSM Representation

Figure 7-23 shows the PSM representation of the code type shown in Figure 7-22 , with an explicit «Restriction» realization to the XSD token primitive type.

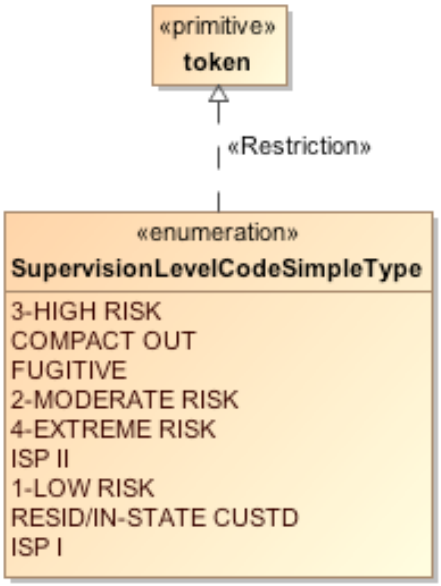


Figure 7-23 A code type represented as a restriction in a PSM

XML Schema Representation

The XML Schema representation for the code type shown in Figure 7-23 is:

```
<xsd:simpleType name="SupervisionLevelCodeSimpleType">
```

```
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A data type for supervision level codes</xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="3-HIGH RISK"/>
    <xsd:enumeration value="COMPACT OUT"/>
    <xsd:enumeration value="FUGITIVE"/>
    <xsd:enumeration value="2-MODERATE RISK"/>
    <xsd:enumeration value="4-EXTREME RISK"/>
    <xsd:enumeration value="ISP II"/>
    <xsd:enumeration value="1-LOW RISK"/>
    <xsd:enumeration value="RESID/IN-STATE CUSTD"/>
    <xsd:enumeration value="ISP I"/>
  </xsd:restriction>
</xsd:simpleType>
```

2.4.4 Unions

2.4.4.1 Background

A *union* is a simple type whose values are the union of the values of one or more other simple types, which are the *member types* of the union. (Adapted from [XMLSchemaDatatypes].)

2.4.4.2 Representation

Common

A union is represented as a UML data type (that is neither a primitive type nor an enumeration) with the stereotype «Union» applied. The member types of the union are represented as data types that are suppliers of UML usage dependencies with the union data type as the supplier and the stereotype «UnionOf» applied. A «Union» datatype shall not have any properties.

A «Union» data type may not be a specialization of another data type. However, a data type with the «ValueRestriction» stereotype applied may be the specialization of a «Union» type.

PIM

There is no further representation for a PIM.

PSM

A «Union» data type is implemented as a union simple type definition. The member types of the union simple type definition are the types represented by the UML data types that realize the «Union» data type.

2.4.4.3 Mapping Summary

PIM to PSM Mapping

- A data type in a PIM with the «Union» stereotype applied shall map to a corresponding data type in the PSM with the «Union» stereotype applied.

- A usage dependency with the «UnionOf» stereotype applied shall map to a corresponding dependency in the PSM between corresponding data types mapped from the PIM.
- If a data type in a PIM has the «Union» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM data type name ends in “SimpleType”, then the NIEM name shall be the PIM data type name.
  - If the PIM data type name ends in “Simple”, then the NIEM name shall be the PIM data type name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM data type name with “SimpleType” appended.

PSM to XML Schema Mapping

- A data type in a PSM with the «Union» stereotype applied shall map to a corresponding union simple type definition.
- For each usage dependency with the «UnionOf» stereotype applied, the type represented by the supplier of the dependency shall appear in the `xsd:union/@xsd:memberTypes` list for the simple type definition mapped from the «Union» type that is the client of the dependency.

2.4.4.4 Example

PIM Representation

Figure 7-24 illustrates a FrictionRidgePositionCode union type from the NIEM biometrics domain. Note that the code values associated with the code types PlantarPositionCodeSimpleType, FingerPositionCodeSimpleType, PalmPositionCodeSimpleType, and UnknownPositionCodeSimpleType have been omitted.

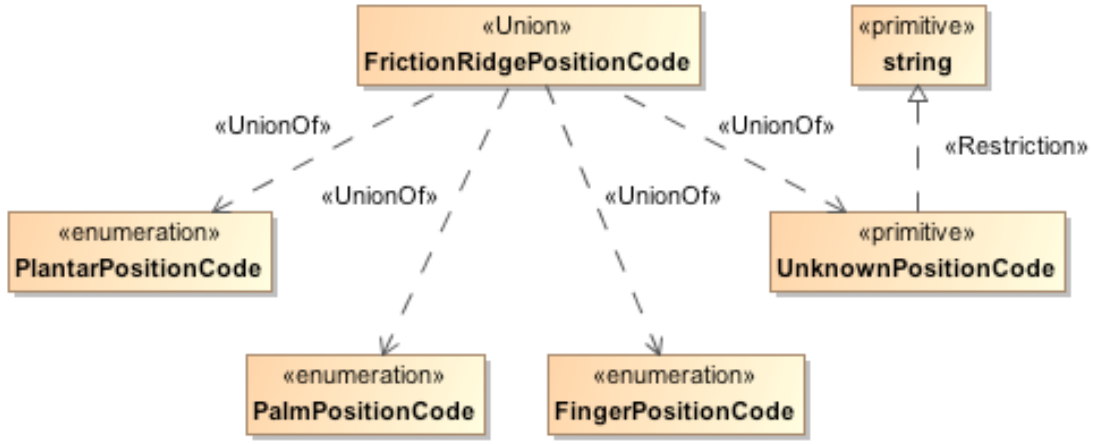


Figure 7-24 Representation of a union as a UML data type

PSM Representation

The PSM representation for the types shown in Figure 7-24 is the same as in the PIM except that the names of the types are proper NIEM names ending in “CodeSimpleType”.

XML Schema Representation

The «Union» data type shown in Figure 7-24 is implemented in XML Schema as follows:

```
<xsd:simpleType name="FrictionRidgePositionCodeSimpleType">
     <xsd:annotation>
```

```
         <xsd:documentation>
             A data type for a friction ridge image position
         </xsd:documentation>
         <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
                    i:name="Object"/>
         </xsd:appinfo>
     </xsd:annotation>
     <xsd:union memberTypes="biom:FingerPositionCodeSimpleType
          biom:PalmPositionCodeSimpleType biom:PlantarPositionCodeSimpleType
          biom:UnknownPositionCodeSimpleType"/>
</xsd:simpleType>
```

2.4.5 Lists

2.4.5.1 Background

A *list* is a simple type having values each of which consists of a finite-length (possibly empty) sequence of atomic values. The values in a list are drawn from some atomic simple type (or from a union of atomic simple types), which is the *item type* of the list. (Adapted from {XMLSchemaDatatypes].)

2.4.5.2 Representation

Common

A list is represented as a UML data type (that is neither a primitive type nor an enumeration) with the stereotype «List» applied. The data type must have a single property with the multiplicity 0..* and a type that represents the item type of the list. The name of the property is arbitrary.

The item type of a list is required to be an atomic type, that is, a type whose values are atomic values. Any primitive type or code list is an atomic type, as is any union of atomic types.

A «List» data type may not be a specialization of another data type. However, a data type with the «ValueRestriction» stereotype applied may be the specialization of a «List» type.

PIM

There is no further representation for a PIM.

PSM

A «List» data type is implemented as a list simple type definition. The item type of the list simple type definition is the type represented by the type of the single property of the «List» data type.

If the «List» data type is the special type in a generalization, then the type represented by the general type in that generalization is the base type of the simple type definition for the «Union» data type. If the generalization is not stereotyped, then simple type definition is an extension. If the generalization is stereotyped as a «ValueRestriction», then the simple type definition is a restriction and the attribute values of the «ValueRestriction» stereotype are implemented as restriction facets.

2.4.5.3 Mapping Summary

PIM to PSM Mapping

- A data type in a PIM with the «List» stereotype applied shall map to a corresponding data type in the PSM with the «List» stereotype applied, with a corresponding property mapped from the single property of the data type in the PIM.
- If a data type in a PIM has the «List» stereotype applied but not the «ReferenceName» stereotype, then its NIEM name is determined as follows:
  - If the PIM data type name ends in “SimpleType”, then the NIEM name shall be the PIM data type name.
  - If the PIM data type name ends in “Simple”, then the NIEM name shall be the PIM data type name with “Type” appended.
  - Otherwise, the NIEM name shall be the PIM data type name with “SimpleType” appended.

PSM to XML Schema Mapping

- A data type in a PSM with the «List» stereotype applied shall map to a corresponding list simple type definition, with an item type given by the simple type mapped from the type of the single required property of the «List» data type.

2.4.5.4 Example

PIM Representation

Figure 7-25 shows the PIM representation of a simple type that is a list of Boolean values. Note that the required property of the «List» data type is represented using an association (see also Section 7.5.1.2)

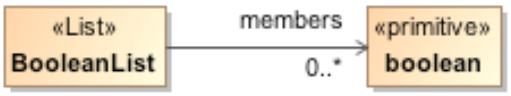


Figure 7-25 Representation of a list in a PIM

PSM Representation

Figure 7-26 shows the PSM representation of the «List» data type shown in Figure 7-25. Note that, in the PSM, the required property of the «List» data type is represented as an attribute.

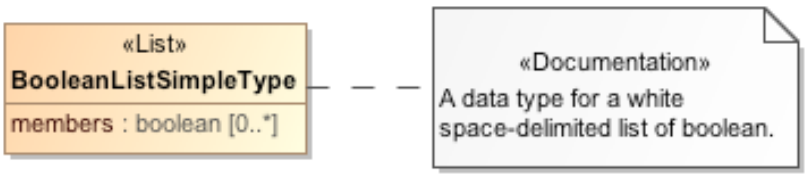


Figure 7-26 Representation of a list in a PSM

XML Schema Representation

The «List» data type shown in Figure 7-26 is implemented in XML Schema as follows:

```
<xsd:simpleType name="BooleanListSimpleType">
    <xsd:annotation>
        <xsd:documentation>
            A data type for a white space-delimited list of boolean.
```

```
        </xsd:documentation>
        <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
                    i:name="Object"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:list itemType="xsd:boolean"/>
</xsd:simpleType>
```

2.5 Modeling Properties

2.5.1 Properties

2.5.1.1 Background

A *property* relates a NIEM object (the *subject*) to another object or to a value (the *object*). Property data describes an object as having a characteristic with a specific value or a particular relationship to another object. [NIEM-NDR 3.2]

2.5.1.2 Representation

Common

A NIEM property is represented as a UML property. The owner of the UML property specifies the type of the subject of the NIEM property, while the type of the UML property itself specifies the type of the object of the NIEM property.

A UML property with aggregation=none represents a NIEM *reference* property while a UML property with aggregation=shared or composite represents a NIEM *content* property. NIEM does not recognize any semantic difference between reference and content properties [NDR 7.6.2] (though their XML Schema representation may differ). A UML property with aggregation=composite, however, carries an additional semantic constraint that any instance may be the value of at most one composite property at any point in time [UML 7.3.3].

PIM

A property may optionally be represented as the end of a UML association. An association end representing a NIEM property is always navigable (since classifier-owned association ends are always navigable in UML). The subject type of the NIEM property is represented by the classifier at the opposite end of the association. A UML association used to represent a NIEM property (or two NIEM properties) may not be an association class.

NOTE. An ordinary UML association does *not* represent a NIEM association type. See Subclause 7.3.4 on the representation of NIEM association types.

While a unidirectional association (i.e., one navigable at only one end) only defines a single NIEM property, UML still provides the ability to model an arbitrary multiplicity on the non-navigable end of the association. This represents an additional constraint on how many instances of the subject type may participate in the NIEM property. This constraint can only be modeled in a NIEM PIM using the UML association notation for a NIEM property.

A bidirectional association (i.e., one navigable at both ends) represents *two* NIEM properties, corresponding to each end, in which the object type of each property is the subject type of the other.

PSM

In a PSM, each UML property owned by a class must may have either the «XSDProperty» or the «XSDAnyProperty» stereotype applied. A property with neither applied is treated as if «XSDProperty» was applied with default values for its attributes.

An «XSDProperty» property represents a NIEM property, which is implemented in XML Schema as either an attribute declaration and use or an element declaration and particle. If the «XSDProperty» attribute kind has the value “attribute”, then the property is implemented as an XML Schema attribute. If the value of kind is “element”, then the property is implemented as an XML Schema element.

If an «XSDProperty» property has kind=attribute, then its multiplicity must be 1..1, its aggregation must not be none and its type must be a data type representing a simple type.

If an «XSDProperty» has kind=element, the multiplicity lower bound for the property gives the value of `minOccurs` for the implemented element particle and the multiplicity upper bound for the property gives the value of `maxOccurs`. The type of the property must not be empty unless the property is a derived union (a UML property without a type that is a derived union represents an *abstract* property – see Subclause 7.5.3). The nillable attribute of the «XSDProperty» stereotype may be used to indicate that the element particle is nillable.

The fixed attribute of the «XSDProperty» stereotype may be used to indicate that the attribute use or element particle must have a certain fixed value.

There are significant differences between the UML representation and XML Schema implementation of a NIEM property. Sections 6.1.6.2 and 6.1.6.3 of [NIEM-NDR], Rule 6-14 and Rule 6-15, require that an attribute or element declaration be a top-level declaration; however, Section 7.3.44 of [UML] requires that a Property be the ownedAttribute of a Classifier. Thus in the UML representation, only one Classifier may reference a Property, while in the XML Schema implementation, more than one type definition may reference the same attribute or element declaration.

To resolve this difference, more than one «XSDProperty» property with the same name contained (directly or indirectly) within the same «Namespace» package (see Subclause 7.2.1) shall have the same attribute or element declaration (and so must all have the same value for kind). All use of the attribute uses or element particles mapped from such properties reference the same attribute or element declaration.

Alternatively, a property declaration may be explicitly modeled separately from property use using a «PropertyHolder» class. This is discussed further in Subclause 7.5.2.

An «XSDAnyProperty» property represents the use of a property that may hold a value of any type, which is implemented in XSD Schema as an `xsd:any` particle. Such a property may not have a type, but also must be a derived union (a UML property without a type that is a derived union represents an *abstract* property – see Subclause 7.5.3). The multiplicity lower and upper bounds of an «XSDAnyProperty» property give the `minOccurs` and `maxOccurs` values, respectively, for the xsd:any particle. If provided, the processContents and valueNamespace attributes of the «XSDAnyProperty» stereotype give the `processContents` and `namespace` values for the `xsd:any` particle.

A «SequenceID» property is mapped to a NIEM `structures:sequenceId` attribute (see [NIEM-NDR 7.6.1]). Such a property must have the name “sequenceId”, the type “integer” and a multiplicity of 1..1.

2.5.1.3 Mapping Summary

PIM Representation Mapping

- A property owned by an association shall be considered equivalent to a property owned by the associated class.
- A UML property that is an association end shall be considered to an otherwise identical UML property that is not an association end.

PIM to PSM Mapping

- A property in a PIM shall map to a corresponding property in the PSM with the same multiplicity and aggregation as the PIM property and with an owner and type (if any) that are the corresponding classifiers mapped from the PIM.

- If a property in a PIM has a type, is owned by a class and is the client of a «References» realization or is marked as a derived union, then the corresponding property in the PSM shall have the «XSDProperty» stereotype applied.
- If a property in a PIM has no type, is owned by a class, but is not marked as a derived union, then the corresponding property in the PSM shall have the «XSDAnyProperty» stereotype applied.
- If a property in a PIM owned by a class does not have the stereotype «ReferenceName» applied and is not the client of a «References» realization, then its NIEM name is determined as follows:
  - If the PIM property has aggregation=none and the property name does not end in "Reference", then the NIEM name shall be the PIM property name with "Reference" appended.
  - Otherwise, the NIEM name shall be the PIM property name.

