Date: June 2014

OBJECT MANAGEMENT GROUP

UML Profile for NIEM

Version 1.0

OMG Document Number: formal/2014-06-01
Standard document URL: http://www.omg.org/spec/NIEM-UML/1.0

Machine Consumable Files:

Normative:

http://www.omg.org/spec/NIEM-UML/20130801/NIEM-UML-Profile.xmi
http://www.omg.org/spec/NIEM-UML/20130801/XmlPrimitiveTypes.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference *
http://www.omg.org/spec/NIEM-UML/20130801/NIEMpim2psm.qvto
http://www.omg.org/spec/NIEM-UML/20130801/NIEMpsm2xsd.qvto
http://www.omg.org/spec/NIEM-UML/20130801/NIEMmpdmodel2artifact.qvto
http://www.omg.org/spec/NIEM-UML/20130801/NIEMmpdartifact2model.qvto
http://www.omg.org/spec/NIEM-UML/20130801/NIEMglobals.qvto
http://www.omg.org/spec/NIEM-UML/20130801/NIEMplatformBinding.qvto

Informative:

http://www.omg.org/spec/NIEM-UML/20130801/Nonnormative/Examples-Clause7.mdzip
http://www.omg.org/spec/NIEM-UML/20130801/Nonnormative/Examples-PetAdoption.mdzip

* List of NIEM Reference Vocabulary Library Model XMI files

http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-ansi_d20.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-apco.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-atf.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-cbrncl.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-census.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-dea.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-dod_ics-pub2.0-misc.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-edxl.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-edxl-cap.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-edxl-de.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-edxl-have.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-fbi.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-fips_5-2.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-fips_6-4.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-fips_10-4.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-geospatial.xmi

http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-have-codes.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-hazmat.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-icism.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-iso_639-3.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-iso_3166.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-iso_4217.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-itis.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-lasd.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-mmucc_2.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-mn_offense.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-nga.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-nlets.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-nonauthoritative-code.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-post-canada.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-sar.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-twpdes.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-ucr.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-unece_rec20-misc.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-usps_states.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-common-ut_offender-tracking-misc.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-core.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-emergency Management.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-familyServices.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-intrastructure Protection.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-intelligence.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-ixdm.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-maritime.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-domains-screening.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-external-cap.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-external-de.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-external-have.xmi
http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference-external-ogc.xmi

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

LICENSES

PATENTS

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/report_issue.htm*).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from this URL:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the URL cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/report_issue.htm*.

1 Scope

1.1 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 that 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.

NIEM is currently defined in terms of the eXtensible Markup Language (XML), XML Schema (XSD), and the normative NIEM platform specifications that 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 UML modelers 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/.

1.2 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 MPD artifacts. 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 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.

1.3 NIEM-UML Profiles

Key components of NIEM-UML are the profiles used by modelers. The NIEM-UML Profile consists of four sub-profiles, as shown in Figure 1-1. Each sub-profile is a subset of UML 2.4 constructs that are extended by UML stereotypes. The subset identifies those NIEM v2.1 concepts for which an analogous representation exists in UML. Use of this subset ensures that a model produced by one user will be interpreted as expected by another user. The UML extensions define the NIEM concepts without an analogous representation in UML. All NIEM-UML models use the standard XMI exchange format specified for UML 2.4 and may exchange NIEM models between conforming UML tools.

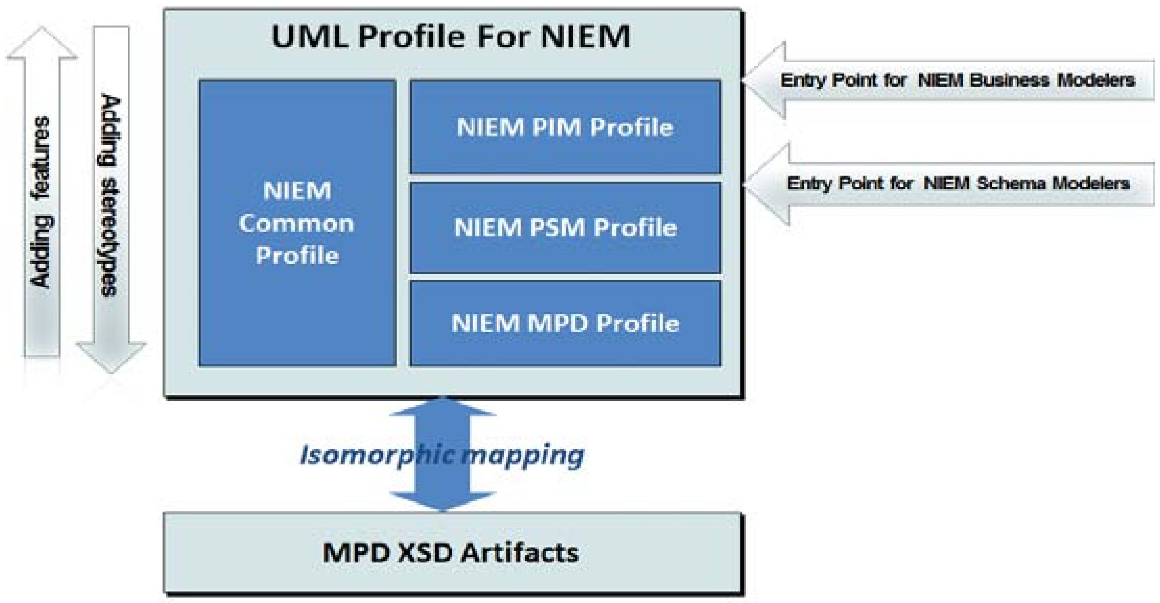


Figure 1.1 - Components of the NIEM-UML Specification

These sub-profiles have distinct purposes and relationships:

- The NIEM Platform Independent Model (PIM) Profile provides stereotypes that enable NIEM business modelers to model an information exchange in a technology agnostic way and create a NIEM PIM.
- The NIEM Platform Specific Model (PSM) Profile provides stereotypes that enable NIEM technical modelers – or, more precisely, NIEM schema modelers – to model the technical aspect of an information exchange represented in a NIEM PSM.
- The NIEM Common Profile, leveraged by both the PIM and PSM profiles, which contains the core stereotypes used to represent NIEM structures in UML.
- The Model Package Description (MPD) Profile provides stereotypes for modeling NIEM MPDs, which are the final artifacts representing a NIEM information exchange, based on either a PIM or PSM model.

As indicated in Figure 1.1, this structure for the NIEM-UML profile provides direct “entry points” for both NIEM modelers who are primarily business oriented and NIEM modelers who are primarily technically oriented. However, it also defines a clear relationship between these levels, allowing modelers to also move flexibly between them using a common set of profile concepts.

1.4 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.

1.5 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.

Note – The NIEM-UML Reference Vocabulary Library is currently provided consistent with the NIEM v2.1 release and for the domains contained under that release. The current version of this model library is the normative representation for the NIEM v2.1 reference vocabulary and should be used by NIEM-UML models based on that release. However, the NIEM PMO may provide updated models for future releases of NIEM. Since the definition of conforming NIEM models given in Clause 2 does not depend on the use of a specific version of the Reference Vocabulary Library, the use of future versions as released by the NIEM PMO does not affect the definition of conformance under this specification.

2 Conformance

2.1 Conformance Points

This specification defines the following conformance points (also referred to as conformance targets):

- NIEM Platform Independent Model (PIM)
- NIEM Platform Specific Model (PSM)
- NIEM Model Package Description (MPD) Model
- NIEM PIM to NIEM PSM transform
- NIEM PSM to NIEM-conforming XML schema transform
- NIEM MPD model to NIEM MPD artifact transform
- NIEM MPD artifact to NIEM MPD model transform

2.2 NIEM Platform Independent Model (PIM)

Sub clause 8.2 of this specification defines the NIEM PIM Profile. A NIEM PIM consists of a set of UML Packages to which this NIEM PIM Profile has been applied such that all the following hold:

- Each member of the set of UML Packages and each model element contained by those packages satisfies the constraints specified by the NIEM PIM Profile.
- Each member of the set of UML Packages and each model element contained by those packages to which a stereotype from the NIEM PIM has been applied satisfies the constraints specified by that stereotype.

Note – The NIEM PIM Profile imports the NIEM Common Profile, so the latter is also necessary in order to meet this conformance point.

2.3 NIEM Platform Specific Model (PSM)

Sub clause 8.3 of this specification defines the NIEM PSM Profile. A NIEM PSM consists of a set of UML Packages to which this NIEM PSM Profile has been applied such that the following hold:

- The NIEM PIM Profile has neither been applied to any member of the set of UML Packages nor to any model element contained by those packages.
- The profile application is "strict" as defined in UML 2.4 Superstructure, 18.3.7 and 18.3.8: each member of the set of UML Packages and each model element contained by those packages belongs to the UML subset specified by the NIEM PSM Profile.
- Each member of the set of UML Packages and each model element contained by those packages satisfies the constraints specified by the NIEM PSM profile.
- Each member of the set of UML Packages and each model element contained by those packages to which a stereotype from the NIEM PSM has been applied satisfies the constraints specified by that stereotype.

A NIEM PSM conforms to this specification only if a NIEM-conformant XML schema set may be successfully generated from it according to the rules of sub clause 9.3 of this specification and as further discussed in 2.6 below.

Note – The NIEM PSM Profile imports the NIEM Common Profile, so the latter is also necessary in order to meet this conformance point.

2.4 NIEM Model Package Description (MPD) Model

Sub clause 8.4 of this specification defines the Model Package Description Profile. A NIEM MPD model consists of a set of UML Packages to which this Model Package Description Profile has been applied and which import UML Packages to which the NIEM PIM Profile and/or the NIEM PSM Profile has been applied, such that the following hold:

- The imported UML Packages with the NIEM PIM Profile applied is a conforming NIEM PIM as defined in 2.2.
- The imported UML Packages with *only* the NIEM PSM Profile applied is a conforming NIEM PSM as defined in 2.3.
- Each member of the set of UML Packages with the Model Package Description Profile applied and each model element contained by those packages satisfies the constraints specified by the Model Package Description Profile.
- Each member of the set of UML Packages with the Model Package Description Profile applied and each model element contained by those packages to which a stereotype from the Model Package Description Profile has been applied satisfies the constraints specified by that stereotype.

2.5 NIEM PIM to NIEM PSM Transform

Sub clause 9.2 of this specification describes the NIEM PIM to NIEM PSM transformation rules. A NIEM PIM to NIEM PSM transform consists of a NIEM PIM and a NIEM PSM such that the NIEM PIM to NIEM PSM transformation rules, when applied to the NIEM PIM, produce the NIEM PSM.

2.6 NIEM PSM to NIEM-Conforming XML Schema Transform

Sub clause 9.3 of this specification describes the NIEM PSM to NIEM-conforming XML schema generation rules. A NIEM PSM to NIEM-conforming XML Schema transform consists of a NIEM PSM and a NIEM-conforming XML schema set (per [NIEM-NDR]) such that the NIEM PSM to NIEM-conforming XML schema generation rules, when applied to the NIEM PSM, produce an XML schema set that is validation-equivalent to the given schema set. A schema set *A* is *validation-equivalent* to a schema set *B* if and only if, for all XML instances *I, I* is valid against schema set *A* if and only if *I* is valid against *B*.

2.7 NIEM MPD Model to NIEM MPD Artifact Transform

Sub clause 9.4 of this specification describes the NIEM MPD model to NIEM MPD artifact generation rules. A NIEM MPD model to NIEM MPD artifact transform consists of a NIEM MPD model and a NIEM MPD (as specified in [NIEM-MPD]) such that the NIEM MPD model to NIEM MPD artifact generation rules, when applied to the NIEM MPD model, produce the NIEM MPD, where conformance of any generated NIEM-conforming XML schema included in the MPD is as defined in 2.6.

2.8 NIEM MPD Artifact to NIEM MPD Model Transform

Sub clause 9.5 of this specification describes the NIEM MPD artifact to NIEM MPD model reverse engineering rules. A NIEM MPD to NIEM MPD artifact model transform consists of a NIEM MPD (as specified in [NIEM-MPD]) and a NIEM MPD model such that the NIEM MPD to NIEM MPD artifact model reverse engineering rules, when applied to the NIEM MPD, produce the NIEM MPD model.

2.9 Tool Conformance

This specification defines tool conformance in terms of conformance points. A tool developer may assert that a given tool supports one or more of the conformance points defined in this specification as follows:

- The tool produces a NIEM PIM as described in 2.3.
- The tool produces a NIEM PSM as described in 2.4.
- The tool consumes a NIEM PIM and produces a NIEM PSM, such that it performs a NIEM PIM to NIEM PSM transform as described in 2.5.
- The tool consumes a NIEM PSM and produces a NIEM-conforming XML schema, such that it performs a NIEM PSM to NIEM-conforming XML schema transform as described in 2.6.

  NOTE - The NIEM PSM to NIEM-conforming XML schema generation rules as described in 9.3 may be formalized using QVT [QVT] (see also Annex C.1). The definition of this QVT as normative does not imply that implementations must execute QVT to conform to this specification. Implementations may use any means to transform a NIEM PSM to a NIEM-conforming XML schema set. Any such transform is considered conformant to this specification if it meets the requirements of this sub clause.

- The tool consumes a NIEM MPD model and produces a NIEM MPD, such that it performs a NIEM MPD model to NIEM MPD transform as described in 2.7.
- The tool consumes a NIEM MPD and produces a NIEM MPD model, such that it performs a NIEM MPD to NIEM MPD model transform as described in 2.8.

At some time in the future tools may be developed that can verify these assertions with some degree of confidence.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[MOF]	OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1, formal/2011-08-07 (http://www.omg.org/spec/MOF/2.4.1/PDF/)
[NIEM]	NIEM Reference Namespaces, Version 2.1 (http://release.niem.gov/niem/2.1/)
[NIEM-Conformance]	NIEM Conformance, Version 1.0 (http://reference.niem.gov/niem/specification/conformance/1.0/conformance-1.0.pdf)
[NIEM-MPD]	NIEM Model Package Description Specification, Version 1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/model-package-description-1.0.pdf)
[NIEM-NDR]	NIEM Naming and Design Rules (NDR), Version 1.3 (http://reference.niem.gov/niem/specification/naming-and-design-rules/1.3/niem-ndr-1.3.pdf) NIEM Type Augmentation Supplement to NDR 1.3, Version 1.0 (http://reference.niem.gov/niem/specification/naming-and-design-rules/1.3/type-augmentation/niem-type-augmentation.pdf)
[OCL]	OMG Object Constraint Language (OCL), Version 2.3.1, formal/2012-01-01 (http://www.omg.org/spec/OCL/2.3.1/PDF)
[QVT]	Meta Object Facility (MOF) Query/View/Transformation Specification, Version 1.1, formal/2011-01-01 (http://www.omg.org/spec/QVT/1.1/PDF)
[RFC2119]	Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997 (http://www.ietf.org/rfc/rfc2119.txt)

[UML]	OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, formal/ 2011-08-06 (http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF)
[XMI]	OMG MOF 2 XMI Mapping Specification, Version 2.4.1. formal/2011-08-09 (http://www.omg.org/spec/XMI/2.4.1/PDF)
[XMLNamespaces]	Namespaces in XML, World Wide Web Consortium 16 August 2006 (http://www.w3.org/TR/2006/REC-xml-names-20060816) Namespaces in XML Errata, 6 December 2002 (http://www.w3.org/XML/xml-names-19990114-errata)
[XMLSchemaDatatypes]	XML Schema Part 2: Datatypes Second Edition, W3C Recommendation (http://www.w3.org/TR/xmlschema-2/)
[XMLSchemaStructures]	XML Schema Part 1: Structures Second Edition, W3C Recommendation (http://www.w3.org/TR/xmlschema-1/)

4 Terms and Definitions

4.1 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.

4.2 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

5 Symbols

There are no symbols defined in this specification.

6 Additional Information

6.1 Acknowledgments

The following entities have played a significant role in driving the development of this specification:

- Submitters:
  - Microsoft
  - Model Driven Solutions
  - Visumpoint, LLC
- Government Stakeholders:
  - NIEM Program Management Office (PMO), and the NIEM Technical Architecture Committee (NTAC)

    - Office of the Program Manager for Information Sharing Environment (PM-ISE) www.ise.gov
    - Office of the Secretary of Defense
- Contributors:
    - Adaptive
    - Escape Velocity
    - Everware-CBDI
    - Georgia Tech Research Institute (GTRI)
    - SEARCH
    - TethersEnd Consulting

6.2 Proof of Concept

Proofs of Concept have been completed during the development of the PIM and the PSM profiles such that NIEM-UML models can be used to forward engineer validated MPDs and existing IEPDs have been able to be reverse engineered into NIEM-UML.

6.3 NIEM-UML Introduction and Concepts

6.3.1 Background

The U.S. Department of Justice (DOJ) and the U.S. Department of Homeland Security (DHS) initiated the National Information Exchange Model (NIEM) in 2005. Its early design was based on its predecessor, the Global Justice XML Data Model (GJXDM). Both programs recognized immediately that widespread adoption and use would require common vocabularies for information sharing and supporting software tools. Responding to a variety of urgent user needs, GJXDM software tools were developed by DOJ. As user needs evolved, these tools were adapted and expanded for NIEM. Since these beginnings substantial controlled NIEM vocabularies have been developed within the NIEM process, these vocabularies are the basis for multiple information sharing solutions.

Relatively rapid adoption of NIEM in the Justice, Public Safety, and other communities made it clear that governance and tool support would need to increase to keep pace. To leverage limited NIEM resources for more rapid expansion of software support, the NIEM Program Management Office (PMO) established an approach (outlined in the *NIEM High Level Tool Architecture*) that supports tool interoperability through standard open interfaces and well-defined import/export artifacts. This removes the need for an all-in-one tool, and allows both existing and new tools to support the functions of NIEM development processes. Existing tools can easily adapt to the interfaces and artifacts to provide support for the functions they do best. Standard imports, exports, and interfaces at key points in NIEM processes also facilitate tool interoperability. New tools can be built to directly support one or more functions, as well as interoperate with other tools. By publishing open interface specifications, the NIEM Program economically facilitates adaptation and interoperability of existing tools and encourages development of new NIEM-aware tools.

6.3.2 NIEM-UML Goals

Consistent with the NIEM *High Level Tool Architecture*, this NIEM-UML is a specification (under the OMG) designed to enable general use of UML and MDA tools to support the development and use of NIEM information exchanges and models. The primary intention of this specification is to enable existing UML tools to build standard UML representations of NIEM information exchanges and models, and to generate associated NIEM-conforming XML schemas and other artifacts. The following key considerations have been implemented in the specification:

- *Standards Based* - Enable leverage standards and standards based tools.
- *Simplicity* - Reduce complexity and lower the barrier for entry for NIEM business and technical modelers.
- *Reuse* - Facilitate reuse of NIEM models and as a result schemas.
- *Agility* - Enable the NIEM profile to be used with other standards, technologies and layers, if required.
- *Audience* - Allow audiences with different levels of knowledge of the NIEM technical concepts to create and use NIEM specifications.
- *Interoperability* - Ensure that a UML representation of a NIEM model produced by one developer can be interpreted as expected by another.
- *Completeness* - Ensure that a developer can produce a UML representation of any NIEM concept, including semantics, XML Schema structure, and metadata.
- *Practicality* - With minimal effort, an architect or developer can employ the profile in current UML development tools to develop a NIEM model.

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

6.3.3.1 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.

6.3.3.2 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 that 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).

6.3.3.3 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.

6.3.3.4 Implementing 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.

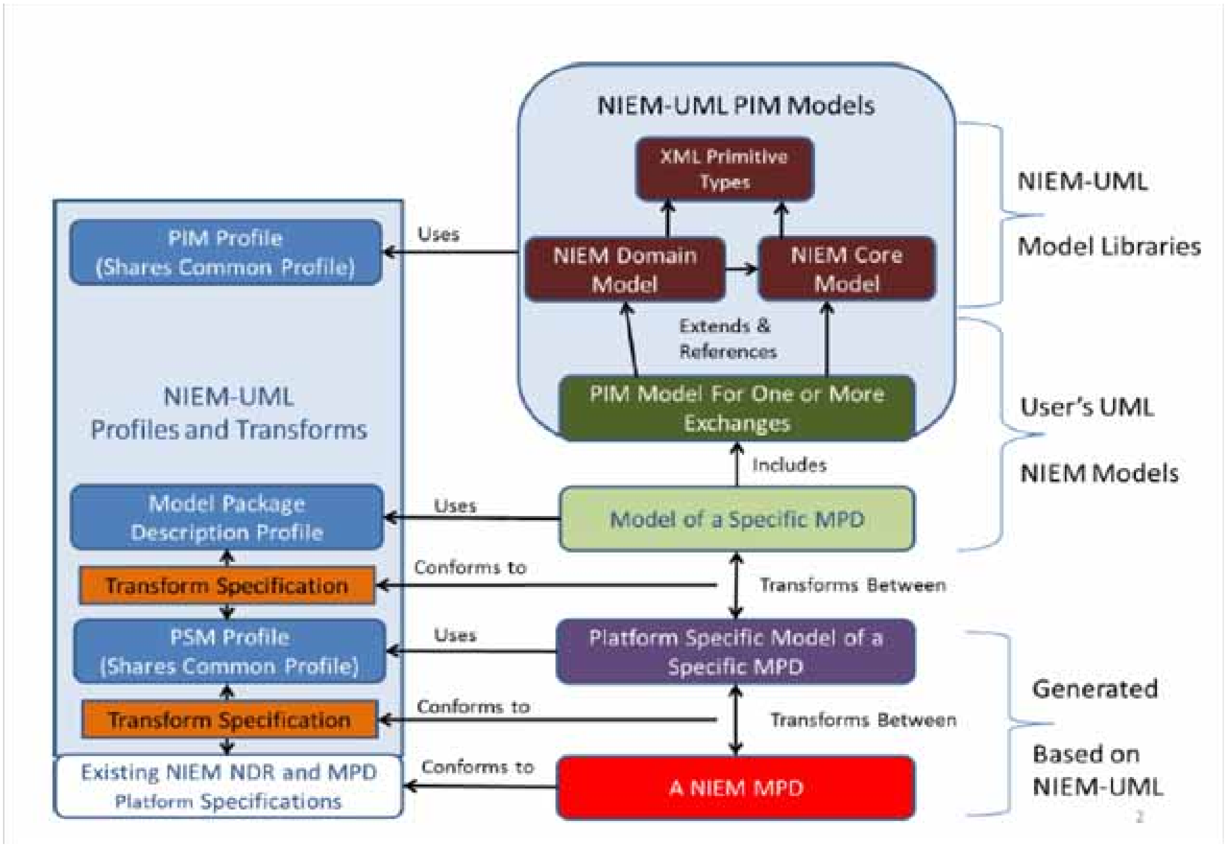


Figure 6.1 - Components of the NIEM-UML specification

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.

7 NIEM-UML Modeling Guide

7.1 Overview

7.1.1 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.

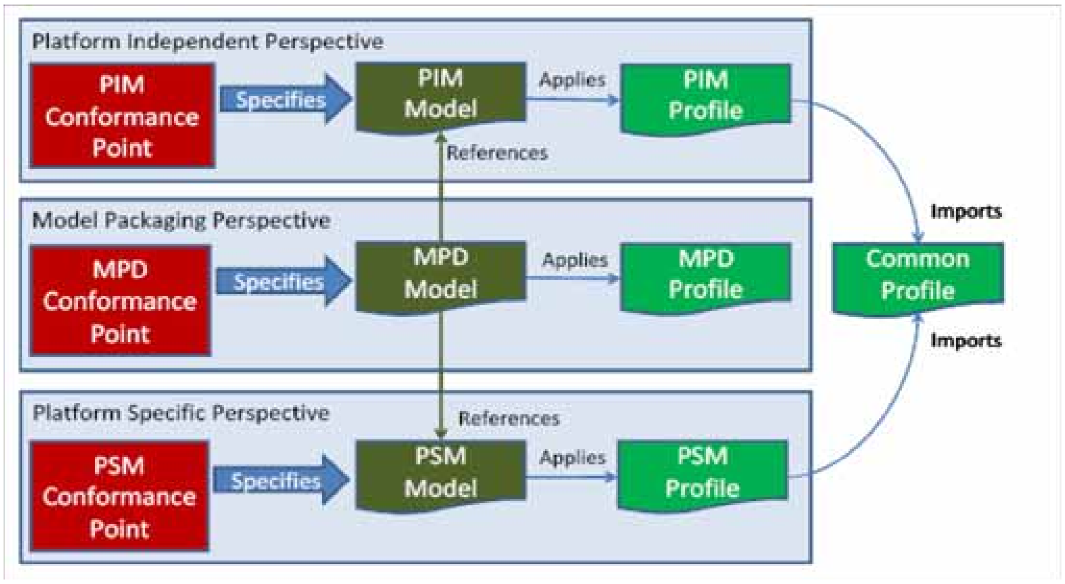


Figure 7.1 - Structure of the NIEM-UML Profile

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

7.1.2 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 mappings specified in Clause 9. 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 «Object Type»	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

	«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.
	«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.

Table 7.1 - Platform Independent Perspective Modeling Summary

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.
	«Subsets»	7.6.1 Subset and reference models	A subset realization establishes the relationship between a subset and reference element. Specializes <<References>>
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.

Table 7.1 - Platform Independent Perspective Modeling Summary

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.
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.

7.1.3 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	«Object Type»	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."
	«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.

Table 7.2 - Platform Specific Perspective Modeling Summary

	«Choice»	7.5.4Choice 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.
	«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 *primi tive 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 Restrictions»	XSD Restriction	Restrictions 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.

Table 7.2 - Platform Specific Perspective Modeling Summary

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.
	«XSDRestrictions»	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.
	«XSDSimpleContent»	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, the second is a simple type. If the complex type definition is a <<Restriction>> of another complex type definition with simple content, then the simple type defines the constraining facets of the `xsd:restriction` to the other complex type. Otherwise, the relationship is implemented in XML Schema through base attribute on the `xsd:extension` 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.

Table 7.2 - Platform Specific Perspective Modeling Summary

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.
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.

7.1.4 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.

7.2 Modeling Namespaces

7.2.1 Namespaces

7.2.1.1 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 of distinct names across all the symbol spaces of a namespace [NDR 9].

7.2.1.2 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 of 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 7.3)
Simple Type	Data Type (see 7.4)
Property Declaration	Property (see 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 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.

The namespace qualification of an XML name is done through the use of a prefix declared for the namespace URL within a specific XML document [XMLNamespaces]. The NIEM community has a number of conventional prefixes used for various standard NIEM namespaces. To accommodate this, the specific prefix to be used for the NIEM namespace associated with a <<Namespace>> package may be specified using the defaultPrefix package. The given prefix can only be considered a default, however, because of the possibility of conflicts if two packages are used with the same defaultPrefix specified. In the case of such a conflict, the actual prefixes used will be the defaultPrefix with a number appended to guarantee uniqueness.

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 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 subset, etc.

An «InformationModel» package provides the logical scoping for the NIEM naming of model elements representing NIEM components (see 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 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.

7.2.1.3 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 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.

7.2.1.4 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 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 7.5.2). Therefore, the names of the object type and all the properties are members of the identified NIEM namespace.

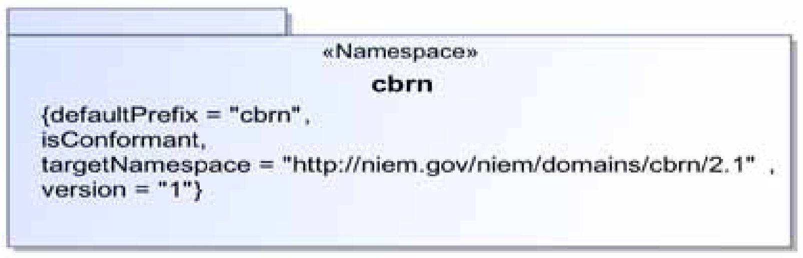


Figure 7.2 - Representation of a NIEM-conforming XML schema as a UML package

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>
```

7.2.2 NIEM Names

7.2.2.1 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.

7.2.2.2 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 sub clauses 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.

7.2.2.3 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.

7.2.2.4 Example

Figure 7.3 shows a class representing a NIEM object type (see 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.

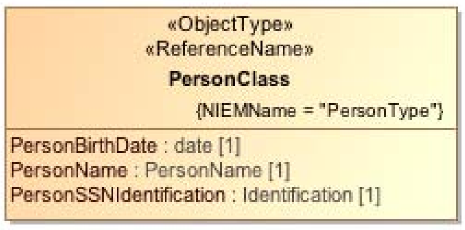

«ObjectType»
«ReferenceName»
PersonClass
{NIEMName = "PersonType"}
PersonBirthDate : date [1]
PersonName : PersonName [1]
PersonSSNIdentification : Identification [1]

Figure 7.3 - Specification of a NIEM name using the «ReferenceName» stereotype

7.3 Modeling Complex Types

7.3.1 Complex Types

7.3.1.1 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

7.3.1.2 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 sub clauses 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 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 7.3.3.
Association Type	Apply «AssociationType» to the class. (In a PIM, alternatively use an association class.) See 7.3.4.
Metadata Type	Apply «MetadataType» to the class. See 7.3.5.
Augmentation Type	Apply «AugmentationType» to the class. (In a PIM, alternatively use an «Augments» generalization.) See 7.3.6.
Adapter Type	Apply «AdapterType» to the class. See 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 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 sub clauses.

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 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.

The class may represent a Complex Type with Simple Content (CSC). In this case, the class may be the client of at most one <<XSDSimpleType>> realization. The supplier of the <<XSDSimpleType>> is a DataType. The class must not be both the special class in a generalization and the client of an <<XSDSimpleType>> realization. If the class is the client of a <<Restriction>> realization, then the supplier DataType defines the constraining facets of the Complex Type's `xsd:restriction`. If the class is not a client of a <<Restriction>> realization, then the supplier DataType is the base of the Complex Type's `xsd:extension`.

7.3.1.3 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, or the client of an <<XSDSimpleType>> 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.
- If the class is a client of a realization with the <<XSDSimpleType>> stereotype applied, and is not the client of a <<Restriction>> realization, then the base type definition for the complex type definition shall be the supplier of the <<XSDSimpleType>> realization and the complex type shall be an extension.
- If the class is a client of a realization with the <<XSDSimpleType>> stereotype applied, and is also the client of a <<Restriction>> realization, then the content of the `xsd:restriction` will include the constraining facets defined by the supplier DataType of the <<XSDSimpleType>> realization.

7.3.2 Object Types

7.3.2.1 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]

7.3.2.2 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 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 7.4 on modeling simple types).

7.3.2.3 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 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 will 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 an «XSDSimpleContent» realization, and is not the client of a <<Restriction>> realization, then the complex type definition mapped from the class shall be an extension having 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 the client of an «XSDSimpleContent» realization, and is also the client of a <<Restriction>> realization, then the complex type definition mapped from the class shall be a restriction whose constraining facets are mapped from the supplier of the realization.
  - If the supplier data type has the «ValueRestriction» stereotype applied, then the attribute values of the stereotype shall map to corresponding restriction facets.
  - If the supplier data type has the « XSDRepresentationRestriction » stereotype applied, then the attribute values of the stereotype shall map to corresponding restriction facets.
  - If the supplier is an Enumeration, then the enumeration literals shall map to corresponding restriction enumeration facets.
- If a class in a PSM with the «ObjectType» stereotype applied is *not* the client of an «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`.

7.3.2.4 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.

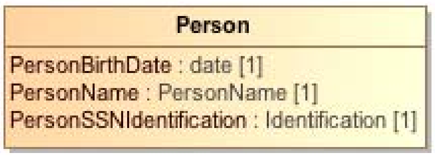

Person
PersonBirthDate : date [1]
PersonName : PersonName [1]
PersonSSNIdentification : Identification [1]

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.

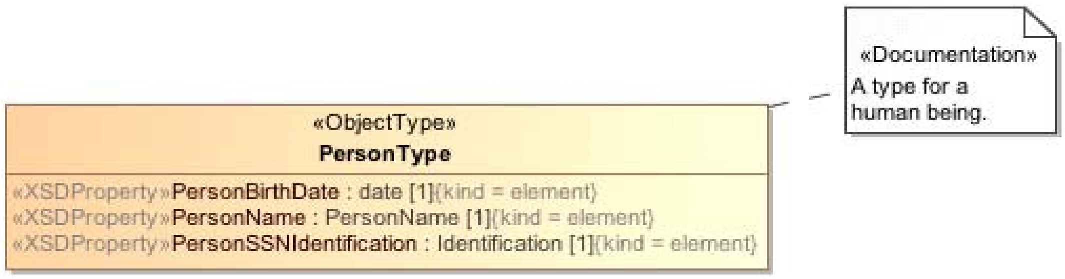

«Documentation»
A type for a human being.
«ObjectType»
PersonType
«XSDProperty»PersonBirthDate : date [1]{kind = element}
«XSDProperty»PersonName : PersonName [1]{kind = element}
«XSDProperty»PersonSSNIdentification : Identification [1]{kind = element}

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>
```

7.3.3 Role Types

7.3.3.1 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]

7.3.3.2 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 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 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 by either 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 above interpretation can't be explicitly represented in a PSM.)

7.3.3.3 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."

7.3.3.4 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.

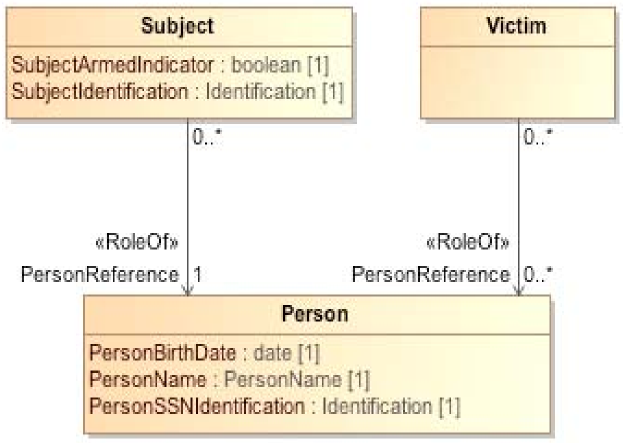

Subject
SubjectArmedIndicator : boolean [1]
SubjectIdentification : Identification [1]
Victim
0..*
0..*
«RoleOf»
PersonReference
1
«RoleOf»
PersonReference
0..*
Person
PersonBirthDate : date [1]
PersonName : PersonName [1]
PersonSSNIdentification : Identification [1]

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 corresponds 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.

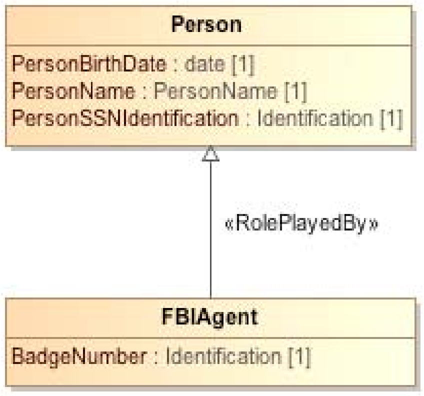

Person
PersonBirthDate : date [1]
PersonName : PersonName [1]
PersonSSNIdentification : Identification [1]
«RolePlayedBy»
FBIAgent
BadgeNumber : Identification [1]

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.

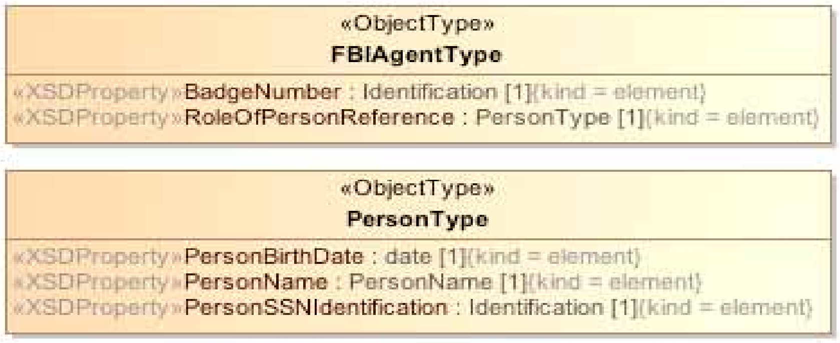

«ObjectType»
FBIAgentType
«XSDProperty»BadgeNumber : Identification [1]{kind = element}
«XSDProperty»RoleOfPersonReference : PersonType [1]{kind = element}
«ObjectType»
PersonType
«XSDProperty»PersonBirthDate : date [1]{kind = element}
«XSDProperty»PersonName : PersonName [1]{kind = element}
«XSDProperty»PersonSSNIdentification : Identification [1]{kind = element}

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>
```

7.3.4 Association Types

7.3.4.1 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]

7.3.4.2 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 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 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.

7.3.4.3 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`.

7.3.4.4 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 7.5.1.)

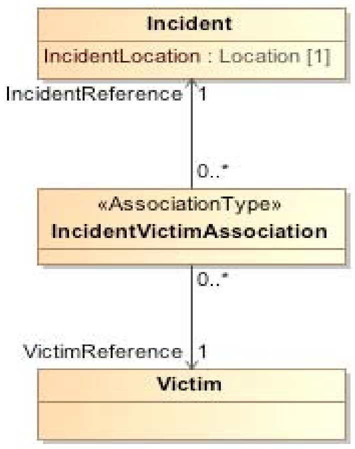


Figure 7.9 - Representation of a NIEM association type as a UML class

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 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.

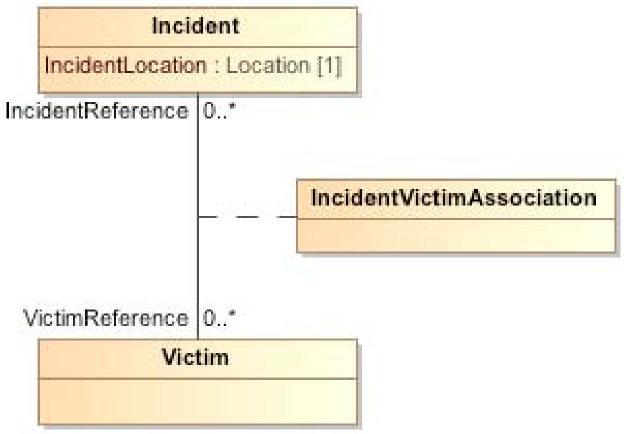


Figure 7.10 - Representation of a NIEM association type as a UML association class

PSM Representation

Figure 7.11 shows the PSM representation of the IncidentVictim association type.

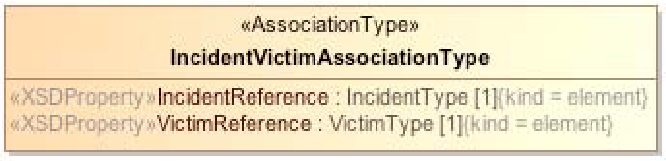

«AssociationType»
IncidentVictimAssociationType
«XSDProperty»IncidentReference : IncidentType [1]{kind = element}
«XSDProperty»VictimReference : VictimType [1]{kind = element}

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>
```

7.3.5 Metadata Types

7.3.5.1 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]

7.3.5.2 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 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 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.

7.3.5.3 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.

7.3.5.4 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.

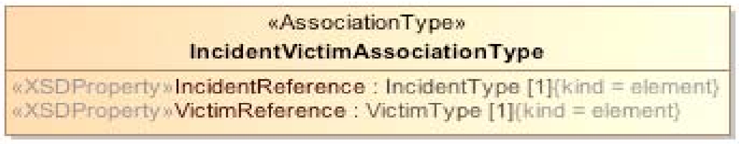

«AssociationType»
IncidentVictimAssociationType
«XSDProperty»IncidentReference : IncidentType [1]{kind = element}
«XSDProperty»VictimReference : VictimType [1]{kind = element}

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.

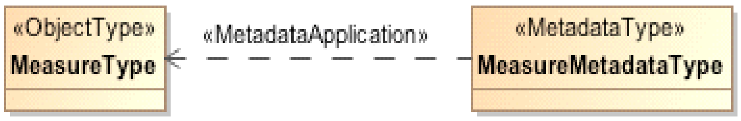


Figure 7.13 - Representation of a metadata application constraint as a UML dependency in a PSM or PIM

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>
```

7.3.6 Augmentation Types

7.3.6.1 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]

7.3.6.2 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 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 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 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.

7.3.6.3 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 «AugmentationApplication» 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 an «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 7.3.5.2).

7.3.6.4 Examples

PIM Representation

Figure 7.14 shows an augmentation type represented as a UML class with the «AugmentationType» stereotype.

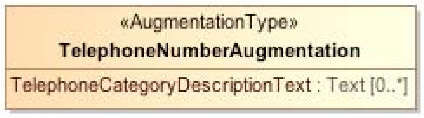

«AugmentationType»
TelephoneNumberAugmentation
TelephoneCategoryDescriptionText : Text [0..*]

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).

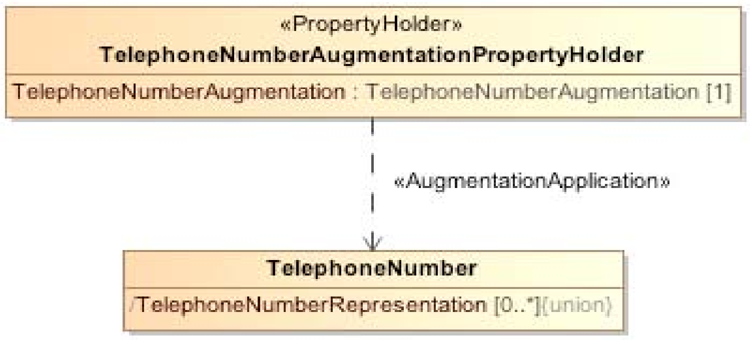

«PropertyHolder»
TelephoneNumberAugmentationPropertyHolder
TelephoneNumberAugmentation : TelephoneNumberAugmentation [1]
«AugmentationApplication»
TelephoneNumber
/TelephoneNumberRepresentation [0..*]{union}

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.

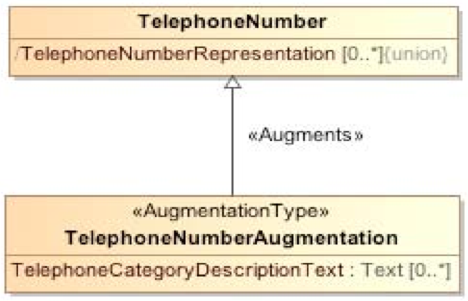

TelephoneNumber
/TelephoneNumberRepresentation [0..*]{union}
«Augments»
«AugmentationType»
TelephoneNumberAugmentation
TelephoneCategoryDescriptionText : Text [0..*]

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>
```

7.3.7 Adapter Types

7.3.7.1 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]

7.3.7.2 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 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 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.

7.3.7.3 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 7.3.2.3), except that the complex type definition mapped from the class has an `xsd:complexType/xsd:annotation/xsd:appInfo/i:ExternalAdapterTypeIndicator` element with value `true`.

7.3.7.4 Example

PSM Representation

Figure 7.17 shows the PSM representation of the AlertAdapterType class.

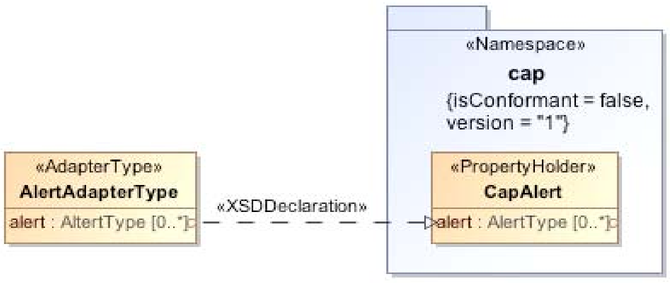


Figure 7.17 - Representation of an adapter type as a UML class

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>
```

7.4 Modeling Simple Types

7.4.1 Simple Types

7.4.1.1 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].)

7.4.1.2 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 sub clauses.

Table 7.7 - Simple Type Representation

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

7.4.1.3 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 a generalization - see 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.
- A PIM Enumeration, or a DataType with applied Stereotype «ValueRestriction» or «XSDRepresentationRestriction», and which derives from a DataType mapped to a PSM «ObjectType» shall map to a Class in the PSM with the «ObjectType» stereotype applied.
  - The PSM «ObjectType» shall be the client of a PSM «Restriction» Realization whose supplier is mapped from the base type of the PIM data type.
  - There shall be a new PSM DataType constructed, which depending upon the PIM DataType, will be:
    - A PSM Enumeration, with enumeration literals mapped from the PIM.
    - DataType stereotyped by «ValueRestriction» and populated with the facet tag values defined on the PIM «ValueRestriction».
    - DataType stereotyped by «XSDRepresentationRestriction», and populated with the facet tag values defined on the PIM «XSDRepresentationRestriction».
  - There shall be a new PSM «XSDSimpleContent» Realization constructed whose client is the PSM «ObjectType» and whose supplier is the PSM DataType.

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 an `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.
- If a data type in a PSM has the «XSDRepresentationRestriction» or «ValueRestriction» stereotype applied, or is an Enumeration, and it is the supplier of an «XSDSimpleContent» Realization whose client is also the client of a «Restriction» Realization, then the DataType is not mapped to a simple type. Instead, it is used to populate the constraining facet content of an `xsd:restriction`, as described in 7.3.2.2.

7.4.2 Primitive Types

7.4.2.1 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].)

7.4.2.2 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 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.

7.4.2.3 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.

7.4.2.4 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.

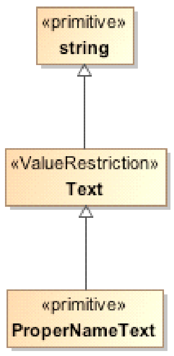


Figure 7.18 - Representation of primitive types in a PIM

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.

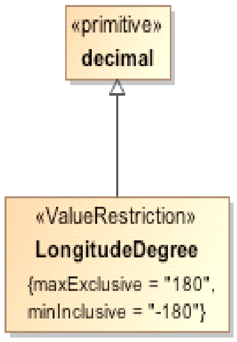


Figure 7.19 - Representation of a primitive type with a value restriction in a PIM

PSM Representation

Figure 7.20 shows the PSM representation of the primitive types modeled in Figure 7.18. A primitive type in a PSM must be stereotyped as a «ValueRestriction», so the ProperNameText type becomes an «ObjectType» class in the PSM.

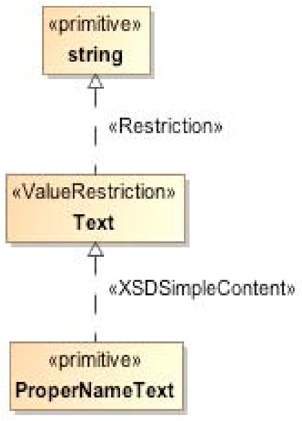


Figure 7.20 - Representation of primitive types in a PSM

Figure 7.21shows the PSM representation of the primitive type shown in Figure 7.19, which uses a «Restriction» realization instead of a generalization.

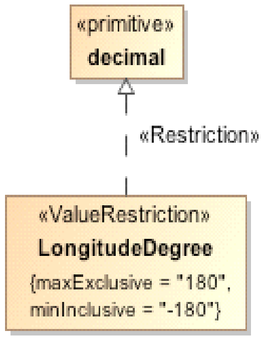


Figure 7.21 - Representation of a primitive type with a value restriction in a PSM

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>
```

7.4.3 Code Types

7.4.3.1 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]

7.4.3.2 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 7.4.1.2).

PSM

The base type of an enumeration in a PSM 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.

7.4.3.3 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.

7.4.3.4 Example

PIM Representation

Figure 7.22 shows the definition of the SupervisionLevelCode type as a UML enumeration.

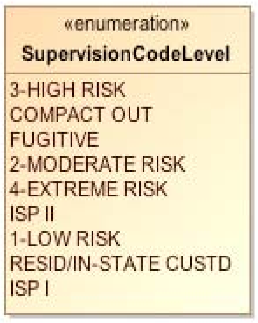


Figure 7.22 - A code type represented as a UML enumeration in a PIM

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.

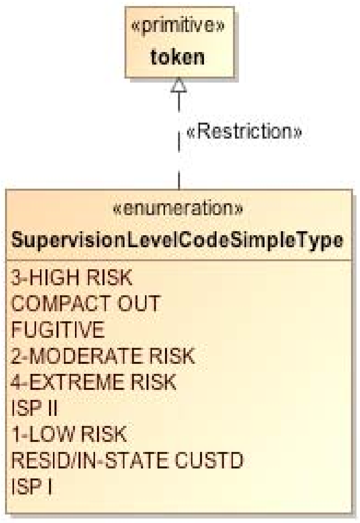


Figure 7.23 - A code type represented as a restriction in a PSM

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>
```

7.4.4 Unions

7.4.4.1 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].)

7.4.4.2 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.

7.4.4.3 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.

7.4.4.4 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.

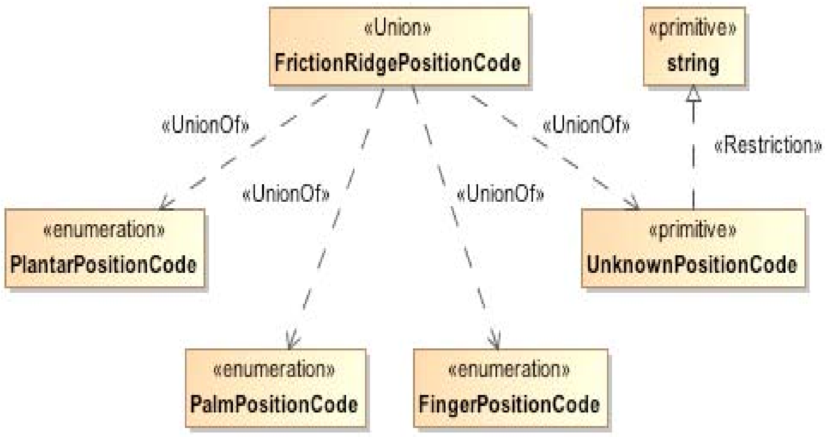


Figure 7.24 - Representation of a union as a UML data type

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>
```

7.4.5 Lists

7.4.5.1 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].)

7.4.5.2 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.

7.4.5.3 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.

7.4.5.4 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 7.5.1.2).

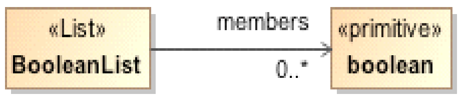


Figure 7.25 - Representation of a list in a PIM

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.

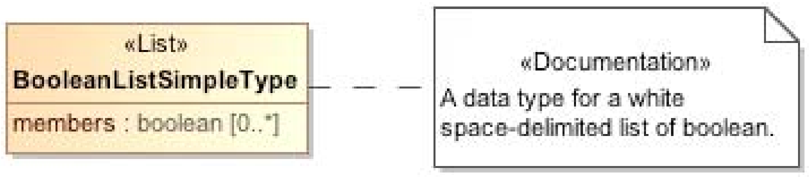


Figure 7.26 - Representation of a list in a PSM

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>
```

7.5 Modeling Properties

7.5.1 Properties

7.5.1.1 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]

7.5.1.2 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 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 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 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. Sub clauses 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, sub clause 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 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 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 sub clause 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.

7.5.1.3 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.

7.5.1.4 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.

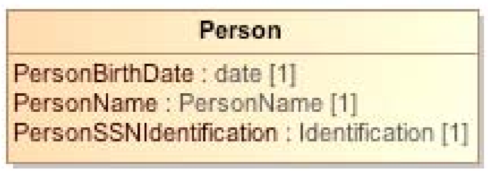

Person
PersonBirthDate : date [1]
PersonName : PersonName [1]
PersonSSNIdentification : Identification [1]

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.

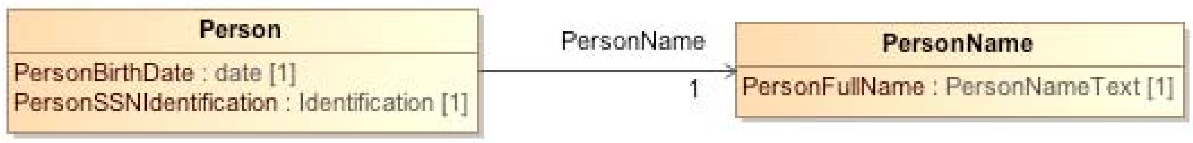

Person
PersonBirthDate : date [1]
PersonSSNIdentification : Identification [1]
PersonName
1
PersonName
PersonFullName : PersonNameText [1]

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.

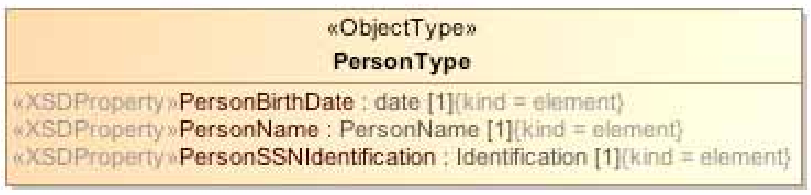

«ObjectType»
PersonType
«XSDProperty»PersonBirthDate : date [1]{kind = element}
«XSDProperty»PersonName : PersonName [1]{kind = element}
«XSDProperty»PersonSSNIdentification : Identification [1]{kind = element}

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"/>
```

7.5.2 Property Holders and Property References

7.5.2.1 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.

7.5.2.2 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 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 «Reference» realizations based on matching NIEM names (see 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.

7.5.2.3 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 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 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.

7.5.2.4 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.)