PSM to XML Schema Mapping

- A property in a PSM with the «XSDProperty» stereotype applied and kind=element, or with no stereotype applied, shall map to XML schema as follows:
  - Unless it is the client of a «References» realization whose supplier is in a «Namespace» package with a *different* target namespace, it shall map to a corresponding element declaration with a name given by the property name. All «XSDProperty» properties with the same name contained within the same «Namespace» package shall map to a *single* such element declaration.
    - If the property has aggregation = none and has a class as its type, then the element declaration shall be for a reference to the complex type mapped from the type of the property. Otherwise, the element declaration shall have the simple or complex type mapped from the type of the property.
  - If the property is owned by a class that does *not* have the «PropertyHolder» stereotype applied, then it shall also map to an element particle within the complex content of the complex type mapped from the owning class, with an `ref` to the element declaration mapped per the above, `nillable` given by the value of the nillable attribute of the «XSDProperty» stereotype and property multiplicity mapped to `minOccurs` and `maxOccurs`. If a value is provided for the fixed attribute of the «XSDProperty» stereotype, then the element particle contains a `fixed` attribute with that value.
- A property in a PSM with the «XSDProperty» stereotype applied and kind=attribute shall map to XML schema as follows:
  - Unless it is the client of a «References» realization whose supplier is in a «Namespace» package with a *different* target namespace, it shall map to a corresponding attribute declaration with the `xsd:attribute/@name` given by .the property name and the `xsd:attribute/@type` given by the corresponding simple type mapped from the property type. All «XSDProperty» properties with the same name contained within the same «Namespace» package shall map to a *single* such attribute declaration.
  - If it is owned by a class that does *not* have the «PropertyHolder» stereotype applied, then it shall also map to an attribute use within the complex content of the complex type mapped from the owning class, with an `xsd:attribute/@ref` to the attribute declaration mapped per the above. If a value is provided for the fixed attribute of the «XSDProperty» stereotype, then the attribute use contains a `fixed` attribute with that value.
- If an «XSDProperty» property has an owned comment with the stereotype «Documentation» applied, then the body of this comment is used for the documentation annotation of the attribute or element declaration mapped from the property.
- A property in a PSM with the «XSDAnyProperty» stereotype applied shall map to an `xsd:any` particle within the complex content of the complex type mapped from the owning class of the property, with the property name mapped to `name`, multiplicity mapped to `minOccurs` and `maxOccurs`, the processContent attribute of the «XSDAnyProperty» stereotype mapped to `processContent` and the valueNamespace attribute mapped to `namespace`.

- A property in a PSM with the «SequenceID» stereotype applied small map to an attribute use with an `xsd:attribute/@ref` to the NIEM `structures:sequenceID` attribute declaration.

2.5.1.4 Example

PIM Representation

Figure 7-27 shows a set of three NIEM properties represented as attributes of a Person class. The complex type represented by this class is thus also modeled as being the subject of these properties.

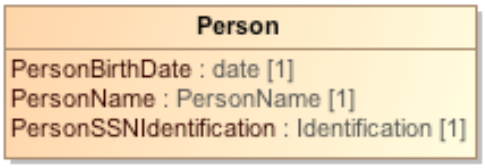


Figure 7-27 Representation of NIEM properties as UML properties in a PIM

Figure 7-28 shows an example the alternative representation of a NIEM property as an association end.

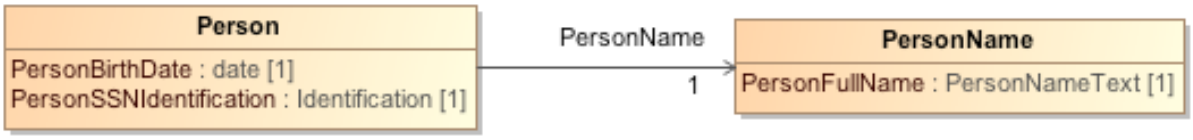


Figure 7-28 Representation of a NIEM property as a UML association end

PSM Representation

Figure 7-29 shows the PSM representation of the class modeled in Figure 7-27, with the «XSDProperty» stereotype applied to all properties.

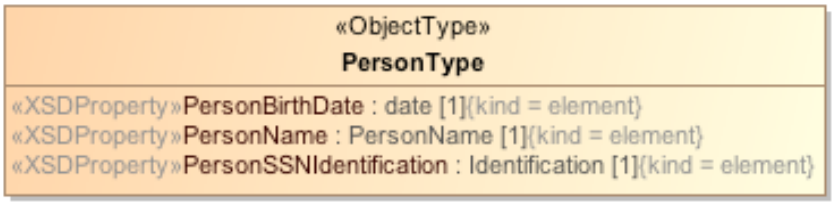


Figure 7-29 Representation of NIEM properties as UML properties in a PSM

XML Schema Representation

The class shown in Figure 7-29 is represented in XML schema as follows:

```
<xsd:complexType name="PersonType">
    <xsd:annotation>
       <xsd:documentation>A data type for a human being.</xsd:documentation>
       <xsd:appinfo>
          <i:Base i:name="Object"
             i:namespace="http://niem.gov/niem/structures/2.0"/>
       </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexContent>
```

```
       <xsd:extension base="s:ComplexObjectType">
          <xsd:sequence>
             <xsd:element maxOccurs="1" minOccurs="1" ref="nc:PersonBirthDate"/>
             <xsd:element maxOccurs="1" minOccurs="1" ref="nc:PersonName"/>
             <xsd:element maxOccurs="1" minOccurs="1"
                 ref="nc:PersonSSNIdentification"/>
          </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>

<xsd:element abstract="false" name="PersonName" nillable="false"
   type="nc:PersonNameType">
      <xsd:annotation>
         <xsd:documentation>A combination of names and/or titles by which a person is
known.</xsd:documentation>
      </xsd:annotation>
 </xsd:element>

<xsd:element abstract="false" name="PersonBirthDate" nillable="false"
   type="nc:DateType">
      <xsd:annotation>
         <xsd:documentation>A date a person was born.</xsd:documentation>
      </xsd:annotation>
</xsd:element>

<xsd:element abstract="false" name="PersonSSNIdentification" nillable="false"
               type="nc:IdentificationType"/>
```

2.5.2 Property Holders and Property References

2.5.2.1 Background

A *property declaration* is the association of the name of a property with the type (object) of the property. A *property reference* is the association of a property declaration with a particular type of subject for the property.

A UML property owned by a class representing a complex type specifies both the subject and object types for the represented NIEM property. A NIEM property may also be declared independently of the definition of any complex type. Such a global property declaration defines the object type of the property but does not restrict its use to a specific type of subject.

2.5.2.2 Representation

Common

Since a UML property cannot be defined outside of a classifier, a global property declaration is still represented as a UML property owned by a class, but that class has the «PropertyHolder» stereotype applied, indicating that its purpose is simply to hold the representations of global property declarations.

The use of a property in the context of a complex type may also be defined *by reference* to a property declaration outside of the definition of the complex type. Such a property reference is represented by a UML realization with the stereotype «References» applied, between two UML properties owned by different classes. A specific property declaration may be referenced at most once within the context of any one complex type.

The client UML property of a «References» realization represents the use, in the context of the complex type represented by the owning class of the property, of the NIEM property declared by the supplier UML property of the realization. The client UML property of the realization must have the same type (or a subclass) as the supplier property and a multiplicity that is consistent with the multiplicity of the supplier property. Multiple properties may be defined by reference to the same property declaration.

A UML property owned by a class representing a complex type that is *not* the client of a «References» realization actually represents both the declaration of a NIEM property and the use of that property in the context of the complex type. Therefore, such a UML property may also be the *supplier* of «References» realizations, in which case the reference is to the implicit property declaration represented by the UML property.

Since all property declarations in NIEM, whether represented explicitly or implicitly in UML, are considered to be "top level", the NIEM names of all UML properties representing such declarations within a single NIEM namespace must have distinct NIEM names (see also Subclause 7.2.1). However, a UML property that is the client of a «References» realization does not represent a property declaration and thus has the same NIEM name as the supplier of the realization.

PIM

It is often the case that more than one property in a class representing a complex type will be defined by reference to property declarations represented by UML properties with the same owner (for example, a «PropertyHolder» class modeling a set of top-level declarations in a namespace). As a convenience notation for this case, a «References» realization may be used between the two *classes*, rather than using multiple realizations between pairs of properties. When one class has a «Reference» realization to another, any UML property in the client class with the same NIEM name as a UML property in the supplier class is considered to be implicitly defined by reference to the property declaration represented by the matching UML property.

Likewise, a «References» realization may be used between packages. This will result in all classes within those packages having «References» realizations based on matching NIEM names (see Subclause 7.6.1).

PSM

A property declaration represented in a PSM may not have the «XSDAnyProperty» stereotype applied. It is implemented as either an attribute or element declaration, depending on the value of the kind attribute of the «XSDProperty» stereotype (or as an element, if no stereotype is applied). A property reference is implemented as an attribute use or element particle referencing the corresponding declaration. If the UML property representing the property declaration is contained in a different «Namespace» package than the UML property representing the property reference, then the implementation of the property reference will refer to a declaration in a different schema.

The «XSDDeclaration» stereotype is a specialization of «References» that may be used in a PSM to denote explicitly that a realization so stereotyped identifies the property declaration referenced by a specific property use. An «XSDProperty» realization must always be between one property and another property or between a property and a «Namespace» package. In the later case, the target namespace of the «Namespace» package is used as the namespace for the property declaration, while the property name is taken from the UML property representing the property use.

2.5.2.3 Mapping Summary

PIM Representation Mapping

- A realization between two packages in a PIM with the stereotype «References» applied shall be considered equivalent to replacing the realization between the packages with multiple «References» realizations between classes with those packages, such that:
    - If a class in the client package of the original realization has the same NIEM name as a class of the supplier package, then there is a realization from the class in the client class to the class in the supplier package.

- A realization between two classes in a PIM with the stereotype «References» applied shall be considered equivalent to replacing the realization between the classes with multiple «References» realizations between properties of the classes, such that:
  - If a property in the client class of the original realization has the same NIEM name as a property of the supplier class, then there is a realization from the property in the client class to the property in the supplier class.

PIM to PSM Mapping

- A class in a PIM with the «PropertyHolder» stereotype applied shall map to a corresponding class in the PSM with «PropertyHolder» stereotype applied.
- A realization between two properties in a PIM with the stereotype «References» applied shall map to a corresponding realization in the PSM with the «References» stereotype applied, between corresponding properties mapped from the PIM.
- A property in a PIM that is the client of a «References» realization with another property as the supplier has the same NIEM name as the supplier property.

PSM to XML Schema Mapping

- A property in a PSM that is owned by a class with the «PropertyHolder» shall be mapped as an attribute or element declaration, as described in Subclause 7.5.1.4. The «PropertyHolder» class will have no representation in the XSD.
- A property in a PSM that is the client of a «References» or «XSDDeclaration» realization whose supplier has a different target namespace shall be mapped as an attribute use or element particle, as described in Subclause 7.5.1.4, but shall *not* be mapped as an attribute or element declaration. The attribute use or element particle shall have its `ref` attribute set to the attribute or element declaration mapped from the supplier of the realization.

2.5.2.4 Example

PIM Representation

Figure 7-30 shows two properties of the Payload class being defined by reference to properties of the same name defined in NIEM Core. The OrganizationAssociation and OrganizationContactInformationAssociation property declarations are modeled as properties of «PropertyHolder» classes, independently of their use in the definition of Payload or any other complex type. (This representation may also be used in a PSM.)

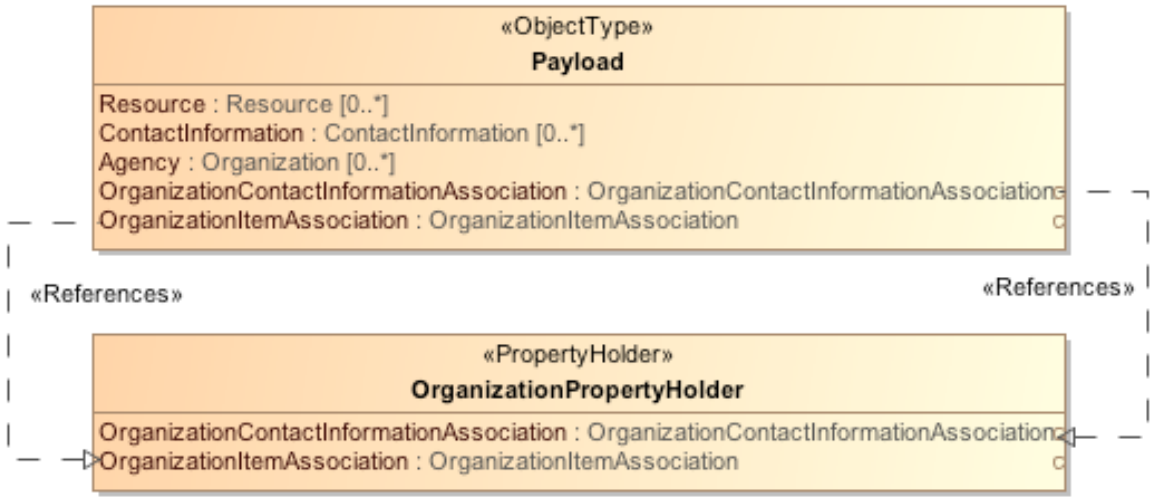


Figure 7-30 Representation of property references using «References» realizations

Figure 7-31 shows an alternative representation of the model shown in Figure 7-30, using a single «Reference» realization between the two classes. Since both of the properties OrganizationAssociation and OrganizationContactInformationAssociation in the Payload match the names of properties of the referenced «PropertyHolder» class, these are both considered to be defined by reference. However, the properties Resource, ContactInformation and Agency are defined in the context of their use in the Payload class.

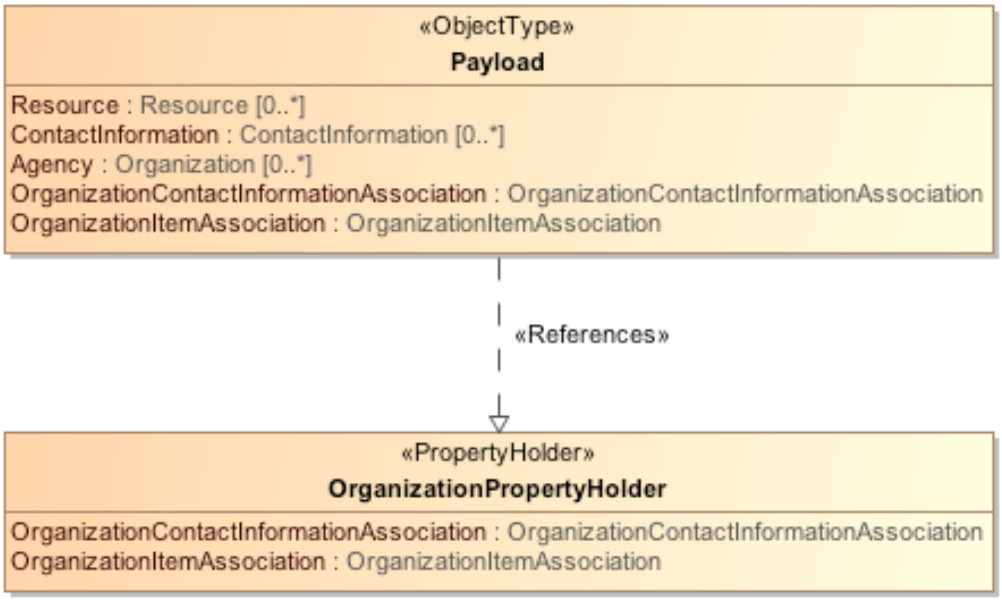


Figure 7-31 Alternative representation using «References» realizations between classes

XML Schema Representation

The property references modeled in Figure 7-30 are represented in XML Schema as follows:

```
<xsd:complexType name="PayloadType">
  <xsd:annotation>
    <xsd:appinfo>
      <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
    </xsd:appinfo>
    <xsd:documentation>A data type for</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="tns:Resource"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="tns:ContactInformation"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="tns:Agency"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="nc:OrganizationContactInformationAssociation"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          ref="nc:OrganizationItemAssociation"/>
      </xsd:sequence>
```

```
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

2.5.3 Substitution Groups

2.5.3.1 Background

One property is potentially *substitutable* for another if either the first property has no type or the type of the second property is a direct or indirect generalization of the type of the first property. The *substitution group* for a property known as the *head* is the set of all properties that are substitutable for it within a certain context. (Adapted from [XMLSchemaStructures 3.3.6.4].)

An *abstract* property is one that cannot be assigned a value itself but can only take values as determined by properties in its substitution group. (Adapted from [XMLSchemaStructures 3.3.1].)

2.5.3.2 Representation

Common

Any UML property owned by a class may represent the head of a substitution group. The context of the substitution group is the «Namespace» package (see Subclause 7.2.1) that directly or indirectly contains the owning class of the head property. Members of the substation group are represented as UML “subset” properties of the head.

A UML property models a member of a substitution group if it is declared to have the head property as a *subsetted property.* The well-formedness rules of UML require that a subsetting property be owned either in the same class or a direct or indirect subclass of any subsetted property (see [UML 7.3.45]). However, a «PropertyHolder» class may be used to define substitution group properties independently of any complex type definition (see Subclause 7.5.2).

An abstract property is represented by a UML property that is marked as a *derived union.* In this case, the collection of values of the property in the context of its substitution group is derived as the strict union of the values of the subsetting properties in that group (see [UML 7.3.45]). If a UML property with no type is used to represent a head property, then it must be marked as a derived union.

PIM

There is no further representation for a PIM.

PSM

A UML property in a PSM that subsets another property must not have the stereotype «XSDProperty» applied with kind=attribute or have the «XSDAnyPropertyt» stereotype applied. It may not subset another an «XSDAnyProperty»..

A UML property in a PSM that is a derived union must have the «XSDProperty» applied with kind=element.

A UML Property that subsets another property will be a member of the substitution group for that property.

2.5.3.3 Mapping Summary

PIM to PSM Mapping

- A property in a PIM that has subsetted properties shall map to a corresponding property in the PSM that subsets the corresponding properties mapped from the subsetted properties in the PIM.
- A property in a PIM that is a derived union shall map to a corresponding property in the PSM that is a derived union.

PSM to XML Schema Mapping

- A property in a PSM that subsets another property maps to an element declaration with a `substitutionGroup` reference to the element declaration mapped from the subsetted property.
- A property in a PSM that is a derived union maps to an element declaration with an `abstract` value of true.

2.5.3.4 Examples

PIM Representation

Figure 7-32 shows an example of a substitution group defined in a «PropertyHolder» class as a set of properties that subset the head property ContactMeans. Since ContactMeans is a derived union, it represents an abstract property. The ContactMeans property of the ContactInformation «ObjectType» class is defined by reference to the head property ContactMeans, meaning that any of the properties in the substitution group for ContactMeans is substitutable for ContactMeans in ContactInformation.

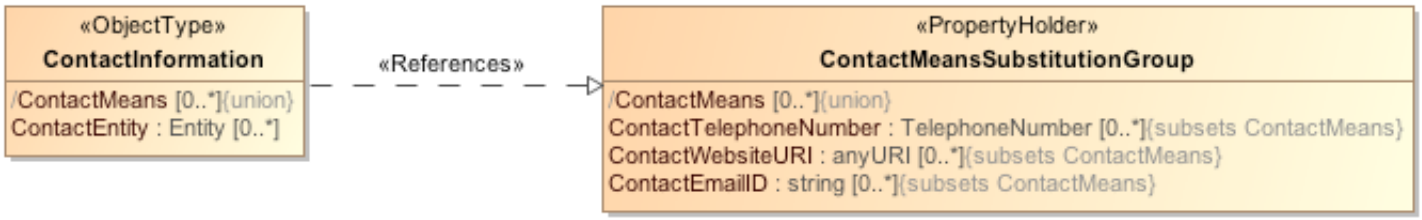


Figure 7-32 Representation of a substitution group using UML subsetted properties in a PIM

Figure 7-33 shows how a substitution group defined in one NIEM namespace may be extended in another namespace. The generalization between ExtendedContactMeansSubstituionGroup and ContactMeansSubstitutionGroup is required in order to establish a subsetting context that allows ContactSkypeID to subset the ContactMeans head property declared in ContactMeansSubstitution Group.

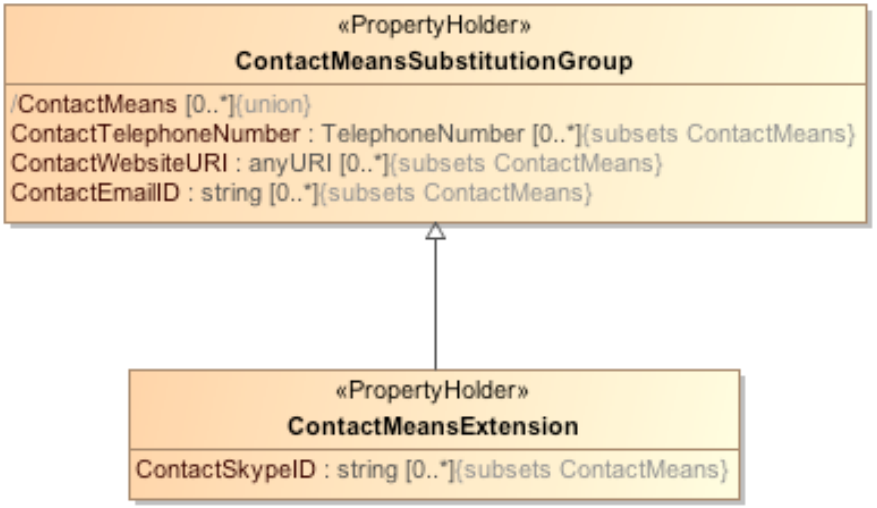


Figure 7-33 Extending a substitution group in a PIM or PSM

XML Schema Representation

The substitution group modeled in Figure 7-32 is represented in XML schema as follows:

```
<xsd:element abstract="true" name="ContactMeans" nillable="false"/>
```

```
<xsd:element name="ContactWebsiteURI" nillable="true"
  substitutionGroup="nc:ContactMeans" type="niem-xsd:anyURI">
   <xsd:annotation>
     <xsd:appinfo>
       <i:Base i:name="ContactMeans"
         i:namespace="http://niem.gov/niem/niem-core/2.0"/>
     </xsd:appinfo>
     <xsd:documentation>A</xsd:documentation>
   </xsd:annotation>
 </xsd:element>

 <xsd:element name="ContactTelephoneNumber" nillable="true"
  substitutionGroup="nc:ContactMeans" type="nc:TelephoneNumberType">
   <xsd:annotation>
     <xsd:appinfo>
       <i:Base i:name="ContactMeans"
         i:namespace="http://niem.gov/niem/niem-core/2.0"/>
     </xsd:appinfo>
     <xsd:documentation>A</xsd:documentation>
   </xsd:annotation>
 </xsd:element>

 <xsd:element name="ContactEmailID" nillable="true"
  substitutionGroup="nc:ContactMeans" type="niem-xsd:string">
   <xsd:annotation>
     <xsd:appinfo>
       <i:Base i:name="ContactMeans"
         i:namespace="http://niem.gov/niem/niem-core/2.0"/>
     </xsd:appinfo>
     <xsd:documentation>A</xsd:documentation>
   </xsd:annotation>
 </xsd:element>

 <xsd:complexType name="ContactInformationType">
   <xsd:annotation>
     <xsd:appinfo>
       <i:Base i:name="Object" i:namespace="http://niem.gov/niem/structures/2.0"/>
     </xsd:appinfo>
     <xsd:documentation>A data type for</xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
     <xsd:extension base="s:ComplexObjectType">
       <xsd:sequence>
         <xsd:element maxOccurs="unbounded" minOccurs="0" ref="nc:ContactMeans"/>
         <xsd:element maxOccurs="unbounded" minOccurs="0" ref="nc:ContactEntity"/>
       </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>
```

2.5.4 Choice Groups

2.5.4.1 Background

A *choice group* is a group of properties of a complex type such that exactly one of them may have a value in any instance of the complex type. (Adapted from [XMLSchemaStructures 3.8.1].)

2.5.4.2 Representation

Common

A choice group is represented as a UML class with the stereotype «Choice» applied, whose owned properties are the members of the group. A «Choice» class must have at least one property, and all the properties of the class must be optional (i.e., have multiplicity lower bound 0). The inclusion of the choice group in a complex type is represented by a normal UML property owned by the class representing the complex type and having the «Choice» class as its type.

PIM

There is no further PIM representation.

PSM

A class in a PSM with the stereotype «Choice» applied is implemented in XML schema as an `xsd:choice` model group in each complex type corresponding to a class with a property that uses the «Choice» class as its type. All the properties of a «Choice» class must represent XSD elements.

2.5.4.3 Mapping Summary

PIM to PSM Mapping

- A class in the PIM with the stereotype «Choice» applied maps to a corresponding class in the PSM with the stereotype «Choice» applied.

PSM to XML Schema Mapping

- A property in a PSM with a «Choice» class as its type maps to an `xsd:choice` model group. The property multiplicity gives the occurrence bounds for the group. The properties of the «Choice» class map as properties (see Subclause 7.5.1) to members of the model group. (Note that the «Choice» class does not itself map to a type in the XML schema.)

2.5.4.4 Example

PIM Representation

Figure 7-34 shows an example of a choice group in which only one of Date or DateTime may have a value. The property DateChoice models the inclusion of the choice group in the complex type represented by the DateType. Note that the names of the «Choice» class and the property that uses it are arbitrary. (The representation in a PSM is similar.)

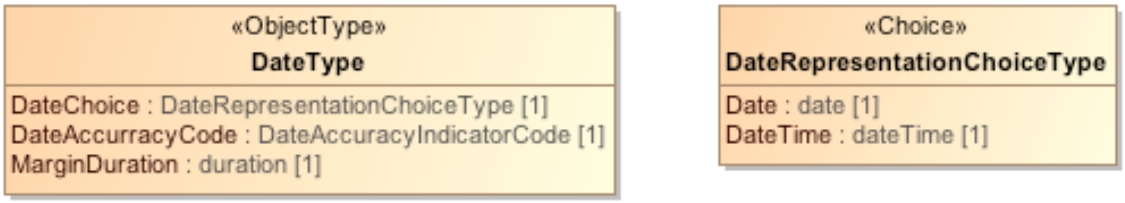


Figure 7-34 Representation of a choice group

XML Schema Representation

The choice group modeled in Figure 7-34 is represented in XML schema as follows:

```
<xsd:complexType name="DateType">
    <xsd:annotation>
        <xsd:documentation>A data type for a calendar date.</xsd:documentation>
        <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
                 i:name="Object"/>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
            <xsd:sequence>
                <xsd:choice>
                    <xsd:element ref="nc:Date" minOccurs="0" maxOccurs="1"/>
                    <xsd:element ref="nc:DateTime" minOccurs="0" maxOccurs="1"/>
                </xsd:choice>
                <xsd:element ref="nc:DateAccuracyCode" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="nc:MarginDuration" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

2.6 Packaging Models

2.6.1 Reference and Subset Models

2.6.1.1 Background

A central aspect of NIEM is the use of a reference model of business vocabularies as the basis for defining standard information exchange messages, transactions, and documents on a large scale: across multiple communities of interest and lines of business. This reference vocabulary includes both a common core and domain-specific updates.

A *reference model* is a model that provides:

- The broadest, most fundamental definitions of components in its namespace.
- The authoritative definition of business semantics for components in its namespace.

A *subset model* is a model with the same target namespace as a reference model that:

- Provides an alternate representation of components that are defined by a reference schema.

- Does not alter the business semantics of components in its namespace from those defined in the reference model.

(Adapted from [NDR 2.2, 2.3].)

2.6.1.2 Representation

Common

Currently, the reusable components of the NIEM reference vocabulary are rendered as XML schema. The NIEM Reference Model Library (see Annex C) provides a NIEM-UML model of all these reference schema. Each NIEM core and domain reference namespace is modeled as a package within the Reference Model Library.

Having the Reference Model Library available means that a NIEM PIM may reference properties declared in a reference namespace (see Subclause 7.5.2), in order to subset the Reference Model for a specific purpose. Such a subset model is required to have the same target namespace URI as some namespace in the Reference Model. Further, the subset model may only declare types and properties that correspond to those already defined for that namespace in the Reference Model, though, as the name indicates, it will only include a subset of what is in the Reference Model. This means that a subset model is not allowed to introduce new content, nor is it allowed to extend the data content defined by a component of the Reference Model.

PIM

A subset model may be represented as a «Namespace» package with the appropriate reference namespace as its target namespace. All NIEM types represented in a subset model must have the same NIEM name as some corresponding type represented in the Reference Model and all NIEM properties in a subset model must be defined by reference to property declarations represented in the Reference Model.

A subset model can be straightforwardly represented using the convenience notation defined in Subclause 7.5.2 to model a «References» realization from each class in the subset model to the corresponding class with the same NIEM name in the Reference Model. Since all the properties in a class in a subset model must have the same NIEM names as corresponding properties in the reference class, having a class-level realization implies that all the properties in the subset class are defined by reference.

As a further convenience, a «References» realization may be used between two «Namespace» packages. In this case, any UML class in the client package with the same NIEM name as a UML class in the supplier package is considered to have an implicit «References» realization to the matching class in the supplier package. In addition, any use in the subset package of an element (e.g., classifier or property) from the reference package is considered to instead refer to a similarly named element included in the subset package. In particular, the use of a property declaration from the reference package as the supplier of a «References» realization from an element of the subset package results in the reference actually being to a corresponding property declaration in the subset package.

Moreover, if the «InformationModel» stereotype is used (see Subclause 7.2.1), then subset and reference packages can be explicitly identified as having those default purposes. If a subset «InformationModel» package has a «References» realization to a reference «InformationModel» package, then the subset package must have the same target namespace as the reference package.

PSM

In a PSM, a subset model is represented as a «Namespace» package with the same target namespace as a reference schema. All classifiers and properties in the subset model must have the same names as corresponding elements in the reference model. Note that «References» realizations to the reference model elements are not used for subset modeling in a PSM – all relevant reference model elements are copied in the subset model.

2.6.1.3 Mapping Summary

PIM Representation Mapping

- A realization between two «Namespace» packages in a PIM with the stereotype «References» applied shall be considered equivalent to replacing the realization between the packages with multiple «References» realizations between classes contained (directly or indirectly) in the packages, such that:
  - If a class in the client package of the original realization has the same NIEM name as a class in the supplier package, then there is a realization from the class in the client package to the class in the supplier package.
- The use of a classifier in «Namespace» package in a PIM that is from another «Namespace» package with the same target namespace as the first shall be considered to be equivalent to replacing the classifier from the second package with a classifier from the first package with the same NIEM name. If no such classifier exists in the first package, one is created.
- A «References» realization in a PIM between two properties in different «Namespace» packages with the same target namespace is considered equivalent to replacing the realization with one having the same client but a supplier from the first package with the same NIEM name as the original supplier. If no such supplier exists in the first package, one is created. If a class already exists with the same NIEM name as the owning class of the original property, then the newly created property is added to that class. Otherwise, a new class is created with the appropriate name to hold the property.

2.6.1.4 Example

Figure 7-35 shows an example of a small subset model with two classes with properties defined by reference to classes in the Reference Model. Figure 7-36 shows an alternative representation of the same model using a «References» realization between the two packages.