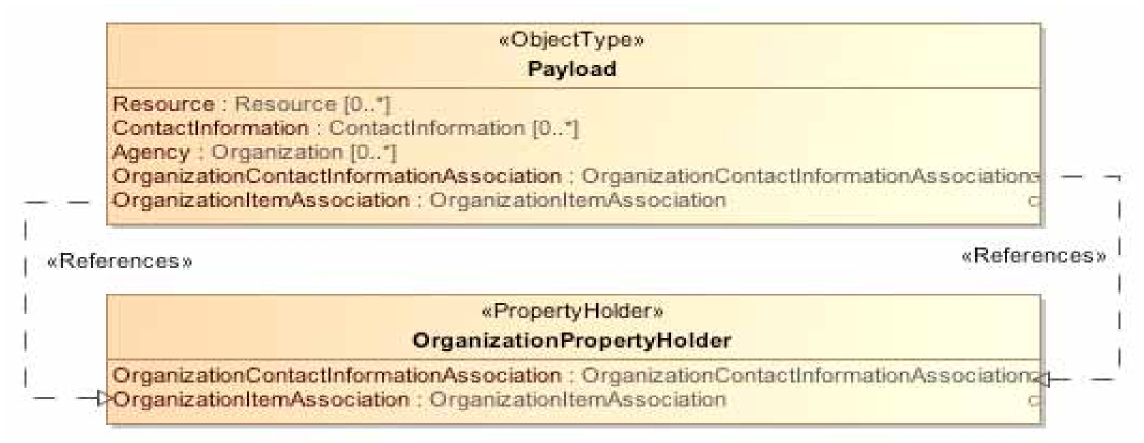

«ObjectType»
Payload
Resource : Resource [0..*]
ContactInformation : ContactInformation [0..*]
Agency : Organization [0..*]
OrganizationContactInformationAssociation : OrganizationContactInformationAssociation
OrganizationItemAssociation : OrganizationItemAssociation
«References»
«References»
«PropertyHolder»
OrganizationPropertyHolder
OrganizationContactInformationAssociation : OrganizationContactInformationAssociation
OrganizationItemAssociation : OrganizationItemAssociation

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.

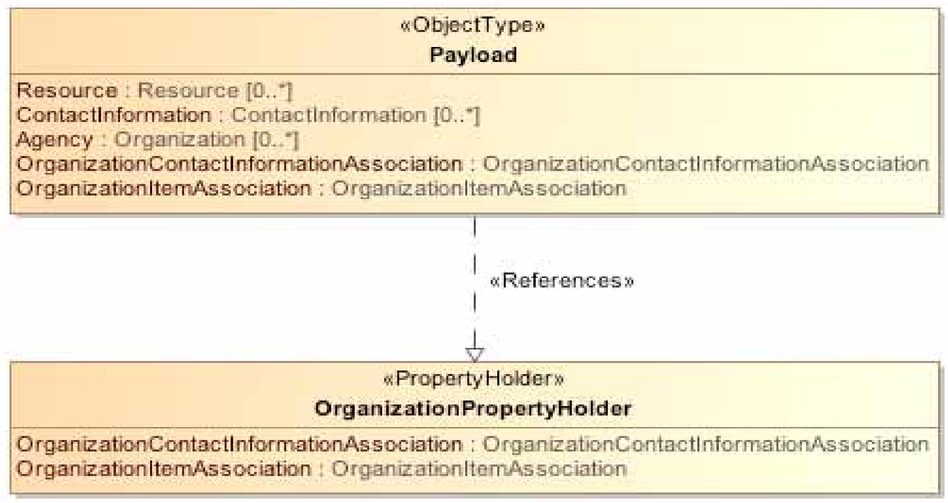

«ObjectType»
Payload
Resource : Resource [0..*]
ContactInformation : ContactInformation [0..*]
Agency : Organization [0..*]
OrganizationContactInformationAssociation : OrganizationContactInformationAssociation
OrganizationItemAssociation : OrganizationItemAssociation
«References»
«PropertyHolder»
OrganizationPropertyHolder
OrganizationContactInformationAssociation : OrganizationContactInformationAssociation
OrganizationItemAssociation : OrganizationItemAssociation

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>
```

7.5.3 Substitution Groups

7.5.3.1 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].)

7.5.3.2 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 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 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 «XSDAnyProperty» stereotype applied. It may not subset another «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.

7.5.3.3 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.

7.5.3.4 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.

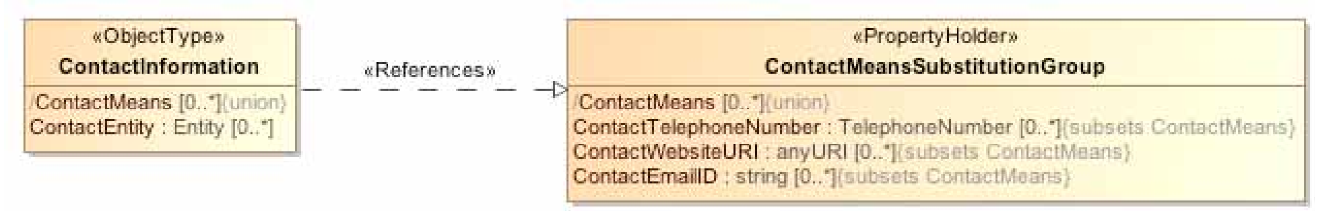

«ObjectType»
ContactInformation
/ContactMeans [0..*]{union}
ContactEntity : Entity [0..*]
«References»
«PropertyHolder»
ContactMeansSubstitutionGroup
/ContactMeans [0..*]{union}
ContactTelephoneNumber : TelephoneNumber [0..*]{subsets ContactMeans}
ContactWebsiteURI : anyURI [0..*]{subsets ContactMeans}
ContactEmailID : string [0..*]{subsets ContactMeans}

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 ContactMeansExtension and ContactMeansSubstitutionGroup is required in order to establish a subsetting context that allows ContactSkypeID to subset the ContactMeans head property declared in ContactMeansSubstitution Group.

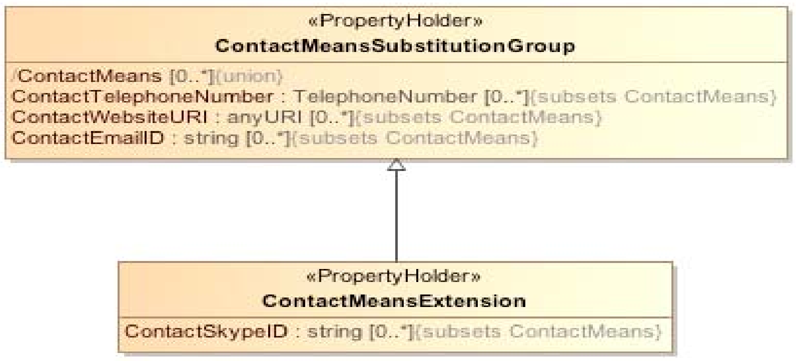

«PropertyHolder»
ContactMeansSubstitutionGroup
/ContactMeans [0..*]{union}
ContactTelephoneNumber : TelephoneNumber [0..*]{subsets ContactMeans}
ContactWebsiteURI : anyURI [0..*]{subsets ContactMeans}
ContactEmailID : string [0..*]{subsets ContactMeans}
«PropertyHolder»
ContactMeansExtension
ContactSkypeID : string [0..*]{subsets ContactMeans}

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>
```

7.5.4 Choice Groups

7.5.4.1 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].)

7.5.4.2 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.

7.5.4.3 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 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.)

7.5.4.4 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.)

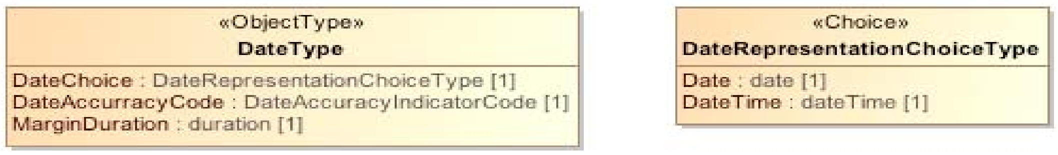

«ObjectType»
DateType
DateChoice : DateRepresentationChoiceType [1]
DateAccurracyCode : DateAccuracyIndicatorCode [1]
MarginDuration : duration [1]
«Choice»
DateRepresentationChoiceType
Date : date [1]
DateTime : dateTime [1]

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>
```

7.6 Packaging Models

7.6.1 Reference and Subset Models

7.6.1.1 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].)

7.6.1.2 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 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 7.5.2 to model a «Subsets» 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 «Subsets» 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 «Subsets» realization to the matching class in the supplier package.

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

In NIEM-XML and in a NIEM-UML-PSM the NIEM rules for a subset schema prohibit a property from being redefined with a subclass defined outside the scope of the reference model. A NIEM-PIM relaxes this constraint and allows a property in a subset to be defined as having a type that is a subclass of the type of the corresponding property in the reference model - such a subclass may be defined in any other NIEM conformant model such as an extension model or EIEM. The relaxing of this constraint is accomplished by using the reference model's definition for such properties in the generated subset schema and also generating a NIEM constraint schema that enforces the more restrictive subtype. This is the only use of constraint schema in NIEM-UML.

The existence of any implicit constraint schema within an IEPD implies the full complement of constraint schemas necessary to validate an XML exchange document instance. Each (implicit) constraint schema has the same namespace as its schema-set counterpart, is implicitly cataloged with a nature/purpose representing a constraint schema, and is implicitly grouped in a constraint-schema-set.

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 «Subsets» 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.

7.6.1.3 Mapping Summary

PIM Representation Mapping

- A realization between two «Namespace» packages in a PIM with the stereotype «Subsets» applied shall be considered equivalent to replacing the realization between the packages with multiple «Subsets» 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 <<Subsets>> from the class in the client package to the class in the supplier package.

The existence of an (implicit) constraint schema set within a PIM model is determined by the existence of a derived (subset) Property whose type is a subtype of the base (reference) Property.

If a constraint schema set exists, then each of the corresponding PIM schema-set <<InformationModel>>s is transformed into a PSM constraint <<Namespace>> Package. The mapping is as described in sub clauses 7.2 through 7.5 with the following caveats:

- Each schema-set <<InformationModel>> is reproduced as a PSM constraint <<Namespace>> package.
- For schema-set PSM <<Namespace>> packages, the type of a Property within a derived (subset) <<InformationModel>> is coerced to be the type of the corresponding Property in the base (reference) <<InformationModel>>.
- The constraint-schema-set is represented as a <<ModelPackageDescriptionFileSet>> Component, which is the supplier of a Usage from the <<ModelPackageDescription>> Component. The <<ModelPackageDescriptionFileSet>> Component tags for purposeURI and natureURI will be set to indicate that it is a constraint-schema-set.
- For each constraint <<Namespace>> transformed from an <<InformationModel>>, there is also a <<ModelPackageDescriptionFile>> Usage from the <<ModelPackageDescriptionFileSet>> Component to the constraint <<Namespace>>. The <<ModelPackageDescriptionFile>> tags for purpose and nature will be set to indicate that the <<Namespace>> is a constraint schema.

7.6.1.4 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 «Subsets» realization between the two packages.

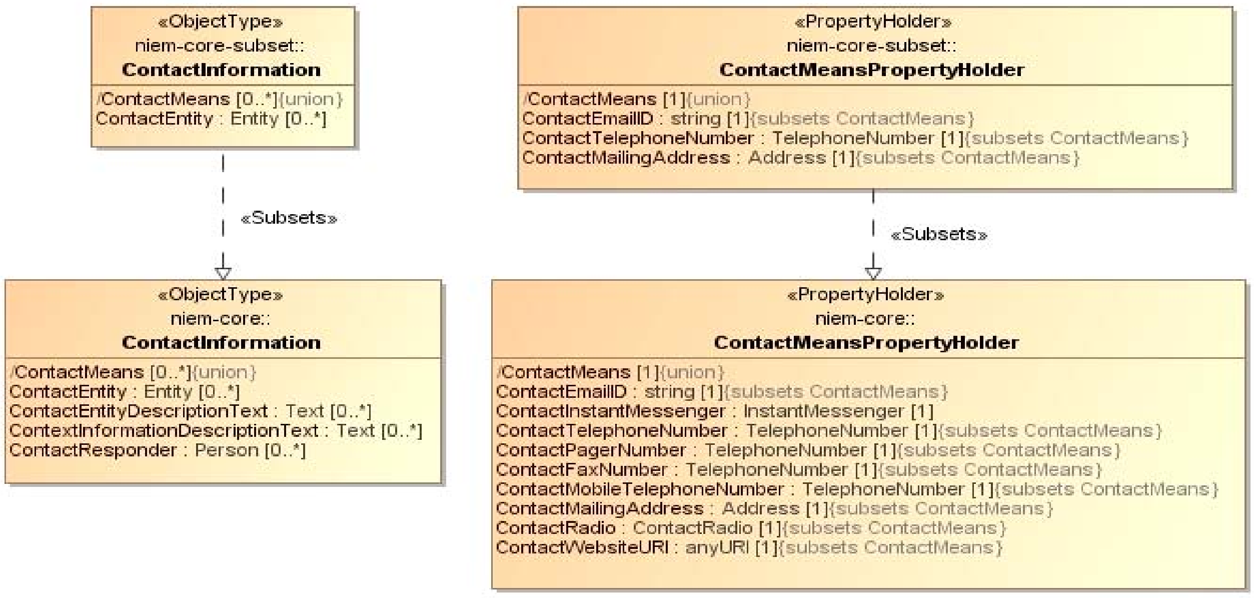

«ObjectType»
niem-core-subset::
ContactInformation
/ContactMeans [0..*]{union}
ContactEntity : Entity [0..*]
«Subsets»
«ObjectType»
niem-core::
ContactInformation
/ContactMeans [0..*]{union}
ContactEntity : Entity [0..*]
ContactEntityDescriptionText : Text [0..*]
ContextInformationDescriptionText : Text [0..*]
ContactResponder : Person [0..*]
«PropertyHolder»
niem-core-subset::
ContactMeansPropertyHolder
/ContactMeans [1]{union}
ContactEmailID : string [1]{subsets ContactMeans}
ContactTelephoneNumber : TelephoneNumber [1]{subsets ContactMeans}
ContactMailingAddress : Address [1]{subsets ContactMeans}
«Subsets»
«PropertyHolder»
niem-core::
ContactMeansPropertyHolder
/ContactMeans [1]{union}
ContactEmailID : string [1]{subsets ContactMeans}
ContactInstantMessenger : InstantMessenger [1]
ContactTelephoneNumber : TelephoneNumber [1]{subsets ContactMeans}
ContactPagerNumber : TelephoneNumber [1]{subsets ContactMeans}
ContactFaxNumber : TelephoneNumber [1]{subsets ContactMeans}
ContactMobileTelephoneNumber : TelephoneNumber [1]{subsets ContactMeans}
ContactMailingAddress : Address [1]{subsets ContactMeans}
ContactRadio : ContactRadio [1]{subsets ContactMeans}
ContactWebsiteURI : anyURI [1]{subsets ContactMeans}

Figure 7.35 - Representation of a subset model using «References» realizations

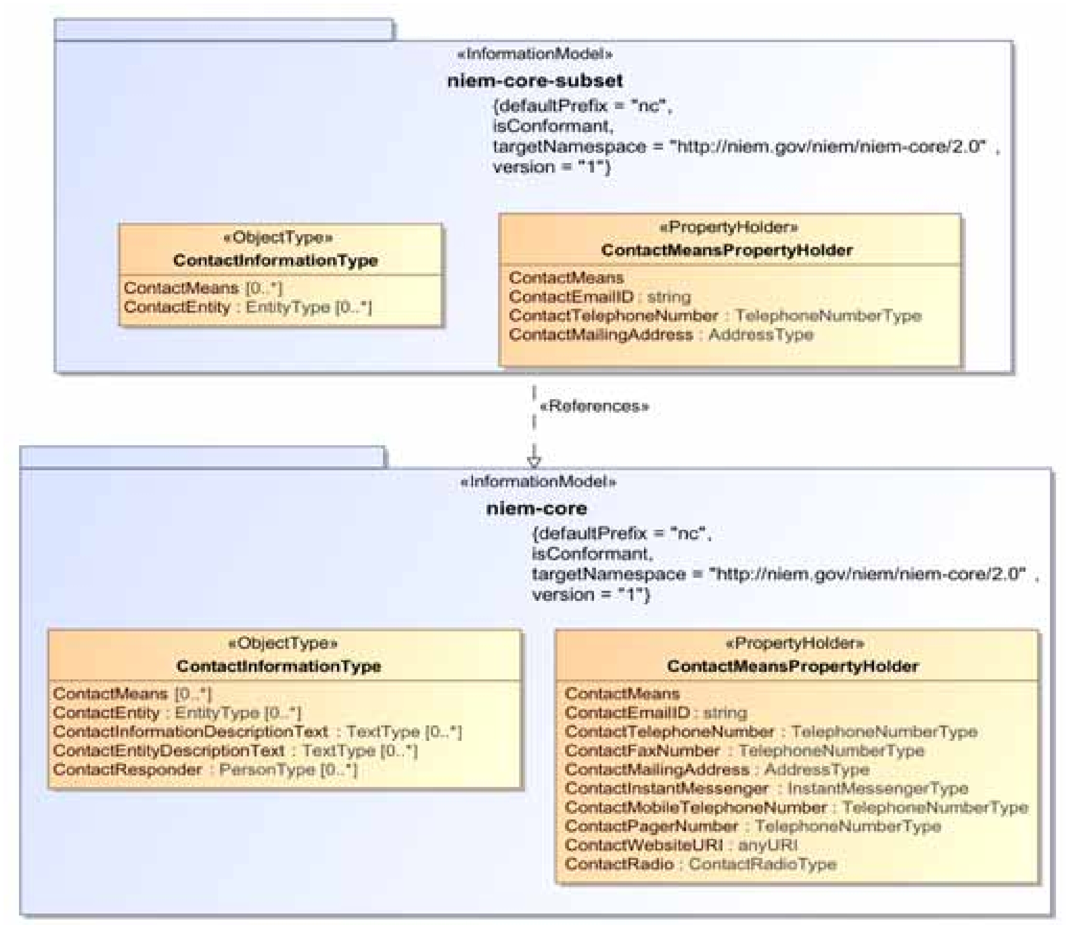

«InformationModel»
niem-core-subset
{defaultPrefix = "nc",
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0" ,
version = "1"}
«ObjectType»
ContactInformationType
ContactMeans [0..*]
ContactEntity : EntityType [0..*]
«PropertyHolder»
ContactMeansPropertyHolder
ContactMeans
ContactEmailID : string
ContactTelephoneNumber : TelephoneNumberType
ContactMailingAddress : AddressType
«References»
«InformationModel»
niem-core
{defaultPrefix = "nc",
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0" ,
version = "1"}
«ObjectType»
ContactInformationType
ContactMeans [0..*]
ContactEntity : EntityType [0..*]
ContactInformationDescriptionText : TextType [0..*]
ContactEntityDescriptionText : TextType [0..*]
ContactResponder : PersonType [0..*]
«PropertyHolder»
ContactMeansPropertyHolder
ContactMeans
ContactEmailID : string
ContactTelephoneNumber : TelephoneNumberType
ContactFaxNumber : TelephoneNumberType
ContactMailingAddress : AddressType
ContactInstantMessenger : InstantMessengerType
ContactMobileTelephoneNumber : TelephoneNumberType
ContactPagerNumber : TelephoneNumberType
ContactWebsiteURI : anyURI
ContactRadio : ContactRadioType

Figure 7.36 - Alternative Representation using «References» realizations between packages

7.6.2 Model Package Descriptions

7.6.2.1 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.

7.6.2.2 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 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.

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.

7.6.2.3 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.)

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 «ModelPackageDescription» 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.
- A dependency in an MPD model with the stereotype «ModelPackageDescriptionRelationship» applied, from a «ModelPackageDescription» component to a <<ChangeLogType>> Package shall map to a relationship recorded in the MPD mapped from the first component to a file component described by the ChangeLog XSD and populated by the metadata attributes corresponding to the ChangeLogType tags. The mechanism for populating the changelog file content is tool specific and not defined by this specification.

7.6.2.4 Example

Figure 7.37 is an example of the representation of two MPDs that share three files (namespaces) through «ModelPackageDescriptionFile» usage dependencies.

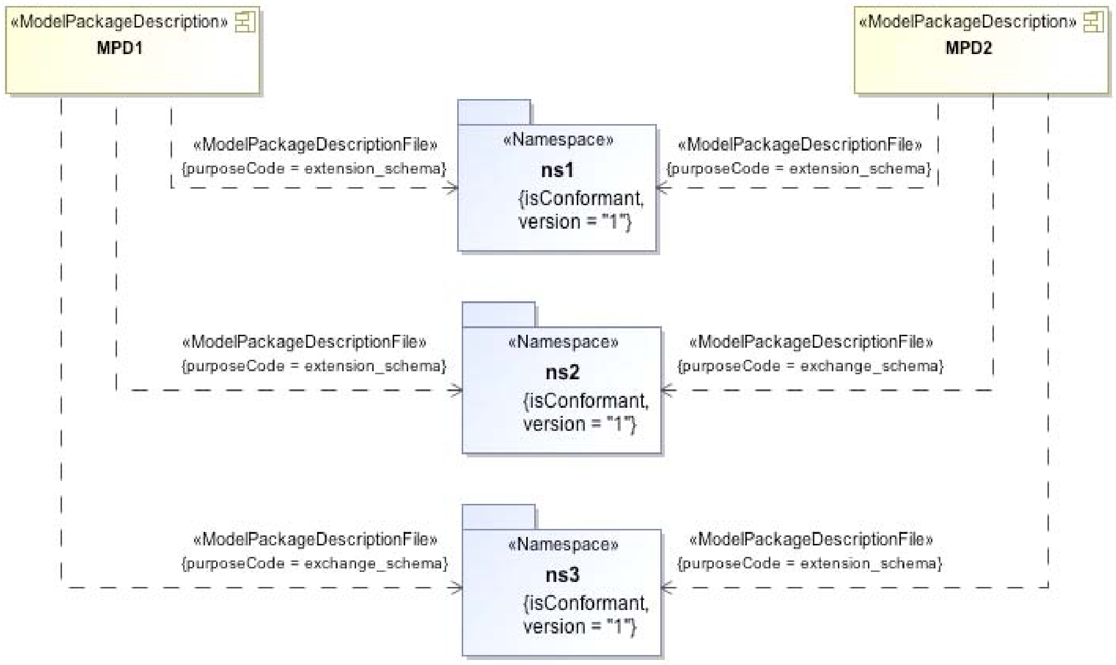

«ModelPackageDescription»
MPD1
«ModelPackageDescription»
MPD2
«ModelPackageDescriptionFile»
{purposeCode = extension_schema}
«Namespace»
ns1
{isConformant, version = "1"}
«ModelPackageDescriptionFile»
{purposeCode = extension_schema}
«ModelPackageDescriptionFile»
{purposeCode = extension_schema}
«Namespace»
ns2
{isConformant, version = "1"}
«ModelPackageDescriptionFile»
{purposeCode = exchange_schema}
«ModelPackageDescriptionFile»
{purposeCode = exchange_schema}
«Namespace»
ns3
{isConformant, version = "1"}
«ModelPackageDescriptionFile»
{purposeCode = extension_schema}

Figure 7.37 - Representation of two NIEM MPDs with included namespaces (files)

Figure 7.38 illustrates the definition of a change log.

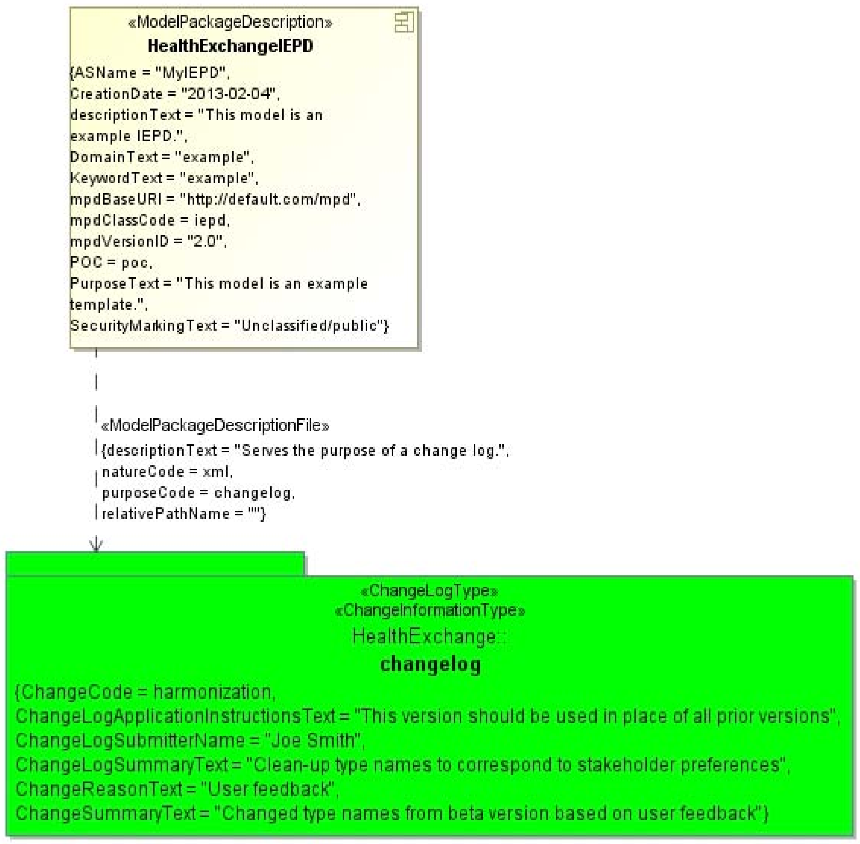

«ModelPackageDescription»
HealthExchangeIEPD
{ASName = "MyIEPD",
CreationDate = "2013-02-04",
descriptionText = "This model is an example IEPD.",
DomainText = "example",
KeywordText = "example",
mpdBaseURI = "http://default.com/mpd",
mpdClassCode = iepd,
mpdVersionID = "2.0",
POC = poc,
PurposeText = "This model is an example template.",
SecurityMarkingText = "Unclassified/public"}
«ModelPackageDescriptionFile»
{descriptionText = "Serves the purpose of a change log.",
natureCode = xml,
purposeCode = changelog,
relativePathName = ""}
«ChangeLogType»
«ChangeInformationType»
HealthExchange::
changelog
{ChangeCode = harmonization,
ChangeLogApplicationInstructionsText = "This version should be used in place of all prior versions",
ChangeLogSubmitterName = "Joe Smith",
ChangeLogSummaryText = "Clean-up type names to correspond to stakeholder preferences",
ChangeReasonText = "User feedback",
ChangeSummaryText = "Changed type names from beta version based on user feedback"}

Figure 7.38 - Example of a Change Log

Please see Figure A.29 for an additional example.

7.7 Detailed Modeling Design Rules

7.7.1 Design Rules Rationale

This non-normative sub clause describes the relationship between NIEM-UML models and NIEM-XSD based on the NIEM NDR (Naming and Design Rules) and MPD (Model Package Description) documents. A detailed understanding of the constraints on NIEM-UML compliant models as they relate to NIEM rules and generated schema will be of interest to tool builders and sophisticated NIEM-UML modelers.

7.7.2 Simple Restrictions

7.7.2.1 Background

Within an XML Schema, every type definition (except the distinguished ur-type definition) is either a restriction or an extension of some other type definition. A simple type must always be a restriction of another simple type. A complex type is either a restriction of another complex type or an extension of either a complex type or a simple type. A type definition used as the basis for a restriction or extension is known as the base type definition. A complex type which extends a simple type is a complex type with simple content. A complex type whose base type is a complex type with simple content is also a complex type with simple content. A complex type with simple content which restricts another complex type specifies the restricted value space using the same facets as used when a simple type restricts a simple type, and is subject to the same constraints: the value space of the restricting type must be within the value space of the base type.

The NIEM NDR prohibits a reference schema from using a restriction between complex types, but enables other types of schema to use restrictions between complex types. A complex type with simple content is always a NIEM Object Type and must always contain the attribute group structures:SimpleObjectAttributeGroup.

The rules for restriction are defined in detail within the XML Schema Specification. Facets are used to specify various forms of restrictions on a value space. Basically, the facets defined for a restriction must be within the value space defined by the base type. The applicability of facets is dependent upon the underlying XML Primitive Type.

7.7.2.2 Representation

Common

Facets are modeled using one of three mechanisms:

1. The whiteSpace facet is represented via tag on <<XSDRepresentationRestriction>>, which may be applied to a DataType.

2. Enumeration facets are represented as EnumerationLiterals on an Enumeration.

3. All other facets are represented as tags on <<ValueRestriction>> Stereotype, which may be applied to a DataType.

The mechanisms may be combined. For example, an Enumeration may have a <<ValueRestriction>> Stereotype applied to enable specification of both enumeration facets and other types of facets.

The representation for a restriction between types is described in 7.3 Modeling Complex Types and 7.4 Modeling Simple Types. When using a <<Restriction>> Realization to represent restrictions, it is possible to restrict the value space of the representation for a simple type, or a complex type with simple content, based on any combination of facet representations.

When a restriction is modeled between simple types or between complex types with simple content, then the validity of the application of facets, and their values, are subject to the constraints defined by the XML Schema Specification for restriction. The model is not well formed if the XML Schema Specification validity constraints are not satisfied.

PIM

PIM representations for modeling simple restrictions are described in 7.3 and 7.4.

PSM

PSM representations for modeling simple restrictions are described in 7.3 and 7.4.

7.7.2.3 Mapping Summary

PIM to PSM Mapping

PIM to PSM mappings are described in 7.3 and 7.4.

PSM to XML Schema Mapping

PSM to XML Schema mappings are described in 7.3 and 7.4.

7.7.3 Complex Restrictions

7.7.3.1 Background

Within an XML Schema, a restriction of a complex type with complex content is subject to the derivation validity defined by the XML Schema Specification for a restriction between complex types with complex content. Derivation validity includes, but is not limited to:

- Constraints on the ordering of elements.
- Constraints on the cardinality of elements.
- Constraints on the name/namespace of included components.
- Constraints on the derivation of a wildcard.
- Constraints on the use of substitution group elements. In effect, there can be only one substitutable element for a base type element. Depending upon whether or not the derived restriction element itself has substitutable elements:
  - If substitutable, then the cardinality of the derived restriction element must be within the bounds of the base element.
  - Otherwise, the cardinality of the derived restriction element must be exactly 1 and the base element must have a cardinality range that includes 1.

The NIEM NDR prohibits a reference schema from using a restriction between complex types, but enables other types of schema to use restrictions between complex types. In addition to the schema constraints related to restriction, the NIEM NDR requires the result of a restriction to include the attribute group structures:SimpleObjectAttributeGroup.

7.7.3.2 Representation

Common

The representation for a restriction between complex types is described in 7.3.

When a restriction is modeled between complex types with complex content, then the validity of the content of the derived type is subject to the constraints defined by the XML Schema Specification for restriction between complex types with complex content. The model is not well formed if the XML Schema Specification validity constraints are not satisfied.

PIM

PIM representations for modeling complex restrictions is described in 7.3.

PSM

PSM representations for modeling complex restrictions is described in 7.3.

7.7.3.3 Mapping Summary

PIM to PSM Mapping

PIM to PSM mappings is described in 7.3.

PSM to XML Schema Mapping

PSM to XML Schema mappings is described in 7.3.

7.7.4 Subsetting Constraints

7.7.4.1 Background

The NIEM MPD defines a subset schema as a schema which is derived from a reference schema, has the same namespace as the reference schema, is a strict subset of the reference schema (i.e., does not provide new definitions or declarations of schema components), and conforms to the constraints expressed within the reference schema (e.g., cardinality, value spaces, type, etc.). The primary reasons for subsetting are to reduce IEPD size and complexity and to focus constraints.

The fundamental rule for schema subsets is: any XML instance that validates against a correct NIEM schema subset will always validate against the entire NIEM reference schema set from which that subset was created.

The NIEM MPD defines an **Enterprise Information Exchange Model (EIEM)** as an MPD that contains NIEM-conforming schemas that define and declare data components to be consistently reused in the IEPDs of an enterprise. An EIEM is a collection of schemas organized into a collection of subset schemas and one or more extension schemas. An information sharing enterprise creates and maintains an EIEM. The same enterprise authors IEPDs by reusing its EIEM content instead of re-subsetting NIEM reference models and/or re-creating extension models. Part of the reuse process includes further subsetting eiem-defined subsets as well as subsetting eiem-defined exchange models.

Thus, the process of subsetting applies to not only NIEM reference models, but also subset models and exchange models. The fundamental rule for subsetting remains the same: any XML instance that validates against a correctly subsetted schema will always validate against the schemas from which the schema was derived.

7.7.4.2 Representation

Common

A subsetted model is represented by a <<Subsets>> Realization from the (client) subsetted model to the (supplier) base model. Note that for backwards compatibility a <<References>> Realization may also be used between information models. A subsetted model is subject to the following constraints:

- The namespace must be the same as the base model.
- If the base model is a reference model, then the subsetted model must be a subset model. Otherwise, the subsetted model must be the same kind as the base model (i.e., a subset model or an extension model).
- The subset model may only include components that are defined in the base model.
- Any simple or complex restrictions in the subset model must be the same or more restrictive than those defined in the base model.
- Any “business-rules” defined in the subset model must be the same or more restrictive than those defined in the base model.

- Any abstract component in the base model must be abstract in the subset model. A concrete component in the base model may be declared abstract in the subset model.
- A Property that is not nillable in the base model must not be nillable in the subset model. A Property that is nillable in the base model may be declared not nillable in the subset model.
- Complex types within a subset model must conform with the base model. This includes:
  - A property may be removed only if the base property has a cardinality range that includes 0.
  - A subset property cardinality must be within the inclusive range of the base property cardinality.
  - Ordering of subset elements must be the same as the base element order.
  - The NIEM name/namespace of included properties must be identical between subset and base models.
  - The type of properties in a subset model must be the same type, or subtype, as the corresponding property in the reference model. Note that for a provisioned subset schema, the type of a property in a subset schema must be the same as the type of a property in a reference schema. If a subset schema is modeled with a property having a subtype of the reference model property type, then it implicitly requires provisioning of constraint schemas to represent the subtype constraint.
  - The aggregation kind must both be none or both be an aggregate.
  - An element in the base model may have a corresponding subset element which has substitution groups in the subset model. In this case, subject to cardinality and unique particle attribution constraints, a decomposition of the base element may be defined. The decomposition allows for an ordered sequence of substitutable elements to be defined in the subset as a replacement for the single element defined in the base model. Each substitutable element may have its own cardinality bound; the sum of cardinalities must be within the bounds of the base element cardinality. The order and cardinality of the replacement sequence must conform to XML Schema constraints related to unique particle attribution.
  - Constraints on the derivation of a wildcard. A wildcard, subject to cardinality, unique particle attribution, and namespace constraints, may be decomposed in the subset model. The decomposition allows for an ordered sequence of elements to be defined in the subset as a replacement for the single wildcard defined in the base model. Each element may have its own cardinality bound; the sum of cardinalities must be within the bounds of the base element cardinality. The order and cardinality of the replacement sequence must conform to XML Schema constraints related to unique particle attribution. The namespace of each element must conform to namespace constraints specified by the wildcard (if any).

The subset model must conform to the basic principle that any instance of an exchange document that is valid for the subset model is also valid (in the context of the exchange) for the base model. The model is not well formed if it is possible to define an instance that is valid for the subset model but not valid for the base model.

PIM

A subsetting model may be represented in the PIM as an <<InformationModel>> that has a <<Subsets>> Realization to another <<InformationModel>> (as supplier of the Realization). The two <<InformationModel>>s must have the same namespace. If the defaultPurpose of the base model is “reference,” then the subsetting model must have a defaultPurpose of “subsets;” otherwise, the defaultPurpose must be the same for both models.

Additional information related to representing an <<InformationModel>> is described in 7.6.

PSM

The PSM does not include a representation for subsetting models.

7.7.4.3 Mapping Summary

PIM Representation Mapping

PIM representation is described in 7.6.

7.7.5 Constraint Models

7.7.5.1 Background

The NIEM MPD defines a constraint schema as an artifact that is used to express business rules for a class of XML documents. It is a schema that adds additional constraints to NIEM-conformant instances. A constraint schema set is a set of related constraint schemas that work together, such as a constraint schema set built by adding constraints to a schema subset. A base schema set is the set of XML schemas that defines the information exchange or model in the MPD. The base schema set may incorporate NIEM-conformant reference schemas, subset schemas, extension schemas, exchange schemas, as well as schemas from an external non-NIEM-conformant source.

A constraint schema set validates additional constraints imposed on an XML instance only after it is known to be NIEM-conforming (i.e., has been validated by reference schemas, subset schemas, extension schemas, and/or exchange schemas). Constraint schema validation is a second-pass validation that occurs independently of and after conformance validation.

According to the NIEM NDR, an XML Schema document is a NIEM-conformant schema if and only if it is a reference schema, a subset schema, an extension schema, an exchange schema, or a constraint schema. Thus, a constraint schema is a NIEM-conformant schema and indeed is explicitly an XML Schema. However, it does not have as many rules as other forms of NIEM-conformant schemas.

A constraint schema must have the same namespace as the schema for which it is adding constraints. Due to its role in second-pass validation, a constraint schema must be a member of a constraint schema set, a set of schemas that together provide the second-pass validation capability. According to the MPD, constraint schemas and constraint schema sets are completely distinct from the base schema set. There is necessarily a one to one correspondence between the members of a constraint schema set and the IEPD schema set. The MPD suggests that a common practice for creating an IEPD or EIEM constraint schema set is to start with a valid set of NIEM subset, extension, and exchange schemas, and modify this set to further restrict the class of XML documents (IEPs) that will validate with this constraint set. Using this suggested common practice yields schemas that share some common characteristics with subsetted models, namely that XML instances that validate against a correctly constrained schema set will validate against the schema set from which it was constrained (with an exception regarding the use of xsi:type="?"). Even though the constraint schema set is progressively refined from an IEPD schema set, the focus is generally on a few constraints that cannot be expressed within the NIEM subset conformance framework. Thus, even with a few addition constraints, the constraint schema set may be large, and it may be difficult to maintain synchronization with the underlying IEPD schema set.

7.7.5.2 Representation

Common

A constraint model is subject to all of the constraints of a subsetting model, with the following exceptions:

- The type of a Property may be refined to be a subtype of the type specified in the base model.

The constraint model should conform to the basic principle that any instance of an exchange document that is valid for the constraint model is also valid (in the context of the exchange) for the base constrained model (except for xsi:type="?"). The model is not well formed if it is possible to define an instance which is valid for the constraint model but not valid for the base constrained model.

PIM

Within a PIM derived <<InformationModel>> for an IEPD, it is allowed to specify the type of a Property to be a subtype of the type specified in the base <<InformationModel>>. Such a derived <<InformationModel>> implicitly represents:

- A schema within the IEPD base schema-set wherein the type of the Property is the type specified in the base <<InformationModel>>.
- A constraint schema within an IEPD constraint-schema-set wherein the type of the Property is as specified in the derived <<InformationModel>>.

Furthermore, the existence of any implicit constraint schema within an IEPD implies the full complement of constraint schemas necessary to validate an XML exchange document instance. Each (implicit) constraint schema has the same namespace as its schema-set counterpart, is implicitly cataloged with a nature/purpose representing a constraint schema, and is implicitly grouped in a constraint-schema-set.

Additional information related to representing an <<InformationModel>> is described in 7.6.

PSM

In the PSM, a constraint model is a <<Namespace>> Package. If there are any constraint models within an IEPD, then there must be a constraint <<Namespace>> Package for every <<Namespace>> in the corresponding IEPD schema set. There must be a <<ModelPackageDescriptionFileSet>> Component whose natureURI/purposeURI indicates a constraint-schema-set and which is the supplier of a Usage from the IEPD <<ModelPackageDescription>> Component. There is a <<ModelPackageDescriptionFile>> Usage from the client <<ModelPackageDescriptionFileSet>> Component to the supplier constraint <<Namespace>> Package. The natureURI and purposeURI tags within the <<ModelPackageDescriptionFile>> must indicate that the <<Namespace>> Package is a constraint model.

Each constraint <<Namespace>> Package must be a completely populated representation of all components defined within the constrained <<Namespace>> Package. Some of the components within the constraint <<Namespace>> may be restrictions of corresponding base schema set components, subject to the conditions for constraining described in the Common representation of this clause.

7.7.5.3 Mapping Summary

PIM to PSM Mapping

The existence of a constraint schema set within a PIM model is determined by the existence of a derived Property whose type is a subtype of the base Property.

If a constraint schema set exists, then each of the corresponding PIM schema-set <<InformationModel>>s is transformed into a PSM constraint <<Namespace>> Package. The mapping is as described in sub clauses 7.2 through 7.6 with the following caveats:

- Each schema-set <<InformationModel>> is reproduced as a PSM constraint <<Namespace>> package.
- For schema-set PSM <<Namespace>> packages, the type of a Property within a derived <<InformationModel>> is coerced to be the type of the corresponding Property in the base <<InformationModel>>.

- The constraint-schema-set is represented as a <<ModelPackageDescriptionFileSet>> Component, which is the supplier of a Usage from the <<ModelPackageDescription>> Component. The <<ModelPackageDescriptionFileSet>> Component tags for purposeURI and natureURI will be set to indicate that it is a constraint-schema-set.
- For each constraint <<Namespace>> transformed from an <<InformationModel>>, there is also a <<ModelPackageDescriptionFile>> Usage from the <<ModelPackageDescriptionFileSet>> Component to the constraint <<Namespace>>. The <<ModelPackageDescriptionFile>> tags for purpose and nature will be set to indicate that the <<Namespace>> is a constraint schema.

PSM to XML Schema Mapping

PSM to XML Schema mappings are unchanged from those described in sub clauses 7.2 through 7.6.

7.7.6 Business Rules

7.7.6.1 Background

The NIEM MPD defines a business rule as an artifact used to document constraints beyond the capability of NIEM and XML Schema; it may be used to validate or verify that such constraints are satisfied.

As an alternative to constraint schemas, NIEM also allows other methods that do not use XML Schema, such as [**ISO-Schematron**] or other language methods. However there are currently no normative rules for how these techniques should be employed in NIEM IEPDs or EIEMs. BIECs in particular may have additional business rules in constraint schemas. A normative NIEM BIEC Specification (not yet published), will supplement or obviate constraint schemas with consistent and formal techniques for representing business rules within NIEM components.

7.7.6.2 Representation

Common

Although formal techniques for representing business rules within NIEM components has not yet been established, business rules can be represented in the NIEM-UML.

The objective in modeling MPD Business Rules is to ensure an executable validation/verification can be implemented and that it can be used to validate/verify modeled XML Instance documents.

Within a UML model, business rules are represented as Constraints. From the UML Specification for Constraint:

> *A constraint is a condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element.*
>
> *Constraint contains a ValueSpecification that specifies additional semantics for one or more elements. Certain kinds of constraints are predefined in UML, others may be user-defined. A user-defined Constraint is described using a specified language, whose syntax and interpretation is a tool responsibility. One predefined language for writing constraints is OCL.*
>
> *Constraint is a condition (a Boolean expression) that restricts the extension of the associated element beyond what is imposed by the other language constructs applied to the element.*
>
> *A Constraint represents additional semantic information attached to the constrained elements. A constraint is an assertion that indicates a restriction that must be satisfied by a correct design of the system. The constrained elements are those elements required to evaluate the constraint specification. In addition, the*

context of the Constraint may be accessed, and may be used as the namespace for interpreting names used in the specification. For example, in OCL 'self' is used to refer to the context element.

Constraints are often expressed as a text string in some language. If a formal language such as OCL is used, then tools may be able to verify some aspects of the constraints.

In general there are many possible kinds of owners for a Constraint. The only restriction is that the owning element must have access to the constrainedElements.

The owner of the Constraint will determine when the constraint specification is evaluated.

The definition of Constraint within UML is fairly general. The specific use and interpretation of OCL based Constraints within UML Models is detailed by the OCL Specification:

For every occurrence of an OCL expression three things need to be separated: the placement, the contextual classifier, and the self instance of an OCL expression.

- *The placement is the position where the OCL expression is used in the UML model (e.g., as invariant connected to class Person).*
- *The contextual classifier defines the namespace in which the expression is evaluated. For example, the contextual classifier of a precondition is the classifier that is the owner of the operation for which the precondition is defined. Visible within the precondition are all model elements that are visible in the contextual classifier.*
- *The self instance is the reference to the object that evaluates the expression. It is always an instance of the contextual classifier. Note that evaluation of an OCL expression may result in a different value for every instance of the contextual classifier.*

*…An **invariant** constraint is a constraint that is linked to a Classifier. The purpose of an **invariant** constraint is to specify **invariants** for the Classifier. An **invariant** constraint consists of an OCL expression of type Boolean. The expression must be true for each **instance** of the classifier at any moment in time.*

When the invariant constraint mechanism is used to represent business rules, they are always specified in the context of a Classifier.

OCL may be used to declare co-constraints and other forms of constraints not expressible by the model/schema directly. The OCL is applicable to any kind of model: reference, subset, extension, or constraint. When subsetting extension models from an EIEM to an IEPD, constraints specified by the subsetted model should be propagated to the subset model. Similarly, when constraining models, the constrained model OCL constraints should be propagated to the constraint model.

Although there is no normative specification for business rule specification in NIEM, the modeling of business rules as OCL in the NIEM-UML model provides the following capabilities:

- OCL itself has a MOF model. As such, it can be transformed using OMG transformation technology, such as QVT, to target a variety of potential business rule implementation technologies.
- Some potential business rule specification languages, such as ISO-Schematron, can be represented as a MOF model and consequently be the target artifacts for QVT. In the case of ISO-Schematron, the use of XPATH as the syntax for rule specification should enable representation of most of the OCL constructs specified for invariant constraints.

PIM

No PIM-specific variations.

PSM

No PSM-specific variations.

7.7.6.3 Mapping Summary

PIM to PSM Mapping

The transformation from PIM to PSM includes propagation of owned rules for Classifiers.

PSM to XML Schema Mapping

Due to lack of normative specifications within NIEM for business-rules, there are currently no identified artifacts to be targeted during transformation from PSM to MPD.

8 NIEM-UML Profile Reference

8.1 Overview

NIEM-UML leverages three profiles. The NIEM PIM Profile is used for NIEM PIMs. The NIEM PSM Profile is used for NIEM PSMs and may also be used to mark up a NIEM PIM for direct provisioning of MPD artifacts. The Model Package Description Profile is used for creating models of MPDs, which may be used in association with either NIEM PIMs or NIEM PSMs.

As shown in Figure 8.1, the NIEM PIM Profile and the NIEM PSM Profile both import the NIEM Common Profile, which contains the core stereotypes used to represent NIEM structures in UML. For convenience, an overall NIEM UML Profile is also included, which imports the NIEM PIM, NIEM PSM, and Model Package Description Profiles. Applying the single NIEM UML Profile is therefore equivalent to individually applying all three of the imported profiles.

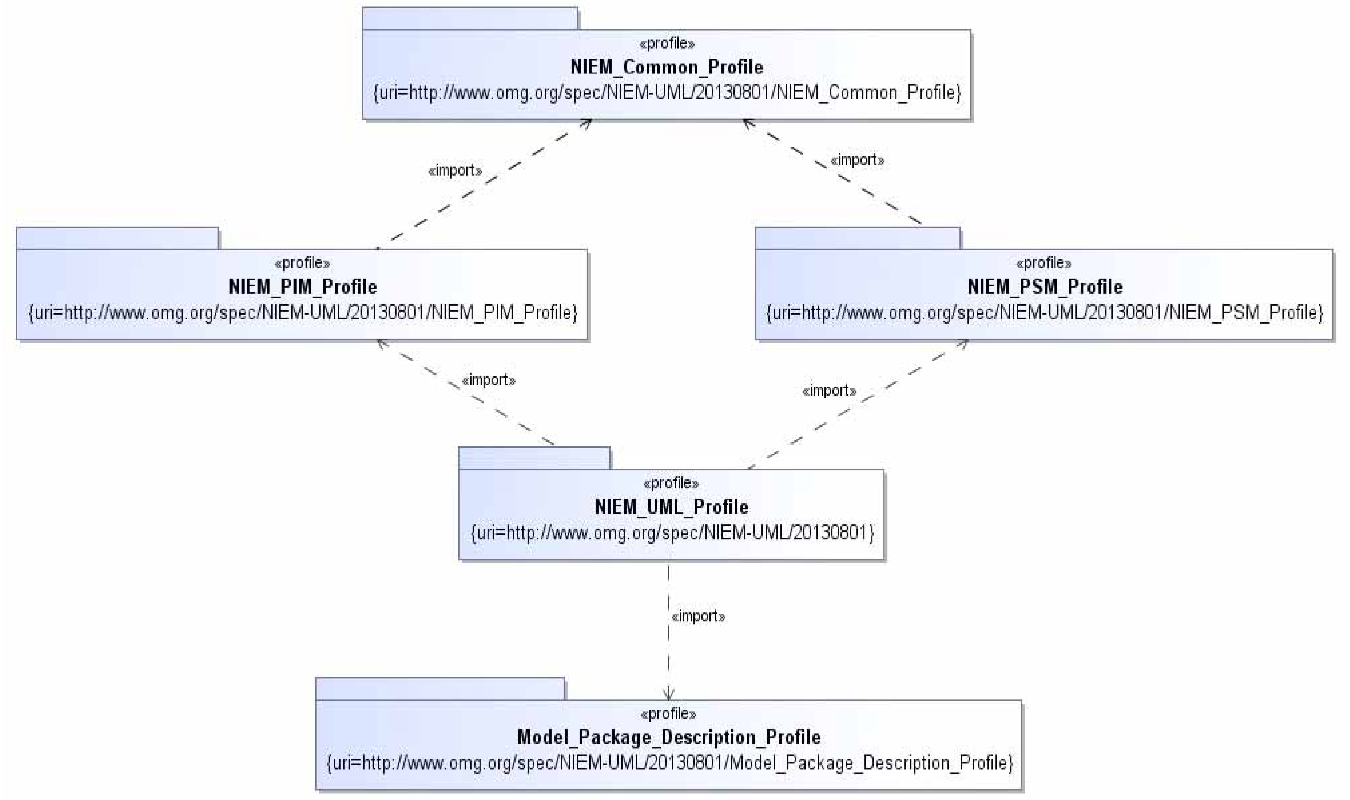
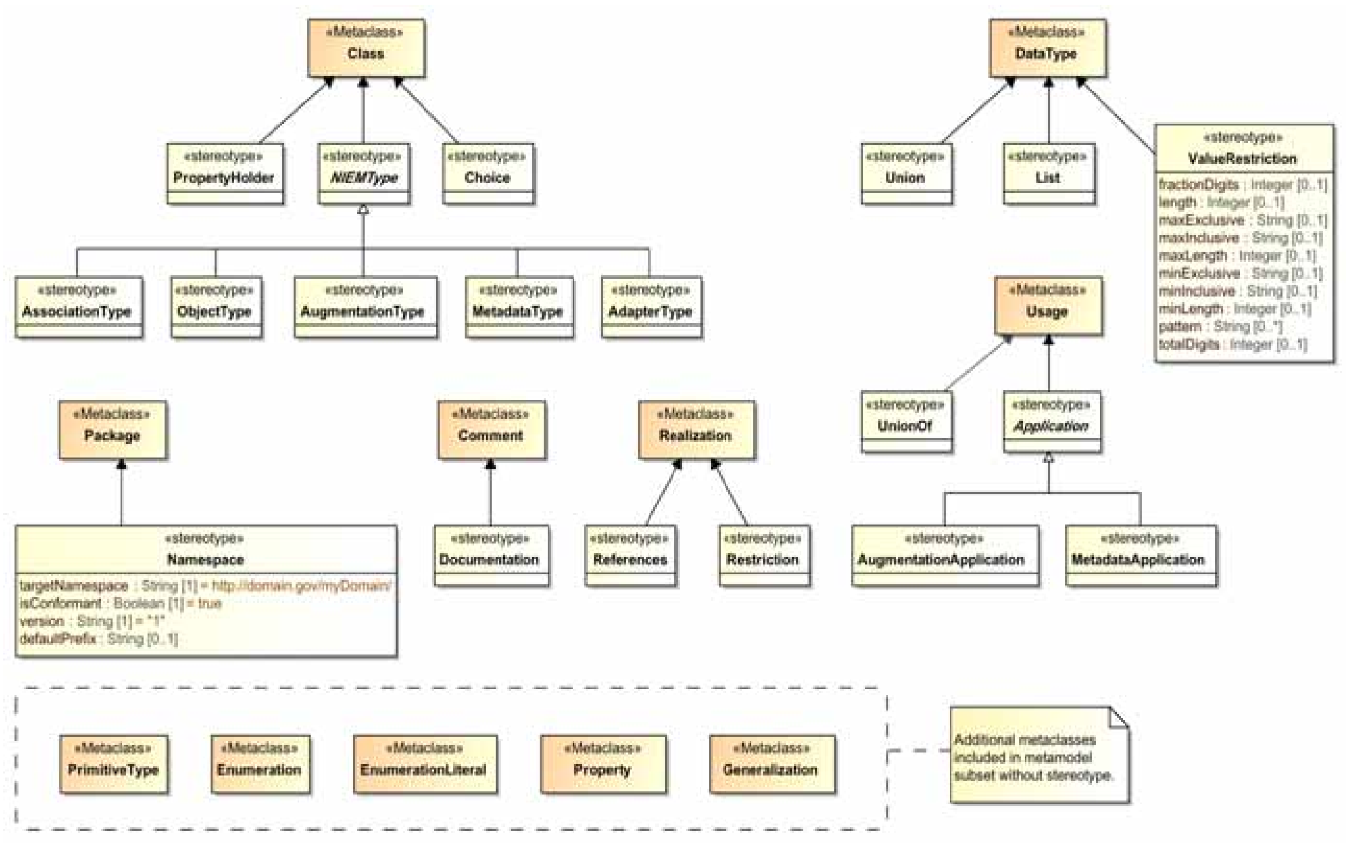

«profile»
NIEM_Common_Profile
{uri=http://www.omg.org/spec/NIEM-UML/20130801/NIEM_Common_Profile}
«import»
«import»
«profile»
NIEM_PIM_Profile
{uri=http://www.omg.org/spec/NIEM-UML/20130801/NIEM_PIM_Profile}
«profile»
NIEM_PSM_Profile
{uri=http://www.omg.org/spec/NIEM-UML/20130801/NIEM_PSM_Profile}
«import»
«import»
«profile»
NIEM_UML_Profile
{uri=http://www.omg.org/spec/NIEM-UML/20130801}
«import»
«profile»
Model_Package_Description_Profile
{uri=http://www.omg.org/spec/NIEM-UML/20130801/Model_Package_Description_Profile}

Figure 8.1 - NIEM UML Profiles

8.2 NIEM Common Profile

8.2.1 Overview

The NIEM Common Profile comprises stereotypes that are used in both the NIEM PIM Profile and the NIEM PSM Profile. In addition, the UML metamodel subset covered by the NIEM Common Profile also includes the metaclasses PrimitiveType, Enumeration, EnumerationLiteral, Property, and Generalization, even though they are not specifically extended by any stereotypes in the profile.



«Metaclass» Class
«stereotype» PropertyHolder
«stereotype» NIEMType
«stereotype» Choice
«stereotype» AssociationType
«stereotype» ObjectType
«stereotype» AugmentationType
«stereotype» MetadataType
«stereotype» AdapterType
«Metaclass» DataType
«stereotype» Union
«stereotype» List
«stereotype» ValueRestriction
fractionDigits : Integer [0..1]
length : Integer [0..1]
maxExclusive : String [0..1]
maxInclusive : String [0..1]
maxLength : Integer [0..1]
minExclusive : String [0..1]
minInclusive : String [0..1]
minLength : Integer [0..1]
pattern : String [0..*]
totalDigits : Integer [0..1]
«Metaclass» Usage
«stereotype» UnionOf
«stereotype» Application
«stereotype» AugmentationApplication
«stereotype» MetadataApplication
«Metaclass» Package
«stereotype» Namespace
targetNamespace : String [1] = http://domain.gov/myDomain/
isConformant : Boolean [1] = true
version : String [1] = "1"
defaultPrefix : String [0..1]
«Metaclass» Comment
«stereotype» Documentation
«Metaclass» Realization
«stereotype» References
«stereotype» Restriction
«Metaclass» PrimitiveType
«Metaclass» Enumeration
«Metaclass» EnumerationLiteral
«Metaclass» Property
«Metaclass» Generalization
Additional metaclasses included in metamodel subset without stereotype.

Figure 8.2 - NIEM Common Profile

8.2.2 <Stereotype> AdapterType

Generalization: NIEM_Common_Profile::NIEMType

Description

An AdapterType is a NIEMType Class that represents a NIEM adapter type. A NIEM adapter type is a NIEM object type that adapts external components for use within NIEM. External components are not NIEM-conforming (e.g., data components from other standards, e.g., GML, ISO, etc.). An adapter type creates a new class of object that embodies a single concept composed of external components. AdapterType is implemented in XML Schema as a complex type definition with complex content. Section 3.4 of XML Schema Structures addresses complex type definitions in XML Schema; Section 7.7 of NIEM NDR v1.3 addresses adapter types in NIEM-conformant XML Schema.

8.2.3 <Stereotype> Application

Extends

- UML::Usage

Description

An Application stereotype is applied to a Usage dependency between a client Property or Class and a supplier Class. It corresponds to the NIEM concept of “AppliesTo,” which constrains the applicability of the client as being to the NIEM type represented by the supplier class. The Application Stereotype is abstract, its two concrete stereotypes being AugmentationApplication and MetadataApplication, representing the use of “AppliesTo” in the context of augmentation and metadata, respectively.

Constraints

- NDR [Rule 7-24]

[OCL2.0]

```
self.base_Usage.supplier->forAll(s|s.oclIsKindOf(Classifier))
and self.base_Usage.client ->forAll(client|
client.stereotypedBy('MetadataType') or ( client.oclIsKindOf(Property)
and client.oclAsType(Property).type.stereotypedBy('AugmentationType') )
)
```

- NDR [Rule 7-25]

[OCL2.0]

```
self.base_Usage.supplier ->forAll(supplier|
(supplier.oclIsKindOf(Class) or supplier.oclIsKindOf(DataType)) and
supplier.getNearestPackage().stereotypedBy('Namespace') )
```

- NDR [Rule 7-26]

[English] This constraint, at the UML level, is identical to NDR [Rule 7-25].

- NDR [Rule 7-27]

[English] This constraint is decomposed into two specific cases. NDR [Rule 7-49] defines constraints related to applying Augmentation and NDR [Rule 7-45] defines constraints related to applying Metadata.

- NDR [Rule 7-28]

[English] This constraint is resolved by the combination of other constraints and the PSM-XSD transformations.

- NDR [Rule 7-29]

[English] This constraint is resolved by other constraints in combination with the PSM - XSD transformations.

- NDR [Rule 7-46]

[English] Definitional constraint. Application of <<AppliesTo>> is optional.

8.2.4 <Stereotype> AssociationType

Generalization: NIEM_Common_Profile::NIEMType

Description

AssociationType is a NIEMType class that represents a NIEM association type. A NIEM association type establishes a relationship between objects, along with the properties of that relationship. A NIEM association is an instance of an association type. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of a UML Association or AssociationClass may not necessarily be sufficient to reflect the variability of a NIEM association. Consequently, the AssociationType Stereotype is applied to a UML Class. Since an AssociationClass is also a Class, the AssociationType Stereotype may be applied to a UML AssociationClass where appropriate. Note that a UML AssociationClass specializing another AssociationClass must have the same number of ends as the other AssociationClass and must have at least two ends. This UML constraint prevents the usage of AssociationClass to model abstract NIEM association types that are intended to be extended by subtypes with additional ends. A UML AssociationClass can specialize an abstract UML Class. AssociationType is implemented in XML Schema as a complex type definition with complex content. Section 3.4 of XML Schema Structures addresses complex type definitions in XML Schema; Section 7.4.3 of NIEM NDR v1.3 addresses association types in NIEM-conformant XML Schema.

Constraints

- NDR [Rule 7-41]

[OCL2.0]

```
(self.base_Class.generalization->size()<=1) and
self.base_Class.generalization.general->forAll(g|g.stereotypedBy('Association'))
and self.base_Class.getTargetDirectedRelationships()
->select(r|r.oclIsKindOf(Generalization)).oclAsType(Generalization).specific
->forAll(s|s.stereotypedBy('Association'))
```

- NDR [Rule 7-42]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.ownedAttribute
->forAll(a|a.name.endsWith('Reference') implies
(a.aggregation=AggregationKind::none))
```

- NDR [Rule 9-27]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.name.match('.*AssociationType')) --For PIM Profile, the
naming constraint is enforced by PIM/PSM transformations.
```

8.2.5 <Stereotype> AugmentationApplication

Generalization: NIEM_Common_Profile::Application

Description

The AugmentationApplication stereotype is a specialization of Application that is always between a Property whose type is an AugmentationType Class and a Class. It represents a constraint on a NIEM augmentation property that limits the application of the property to specific types. When an augmentation property is the client of an AugmentationApplication Usage, this serves to constrain the NIEM types to which the augmentation property may apply. If the client augmentation Property is not in a PropertyHolder, then its Class must be a (direct or indirect) subclass of the supplier Class of the AugmentationApplication. If the augmentation Property is in a PropertyHolder, then any Property defined by reference to the augmentation Property must be for a Class that is a subclass of the supplier Class. An AugmentationApplication Usage is implemented in a NIEM XML schema as an AppliesTo element in the annotation element of the property declaration represented by the client of the Usage, referencing the complex type represented by the supplier class of the Usage.

Constraints

- NDR [Rule 7-50]

[English] Implemented as part of PIM/PSM transformations. This constraint is definitional, the absence of an <<AugmentationApplication>> on an augmentation element will result in applicability of the element to any Object/ Association at runtime.

8.2.6 <Stereotype> AugmentationType

Generalization: NIEM_Common_Profile::NIEMType

Description

AugmentationType is a NIEMType Class that represents a NIEM augmentation type. 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. An augmentation of an object type is a block of additional data that is an instance of an augmentation type, added to an object type to carry additional data beyond that of the original object definition. The applicability of an augmentation may be restricted using an AugmentationApplication Dependency or an Augments Generalization. A Class that is the specific Classifier of an Augments Generalization shall be inferred to be an AugmentationType. AugmentationType is implemented in XML Schema as a complex type definition with complex content. Section 3.4 of XML Schema Structures addresses complex type definitions in XML Schema; Section 7.4.5 of NIEM NDR v1.3 addresses augmentation types in NIEM-conformant XML Schema.

Constraints

- NDR [Rule 7-47]

[OCL2.0]

```
(self.base_Class.general()->size()<=1) and
self.base_Class.general()->forAll(g|g.stereotypedBy('AugmentationType'))
and self.base_Class.getTargetDirectedRelationships()
->forAll(r|r.oclIsKindOf(Gneralization) implies
r.oclAsType(Generalization).specific.stereotypedBy('AugmentationType'))
```

- NDR [Rule 7-48]

[English] The constraint is enforced by the transformation from PSM to XSD Schema artifact. A property whose type is an <<AugmentationType>> is an augmentation element. The property may directly or indirectly use the UML subsettedProperty mechanism to identify a substitutionGroup, which will be transitively substitutable for the element structures:Augmentation.

- NDR [Rule 9-28]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.name.match('.*AugmentationType')) --For PIM Profile, the
naming constraint is enforced by PIM/PSM transformations.
```

8.2.7 <Stereotype> Choice

Extends

- UML::Class

Description

A Choice Class groups a set of attributes whose values are mutually exclusive. That is, in any instance of a Choice Class, at most one of its attributes may be non-empty. Choice represents the use of a choice model group in XML Schema. Section 3.8 of XML Schema Structures addresses choice model groups in XML Schema. Sections 6.1.8.1 and 6.1.8.2 of NIEM NDR v1.3 address choice model groups in NIEM-conformant XML Schema.

Constraints

- No Generalizations or subtypes

[OCL2.0]

```
self.base_Class.generalization->isEmpty() and
self.base_Class.getTargetDirectedRelationships()->select(d|d.oclIsKindOf(Generalization))->isEmpty()
```

- ownedAttributes have multiplicity 0..1.

[OCL2.0]

```
self.base_Class.ownedAttributes->forAll(a|(a.lower=0) and
(a.upper=1))
```

8.2.8 <Stereotype> Documentation

Extends

- UML::Comment

Description

A Documentation Comment is the data definition of the Element that owns it. For an Element owning only one Comment, that Comment will be inferred to be a Documentation Comment. A Documentation Comment owned by an Element representing a NIEM type or property is implemented as a documentation element of the annotation for the corresponding type definition or property declaration.

Constraints

- Max One <<Documentation>> per Element

[OCL2.0]

```
self.base_Comment.annotatedElement->notEmpty() and
self.base_Comment.annotatedElement->forAll(e|e=self.base_Comment.owner)

and
(self.base_Comment.owner.ownedComment->select(c|c.stereotypedBy('Documentation'))->size()=1)
```

8.2.9 <Stereotype> List

Extends

- UML::DataType

Description

A List is a DataType whose values consist of a finite length (possibly empty) sequence of values of another DataType, which is the item type of the List. A List DataType must have a single Property with multiplicity 0..* whose type is the item type. The name of this element is not material. A List DataType is implemented in XML schema as a list simple type definition. List represents a relationship between two simple type definitions: the first is a list simple type definition whose item type definition is the second. This relationship is implemented in XML Schema through the itemType attribute on the `xsd:list` element of the list simple type definition, the actual value of which resolves to the second type definition. Section 3.14 of XML Schema Structures addresses list simple type definitions in XML Schema; Section 7.3 of NIEM NDR v1.3 addresses list simple type definitions in NIEM-conformant XML Schema.

Constraints

- single ownedAttribute with multiplicity 0..* typed <DataType>>

[OCL2.0]

```
(self.base_DataType.ownedAttribute->size()=1) and
self.base_DataType.ownedAttribute ->forAll(a|(a.lower=0) and
(a.upper=-1))
```

- no generalizations

[OCL2.0]

```
self.base_DataType.generalization->isEmpty()
```

8.2.10 <Stereotype> MetadataApplication

Generalization: NIEM_Common_Profile::Application

Description

The MetadataApplication stereotype is a specialization of Application that is always between a MetadataType Class and another Class. It represents a constraint on a NIEM metadata type that limits the application of the NIEM metadata type to specific types. If a MetadataType Class is the client of a MetadataApplication Usage, then any Property with the MetadataType Class as its type must be for a Class that is a (direct or indirect) subclass of the supplier Class of the MetadataApplication. A MetadataType Class may be the client of multiple MetadataApplication Usages, in which case a Property for it may be in a Class that is a subclass of a supplier Class of any of the MetadataApplications. If a MetadataType is not a client of any MetadataApplication, then it applies to any type. A MetadataApplication Usage is implemented in XML schema as a NIEM AppliesTo element in the annotation element of the complex type definition represented by the client of the Usage, referencing the complex type represented by the supplier class of the Usage.

8.2.11 <Stereotype> MetadataType

Generalization: NIEM_Common_Profile::NIEMType

Description

A MetadataType is a NIEMType Class that represents a NIEM metadata type. A NIEM 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. Metadata is specified as an instance of a metadata type and may include information such as the security of a piece of data or the source of the data. The applicability of such metadata may be modeled using MetadataApplication dependencies to one or more classes representing the applicable types. MetadataType is implemented in XML Schema as a complex type definition with complex content. Section 3.4 of XML Schema Structures addresses complex type definitions in XML Schema; Section 7.4.4 of NIEM NDR v1.3 addresses metadata types in NIEM-conformant XML Schema.

Constraints

- NDR [Rule 7-43]

[English] Containing Elements appropriate for a specific class of data about data is a Non-computational constraint.

- NDR [Rule 7-44]

[OCL2.0]

```
self.base_Class.general->select(g|g.stereotypedBy('MetadataType'))->isEmpty()
```

- NDR [Rule 9-29]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.name.match('.*MetadataType')) --For PIM Profile, the
naming constraint is enforced by PIM/PSM transformations.
```

8.2.12 <Stereotype> NIEMType

Extends

- UML::Class

Description

A NIEMType is a Class that represents one of the specific semantic kinds of NIEM complex types (i.e., types that may have attributive structure). NIEMType is abstract. A NIEMType Class is implemented in XML Schema as a complex type definition with complex content.

Constraints

- NDR [Rule 7-45]

[OCL2.0]

```
self.base_Class.ownedAttribute ->forAll(a|
a.type.stereotypedBy('MetadataType') implies a.type.clientDependency
->select(d|d.stereotypedBy('MetadataApplication')).target.oclAsType(Class)
->exists(appliedTo|self.base_Class.isConsistentWith(appliedTo)) )
```

- NDR [Rule 7-49]

[OCL2.0]

```
self.base_Class.ownedAttribute ->forAll(a|
a.type.stereotypedBy('AugmentationType') implies a.clientDependency
->union(a.clientDependency->select(d|d.stereotypedBy('References')).target.clientDependency)
->select(d|d.stereotypedBy('AugmentationApplication')).target.oclAsType(Class)
->exists(appliedTo|self.base_Class.isConsistentWith(appliedTo)) )
```

- NDR [Rule 9-32]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.ownedAttribute
->forAll(a|a.type.stereotypedBy('AssociationType') implies
a.name.match('.*Association.*')) --For PIM Profile, the naming
constraint is enforced by PIM/PSM transformations. --An association
element corresponds to a UML <Property> whose type is an
<<AssociationType>>.
```

- NDR [Rule 9-33]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.ownedAttribute
->forAll(a|a.type.stereotypedBy('AugmentationType') implies
a.name.match('.*Augmentation.*')) --For PIM Profile, the naming
constraint is enforced by PIM/PSM transformations. --An augmentation
element corresponds to a UML <Property> whose type is an
```

<<AugmentationType>>.

- NDR [Rule 9-34]

[OCL2.0]

```
self.base_Class.profiledBy('NIEM_PSM_Profile') implies
self.base_Class.ownedAttribute
->forAll(a|a.type.stereotypedBy('MetadataType') implies
a.name.match('.*Metadata.*')) --For PIM Profile, the naming constraint
is enforced by PIM/PSM transformations. --A metadata element corresponds
to a UML <Property> whose type is a
<<Metadata>>.
```

8.2.13 <Stereotype> Namespace

Extends

- UML::Package

Description

A Namespace Package represents a NIEM namespace identified by a target namespace URI. All UML model elements contained, directly or indirectly within the Package, that represents NIEM types and properties, are considered to be in this target namespace. A Namespace Package is implemented in XML Schema as an XML schema document.

Attributes

- defaultPrefix :PrimitiveTypes::String [0..1] {unique }

The default prefix for the namespace, used to represent common NIEM prefixes. This prefix is used on all XML and/or XML Schema serializations using the namespace, unless it conflicts with another XML and/or XML Schema serialization. If there is a conflict, the actual prefix used is the given default prefix with a number appended in order to make it unique.

- isConformant :PrimitiveTypes::Boolean [1] {unique }

Indicates whether the namespace is NIEM-conformant. If the namespace is NIEM-conformant, it is implemented in XML Schema as the content of the `i:ConformantIndicator` application information on the `xsd:schema` document element. Per Rule 7-1 of NIEM NDR v1.3, the content must be "true." If the namespace is not NIEM-conformant, it is implemented in XML Schema as the content of the i:ConformantIndicator application information on the `xsd:import` element. Per Rule 7-61 of NIEM NDR v1.3, the content must be "false."

- targetNamespace :PrimitiveTypes::String [1] {unique }

The target namespace URI for this NIEM namespace. It is implemented in XML Schema as the value of the targetNamespace attribute on the `xsd:schema` document element. Per Rules 6-35 and 6-36 of NIEM NDR v1.3, the value of the targetNamespace attribute must be present and must be an absolute URI.

- version :PrimitiveTypes::String [1] {unique }

The version of the NIEM namespace. It is implemented in XML Schema as the value of the version attribute on the `xsd:schema` document element. Per Rules 6-37 and 6-38 of NIEM NDR v1.3, the value of the version attribute must be present and must not be the empty string. Default is "1."

8.2.14 <Stereotype> ObjectType

Generalization: NIEM_Common_Profile::NIEMType

Description

ObjectType is a NIEMType Class that represents a NIEM object type. A NIEM 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. ObjectType is implemented in XML Schema as a complex type definition. Section 3.4 of XML Schema Structures addresses complex type definitions in XML Schema; Section 7.4.1 of NIEM NDR v1.3 addresses object types in NIEM-conformant XML Schema.

8.2.15 <Stereotype> PropertyHolder

Extends

- UML::Class

Description

PropertyHolder is a Class holding global Properties that are not the subject of any specific NIEM type. A Property of a NIEM type may then be defined by reference to a Property of a PropertyHolder by using a References realization with the Property in the PropertyHolder as the supplier. Note that the multiplicity of Properties in a PropertyHolder is immaterial -- the multiplicities are established by Properties in the corresponding References client. The target namespace of Properties in a PropertyHolder is the target namespace of the Namespace Package that contains the PropertyHolder (which may be different than the target namespace of NIEM types that use the Properties in the PropertyHolder). PropertyHolder does not represent any NIEM concept; it exists to permit the user to define a NIEM property that is not the subject of any NIEM type. 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 v1.3, Rule 6-14 and Rule 6-15, require that an attribute or element declaration be a top-level declaration, but NIEM NDR v1.3 does not require a corresponding attribute use or element particle; however, Section 7.3.44 of [UML] requires that a Property be the ownedAttribute of a Classifier. Thus in the UML representation, the declaration and use of a Property are not distinct, and the declaration of a Property requires its use. In the XML Schema implementation, the declaration and use are distinct, and the declaration does not require a corresponding use. To resolve this difference, any Property within a PropertyHolder shall represent an attribute or element declaration without a corresponding attribute use or element particle. PropertyHolders may be used to hold the properties of a substitution group. Where a PropertyHolder is used to define a substitution group an extension of that substitution group shall be a subclass of the substitution group PropertyHolder.

8.2.16 <Stereotype> References

Extends

- UML::Realization

Description

The References Stereotype applies to a Realization between Properties, Classes, or Packages. It allows for Properties in one Class to be defined by reference to Properties in another class. A References Realization between two classes is defined to be equivalent to having References Realizations between matching Properties of the Classes where matching is determined by identical NIEM names. A References Realization between two packages is defined to be equivalent to having References Realizations between matching Classes contained in the Packages where matching is determined by having identical NIEM names. Matching is based on the NIEMName of the elements, either as derived implicitly or as set explicitly using the ReferenceName stereotype. If a Property is the client of a References Realization, then it represents a NIEM property defined by reference to the NIEM property declaration represented by the supplier of the Realization. It is implemented in XSD schema as an attribute use or element particle that references the attribute or element declaration that implements the supplier of the Realization. Note that the supplier Property may be in a different Namespace than the client property, in which case the attribute or element declaration represented by the supplier will be in a different target namespace than the use represented by the client.

Constraints

- MDR [Rule 3-02]

[OCL2.0]

```
( self.base_Realization.client->size()=1) and (
self.base_Realization.supplier->size()=1) and
self.base_Realization.client->forAll(client|client.oclIsKindOf(Classifier))
and
self.base_Realization.supplier->forAll(supplier|supplier.oclIsKindOf(Classifier)
and not(supplier.stereotypedBy('PropertyHolder')) ) ) implies ( (
self.base_Realization.client.oclAsType(Classifier).attribute
->forAll(clientAttribute|
self.base_Realization.supplier.oclAsType(Classifier).attribute
->forAll(supplierAttribute|
(clientAttribute.name=supplierAttribute.name) implies (
(clientAttribute.lower>=supplierAttribute.lower) and (
(supplierAttribute.upper=-1) or
(clientAttribute.upper<=supplierAttribute.upper) ) and (
(clientAttribute.upper=-1) or
(clientAttribute.lower<=clientAttribute.upper) ) ) ) ) ) and(
self.base_Realization.supplier.oclAsType(Classifier).attribute
->select(a|a.lower>0) ->forAll(supplierAttribute|
self.base_Realization.client.oclAsType(Classifier).attribute->exists(clientAttribute|clientAttribute.name=supplierName)
) ) )
```

8.2.17 <Stereotype> Restriction

Extends

- UML::Realization

Description

A Restriction Realization represents a relationship between two type definitions: the first is derived by restriction from the second. The two types must either both be NIEMType Classes or both be DataTypes. If the two types are Classes, then the attributes of the client class must be a subset of the attributes of the supplier class and omitted attributes must have a multiplicity lower bound of zero. If the two classes are DataTypes, then the client type is considered to have a value space that is a subset of that of the supplier, as may be further specified using a ValueRestriction stereotype on the client. This relationship is implemented in XML Schema through the base attribute on the `xsd:restriction` element of the first type definition, the actual value of which resolves to the second type definition. If a type is a ValueRestriction, the generalization owned by that type is implicitly an XSDRestriction. Sections 3.4 and 3.14 of XML Schema Structures address the use of restriction in XML Schema; Sections 6.5.2 and 6.5.3 of NIEM NDR v1.3 address the use of restriction in NIEM-conformant XML Schema.

Constraints

- XSDRestrictionComplexTypeComplexContent

[OCL2.0]

```
self.base_Generalization.general.stereotypedBy('NIEMType') and
self.base_Generalization.general.clientDependency->select(d|d.stereotypedBy('NIEMSimpleContent'))->isEmpty()
implies
self.base_Generalization.specificl.clientDependency->select(d|d.stereotypedBy('NIEMSimpleContent'))->isEmpty()
```

- XSDRestrictionComplexTypeSimpleContent

[OCL2.0]

```
self.base_Generalization.general.stereotypedBy('NIEMType') and
self.base_Generalization.general.clientDependency->select(d|d.stereotypedBy('XSDSimpleContent'))->notEmpty()
implies self.base_Generalization.specific.stereotypedBy('NIEMType') and
self.base_Generalization.specific.clientDependency->select(d|d.stereotypedBy('XSDSimpleContent'))->notEmpty()
```

- XSDRestrictionSimpleType

[OCL2.0]

```
self.base_Generalization.general.oclIsKindOf(DataType) implies
self.base_Generalization.specific.oclIsKindOf(DataType)
```

8.2.18 <Stereotype> Union

Extends

- UML::DataType

Description

A Union is a DataType whose value space is the union of one or more other DataTypes, which are the member types of the Union. The member types are specified using UnionOf Usage dependencies. A Union DataType is implemented in XML Schema as a union simple type definition. Each UnionOf dependency of which the Union is the client represents a relationship between two type definitions: the first is a union simple type definition whose member type definition is the second. This relationship is implemented in XML Schema through the memberTypes attribute on the `xsd:union` element of the union simple type definition, the actual value of which resolves to the second type definition. Section 3.14 of XML Schema Structures addresses union simple type definitions in XML Schema.

Constraints

- no generalizations

[OCL2.0]

```
self.base_DataType.generalization->isEmpty()
```

- no owned attributes

[OCL2.0]

self.base_DataType.ownedAttribute->isEmpty()

8.2.19 <Stereotype> UnionOf

Extends

- UML::Usage

Description

The UnionOf stereotype is applied to a Usage dependency, the client of which must be a Union DataType and the supplier of which must be a DataType that represents a legal union member type. A UnionOf dependency specifies that the supplier DataType is a member type of the client Union.

Constraints

- client must be union

[OCL2.0]

self.base_Usage.client.stereotypedBy('Union')

- supplier must be data type

[OCL2.0]

self.base_Usage.supplier.oclKindOf(DataType)

8.2.20 <Stereotype> ValueRestriction

Extends

- UML::DataType

Description

The ValueRestriction stereotype applies to a DataType (Enumeration or Primitive type) that is a specialization of a general DataType. It defines restrictions on which values of the general DataType that are allowed as values of the specialized DataType. A ValueRestriction DataType is implemented in XML Schema as a simple type definition that is a restriction of the simple type that implements the general DataType. The attributes of the ValueRestriction are implemented as restriction facets. ValueRestriction represents a NIEM type which is implemented in XML Schema as a simple type definition. Section 3.14 of XML Schema Structures addresses simple type definitions in XML Schema; Sections 6.1.6.1, 7.2.1, 7.3, and 9.12.2 of NIEM NDR v1.3 address simple type definitions in NIEM-conformant XML Schema. The variety of the simple type definition may be union, list, or atomic. As the ValueRestriction stereotype is a specialization of DataType, it may be applied to Enumeration. In this case, the ValueRestriction represents a NIEM code type, which is implemented in XML Schema as a simple type definition that contains multiple `xsd:enumeration` facets.

Attributes

- fractionDigits :PrimitiveTypes::Integer [0..1] {unique }

A restriction on the value space of a numeric data type that places an upper limit on the arithmetic precision of decimal values. The value space is restricted to those values that can be represented lexically in decimal notation using at most fractionDigits to the right of the decimal point. fractionDigits is implemented in XML Schema as the value of the value attribute on the `xsd:fractionDigits` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- length :PrimitiveTypes::Integer [0..1] {unique }

A restriction on the value space of a data type to values with a specific length, where the units of length depends on the base type being restricted. For String and URI values, the units are characters. For Binary values, the units are octets. For lists, the length is the number of items in the list. length is implemented in XML Schema as the value of the value attribute on the `xsd:length` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- maxExclusive :PrimitiveTypes::String [0..1] {unique }

The exclusive upper bound of the value space for a data type with ordered values. The value of maxExclusive must be equal to some value in the value space of the base data type or to the maxExclusive restriction of the base type (if it has one). maxExclusive is implemented in XML Schema as the value of the value attribute on the `xsd:maxExclusive` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- maxInclusive :PrimitiveTypes::String [0..1] {unique }

The inclusive upper bound of the value space for a data type with ordered values. The value of maxInclusive must be equal to some value in the value space of the base data type. maxInclusive is implemented in XML Schema as the value of the value attribute on the `xsd:maxInclusive` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- maxLength :PrimitiveTypes::Integer [0..1] {unique }

A restriction on the value space of a data type to values with a specific maximum length, where the units of length depends on the base type being restricted. For String and URI values, the units are characters. For Binary values, the units are octets. For lists, the length is the number of items in the list. maxLength is implemented in XML Schema as the value of the value attribute on the `xsd:maxLength` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- minExclusive :PrimitiveTypes::String [0..1] {unique }

The exclusive lower bound of the value space for a data type with ordered values. The value of minExclusive must be equal to some value in the value space of the base data type or to the minExclusive restriction of the base type (if it has one). minExclusive is implemented in XML Schema as the value of the value attribute on the `xsd:minExclusive` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- minInclusive :PrimitiveTypes::String [0..1] {unique }

The inclusive lower bound of the value space for a data type with ordered values. The value of minInclusive must be equal to some value in the value space of the base data type. minInclusive is implemented in XML Schema as the value of the value attribute on the `xsd:minInclusive` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- minLength :PrimitiveTypes::Integer [0..1] {unique }

A restriction on the value space of a data type to values with a specific minimum length, where the units of length depends on the base type being restricted. For String and URI values, the units are characters. For Binary values, the units are octets. For lists, the length is the number of items in the list. minLength is implemented in XML Schema as the value of the value attribute on the `xsd:minLength` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- pattern :PrimitiveTypes::String [*] {unique }

A constraint on the value space of a data type achieved by constraining the value space to those values represented by literals that match each member of a set of regular expressions. Each pattern must be a valid regular expression. pattern is implemented in XML Schema as the value of the value attribute on the `xsd:pattern` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

- totalDigits :PrimitiveTypes::Integer [0..1] {unique }

Restricts the magnitude and arithmetic precision of values in the value space of a numeric data type. The value space is restricted to those values that can be represented lexically using at most totalDigits digits in decimal notation or at most totalDigits digits for the coefficient, in scientific notation. totalDigits is implemented in XML Schema as the value of the value attribute on the `xsd:totalDigits` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

Constraints

- ValueRestrictionGeneralization

[OCL2.0]

```
self.base_DataType.generalization.general->size()=1
```

8.3 NIEM PIM Profile

8.3.1 Overview

The NIEM PIM Profile comprises stereotypes that are used in NIEM PIMs but not NIEM PSMs. Further, the NIEM PIM Profile imports the NIEM Common Profile and, therefore, includes all the stereotypes and metaclasses covered by that profile. In addition, the UML metamodel subset covered by the NIEM PIM Profile also includes the metaclasses Association and AssociationClass, even though they are not specifically extended by any stereotypes in the profile.

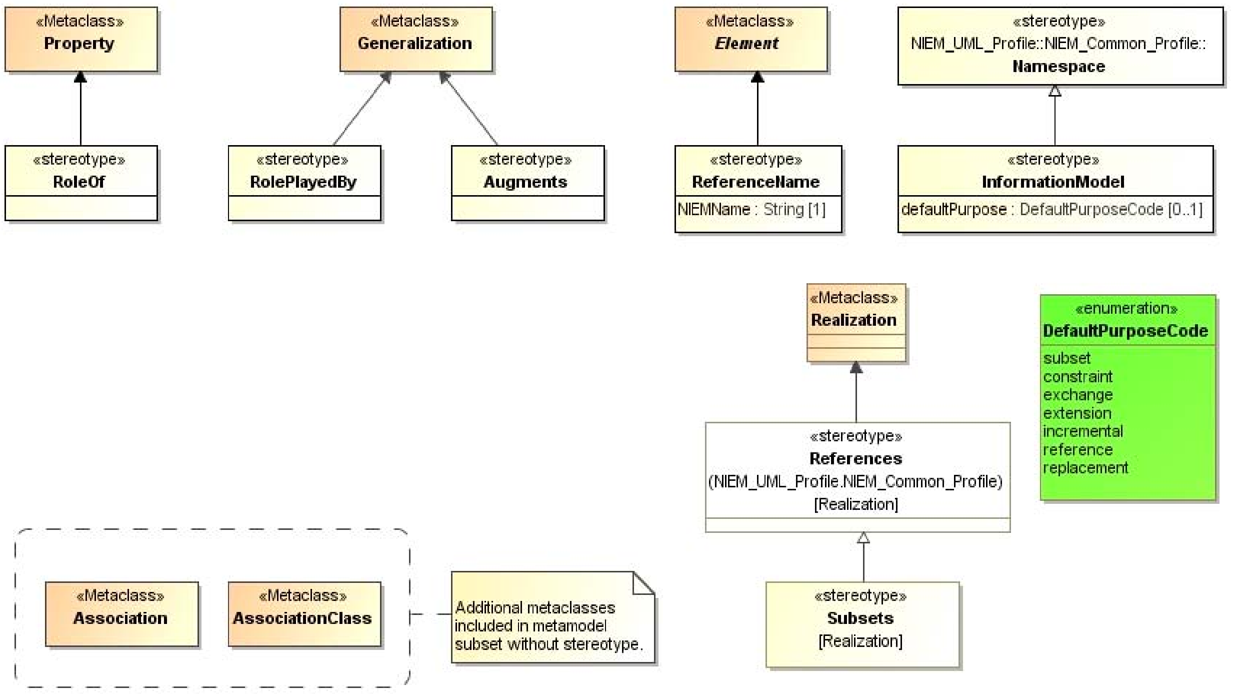

«Metaclass» Property
«Metaclass» Generalization
«Metaclass» Element
«stereotype» NIEM_UML_Profile::NIEM_Common_Profile:: Namespace
«stereotype» RoleOf
«stereotype» RolePlayedBy
«stereotype» Augments
«stereotype» ReferenceName
NIEMName : String [1]
«stereotype» InformationModel
defaultPurpose : DefaultPurposeCode [0..1]
«Metaclass» Realization
«enumeration» DefaultPurposeCode
subset
constraint
exchange
extension
incremental
reference
replacement
«stereotype» References (NIEM_UML_Profile.NIEM_Common_Profile) [Realization]
«Metaclass» Association
«Metaclass» AssociationClass
Additional metaclasses included in metamodel subset without stereotype.
«stereotype» Subsets [Realization]

Figure 8.3 - NIEM PIM Profile

8.3.2 <Stereotype> Augments

Extends

- UML::Generalization

Description

An Augments Generalization specifies that the special Class is an augmentation type that is restricted to apply to instances of the general Class.

8.3.3 <Enumeration> DefaultPurposeCode

Description

The possible purposes for an information model. This enumeration provides the allowed values for the defaultPurpose attribute of the InformationModel stereotype. The values correspond to the schema purpose codes for an MPD artifact.

Enumeration Literals

- constraint
- exchange
- extension
- incremental

- reference
- replacement
- subset

8.3.4 <Stereotype> InformationModel

Generalization: NIEM_Common_Profile::Namespace

Extends

- UML::Package

Description

The contents of an InformationModel Package provide a platform-independent perspective on the structure of information to be exchanged in NIEM messages. Such a model is always taken to represent a NIEM namespace, but it may also be given a default purpose as modeled, independent of the implementation of that namespace. This allows a modeler to identify the intended purposes (e.g., reference, subset, exchange, etc.) of various information models within a set, without having to create a complete MPD model for the set.

Attributes

- defaultPurpose : NIEM_PIM_Profile :: DefaultPurposeCode [0..1] {unique }

The default purpose for which an information model is intended. If an InformationModel Package is modeled as being included as an artifact in an MPD, then, unless otherwise specified, the purpose of the artifact is by default taken to be the schema purpose code corresponding to the value of the defaultPurpose attribute.

8.3.5 <Stereotype> ReferenceName

Extends

- UML::Element

Description

The ReferenceName stereotype is used on an Element that has a name that does not conform to the naming conventions required by the NIEM NDR or is otherwise not the desired NIEM name. The NIEMName attribute must provide a name for the Element that conforms to the relevant NDR naming rules for the specific kind of Element to which the stereotype is applied.

Attributes

- NIEMName :PrimitiveTypes::String [1] {unique }

A NIEM NDR-conformant name to be applied to an Element. The NIEMName will override any name generated from the UML name.

8.3.6 <Stereotype> RoleOf

Extends

- UML::Property

Description

The RoleOf stereotype is applied to a Property of a Class representing a NIEM role type, whose type identifies the base type of that role type. A RoleOf Property must be a reference (i.e., have aggregation=none). A NIEM role type is a type that represents a particular function, purpose, usage, or role of an object.

Constraints

- NDR [Rule 7-40]

[English] This constraint is implemented by the PIM/PSM transformation. Identifying a <Property> as a <<RoleOf>> corresponds to the NIEM naming convention used to identify the roleOf...reference and furthermore establishes the owning <Classifier> as a NIEM Role.

- NDR [Rule 9-35]

[English] This constraint is enforced by the PIM/PSM transformation. The Transformation ensures that the “RoleOf” property term becomes part of the target PSM property name.

8.3.7 <Stereotype> RolePlayedBy

Extends

- UML::Generalization

Description

RolePlayedBy Generalization specifies that the special class is to be considered the type of a role that is played by instances of the general class. In the PSM this will map to a property with the “RoleOf” prefix.

8.3.8 <Stereotype> Subsets

Generalization: <Stereotype> References

Extends

- UML::Realization

Description

A Realization signifying a NIEM subsetting relationship between a client derived (subset) element and a supplier base (reference) element. The <<Subsets>> Realization must be between the same meta-types: either Properties, Classifiers, or <<InformationModel>> packages. The <<Subsets>> Realization must be between elements owned by different <<InformationModel>> packages. The targetNamespace of the distinct <<InformationModel>> packages must be identical. The defaultPurpose of client and supplier may be one of the following combinations: client is subset, supplier is reference; client is reference, supplier is reference; client is extension, supplier is extension; client is constraint, supplier is exchange, subset, extension, or reference.

Constraints

- NDR [Rule 7-69]

[English] The namespace of the subset and reference models must match.

[OCL2.0]

```
self.base_Realization.client
   ->union(self.base_Realization.supplier)
   ->forAll(e1,e2|e1.getTargetNamespace()=e2.getTargetNamespace());
```

Where the query getTargetNamespace() returns the value of the targetNamespace tag of the closest containing <<Namespace>> Package.

8.4 NIEM PSM Profile

8.4.1 Overview

The NIEM PSM Profile comprises stereotypes that are used in NIEM PSMs. These stereotypes need not be used with a NIEM PIM, but they may be in order to provide additional platform-specific markup. Further, the NIEM PIM Profile imports the NIEM Common Profile and, therefore, includes all the stereotypes and metaclasses covered by that profile.

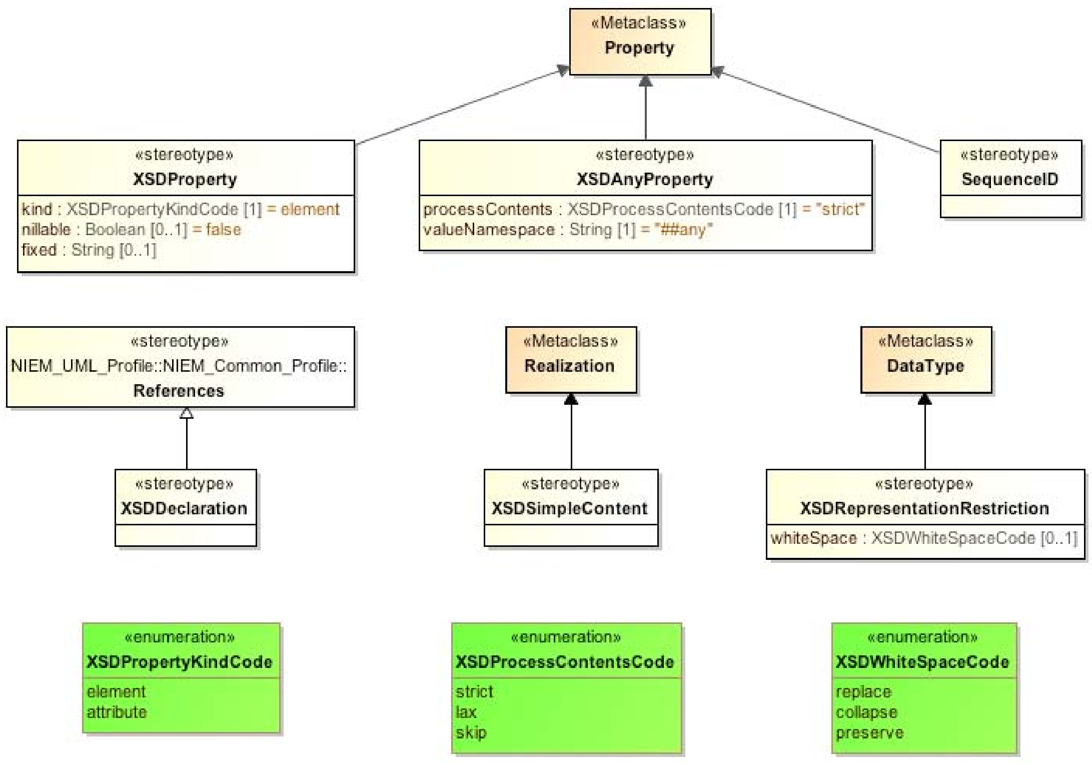


Figure 8.4 - NIEM PSM Profile

8.4.2 <Stereotype> SequenceID

Extends

- UML::Property

Description

A SequenceId Property is implemented in XML schema as a use of the structures:sequenceId property. The name of the property must be “sequenceId,” the type must be integer, and the multiplicity must be 1..1.

Constraints

- XSDStructureId

[OCL2.0]

```
self.base_Property.name = "sequenceId" and
self.base_Property.type.name = "integer" and
self.base_Property.type.oclIsKindOf(PrimitiveType) and
self.base_Property.lower = 1 and self.base_Property.upper = 1
```

8.4.3 <Stereotype> XSDAnyProperty

Extends

- UML::Property

Description

XSDAnyProperty stereotype represents a property that is unrestricted with respect to the properties type, which is implemented in XML Schema as the `xsd:any` particle.

Attributes

- processContents : NIEM_PSM_Profile :: XSDProcessContentsCode [1] {unique }

Determines how or if the value of a NIEM property should be processed; values are: “lax,” “skip,” and “strict.”

- valueNamespace :PrimitiveTypes::String [1] {unique }

The namespace in which values of this property must be defined. Implemented in XML Schema as the value of the namespace attribute on the `xsd:any` element.

Constraints

- XSDAnyPropertyType

[OCL2.0]

```
self.base_Property.type.oclIsUndefined() and
not(self.base_Property.isDerivedUnion) and
self.base_Property.subsettedProperty->isEmpty()
```

8.4.4 <Stereotype> XSDDeclaration

Generalization: NIEM_Common_Profile::References

Description

The XSDDeclaration stereotype is a specialization of the common References stereotype. However, it is constrained such that its client must be an XSDProperty Property and its supplier must be an XSDProperty Property or a Namepsace Package. By default, the namespace of the global XSD property declaration referenced by XSDProperty is the namespace of its class. The XSDDeclaration stereotype allows the modeler to specify the namespace a XSDProperty will reference based on the namespace of another XSDProperty or the target namespace of a Namespace Package. Specifically, the client of the XSDDeclaration Realization shall reference the namespace indicated by the supplier of the XSDDeclaration Realization, the client of the maps to one of the following: an attribute use schema component or a particle component whose term property is an element declaration schema component. In the first case, the supplier maps to the attribute declaration schema component for the attribute use component. In the second case, the supplier maps to the element declaration schema component for the particle schema component.

8.4.5 <Enumeration> XSDProcessContentsCode

Description

XSDProcessContentsCode supports the processContents attribute of the XSDAnyProperty stereotype.

Enumeration Literals

- lax
- skip
- strict

8.4.6 <Stereotype> XSDProperty

Extends

- UML::Property

Description

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 an XSDProperty Property is the client of a References Realization, then the supplier of the Realization defines the declaration of the NIEM property. Otherwise, the declaration of the NIEM property is defined implicitly to be the top-level attribute or element definition of the same name within the target namespace of the Namespace Package that contains the XSDProperty Property. All NIEM properties represented by XSDProperty Properties with the same name within the same package that are not clients of References Realizations share the same implicit attribute or element declaration.

Attributes

- fixed :PrimitiveTypes::String [0..1] {unique }

If present, implemented as the value of the fixed attribute of the `xsd:attribute` or `xsd:element`.

- kind : NIEM_PSM_Profile :: XSDPropertyKindCode [1] {unique }

Indicates whether the NIEM property is implemented in XML Schema as an attribute declaration and attribute use or element declaration and element particle: if "attribute," the NIEM property is implemented in XML Schema as an attribute declaration and attribute use; if "element," the NIEM property is implemented as an element declaration and element particle.

- nillable :PrimitiveTypes::Boolean [0..1] {unique }

Implemented in XML Schema as the value of the nillable attribute on the `xsd:element` element. Note that an XSDProperty that represents an XML attribute may not have a nillable value.

Constraints

- XSDPropertyAttributeKind

[OCL2.0]

```
( (self.kind=XSDPropertyKindCode::element) implies(
(self.base_Property.upper=1) and (self.base_Property.lower=1) and not
(self.base_Property.isDerivedUnion) and
self.base_Property.subsettedProperty->isEmpty() ) ) and(
not(self.base_Property.type.oclIsUndefined()) implies
self.base_Property.type.oclIsKindOf(DataType) )
```

- XSDPropertyElementKind

[OCL2.0]

```
( ( (self.kind=XSDPropertyKindCode::element) and
not(self.base_Property.type.oclIsUndefined()) ) implies
self.base_Property.type.stereotypedBy('NIEMType') ) and ( (
(self.kind=XSDPropertyKindCode::element) and
self.base_Property.type.oclIsUndefined() ) implies
self.base_Property.isDerivedUnion )
```

- XSDPropertyOwner

[OCL2.0]

```
self.base_Property.owner.oclIsKindOf(DataType) or
self.base_Property.owner.stereotypedBy(NIEMType)
```

8.4.7 <Enumeration> XSDPropertyKindCode

Description

XSDPropertyKindCode supports the kind attribute of XSDProperty by providing values to specify if an XSD property is represented as an `xsd:element` or `xsd:attribute`.

Enumeration Literals

- attribute
- element

8.4.8 <Stereotype> XSDRepresentationRestriction

Extends

- UML::DataType

Description

XSDRepresentationRestriction specifies a restriction on the representation in an XML schema of the values of a base DataType.

Attributes

- whiteSpace : NIEM_PSM_Profile :: XSDWhiteSpaceCode [0..1] {unique }

whiteSpace is a restriction on the value space of the DataType. It is implemented in XML Schema as the value of the value attribute on the `xsd:whiteSpace` element, the child of the `xsd:restriction` element that is the immediate child of the `xsd:simpleType` element.

Constraints

- must have one generalization

[OCL2.0]

self.base_DataType.generalization->notEmpty()

8.4.9 <Stereotype> XSDSimpleContent

Extends

- UML::Realization

Description

The «XSDSimpleContent» stereotype represents a relationship between two type definitions: the first is a complex type definition with simple content, the second is a simple type.

If the complex type definition is a <<Restriction>> of another complex type definition with simple content, then the simple type defines the constraining facets of the `xsd:restriction` to the other complex type.

Otherwise, the relationship is implemented in XML Schema through base attribute on the xsd:extension element of the first type definition, the actual value of which resolves to the second type definition. Section 3.4 of XML Schema Structures addresses simple content types in XML Schema; Sections 6.5.1, 6.5.2, and 7.4 of NIEM NDR v1.3 address simple content types in NIEM-conformant XML Schema.

Constraints

- Client must be a <<NIEMType>>

[OCL2.0]

self.base_Realization.client->forAll(c|c.stereotypedBy('NIEMType'))

- supplier must be a <DataType>

[OCL2.0]

self.base_Realization.supplier->forAll(s|s.oclIsKindOf(DataType))

8.4.10 <Enumeration> XSDWhiteSpaceCode

Description

Enumeration XSDWhiteSpaceCode supports the whiteSpace attribute of the XSDWhiteSpaceCode attribute as per the XSD definitions.

Enumeration Literals

- collapse
- preserve
- replace

8.5 Model Package Description Profile

8.5.1 Overview

The Model Package Description Profile comprises stereotypes that are used to model NIEM MPDs.

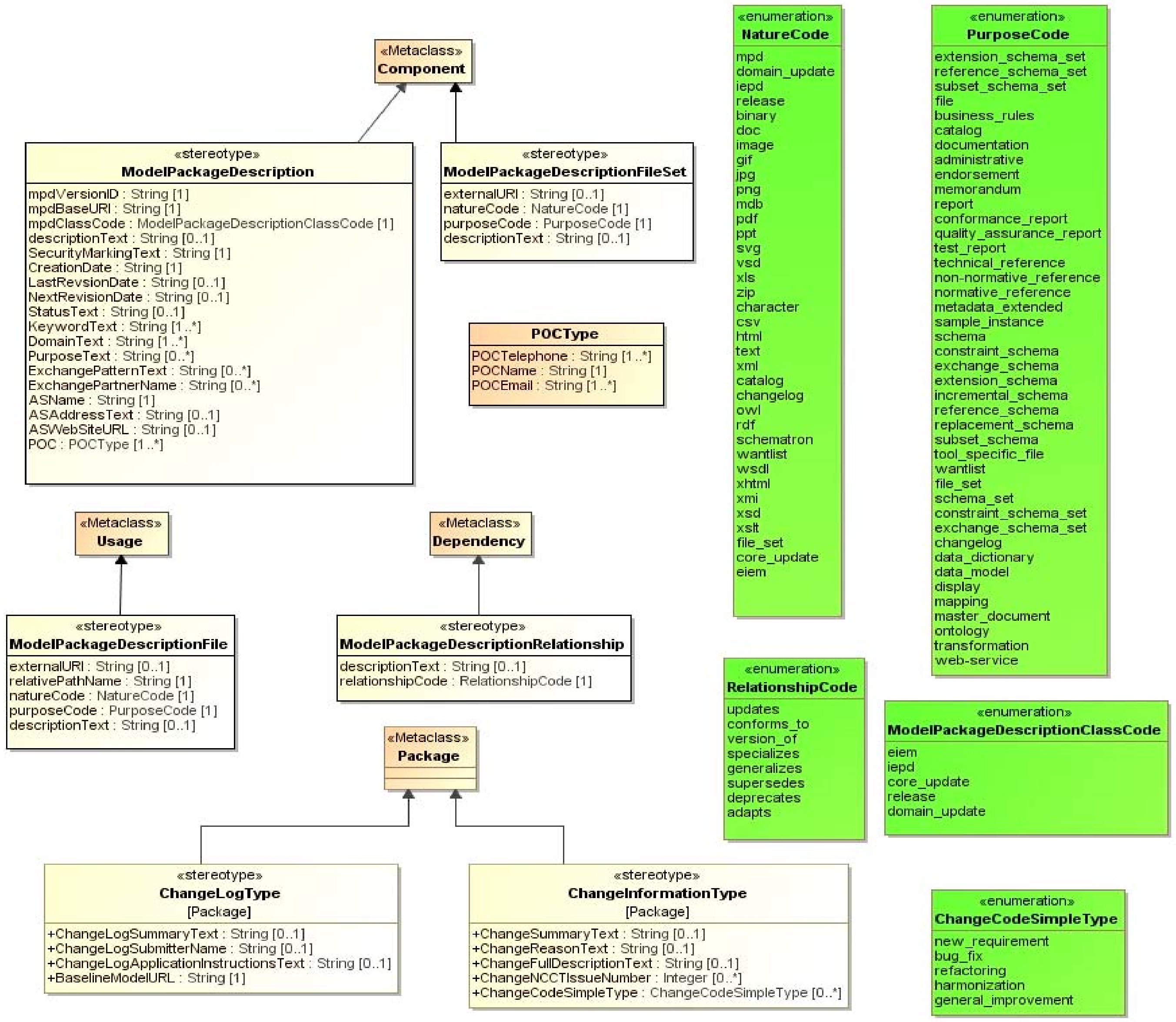


Figure 8.5 - Model Package Description Profile

8.5.2 <Stereotype> ModelPackageDescription

Extends

- UML::Component

Description

A ModelPackageDescription Component represents a NIEM Model Package Description (MPD). Specifically, it represents the information in an MPD catalog. Reference NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/)

An MPD is a logical set of electronic files aggregated and organized to fulfill a specific purpose in NIEM. Directory organization and packaging of an MPD should be designed around major themes in NIEM: reuse, sharing, interoperability, and efficiency. The inclusion of artifacts in an MPD is modeled using a Usage dependency from the Component representing the MPD to the model element representing the artifact (most commonly a Namespace Package).

Attributes

- ASAddressText :PrimitiveTypes::String [0..1] {unique ,composite }

An address or description for the location of the authoritative source for the MPD. Implemented as the value of the ASAddressText element in the catalog instance.

- ASName :PrimitiveTypes::String [1] {unique ,composite }

A name for the authoritative source for the MPD; can be author, creator, sponsor, etc. (person, organization, or entity). Implemented as the value of the ASName element in the catalog instance.

- ASWebSiteURL :PrimitiveTypes::String [0..1] {unique ,composite }

A URL for the Web site of the authoritative source for the MPD. Implemented as the value of the ASWebSiteURL element in the catalog instance.

- CreationDate :PrimitiveTypes::String [1] {unique ,composite }

Date this MPD was published or created. Implemented as the value of the CreationDate element in the catalog instance.

- DomainText :PrimitiveTypes::String [1..*] {unique ,composite }

A NIEM Domain applicable to, associated with, or that uses the MPD. Implemented as the value of the DomainText element in the catalog instance.

- ExchangePartnerName :PrimitiveTypes::String [*] {unique ,composite }

Name of an agency, organization, or entity that uses the MPD (in particular to share or exchange data). Implemented as the value of the ExchangePartnerName element in the catalog instance.

- ExchangePatternText :PrimitiveTypes::String [*] {unique ,composite }

A description of a transactional, design, or exchange pattern the MPD uses (generally, applicable to IEPDs only). Implemented as the value of the ExchangePatternText element in the catalog instance.

- KeywordText :PrimitiveTypes::String [1..*] {unique ,composite }

A keyword associated with the MPD; a common alias, term, or phrase that would help to facilitate search and discovery of this MPD. Implemented as the value of the KeywordText element in the catalog instance.

- LastRevsionDate :PrimitiveTypes::String [0..1] {unique }

Date the MPD was last revised. Implemented as the value of the LastRevisionDate element in the catalog instance.

- NextRevisionDate :PrimitiveTypes::String [0..1] {unique }

An estimate of the projected date the MPD is expected to be revised again (if known). Implemented as the value of the NextRevisionDate element in the catalog instance.

- PurposeText :PrimitiveTypes::String [*] {unique ,composite }

A description for the purpose, function, intended use of, or reason for the existence of the MPD. Implemented as the value of the PurposeText element in the catalog instance.

- SecurityMarkingText :PrimitiveTypes::String [1] {unique ,composite }

A label that defines how this MPD must be handled or can be distributed to protect the information it contains; the security marking for the MPD. Implemented as the value of the SecurityMarkingText element in the catalog instance. Default is “unclassified.”

- StatusText :PrimitiveTypes::String [0..1] {unique ,composite }

Description of the current state of development or usage of the MPD; may also project future plans for the MPD. Implemented as the value of the StatusText element in the catalog instance.

- descriptionText :PrimitiveTypes::String [0..1] {unique ,composite }

A description of the MPD. A statement that provides an explanation or additional detail. Implemented as the value of the descriptionText attribute of the Catalog element in the catalog instance.

- mpdBaseURI :PrimitiveTypes::String [1] {unique ,composite }

The left hand substring of an MPD URI that does not include its mpdVersionID. The concatenation of mpdBaseURI and mpdVersionID becomes the value of the mpdURI attribute of the Catalog element in the catalog instance.

- mpdClassCode : Model_Package_Description_Profile :: ModelPackageDescriptionClassCode [1] {unique ,composite }

The classification code of the MPD. Implemented as the value of the mpdClassCode attribute of the Catalog element in the catalog instance. This code designates the classification or kind of the MPD.

- mpdVersionID :PrimitiveTypes::String [1] {unique ,composite }

Many published MPDs will be periodically revised and updated; therefore, versioning is required to clearly indicate that changes have occurred. A version number is actually part of the unique identification for an MPD (to be discussed in a subsequent sub clause). All NIEM version numbers adhere to the regular expression: [0-9]+(\.[0-9]+)*((alpha|beta|rc|rev)[0-9]+)? Where: “alpha” indicates early development, “beta” indicates late development; but changing or incomplete “rc” indicates release candidate; complete but not approved as operational “rev” indicates very minor revision that does not impact schema validation.

Associations

- POC : Model_Package_Description_Profile :: POCType [1..*] {unique ,composite }

A point of contact (POC) for the authoritative source for the MPD; metadata used to contact the authoritative source. Implemented as a POCType with values for name, email, and telephone in the catalog instance.

Constraints

- MPD [Rule 3-06]

[OCL2.0]