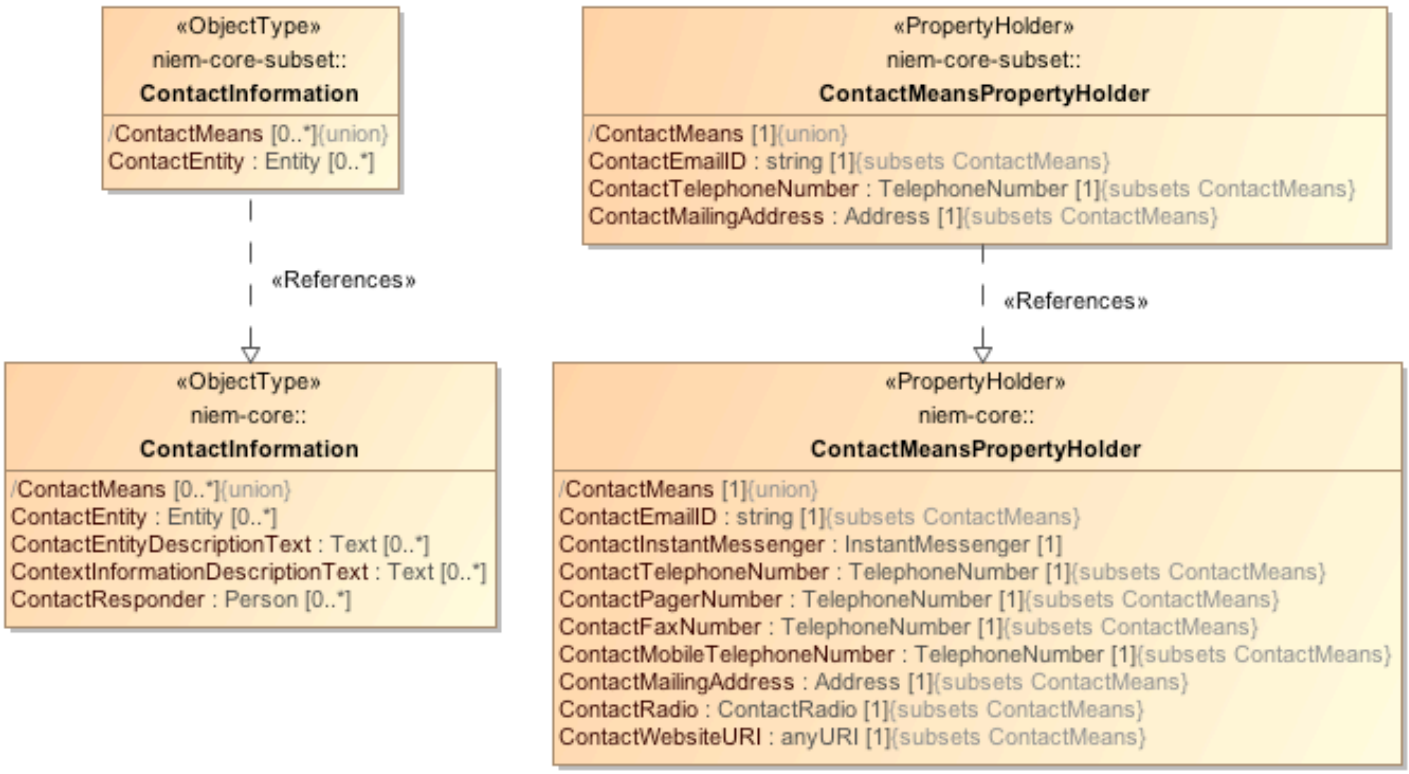


Figure 7-35 Representation of a subset model using «References» realizations

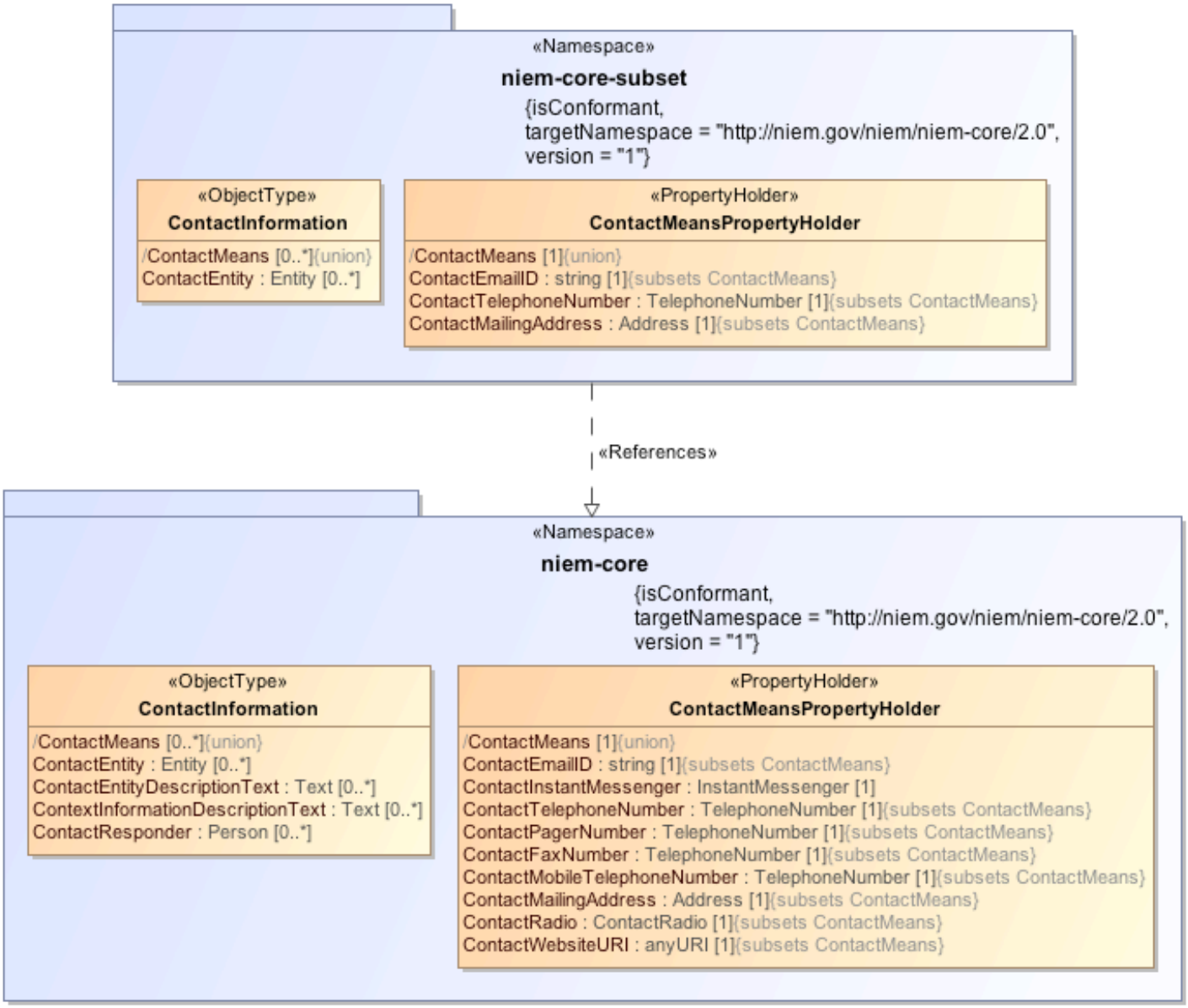


Figure 7-36 Alternative Representation using «References» realizations between packages

2.6.2 Model Package Descriptions

2.6.2.1 Background

A *Model Package Description* (MPD) is a compressed archive of files that contains one and only one of the five classes of NIEM IEM, as well as supporting documentation and other artifacts. An MPD is self-documenting and provides sufficient normative and non-normative information to allow technical personnel to understand how to use and/or implement the IEM it contains. [NIEM-MPD 1.1]

An *Information Exchange Model* (IEM) is one or more NIEM-conforming XML schemas that together specify the structure, semantics, and relationships of XML objects. These objects are consistent XML representations of information. Currently, five IEM classes exist in NIEM: (1) numbered release, (2) domain update, (3) core update, (4) Information Exchange Package Documentation (IEPD), and (5) Enterprise Information Exchange Model (EIEM).

The primary type of MPD supported by this specification is the IEPD, which is an MPD that contains NIEM-conforming schemas that define one or more recurring XML data exchanges.

2.6.2.2 Representation

Common

A MPD is represented as a UML component with the «ModelPackageDescription» stereotype applied. The attributes of the stereotype can be used to set the various properties of the MPD.

Artifacts are modeled as being included in an MPD using a UML usage dependency stereotyped as «ModelPackageDescriptionFile» from the «ModelPackageDescription» component to the artifact to be included. The «ModelPackageDescriptionFile» stereotype includes natureCode and purposeCode attributes used to identify the nature and purpose of the artifact being included [NIEM-MPD 4.2.4 and Appendix G]. In particular, the inclusion of an XML schema in an MPD is represented by using the «Namespace» package representing the schema (see Subclause 7.2.1) as the included artifact.

An MPD may also group artifacts into *file sets* [NIEM-MPD 4.2.3]. Such a file set is represented in an MPD model as a UML component with the «ModelPackageDescriptionFileSet» stereotype applied. The «ModelPackageDescriptionFileSet» stereotype includes attributes for identifying the nature and purpose of the file set. A «ModelPackageDescriptionFileSet» component must be the supplier of exactly one usage dependency whose client is the corresponding «ModelPackageDescription» component. Artifacts are modeled as being included in a file set be using «ModelPackageDescriptionFile» usage dependencies from the «ModelPackageDescriptionFileSet» component, in the same way as they are used to include artifacts directly in a «ModelPackageDescription» component. Note that one artifact may be included in multiple file sets.

Relationships between MPDs may be representing by using a dependency between the packages with the «ModelPackageDescription» stereotype applied.

PIM

If a PIM «Namespace» package is included in an MPD model, then the NIEM schema included in the MPD is considered to be the schema represented by the PSM representation of the «Namespace» package and its content, as mapped from the PIM. If the package is an «InformationModel» package with a default purpose, then the usage dependency between the «ModelPackageDescriptionFile» component and the package need not be stereotyped. Instead, the nature of the artifact is implicitly assumed to be “XSD” and the purpose is given by the default purpose. However, if the «InformationModel» package is to be used for a purpose other than the default purpose, then an explicitly stereotyped «ModelPackageDescriptionFile» usage dependency may be used, and the purposeCode specified for that dependency overrides the default purpose for the package.

If a PIM «InformationModel» package that is modeled as being included in an MPD has a usage dependency on another «InformationModel» package, then that latter package is also considered to be included in the MPD, even if there is no direct usage dependency between the component representing the MPD and that package. The value of the default purpose of the «InformationModel» stereotype is used to determine the purpose for the inclusion of the package in the MPD, as above.

Further, if the default purpose of the used package is subset and the using package uses any elements of the reference model (e.g., classifiers used as types or properties used as the suppliers of «References» realizations), then these uses are considered to instead be substituted with uses of elements from the subset package with the same name. If such elements do not exist in the subset package, then they are considered to be implicitly created *in the subset package*, in the same way as would result from reference element uses within the subset package itself (see Subclause 7.6.1).

Note that this means that the schema content mapped from a subset package may be *contextual*, depending on how the subset package is actually used within an MPD model. Essentially, such a subset model may be considered a model of the *intent* to create a subset schema to support a certain schema set within an MPD, rather than a detailed specification of exactly what that subset must be.

PSM

A PSM «Namespace» package is always included in an MPD model using an explicit «ModelPackageDescriptionFile» usage dependency, with the natureCode and purposeCode given.

2.6.2.3 Mapping Summary

PIM Representation Mapping

- An unstereotyped usage dependency from a «ModelPackageDescription» or «ModelPackageDescriptionFileSet» component to an «InformationModel» package shall be considered equivalent to the usage dependency having the «ModelPackageDescriptionFile» stereotype applied with a natureCode of “XSD” and a purposeCode corresponding to the value of the defaultPurpose attribute of the «InformationModel» stereotype.
- If an «InformationModel» package included in an MPD has an unstereotyped usage dependency on another «InformationModel» package in an MPD model, and the later package is not the client of a «ModelPackageDescriptionFile» usage dependency, then this shall be considered equivalent to explicitly modeling a «ModelPackageDescriptionFile» usage from the component representing the MPD to the second «InformationModel» package, with the purpose being given by the value of the defaultPurpose attribute of the «InformationModel» stereotype, as above. (Note that this rule may then need to be applied recursively to the second package.)
- If an «InformationModel» package has an MPD model has an unstereotyped usage dependency on an «InformationModel» package with defaultPurpose=subset, then any uses of elements of the reference model corresponding to the subset package shall be considered to instead be uses of corresponding elements from the subset package with the same NIEM name. If a corresponding element does not exist in the subset package, one is created in the same way as specified for the PIM Representation Mapping in Subclause 7.6.1.3.

MPD Model to MPD Artifact Mapping

- A component in an MPD model with the stereotype «ModelPackageDescription» applied shall map to an MPD file with the corresponding properties given by the values of the attributes of the «ModelPackageDescription» stereotype.
- A usage dependency in an MPD model with the stereotype «ModelPackageDescriptionFile» applied, from a «ModelPackageDescription» component to a «Namespace» package, shall map to the inclusion of the XML schema mapped from the «Namespace» package in the MPD represented by the «ModelDescriptionPackage» component, with a File element determined by the values of the attributes of the «ModelPackageDescriptionFile» stereotype. (If the «Namespace» package is from a PIM, it is first mapped to a PSM representation before being mapped to an XML schema.)
- A usage dependency in an MPD model from a «ModelPackageDescription» component to a «ModelPackageDescriptionFileSet» component shall map to a FileSet element in the modeled MPD as determined by the values of the attributes of the «ModelPackageDescriptionFile» stereotype. A usage dependency with the stereotype «ModelPackageDescriptionFile» applied, from the «ModelPackageDescriptionFileSet» component to a «Namespace» package, shall map to the inclusion of the XML schema mapped from the «Namespace» package as a file in the FileSet, with a corresponding File element as determined by the values of the attributes of the «ModelPackageDescriptionFile» stereotype.
- A dependency in an MPD model with the stereotype «ModelPackageDescriptionRelationship» applied, from one «ModelPackageDescription» component to another, shall map to a relationship recorded in the MPD mapped from the first component with a URI referencing the MPD mapped from the second component.

2.6.2.4 Example

Figure 7-37 is an example of the representation of two MPDS that share three files (namespaces) through «ModelPackageDescriptionFile» usage dependencies.

Figure 7-37 Representation of two NIEM MPDs with included namespaces (files)

Please see Figure A-27 for an additional example.

Annex A NIEM-UML PIM Example

A.1 Example Description

This example is intended to illustrate use of the NIEM-UML PIM. This is a fictitious example that uses many, but not all, of the NIEM-UML features. This example assumes some knowledge of UML and NIEM, but you don't need to be an expert. Note that this example is intended to be read with the normative NIEM-UML specification.

The business use case is for "Pet Adoption Centers" which need to share information on their pet adoptions with each other and with government agencies. The information required includes data about the pets, the people adopting the pets and the pet adoption centers. Information for sets of adoptions is defined in a NIEM exchange as part of an "IEPD".

A.2 Organization of NIEM Information Models and Classes

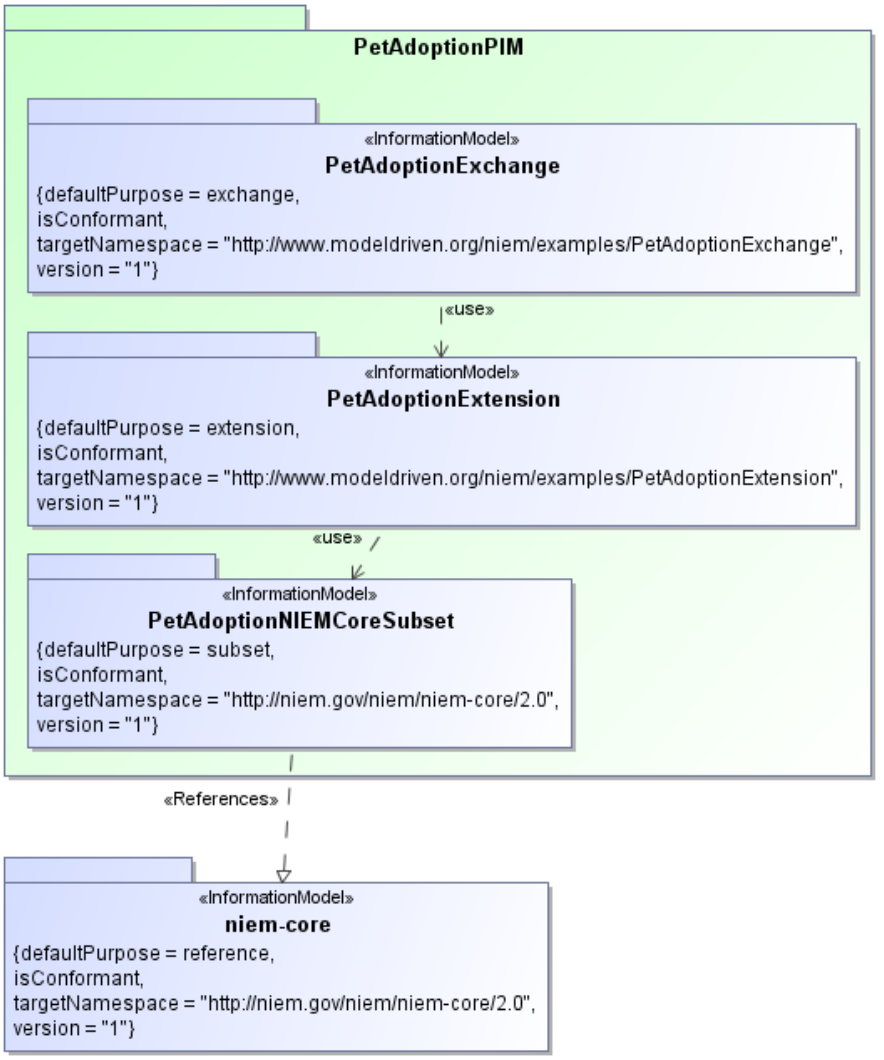


Figure A-1 Namespace Organization

As with all NIEM exchanges an essential part of the analysis is the reuse of the NIEM reference vocabularies. Since there is no established domain for pet adoptions NIEM-Core is reused, much of the information required is already defined in NIEM-Core and thus needs to be structured for our particular use case.

What cannot be found in NIEM-Core is defined as new information types in an "extension information model" for pet adoptions. The reused and extended classes are then combined to form an exchange information model – the actual data structure for a specific data exchange.

The Pet Adoption PIM model is organized into three subpackages, each representing a particular kind of NIEM information model. Each package is stereotyped as a NIEM «InformationModel» which has tags for the URI, version and NIEM compliance. The nature of the namespace (as subset, exchange, extension, etc. is defined as the package is used in an MPD). The packages are:

- **PetAdoptionNIEMCoreSubset** – this is a NIEM "subset namespace" (or subset schema) that has the special role of subsetting a single reference namespace for use in a particular MPD. A subset namespace can't add any new information.
- **PetAdoptionExtension** – this is a NIEM "extension schema" and includes new concepts about pets and pet adoptions that could be reused in other MPDs. The extension namespace uses and extends elements from the subset namespace.
- **PetAdoptionExchange** – this is a NIEM exchange namespace and includes the classes representing actual exchanges between parties. The exchange namespace uses classes from the subset and exchange namespace to specify these data packages.

The model elements, below, are all defined inside one of these namespaces, the namespace name is shown below the class names.

Note that there is one additional package, which is used to hold the Model Package Description, defined in Subclause A.23.