```
( self.base_Component.profiledBy('NIEM_PSM_Profile') and
(self.mpdClassCode=ModelPackageDescriptionClassCode::iepd) ) implies
self.base_Component.elementImport ->exists(elementImport|
elementImport.stereotypedBy('ModelPackageDescriptionFile') and (
elementImport.getStereotypeApplication('ModelPackageDescriptionFile').purposeURI=
'http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#exchange-schema'
) and elementImport.importElement.oclAsType(Package).packagedElement
->exists(e|e.stereotypedBy('PropertyHolder') and
e.oclAsType(Class).ownedAttribute->notEmpty()) )
```

- MPD [Rule 3-09]

[OCL2.0]

```
( self.base_Component.profiledBy('NIEM_PSM_Profile') and (
(self.mpdClassCode=ModelPackageDescriptionClassCode::iepd) or
(self.mpdClassCode=ModelPackageDescriptionClassCode::eiem) ) ) implies
self.base_Component.elementImport
-
>select(elementImport|elementImport.stereotypedBy('ModelPackageDescriptionFile')).getStereotypeApplication('ModelPackag
eDescriptionFile').purposeURI
->exists(purposeURI|
(purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#subset-schema')
or
(purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
)
```

- MPD [Rule 3-10]

[English] This constraint is realized by PSM-MPD transformations.

- MPD [Rule 4-01]

[English] Constraint is resolved via PSM-MPD transformations which produce the catalog as specified by the MPD.

- MPD [Rule 4-02]

[OCL2.0]

```
not(self.mpdVersionID.oclIsUndefined()) and
(self.mpdVersionID<>'') -- This constraint also satisfied
by tag mpdVersionID, which is required to have a value.
```

- MPD [Rule 4-03.1]

[English] Satisfaction of this constraint requires comparative analysis between versions; cannot be expressed easily in OCL.

- MPD [Rule 4-03]

[OCL2.0]

```
self.mpdVersionID.match('[0-9]+(\\.[0-9]+)*((alpha|beta|rc|rev)[0-9]+)?')
```

- MPD [Rule 4-04]

[OCL2.0]

```
self.mpdBaseURI.repr().startsWith('http:/')
```

- MPD [Rule 4-06]

[English] Constraints for catalog construction are resolved in PSM-MPD transformation.

- MPD [Rule 4-07]

[English] Constraints for catalog construction are resolved in PSM-MPD transformations.

- MPD [Rule 4-08]

[English] All catalog constraints are resolved in PSM-MPD transformation.

- MPD [Rule 4-09]

[English] Explicit URI references are not modeled for well known artifacts such as schemas. Serialization of MPD artifacts such as schemas are via PSM-MPD transformations which construct URI references according to this MPD rule.

- MPD [Rule 4-10]

[English] Constraints on artifact URIs are resolved during PSM-MPD transformations.

- MPD [Rule 4-11]

[English] Constraints on changelog are resolved during PSM-MPD transformations.

- MPD [Rule 4-12]

[English] Constraints on changelogs are resolved by PSM-MPD transformations.

- MPD [Rule 4-13.1]

[English] Constraints on changelogs are resolved by PSM-MPD transformations.

- MPD [Rule 4-13]

[English] Constraints on changelogs are resolved during PSM-MPD transformation.

- MPD [Rule 4-14]

[English] Constraints on master document are resolved during PSM-MPD transformation.

- MPD [Rule 4-15]

[English] Constraints on master document are resolved by PSM-MPD transformations.

- MPD [Rule 6-1]

[English] Constraints on packaging are resolved during PSM-MPD transformation.

- MPD [Rule 6-2]

[English] This constraint is resolved by a combination of applying all specified NDR-based constraints and transformations to target artifacts.

- MPD [Rule 6-3]

[English] Packaging constraints are resolved by PSM-MPD transformations.

- MPD [Rule 6-3a]

[English] Packaging constraints are resolved by PSM-MPD transformations.

- MPD [Rule 6-3b]

[English] Packaging constraints are resolved by PSM-MPD transformations.

- MPD [Rule 6-3c]

[English] Packaging constraints are resolved by PSM-MPD transformations.

- MPD [Rule 6-3d]

[English] Packaging constraints are resolved by transformations.

- MPD [Rule 6-4]

[English] Packaging constraints are resolved by PSM-MPD transformations.

- MPD [Rule 6-5]

[English] Constraints on URIs are partially satisfied by specific URI Constraints expressed elsewhere in the NDR and MPD. For URI references embedded elsewhere in the model, it would be difficult to express the constraint in OCL. This constraint must be manually resolved by the modeler.

- MPD [Rule 6-6]

[English] This constraint is resolved by PSM-MPD transformations.

- MPD [Rule 6-7]

[English] An EIEM is an MPD with a packageCode of EIEM. An EIEM is typically bundled as a reusable model library which can be referenced from IEPDs. This relationship between EIEM and IEPD is used by PSM-MPD transformations to construct the catalog entries in resolution of this constraint.

- MPD [Rule 6-8]

[English] The schemaLocation constraints are resolved during PSM-MPD transformation.

- MPD [Rule 6-9]

[English] Packaging constraints are resolved by PSM-MPD transformations.

8.5.3 <Enumeration> ModelPackageDescriptionClassCode

Description

A specified classification (type or kind) of the MPD. Implemented as the value of the mpdClassCode attribute of the Catalog element in the catalog instance. One and only one classification is allowed for any given MPD. [Note these NIEM-UML enumeration literals differ from the NIEM MPD Specification v1.0 in that they use underscore (“_”) instead of dash (“-”). This is due to issues with dashes in some UML tools.] Reference 4.2.5 and Appendix B of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Enumeration Literals

- **core_update** - When necessary, the NIEM PMO can publish a core update. This is essentially identical to a domain update in terms of structure and use, with two important exceptions. First, a core update records changes that apply to a particular NIEM core version or another core update. This also means it is applicable to all NIEM releases using that same core version. Second, a core update is never published to replace a NIEM core. It is intended to add new schemas, new data components, new code values, etc. to a core without waiting for the next major release. In some cases, minor modifications to existing data components are possible.

- **domain_update** - A domain update is an MPD containing reference schemas that represent changes to NIEM domains. The [NIEM-HLVA] defines a domain update as both a process and a NIEM product. Through use and analysis of NIEM releases and published content, domain users will identify issues and new data requirements for the domain and sometimes Core. NIEM domains use these issues as the basis for incremental improvements, extensions, and proposed changes to future NIEM releases. Both the process and product of the process are referred to as domain update.

- **eiem** - An Enterprise Information Exchange Model (EIEM) is an MPD that incorporates BIECs that meet enterprise business needs for exchanging data using NIEM [NIEM-BIEC]. An EIEM is an adaptation of NIEM schemas, tailored and constrained for and by an enterprise. An EIEM will contain the following schemas that are commonly used or expected to be used by the authoring enterprise: one standard NIEM schema subset and one or more NIEM extension schemas that extend existing NIEM data components or establish new data components.

- **iepd** - NIEM Information Exchange Package Documentation (IEPD) is an MPD that defines a recurring XML data exchange. An NIEM IEPD is a set of valid XML schemas that may include portions of NIEM Core schemas, portions of NIEM Domain schemas, enterprise-specific or IEPD-specific extension schemas, and at least one exchange schema that defines a document element (as defined in [W3-XML-InfoSet]). The schemas contained in an IEPD work together to define a class of XML instances that consistently encapsulate data for information exchanges. Each XML instance in this class validates against the set of XML schemas contained within the IEPD.

- **release** - A NIEM release is an MPD containing a full set of harmonized reference schemas that coherently define all content within a single version of NIEM. NIEM releases include major, minor, and micro releases (as defined in the NIEM High Level Version Architecture (HLVA)).

8.5.4 <Stereotype> ModelPackageDescriptionFile

Extends

- UML::Usage

Description

The ModelPackageDescriptionFile stereotype applies to a Usage dependency that represents a relationship between an MPD or a file set and an artifact (generally a namespace) to be included in the MPD. Reference 4.2.3 and 4.2.4 of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Attributes

- descriptionText :PrimitiveTypes::String [0..1] {unique }

A description of the file. Implemented as the value of the descriptionText attribute of the File element in the catalog instance.

- externalURI :PrimitiveTypes::String [0..1] {unique }

An external URI for the file; indicates a same-as relationship to a copy of the file. Implemented as the value of the externalURI attribute of the File element in the catalog instance.

- natureCode : Model_Package_Description_Profile :: NatureCode [1] {unique }

The nature (type) of the file. Implemented as the value of the natureURI attribute of the File element in the catalog instance.

- purposeCode : Model_Package_Description_Profile :: PurposeCode [1] {unique }

The purpose for or function of the file. Implemented as the value of the purposeURI attribute of the File element in the catalog instance.

- relativePathName :PrimitiveTypes::String [1] {unique }

The relative path name to the file within the MPD directory structure. Implemented as the value of the relativePathName attribute of the File element in the catalog instance.

Constraints

- MPD [Rule 3-01]

[OCL2.0]

```
( self.base_ElementImport.profiledBy('NIEM_PSM_Profile') and (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#subset-schema')
) implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->forEach(subsetElement|
subsetElement.clientDependency->exists(d|d.stereotypedBy('References'))
and
subsetElement.clientDependency->select(d|d.stereotypedBy('References')).supplier
->forEach(referenceElement|
(subsetElement.name=referenceElement.name) and
(subsetElement.metaClass()=referenceElement.metaClass()) and(
subsetElement.oclIsKindOf(Namespace) implies
subsetElement.oclAstype(Namespace).ownedMember
->forEach(subsetMember|
referenceElement.oclAstype(Namespace).ownedMember
->exists(referenceMember|
(subsetMember.name=referenceMember.name) and
(subsetMember.metaClass()=referenceMember.metaClass()) ) ) ) ) )
```

- MPD [Rule 3-03]

[English] Constraint satisfied by other documentedComponent rules that exclude subset schemas.

- MPD [Rule 3-04]

[OCL2.0]

```
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#subset-schema'
```

```
implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->exists(element| element.clientDependency
->select(d|d.stereotypedBy('References')).supplier.getNearestPackage()

->select(r|r.stereotypedBy('Namespace')).getStereotypeApplication('Namespace').targetNamespace=
element.owner.getStereotypeApplication('Namespace').targetNamespace ) )
```

- MPD [Rule 3-05.1]

[English] Constraint on xsd:import is realized by the PSM-XSD transformations.

- MPD [Rule 3-05]

[English] UML well-formedness rules and semantics realize the constraint for referential integrity.

- MPD [Rule 3-07]

[English] Non-computable constraint.

- MPD [Rule 3-08]

[English] Constraint schemas are not modeled with the NIEM Profile. This constraint is not applied.

- NDR [Rule 5-1]

[English] This constraint is realized by PSM-XSD transformations.

- NDR [Rule 5-2]

[English] This constraint is realized by PSM-XSD transformations.

- NDR [Rule 5-3]

[English] This constraint is realized by PSM-XSD transformations.

  - NDR [Rule 5-4]

[English] Non-computable constraint.

- NDR [Rule 5-5]

[English] Non-computable constraint.

- NDR [Rule 6-01]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-02]

[English] This constraint realized by PSM-XSD transformations.

- NDR [Rule 6-03]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-04]

[English] Constrained realized by PSM-XSD transformations.

- NDR [Rule 6-05]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-06]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-07]

[English] This constraint is realized by not defining an xsd:anyType as part of the type system.

- NDR [Rule 6-08]

[English] Constraint is realized by PSM-XSD transformations.

- NDR [Rule 6-09]

[English] Constraint is realized by PSM-XSD transformations.

- NDR [Rule 6-12]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-13]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-14]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-15]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-16]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-17]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-19]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-20]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-21]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-22]

[English] Definitional.

- NDR [Rule 6-23]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-24]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-25]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-26]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-27]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-28]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-29]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-30]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-31]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-32]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-33]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-34]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-39]

[English] Constraint realized by PSM-MPD transformations.

- NDR [Rule 6-40]

[English] Since imports are derived (and not modeled), the namespace for an import is the same as the targetNamespace modeled for the referenced schema, and the implementation of NDR [Rule 6-36] resolves this constraint.

- NDR [Rule 6-41]

[English] Constraint realized by PSM-MPD transformations.

- NDR [Rule 6-42]

[English] This constraint is superceded by MPD [Rule 6-4]. Implementation of MPD [Rule 6-4] resolves this constraint.

- NDR [Rule 6-43]

[English] Constraint realized by PSM-MPD transformations.

- NDR [Rule 6-44]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-45]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-46]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-47]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-48]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-49]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-50]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-51]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-52]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-54]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-56]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-58]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 6-59]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-01]

[OCL2.0]

```
self.base_ElementImport.profiledBy('NIEM_PSM_Profile') implies ( (
(self.natureURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
or
(self.natureURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#extension-schema')
) implies
self.importedElement.getStereotypeApplication('Namespace').isConformant
)
```

- NDR [Rule 7-02]

[English] Not computable.

- NDR [Rule 7-03]

[English] Non-computable constraint.

- NDR [Rule 7-04]

[OCL2.0]

```
( self.base_ElementImport.profiledBy('NIEM_PSM_Profile') and ( (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
or (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#extension-schema')
) ) implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->select(c|c.oclIsKindOf(Class) and
not(c.stereotypedBy('PropertyHolder'))).oclAsType(Class)
->forAll(complexType |
complexType.ownedComment->exists(documentation|documentation.stereotypedBy('Documentation')))
```

- NDR [Rule 7-05]

[OCL2.0]

```
( self.base_ElementImport.profiledBy('NIEM_PSM_Profile') and ( (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
or (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#extension-schema')
) ) implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->select(c|c.oclIsKindOf(DataType)).oclAsType(DataType)
->forAll(simpleType |
simpleType.ownedComment->exists(documentation|documentation.stereotypedBy('Documentation')))
```

- NDR [Rule 7-06]

[OCL2.0]

```
( self.base_ElementImport.profiledBy('NIEM_PSM_Profile') and ( (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
or (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#extension-schema')
) ) implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->select(c|c.oclIsKindOf(Classifier)).oclAsType(Classifier).attribute
->select(p|p.stereotypeApplication('XSDProperty').kind=XSDPropertyKindCode::element)
->forAll(attribute |
attribute.ownedComment->exists(documentation|documentation.stereotypedBy('Documentation')))
```

• NDR [Rule 7-08]

[OCL2.0]

```
( self.base_ElementImport.profiledBy('NIEM_PSM_Profile') and ( (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#reference-schema')
or (
self.purposeURI='http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#extension-schema')
) ) implies
self.base_ElementImport.importedElement.oclAsType(Package).packagedElement
->select(c|c.oclIsKindOf(Enumeration)).oclAsType(Enumeration).ownedLiteral
->forAll(literal |
literal.ownedComment->exists(documentation|documentation.stereotypedBy('Documentation')))
```

• NDR [Rule 7-10]

[English] Non-computable constraint.

• NDR [Rule 7-11]

[English] Non-computable constraint.

• NDR [Rule 7-12]

[English] Constraint realized by PSM-XSD transformation.

• NDR [Rule 7-13]

[English] Non-computable constraint.

• NDR [Rule 7-15]

[English] Constraint realized by PSM-XSD transformations.

• NDR [Rule 7-16]

[English] Deprecated indicator not currently in NIEM Profiles, no constraint specified.

• NDR [Rule 7-17]

[English] Constraint realized by PSM-XSD transformations.

• NDR [Rule 7-18]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-19]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-20]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-21]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-22]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-23]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-30]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-31]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-32]

[English] Constraint resolved by PSM-XSD transformations.

- NDR [Rule 7-33]

[English] Constraint resolved by PSM-XSD transformations.

- NDR [Rule 7-34]

[English] Constraint resolved by PSM-XSD transformations.

- NDR [Rule 7-35]

[English] Constraint resolved by PSM-XSD transformations.

- NDR [Rule 7-36]

[English] Definitional.

- NDR [Rule 7-38]

[English] Constraint is resolved by PSM-XSD transformations. Constraint also addressed by UML Property order and naming constraints.

- NDR [Rule 7-39]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-51]

[English] Constraint resolved by PSM-XSD transformations.

- NDR [Rule 7-52]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-53]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-54]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-55]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-56]

[English] (Constraint not enforced by this model).

- NDR [Rule 7-57]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-58]

[English] Constraint realized by PSM-XSD transformations.

- NDR [Rule 7-59]

[English] Definitional.

- NDR [Rule 7-60]

[English] A Reference is modeled as a non-aggregate Property. A given property (from a referenced PropertyHolder) may be redefined to be an aggregate (i.e., containment) Property. The naming and type constraints are realized by PSM-XSD transformations.

- NDR [Rule 7-61]

[English] Constraint realized by PSM-MPD transformations, based on isConformant tag for referenced (external) schema.

- NDR [Rule 7-62]

[English] Constraint realized by PSM-MPD transformations, based on required documentation for modeled external schema.

- NDR [Rule 7-69]

[English] Constraint resolved by resolution to MPD [Rule 3-4]

- NDR [Rule 7-70]

[English] Constraint resolved by resolution to MPD [Rule 3-1]

- NDR [Rule 9-10]

[English] Non-computable constraint.

- NDR [Rule 9-11]

[English] Non-computable constraint.

- NDR [Rule 9-12]

[English] Non-computable constraint.

- NDR [Rule 9-13]

[English] Non-computable constraint.

- NDR [Rule 9-14]

[English] Non-computable constraint.

- NDR [Rule 9-15]

[English] Non-computable constraint.

- NDR [Rule 9-16]

[English] Non-computable constraint.

- NDR [Rule 9-17]

[English] Non-computable constraint.

- NDR [Rule 9-18]

[English] Non-computable constraint.

- NDR [Rule 9-20]

[English] Non-computable constraint.

- NDR [Rule 9-21]

[English] Non-computable constraint.

- NDR [Rule 9-2]

[English] Non-computable constraint.

- NDR [Rule 9-30]

[English] The only Attribute Group allowed in NIEM is the structures:SimpleObjectAttributeGroup. The constraint is realized by PSM-XSD transformations.

- NDR [Rule 9-4]

[English] Non-computable constraint.

- NDR [Rule 9-7]

[English] Non-computable constraint.

- NDR [Rule 9-8]

[English] Non-computable constraint.

- NDR [Rule 9-9]

[English] Non-computable constraint.

8.5.5 <Stereotype> ModelPackageDescriptionFileSet

Extends

- UML::Component

Description

A ModelPackageDescriptionFileSet Component represents a set of files in an MPD that are grouped for a specific purpose, function, or classification. For example, a set of MPD files might represent a schema subset, extension schema set, set of documentation, or set of test files. The MPD catalog uses the File element to represent artifacts, and the FileSet element to represent a set of artifacts. Note that both files and file sets are considered MPD artifacts. Reference 4.2.3 and 4.2.4 of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Attributes

- descriptionText :PrimitiveTypes::String [0..1] {unique }

A description of the file set. Implemented as the value of the descriptionText attribute of the FileSet element in the catalog instance.

- externalURI :PrimitiveTypes::String [0..1] {unique }

The external URI for the file set; indicates a same-as relationship to a copy of the file set. Implemented as the value of the externalURI attribute of the FileSet element in the catalog instance.

- natureCode : Model_Package_Description_Profile :: NatureCode [1] {unique }

The nature (type) of the file set. Implemented as the value of the natureURI attribute of the FileSet element in the catalog instance.

- purposeCode : Model_Package_Description_Profile :: PurposeCode [1] {unique }

The purpose or function of the file set. Implemented as the value of the purposeURI attribute of the FileSet element in the catalog instance.

Constraints

- unnamed1

[OCL2.0]

```
self.base_Package.namespace.stereotypedBy('ModelPackageDescription')
```

8.5.6 <Stereotype> ModelPackageDescriptionRelationship

Extends

- UML::Dependency

Description

The ModelPackageDescriptionRelationship stereotype applies to a Dependency that represents a relationship between MPDs or between an MPD and another resource (such as a NIEM specification; as in the case of conforms-to). There are many ways one MPD may relate to another. This makes it extremely difficult to specify a fixed set of values that could objectively define an exact relationship between a pair of MPDs. Therefore, the optional descriptionText attribute is provided to further explain the nature of any of the eight relationshipCode values available (version_of, specializes, generalizes, deprecates, supersedes, adapts, conforms_to, updates). In some cases, the value of relationshipCode may be generic enough to require a more detailed explanation in descriptionText (for example, if the value is “adapts”).

Attributes

- descriptionText :PrimitiveTypes::String [0..1] {unique ,composite }

A more detailed or specific textual explanation of the relationship between the MPDs or between an MPD and a resource (such as a specification). The catalog provides a Relationship element with three attributes (resourceURI, relationshipCode, and descriptionText) to identify the pedigree of an MPD. There are many ways that one MPD may relate to another. This makes it extremely difficult to specify a fixed set of values that could objectively define an exact relationship between a pair of MPDs. Therefore, the optional descriptionText attribute is provided to further explain the nature of any of the eight relationshipCode values available (version_of, specializes, generalizes, deprecates, supersedes, adapts, conforms_to, updates). In some cases, the value of relationshipCode may be generic enough to require a more detailed explanation in descriptionText (for example, if the value is “adapts”).

- relationshipCode : Model_Package_Description_Profile :: RelationshipCode [1] {unique ,composite }

A classification or reason for the connectedness between the MPDs or between an MPD and a resource.

8.5.7 <Enumeration> NatureCode

Description

An indication of the type of an MPD artifact. This further indicates how it should be processed by software tools. The literals of this enumeration correspond to MPD nature URIs of the form “http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/nature#<nature-code>.” [Note these NIEM-UML enumeration literals differ from the NIEM MPD Specification v1.0 in that they use underscore (“_”) instead of dash (“-”). This is due to issues with dashes in some UML tools.] Reference 4.2.5 and Appendix G of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Enumeration Literals

- binary

A nature of a binary file with unknown or unspecified encoding or format.

- catalog

A nature of a NIEM model package description (MPD) catalog file format.

- changelog
- character
- core_update

A nature of a NIEM core update model package description (MPD).

- csv
- doc
- domain_update
- eiem

A nature of a NIEM Enterprise Information Exchange Model (EIEM) model package description (MPD).

- file_set
- gif
- html
- iepd
- image
- jpg
- mdb
- mpd
- owl
- pdf
- png
- ppt
- rdf
- release
- schematron
- svg
- text
- vsd
- wantlist
- wsdl
- xhtml
- xls
- xmi
- xml
- xsd
- xslt
- zip

8.5.8 <Class> POCType

Description

A set of metadata used to contact the authoritative source for an MPD.

Attributes

- POCEmail :PrimitiveTypes::String [1..*] {unique ,composite }

An email address.

- POCName :PrimitiveTypes::String [1] {unique ,composite }

A name for a person, position, or title.

- POCTelephone :PrimitiveTypes::String [1..*] {unique ,composite }

A telephone number.

8.5.9 <Enumeration> PurposeCode

Description

An indication of the type of an MPD artifact. This further indicates how it should be processed by software tools. The literals of this enumeration correspond to MPD purpose URIs of the form "http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#". [Note these NIEM-UML enumeration literals differ from the NIEM MPD Specification v1.0 in that they use underscore ("_") instead of dash ("-"). This is due to issues with dashes in some UML tools.] Reference 4.2.4 and Appendix G of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Enumeration Literals

- administrative
- business_rules
- catalog
- changelog

Serves the purpose of recording all changes (additions, deletions, modifications) to a model package description relative to a previous version.

- conformance_report
- constraint_schema
- constraint_schema_set
- data_dictionary

Serves the purpose of a record of information about data (i.e., metadata) such as names, meaning, relationships to other data, origin, usage, and format.

- data_model

Serves the purpose of a formal representation (e.g., UML) of business data requirements that indicates data semantics, structure, and relationships.

- display

Serves the purpose of formatting information for viewing on a computer monitor.

- documentation

- endorsement
- exchange_schema
- exchange_schema_set
- extension_schema
- extension_schema_set
- file
- file_set
- incremental_schema
- mapping

Serves the purpose of describing how one data model or set of data components corresponds to another by identifying semantic equivalence or similarity.

- master_document

Serves the purpose of a primary source of documentation; a readme file; a first documentation source to consult and one that may reference other supplemental documentation.

- memorandum
- metadata_extended
- non_normative_reference
- normative_reference
- ontology

Serves the purpose of a standardized representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain, and may be used to describe the domain.

- quality_assurance_report
- reference_schema
- reference_schema_set
- replacement_schema
- report
- sample_instance
- schema
- schema_set
- subset_schema
- subset_schema_set
- technical_reference
- test_report
- tool_specific_file
- transformation

Serves the purpose of translating an artifact into another format or representation.

- wantlist
- web_service

Serves the purpose of establishing a method for communication between two electronic devices over a network.

8.5.10 <Enumeration> RelationshipCode

Description

The possible reasons for the connectedness between the MPDs or between an MPD and a resource. This enumeration defines the possible values for the relationshipCode attribute of the ModelPackageDescriptionRelationship stereotype. [Note these NIEM-UML enumeration literals differ from the NIEM MPD Specification v1.0 in that they use underscore ("_") instead of dash ("-"). This is due to issues with dashes in some UML tools.] Reference 4.2.5 and Appendix B of NIEM MPD Specification v1.0 (http://reference.niem.gov/niem/specification/model-package-description/1.0/).

Enumeration Literals

- **adapts** - A relationshipCode value for indicating that this MPD is an adaptation of the MPD referenced in resourceURI.
- **conforms_to** - A relationshipCode value for indicating that this MPD conforms to the specification or standard referenced in resourceURI.
- **deprecates** - A relationshipCode value for indicating that content in this MPD is preferred over content in the MPD referenced in resourceURI; and at some time in the future will supersede the MPD referenced in resourceURI.
- **generalizes** - A relationshipCode value for indicating that this MPD is a generalization of the MPD referenced in resourceURI. This value is the inverse of specializes.
- **specializes** - A relationshipCode value for indicating that this MPD is a specialization of the MPD referenced in resourceURI. This value is the inverse of generalizes.
- **supersedes** - A relationshipCode value for indicating that this MPD replaces the MPD referenced in resourceURI.
- **updates** - A relationshipCode value for indicating that this MPD is an incremental update to the resource referenced in resourceURI. Used by a core or domain update to identify the domain schema in a NIEM release being incrementally updated (not replaced).
- **version_of** - A relationshipCode value for indicating that this MPD is a different version of the MPD referenced in resourceURI. This code value is only needed in cases where significant name changes might obscure the relationship to the previous version. For example, NIEM Justice 4.1 is a version of GJXDM 3.0.3.

8.5.11 <Stereotype> ChangeLogType

Extends

- UML::Package

Description

The ChangeLogType stereotype applies to a Package that represents the required MPD changelog artifact. The changelog artifact contains descriptive information about the changelog as a whole. The attributes defined for <<ChangeLogType>> reflect the required changelog descriptive information.

Attributes

- ChangeLogSummaryText (String 0..1)

Descriptive text providing a summary of the change log.

- ChangeLogSubmitterName (String 0..1)

A name of the person, or organization submitting the change log.

- ChangeLogApplicationInstructionsText (String 0..1)

Descriptive text representing change log applications instructions.

- BaselineModelURL (String 1..1)

URL of baseline model the change log applies to.

8.5.12 <Stereotype> ChangeInformationType

Extends

- UML::Package

Description

The ChangeInformationType stereotype applies to a Package that represents one or more detailed change entries. The <<ChangeInformationType>> is a nested UML::Package of <<ChangeLogType>>. It contains descriptive information about one or more detailed change entries. The attributes defined for <<ChangeInformationType>> reflect the required changelog descriptive information for change entries. The change entries themselves, and their relationship with <<ChangeInformationType>> is an implementation detail not constrained by this specification.

Attributes

- ChangeSummaryText (String 0..1)

Text outlining a summary of a specific change contained in the change log.

- ChangeReasonText (String 0..1)

Descriptive text providing context to the reason a change noted in the change log was made.

- ChangeFullDescriptionText (String 0..1)

Descriptive text outlining the details of a specific change contained in a change log.

- ChangeNCCTIssueNumber (Integer 0..*)

Text outlining the NIEM Change Configuration Tool number associated to the specific change contained in the change log.

- ChangeCode (ChangeCodeSimpleType 0..*)

An enumeration of change codes based on the type of change that is contained in the change log.

8.5.13 <Enumeration> ChangeCodeSimpleType

Description

The ChangeCodeSimpleType represents the reason for change. ChangeCodeSimpleType instances are the values of the <<ChangeInformationType>> ChangeCode attribute.

Enumeration Literals

- new_requirement
- bug_fix
- refactoring
- harmonization
- general_improvement

9 NIEM-UML Transformation Reference

9.1 Introduction

This clause provides component, structural, and abstract orientation to the transformations between the UML Profile for NIEM and the concrete NIEM architectural artifacts, as specified in [NIEM-NDR] and [NIEM-MPD]. The transformations are expressed in terms of OMG QVT [QVT]. The QVT and related metamodels and profiles are provided as machine-readable artifacts associated with this specification (see Annex B). This clause, and its associated QVT, are presented from a transformation engineering perspective and illustrate abstract model manipulation. Other clauses in this specification provide illustrations of concrete target artifact syntax. The associated QVT are the normative expression for the mapping (in the sense defined in Clause 2). In case of apparent conflict between the orientation provided in this clause and the QVT, the QVT takes precedence.

9.1.1 NIEM Provisioning Context

The transformations referenced in this clause are intended to constitute provisioning process that enables representation of MPD artifacts as UML Models or in their native NIEM-conformant XML format. The overall provisioning process is illustrated in Figure 9.1. The focus of this clause is to transform UML Models between the NIEM PIM and NIEM PSM, and between the NIEM PSM and the MPD Artifacts. The MPD Artifacts addressed by these transformations are NIEM Conformant Schemas and the MPD Catalog. A meta-model for Schemas is specified in Clause 10 (XML Schema InfosetModel) of the OMG MOF 2 XMI Mapping Specification [XMI]. A metamodel for the MPD is included in the machine-readable artifacts for this specification (see Annex B). MPD Artifacts are represented (serialized) in their native XSD form.

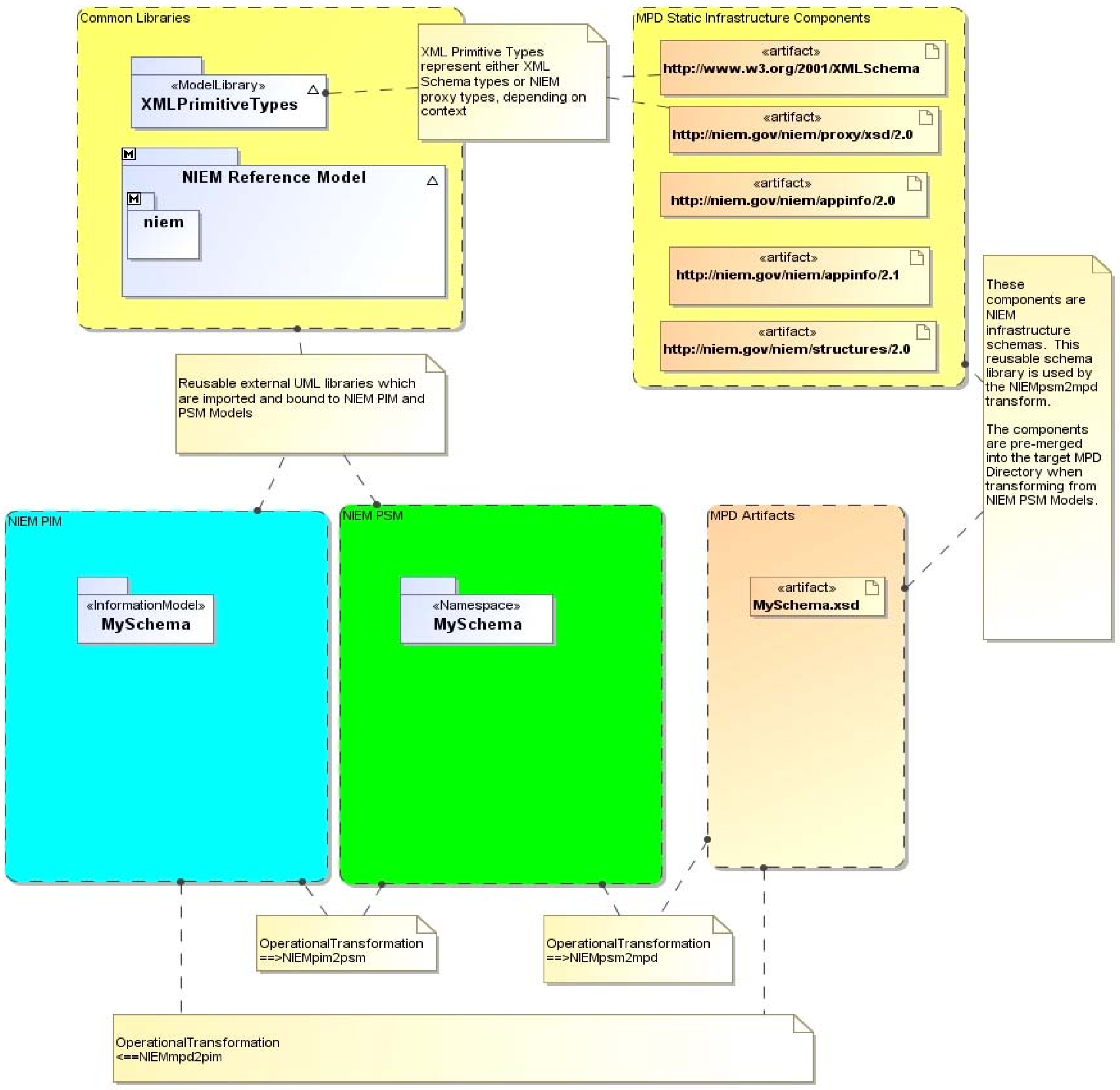
The NIEM MPD is pre-populated with a set of infrastructure schemas. During transformation, the schemas transformed from the UML Models are wired into these infrastructure schemas, as specified in the NIEM NDR. The components include:

- *structures* - The NIEM NDR Schema whose target namespace is “http://niem.gov/niem/structures/2.0.” Used primarily to provide the base definitions for top-level XSDComplexTypeDefinitions originating from the NIEM PSM.
- *xsd* - The NIEM NDR Schema whose target namespace is “http://niem.gov/niem/proxy/xsd/2.0.” This is the NIEM “proxy” schema and is used to “wrap” XML Primitive Types with XSDComplexTypeDefinitions in order to provide attributes carrying metadata.
- *appinfo* - Two versions of appinfo are included. The NIEM NDR Schema whose target namespaces are “http://niem.gov/niem/appinfo/2.0” and “http://niem.gov/niem/appinfo/2.1.” These schemas are primarily used to define names/namespaces used in NIEM Schema Annotations.
- *XML Schema* - A representation the XML Schema for Schemas. All versions of this schema are built into the XSD meta-model. The XML Schema for Schemas is not physically materialized in the MPD, but is referenced as the meta-model for all schemas, defining structure and constraints for all schema constructs. Additionally, it defines the SimpleTypeDefinitions corresponding to the XML Primitive Type library used within a NIEM UML model.

The transformations use a set of shared, reusable libraries for NIEM PIM and NIEM PSM:

- *XML Primitive Types* - The UML XML Primitive Types library represents the data types defined in the XML Schema for Schemas. There is an isomorphic mapping between the types in the UML XML Primitive Type library and the explicitly defined SimpleTypeDefinitions in the Schema for Schemas. In some cases, Schema for Schema datatypes are “wrapped” by ComplexTypeDefinitions within the NIEM Infrastructure xsd Proxy Schema in order to define meta information associated with the application of these datatypes.

- *NIEM Reference Models* - An optional extension to the NIEMmpdartifact2model transformation provides for binding NIEM subset schemas to NIEM reference schemas, thus enabling NIEM-NDR and NIEM-MPD conformance testing of NIEM MPD defined schema subsetting.



Common Libraries
«ModelLibrary»
XMLPrimitiveTypes
NIEM Reference Model
niem
XML Primitive Types represent either XML Schema types or NIEM proxy types, depending on context
MPD Static Infrastructure Components
«artifact»
http://www.w3.org/2001/XMLSchema
«artifact»
http://niem.gov/niem/proxy/xsd/2.0
«artifact»
http://niem.gov/niem/appinfo/2.0
«artifact»
http://niem.gov/niem/appinfo/2.1
«artifact»
http://niem.gov/niem/structures/2.0
These components are NIEM infrastructure schemas. This reusable schema library is used by the NIEMpsm2mpd transform.
The components are pre-merged into the target MPD Directory when transforming from NIEM PSM Models.
Reusable external UML libraries which are imported and bound to NIEM PIM and PSM Models
NIEM PIM
«InformationModel»
MySchema
NIEM PSM
«Namespace»
MySchema
MPD Artifacts
«artifact»
MySchema.xsd
OperationalTransformation ==>NIEMpim2psm
OperationalTransformation ==>NIEMpsm2mpd
OperationalTransformation <==NIEMmpd2pim

Figure 9.1 - NIEM Provisioning Context

The transformations referenced in this clause include:

- *NIEMpim2psm* - Transforms a NIEM PIM to a NIEM PSM.
- *NIEMpsm2xsd* - Transforms a NIEM PSM to MPD Schema Artifacts.

- *NIEMmpdmodel2artifact* - Transforms a NIEM MPD Model «ModelPackageDescription» and all its associated «Namespace»s to an MPD Catalog.xml and its associated NIEM Conformant Schemas.
- *NIEMmpdartifact2model* - Transforms an MPD Catalog and its associated set of MPD Schemas to a NIEM MPD Model.

Additionally, there are inherited common transformations:

- *NIEMplatformBinding* - A set of platform-specific operations. For the purposes of this specification, these are defined as abstract operations.
- *NIEMglobals* - A set of variables initialized at the beginning of the transformation, including references to Profiles and Stereotypes from NIEM-UML, and various constants referenced in the NIEM NDR and MPD.

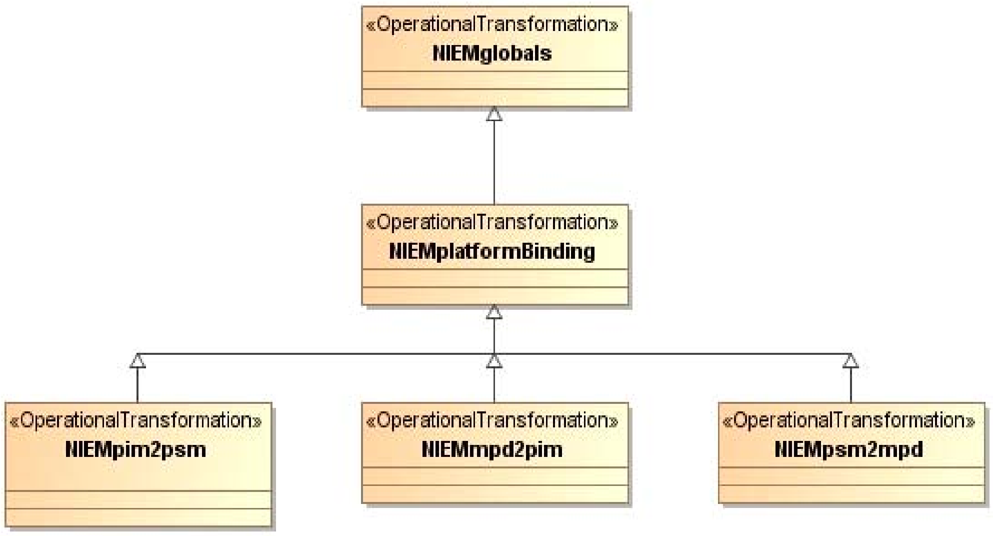

«OperationalTransformation»
NIEMglobals
«OperationalTransformation»
NIEMplatformBinding
«OperationalTransformation»
NIEMpim2psm
«OperationalTransformation»
NIEMmpd2pim
«OperationalTransformation»
NIEMpsm2mpd

Figure 9.2 - NIEM Transformations

9.1.2 Transformation Notation

Reuse and composition facilities are associated with QVT mapping operations. Disjunction enables selecting, among the set of disjunctive mappings, the first that satisfies the when clause and then invoking it. For the NIEM transformations, disjunction is used to identify a concrete MappingOperation to be selected from a given disjunctive MappingOperation. The disjunction hierarchy generally follows the Schema component inheritance hierarchy and/or the UML metamodel inheritance hierarchy. Another reuse and composition facility associated with QVT mapping operations is inheritance. Inheritance enables reuse of the execution logic of an inherited mapping. Thus, disjunction is used to initially select a leaf mapping operation and inheritance is used to share common execution logic. For the NIEM transformations, inheritance is used to identify the hierarchy of execution logic required to populate target Elements from a source Element. The mapping inheritance generally follows the Schema component inheritance hierarchy and/or the UML Meta-model inheritance hierarchy. Figure 9.3 illustrates the general pattern of disjunction and inheritance used for all transformations. A detailed disjunction/inheritance hierarchy is provided for each individual transformation.

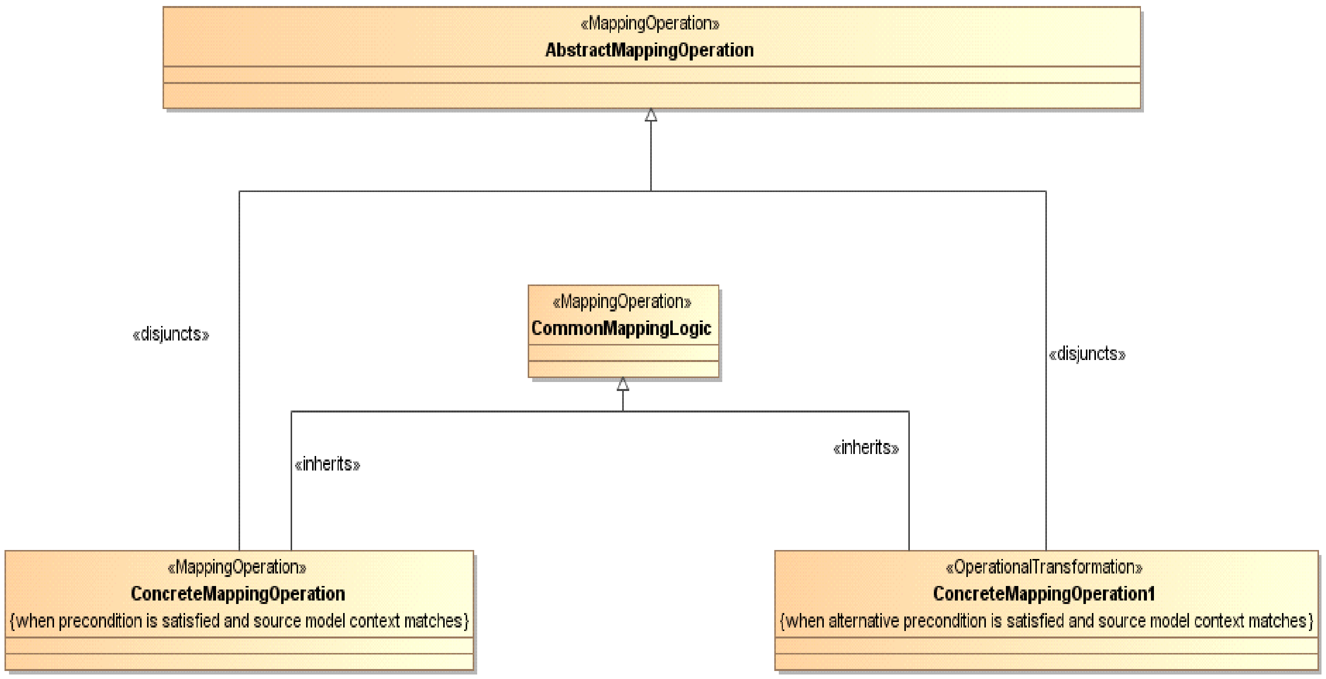

«MappingOperation»
AbstractMappingOperation
«disjuncts»
«MappingOperation»
CommonMappingLogic
«disjuncts»
«inherits»
«inherits»
«MappingOperation»
ConcreteMappingOperation
{when precondition is satisfied and source model context matches}
«OperationalTransformation»
ConcreteMappingOperation1
{when alternative precondition is satisfied and source model context matches}

Figure 9.3 - NIEM Transformation Disjunction and Inheritance

Figure 9.4 provides an example of how mappings are described for each transformation.

- Each transformation is decomposed into several abstract mapping figures, each figure depicting a related set of model concepts.
- Each mapping figure has two models depicted, one being the source and the other being the target of the transformation.
- Each model is adorned with sample model notation used to depict concepts associated with that model.
- MappingOperations are depicted as Realizations directed from a source model element to a target model element. In cases where a Realization cannot be depicted, a Comment is shown annotating one or more model elements from the source model and one or more model elements from the target model.
- Each MappingOperation is shown with the QVT mapping operation name. Details of the operation can be found in the associated QVT Files for this specification.
- Note that the figures in this clause are primarily intended as a high-level orientation to key «mappingOperations»s of the QVTs. Neither the figures nor the accompanying narrative provide all detail associated with a mapping operation. For definitive information about fine-grained aspects of the mapping, please consult the associated QVT Files for this specification.

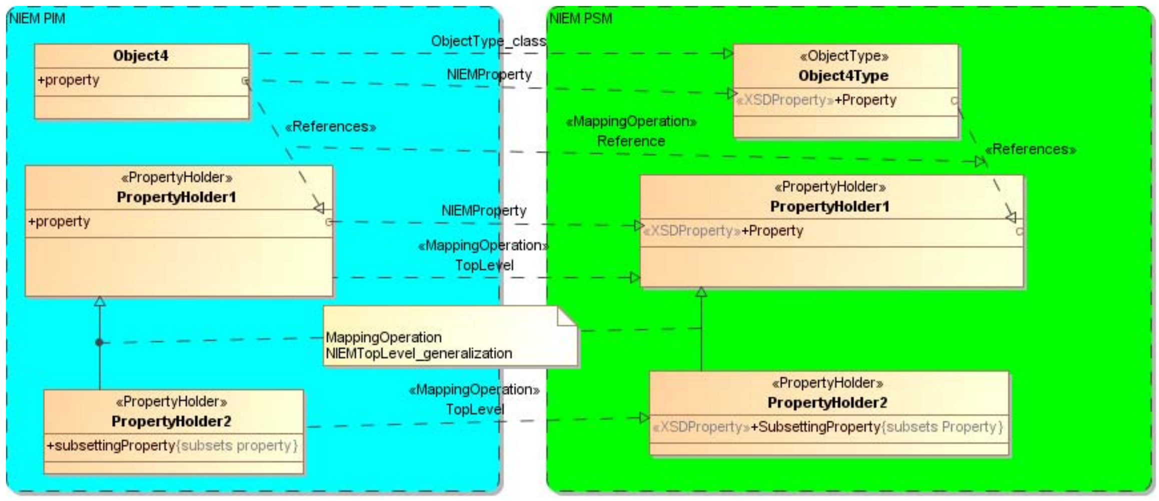


Figure 9.4 - NIEM Transformation Mapping Notation Overviews

9.1.3 Platform Binding

Platform Binding

There are variations in UML Platform implementations, particularly with respect to management of Profile/Stereotype/tag values. Some platforms implement Profiles via MOF, others provide implementation of applied Stereotypes via UML InstanceSpecifications. Transformation Operations that have variant implementations across platforms have been isolated from the specified transformations, enabling the core transformation to be applied to different platforms via a platform binding layer. In most cases, the variations can be specified directly in QVT. Examples of core UML utility functions that have platform variations include:

- *abstract query UML::Profile::getOwnedStereotype(stereotypeName:String):UML::Stereotype;*

  Retrieves the first Stereotype with the specified "Name" from the "Owned Stereotype" reference list.

- *abstract query UML::Element::getNearestPackage():UML::Package;*

  Retrieves the nearest package that owns (either directly or indirectly) this element, or the element itself (if it is a package).

- *abstract query UML::Element::isStereotypeApplied(stereotype:UML::Stereotype):Boolean;*

  Determines whether the specified stereotype is applied to this element.

- *abstract query UML::Element::getStereotypeApplication(stereotype:UML::Stereotype):Stdlib::Element;*

  Retrieves the application of the specified stereotype for this element, or null if no such stereotype application exists. The result is a Stdlib::Element, which may be implemented as a MOF instance or a UML <InstanceSpecification>, depending upon platform.

- *abstract helper Stdlib::Element::get<Classifier.name><Property.name>():<result>;*

  A basic getter for tag values. The context (Stdlib::Element) is an instance of a Classifier defined in the profile. <Classifier.name> is the name of the Classifier (without the XSD prefix). <Property.name> (first character capitalized) is the property to be retrieved.

<result> may be : an OCL Primitive type or Stdlib::Element (if it represents an instance of a Classifier in the Profile) or some form of OCL Collection of OCL Primitive types or Stdlib::Elements.

- *abstract helper Stdlib::Element::set<Classifier.name><Property.name>(value:<valueType>);*

  A setter for tag values. The context (Stdlib::Element) is an instance of a Classifier defined in the profile. <Classifier.name> is the name of the Classifier (without the prefix). <Property.name> (first character capitalized) is the property to be set. The value argument may be : an OCL Primitive type or some form of Enumeration defined within the Profile.

- *abstract helper Stdlib::Element::get<Classifier.name><Property.name>List():Stdlib::Element;*

  The context is an instance of a Classifier from the Profile. <Classifier.name> is the name of the Classifier (without the prefix). <Property.name> (first character capitalized) is the property to be retrieved. The value returned represents a logical "Slot" for a list of objects.

- *abstract helper Stdlib::Element::create<Classifier.name>Instance():Stdlib::Element;*

  The context is a logical "Slot." The operation creates an instance of the Classifier named <Classifier.name> from the Profile and adds it to the context.

- *abstract helper UML::MultiplicityElement::setLower(lower:Integer);*

  Context is a UML Multiplicity Element. Platform-specific implementation of setting the lower bound of the multiplicity interval.

- *abstract helper UML::MultiplicityElement::setUpper(upper:Integer);*

  Context is a UML Multiplicity Element. Platform-specific implementation of setting the upper bound of the multiplicity interval.

- *abstract helper UML::Package::applyProfile(profile : UML::Profile);*

  Context is a UML Package. Applies the current definition of the specified profile to this package and automatically applies required stereotypes in the profile to elements within this package's namespace hierarchy. If a different definition is already applied, automatically migrates any associated stereotype values on a "best effort" basis (matching classifiers and structural features by name).

- *abstract helper UML::Element::applyStereotype(stereotype:UML::Stereotype):Stdlib::Element;*

  Context is any UML Element. Applies the specified stereotype to this element. Returns an instance of the applied stereotype.

Global Properties

Property names are shared between the transformations. Properties may be one of the following kinds, depending upon the name syntax:

- *<name>Profile* - The value is a UML Profile initialized during transformation startup.
- *<name>Stereotype* - The value is a UML Stereotype initialized during transformation startup.
- *Other* - All other properties are string constants statically initialized.

9.2 NIEM PIM to NIEM PSM

The NIEMpim2psm transformation is defined as a set of mappings from the NIEM PIM Elements to Elements in the NIEM PSM. In general, there is a one-to-one correspondence between Elements in the NIEM PIM and Elements in the NIEM PSM. The transformation is minimal in the sense that any information in the NIEM PIM that is not relevant to the NIEM PSM is not mapped, which may include state machines, applications of foreign profiles and stereotypes, use cases, interfaces, ports, etc. Implementations of this transformation may extend the scope of mapped elements to include modeling constructs in support of provisioning target MPD artifacts not specifically addressed by this specification. Figure 9.5 illustrates the high-level packaging map between a NIEM PIM and a NIEM PSM with respect to the Model_Package_Description_Profile.

- Mapping to a NIEM PSM is driven from NIEM PIM ModelPackageDescription Component. The target NIEM PSM will contain all NIEM-relevant elements mapped from the NIEM PIM, including packaging structure nested to any level.
- A top level NIEM PSM Model is constructed for a «ModelPackageDescription» Component. NIEM PSM Profiles are applied to the target top-level model. The model will contain a mapping of the transitive closure of all «InformationModel» Packages directly or indirectly referenced via «ModelPackageDescriptionFile» Dependencies from the NIEM PIM «ModelPackageDescription» Component.
- NIEM PIM UML Packages nesting an «InformationModel» Package are mapped to the NIEM PSM, in the same relative containment structure. Nesting Packages are followed until a package with applied NIEM PIM Profiles is encountered. The nesting packages may be used to represent an MPD Folder Type.
- Stereotype applications from the NIEM PIM are cloned and applied to their mapped counterparts in the NIEM PSM.

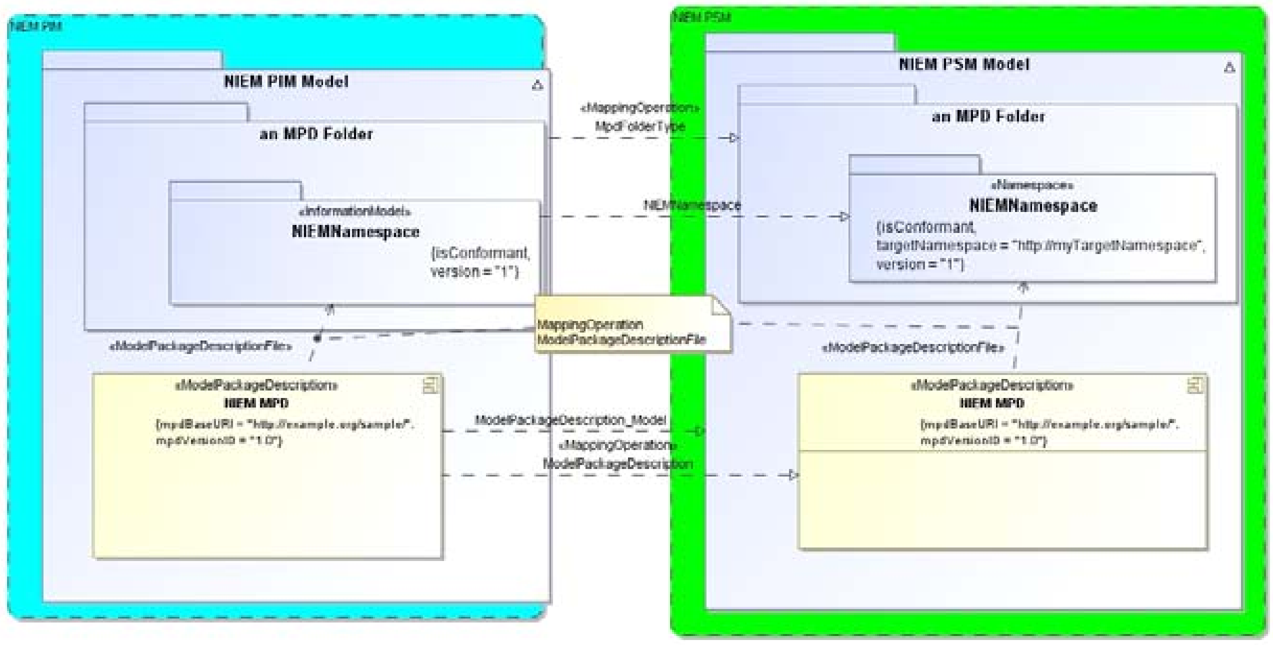

NIEM PIM
NIEM PIM Model
an MPD Folder
«InformationModel»
NIEMNamespace
{isConformant, version = "1"}
«ModelPackageDescriptionFile»
«ModelPackageDescription»
NIEM MPD
{mpdBaseURI = "http://example.org/sample/", mpdVersionID = "1.0"}
«MappingOperation»
MpdFolderType
NIEMNamespace
MappingOperation
ModelPackageDescriptionFile
ModelPackageDescription_Model
«MappingOperation»
ModelPackageDescription
NIEM PSM
NIEM PSM Model
an MPD Folder
«Namespace»
NIEMNamespace
{isConformant, targetNamespace = "http://myTargetNamespace", version = "1"}
«ModelPackageDescriptionFile»
«ModelPackageDescription»
NIEM MPD
{mpdBaseURI = "http://example.org/sample/", mpdVersionID = "1.0"}

Figure 9.5 - NIEM PIM to NIEM PSM - Model Package Description Profile Mapping Overview

Figure 9.6 illustrates mappings between NIEM PIM and NIEM PSM Perspectives related to the NIEM_PSM_Profile. Many of the NIEM PIM Elements are mapped to nearly identical counterpart NIEM PSM Elements within the NIEM_PSM_Profile. Variations from an isomorphic representation include:

- NIEM PIM primitives are modeled as PrimitiveTypes and/or Enumerations. On the NIEM PSM side they are mapped to Class, if they do not contain constraining facets or enumeration literals. A UML Generalization in the NIEM PIM is mapped to a «Restriction» in the target NIEM PSM.
- Naming in a NIEM PIM may need to be coerced to be compliant with NDR naming rules during map to NIEM PSM. This includes representation terms for Property names based on derivation from specific XML Primitive types.
- Generalizations in NIEM PIM may map to Properties in the target NIEM PSM.
- Explicit stereotype application of NIEM concepts in the NIEM PIM may map to semantic elements based on NDR naming rules in the target NIEM PSM.
- Use of Associations in NIEM PIM may map to simple Properties in the target NIEM PSM.
- NIEM PIM comments may be adjusted according to Standard Opening Phrase rules in NDR when mapped to NIEM PSM.
- The foundational XML Primitive Types library is used to represent XML Primitives for both the NIEM PIM and NIEM PSM.

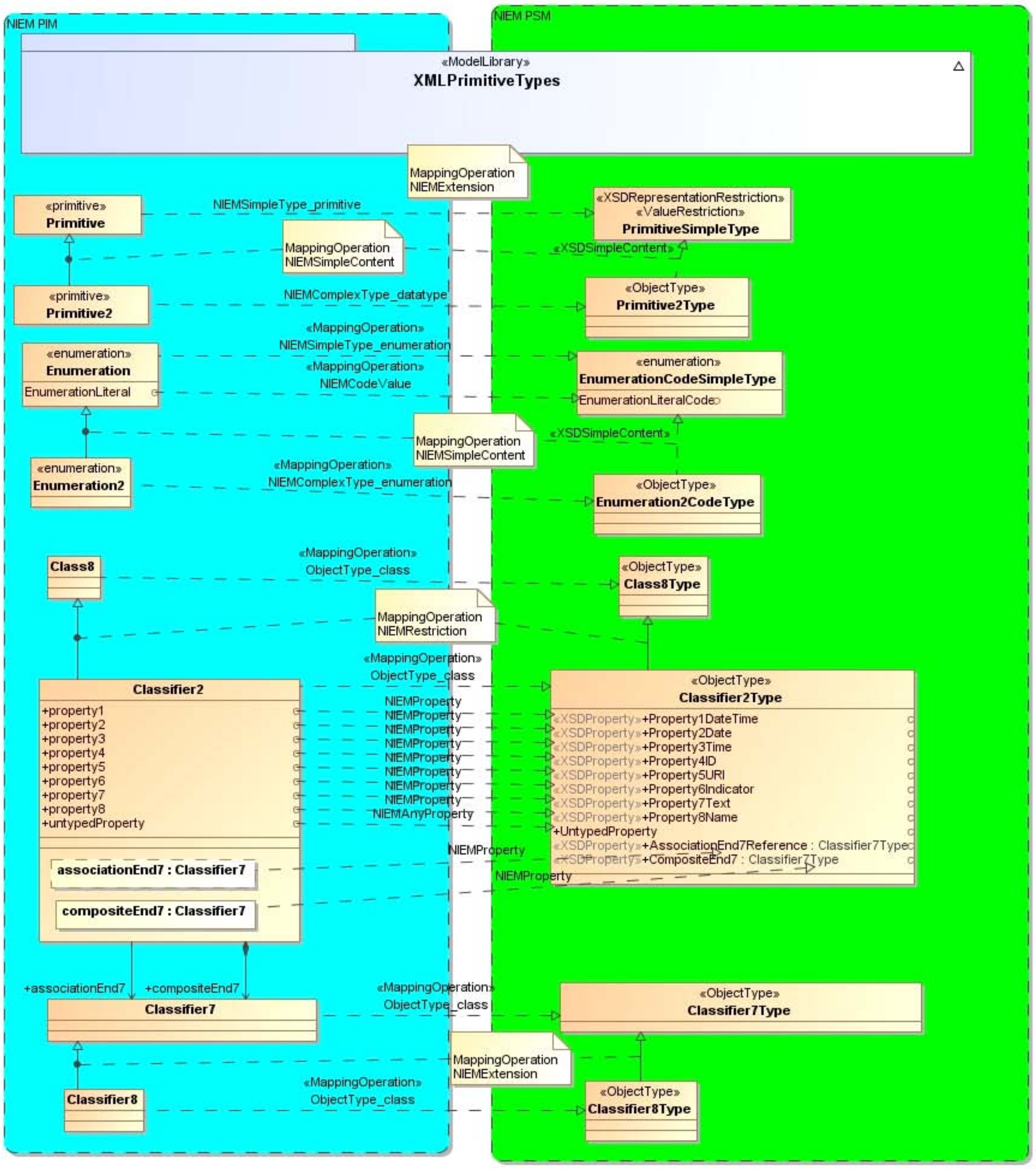

NIEM PIM
NIEM PSM
«ModelLibrary» XMLPrimitiveTypes
MappingOperation NIEMExtension
«primitive» Primitive
NIEMSimpleType_primitive
«XSDRepresentationRestriction» «ValueRestriction» PrimitiveSimpleType
MappingOperation NIEMSimpleContent
«XSDSimpleContent»
«primitive» Primitive2
NIEMComplexType_datatype
«ObjectType» Primitive2Type
«enumeration» Enumeration
EnumerationLiteral
«MappingOperation» NIEMSimpleType_enumeration
«MappingOperation» NIEMCodeValue
«enumeration» EnumerationCodeSimpleType
EnumerationLiteralCode
MappingOperation NIEMSimpleContent
«XSDSimpleContent»
«enumeration» Enumeration2
«MappingOperation» NIEMComplexType_enumeration
«ObjectType» Enumeration2CodeType
Class8
«MappingOperation» ObjectType_class
«ObjectType» Class8Type
MappingOperation NIEMRestriction
«MappingOperation» ObjectType_class
Classifier2
+property1
+property2
+property3
+property4
+property5
+property6
+property7
+property8
+untypedProperty
associationEnd7 : Classifier7
compositeEnd7 : Classifier7
NIEMProperty
NIEMAnyProperty
«ObjectType» Classifier2Type
«XSDProperty»+Property1DateTime
«XSDProperty»+Property2Date
«XSDProperty»+Property3Time
«XSDProperty»+Property4ID
«XSDProperty»+Property5URI
«XSDProperty»+Property6Indicator
«XSDProperty»+Property7Text
«XSDProperty»+Property8Name
+UntypedProperty
«XSDProperty»+AssociationEnd7Reference : Classifier7Type
«XSDProperty»+CompositeEnd7 : Classifier7Type
+associationEnd7
+compositeEnd7
Classifier7
«MappingOperation» ObjectType_class
«ObjectType» Classifier7Type
MappingOperation NIEMExtension
Classifier8
«MappingOperation» ObjectType_class
«ObjectType» Classifier8Type

Figure 9.6 - NIEM PIM to NIEM PSM - PSM Profile Mapping Overview

The Figure 9.7 illustrates some additional mappings between NIEM PIM and NIEM PSM related to the NIEM_PSM_Profile. With the exception of some naming coercion, the mappings depicted in this diagram are isomorphic.

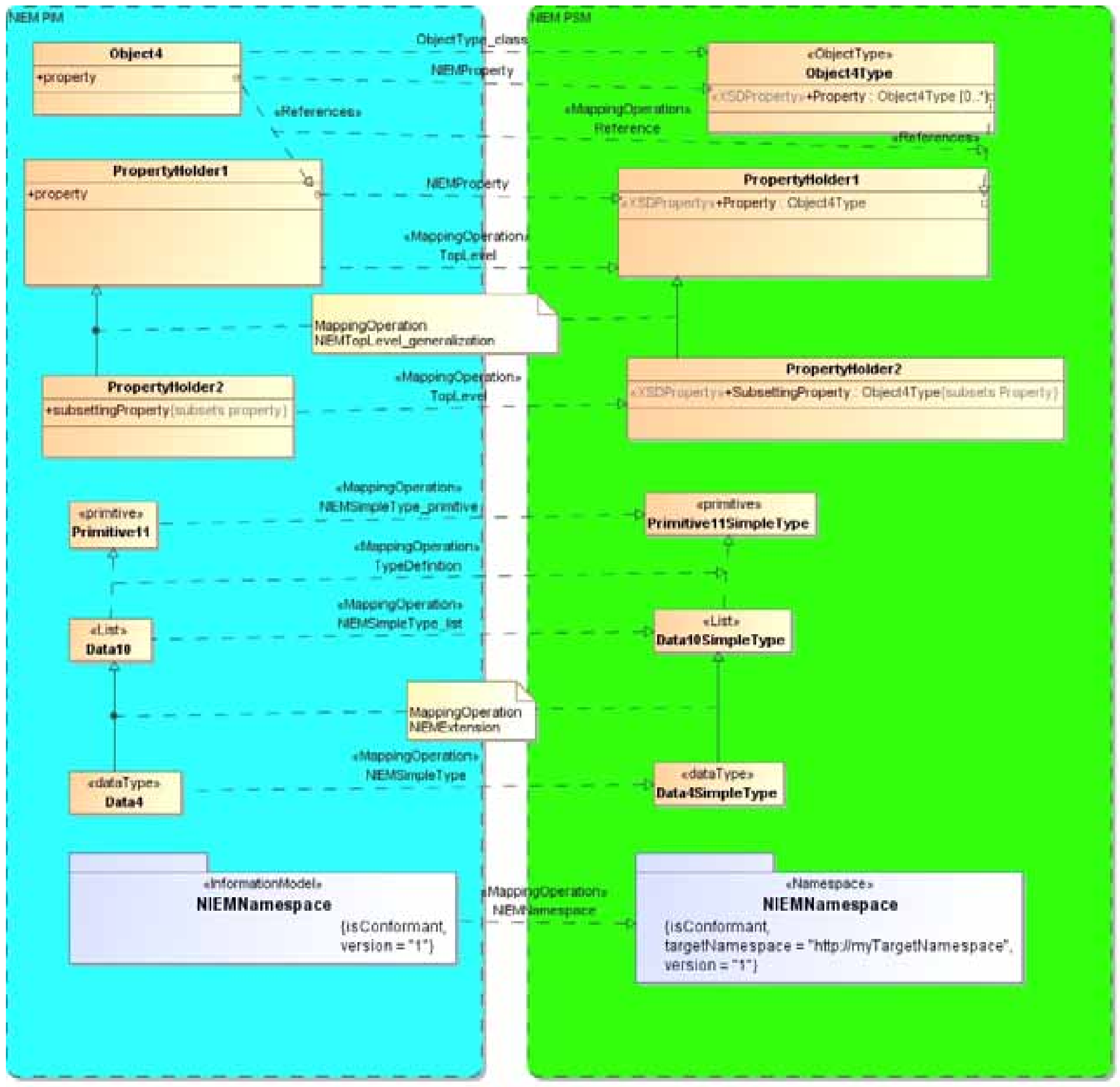


Figure 9.7 - NIEM PIM to NIEM PSM - PSM Profile Mapping Overview (2)

The NIEM_PIM_Profile provides alternate notations, constraints, and defaults for modeling NIEM. In many cases, the NIEM_PIM_Profile Elements are mapped to nearly identical counterpart NIEM PSM Elements. Variations from an isomorphic representation include:

- Naming in NIEM PIM is not necessarily constrained to NIEM NDR naming rules. Transformations must coerce names to be compliant with NDR naming rules during map to NIEM PSM.
- Generalizations within NIEM PIM may map to Properties in the target NIEM PSM.

- Explicit stereotype application of NIEM concepts within NIEM PIM may map to semantic elements based on NDR naming rules in the target NIEM PSM (such as “RoleOf”).

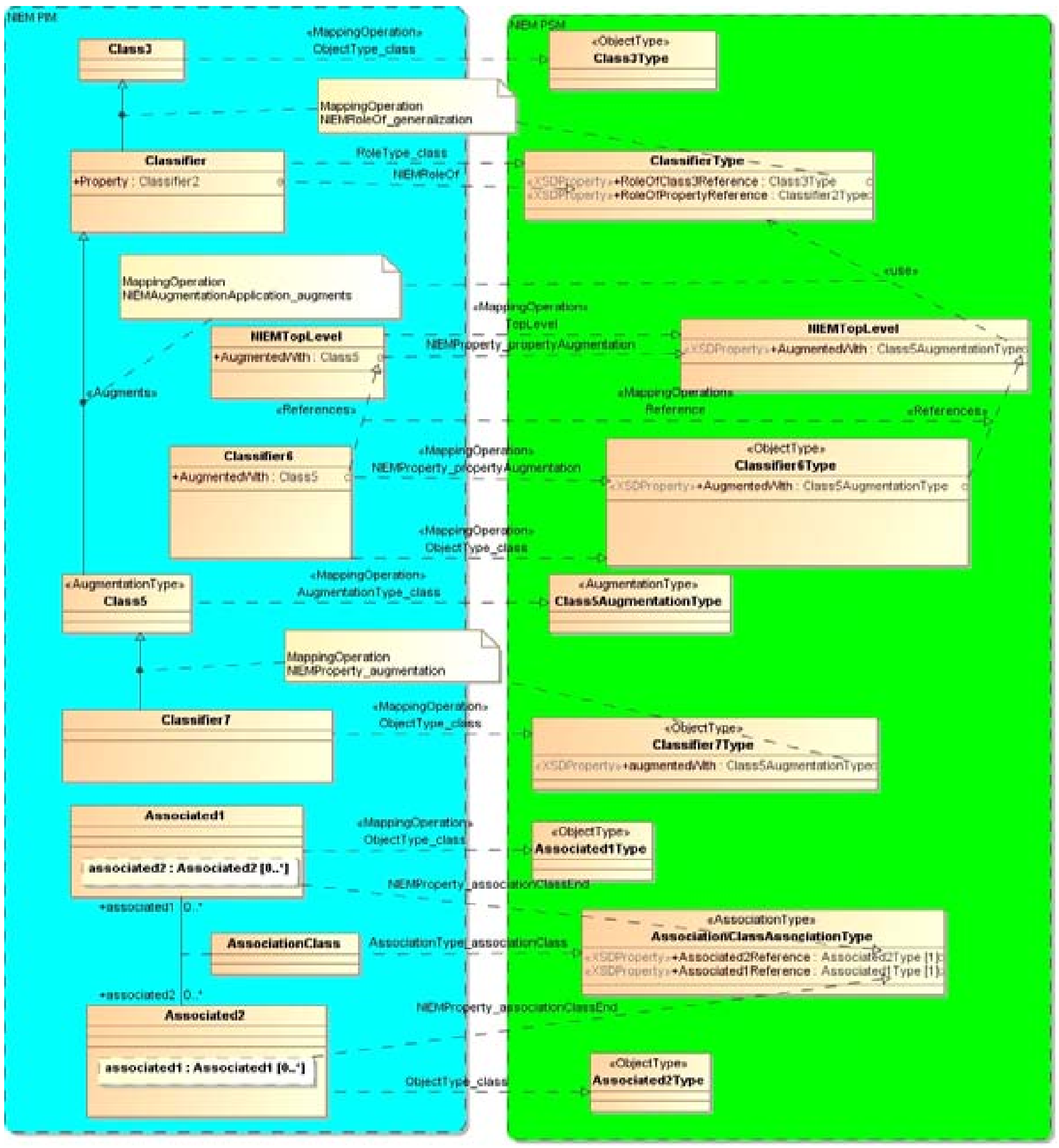


Figure 9.8 - NIEM PIM to NIEM PSM - PIM Profile Mapping Overview

Many of the NIEM PIM Elements within the NIEM_Common_Profile are mapped to nearly identical counterparts in the target NIEM PSM. Variations from an isomorphic representation include:

- Naming within NIEM PIM may need to be coerced to be compliant with NDR naming rules during map to NIEM PSM.
- Generalizations within NIEM PIM may map to Properties in the target NIEM PSM.

- Explicit stereotype application of NIEM concepts within NIEM PIM may map to semantic elements based on NDR naming rules in the target NIEM PSM.
- Use of Associations within NIEM PIM map to simple Properties in the target NIEM PSM.

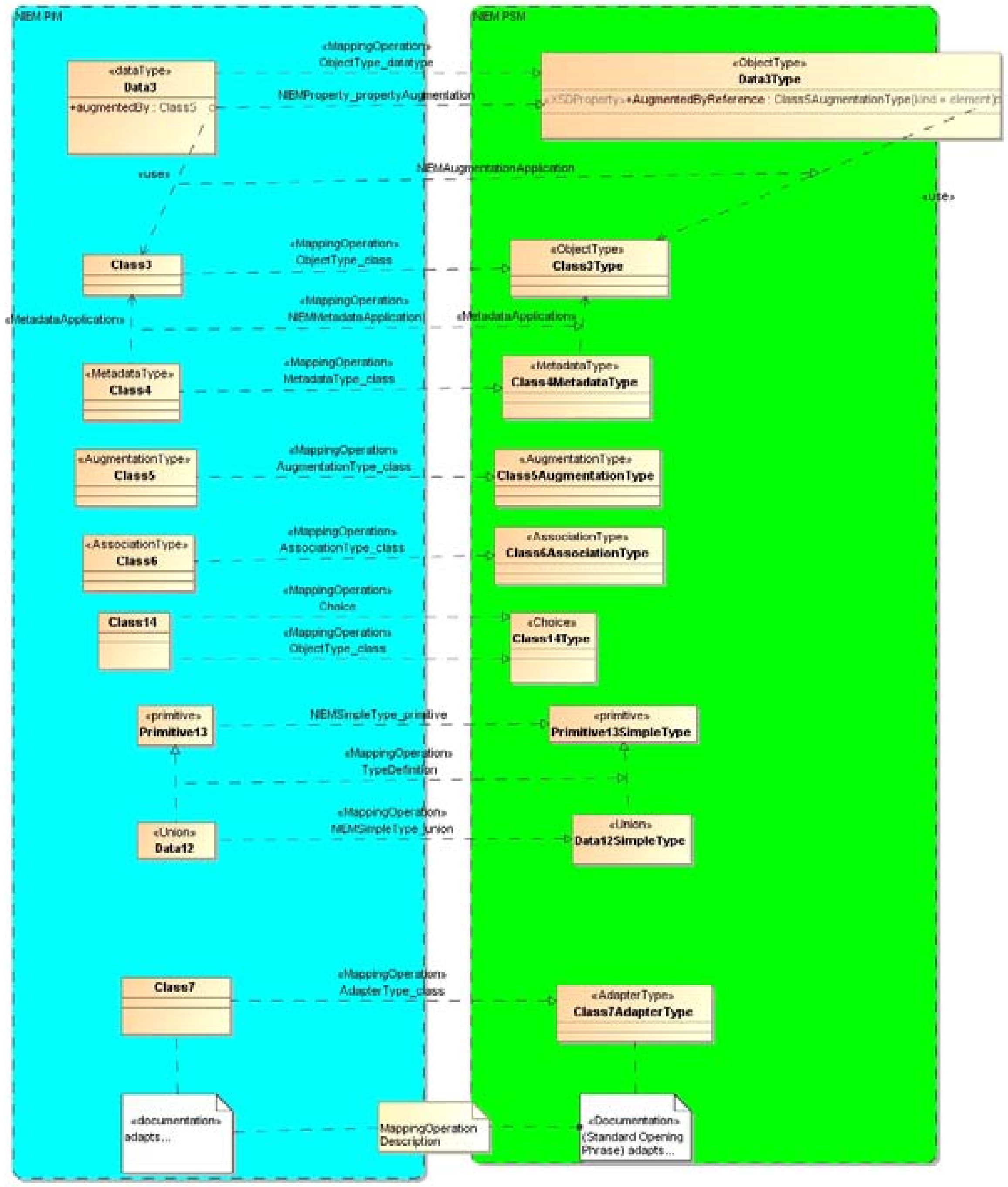

NIEM PIM
NIEM PSM
«dataType» Data3
+augmentedBy : Class5
«MappingOperation» ObjectType_datatype
NIEMProperty_propertyAugmentation
«ObjectType» Data3Type
«XSDProperty» +AugmentedByReference : Class5AugmentationType{kind = element}
«use»
NIEMAugmentationApplication
«use»
«MappingOperation» ObjectType_class
Class3
«ObjectType» Class3Type
«MetadataApplication»
«MappingOperation» NIEMMetadataApplication
«MetadataApplication»
«MetadataType» Class4
«MappingOperation» MetadataType_class
«MetadataType» Class4MetadataType
«AugmentationType» Class5
«MappingOperation» AugmentationType_class
«AugmentationType» Class5AugmentationType
«AssociationType» Class6
«MappingOperation» AssociationType_class
«AssociationType» Class6AssociationType
Class14
«MappingOperation» Choice
«MappingOperation» ObjectType_class
«Choice» Class14Type
«primitive» Primitive13
NIEMSimpleType_primitive
«primitive» Primitive13SimpleType
«MappingOperation» TypeDefinition
«Union» Data12
«MappingOperation» NIEMSimpleType_union
«Union» Data12SimpleType
Class7
«MappingOperation» AdapterType_class
«AdapterType» Class7AdapterType
«documentation» adapts...
MappingOperation Description
«Documentation» (Standard Opening Phrase) adapts...

Figure 9.9 - NIEM PIM to NIEM PSM - Common Profile Mapping Overview

- An unstereotyped AssociationClass within NIEM PIM maps to a Class stereotyped as AssociationType in target NIEM PSM, with some cardinality adjustments.
- Use of Generalizations related to PrimitiveTypes/SimpleTypes may map to Dependencies in target NIEM PSM.
- NIEM PIM defaults, including Documentation, ObjectType, ValueRestriction are explicitly stereotyped in target NIEM PSM.
- NIEM PIM comments may be adjusted according to Standard Opening Phrase rules in NDR when mapped to target NIEM PSM.

For NIEMpim2psm, mapping operations are generally invoked with the context of a source NIEM PIM Element and produce some form of target NIEM PSM Element. Most mapping operations are also provided an argument that is the NIEM PSM container context. The "when" condition for the MappingOperations is normally a function of the source NIEM PIM element, the type of the source NIEM PIM element, the source NIEM PIM element's applied stereotype, and/or the target NIEM PSM container context. The mapping operation connects its target NIEM PSM Element to its container, applies a Stereotype as appropriate (or clones the NIEM PIM Stereotype application), and populates the underlying UML Element properties and/or Stereotype Application tag values. Figure 9.10 illustrates the disjunction pattern for the NIEMpim2psm transformation.

Figure 9.11 illustrates the inheritance for most of the MappingOperations in the NIEMpim2psm transformation. Figure 9.12 illustrates the remaining MappingOperations for the NIEMpim2psm transformation.

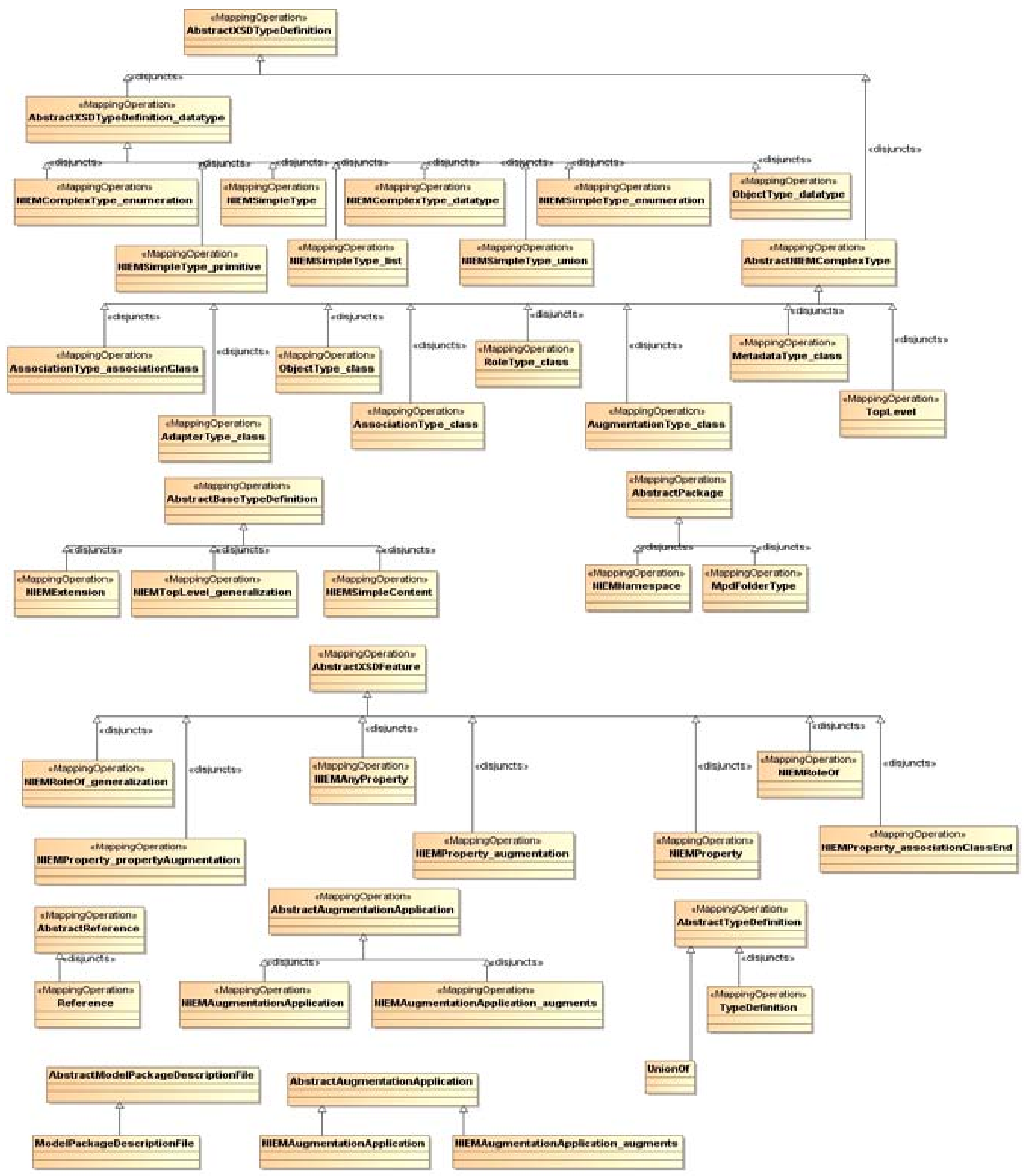

«MappingOperation» AbstractXSDTypeDefinition
«disjuncts»
«MappingOperation» AbstractXSDTypeDefinition_datatype
«MappingOperation» NIEMComplexType_enumeration
«MappingOperation» NIEMSimpleType
«MappingOperation» NIEMComplexType_datatype
«MappingOperation» NIEMSimpleType_enumeration
«MappingOperation» ObjectType_datatype
«MappingOperation» NIEMSimpleType_primitive
«MappingOperation» NIEMSimpleType_list
«MappingOperation» NIEMSimpleType_union
«MappingOperation» AbstractNIEMComplexType
«MappingOperation» AssociationType_associationClass
«MappingOperation» ObjectType_class
«MappingOperation» RoleType_class
«MappingOperation» MetadataType_class
«MappingOperation» TopLevel
«MappingOperation» AdapterType_class
«MappingOperation» AssociationType_class
«MappingOperation» AugmentationType_class
«MappingOperation» AbstractBaseTypeDefinition
«MappingOperation» NIEMExtension
«MappingOperation» NIEMTopLevel_generalization
«MappingOperation» NIEMSimpleContent
«MappingOperation» AbstractPackage
«MappingOperation» NIEMNamespace
«MappingOperation» MpdFolderType
«MappingOperation» AbstractXSDFeature
«MappingOperation» NIEMRoleOf_generalization
«MappingOperation» NIEMAnyProperty
«MappingOperation» NIEMRoleOf
«MappingOperation» NIEMProperty_propertyAugmentation
«MappingOperation» NIEMProperty_augmentation
«MappingOperation» NIEMProperty
«MappingOperation» NIEMProperty_associationClassEnd
«MappingOperation» AbstractReference
«MappingOperation» Reference
«MappingOperation» AbstractAugmentationApplication
«MappingOperation» NIEMAugmentationApplication
«MappingOperation» NIEMAugmentationApplication_augments
«MappingOperation» AbstractTypeDefinition
«MappingOperation» TypeDefinition
UnionOf
AbstractModelPackageDescriptionFile
ModelPackageDescriptionFile
AbstractAugmentationApplication
NIEMAugmentationApplication
NIEMAugmentationApplication_augments

Figure 9.10 - NIEM PIM to NIEM PSM Disjunction

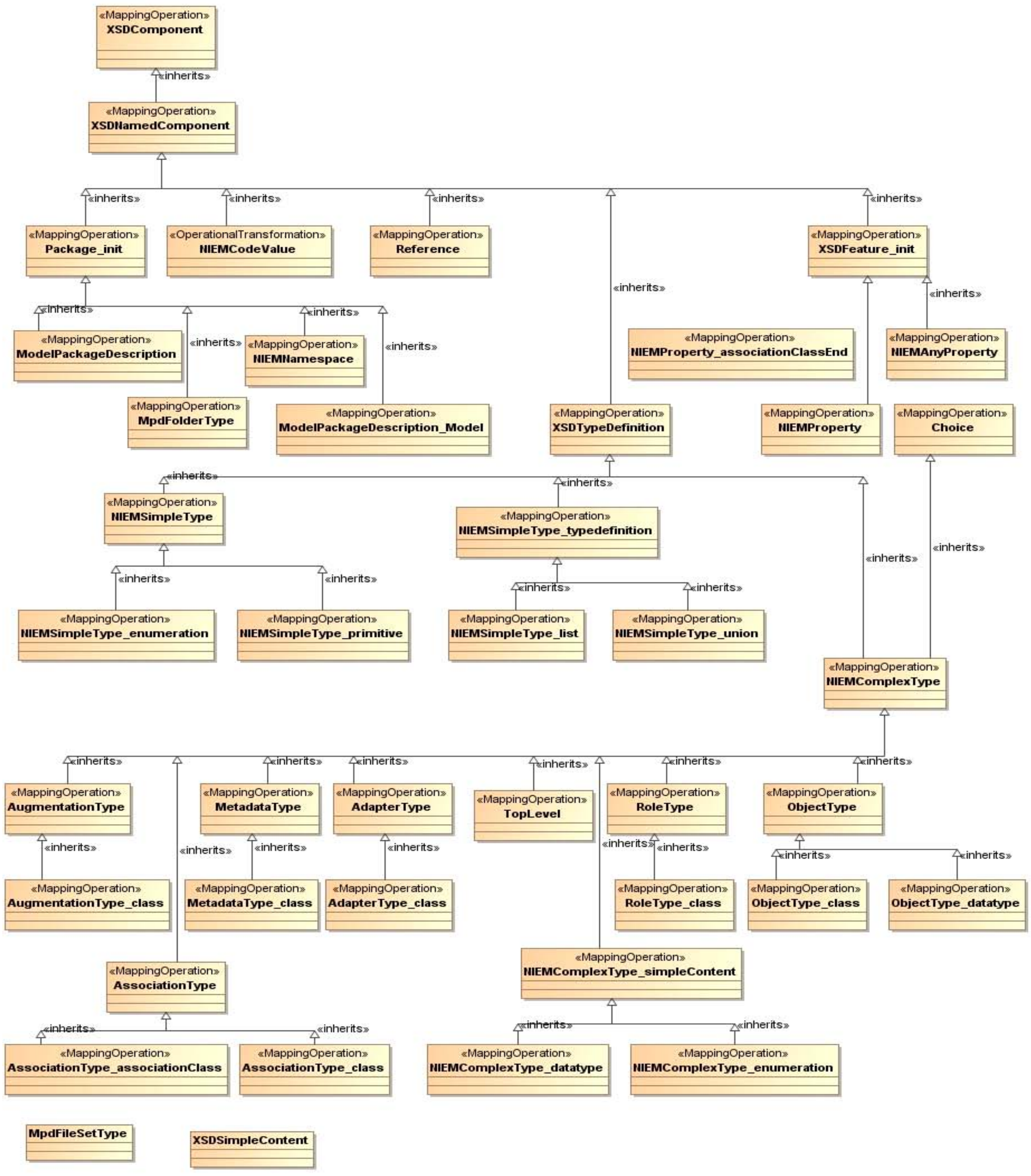

«MappingOperation» XSDComponent
«inherits»
«MappingOperation» XSDNamedComponent
«MappingOperation» Package_init
«OperationalTransformation» NIEMCodeValue
«MappingOperation» Reference
«MappingOperation» XSDFeature_init
«MappingOperation» ModelPackageDescription
«MappingOperation» NIEMNamespace
«MappingOperation» MpdFolderType
«MappingOperation» ModelPackageDescription_Model
«MappingOperation» NIEMProperty_associationClassEnd
«MappingOperation» NIEMAnyProperty
«MappingOperation» XSDTypeDefinition
«MappingOperation» NIEMProperty
«MappingOperation» Choice
«MappingOperation» NIEMSimpleType
«MappingOperation» NIEMSimpleType_typedefinition
«MappingOperation» NIEMSimpleType_enumeration
«MappingOperation» NIEMSimpleType_primitive
«MappingOperation» NIEMSimpleType_list
«MappingOperation» NIEMSimpleType_union
«MappingOperation» NIEMComplexType
«MappingOperation» AugmentationType
«MappingOperation» MetadataType
«MappingOperation» AdapterType
«MappingOperation» TopLevel
«MappingOperation» RoleType
«MappingOperation» ObjectType
«MappingOperation» AugmentationType_class
«MappingOperation» MetadataType_class
«MappingOperation» AdapterType_class
«MappingOperation» RoleType_class
«MappingOperation» ObjectType_class
«MappingOperation» ObjectType_datatype
«MappingOperation» AssociationType
«MappingOperation» NIEMComplexType_simpleContent
«MappingOperation» AssociationType_associationClass
«MappingOperation» AssociationType_class
«MappingOperation» NIEMComplexType_datatype
«MappingOperation» NIEMComplexType_enumeration
MpdFileSetType
XSDSimpleContent

Figure 9.11 - NIEM PIM to NIEM PSM Inheritance (1)

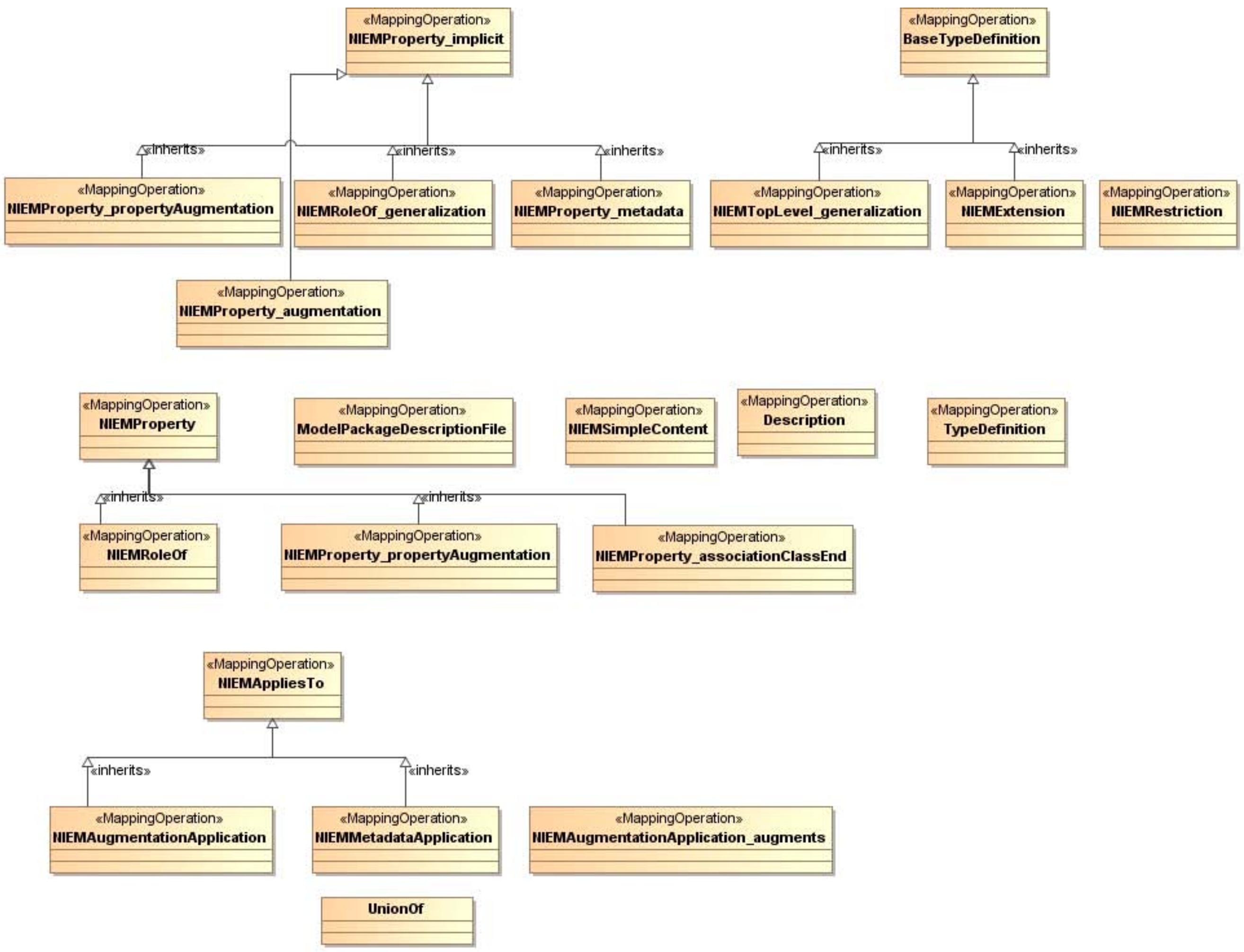

«MappingOperation» NIEMProperty_implicit
«MappingOperation» BaseTypeDefinition
«inherits»
«MappingOperation» NIEMProperty_propertyAugmentation
«MappingOperation» NIEMRoleOf_generalization
«MappingOperation» NIEMProperty_metadata
«MappingOperation» NIEMTopLevel_generalization
«MappingOperation» NIEMExtension
«MappingOperation» NIEMRestriction
«MappingOperation» NIEMProperty_augmentation
«MappingOperation» NIEMProperty
«MappingOperation» ModelPackageDescriptionFile
«MappingOperation» NIEMSimpleContent
«MappingOperation» Description
«MappingOperation» TypeDefinition
«MappingOperation» NIEMRoleOf
«MappingOperation» NIEMProperty_propertyAugmentation
«MappingOperation» NIEMProperty_associationClassEnd
«MappingOperation» NIEMAppliesTo
«MappingOperation» NIEMAugmentationApplication
«MappingOperation» NIEMMetadataApplication
«MappingOperation» NIEMAugmentationApplication_augments
UnionOf

Figure 9.12 - NIEM PIM to NIEM PSM Inheritance (2)

9.3 NIEM PSM to NIEM-Conforming XML Schema

There are various forms of metadata embodied in the components of a NIEM conformant Schema. The metadata are represented in schemas as text-based user/application information embodied within an XSDAnnotation. The metadata includes documentation, cross-component references, and extended properties for the XSDComponents. All forms of XSDComponent metadata are based on NIEM-NDR rules for representation within constructs of an XSDAnnotation. Figure 9.13 illustrates many specific cases of metadata usage:

- XSDAnnotations are owned by an XSDComponent. The ownership association name and semantic varies by specific XSDComponent. The UML Element/ownedComment association maps to one of the XSDComponent-specific ownership associations with XSDAnnotation.
- An XSDAnnotation has a property “userInformation” that contains an `xsd:documentation` (DOM) Element. A UML «Documentation» Comment body is mapped to the textual value of the `xsd:documentation` Element.

- An XSDAnnotation also has a property “`applicationinformation`” that contains an `xsd:appinfo` (DOM) Element. The `xsd:appinfo` Element contains (DOM) Elements defined by the NDR rules. All NDR defined elements are either in the NIEM appinfo namespace or the NIEM appinfo2 namespace.
- appinfo:ConformantIndicator is used to indicate NIEM conformance for an XSDSchema or an XSDImport of an XSDSchema. The UML «Namespace» isConformant tag is mapped to the value of the `appinfo:ConformantIndicator` for either a target XSDSchema or a referencing XSDImport.
- `appinfo:Base` is a NIEM defined element that has 2 attributes: `appinfo:name` and `appinfo:namespace`. Together, the name and namespace uniquely reference an XSDComponent within a particular symbol space. The value of `appinfo:name` and appinfo:namespace are mapped from UML Generalizations, «Restriction»s, and/or NDR rules regarding default `appinfo:Base`, depending upon the specific «NIEMType».
- `appinfo:ReferenceTarget` is a NIEM defined element that has 2 attributes: `appinfo:name` and `appinfo:namespace`. Together, the name and namespace uniquely reference an XSDTypeDefinition. The value of `appinfo:name` and `appinfo:namespace` are mapped from a UML Property type declaration for a “Reference” Element. In these cases the typeDefinition of the target XSDElementDeclaration is set to `structures:ReferenceType`.
- `appinfo:AppliesTo` is a NIEM defined element that has 2 attributes: `appinfo:name` and `appinfo:namespace`. Together, the name and namespace uniquely reference an XSDTypeDefinition. The value of `appinfo:name` and appinfo:namespace are mapped from the supplier of a UML «Application» Usage.
- `appinfo:ExternalAdapterTypeIndicator` is a NIEM-defined (DOM) Element. A UML «AdapterType» is mapped to an `appinfo:ExternalAdapterTypeIndicator` with value “true.”

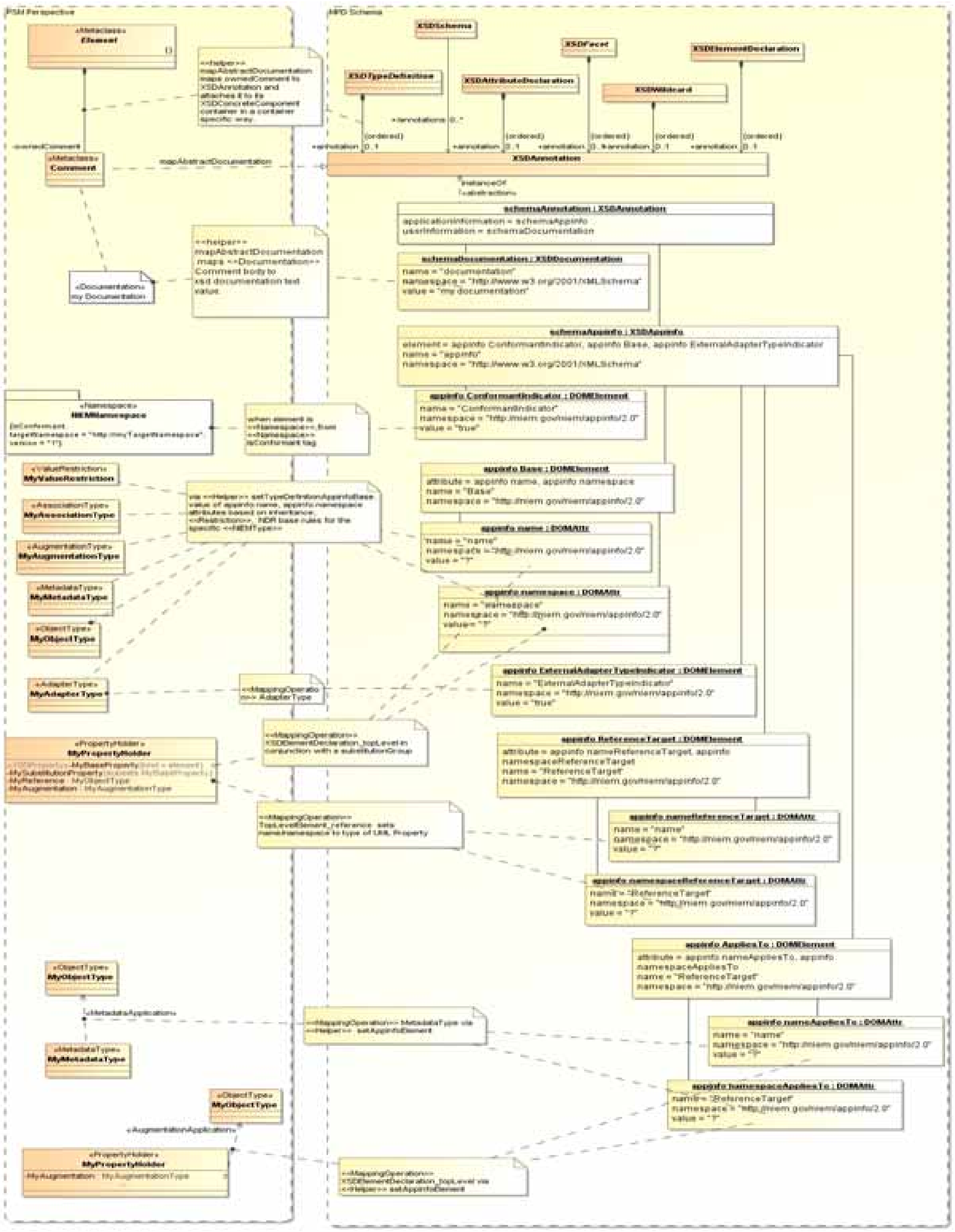


Figure 9.13 - NIEM PSM to MPD Schema Artifacts - Annotation Mapping Overview

A UML «Namespace» maps to an XSDSchema, as illustrated in Figure 9.14.

- Tags on the «Namespace» are mapped to either tags on the XSDSchema, or to XSDAnnotation as outlined in the previous paragraph.
- XSDImports are produced for the NIEM Infrastructure Schemas.
- XSDImports are produced for any XSDSchema referenced by the (nested) components of the target XSDSchema.
- Any packagedElements of the UML «Namespace», plus the contents of any container representing a schema symbol space (such as «PropertyHolder») are mapped to XSDSchemaContent.

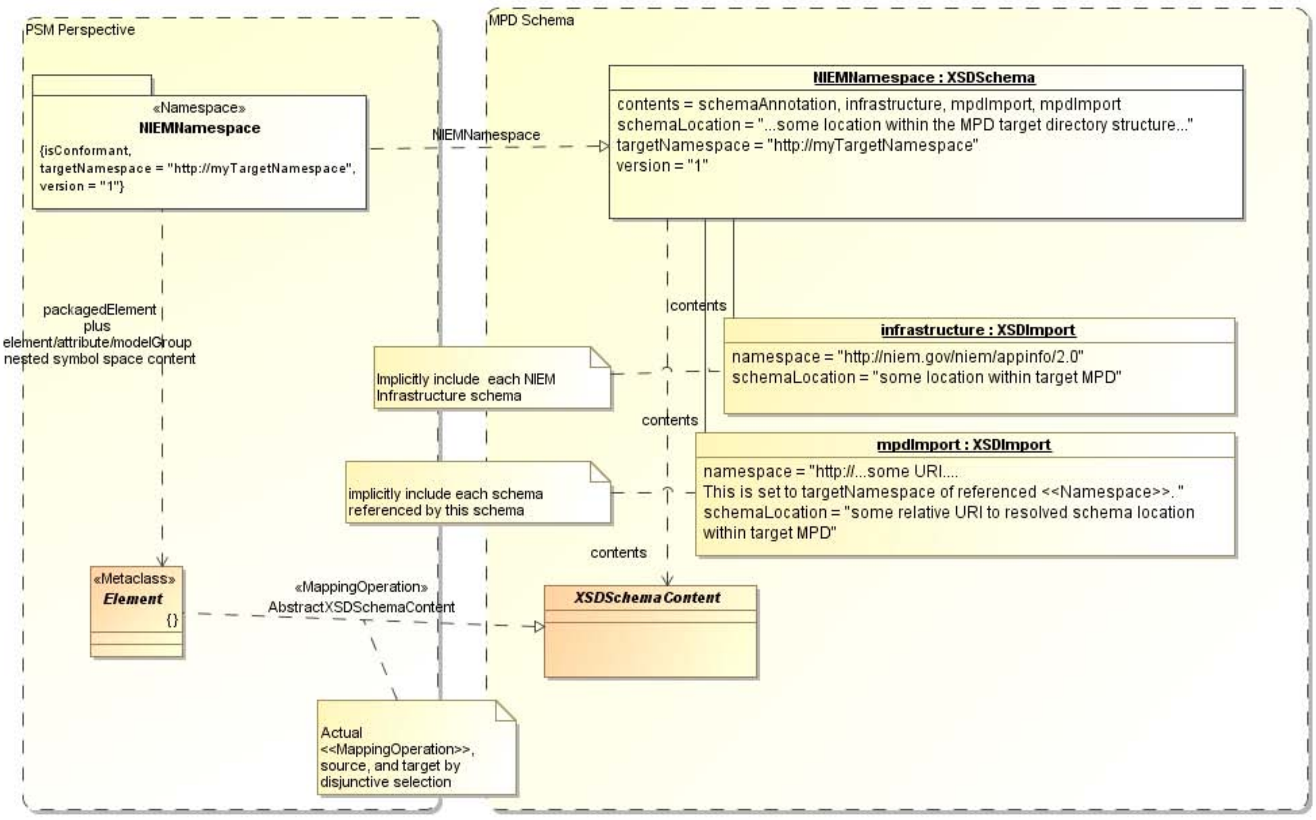

PSM Perspective
MPD Schema
«Namespace»
NIEMNamespace
{isConformant,
targetNamespace = "http://myTargetNamespace",
version = "1"}
NIEMNamespace
NIEMNamespace : XSDSchema
contents = schemaAnnotation, infrastructure, mpdImport, mpdImport
schemaLocation = "...some location within the MPD target directory structure..."
targetNamespace = "http://myTargetNamespace"
version = "1"
packagedElement
plus
element/attribute/modelGroup
nested symbol space content
contents
infrastructure : XSDImport
namespace = "http://niem.gov/niem/appinfo/2.0"
schemaLocation = "some location within target MPD"
Implicitly include each NIEM Infrastructure schema
contents
mpdImport : XSDImport
namespace = "http://...some URI....
This is set to targetNamespace of referenced <<Namespace>>."
schemaLocation = "some relative URI to resolved schema location within target MPD"
implicitly include each schema referenced by this schema
contents
«Metaclass»
Element
{}
«MappingOperation»
AbstractXSDSchemaContent
XSDSchemaContent
Actual
<<MappingOperation>>,
source, and target by
disjunctive selection

Figure 9.14 - NIEM PSM to MPD Schema Artifacts - «Namespace» Mapping Overview

«NIEMType»s are mapped to XSDComplexTypeDefinitions, as illustrated in Figure 9.15. Properties of the target XSDComplexTypeDefinition are set in conformance with NDR rules. NIEM-specific meta information is set in the XSDAnnotation, as outlined earlier.

- Inheritance in the NIEM-UML model may be specified as a Generalization or as a «Restriction »Realization. In the case of «Restriction» the derivationMethod of the target XSDComplexTypeDefinition is set to *restriction*. In all other cases, the derivationMethod is set to *extension*.
- When there is no inheritance defined for a source model «NIEMType», then the target model XSDComplexTypeDefinition will have a baseTypeDefinition set to one of the NIEM NDR-defined XSDComplexTypeDefinitions defined in the “structures” XSDSchema, based on the specific subtype of «NIEMType».
- When the baseTypeDefinition is `structures:ComplexObjectType`, then `appinfo:Base` will be either *structures:Object* or *structures:Association*, depending upon the source model NIEMType.

- When the baseTypeDefinition is not `structures:ComplexObjectType`, the `appinfo:Base` is set to the baseTypeDefinition.
- OwnedAttributes of «NIEMType» that are «XSDProperty» {kind=attribute} are mapped to XSDAttributeGroupContent as the attributeContents of the target XSDComplexTypeDefinition.
  For NIEM conformant schemas, the XSDAttributeGroupContent will be more specifically an XSDAttributeUse.
- The «NIEMType» is also mapped to XSDComplexTypeContent, the content of the target XSDComplexTypeDefinition. The abstract XSDComplexTypeContent will be either an XSDSimpleTypeDefinition or an XSDParticle, depending upon the baseTypeDefinition.

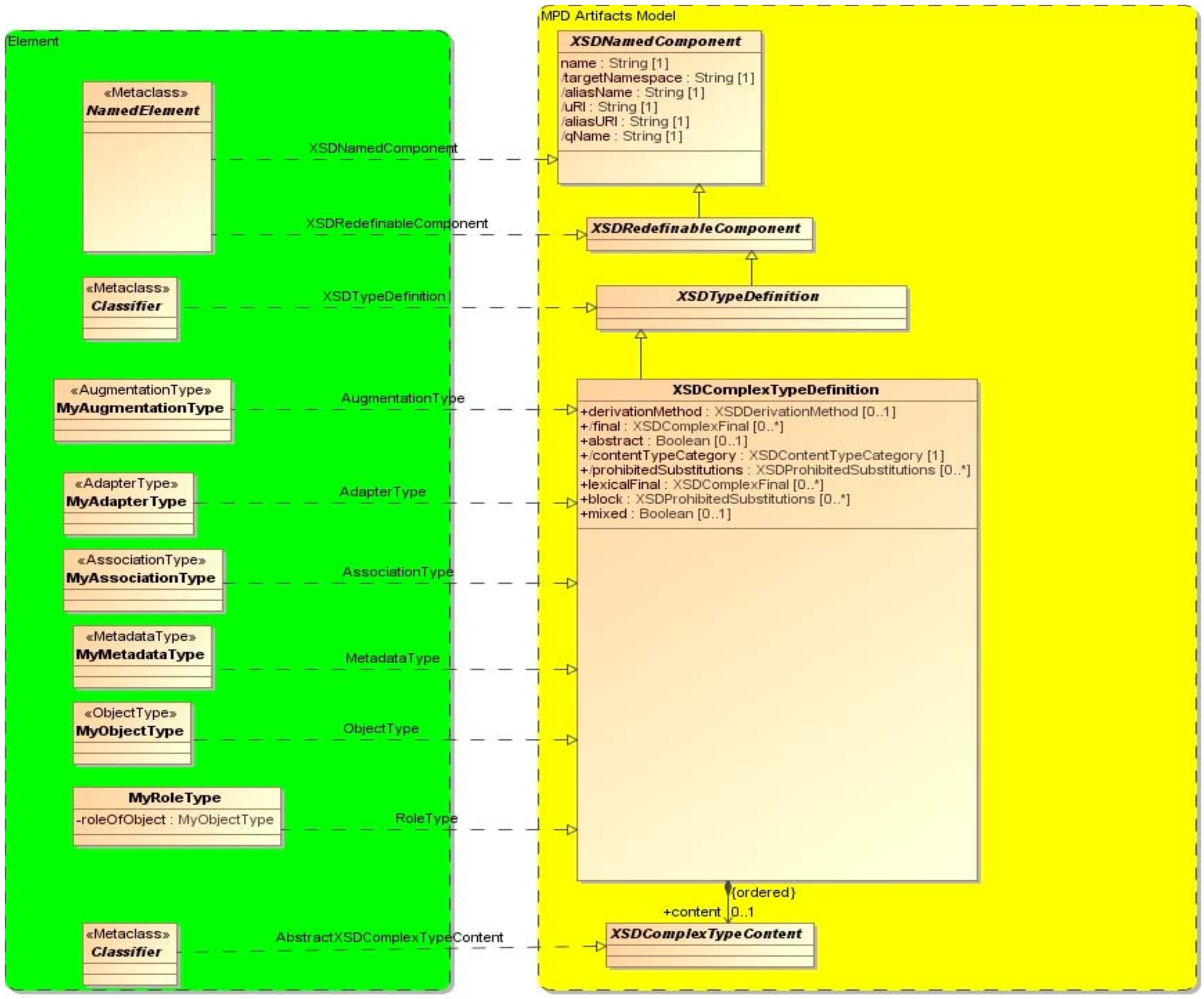

Element
MPD Artifacts Model
«Metaclass» NamedElement
«Metaclass» Classifier
«AugmentationType» MyAugmentationType
«AdapterType» MyAdapterType
«AssociationType» MyAssociationType
«MetadataType» MyMetadataType
«ObjectType» MyObjectType
MyRoleType
-roleOfObject : MyObjectType
«Metaclass» Classifier
XSDNamedComponent
XSDRedefinableComponent
XSDTypeDefinition
AugmentationType
AdapterType
AssociationType
MetadataType
ObjectType
RoleType
AbstractXSDComplexTypeContent
XSDNamedComponent
name : String [1]
/targetNamespace : String [1]
/aliasName : String [1]
/uRI : String [1]
/aliasURI : String [1]
/qName : String [1]
XSDRedefinableComponent
XSDTypeDefinition
XSDComplexTypeDefinition
+derivationMethod : XSDDerivationMethod [0..1]
+/final : XSDComplexFinal [0..*]
+abstract : Boolean [0..1]
+/contentTypeCategory : XSDContentTypeCategory [1]
+/prohibitedSubstitutions : XSDProhibitedSubstitutions [0..*]
+lexicalFinal : XSDComplexFinal [0..*]
+block : XSDProhibitedSubstitutions [0..*]
+mixed : Boolean [0..1]
{ordered}
+content
0..1
XSDComplexTypeContent

Figure 9.15 - NIEM PSM to MPD Schema Artifacts-«NIEMType» Mapping Overview

Figure 9.16 illustrates mappings between a NIEM PSM and MPD Schema Artifacts, as related to XSDFacets. Facets in the NIEM PSM are represented as tag values on a «ValueRestriction». Facets in the XSD meta-model are XSDFacets owned by an XSDSimpleTypeDefinition. The mapping provides for the construction of a specific XSDFacet for each populated tag value in the source model «ValueRestriction». An Enumeration in the NIEM PSM is mapped to an XSDSimpleTypeDefinition in the MPD Schema Artifact. Unless otherwise specified, the baseTypeDefinition for the Enumeration mapped XSDSimpleTypeDefinition is the XML Schema *token* type.

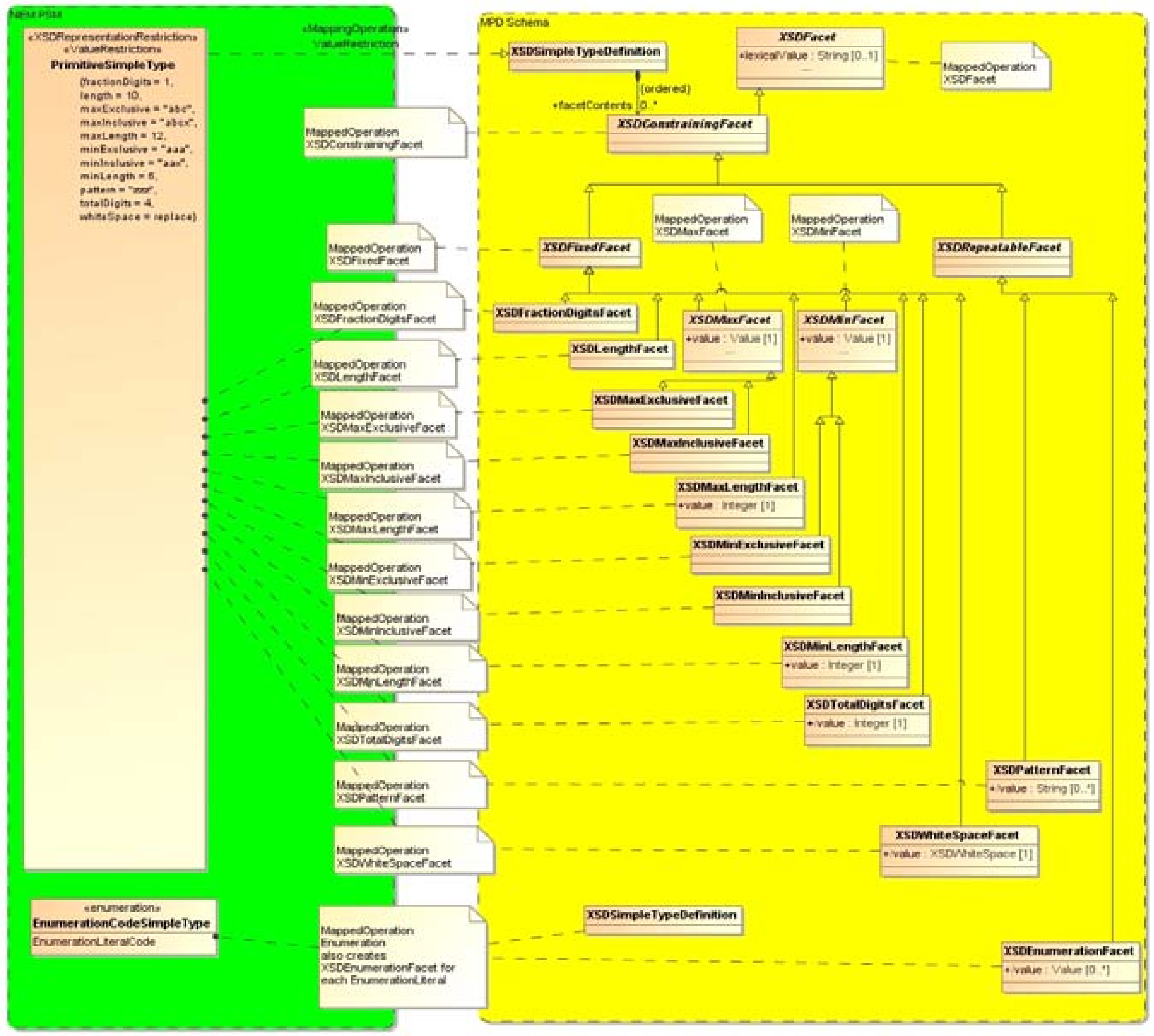

NIEM PSM
«XSDRepresentationRestriction»
«ValueRestriction»
PrimitiveSimpleType
{fractionDigits = 1, length = 10, maxExclusive = "abc", maxInclusive = "abcx", maxLength = 12, minExclusive = "aaa", minInclusive = "aax", minLength = 6, pattern = "zzz", totalDigits = 4, whiteSpace = replace}
«enumeration»
EnumerationCodeSimpleType
EnumerationLiteralCode
«MappingOperation»
ValueRestriction
MPD Schema
XSDSimpleTypeDefinition
XSDFacet
+lexicalValue : String [0..1]
MappedOperation XSDFacet
{ordered}
+facetContents 0..*
XSDConstrainingFacet
MappedOperation XSDConstrainingFacet
MappedOperation XSDMaxFacet
MappedOperation XSDMinFacet
XSDFixedFacet
MappedOperation XSDFixedFacet
XSDRepeatableFacet
XSDFractionDigitsFacet
MappedOperation XSDFractionDigitsFacet
XSDMaxFacet
+value : Value [1]
XSDMinFacet
+value : Value [1]
XSDLengthFacet
MappedOperation XSDLengthFacet
XSDMaxExclusiveFacet
MappedOperation XSDMaxExclusiveFacet
XSDMaxInclusiveFacet
MappedOperation XSDMaxInclusiveFacet
XSDMaxLengthFacet
+value : Integer [1]
MappedOperation XSDMaxLengthFacet
XSDMinExclusiveFacet
MappedOperation XSDMinExclusiveFacet
XSDMinInclusiveFacet
MappedOperation XSDMinInclusiveFacet
XSDMinLengthFacet
+value : Integer [1]
MappedOperation XSDMinLengthFacet
XSDTotalDigitsFacet
+/value : Integer [1]
MappedOperation XSDTotalDigitsFacet
XSDPatternFacet
+/value : String [0..*]
MappedOperation XSDPatternFacet
XSDWhiteSpaceFacet
+/value : XSDWhiteSpace [1]
MappedOperation XSDWhiteSpaceFacet
XSDSimpleTypeDefinition
MappedOperation Enumeration also creates XSDEnumerationFacet for each EnumerationLiteral
XSDEnumerationFacet
+/value : Value [0..*]

Figure 9.16 - NIEM PSM to MPD Schema Artifacts - Common Profile Facet Mapping Overview

Figure 9.17 illustrates mappings to non-atomic XSDSimpleTypeDefinitions, XSDComplexTypeDefinitions, and top level features:

- «List»s are represented in the target MPD Schema as XSDSimpleTypeDefinitions with the “variety” tag value computed as *list*. A «List» has a single property. The type of that property is mapped to the itemTypeDefinition property of XSDSimpleTypeDefinition, making it a *list*.