A.3 High-level design

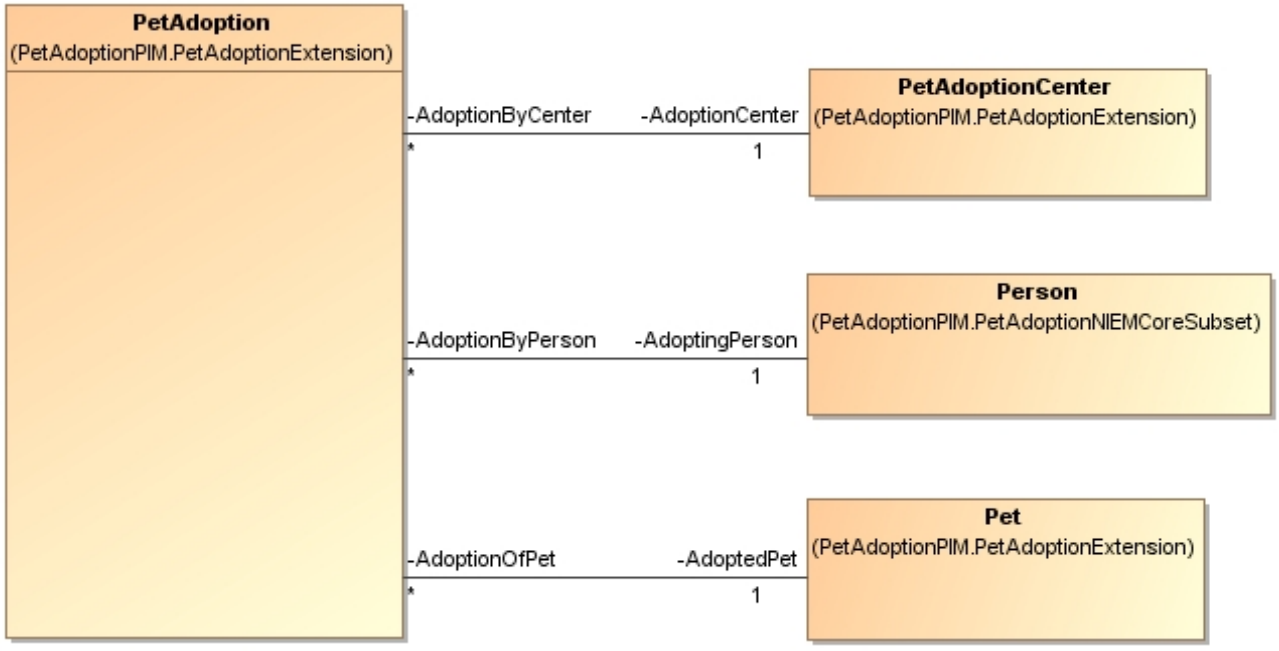


Figure A-2 High Level Design

This high-level UML model shows the primary classes of our information model:

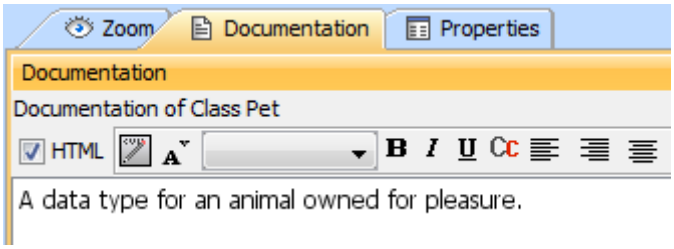
- Pets

- Persons
- Pet Adoption Centers
- Pet Adoptions

A pet adoption is the event that binds together all of these elements and will be the primary subject of our information exchange.

Each UML class has both a name and an owning package (shown below the name). The package owning a class corresponds with a NIEM namespace and is an essential element of the design. Note that “Pet”, “PetAdoption” and “PetAdotionCenter” are part of the “PetAdoptionExtension” schema. An extension schema is normally where new domain concepts are defined.

A.4 Documenting Elements



NIEM requires most elements to be documented. A single UML comment is used to document an element using the NIEM rules regarding the format of documentation. Most UML tools provide an easy way to create such documentation elements. The documentation for class “Pet” illustrates NIEM compatible documentation in Figure A-3.

Figure A-3 Documenting Elements

A.5 UML Associations Defining NIEM Properties

The lines between the classes in the above diagram are UML associations. UML associations define the relationship between classes. Note that there is also the concept of a NIEM association which has some additional capabilities; we will look at these later. At each end of the association is an “association end”. Each association end defines a *NIEM property* for the *opposite end* that specifies the property’s name, type and multiplicity. Along with the end you will notice a multiplicity notation; multiplicity defines how many values a property may have. Where there is a “*” any number is allowed. Frequently you will see a range of values, such as 1..* which means at least one with no limit. Given this you can see that six properties are defined by the above associations as follows:

- The “AdoptionOfPet” property of pet (a Pet Adoption), of which there can be any number of values (including zero)
- The “AdoptedPet” property of “PetAdoptions” (a Pet), which must have one value (each adoption is for a single pet)
- The “AdoptingPerson” property of “PetAdoption” (A Person), which has one value per adoption.
- The “AdoptionCenter” property of “PetAdoption”, which also has one value, and,
- The “AdoptionByCenter” property of “PetAdoptionCenter” (An adoption), which can have any number of values (an adoption center adopts many pets).
- The "AdoptionByPerson" property of "Person” (An Adoption) which can have any number of values.

Each of these properties is a reference to an instance of the other class – they are each separate entities connected by these associations. If the information about the entities was contained in one of these classes (as embedded content) a UML “Aggregation” (diamond) would be used, we will see this later as well.

A.6 UML Enumerations Defining NIEM Code Types

Figure A-4 Enumeration

One thing we would like to know about our pets is what kind of pet they are. We will define a UML "Enumeration" for our kinds of pets called "PetKind".

The enumeration is a type with a specific set of values. In Figure A-4 we have defined values with tokens for each kind of pet we are concerned with; Dog, Cat, Bird, etc. The UML enumeration defines a NIEM "Code Type".

A.7 Defining a simple property for an adoption center

Figure A-5 Simple Properties

Now that we have our enumeration type defined we can use it as the type of a property. We will add a simple property to the "PetAdoptionCenter" to define the kinds of pets it offers.

In Figure A-5 we have added a property called "PetKindsOffered" to "PetAdoptionCenter" with a "type" of "PetKind". By putting a "*" multiplicity after it we have also said that an adoption center may offer many kinds of pets. As a property the value for "PetKindsOffered" is an "aggregation" and will be contained as nested elements inside of PetAdoptionCenter data.

A.8 Properties of Pet

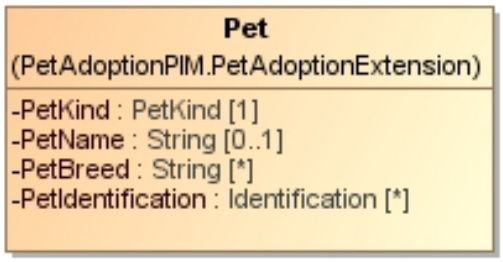


Figure A-6 Properties of Pet

Classes may have any number of properties of any type. There are also built in types for strings, integers, numbers, dates, etc. Any primitive data type that you can use in XML schema can be used in NIEM-UML. The expanded "Pet" class shows additional properties.

Pet has four properties:

- PetKind, using the same enumeration, above. A pet can only have one kind.
- PetName, a string. The pet name is optional.
- PetBreed, also a string. To support mixed breeds pets can have multiple breeds.
- And PetIdentification, which we will explore next.

A.9 Properties Using Classes as Their Types

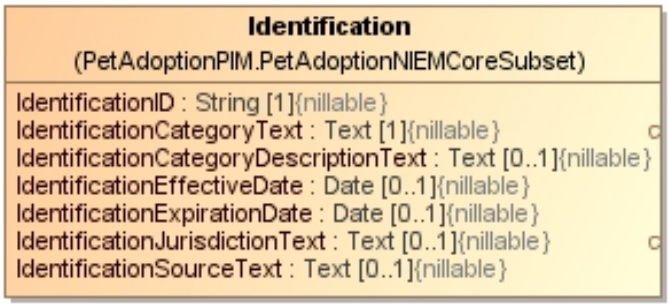


Figure A-7 Properties Using Identification Class

The type of the PetIdentification property is "Identification", let's take a look at the Identification class.

The identification class in Figure A-7 is another class like pet or person, but it already has several properties. Note that the properties have types like "String", "Text" and "Date" – these are all built-in primitive types. But where did all this come from? Note that it is in "PetAdoptionNIEMCoreSubset". This means that Identification reuses an existing class in NIEM Core.

A.10 Finding Classes in Reference Namespaces

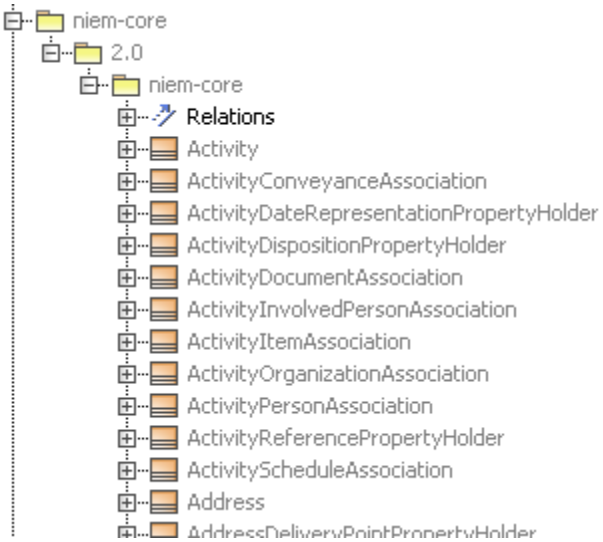


Figure A-8 NIEM-Core Listing

A primary feature of NIEM is the ability to reuse existing definitions. Being able to identify something is very common task for NIEM modelers so we looked in "NIEM-Core" to see what was there. NIEM-Core is one of the several reusable vocabularies defined in NIEM. On the right are a few of the classes in NIEM-Core as seen from a UML tool.

You can see in Figure A-8 that we are "in" the "niem-core", version "2.0". We see a start of the list of classes; there are a lot of them. We may use search features of the UML tool or just scan for what we want.

Further down we see classes having to do with Identification:

Figure A-9 Finding the Identification Class

Pulling the identification class into a UML diagram in Figure A-10 we see:

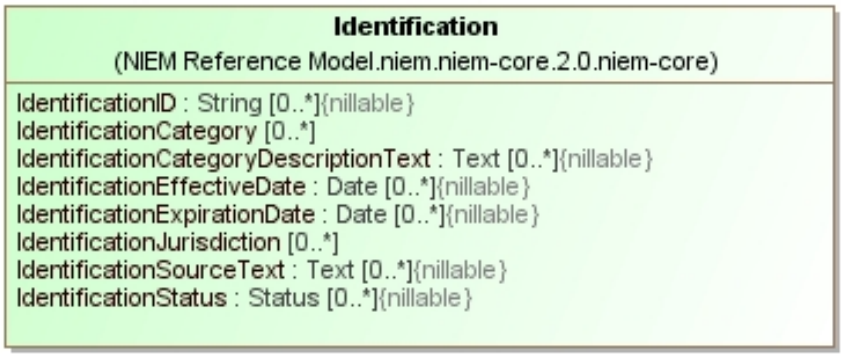


Figure A-10 Identification Class Contents

This seems to have a lot of the identification properties we need, perhaps more than we need! We also see that some of the properties have no type, these are placeholders for a "substitution group". A substitution groups allows different representations of a concept that can either be defined in your model or at runtime. Finding these in our model we see what representations these properties can have.

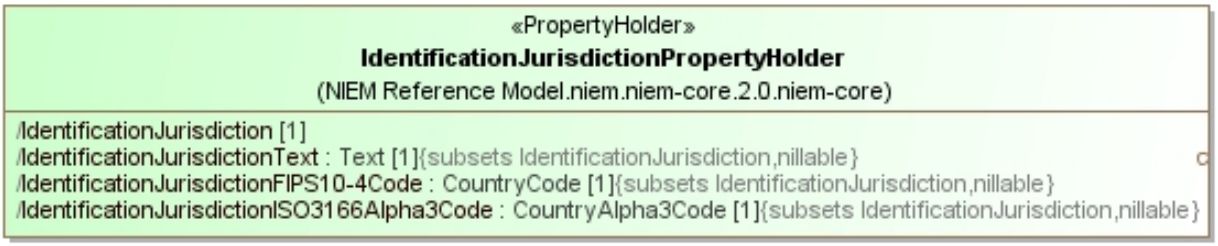


Figure A-11 Identification Substitution Groups

Note that in NIEM-Core "IdentificationCategory" can only have one representation (Text), however this could be expanded in other information models. On the other hand "IdentificationJurisdiction" can have 3 kinds of values, lets say that for our simple pets we only want one of these, the text version. We know that each of these is an alternate representation because it "subsets" another property. When one property subsets another, it defines a NIEM substitution group and these subset properties can be used in place of the property they subset.

There is one other special feature being used here, that is «PropertyHolder». The property holders define properties that can be used in classes but aren't used in any class yet. The property class is kind of invisible to NIEM. The properties in a property holder are known as "global properties". Since these global properties subset another they can be used in place of them, anywhere.

What we want to do now is use all these parts and pieces to define "Identification" in our model.

A.11 Defining a subset namespace with «References»

A NIEM subset information model (which should not be confused with UML subset properties) is a special information model where standard NIEM elements are reused and tailored for a specific purpose. A subset

information model can only tailor existing material, not define anything new. What we are going to do is define our own configuration for “Identification” that builds on all these parts.

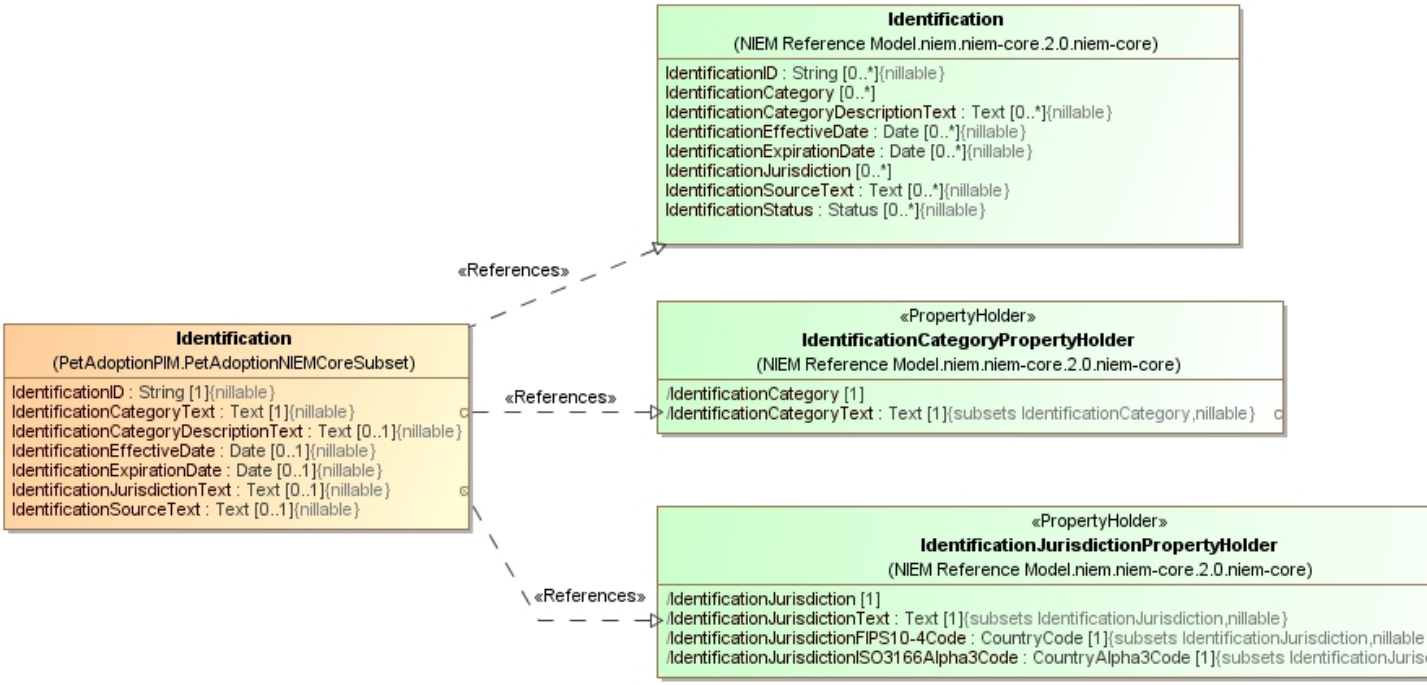


Figure A-12 Identification Subset Class Using References

The “Identification” class in “PetAdoptionNIEMCoreSubset” «References» Identification in NIEM-Core. What we did is copy-paste properties from the NIEM-core version to make the class we want for our pet model. We also don’t need all these options for categories and jurisdictions so we also referenced specific properties in those property holders. Since these subset properties in NIEM-Core “Identification” it is legal to reuse them here. We could have also chosen to keep all these options, but that just seemed like overkill.

Note that «References» is used between information models, classes or properties. When between classes all the properties with matching names are implicitly referenced. Each property that is referenced uses its definition from NIEM-Core and must be compatible with it.