- «Union»s are represented in the MPD Schema as XSDSimpleTypeDefinitions with the "variety" tag value computed as *union*. The suppliers of any «UnionOf» Usage cliented by the «Union» are mapped to the memberTypeDefinition property of XSDSimpleTypeDefinition, making it a *union*.
- A «NIEMType» is mapped to an XSDComplexTypeDefinition. The «NIEMType» is also mapped to XSDComplexTypeContent, the content of the target XSDComplexTypeDefinition. The abstract XSDComplexTypeContent will be either an XSDSimpleTypeDefinition or an XSDParticle, depending upon the baseTypeDefinition.
  - When the XSDComplexTypeContent is an XSDParticle, then the «NIEMType» is also mapped to the content of the XSDParticle, which is an XSDParticleContent. The XSDParticleContent is typically an XSDModelGroup{compositor=*sequence*}.
  - The ownedAttributes of the UML «NIEMType» that are «XSDProperty»{kind=element} are mapped to contents of XSDModelGroup as XSDParticles. The upper/lower multiplicity bounds of the «XSDProperty» are mapped to the maxOccurs/minOccurs of the XSDParticle.
  - The «XSDProperty» is also mapped to the content of the XSDParticle as an XSDElementDeclaration. The XSDElementDeclaration will have no name and no typeDefinition.
  - For NIEM-compliant features, there will always be a «References» Realization from the «XSDProperty» owned by a «NIEMType» to a resolved top-level Element owned by a «PropertyHolder» (in the same or a different Schema). The «References» Realization is mapped to the resolvedElementDeclaration property of the target XSDElementDeclaration.
- An «XSDProperty» contained by a «PropertyHolder» is mapped to an XSDElementDeclaration directly contained by an XSDSchema.
- An «XSDProperty» that has a subsetProperty reference to another «XSDProperty» is mapped to a *substitutionGroup* reference between elements.
- «PropertyHolder»s contained by a «Namespace» represent schema symbol spaces. Within a «PropertyHolder», an «XSDProperty»{kind=element} represents a member of the schema element symbol space. Correspondingly, an «XSDProperty»{kind=attribute} represents a member of the schema attribute symbol space. There is no direct physical manifestation of symbol spaces within an XSDSchema, they are implicit based on whether the top level components are XSDAttribute or XSDElement (hence the «PropertyHolder» itself is not mapped to an XSDComponent). Additionally, any Generalization relationship between «PropertyHolder»s (which may be required to satisfy *subsetsProperty* reference semantics) are not mapped.

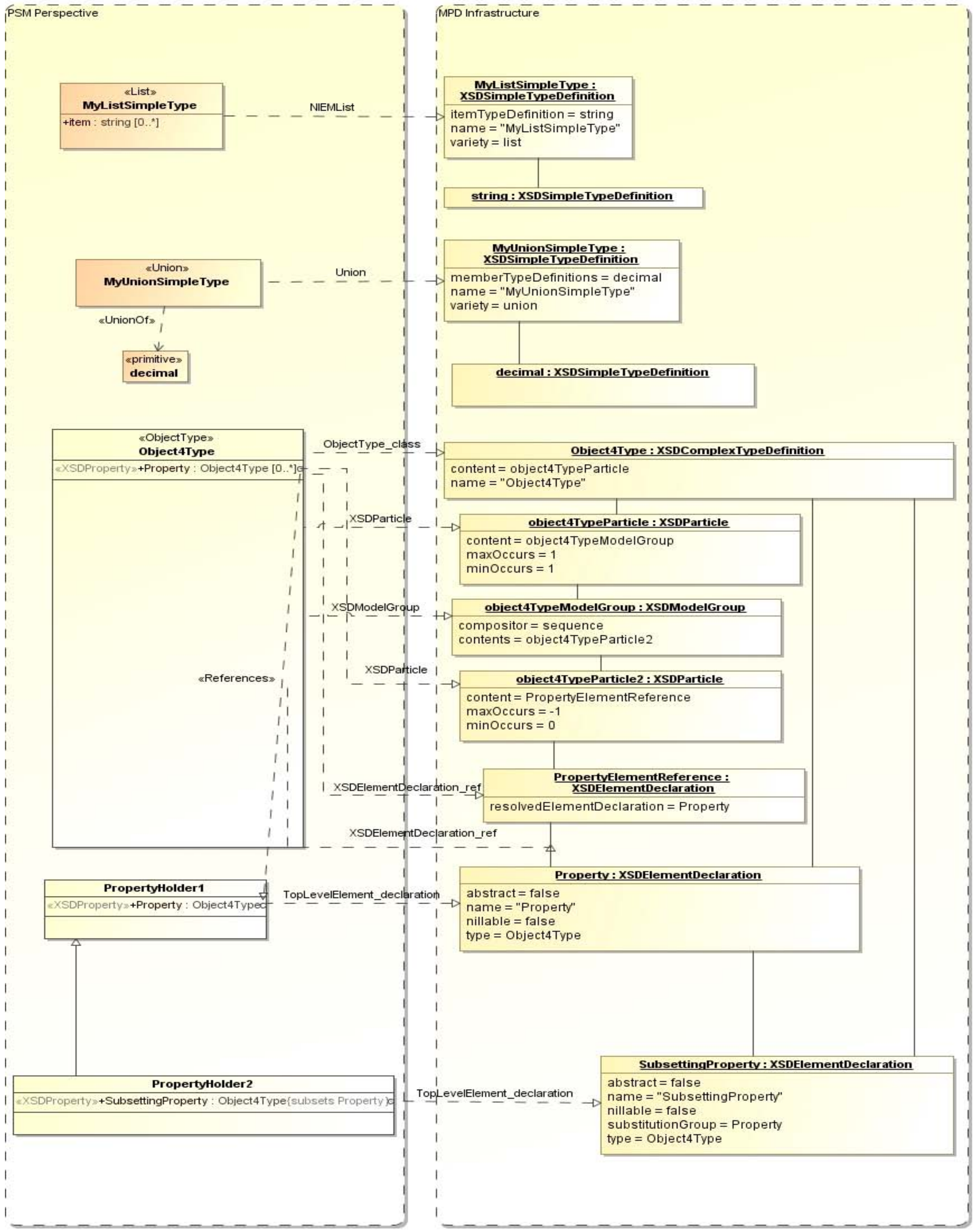

PSM Perspective
MPD Infrastructure
«List» MyListSimpleType
+item : string [0..*]
NIEMList
MyListSimpleType : XSDSimpleTypeDefinition
itemTypeDefinition = string
name = "MyListSimpleType"
variety = list
string : XSDSimpleTypeDefinition
«Union» MyUnionSimpleType
Union
MyUnionSimpleType : XSDSimpleTypeDefinition
memberTypeDefinitions = decimal
name = "MyUnionSimpleType"
variety = union
«UnionOf»
«primitive» decimal
decimal : XSDSimpleTypeDefinition
«ObjectType» Object4Type
«XSDProperty»+Property : Object4Type [0..*]
ObjectType_class
Object4Type : XSDComplexTypeDefinition
content = object4TypeParticle
name = "Object4Type"
XSDParticle
object4TypeParticle : XSDParticle
content = object4TypeModelGroup
maxOccurs = 1
minOccurs = 1
XSDModelGroup
object4TypeModelGroup : XSDModelGroup
compositor = sequence
contents = object4TypeParticle2
XSDParticle
object4TypeParticle2 : XSDParticle
content = PropertyElementReference
maxOccurs = -1
minOccurs = 0
«References»
XSDElementDeclaration_ref
PropertyElementReference : XSDElementDeclaration
resolvedElementDeclaration = Property
XSDElementDeclaration_ref
PropertyHolder1
«XSDProperty»+Property : Object4Type
TopLevelElement_declaration
Property : XSDElementDeclaration
abstract = false
name = "Property"
nillable = false
type = Object4Type
PropertyHolder2
«XSDProperty»+SubsettingProperty : Object4Type{subsets Property}
TopLevelElement_declaration
SubsettingProperty : XSDElementDeclaration
abstract = false
name = "SubsettingProperty"
nillable = false
substitutionGroup = Property
type = Object4Type

Figure 9.17 - NIEM PSM to MPD Schema Artifacts - Common Profile Type Overview

Figure 9.18 illustrates mappings related to «Choice». The mapping is similar to a «NIEMType» to XSDComplexTypeDefinition, the variation being that the XSDParticleContent mapped from a Property is an XSDModelGroup{compositor=choice} instead of an XSDElementDeclaration:

- A «NIEMType» typically maps to an XSDComplexTypeDefinition, an XSDParticle, and an XSDModelGroup.
- A «NIEMType» may have ownedAttributes that are typed by a «Choice». As with «XSDProperty», these are mapped to an XSDParticle contained by the XSDModelGroup just mentioned.
- For a Property typed by a «Choice», the content of the XSDParticle is an XSDModelGroup (instead of an XSDElementDeclaration). The compositor of the XSDModelGroup is *choice*.
- The contents of the XSDModelGroup are XSDParticles mapped from the «XSDProperty» owned by the «Choice». The content of each of these XSDParticles is an XSDElementDeclaration (also mapped from the «XSDProperty»), as in the case for «XSDProperty»s owned directly by a «NIEMType».

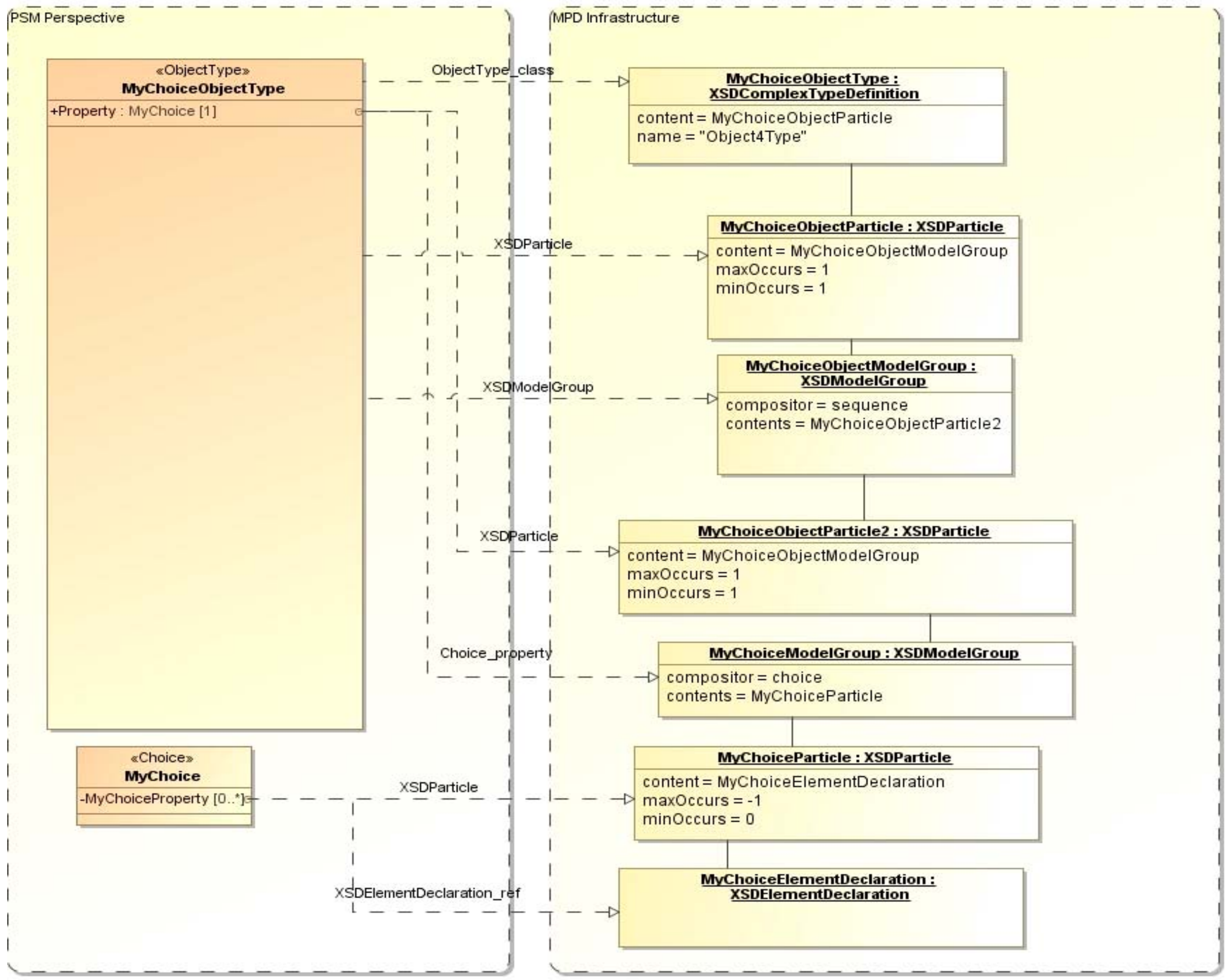

PSM Perspective
MPD Infrastructure
«ObjectType»
MyChoiceObjectType
+Property : MyChoice [1]
«Choice»
MyChoice
-MyChoiceProperty [0..*]
ObjectType_class
XSDParticle
XSDModelGroup
XSDParticle
Choice_property
XSDParticle
XSDElementDeclaration_ref
MyChoiceObjectType : XSDComplexTypeDefinition
content = MyChoiceObjectParticle
name = "Object4Type"
MyChoiceObjectParticle : XSDParticle
content = MyChoiceObjectModelGroup
maxOccurs = 1
minOccurs = 1
MyChoiceObjectModelGroup : XSDModelGroup
compositor = sequence
contents = MyChoiceObjectParticle2
MyChoiceObjectParticle2 : XSDParticle
content = MyChoiceObjectModelGroup
maxOccurs = 1
minOccurs = 1
MyChoiceModelGroup : XSDModelGroup
compositor = choice
contents = MyChoiceParticle
MyChoiceParticle : XSDParticle
content = MyChoiceElementDeclaration
maxOccurs = -1
minOccurs = 0
MyChoiceElementDeclaration : XSDElementDeclaration

Figure 9.18 - NIEM PSM to MPD Schema Artifacts - «Choice» Mapping Overview

Figure 9.19 illustrates some mappings related to baseTypeDefinitions:

- «ValueRestriction» inheritance from NIEM XML Primitive Types is mapped to an XSDSimpleTypeDefinition baseTypeDefinition referencing the datatype counterpart from the XML Schema for Schemas.
- «ValueRestriction» specialization from a general «ValueRestriction» is mapped to an XSDSimpleTypeDefinition baseTypeDefinition referencing the mapped general «ValueRestriction».
- A «NIEMType» is mapped to an XSDComplexTypeDefinition.
  - When the «NIEMType» has an «XSDSimpleContent» Realization to a «ValueRestriction» then the content of the XSDComplexTypeDefinition is an XSDSimpleTypeDefinition whose baseTypeDefinition is the XSDTypeDefinition mapped from the supplier of the «XSDSimpleContent».
    - When the supplier is a type from the XML Primitive Types library, and the type is also present in the NIEM Infrastructure proxy schema, then the proxy type becomes the baseTypeDefinition.
    - When the supplier is a type from the XML Primitive Types library, and the type is not present in the NIEM Infrastructure proxy schema, then the XML Schema for Schemas XSDSimpleTypeDefinition is the baseTypeDefinition.
    - In all other cases, the baseTypeDefinition is the XSDSimpleTypeDefinition mapped from the «XSDSimpleContent» supplier.
    - When the baseTypeDefinition is not a proxy, then an XSDAttributeGroupDefinition is added to the attributeContents of the XSDComplexTypeDefinition. The resolvedAttributeGroupDefinition of the XSDAttributeGroupDefinition is set to the NIEM Infrastructure `structures:SimpleObjectAttributeGroup`.
- A «NIEMType» that has a «Restriction» Realization to a supplier «NIEMType» is mapped to an XSDComplexTypeDefinition with a derivationMethod of *restriction*. The baseTypeDefinition is set to the XSDComplexTypeDefinition mapped from the supplier of the «Restriction».
- «NIEMType»s that have no Generalizations or «Restriction»s are coerced to inherit from an appropriate Type in the “structures” schema, depending upon the subtype of «NIEMType».

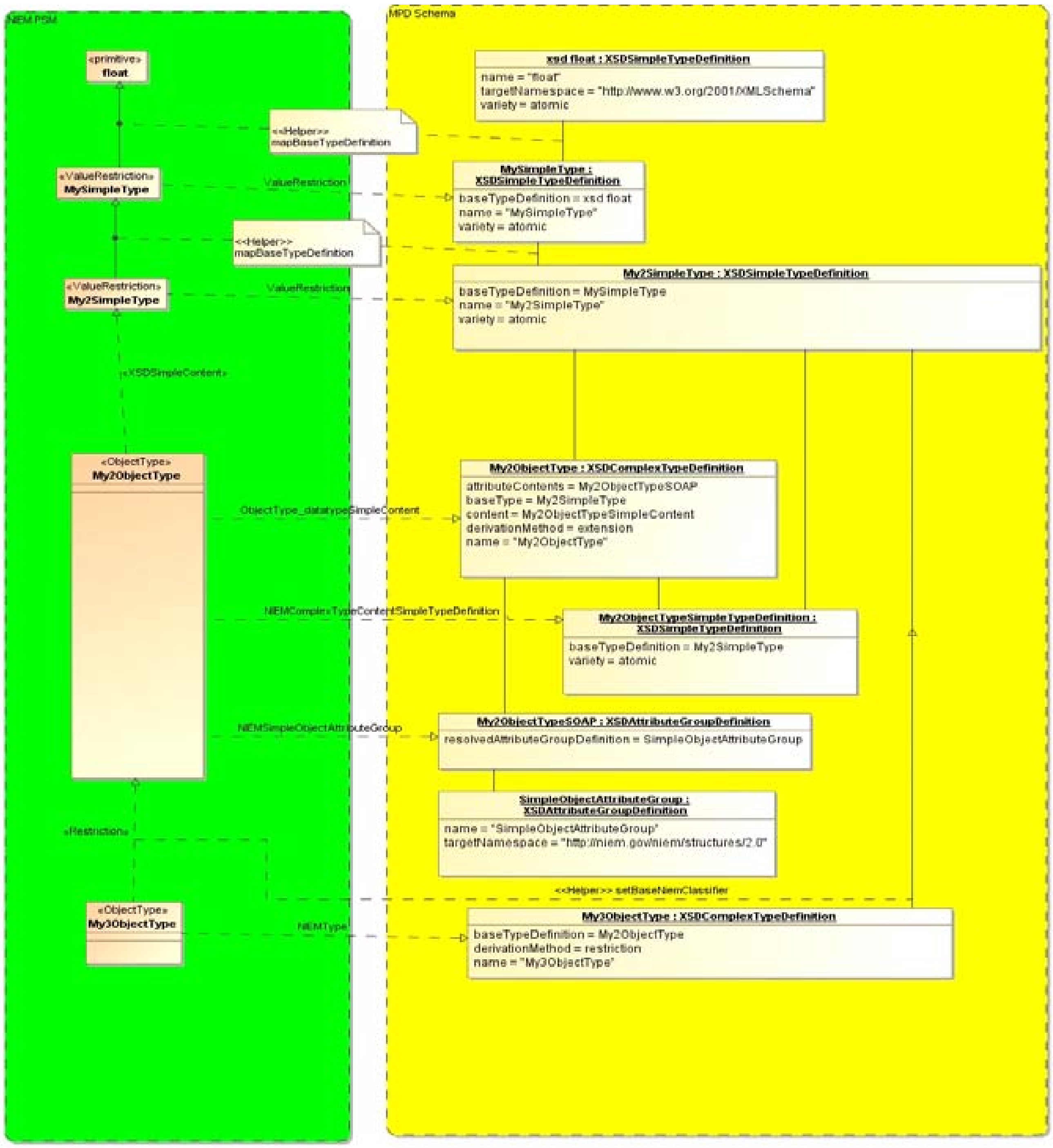

NIEM PSM
MPD Schema
«primitive» float
«ValueRestriction» MySimpleType
«ValueRestriction» My2SimpleType
«XSDSimpleContent»
«ObjectType» My2ObjectType
«Restriction»
«ObjectType» My3ObjectType
<<Helper>> mapBaseTypeDefinition
ValueRestriction
ObjectType_datatypeSimpleContent
NIEMComplexTypeContentSimpleTypeDefinition
NIEMSimpleObjectAttributeGroup
NIEMType
<<Helper>> setBaseNiemClassifier
xsd float : XSDSimpleTypeDefinition
name = "float"
targetNamespace = "http://www.w3.org/2001/XMLSchema"
variety = atomic
MySimpleType : XSDSimpleTypeDefinition
baseTypeDefinition = xsd float
name = "MySimpleType"
variety = atomic
My2SimpleType : XSDSimpleTypeDefinition
baseTypeDefinition = MySimpleType
name = "My2SimpleType"
variety = atomic
My2ObjectType : XSDComplexTypeDefinition
attributeContents = My2ObjectTypeSOAP
baseType = My2SimpleType
content = My2ObjectTypeSimpleContent
derivationMethod = extension
name = "My2ObjectType"
My2ObjectTypeSimpleTypeDefinition : XSDSimpleTypeDefinition
baseTypeDefinition = My2SimpleType
variety = atomic
My2ObjectTypeSOAP : XSDAttributeGroupDefinition
resolvedAttributeGroupDefinition = SimpleObjectAttributeGroup
SimpleObjectAttributeGroup : XSDAttributeGroupDefinition
name = "SimpleObjectAttributeGroup"
targetNamespace = "http://niem.gov/niem/structures/2.0"
My3ObjectType : XSDComplexTypeDefinition
baseTypeDefinition = My2ObjectType
derivationMethod = restriction
name = "My3ObjectType"

Figure 9.19 - NIEM PSM to MPD Schema Artifact- baseTypeDefinition Overview

For NIEMpsm2mpd, mapping operations are generally invoked with the context of a source NIEM-UML Element and produce some form of target XSDComponent. Most mapping operations are also provided an argument that is the XSDComponent container context. The “when” condition for the MappingOperations are normally a function of the source NIEM-UML element, the type of the source NIEM-UML element, the source NIEM-UML element's applied stereotype, and/or the target XSDComponent container context. The mapping operation connects its target XSDComponent to its container and populates the XSDComponent properties.

Figure 9.20 illustrates the disjunction pattern for the NIEMpsm2mpd transformation.

For the NIEMpsm2mpd transformation, each level of the inheritance hierarchy populates properties of a target XSDComponent from the stereotype tag values and/or UML elements of the source NIEM-UML Model. Figure 9.21 illustrates the inheritance for most of the MappingOperations in the NIEMpsm2mpd transformation.

Figure 9.22 illustrates the remaining MappingOperations for the NIEMpsm2mpd transformation.

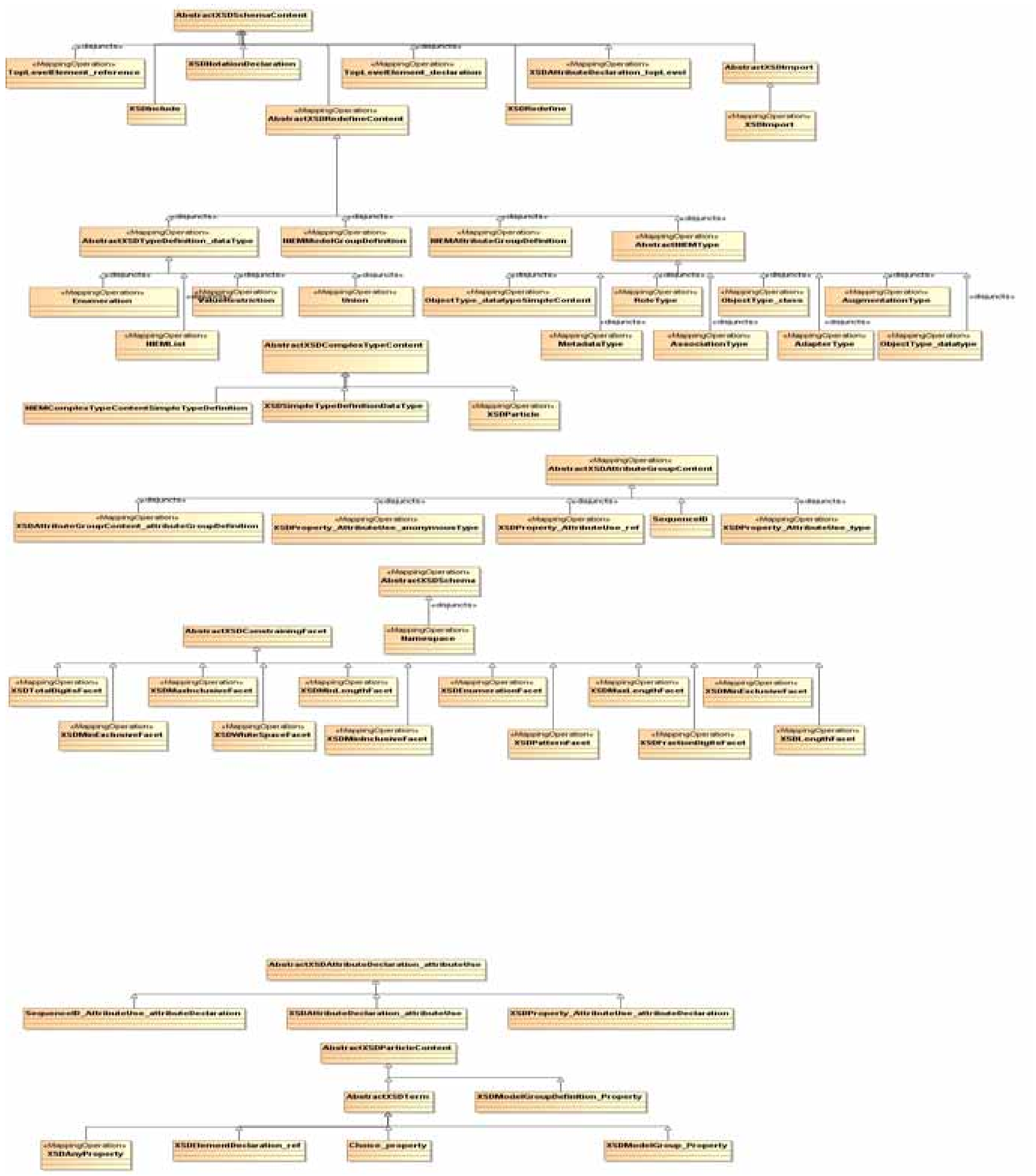

AbstractXSDSchemaContent
«disjuncts»
«MappingOperation» TopLevelElement_reference
XSDInclude
XSDNotationDeclaration
«MappingOperation» AbstractXSDRedefineContent
«MappingOperation» TopLevelElement_declaration
XSDRedefine
«MappingOperation» XSDAttributeDeclaration_topLevel
AbstractXSDImport
«MappingOperation» XSDImport
«MappingOperation» AbstractXSDTypeDefinition_dataType
«MappingOperation» NIEMModelGroupDefinition
«MappingOperation» NIEMAttributeGroupDefinition
«MappingOperation» AbstractNIEMType
«MappingOperation» Enumeration
«MappingOperation» ValueRestriction
«MappingOperation» Union
«MappingOperation» NIEMList
AbstractXSDComplexTypeContent
«MappingOperation» ObjectType_datatypeSimpleContent
«MappingOperation» RoleType
«MappingOperation» ObjectType_class
«MappingOperation» AugmentationType
«MappingOperation» MetadataType
«MappingOperation» AssociationType
«MappingOperation» AdapterType
«MappingOperation» ObjectType_datatype
NIEMComplexTypeContentSimpleTypeDefinition
XSDSimpleTypeDefinitionDataType
«MappingOperation» XSDParticle
«MappingOperation» AbstractXSDAttributeGroupContent
«MappingOperation» XSDAttributeGroupContent_attributeGroupDefinition
«MappingOperation» XSDProperty_AttributeUse_anonymousType
«MappingOperation» XSDProperty_AttributeUse_ref
SequenceID
«MappingOperation» XSDProperty_AttributeUse_type
«MappingOperation» AbstractXSDSchema
«MappingOperation» Namespace
AbstractXSDConstrainingFacet
«MappingOperation» XSDTotalDigitsFacet
«MappingOperation» XSDMaxInclusiveFacet
«MappingOperation» XSDMinLengthFacet
«MappingOperation» XSDEnumerationFacet
«MappingOperation» XSDMaxLengthFacet
«MappingOperation» XSDMinExclusiveFacet
«MappingOperation» XSDMaxExclusiveFacet
«MappingOperation» XSDWhiteSpaceFacet
«MappingOperation» XSDMinInclusiveFacet
«MappingOperation» XSDPatternFacet
«MappingOperation» XSDFractionDigitsFacet
«MappingOperation» XSDLengthFacet
AbstractXSDAttributeDeclaration_attributeUse
SequenceID_AttributeUse_attributeDeclaration
XSDAttributeDeclaration_attributeUse
XSDProperty_AttributeUse_attributeDeclaration
AbstractXSDParticleContent
AbstractXSDTerm
XSDModelGroupDefinition_Property
«MappingOperation» XSDAnyProperty
XSDElementDeclaration_ref
Choice_property
XSDModelGroup_Property

Figure 9.20 - NIEM PSM to NIEM-Conforming XML Schema - Disjunction

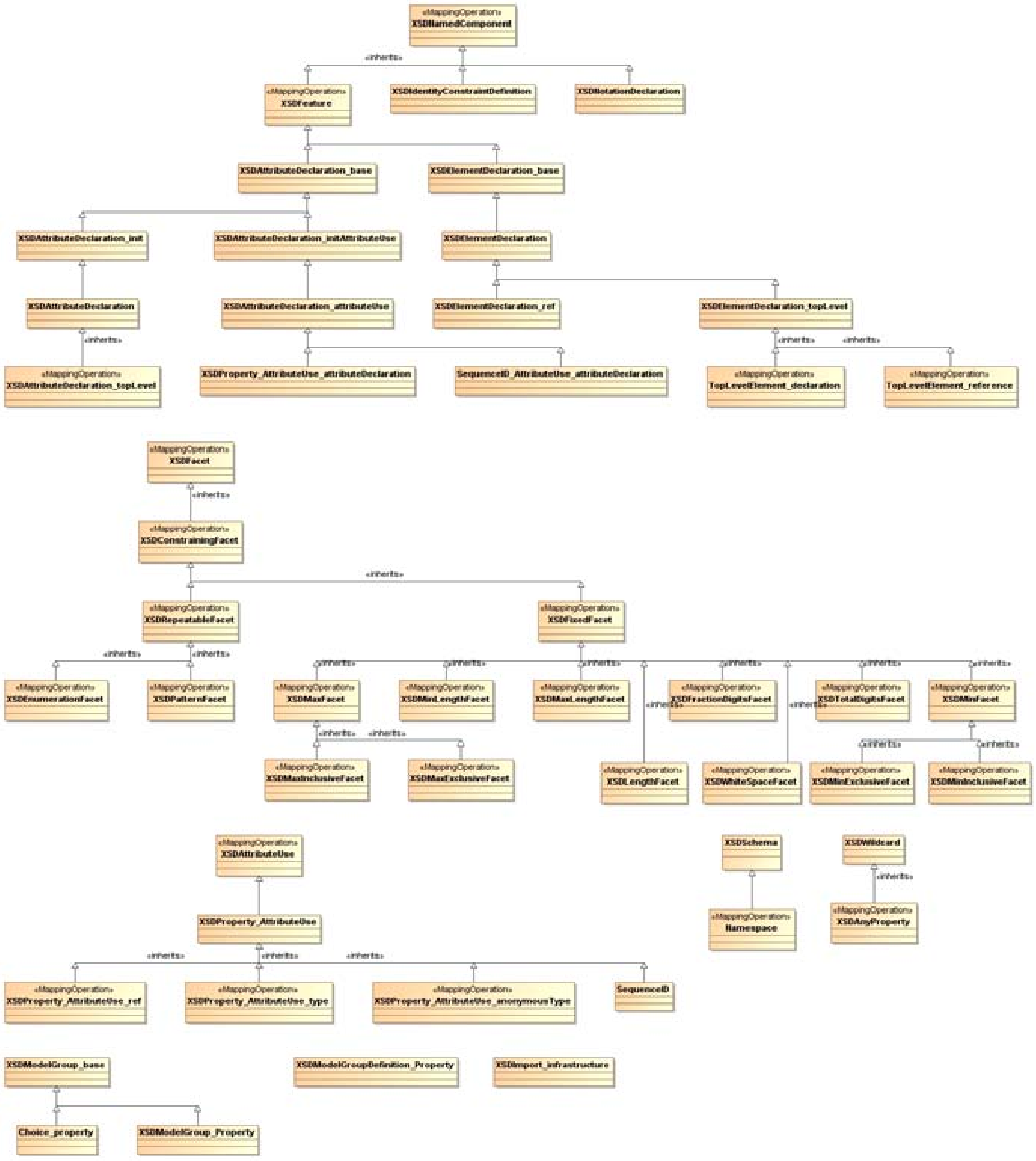
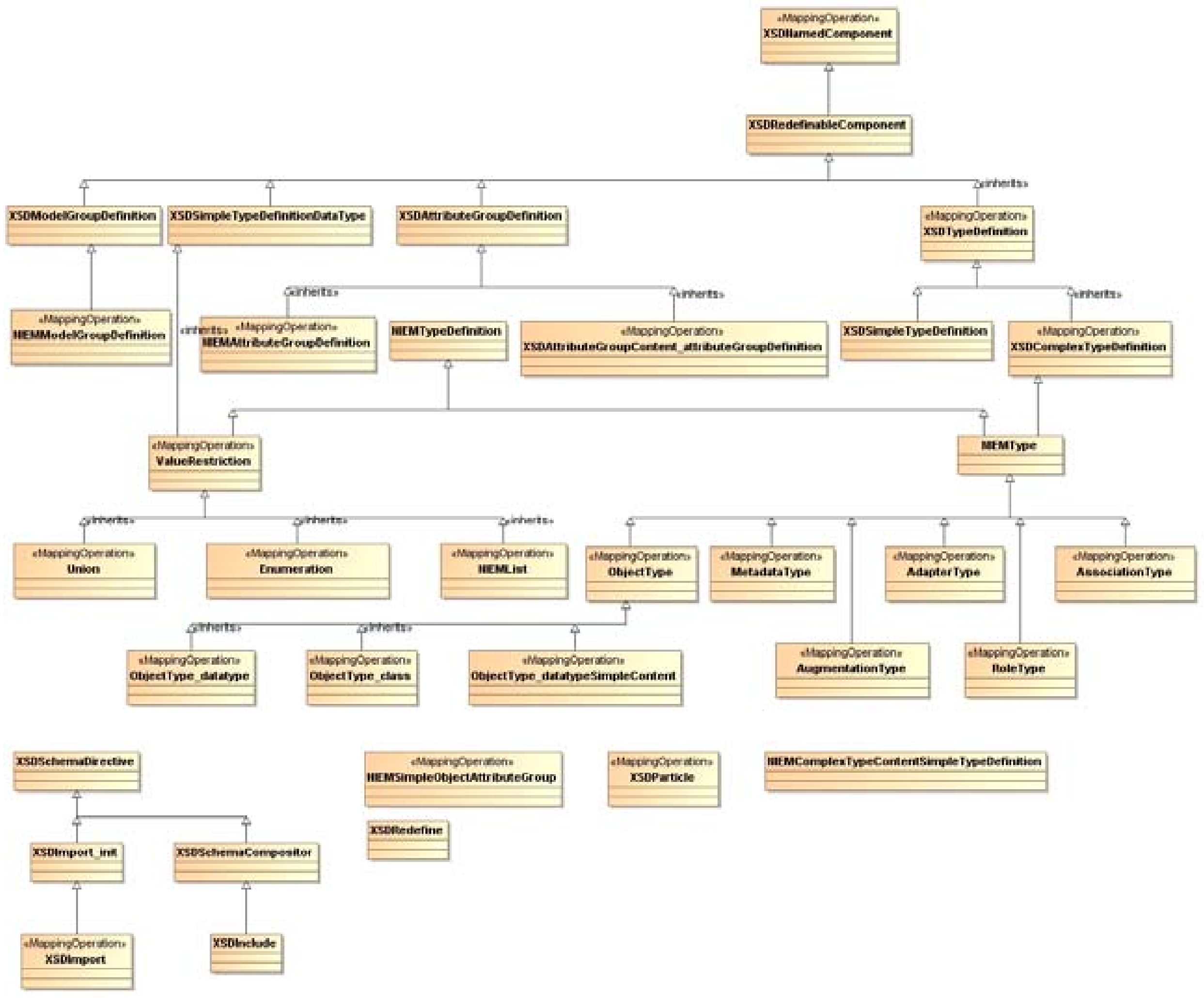

«MappingOperation» XSDNamedComponent
«inherits»
«MappingOperation» XSDFeature
XSDIdentityConstraintDefinition
XSDNotationDeclaration
XSDAttributeDeclaration_base
XSDElementDeclaration_base
XSDAttributeDeclaration_init
XSDAttributeDeclaration_initAttributeUse
XSDElementDeclaration
XSDAttributeDeclaration
XSDAttributeDeclaration_attributeUse
XSDElementDeclaration_ref
XSDElementDeclaration_topLevel
«MappingOperation» XSDAttributeDeclaration_topLevel
XSDProperty_AttributeUse_attributeDeclaration
SequenceID_AttributeUse_attributeDeclaration
«MappingOperation» TopLevelElement_declaration
«MappingOperation» TopLevelElement_reference
«MappingOperation» XSDFacet
«MappingOperation» XSDConstrainingFacet
«MappingOperation» XSDRepeatableFacet
«MappingOperation» XSDFixedFacet
«MappingOperation» XSDEnumerationFacet
«MappingOperation» XSDPatternFacet
«MappingOperation» XSDMaxFacet
«MappingOperation» XSDMinLengthFacet
«MappingOperation» XSDMaxLengthFacet
«MappingOperation» XSDFractionDigitsFacet
«MappingOperation» XSDTotalDigitsFacet
«MappingOperation» XSDMinFacet
«MappingOperation» XSDMaxInclusiveFacet
«MappingOperation» XSDMaxExclusiveFacet
«MappingOperation» XSDLengthFacet
«MappingOperation» XSDWhiteSpaceFacet
«MappingOperation» XSDMinExclusiveFacet
«MappingOperation» XSDMinInclusiveFacet
«MappingOperation» XSDAttributeUse
XSDProperty_AttributeUse
«MappingOperation» XSDProperty_AttributeUse_ref
«MappingOperation» XSDProperty_AttributeUse_type
«MappingOperation» XSDProperty_AttributeUse_anonymousType
SequenceID
XSDSchema
«MappingOperation» Namespace
XSDWildcard
«MappingOperation» XSDAnyProperty
XSDModelGroup_base
Choice_property
XSDModelGroup_Property
XSDModelGroupDefinition_Property
XSDImport_infrastructure

Figure 9.21 - NIEM PSM to NIEM-Conforming XML Schema - Inheritance



«MappingOperation» XSDNamedComponent
XSDRedefinableComponent
«inherits»
XSDModelGroupDefinition
XSDSimpleTypeDefinitionDataType
XSDAttributeGroupDefinition
«MappingOperation» XSDTypeDefinition
«MappingOperation» NIEMModelGroupDefinition
«MappingOperation» NIEMAttributeGroupDefinition
NIEMTypeDefinition
«MappingOperation» XSDAttributeGroupContent_attributeGroupDefinition
XSDSimpleTypeDefinition
«MappingOperation» XSDComplexTypeDefinition
«MappingOperation» ValueRestriction
NIEMType
«MappingOperation» Union
«MappingOperation» Enumeration
«MappingOperation» NIEMList
«MappingOperation» ObjectType
«MappingOperation» MetadataType
«MappingOperation» AdapterType
«MappingOperation» AssociationType
«MappingOperation» ObjectType_datatype
«MappingOperation» ObjectType_class
«MappingOperation» ObjectType_datatypeSimpleContent
«MappingOperation» AugmentationType
«MappingOperation» RoleType
XSDSchemaDirective
«MappingOperation» NIEMSimpleObjectAttributeGroup
«MappingOperation» XSDParticle
NIEMComplexTypeContentSimpleTypeDefinition
XSDRedefine
XSDImport_init
XSDSchemaCompositor
«MappingOperation» XSDImport
XSDInclude

Figure 9.22 - NIEM PSM to NIEM-Conforming XML Schema - Inheritance Other

9.4 NIEM MPD Model to NIEM MPD Artifact

Figure 9.23 illustrates the high-level packaging map between NIEM MPD Model and MPD Artifacts.

- A NIEM MPD «ModelPackageDescription» component is mapped to an MPD Catalog, and will contain all NIEM PSM packaging structure nested to any level. The Catalog includes `File`, `Folder`, and `FileSet` entries related to all component schemas, plus (at least) placeholder entries for MPD required and/or recommended artifacts.

  «Namespace» is mapped to XSDSchema (via NIEMpsm2xsd transformation) within an MPD directory structure determined by the relativePathName of «ModelPackageDescriptionFile» Usages. XSDImports for each XSDSchema is determined from the transitive closure of all cross-schema references embodied in the source «Namespace», plus all the NIEM NDR-required imports. Any «ModelPackageDescriptionRelationship» Usage from the source model «ModelPackageDescription» maps to a Catalog RelationshipType contained by the MetadataType entry within the CatalogType.

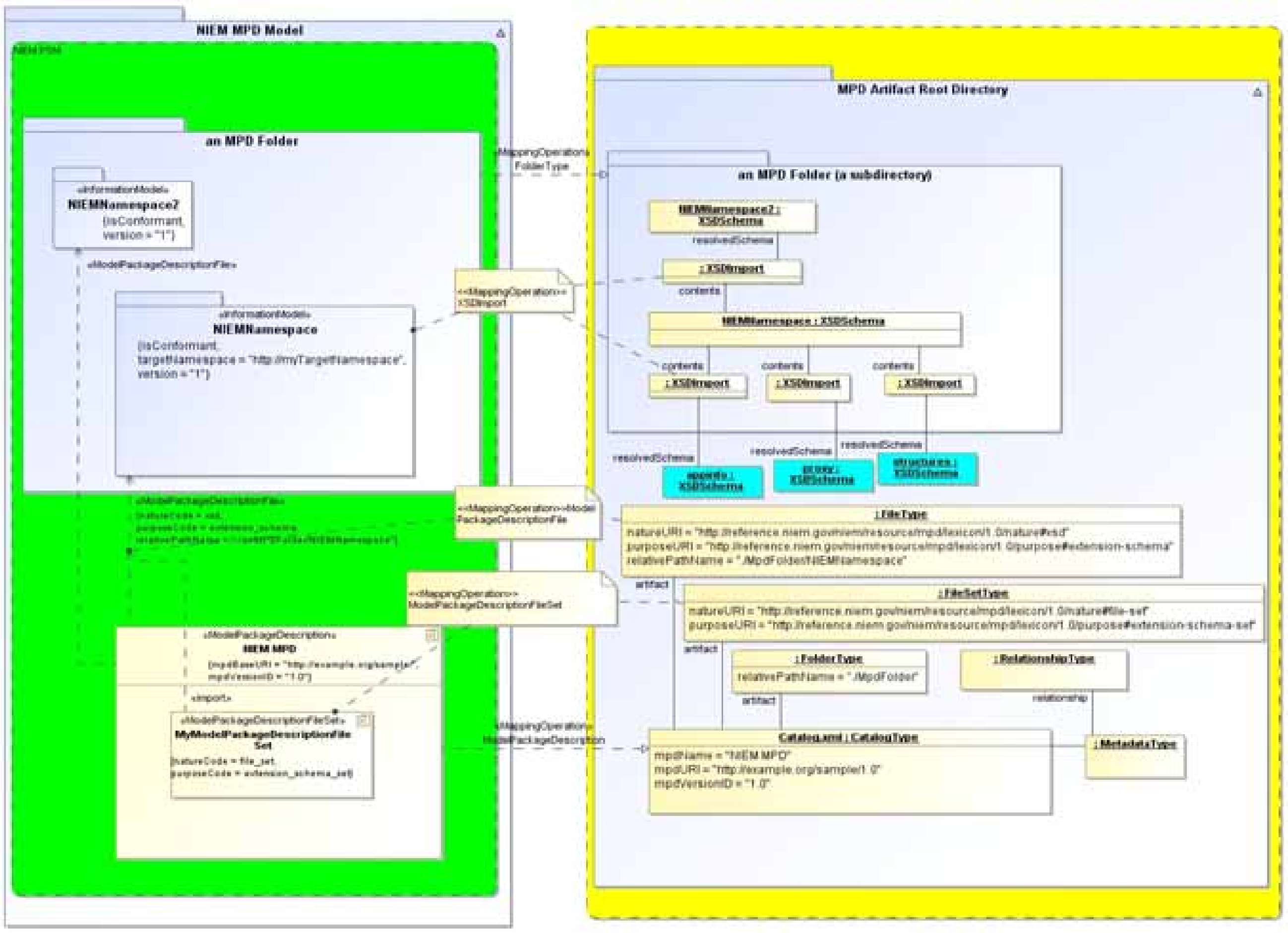

NIEM MPD Model
MPD Artifact Root Directory
an MPD Folder
an MPD Folder (a subdirectory)
NIEMNamespace2
NIEMNamespace
NIEMNamespace : XSDSchema
: XSDImport
appinfo : XSDSchema
proxy : XSDSchema
structures : XSDSchema
: FileType
: FileSetType
: FolderType
: RelationshipType
Catalog.xml : CatalogType
: MetadataType
«ModelPackageDescription» NIEM MPD
«ModelPackageDescriptionFileSet» MyModelPackageDescriptionFile Set
«MappingOperation» XSDImport
«MappingOperation» FolderType
«MappingOperation» ModelPackageDescriptionFile
«MappingOperation» ModelPackageDescriptionFileSet
«MappingOperation» ModelPackageDescription

Figure 9.23 - NIEM MPD Model to NIEM MPD Artifact Mapping Overview

9.5 NIEM MPD Artifact to NIEM MPD Model

The mapping of MPD Catalog and Schema Artifacts to the NIEM MPD Model removes much of the explicit representation of XSD constructs, NIEM relations, and binding to the NIEM NDR infrastructure.

Figure 9.24 illustrates the high-level packaging map between NIEM MPD Artifacts and the NIEM MPD Model.

- A «ModelPackageDescription» component is mapped from an MPD Catalog, and will contain MPD packaging structure (as specified by Catalog Files, Folders, and FileSets), nested to any level. The Catalog includes file, folder, and fileSet entries related to all component XSDSchemas, plus entries for other MPD-required and/or recommended artifacts.
- «InformationModel» is mapped from an XSDSchema. XSDImports for that XSDSchema are used to ensure transitive closure of all Schemas required, even if they have not been properly registered in the MPD Catalog. Those XSDSchemas constituting the NIEM Infrastructure components are not mapped. The actual XSDImport is also not mapped, since it can be derived based on cross «InformationModel» relations.

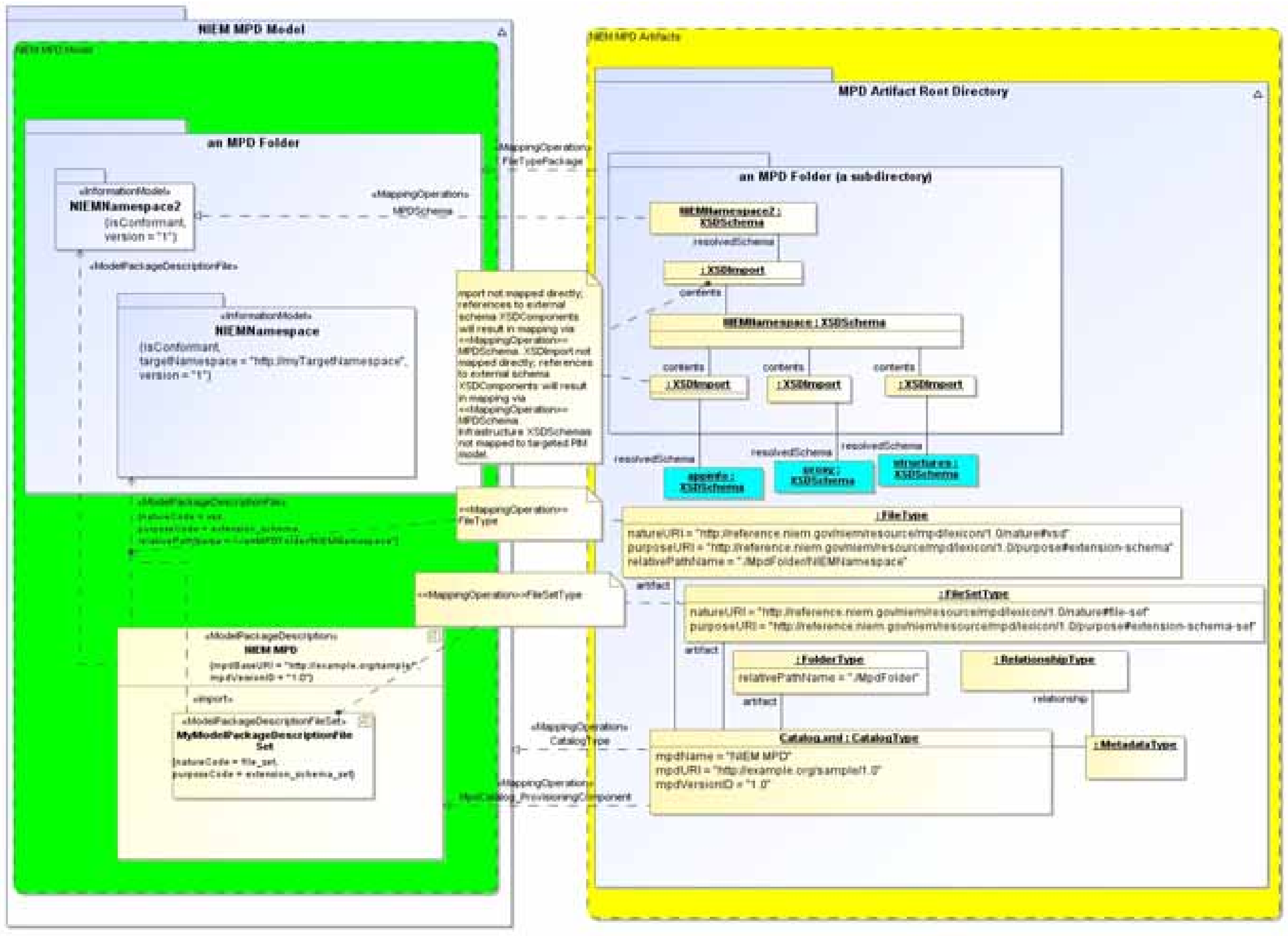

NIEM MPD Model
an MPD Folder
«InformationModel» NIEMNamespace2
«ModelPackageDescriptionFile»
«InformationModel» NIEMNamespace
«ModelPackageDescription» NIEM MPD
«ModelPackageDescriptionFileSet» MyModelPackageDescriptionFile Set
NIEM MPD Artifacts
MPD Artifact Root Directory
an MPD Folder (a subdirectory)
NIEMNamespace2 : XSDSchema
: XSDImport
NIEMNamespace : XSDSchema
appinfo : XSDSchema
proxy : XSDSchema
structures : XSDSchema
: FileType
: FileSetType
: FolderType
: RelationshipType
Catalog.xml : CatalogType
: MetadataType
«MappingOperation» FileTypePackage
«MappingOperation» MPDSchema
«MappingOperation» FileType
«MappingOperation» FileSetType
«MappingOperation» CatalogType
«MappingOperation» MpdCatalog_ProvisioningComponent

Figure 9.24 - NIEM MPD Artifact to NIEM MPD Model - Overview

There are various forms of metadata embodied in the components of a NIEM conformant Schema. The metadata are represented in schemas as text-based user/application information embodied within an XSDAnnotation. The metadata includes documentation, cross-component references, and extended properties for the XSDComponents. All forms of XSDComponent metadata are based on NIEM-NDR rules for representation within constructs of an XSDAnnotation. Figure 9.25 illustrates many specific cases of metadata usage:

- XSDAnnotations are owned by an XSDComponent. The ownership association name and semantic varies by specific XSDComponent.
- An XSDAnnotation has a property "userInformation" that contains an `xsd:documentation` (DOM) Element. The value of the `xsd:documentation` Element is mapped to a UML «Documentation» Comment body. The UML contextual container has the new Comment added to its ownedComments.
- An XSDAnnotation also has a property "`applicationinformation`" that contains an `xsd:appinfo` (DOM) Element. The `xsd:appinfo` Element contains (DOM) Elements defined by the NDR rules. All NDR defined elements are either in the NIEM appinfo namespace or the NIEM appinfo2 namespace.

- `appinfo:ConformantIndicator` is used to indicate NIEM conformance for an XSDSchema or an XSDImport of an XSDSchema. The UML «Namespace» isConformant tag is mapped from the value of the `appinfo:ConformantIndicator` in the source XSDSchema.
- `appinfo:Base` is a NIEM defined element that has 2 attributes: appinfo:name and appinfo:namespace. Together, the name and namespace uniquely reference an XSDComponent within a particular symbol space. The value of `appinfo:name` and appinfo:namespace are used in the determination of a particular subtype of the «NIEMType» Stereotype is to be applied to the UML Class. The information is used when it identifies an XSDTypeDefinition within the *structures* schema. For XSDTypeDefinitions located in schemas other than the *structures* schema, the baseTypeDefinition inheritance chain is followed until a *structures* reference is found. For an ObjectType or RoleType, the stereotype is not applied.
- `appinfo:ReferenceTarget` is a NIEM defined element that has 2 attributes: `appinfo:name` and `appinfo:namespace`. Together, the name and namespace uniquely reference an XSDTypeDefinition. The value of `appinfo:name` and `appinfo:namespace` are mapped to a UML Property type declaration representing a “Reference” Element.
- `appinfo:AppliesTo` is a NIEM defined element that has 2 attributes: appinfo:name and appinfo:namespace. Together, the name and namespace uniquely reference an XSDTypeDefinition. The value of `appinfo:name` and `appinfo:namespace` are mapped to the supplier of a UML «Application» Usage.
- `appinfo:ExternalAdapterTypeIndicator` is a NIEM-defined (DOM) Element. A UML «AdapterType» is mapped from an `appinfo:ExternalAdapterTypeIndicator` with value “true”.

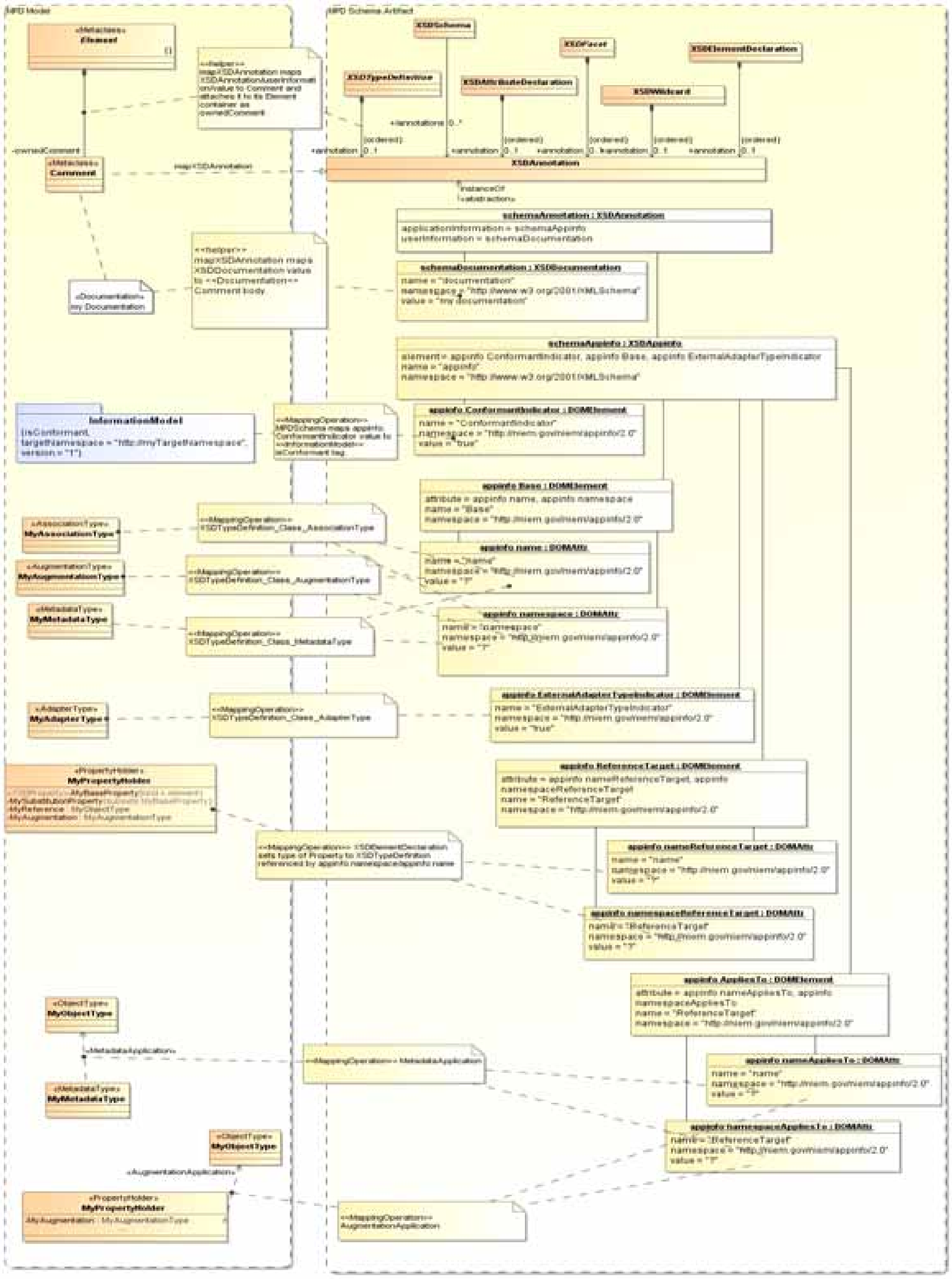

MPD Model
MPD Schema Artifact
XSDSchema
XSDFacet
XSDElementDeclaration
XSDTypeDefinition
XSDAttributeDeclaration
XSDWildcard
XSDAnnotation
«Metaclass» Element
«Metaclass» Comment
schemaAnnotation : XSDAnnotation
schemaDocumentation : XSDDocumentation
schemaAppinfo : XSDAppinfo
InformationModel
MyAssociationType
MyAugmentationType
MyMetadataType
MyAdapterType
MyPropertyHolder
MyObjectType
appinfo:ConformantIndicator : DOMElement
appinfo:Base : DOMElement
appinfo:name : DOMAttr
appinfo:namespace : DOMAttr
appinfo:ExternalAdapterTypeIndicator : DOMElement
appinfo:ReferenceTarget : DOMElement
appinfo:AppliesTo : DOMElement

Figure 9.25 - MPD Schema Artifacts to NIEM-UML MPD Model – Annotation Mapping Overview

An «InformationModel» is mapped from an XSDSchema, as illustrated in Figure 9.26.

- Tags on the «InformationModel» are mapped from either properties of the XSDSchema, or from XSDAnnotation as outlined in the previous paragraph.
- XSDImports are ignored for mapping. Any reference to XSDComponents within an external Schema will result in mapping the Schema to an «InformationModel».
- XSDSchemaContent is mapped to packagedElements within the «InformationModel». For schema symbol spaces other than XSDTypeDefinitions, a container is produced (such as «PropertyHolder» to hold element and attribute symbol spaces).

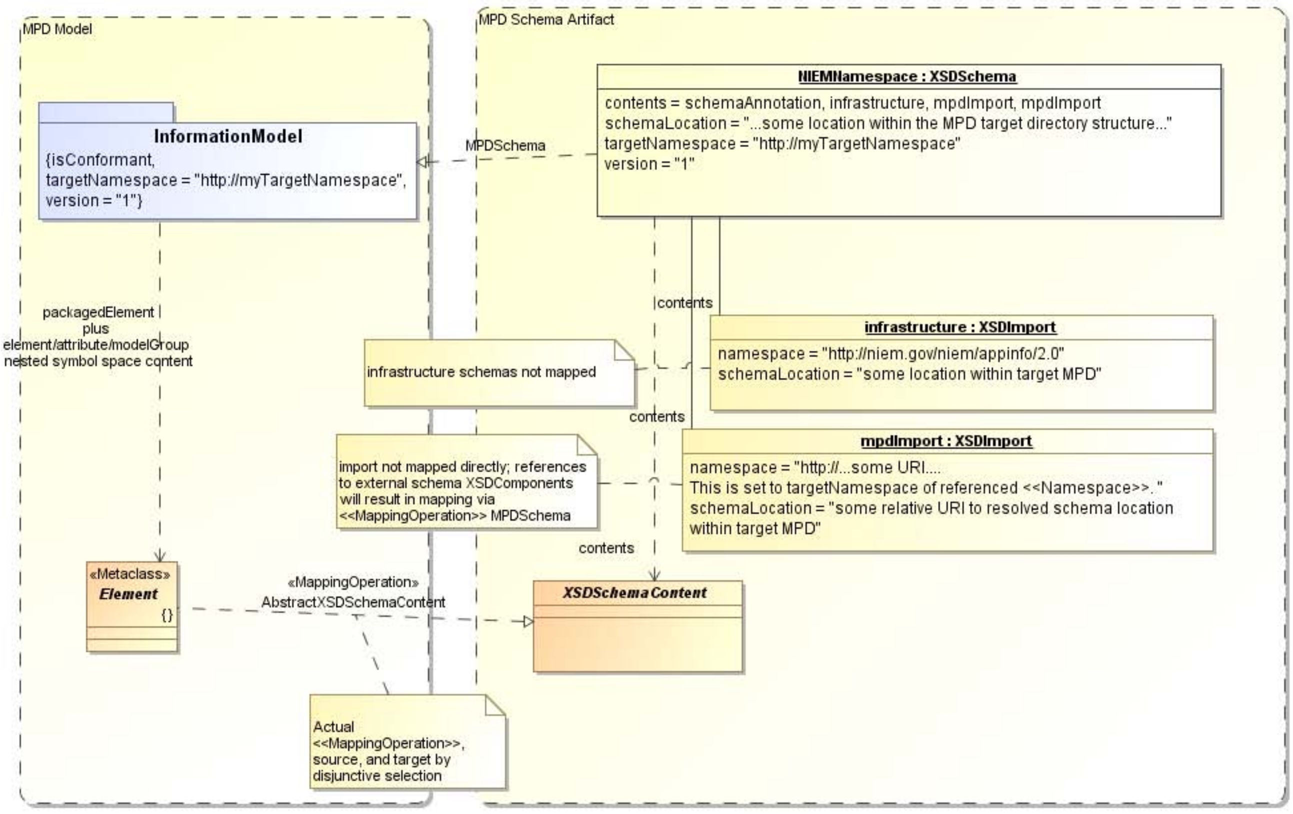

MPD Model
MPD Schema Artifact
InformationModel
{isConformant,
targetNamespace = "http://myTargetNamespace",
version = "1"}
MPDSchema
NIEMNamespace : XSDSchema
contents = schemaAnnotation, infrastructure, mpdImport, mpdImport
schemaLocation = "...some location within the MPD target directory structure..."
targetNamespace = "http://myTargetNamespace"
version = "1"
packagedElement
plus
element/attribute/modelGroup
nested symbol space content
contents
infrastructure : XSDImport
namespace = "http://niem.gov/niem/appinfo/2.0"
schemaLocation = "some location within target MPD"
infrastructure schemas not mapped
contents
mpdImport : XSDImport
namespace = "http://...some URI....
This is set to targetNamespace of referenced <<Namespace>>. "
schemaLocation = "some relative URI to resolved schema location
within target MPD"
import not mapped directly; references
to external schema XSDComponents
will result in mapping via
<<MappingOperation>> MPDSchema
contents
«Metaclass»
Element
{}
«MappingOperation»
AbstractXSDSchemaContent
XSDSchemaContent
Actual
<<MappingOperation>>,
source, and target by
disjunctive selection

Figure 9.26 - MPD Schema Artifacts to NIEM-UML MPD Model – «InformationModel» Mapping Overview

«NIEMType»s are mapped from XSDComplexTypeDefinitions, as illustrated in Figure 9.27. NIEM-specific meta information from XSDAnnotations are used to help determine the stereotype to be applied, as outlined earlier.

- Inheritance in the NIEM-UML model may be specified as a Generalization or as a «Restriction» Realization. «Restriction» will be used when the derivationMethod of the source XSDComplexTypeDefinition is set to *restriction*. In all other cases, Generalization is used to represent inheritance.
- When the baseTypeDefinitions of the source XSDComplexTypeDefinition is one of the NIEM NDR-defined XSDComplexTypeDefinitions from the “structures” XSDSchema, no inheritance is produced for the target UML Class.

- The attributeContents of the source XSDComplexTypeDefinition are mapped to «XSDProperty»{kind=attribute}.
- For a NIEM-PIM, an XSDComplexTypeDefinition whose XSDComplexTypeContent is an XSDSimpleTypeDefinition will normally be mapped to the same UML MetaClass as the baseTypeDefinition of the XSDSimpleTypeDefinition. A ComplexType with simpleContent derived from a schema datatype will be mapped to a PrimitiveType and will be a specialization of a PrimitiveType. In this case, the «ValueRestriction» Stereotype is not applied. A ComplexType with simpleContent derived from an Enumeration (i.e., a SimpleType with enumeration facets) will be an Enumeration with no owned literals. A ComplexType with simpleContent derived from a DataType mapping will be an unstereotyped DataType.

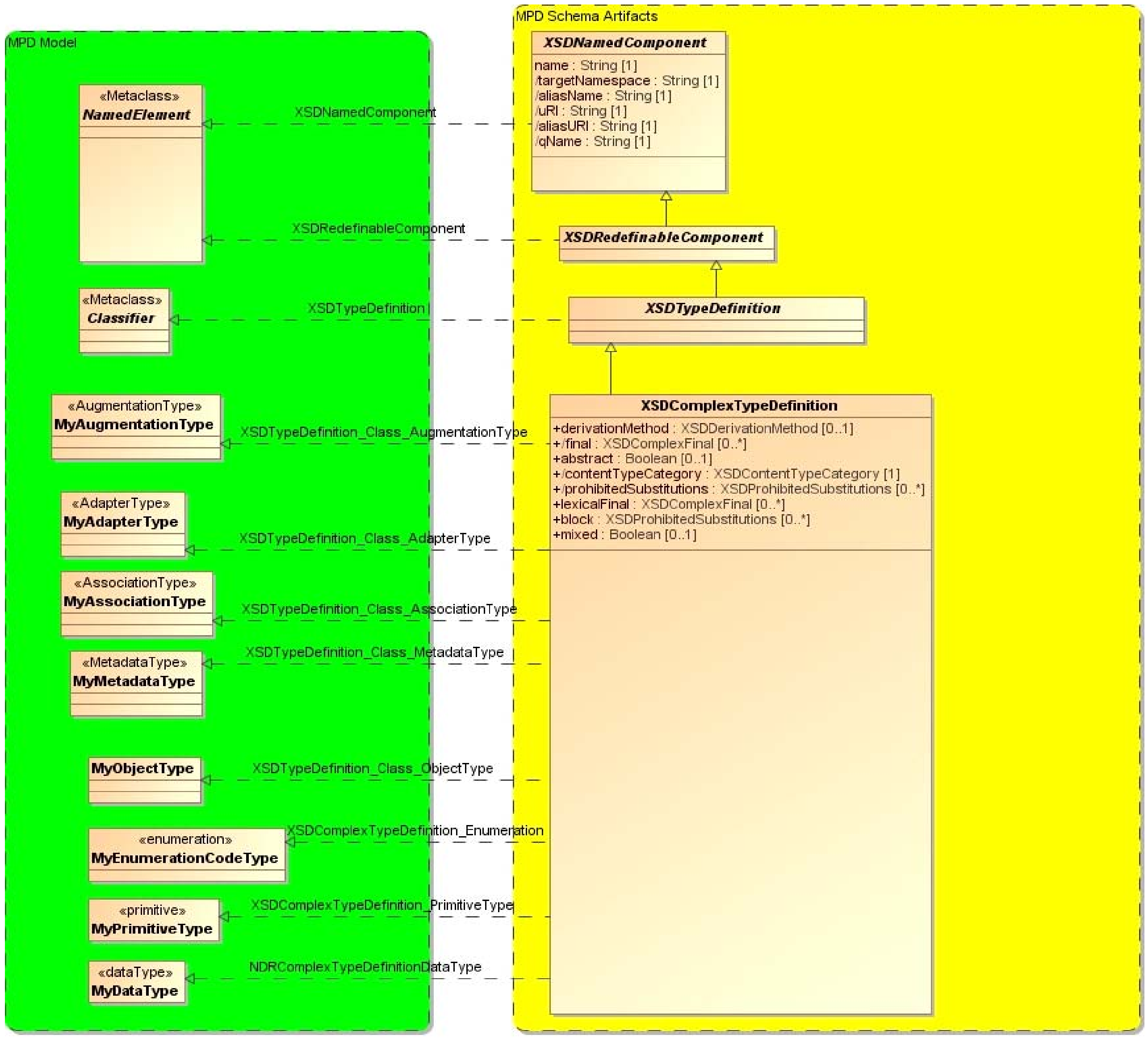

MPD Model
MPD Schema Artifacts
«Metaclass» NamedElement
XSDNamedComponent
name : String [1]
/targetNamespace : String [1]
/aliasName : String [1]
/uRI : String [1]
/aliasURI : String [1]
/qName : String [1]
XSDRedefinableComponent
«Metaclass» Classifier
XSDTypeDefinition
«AugmentationType» MyAugmentationType
XSDTypeDefinition_Class_AugmentationType
XSDComplexTypeDefinition
+derivationMethod : XSDDerivationMethod [0..1]
+/final : XSDComplexFinal [0..*]
+abstract : Boolean [0..1]
+/contentTypeCategory : XSDContentTypeCategory [1]
+/prohibitedSubstitutions : XSDProhibitedSubstitutions [0..*]
+lexicalFinal : XSDComplexFinal [0..*]
+block : XSDProhibitedSubstitutions [0..*]
+mixed : Boolean [0..1]
«AdapterType» MyAdapterType
XSDTypeDefinition_Class_AdapterType
«AssociationType» MyAssociationType
XSDTypeDefinition_Class_AssociationType
«MetadataType» MyMetadataType
XSDTypeDefinition_Class_MetadataType
MyObjectType
XSDTypeDefinition_Class_ObjectType
«enumeration» MyEnumerationCodeType
XSDComplexTypeDefinition_Enumeration
«primitive» MyPrimitiveType
XSDComplexTypeDefinition_PrimitiveType
«dataType» MyDataType
NDRComplexTypeDefinitionDataType

Figure 9.27 - MPD Schema Artifacts to NIEM-UML MPD Model – Type Mapping Overview

Figure 9.28 illustrates mappings between a NIEM PSM and MPD Schema Artifacts, as related to XSDFacets. Facets in the NIEM PSM are represented as tag values on a «ValueRestriction». Facets in the XSD meta-model are XSDFacets owned by an XSDSimpleTypeDefinition. The mapping provides for populating «ValueRestriction» tag values from XSDFacets. An XSDSimpleTypeDefinition containing enumeration facets is mapped to a UML Enumeration.

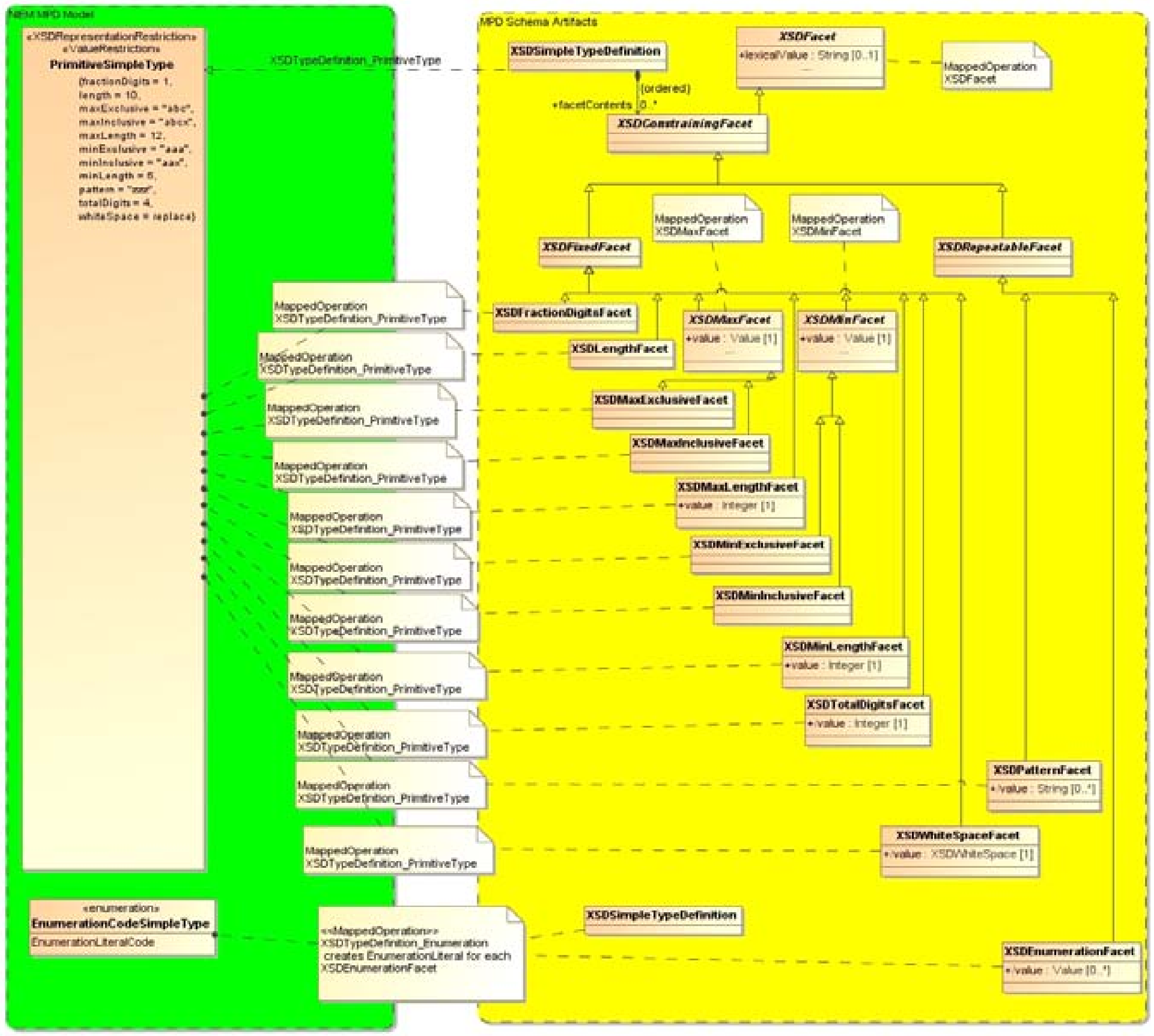


Figure 9.28 - MPD Schema Artifacts to NIEM-UML MPD Model – Facet Mapping Overview

Figure 9.29 illustrates mappings from non-atomic XSDSimpleTypeDefinitions, XSDComplexTypeDefinitions, and top level features:

- «List»s are represented in the source MPD Schema as XSDSimpleTypeDefinitions with the "variety" tag value computed as *list*. This maps to a «List» with a single property. The type of that property is mapped from the itemTypeDefinition property of XSDSimpleTypeDefinition.

- «Union»s are represented in the MPD Schema as XSDSimpleTypeDefinitions with the "variety" tag value computed as *union*. This maps to a «Union» with a «UnionOf» Usage for each memberTypeDefinition.
- A «NIEMType» is mapped from an XSDComplexTypeDefinition. The abstract XSDComplexTypeContent will be either an XSDSimpleTypeDefinition or an XSDParticle, depending upon the baseTypeDefinition.
  - When the XSDComplexTypeContent is an XSDParticle, then the «NIEMType» is also mapped from the content of the XSDParticle, which is an XSDParticleContent. The XSDParticleContent is typically an XSDModelGroup{compositor=*sequence*}.
  - The contents of the XSDModelGroup are XSDParticles. When the particleContent is an XSDElementDeclaration, it is mapped to an «XSDProperty»{kind=element} owned by the UML container context. The upper/lower multiplicity bounds of the «XSDProperty» are mapped from the maxOccurs/minOccurs of the XSDParticle.
  - The resolvedElementDeclaration property of a source XSDElementDeclaration is mapped to a «References» Realization. The supplier of the «References» is set to the Property mapped from the resolvedElementDeclaration.
- An «XSDProperty» contained by a «PropertyHolder» is mapped from an XSDElementDeclaration directly contained by an XSDSchema.
- An «XSDProperty» that has a subsetProperty reference to another «XSDProperty» is mapped from a *substitutionGroup* reference between elements.
- «PropertyHolder»s contained by a «Namespace» represent schema symbol spaces. Within a «PropertyHolder», an «XSDProperty»{kind=element} represents a member of the schema element symbol space. Correspondingly, an «XSDProperty»{kind=attribute} represents a member of the schema attribute symbol space. There is no direct physical manifestation of symbol spaces within an XSDSchema, they are implicit based on whether the top level components are XSDAttribute or XSDElement (hence the «PropertyHolder» itself is not mapped from an XSDComponent and «PropertyHolder»s are generated on demand when mapping a top-level XSDElementDeclaration and/or XSDAttributeDeclaration). Generalizations may be added to some «PropertyHolder»s to satisfy *subsetsProperty* reference semantics.
- XSDComplexTypeDefinitions that have baseTypeDefinitions residing in the structures XSDSchema become some subtype of NIEMType having no inheritance.

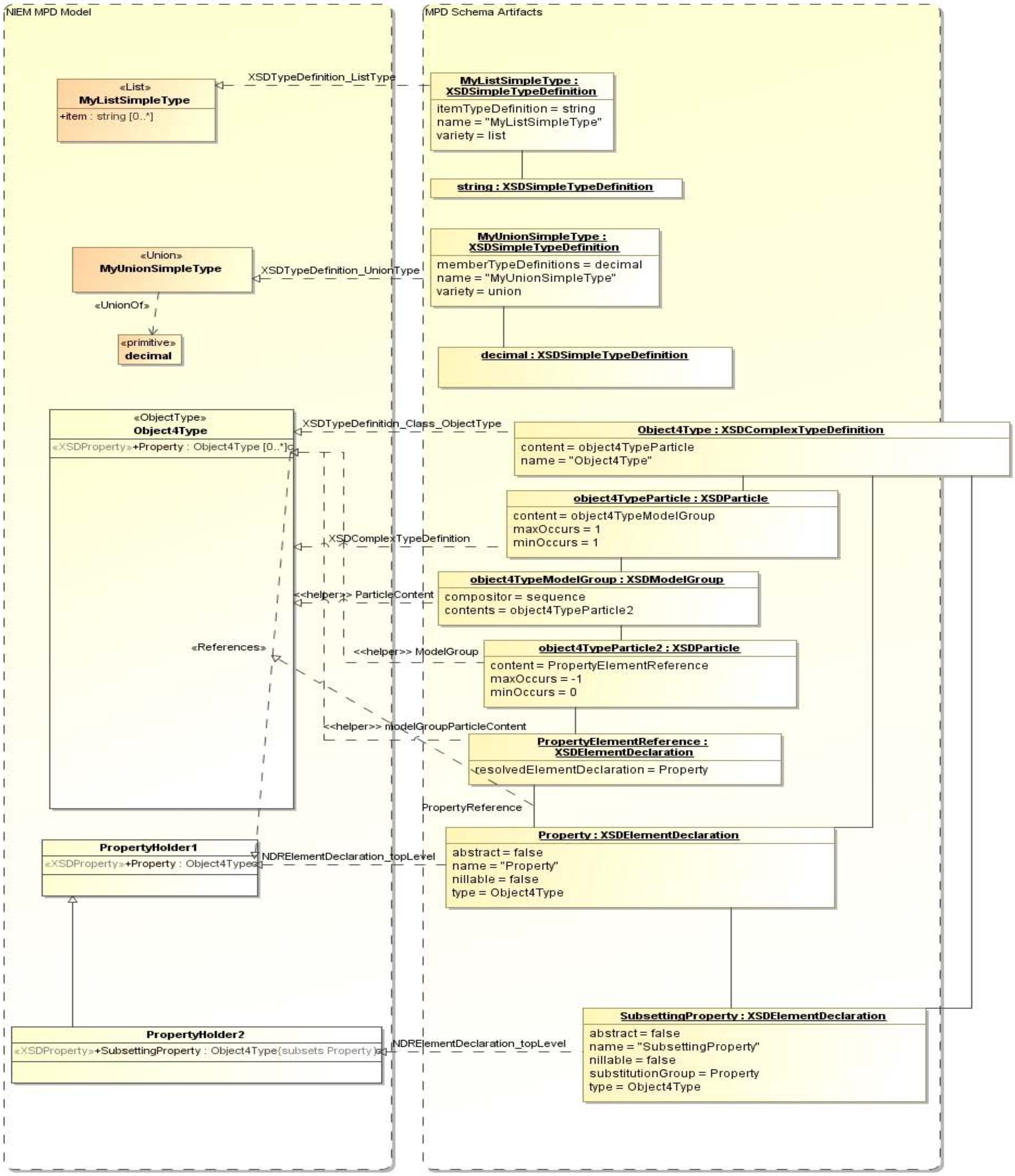

NIEM MPD Model
MPD Schema Artifacts
«List» MyListSimpleType
+item : string [0..*]
XSDTypeDefinition_ListType
MyListSimpleType : XSDSimpleTypeDefinition
itemTypeDefinition = string
name = "MyListSimpleType"
variety = list
string : XSDSimpleTypeDefinition
«Union» MyUnionSimpleType
XSDTypeDefinition_UnionType
MyUnionSimpleType : XSDSimpleTypeDefinition
memberTypeDefinitions = decimal
name = "MyUnionSimpleType"
variety = union
«UnionOf»
«primitive» decimal
decimal : XSDSimpleTypeDefinition
«ObjectType» Object4Type
«XSDProperty»+Property : Object4Type [0..*]
XSDTypeDefinition_Class_ObjectType
Object4Type : XSDComplexTypeDefinition
content = object4TypeParticle
name = "Object4Type"
object4TypeParticle : XSDParticle
content = object4TypeModelGroup
maxOccurs = 1
minOccurs = 1
XSDComplexTypeDefinition
object4TypeModelGroup : XSDModelGroup
compositor = sequence
contents = object4TypeParticle2
<<helper>> ParticleContent
«References»
<<helper>> ModelGroup
object4TypeParticle2 : XSDParticle
content = PropertyElementReference
maxOccurs = -1
minOccurs = 0
<<helper>> modelGroupParticleContent
PropertyElementReference : XSDElementDeclaration
resolvedElementDeclaration = Property
PropertyReference
PropertyHolder1
«XSDProperty»+Property : Object4Type
NDRElementDeclaration_topLevel
Property : XSDElementDeclaration
abstract = false
name = "Property"
nillable = false
type = Object4Type
PropertyHolder2
«XSDProperty»+SubsettingProperty : Object4Type{subsets Property}
NDRElementDeclaration_topLevel
SubsettingProperty : XSDElementDeclaration
abstract = false
name = "SubsettingProperty"
nillable = false
substitutionGroup = Property
type = Object4Type

Figure 9.29 - MPD Schema Artifacts to NIEM-UML MPD Model – Non-atomic Type Mapping Overview

Figure 9.30 illustrates mappings related to «Choice». The mapping is similar to a «NIEMType» to XSDComplexTypeDefinition, the variation being that the XSDParticleContent mapped to a Property is from an XSDModelGroup{compositor=*choice*} instead of an XSDElementDeclaration:

- A «NIEMType» typically maps from an XSDComplexTypeDefinition, an XSDParticle, and an XSDModelGroup. For a NIEM-conformant schema, the XSDModelGroup compositor is *sequence*.
- The XSDModelGroup contents are an ordered set of XSDParticles, each having an XSDParticleContent as content. Each XSDParticle normally maps to a UML Property with multiplicity as specified by the XSDParticle minOccurs/maxOccurs. The XSDParticleContent is typically an XSDElementDeclaration and further refines the Property to be an «XSDProperty»{kind=element}. If the XSDParticleContent is an XSDModelGroup{compositor=*choice*}, the XSDModelGroup is mapped to a «Choice» and the Property is refined to have a type of that «Choice».
- The XSDModelGroup{compositor=*choice*} has contents that are an ordered set of XSDParticles, each having an XSDParticleContent as content. These XSDParticles define the contents of a «Choice», which typically map to the sequence of «XSDProperty»s.

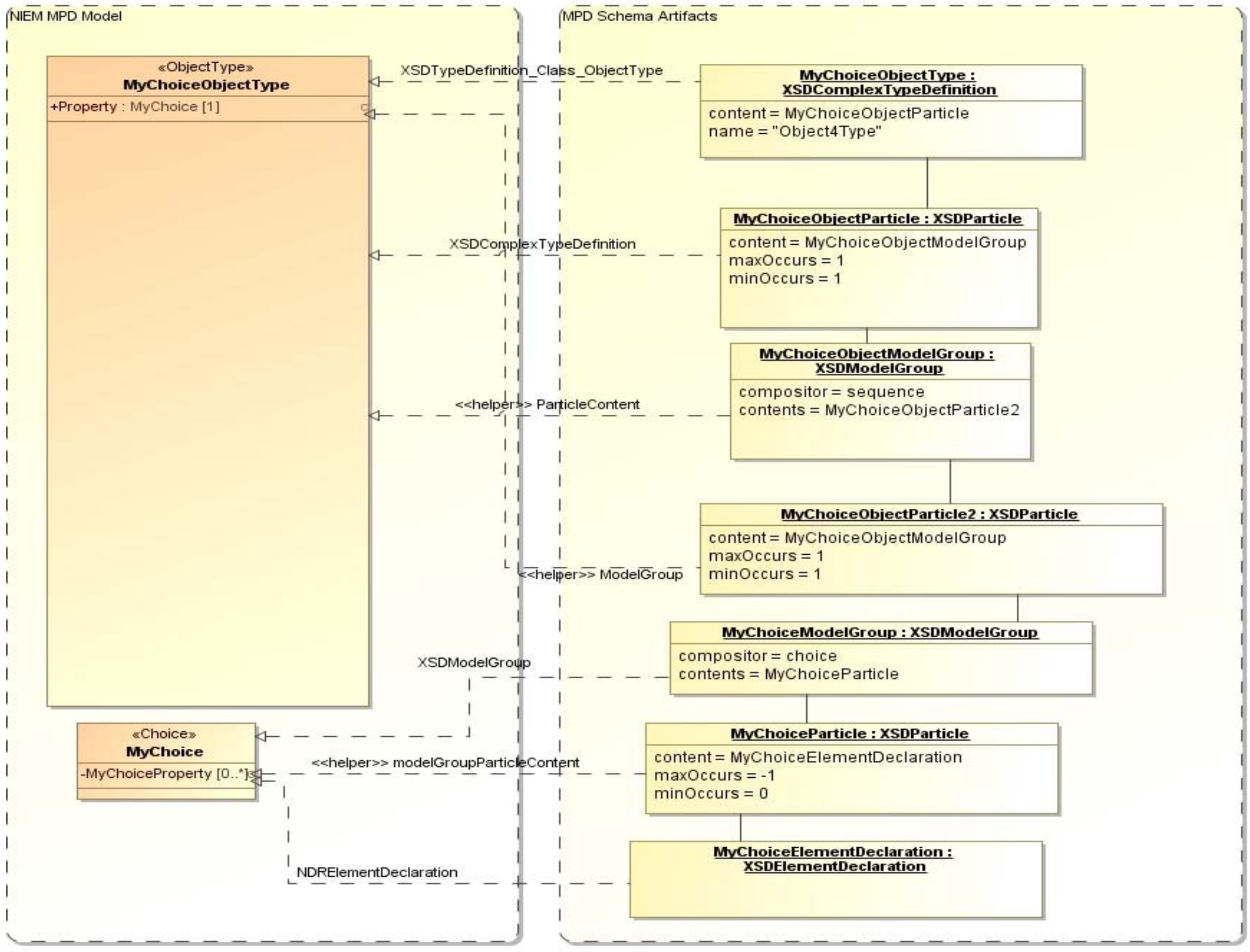

NIEM MPD Model
MPD Schema Artifacts
«ObjectType»
MyChoiceObjectType
+Property : MyChoice [1]
«Choice»
MyChoice
-MyChoiceProperty [0..*]
XSDTypeDefinition_Class_ObjectType
XSDComplexTypeDefinition
<<helper>> ParticleContent
<<helper>> ModelGroup
XSDModelGroup
<<helper>> modelGroupParticleContent
NDRElementDeclaration
MyChoiceObjectType : XSDComplexTypeDefinition
content = MyChoiceObjectParticle
name = "Object4Type"
MyChoiceObjectParticle : XSDParticle
content = MyChoiceObjectModelGroup
maxOccurs = 1
minOccurs = 1
MyChoiceObjectModelGroup : XSDModelGroup
compositor = sequence
contents = MyChoiceObjectParticle2
MyChoiceObjectParticle2 : XSDParticle
content = MyChoiceObjectModelGroup
maxOccurs = 1
minOccurs = 1
MyChoiceModelGroup : XSDModelGroup
compositor = choice
contents = MyChoiceParticle
MyChoiceParticle : XSDParticle
content = MyChoiceElementDeclaration
maxOccurs = -1
minOccurs = 0
MyChoiceElementDeclaration : XSDElementDeclaration

Figure 9.30 - MPD Schema Artifacts to NIEM-UML MPD Model – Choice Mapping Overview

Figure 9.31 illustrates some mappings related to baseTypeDefinitions:

- «ValueRestriction» inheritance from NIEM XML Primitive Types is mapped from an XSDSimpleTypeDefinition baseTypeDefinition referencing the datatype counterpart from the XML Schema for Schemas.
- «ValueRestriction» specialization from a general «ValueRestriction» is mapped from an XSDSimpleTypeDefinition baseTypeDefinition.
- A «NIEMType» is mapped from an XSDComplexTypeDefinition.
  - When the content of the XSDComplexTypeDefinition is an XSDSimpleTypeDefinition the mapping creates an «XSDSimpleContent» Realization from the client «NIEMType» (mapped from XSDComplexTypeDefinition) to the supplier «ValueRestriction» (mapped from the baseTypeDefinition of the XSDSimpleTypeDefinition).
    - When the referenced XSDSimpleTypeDefinition is a type defined by the XML Schema for Schemas, or by the NIEM Infrastructure proxy schema, then the XSDSimpleTypeDefinition maps to a type in the XML Primitive Types library of the same name.
    - In all other cases, the «ValueRestriction» mapped from the baseTypeDefinition is the supplier for the «XSDSimpleContent».
    - Any occurance of an XSDAttributeGroupDefinition resolving to *structures:SimpleObjectAttributeGroup* within the attributeContents of the XSDComplexTypeDefinition are not mapped.
- XSDComplexTypeDefinition with a derivationMethod of *restriction* results in creation of a «Restriction» Realization to a supplier «NIEMType» (mapped from the baseTypeDefinition).
- No Generalization or «Restriction» is created when the baseTypeDefinition of the XSDComplexTypeDefinition is an XSDTypeDefinition contained by the NIEM Infrastructure structures schema.

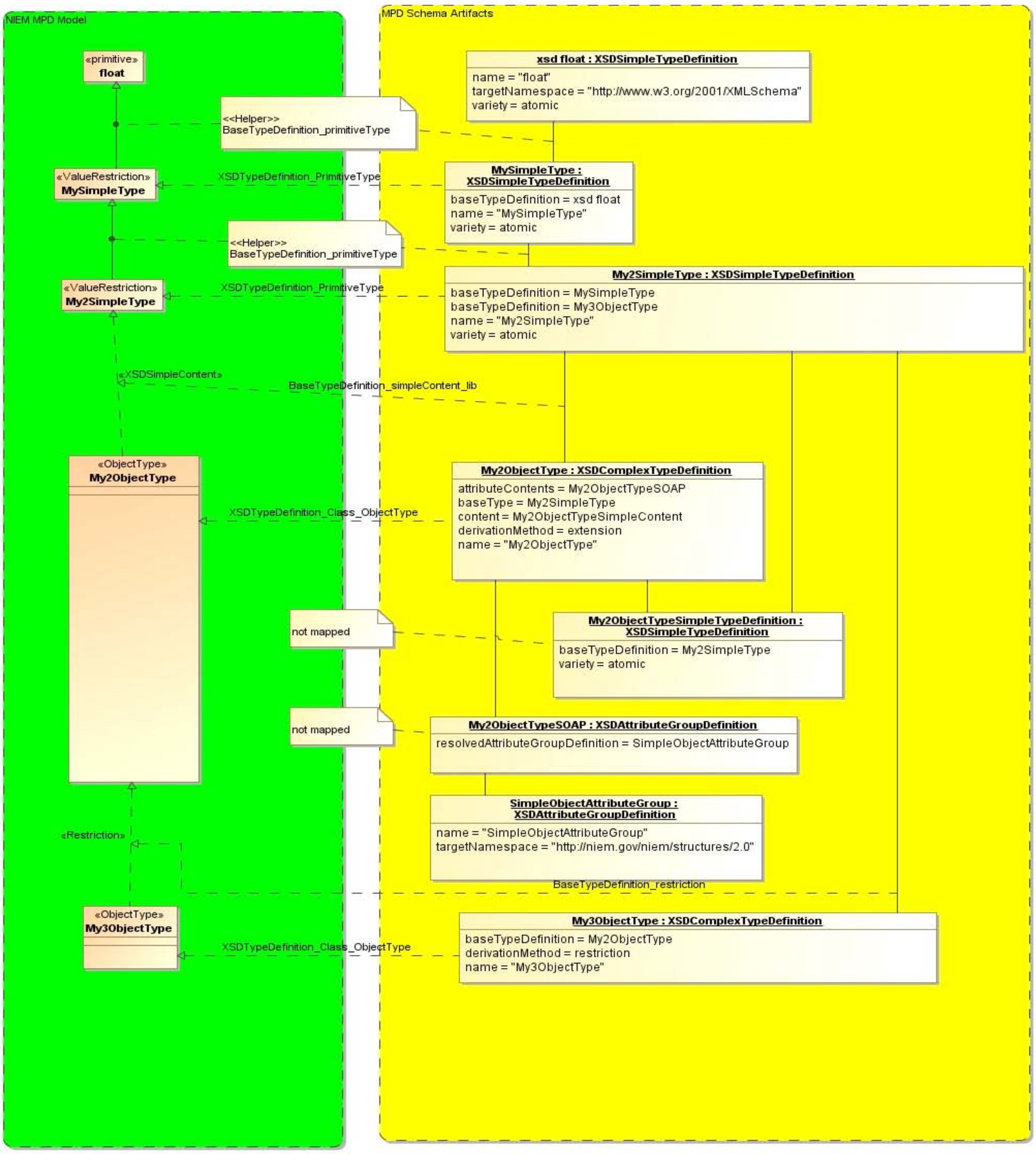

NIEM MPD Model
«primitive» float
<<Helper>> BaseTypeDefinition_primitiveType
«ValueRestriction» MySimpleType
XSDTypeDefinition_PrimitiveType
<<Helper>> BaseTypeDefinition_primitiveType
«ValueRestriction» My2SimpleType
XSDTypeDefinition_PrimitiveType
«XSDSimpleContent»
BaseTypeDefinition_simpleContent_lib
«ObjectType» My2ObjectType
XSDTypeDefinition_Class_ObjectType
not mapped
not mapped
«Restriction»
BaseTypeDefinition_restriction
«ObjectType» My3ObjectType
XSDTypeDefinition_Class_ObjectType
MPD Schema Artifacts
xsd float : XSDSimpleTypeDefinition
name = "float"
targetNamespace = "http://www.w3.org/2001/XMLSchema"
variety = atomic
MySimpleType : XSDSimpleTypeDefinition
baseTypeDefinition = xsd float
name = "MySimpleType"
variety = atomic
My2SimpleType : XSDSimpleTypeDefinition
baseTypeDefinition = MySimpleType
baseTypeDefinition = My3ObjectType
name = "My2SimpleType"
variety = atomic
My2ObjectType : XSDComplexTypeDefinition
attributeContents = My2ObjectTypeSOAP
baseType = My2SimpleType
content = My2ObjectTypeSimpleContent
derivationMethod = extension
name = "My2ObjectType"
My2ObjectTypeSimpleTypeDefinition : XSDSimpleTypeDefinition
baseTypeDefinition = My2SimpleType
variety = atomic
My2ObjectTypeSOAP : XSDAttributeGroupDefinition
resolvedAttributeGroupDefinition = SimpleObjectAttributeGroup
SimpleObjectAttributeGroup : XSDAttributeGroupDefinition
name = "SimpleObjectAttributeGroup"
targetNamespace = "http://niem.gov/niem/structures/2.0"
My3ObjectType : XSDComplexTypeDefinition
baseTypeDefinition = My2ObjectType
derivationMethod = restriction
name = "My3ObjectType"

Figure 9.31 - MPD Schema Artifacts to NIEM-UML MPD Model – baseType Mapping Overview

For NIEMmpdartifact2model, mapping operations are generally invoked with the context of a source XSDComponent and produce some form of target NIEM PSM Element. Most mapping operations are also provided an argument that is the target NIEM-UML container context. The “when” condition for the MappingOperations is normally a function of the source MPD XSDComponent, the type of the source XSDComponent, and/or the target NIEM-UML container context. The mapping operation connects its target NIEM-UMLElement to its container, applies a Stereotype as appropriate, and populates the underlying UML Element properties and/or applied Stereotype tag values.

Figure 9.32 illustrates the disjunction pattern for the NIEMmpdartifact2model transformation.

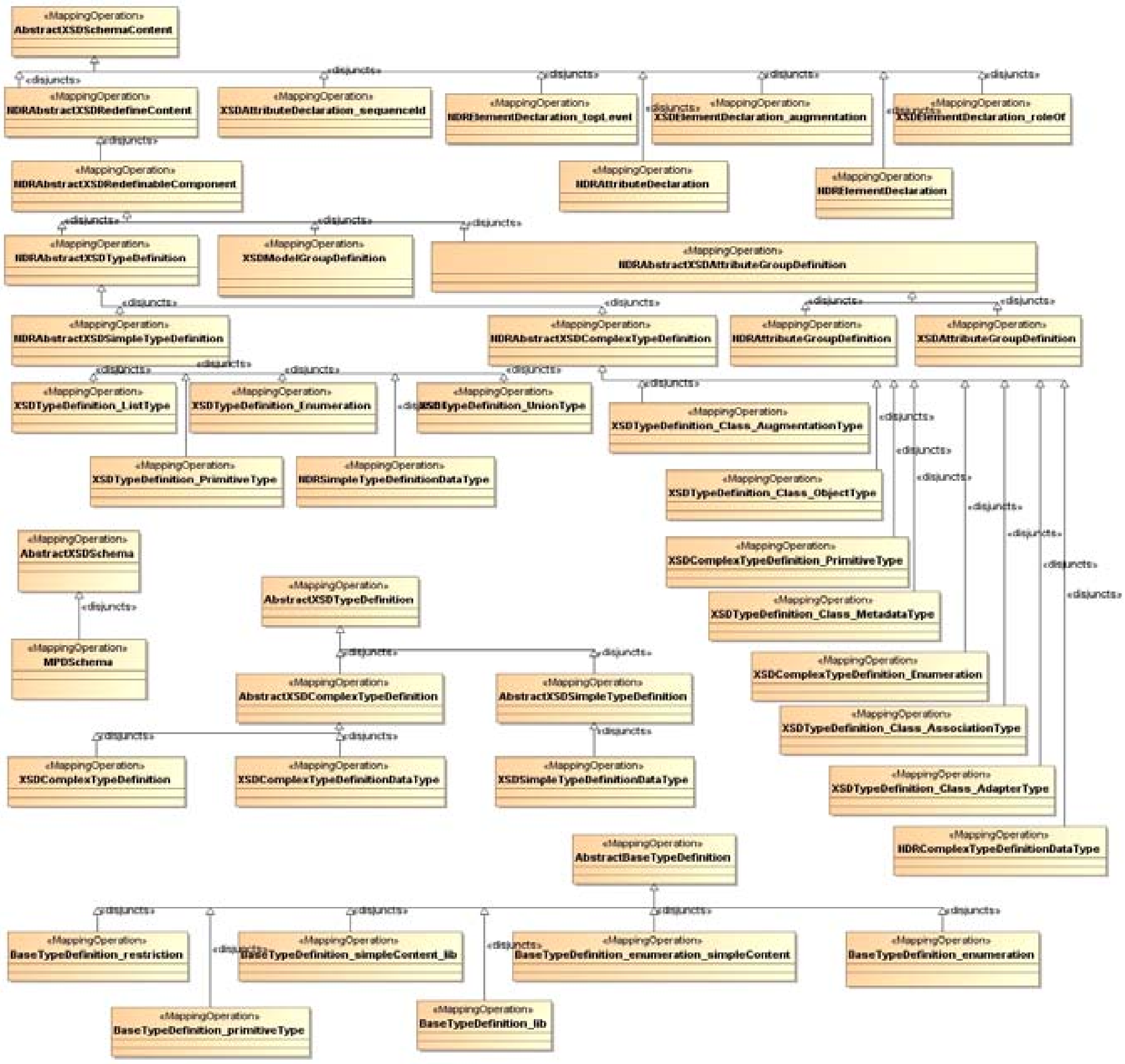

«MappingOperation» AbstractXSDSchemaContent
«disjuncts»
«MappingOperation» NDRAbstractXSDRedefineContent
«MappingOperation» XSDAttributeDeclaration_sequenceId
«MappingOperation» NDRElementDeclaration_topLevel
«MappingOperation» XSDElementDeclaration_augmentation
«MappingOperation» XSDElementDeclaration_roleOf
«MappingOperation» NDRAbstractXSDRedefinableComponent
«MappingOperation» NDRAttributeDeclaration
«MappingOperation» NDRElementDeclaration
«MappingOperation» NDRAbstractXSDTypeDefinition
«MappingOperation» XSDModelGroupDefinition
«MappingOperation» NDRAbstractXSDAttributeGroupDefinition
«MappingOperation» NDRAbstractXSDSimpleTypeDefinition
«MappingOperation» NDRAbstractXSDComplexTypeDefinition
«MappingOperation» NDRAttributeGroupDefinition
«MappingOperation» XSDAttributeGroupDefinition
«MappingOperation» XSDTypeDefinition_ListType
«MappingOperation» XSDTypeDefinition_Enumeration
«MappingOperation» XSDTypeDefinition_UnionType
«MappingOperation» XSDTypeDefinition_Class_AugmentationType
«MappingOperation» XSDTypeDefinition_PrimitiveType
«MappingOperation» NDRSimpleTypeDefinitionDataType
«MappingOperation» XSDTypeDefinition_Class_ObjectType
«MappingOperation» XSDComplexTypeDefinition_PrimitiveType
«MappingOperation» XSDTypeDefinition_Class_MetadataType
«MappingOperation» XSDComplexTypeDefinition_Enumeration
«MappingOperation» XSDTypeDefinition_Class_AssociationType
«MappingOperation» XSDTypeDefinition_Class_AdapterType
«MappingOperation» NDRComplexTypeDefinitionDataType
«MappingOperation» AbstractXSDSchema
«MappingOperation» MPDSchema
«MappingOperation» AbstractXSDTypeDefinition
«MappingOperation» AbstractXSDComplexTypeDefinition
«MappingOperation» AbstractXSDSimpleTypeDefinition
«MappingOperation» XSDComplexTypeDefinition
«MappingOperation» XSDComplexTypeDefinitionDataType
«MappingOperation» XSDSimpleTypeDefinitionDataType
«MappingOperation» AbstractBaseTypeDefinition
«MappingOperation» BaseTypeDefinition_restriction
«MappingOperation» BaseTypeDefinition_simpleContent_lib
«MappingOperation» BaseTypeDefinition_enumeration_simpleContent
«MappingOperation» BaseTypeDefinition_enumeration
«MappingOperation» BaseTypeDefinition_primitiveType
«MappingOperation» BaseTypeDefinition_lib

Figure 9.32 - NIEM MPD Artifact to NIEM MPD Model - Disjunction

For the NIEMmpdartifact2model transformation, each level of the inheritance hierarchy populates properties of a target NIEM-UML Element from the source MPD Artifacts model. Figure 9.33 illustrates the inheritance for most of the MappingOperations in the NIEMmpdartifact2model transformation.

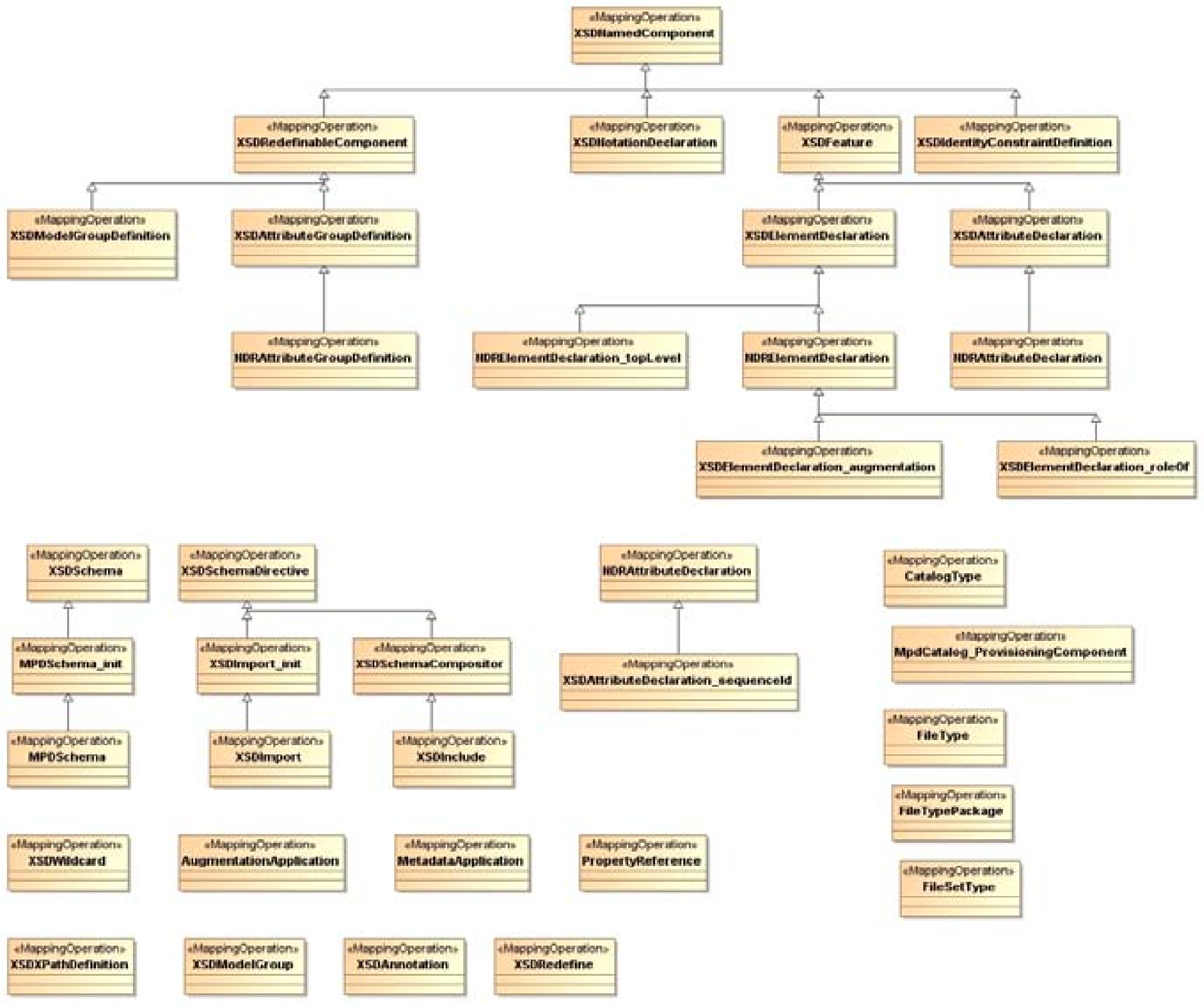

«MappingOperation» XSDNamedComponent
«MappingOperation» XSDRedefinableComponent
«MappingOperation» XSDNotationDeclaration
«MappingOperation» XSDFeature
«MappingOperation» XSDIdentityConstraintDefinition
«MappingOperation» XSDModelGroupDefinition
«MappingOperation» XSDAttributeGroupDefinition
«MappingOperation» XSDElementDeclaration
«MappingOperation» XSDAttributeDeclaration
«MappingOperation» NDRAttributeGroupDefinition
«MappingOperation» NDRElementDeclaration_topLevel
«MappingOperation» NDRElementDeclaration
«MappingOperation» NDRAttributeDeclaration
«MappingOperation» XSDElementDeclaration_augmentation
«MappingOperation» XSDElementDeclaration_roleOf
«MappingOperation» XSDSchema
«MappingOperation» XSDSchemaDirective
«MappingOperation» NDRAttributeDeclaration
«MappingOperation» CatalogType
«MappingOperation» MPDSchema_init
«MappingOperation» XSDImport_init
«MappingOperation» XSDSchemaCompositor
«MappingOperation» XSDAttributeDeclaration_sequenceId
«MappingOperation» MpdCatalog_ProvisioningComponent
«MappingOperation» MPDSchema
«MappingOperation» XSDImport
«MappingOperation» XSDInclude
«MappingOperation» FileType
«MappingOperation» FileTypePackage
«MappingOperation» XSDWildcard
«MappingOperation» AugmentationApplication
«MappingOperation» MetadataApplication
«MappingOperation» PropertyReference
«MappingOperation» FileSetType
«MappingOperation» XSDXPathDefinition
«MappingOperation» XSDModelGroup
«MappingOperation» XSDAnnotation
«MappingOperation» XSDRedefine

Figure 9.33 - NIEM MPD Artifact to NIEM MPD Model - Inheritance

For the NIEMmpdartifact2model transformation, each level of the inheritance hierarchy populates properties of a target NIEM-UML Element from the stereotype tag values and/or UML elements of the source MPD Schemas. Figure 9.34 illustrates the inheritance for most of the MappingOperations in the NIEMmpdartifact2model transformation.

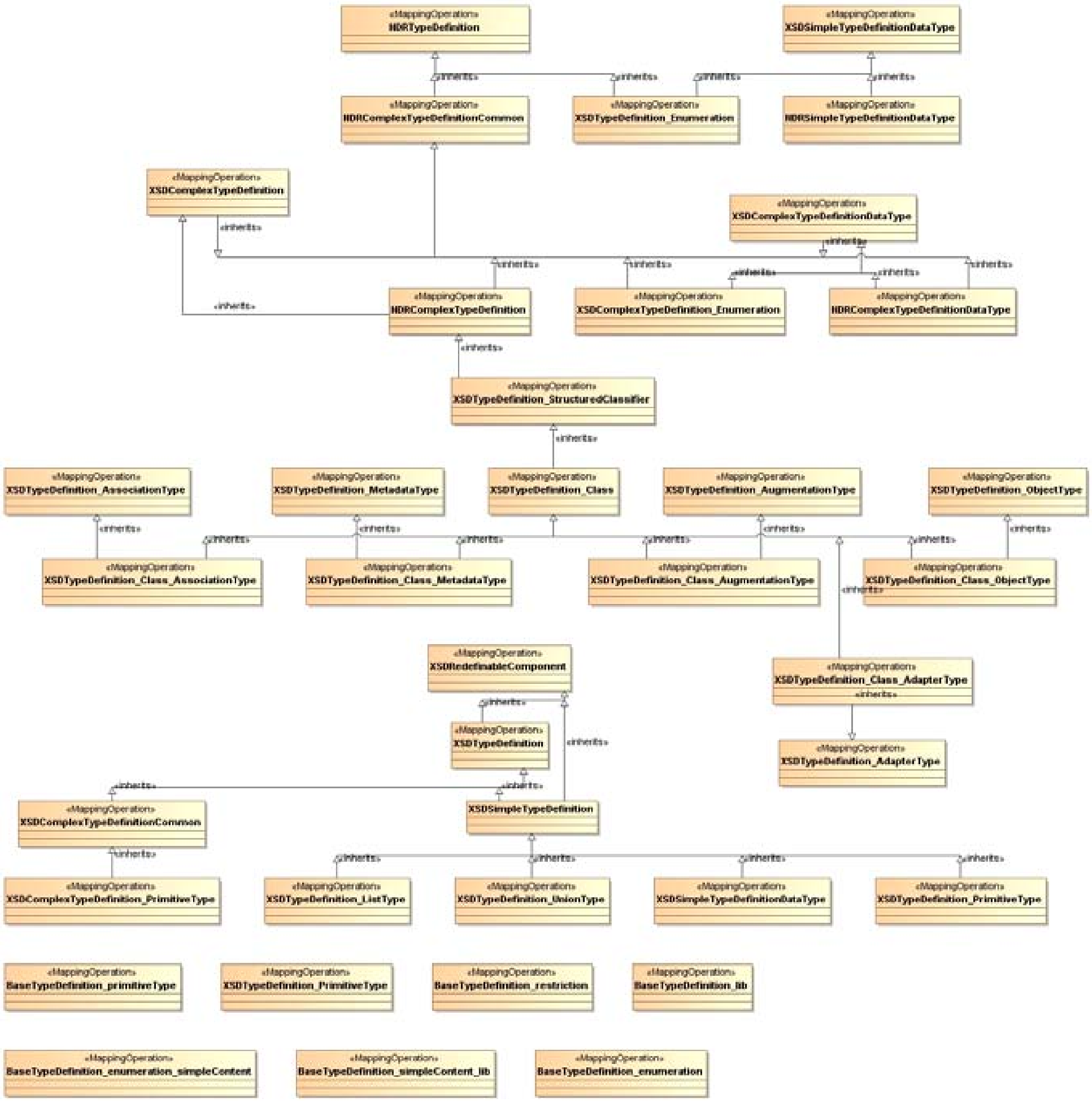


Figure 9.34 - NIEM MPD Artifact to NIEM MPD Model - Inheritance NIEM Type Mapping

Annex A
NIEM-UML PIM Example

(informative)

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 do not have 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” that 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” (Information Exchange Package Documentation).

A.2 Organization of NIEM Information Models and Classes

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.

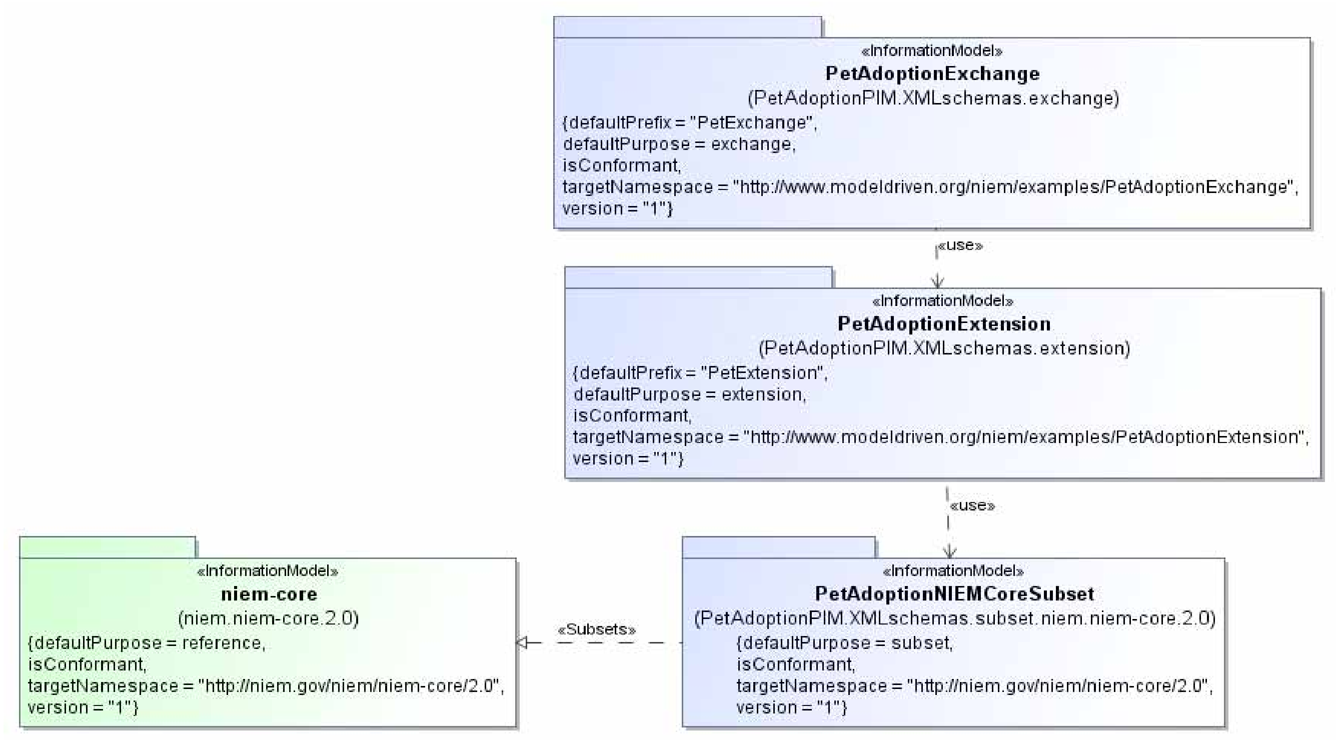

«InformationModel»
PetAdoptionExchange
(PetAdoptionPIM.XMLschemas.exchange)
{defaultPrefix = "PetExchange",
defaultPurpose = exchange,
isConformant,
targetNamespace = "http://www.modeldriven.org/niem/examples/PetAdoptionExchange",
version = "1"}
«use»
«InformationModel»
PetAdoptionExtension
(PetAdoptionPIM.XMLschemas.extension)
{defaultPrefix = "PetExtension",
defaultPurpose = extension,
isConformant,
targetNamespace = "http://www.modeldriven.org/niem/examples/PetAdoptionExtension",
version = "1"}
«use»
«InformationModel»
niem-core
(niem.niem-core.2.0)
{defaultPurpose = reference,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}
«Subsets»
«InformationModel»
PetAdoptionNIEMCoreSubset
(PetAdoptionPIM.XMLschemas.subset.niem.niem-core.2.0)
{defaultPurpose = subset,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}

Figure A.1 - Namespace Organization

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.

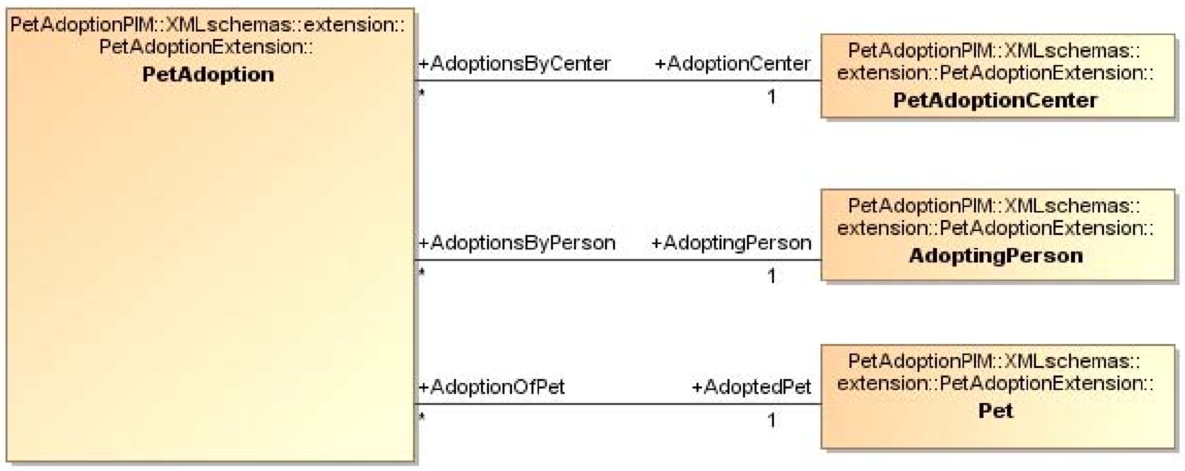
The Pet Adoption PIM model is organized into three subpackages, each representing a particular kind of NIEM information model. Each package is stereotyped as a NIEM «InformationModel» that has tags for the URI, version, and NIEM compliance. The nature of the namespace (as subset, exchange, extension, 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; it selects what is needed from NIEM-Core.
- **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 properties representing actual exchanges between parties. The exchange namespace uses classes from the exchange namespace to specify these data packages.
- **Niem-core** – NIEM-core is the standard namespace as supplied by NIEM, it is not unique to this model. The PetAdoptionNEIMCoreSubset defines the subset of NIEM core needed for pet adoptions.

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 sub clause A.23.

A.3 High-Level Design



PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoption
+AdoptionsByCenter
*
+AdoptionCenter
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoptionCenter
+AdoptionsByPerson
*
+AdoptingPerson
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
AdoptingPerson
+AdoptionOfPet
*
+AdoptedPet
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
Pet

Figure A.2 - High Level Design

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

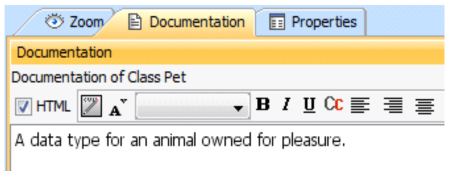
- Pets
- Adopting Persons
- Pet Adoption Centers
- Pet Adoptions

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 above the name). The package owning a class corresponds with a NIEM namespace and is an essential element of the design. Note that “Pet,” “PetAdoption,” “AdoptingPerson,” 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.



Documentation

Documentation of Class Pet

A data type for an animal owned for pleasure.

Figure A.3 - Documenting Elements

A.5 UML Associations Defining NIEM Properties

The lines between the classes in Figure A.2 are UML associations. UML associations define the relationship between classes. Note that there is also the concept of a NIEM association that 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..* that means at least one with no limit. Given this you can see that six properties are defined by the 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 "AdoptingPerson," 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 (Nested in an XML sense) 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

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."

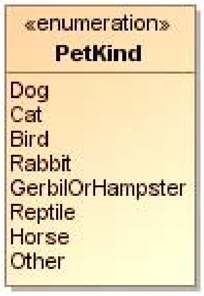


Figure A.4 - Enumeration

A.7 Defining a simple property for an adoption center

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.

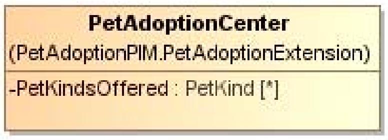

PetAdoptionCenter
(PetAdoptionPIM.PetAdoptionExtension)
-PetKindsOffered : PetKind [*]

Figure A.5 - Simple Properties

A.8 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.

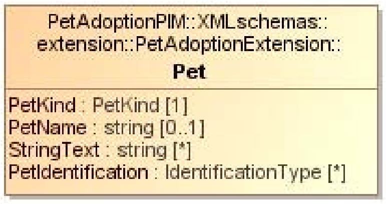

PetAdoptionPIM::XMLschemas::
extension::PetAdoptionExtension::
Pet
PetKind : PetKind [1]
PetName : string [0..1]
StringText : string [*]
PetIdentification : IdentificationType [*]

Figure A.6 - Properties of Pet

Pet has four properties:

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

A.9 Properties Using Classes as Their Types

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.

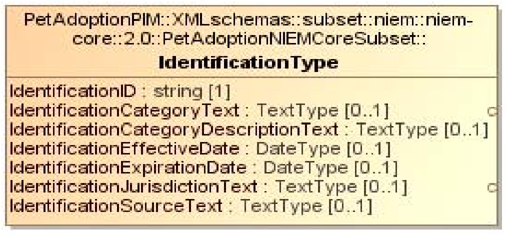
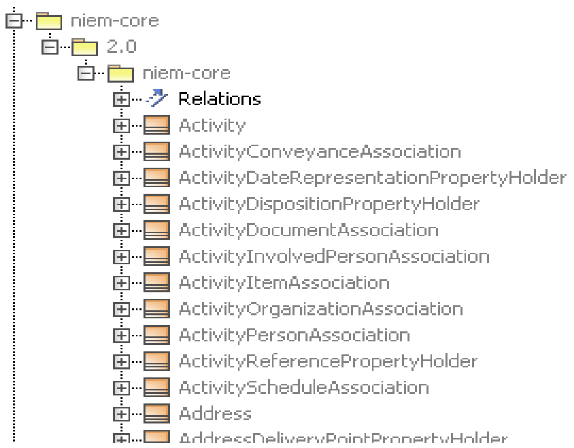

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
IdentificationType
IdentificationID : string [1]
IdentificationCategoryText : TextType [0..1]
IdentificationCategoryDescriptionText : TextType [0..1]
IdentificationEffectiveDate : DateType [0..1]
IdentificationExpirationDate : DateType [0..1]
IdentificationJurisdictionText : TextType [0..1]
IdentificationSourceText : TextType [0..1]

Figure A.7 - Properties Using Identification Class

A.10 Finding Classes in Reference Namespaces

A primary feature of NIEM is the ability to reuse existing definitions. Being able to identify something is a 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. Below are a few of the classes in NIEM-Core as seen from a UML tool.

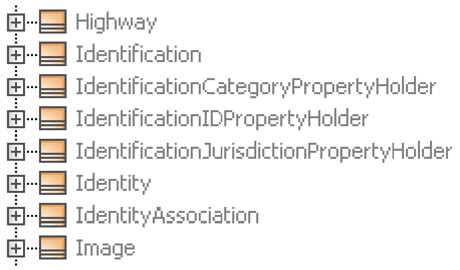
You can see in Figure A.8 that we are "in" the "niem-core," version "2.0." We see a start of the list of classes; there are a lot of them. We may use search features of the UML tool or just scan for what we want.