So the “Identification” class on the left is the one we are going to use, it is the one that is the type of the pets identification but will also be used to identify people and adoption centers. Note that the multiplicities of our properties have been narrowed – this is a legal and normal thing to do in a subset schema.

So what we have done is find existing concepts in NIEM-Core and configure these for reuse in our customized class. This is a primary activity in NIEM – get used to it!

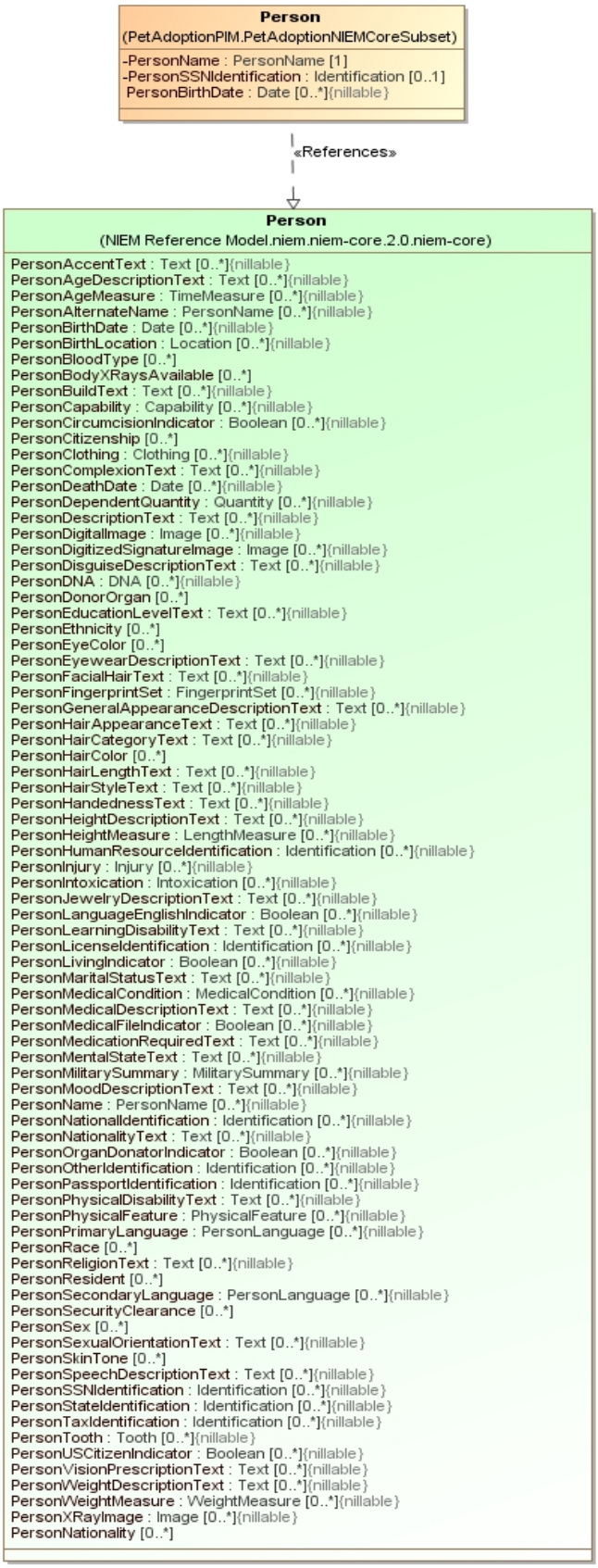


Figure A-13 Reuse of Person Class

A.12 Reusing Person

Our exchange also deals with people. NIEM-Core has a LOT of information about people. Here we see “Person” from NIEM core and also the small subset of it we will use in our example.

As you can see in Figure A-13, “Person” in NIEM-Core is huge! What we did is just pick 2 properties that we want out NIEM-Core person – we really don’t care much about their DNA or Disguises! As before, we just made properties with the same name and type as the reference person to reuse them, in most tools you can do this with a copy/paste.

Notice that we are already reusing parts of our own model, “Identification”. We have been able to use the very general idea of identification for both Pets and people, and we will be able to use if for adoption centers as well.

A.13 Reusing Person Name

The name of a person in NIEM core has a type of "PersonName". Without much problem we find "PersonName" in NIEM-Core. This is a very complete treatment of name, but since this has all been thought out we will use most of it, just not "personNameCommentText". So we have a person and a person's name. We can now use the "PersonName" property already defined in NIEM-Core

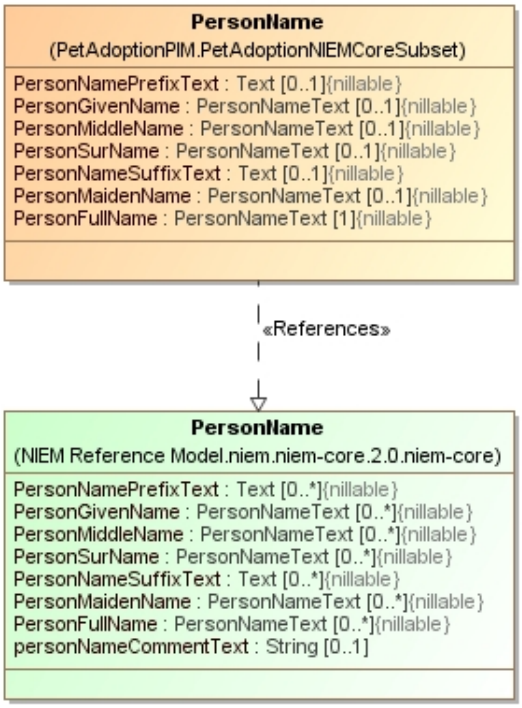


Figure A-14 Reusing Person Name

A.14 Contact Information

In addition to their name we are also going to want to record multiple ways to contact people (as well as adoption centers, but that is later).

There are two considerations for contact information with respect to Person – defining the contact information as well as creating an association between the contact information and person. First let's look at our definition of contact information; you should understand most of this picture now in Figure A-15.

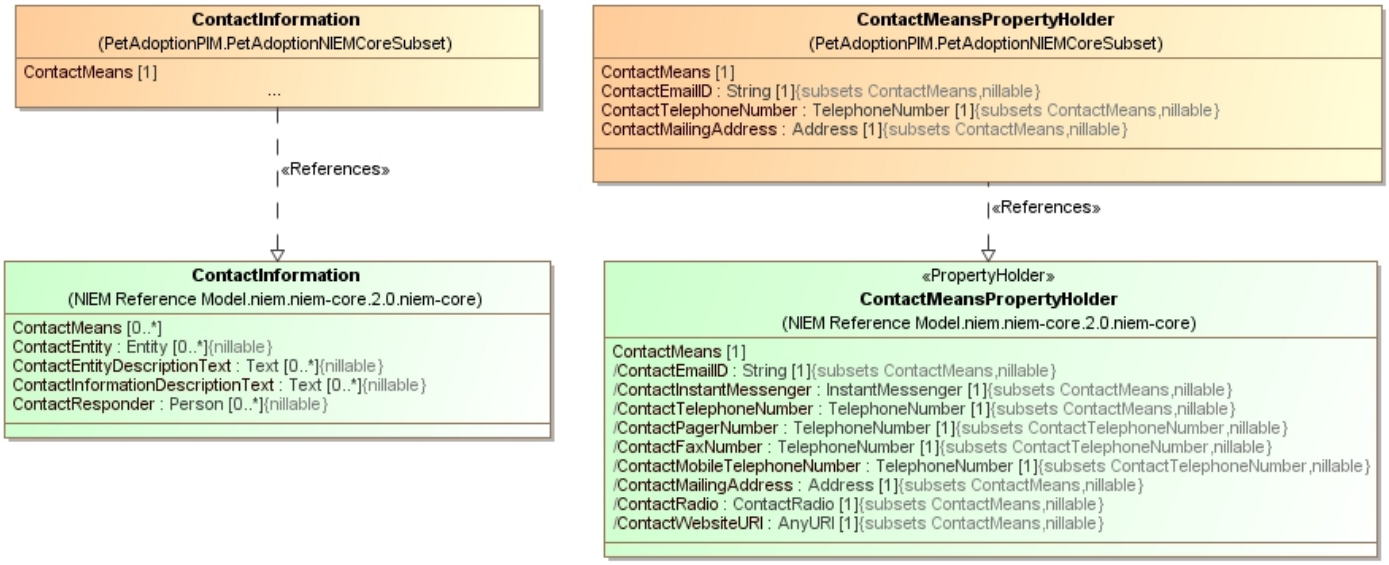


Figure A-15 Referencing Contact Information

As you can see in Figure A-15, this is a very general idea of contact information that can have any number of various kinds of contacts. Note that "ContactInformation" references "ContactMeans" which is used as the base of a substitution group with a set of properties that subset it. Contact means is further defined on the right as a set of possible properties. We have chosen to allow 4 possibilities out of the 10 pre-defined in NIEM-Core:

ContactEmailID, ContactTelephoneNumner, ContactMobileTelephoneNumber and ContactMailingAddress. Any of these forms of contact may be used as contact information in our model. By the way, if we didn't find it here we can also extend the list of possible representations by subclassing the property holder. Since we have used telephone number we have to define this as well, drawing from NIEM-Core. In Figure A-16 we will just show you this model fragment as it uses the same pattern of referencing.

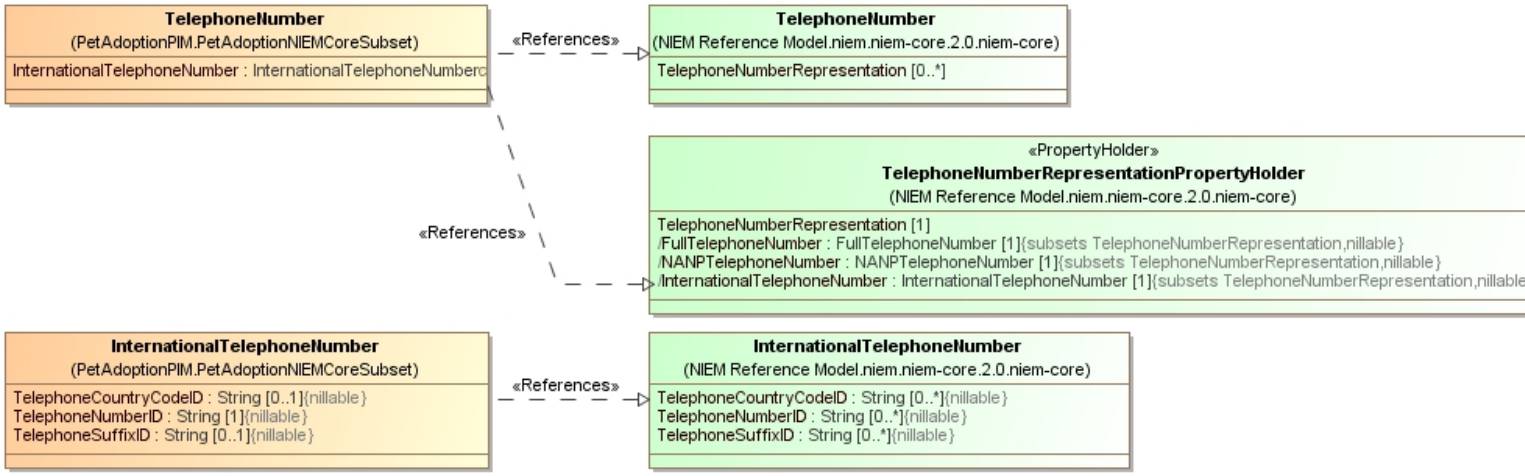
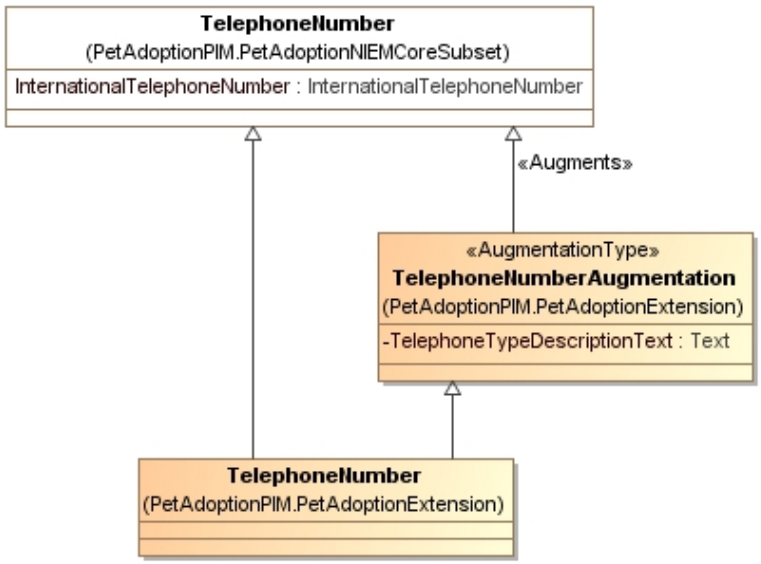


Figure A-16 Telephone Number

A.15 Augmenting Telephone Number

Consider that we wanted some additional information about telephone numbers and wanted to be able to "mix in" that information with a variety of telephone numbers. This is the use case for NIEM augmentations. An augmentation defines a type that can be "mixed in" with other types. Unlike regular objects it is legal to inherit multiple augmentations into a type since they are not represented as XML extensions, rather as augmentation properties. Augmentation properties can also be defined directly in NIEM-UML.

Our use case is that we want to define a telephone number augmentation for the type of telephone (i.e. land line, mobile, etc). We then want to extend the NIEM-core TelephoneNumber and define a new type in our extension schema that includes the telephone type.



The augmentation type in Figure A-17 is stereotyped as an «AugmentationType» and is also specified to «Augment» telephone number. This means that anything that uses TelephoneNumberAugmentation must be a telephone, note that specifying «Augments» is optional. In the NIEM XSD the TelephoneNumber in PetAdoptionExtension will extend telephone number but also include an augmentation property for TelephoneNumberAugmentation based on the NDR pattern for supporting augmentation in XML. Our new TelephoneNumber can now be used anywhere the NIEM-Core representation of telephone number may be used.

Figure A-17 Augmenting Telephone Number

A.16 Using a NIEM Association for Contact Information

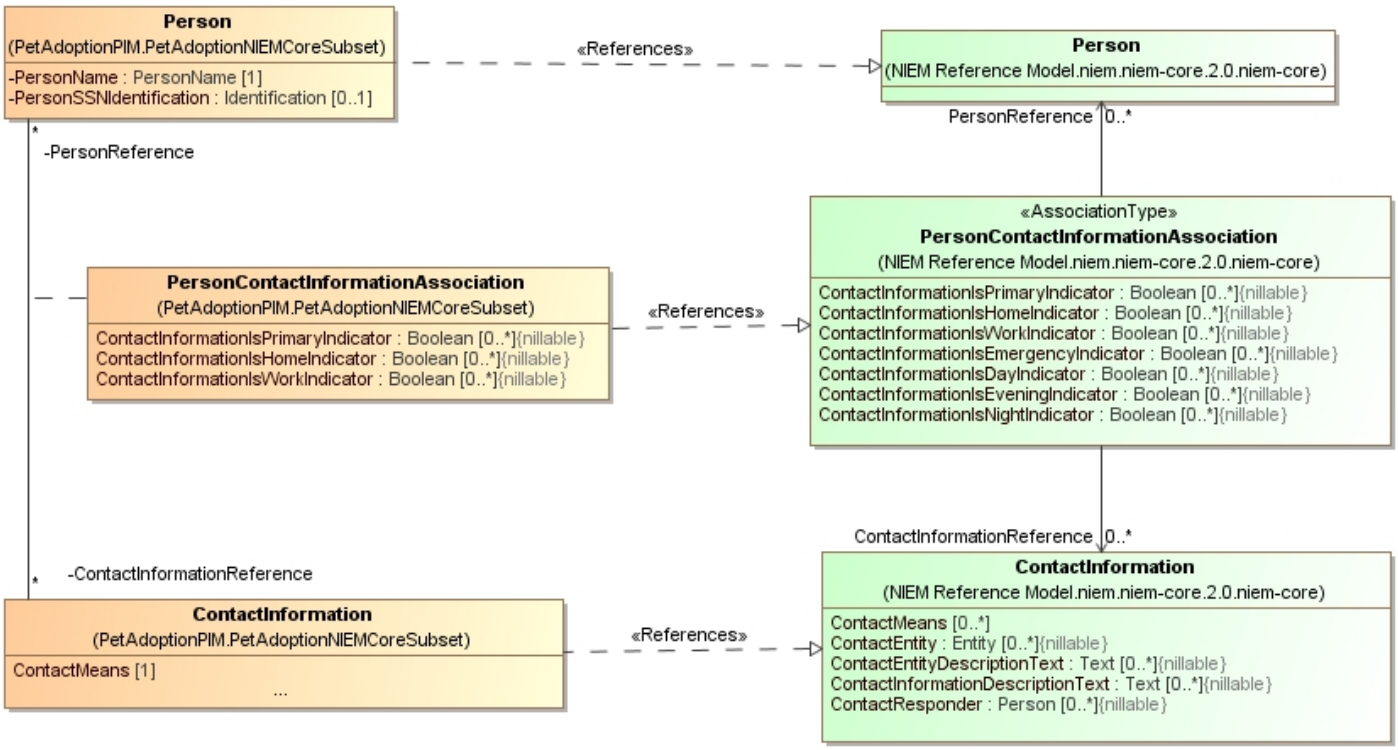


Figure A-18 Contact Information Association

Remember that when we used a UML association it made properties in the classes on the ends. For somewhat more flexibility NIEM-Core uses a "NIEM Association" between Person and Contact Information. A NIEM association isn't just a property; it is a first-class, stand-alone piece of data that relates two or more things.

In this Figure A-18 (which is still a subset of all the information) we put more of the pieces together and see a NIEM association "PersonContactInformationAssoiciation" that is defined in NIEM core and then reused in our example. This association class allows us to connect any person with any piece of contact information. So, for example, we could represent two people with the same address. As always, we reference the NIEM-Core and pick out the properties we want.

Note that PersonContactInformationAssoiciation on the right is an «AssociationType» where as we modeled it as a UML "Association Class", both of these representations mean the same thing in NIEM but we are using some built-in UML functionality in our PIM whereas the NIEM-Core is based on the structure as it is found in the XML schema. You have the choice of using association classes or classes with the «AssoiciatonType» stereotype. Note that there are some edge-cases that can only be expressed using «AssoiciatoonType» but that association classes have some additional power to define the multiplicity *between* the associated classes.

Since PersonContactInformationAssoiciation in our PIM is both an association and a class we can use it like any other class, it can be the type of properties or other associations – this provides for a very powerful network of information.

A.17 Pet Adoptions as a kind of activity

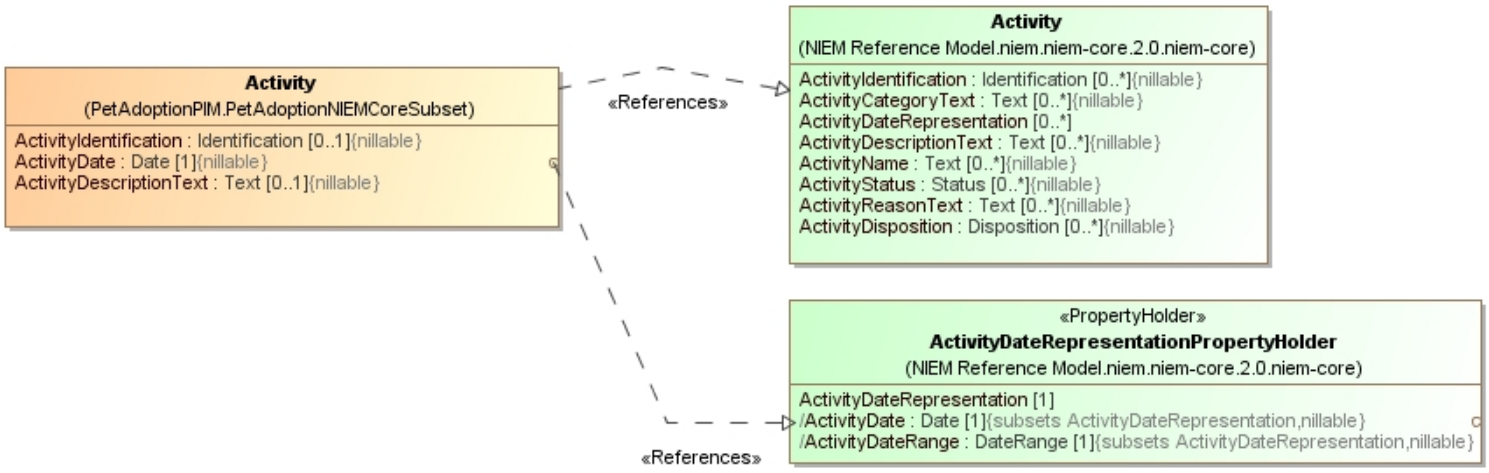


Figure A-19 Defining Activity From NIEM-Core

There is some more properties of a pet adoption we would like to consider, these come from it being a kind of *activity* and activities being available in NIEM-Core.

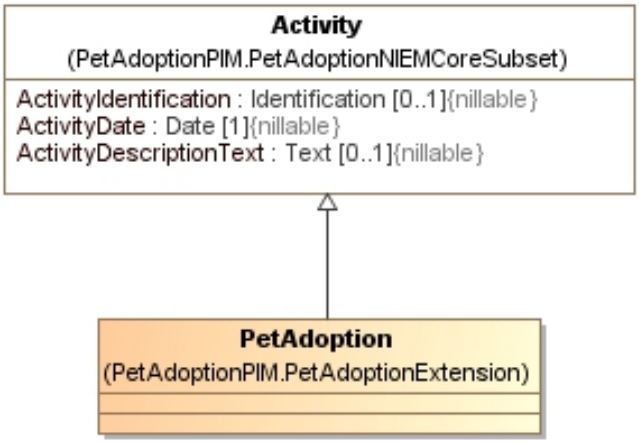


Figure A-20 PetAdoption as a kind of Activity

This pattern of reuse in Figure A-19 should look quite familiar now; we are combining the NIEM-Core concept of activity with one of the representations of that activities date. Since pet adoptions are a kind of activity it makes sense for pet adoptions to be a subclass of activity.

By making PetAdoption a UML Subclass of Activity all the properties of activity become available to Pet Adoption – every pet adoption is an activity.

The constraint in NIEM (Due to XML Schema restrictions) is that a class can only be a subclass of at most one other class. So you want to use a subclass only when the superclass can't be anything else at the same time (in the next section we will see how to handle other cases).

A UML subclass, or generalization, maps to an "extension" in XML schema unless it is generalizing an augmentation type or uses RolePlayedBy (see below). So in the XML, PetAdoption will extend Activity.

A.18 Pet Adoption Centers as a role of an organization

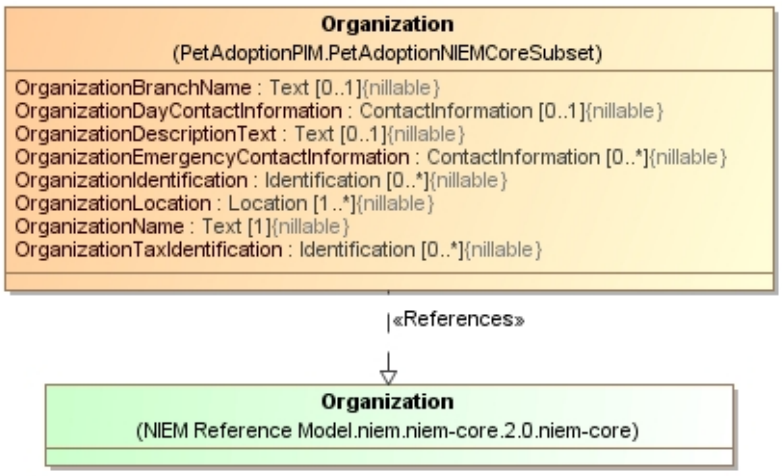


Figure A-21 Using Organization from Core

It may have occurred to you by now that while pet adoption centers are not that common a concept, that these are organizations that have a lot in common with other organizations. There is a well-developed model for organizations in NIEM-Core which we can reuse. Figure A-21 uses the same «References» pattern we have seen previously and we pick up some of the standard properties. We are not going to reproduce all the properties of Organization to save some space – you should understand this pattern by now and can look at the model for the details.

But, what is the relationship between pet adoption center and organization? One common way to express this is for something like adoption center to subclass organization –after all it is an organization. Another option is for per adoption center to be considered a "role" of an organization – this would allow for an organization to "play the role" of an adoption center while also playing other roles, perhaps as a rescue center or veterinarian.

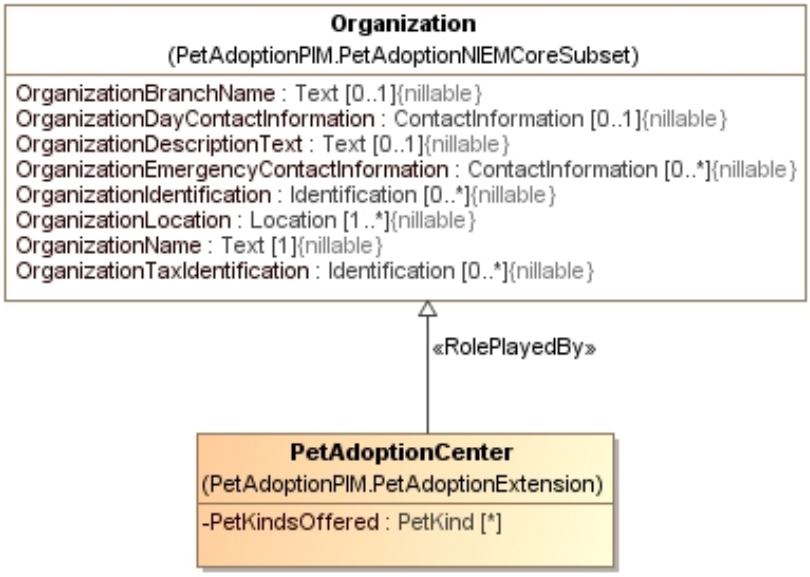


Figure A-22 Adoption Centers as a ROLE of an Organization

Figure A-22 shows that PetAdoptionCenter is a «RolePlayedBy» an organization. An organization may play multiple roles and these roles can come and go over time. This kind of role happens at most once for an organization, the organization isn't a PetAdoptionCenter multiple times. Such roles are defined by using the «RolePlayedBy» stereotype on a generalization. There is another type of role where a base type can play the role many times – for example a person may be a prize winner multiple times and each time has different characteristics. This other kind of role uses «RoleOf» properties. NIEM-UML supports both kinds of roles but they are both represented as properties in NIEM-XML (NIEM XML does not formally distinguish role types, it is implied by their multiplicities).

A.19 Putting together the high-level picture

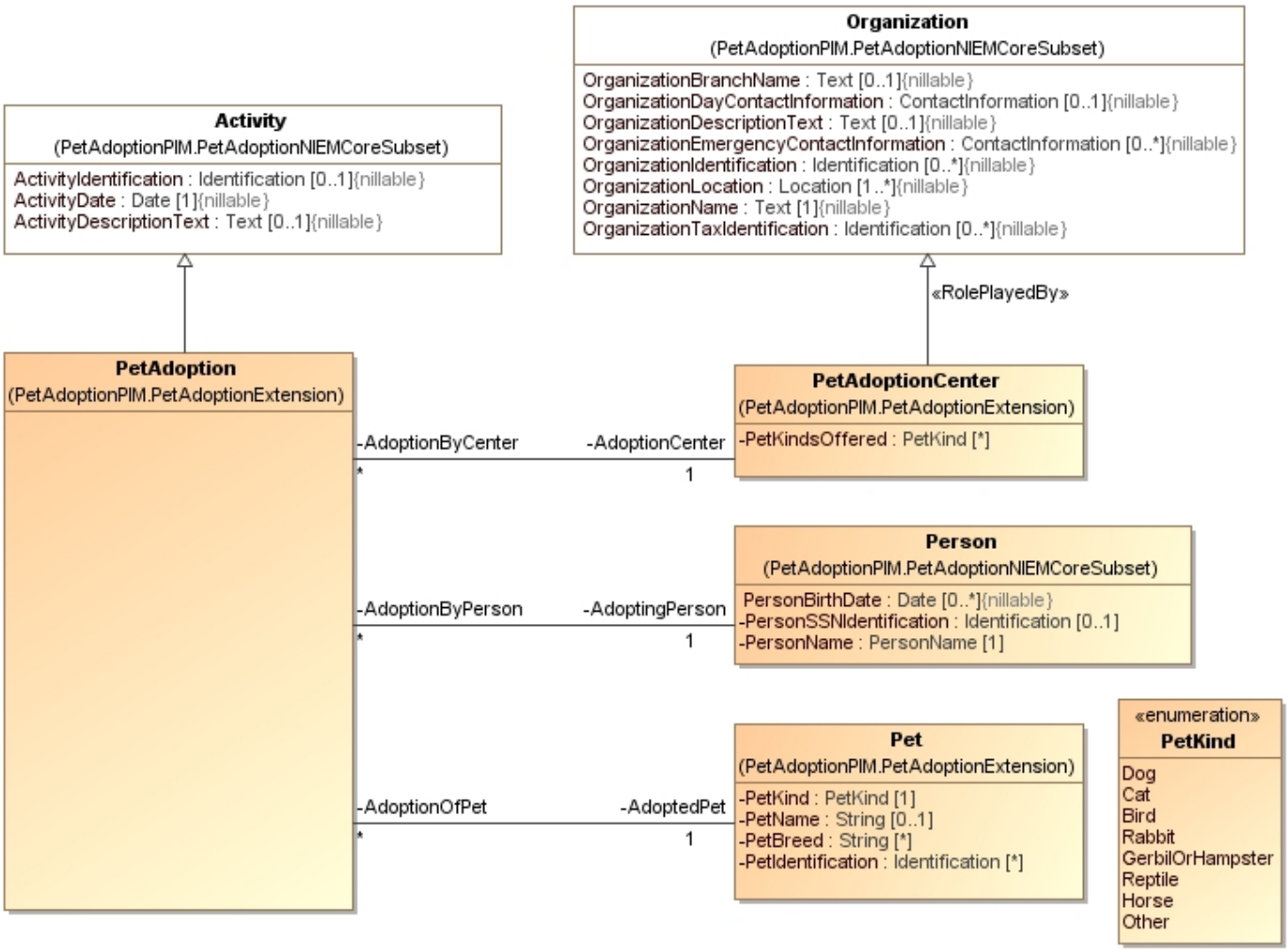


Figure A-23 Completed High Level Picture

With the “parts and pieces” we have built-up so far we can now fill out our high level diagram in Figure A-22, complete with properties, but not showing all the associations.

As we can see, the high-level diagram we started with has been filled-out with a combination of properties from NIEM-Core as well as some we defined for this domain. Of course this is augmented by the additional classes referenced by this model and the NIEM associations for contact information. While pet adoption is an unusual use case, we were still able to reuse most of our classes and properties from NIEM-Core. Note that there are additional contact associations for people and adoption centers that are not shown here.

A.20 Primitive types

We have been using properties with “primitive types” like strings numbers and dates. We have also seen some more specialized primitive types like “PersonNameText”. Where do these come from? The basic types like Strings come from a “Primitive type library” that is standard in NIEM-UML. It is imported by using the profile and always available. More specialized primitive types are either in NIEM-Core or defined within a model as subtypes of these built-in types. The following are the primitive types used in this example model.

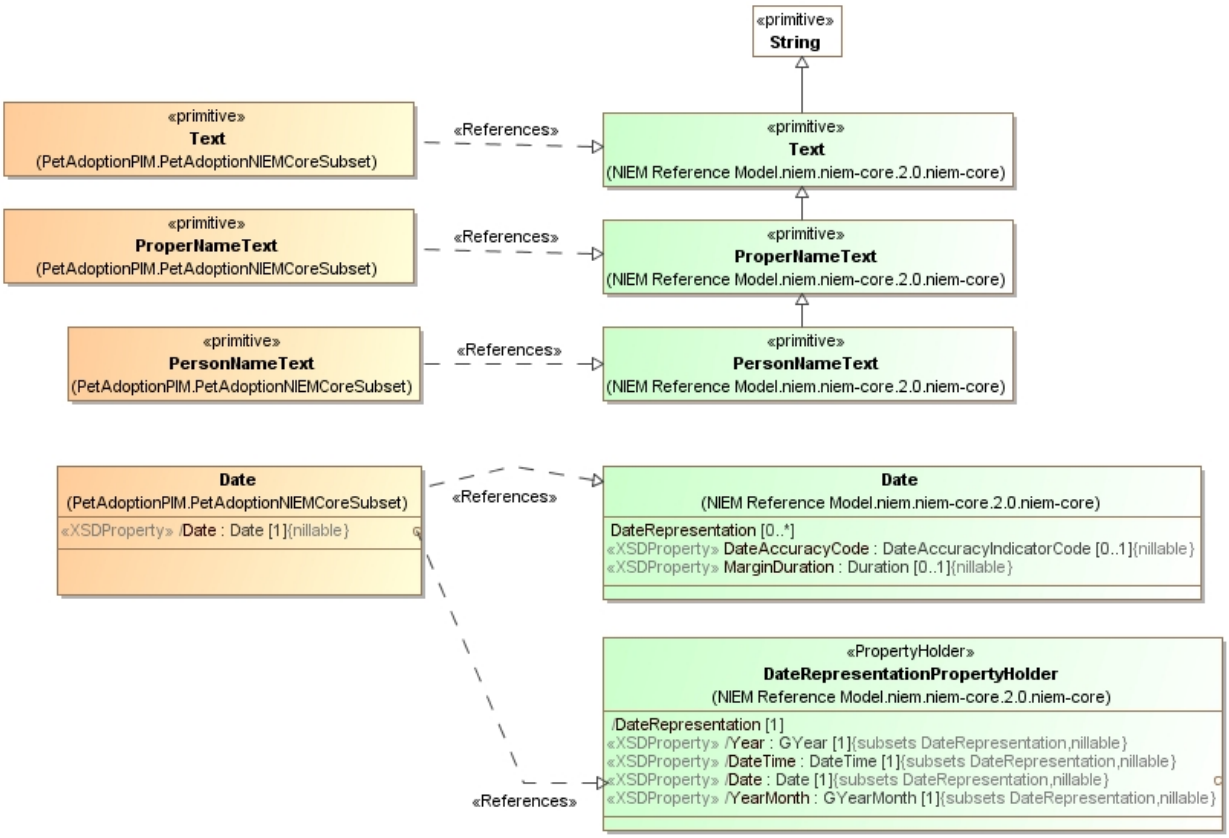


Figure A-24 Primitive Types

The above are shown for clarity – it is not nessisary to reference each and every primitive type. Any type that is referenced is automaticly included in the subset information model.