```
niem-core
  2.0
    niem-core
      Relations
      Activity
      ActivityConveyanceAssociation
      ActivityDateRepresentationPropertyHolder
      ActivityDispositionPropertyHolder
      ActivityDocumentAssociation
      ActivityInvolvedPersonAssociation
      ActivityItemAssociation
      ActivityOrganizationAssociation
      ActivityPersonAssociation
      ActivityReferencePropertyHolder
      ActivityScheduleAssociation
      Address
      AddressDeliveryPointPropertyHolder
```

Figure A.8 - NIEM-Core Listing

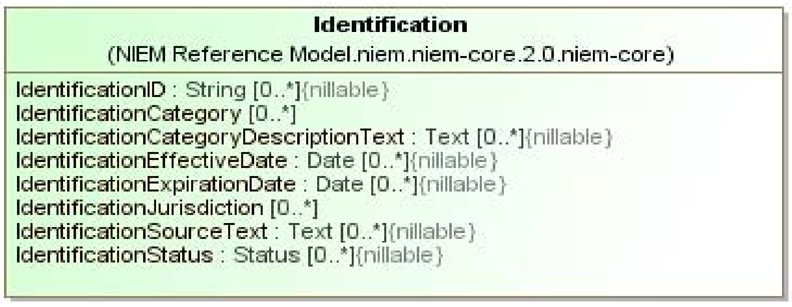
Further down we see classes having to do with Identification:



```
Highway
Identification
IdentificationCategoryPropertyHolder
IdentificationIDPropertyHolder
IdentificationJurisdictionPropertyHolder
Identity
IdentityAssociation
Image
```

Figure A.9 - Finding the Identification Class

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



```
Identification
(NIEM Reference Model.niem.niem-core.2.0.niem-core)
IdentificationID : String [0..*]{nillable}
IdentificationCategory [0..*]
IdentificationCategoryDescriptionText : Text [0..*]{nillable}
IdentificationEffectiveDate : Date [0..*]{nillable}
IdentificationExpirationDate : Date [0..*]{nillable}
IdentificationJurisdiction [0..*]
IdentificationSourceText : Text [0..*]{nillable}
IdentificationStatus : Status [0..*]{nillable}
```

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 group 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.

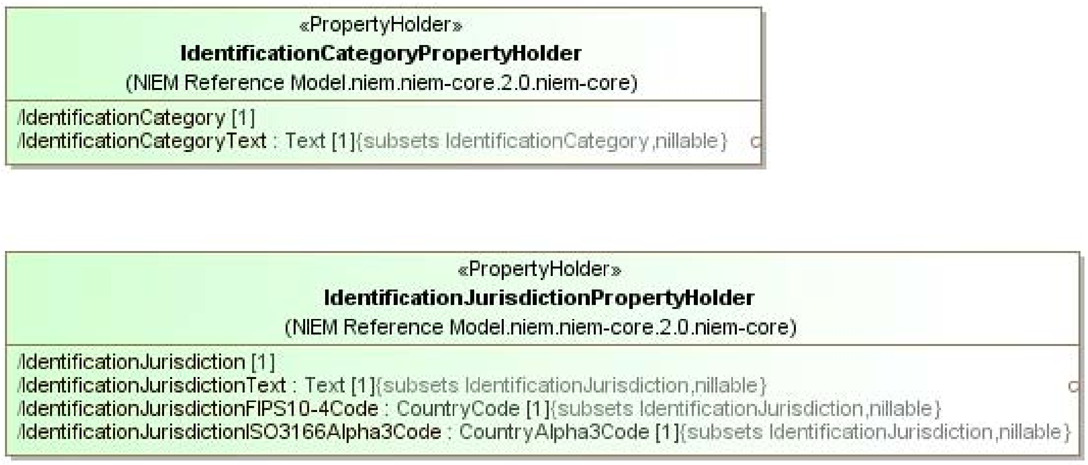

«PropertyHolder»
IdentificationCategoryPropertyHolder
(NIEM Reference Model.niem.niem-core.2.0.niem-core)
/IdentificationCategory [1]
/IdentificationCategoryText : Text [1]{subsets IdentificationCategory,nillable}
«PropertyHolder»
IdentificationJurisdictionPropertyHolder
(NIEM Reference Model.niem.niem-core.2.0.niem-core)
/IdentificationJurisdiction [1]
/IdentificationJurisdictionText : Text [1]{subsets IdentificationJurisdiction,nillable}
/IdentificationJurisdictionFIPS10-4Code : CountryCode [1]{subsets IdentificationJurisdiction,nillable}
/IdentificationJurisdictionISO3166Alpha3Code : CountryAlpha3Code [1]{subsets IdentificationJurisdiction,nillable}

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, let’s 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 «Subsets»

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 has a <<Subsets>> relationship to the model it subsets. Everything in that subset namespace automatically (by name) subsets the corresponding element in the reference namespace.

The following figure shows that PetAdoptionNIEMCoreSubset subsets niem-core.

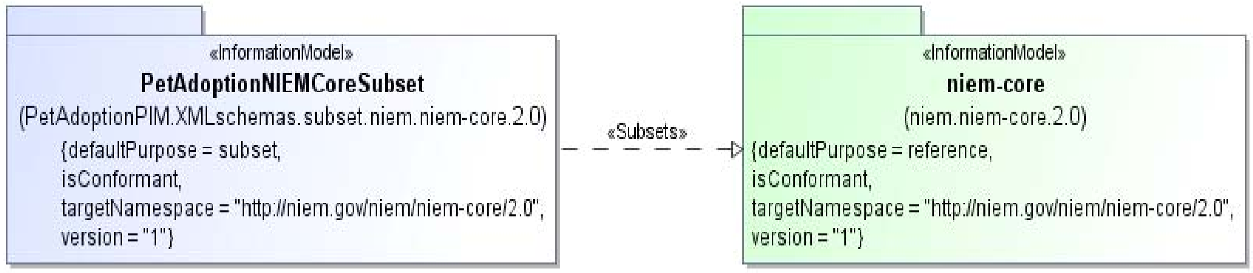

«InformationModel»
PetAdoptionNIEMCoreSubset
(PetAdoptionPIM.XMLschemas.subset.niem.niem-core.2.0)
{defaultPurpose = subset,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}
«Subsets»
«InformationModel»
niem-core
(niem.niem-core.2.0)
{defaultPurpose = reference,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}

Figure A.12 - Identification Subset Class Using <<subsets>>

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.

The “IdentificationType” class in “PetAdoptionNIEMCoreSubset” «subsets» Identification in NIEM-Core. *Note that since it is in PetAdoptionNIEMCoreSubset the explicit subset to the niem-core IdentificationType class is not required (it is automatically inserted by NIEM-UM), but making it explicit is legal and shown here for clarity.*

What we did is copy properties from the NIEM-core version to make the class we want for our pet model, how this is done is tool specific. 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 «Subsets» 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 “IdentificationType” 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!

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.15, “Person” in NIEM-Core is huge!

What we did is just pick 2 properties that we want out of 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. Here we also show an explicit <<Subsets>> to NIEM-CORE, but as with the other subset classes, this is just for clarity and will be put in automatically.

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 it for adoption centers as well.

In our conceptual model we identified an “AdoptingPerson,” a person who participates in adoptions. Adopting person has an additional property “AdoptionsByPerson” that is not part of NIEM-Core, so NIEM-Core person can’t be used directly. What is the relationship between an “AdoptingPerson” and a “Person?” In NIEM, we say that an AdoptingPerson is a “Role” of a person. This allows the *same person* to have multiple roles in the same or different exchanges.

AdoptingPerson is defined as a <<RolePlayedBy>> a person (a stereotype of UML generalization). By using a role rather than ordinary extension, the same person can play multiple roles – or no roles at all.

Also, the roles a person plays may change over time.

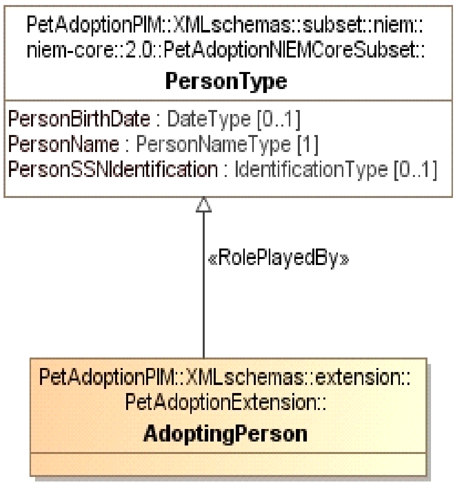

PetAdoptionPIM::XMLschemas::subset::niem::
niem-core::2.0::PetAdoptionNIEMCoreSubset::
PersonType
PersonBirthDate : DateType [0..1]
PersonName : PersonNameType [1]
PersonSSNIdentification : IdentificationType [0..1]
«RolePlayedBy»
PetAdoptionPIM::XMLschemas::extension::
PetAdoptionExtension::
AdoptingPerson

Figure A.13 - Role of Person

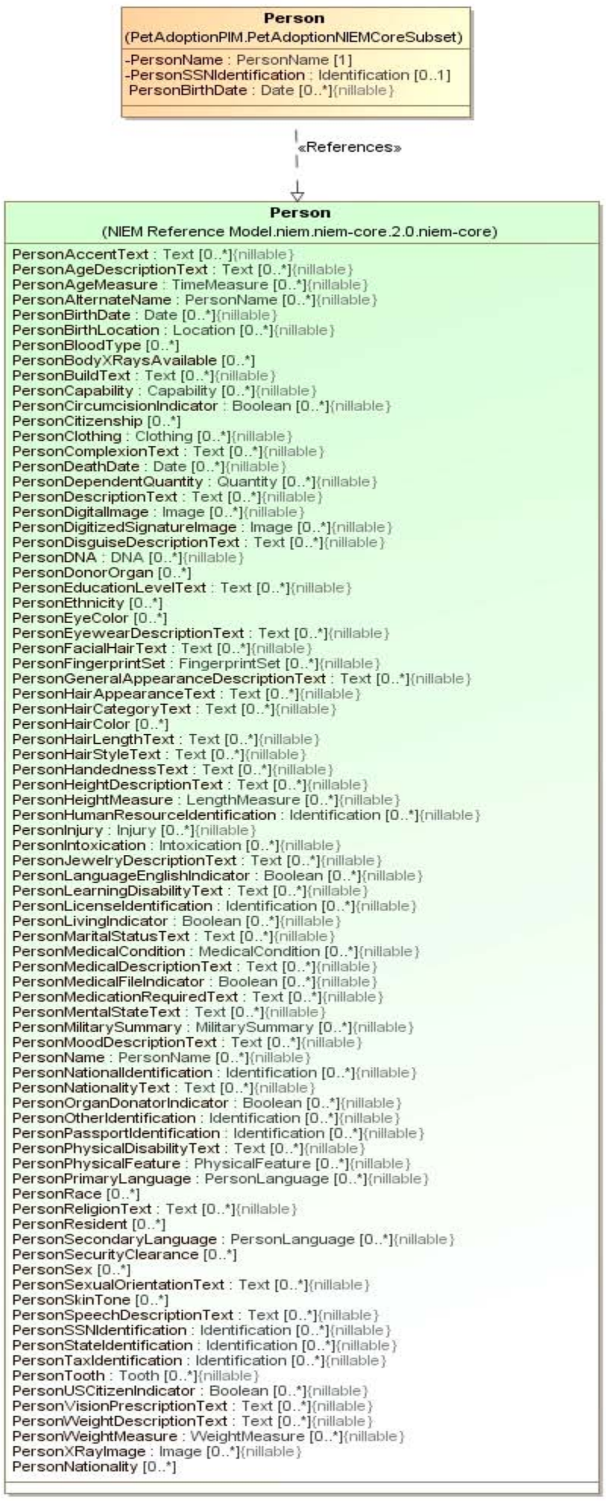
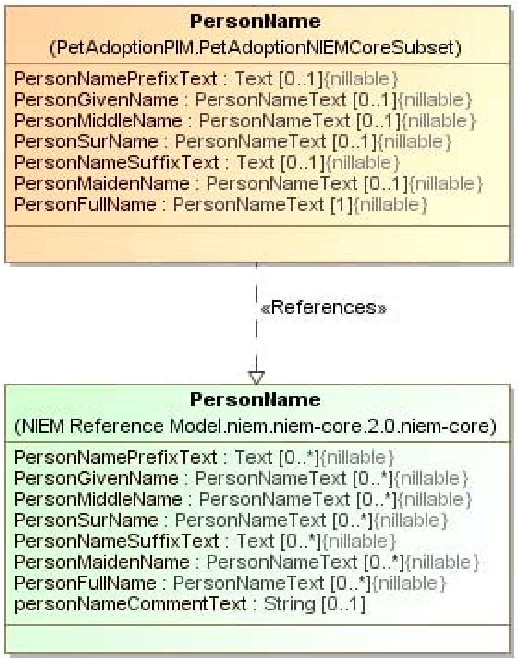

Person
(PetAdoptionPIM.PetAdoptionNIEMCoreSubset)
-PersonName : PersonName [1]
-PersonSSNIdentification : Identification [0..1]
PersonBirthDate : Date [0..*]{nillable}
«References»
Person
(NIEM Reference Model.niem.niem-core.2.0.niem-core)
PersonAccentText : Text [0..*]{nillable}
PersonAgeDescriptionText : Text [0..*]{nillable}
PersonAgeMeasure : TimeMeasure [0..*]{nillable}
PersonAlternateName : PersonName [0..*]{nillable}
PersonBirthDate : Date [0..*]{nillable}
PersonBirthLocation : Location [0..*]{nillable}
PersonBloodType [0..*]
PersonBodyXRaysAvailable [0..*]
PersonBuildText : Text [0..*]{nillable}
PersonCapability : Capability [0..*]{nillable}
PersonCircumcisionIndicator : Boolean [0..*]{nillable}
PersonCitizenship [0..*]
PersonClothing : Clothing [0..*]{nillable}
PersonComplexionText : Text [0..*]{nillable}
PersonDeathDate : Date [0..*]{nillable}
PersonDependentQuantity : Quantity [0..*]{nillable}
PersonDescriptionText : Text [0..*]{nillable}
PersonDigitalImage : Image [0..*]{nillable}
PersonDigitizedSignatureImage : Image [0..*]{nillable}
PersonDisguiseDescriptionText : Text [0..*]{nillable}
PersonDNA : DNA [0..*]{nillable}
PersonDonorOrgan [0..*]
PersonEducationLevelText : Text [0..*]{nillable}
PersonEthnicity [0..*]
PersonEyeColor [0..*]
PersonEyewearDescriptionText : Text [0..*]{nillable}
PersonFacialHairText : Text [0..*]{nillable}
PersonFingerprintSet : FingerprintSet [0..*]{nillable}
PersonGeneralAppearanceDescriptionText : Text [0..*]{nillable}
PersonHairAppearanceText : Text [0..*]{nillable}
PersonHairCategoryText : Text [0..*]{nillable}
PersonHairColor [0..*]
PersonHairLengthText : Text [0..*]{nillable}
PersonHairStyleText : Text [0..*]{nillable}
PersonHandednessText : Text [0..*]{nillable}
PersonHeightDescriptionText : Text [0..*]{nillable}
PersonHeightMeasure : LengthMeasure [0..*]{nillable}
PersonHumanResourceIdentification : Identification [0..*]{nillable}
PersonInjury : Injury [0..*]{nillable}
PersonIntoxication : Intoxication [0..*]{nillable}
PersonJewelryDescriptionText : Text [0..*]{nillable}
PersonLanguageEnglishIndicator : Boolean [0..*]{nillable}
PersonLearningDisabilityText : Text [0..*]{nillable}
PersonLicenseIdentification : Identification [0..*]{nillable}
PersonLivingIndicator : Boolean [0..*]{nillable}
PersonMaritalStatusText : Text [0..*]{nillable}
PersonMedicalCondition : MedicalCondition [0..*]{nillable}
PersonMedicalDescriptionText : Text [0..*]{nillable}
PersonMedicalFileIndicator : Boolean [0..*]{nillable}
PersonMedicationRequiredText : Text [0..*]{nillable}
PersonMentalStateText : Text [0..*]{nillable}
PersonMilitarySummary : MilitarySummary [0..*]{nillable}
PersonMoodDescriptionText : Text [0..*]{nillable}
PersonName : PersonName [0..*]{nillable}
PersonNationalIdentification : Identification [0..*]{nillable}
PersonNationalityText : Text [0..*]{nillable}
PersonOrganDonatorIndicator : Boolean [0..*]{nillable}
PersonOtherIdentification : Identification [0..*]{nillable}
PersonPassportIdentification : Identification [0..*]{nillable}
PersonPhysicalDisabilityText : Text [0..*]{nillable}
PersonPhysicalFeature : PhysicalFeature [0..*]{nillable}
PersonPrimaryLanguage : PersonLanguage [0..*]{nillable}
PersonRace [0..*]
PersonReligionText : Text [0..*]{nillable}
PersonResident [0..*]
PersonSecondaryLanguage : PersonLanguage [0..*]{nillable}
PersonSecurityClearance [0..*]
PersonSex [0..*]
PersonSexualOrientationText : Text [0..*]{nillable}
PersonSkinTone [0..*]
PersonSpeechDescriptionText : Text [0..*]{nillable}
PersonSSNIdentification : Identification [0..*]{nillable}
PersonStateIdentification : Identification [0..*]{nillable}
PersonTaxIdentification : Identification [0..*]{nillable}
PersonTooth : Tooth [0..*]{nillable}
PersonUSCitizenIndicator : Boolean [0..*]{nillable}
PersonVisionPrescriptionText : Text [0..*]{nillable}
PersonWeightDescriptionText : Text [0..*]{nillable}
PersonWeightMeasure : WeightMeasure [0..*]{nillable}
PersonXRayImage : Image [0..*]{nillable}
PersonNationality [0..*]

Figure A.14 - Reuse of Person Class

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.



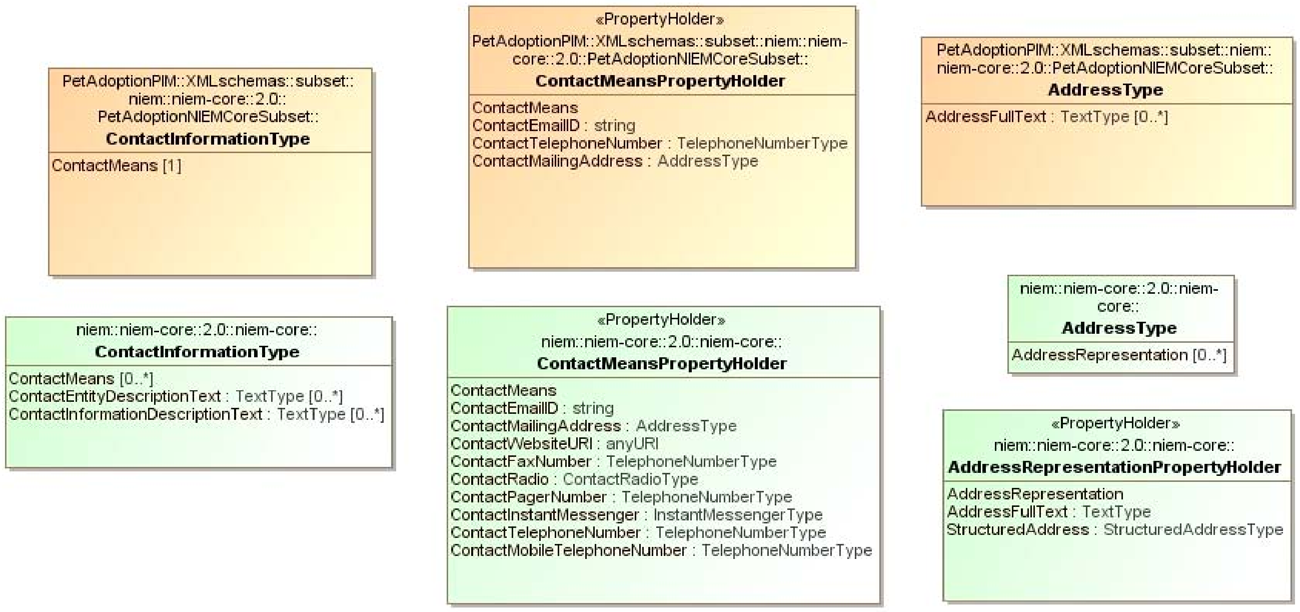
PersonName
(PetAdoptionPIM.PetAdoptionNIEMCoreSubset)
PersonNamePrefixText : Text [0..1]{nillable}
PersonGivenName : PersonNameText [0..1]{nillable}
PersonMiddleName : PersonNameText [0..1]{nillable}
PersonSurName : PersonNameText [0..1]{nillable}
PersonNameSuffixText : Text [0..1]{nillable}
PersonMaidenName : PersonNameText [0..1]{nillable}
PersonFullName : PersonNameText [1]{nillable}
«References»
PersonName
(NIEM Reference Model.niem.niem-core.2.0.niem-core)
PersonNamePrefixText : Text [0..*]{nillable}
PersonGivenName : PersonNameText [0..*]{nillable}
PersonMiddleName : PersonNameText [0..*]{nillable}
PersonSurName : PersonNameText [0..*]{nillable}
PersonNameSuffixText : Text [0..*]{nillable}
PersonMaidenName : PersonNameText [0..*]{nillable}
PersonFullName : PersonNameText [0..*]{nillable}
personNameCommentText : String [0..1]

Figure A.15 - 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).