A.21 The Pet Adoption Exchange

The classes above serve to define information about pet adoption, but what exactly does a particular data exchange for a pet adoption look like? For a particular exchange we could have multiple adoptions, people, pets, etc.

Figure A-25 Pet Adoption Exchange

What we do is gather all of the classes we are concerned with together into a class defined in an "exchange schema package", these become the top level messages. The PetAdoptionExchange" in Figure A-25 is defined to contain a set of: People, Pets, PetAdoptions, PetAdoptionCenters, Addresses, Contact Information and associations. This information package aggregates all the others into a handy grab bag of pet related data.

A.22 Using classes by default

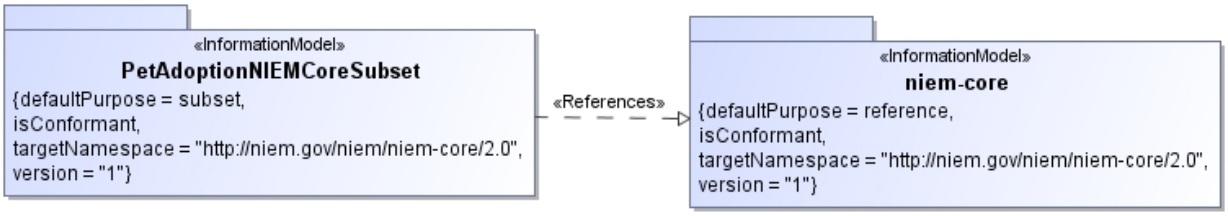


Figure A-26 Referencing NIEM Core

Figure A-26 shows that the entire “PetAdoptionNIEMCoreSubset” model «references» the “niem-core” model. By referencing the entire package anyNEIM-CORE elements that are used by PetAdoptionNIEMCoreSubset but not explicitly included are included by default. Also, any reference to an element in NEIM_CORE will automaticly be redefined to reference the cooresponding element in PetAdoptionNIEMCoreSubset. It is good practice to inspect the types and properties automaticly included to make sure you are not including more than is nessisary – this may be done in the PSM model or some tools may provide features for doing so.

Using the Cameo NIEM-UML features subsets are simplified. A subset namespace is created and populated with the customized user interface.

This concludes the platform independent part of the example.

A.23 The Pet Adoption IEPD Model

The platform independent model, above, specifies the information model relative to pet adoption, but not the IEPD its self. An IEPD is a NIEM artifact that includes all of the data and metadata relative to a kind of data exchange. An IEPD is a kind of "MPD" as defined in the NIEM "Model Package Description" specification. An IEPD has a very specific format and contents, part of which is the XML Schema files that may have been produced from a NIEM-UML model.

MPDs and IEPDs are also defined in a model, a model of the MPD artifact. This model is primarily the metadata concerning the MPD. It also references the PIM and PSM packages that contain the classes used in an exchange. Figure A-27 is the IEPD model for our example.

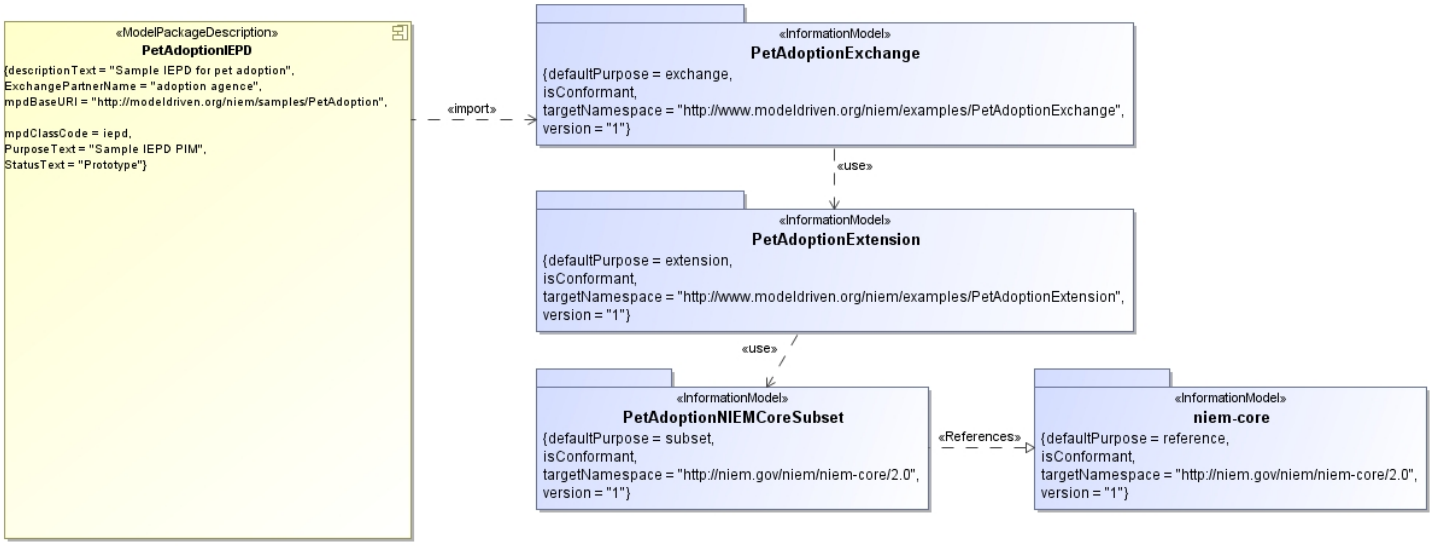


Figure A-27 IEPD Specification

The component stereotyped as "ModelPackageDescription" on the left represents the IEPD. The tag values of this stereotype define the IEPD metadata such as the description, URI and purpose. The "meat" of an IEPD is in the information models imported using a UML «import». The packages imported are those that we have been building all-along in our PIM, these are:

- PetAdoptionExchange – the package that holds the PetAdoptionExchange class. Each class in such a package will be a top-level unit of information exchange.
- PetAdoptionExtension – these are the new concepts defined for Pet Adoption, some of which use or extend NIEM-Core concepts.
- PetAdoptionNIEMCoreSubset – this is the namespace that subsets NIEM-Core using «References» and configures those concepts for our use in pet adoption.

Note that each information model has a defaultPurpose that cooresponds to the NIEM schema types. This purpose will be used unless that purpose is overridden using the «ModelPackegDescriptionFile» stereotype of an import.

Each namespace will produce an XML schema that becomes part of the IEPD.

Note that each namespace has a "targetNamespace" tag, its URI, a version and an indicator that the namespace conforms to NIEM. Also, for each package that uses elements of another, there is a UML «uses» dependency between those packages – this is required to establish the context for each information model.

Based on the PIM model combined with the MPD model, NIEM-UML compliant tools are able to produce a complete NIEM-Compliant MPD from a NIEM-UML model. The machine-readable files of this specification include the complete model as well as the generated IEPD.

Annex B Terms and Definitions

2.7 Definitions

Artifact (NIEM)

An electronic file or a labeled set of logically cohesive electronic files. For example, an IEPD is usually composed of many artifacts (XML schemas, XML files, documentation files, etc.)

Association (NIEM)

Establishes a relationship between objects, along with the properties of that relationship; provides a structure that does not establish existence of an object but instead specifies relationships between objects. A NIEM association may relate multiple objects.

Augmentation (NIEM)

A container element that bears additional properties that may be added to an object type to supplement the properties of the original object definition. Augmenting a type does not change the semantics of that type. A NIEM augmentation can only be applied to the types specified in its definition. Augmentations may be used in combination as needed to supplement an object.

Catalog (NIEM)

An artifact for an IEPD that identifies and classifies all artifacts that comprise the IEPD, and that also contains metadata associated with the IEPD. A catalog is an XML instance defined by the XML catalog schema specified in the NIEM Model Package Description (MPD) Specification.

Change Log (NIEM)

A formal or informal artifact that records the changes applied since the last release of the product the change log is associated with.

Core Update (NIEM)

Used to add new schemas, new data components, new code values, etc. to NIEM Core; in some cases a core update can make minor modifications to existing core data components; however it is never used to replace a NIEM core version.

NIEM Conformance (also NIEM-conforming)

Adherence to the NIEM Naming and Design Rules (NDR), Model Package Description Specification (MPD), and the more general NIEM Conformance Specification when developing a NIEM release, domain update, core update, IEPD (for an information exchange), or an EIEM (composed of BIECs) and their associated artifacts.

Constraint Schema (NIEM)

An IEPD schema with the purpose of restricting or constraining content that appears in instances of the subject schema. A constraint schema is not NIEM-conforming. Use of constraint schemas in IEPDs are a technique for enforcing additional constraints on schemas that cannot otherwise be enforced through the NIEM reference schemas.

Data Component (NIEM)

A W3C XML Schema definition for an XML type, element, attribute, or any other NIEM-conforming XML Schema construct. Sometimes also referred to as “metadata component.”

Domain Update (NIEM)

One or more XML schemas that are a replacement for or that supplement a given version of a published NIEM domain release or another domain update.

Exchange Schema (NIEM)

An IEPD schema with the purpose of defining the content model of the information exchange. An exchange schema works in conjunction with the subset, extension, and constraint schemas to form a complete package that represents the exchange. The exchange schema is essentially the root schema within the set of schemas that defines an exchange.

Extension Schema (NIEM)

An IEPD schema that extends existing NIEM data components (i.e., types and elements), or that defines new NIEM-conforming data components to be used in an information exchange.

NIEM Information Exchange Model (IEM)

One or more NIEM-conforming XML schemas that together specify the structure, semantics, and relationships of XML objects that are consistent representations of information. The five IEM classes in NIEM are: (1) release, (2) core update, (3) domain update, (4) Information Exchange Package Documentation (IEPD), and (5) Enterprise Information Exchange Model (EIEM).

NIEM Information Exchange Package (IEP)

An XML instance of an IEPD that is or will be the specific information exchanged between a sender and a receiver on-the-wire. In general, an IEPD contains schema and documentation artifacts. As part of its documentation, an IEPD is required to contain at least one sample IEP for each document (root) element defined within its exchange schema(s).

NIEM Information Exchange Package Documentation (IEPD)

The aggregation of XML schemas and associated documentation artifacts that completely specify and describe an information exchange. Documentation must include a catalog, change log, master document, and sample IEPs for each document element, and may optionally include other artifacts that may be useful to implementing the IEPD (e.g., business rules, business requirements, etc.).

Master Document (NIEM)

An artifact required in an IEPD that is the primary text-based documentation for the IEPD. The Master Document generally establishes baseline information about the IEPD and references any other optional and supplementary documentation. Similar to a “readme” file.

Metadata

Describes data about data, that is, information that is not descriptive of objects and their relationships, but is descriptive of the data itself.

Model

A formal specification of the function, structure and/or behavior of an application or system.

Model Driven Architecture (MDA)

An approach to system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

NIEM Model Package Description (MPD)

An organized set of files that contains one and only one of the five classes of NIEM IEM, as well as supporting documentation and other artifacts. An MPD is self-documenting and provides sufficient normative and non-normative information to allow technical personnel to understand how to use and implement the IEM it contains. An MPD is packaged as a compressed archive.

NIEM Core

The NIEM namespace (or corresponding XML schema) that contains all data components determined to have relevance to and semantic agreement by most or all participating domains. Notionally, NIEM Core contains all reusable data components that are not domain-specific and are governed by the NIEM Business Architecture Committee (NBAC).

NIEM Domain

A line-of-business, community-of-interest, or other similar grouping that is assigned a NIEM namespace, has responsibility to act as an authoritative source and steward of domain-specific data components, and can propose promotions of data components to the NIEM Core namespace.

NIEM-conformant Schema

An XML Schema document conforms to the NIEM Naming and Design Rules (NDR). These generally include reference schemas, subset schemas, extension schemas, and exchange schemas.

Normative

Provisions that one must conform to in order to claim compliance with the standard. (as opposed to non-normative or informative which is explanatory material that is included in order to assist in understanding the standard and does not contain any provisions that must be conformed to in order to claim compliance).

Normative Reference

References or specifications that contain provisions that one must conform to in order to claim compliance with the standard that contains said normative reference.

Object Constraint Language (OCL)

An adopted OMG standard and formal language used to describe expressions on MOF models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects; i.e., their evaluation cannot alter the state of the corresponding executing system. For the purpose of this specification, references to OCL should be considered references to the Object Constraint Language Specification, cited in Normative References, above.

Platform Independent Model (PIM)

A model of a subsystem at a logical level that contains no information specific to the platform or the technology that is used to realize it.

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

Query/View/Transformation (QVT)

A standard for writing transformation specifications between MOF based metamodels; a QVT engine is able to execute transformations and create or update a target model from a source model.

Reference Schema (NIEM)

An XML Schema document that meets all of the following criteria:

- It is explicitly designated as a reference schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It provides the broadest, most fundamental definitions of components in its namespace.
- It provides the authoritative definition of business semantics for components in its namespace.
- It is intended to serve as the basis for components in IEPD schemas, including subset schemas, constraint schemas, extension schemas, and exchange schemas.
- It satisfies all rules specified in the Naming and Design Rules for reference schemas.
- In general, NIEM releases are composed of NIEM reference schemas.

Release (NIEM)

A set of schemas published by the NIEM Program Management Office (PMO) and assigned a unique version number; a release is of high quality and has been vetted by NIEM governance bodies; includes micro, minor or major releases.

W3C Resource Description Framework (RDF)

A language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. By generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be *identified* on the Web, even when they cannot be directly *retrieved* on the Web. RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning.

Root Element (NIEM)

A globally defined element in a NIEM IEPD exchange schema. A root element can always be used as the top-level XML document element within an XML instance defined by the IEPD.

Schema Subset (NIEM)

A set of subset schemas derived from a NIEM reference schema set, usually a NIEM release. Any XML instance that validates with a correct schema subset will also validate with the complete reference schema set from which the schema subset was derived (See also "subset schema.").

Subset Schema (NIEM)

A schema that constitutes a part (i.e., subset) of a NIEM reference schema; a schema whose data components are taken entirely from a NIEM reference schema while excluding those components that are unnecessary for a given exchange. Subset schemas are generally used in an IEPD as related set, i.e., from the same reference schema set such as a NIEM release (See also "schema subset.").

Unified Profile for DoDAF and MODAF (UPDM)

A profile that defines a standard set of elements, relationships that exist between them, and a number of views and viewpoints which are used to support the development of an Enterprise Architecture primarily for the military community of interest.

XML Metadata Interchange (XMI)

XMI is a widely used interchange format for sharing objects using XML. Sharing objects in XML is a comprehensive solution that build on sharing data with XML. XMI is applicable to a wide variety of objects: analysis (UML), software (Java, C++), components (EJB, IDL, CORBA Component Model), and databases (CWM). For the purpose of this specification, references to XMI should be considered references to the XML Metadata Interchange (XMI) 2.0 Specification, cited in Normative References, above.

XML Schema Document (XSD)

A document written in the W3C XML Schema language, typically containing the “xsd:” or “xs:” XML namespace prefix and stored with the “.xsd” filename extension. Like all XML schema languages, XSD can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. Sometimes also referred to as XML Schema Definition.

eXtended Markup Language (XML)

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

2.8 Acronyms

BIEC	Business Information Exchange Component
DoD	Department of Defense
EIEM	Enterprise Information Exchange Model
IC	Intelligence Community
IEPD	Information Exchange Package Documentation
MDA	Model Driven Architecture
MPD	Model Package Description
NDR	Naming and Design Rules
NIEM	National Information Exchange Model
OCL	Object Constraint Language
PIM	Platform Independent Model
PM-ISE	Program Manager for the Information Sharing Environment
PSM	Platform Specific Model
QVT	Query/View/Transformation
UML	Unified Modeling Language
UPDM	Unified Profile for DoDAF/MODAF
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition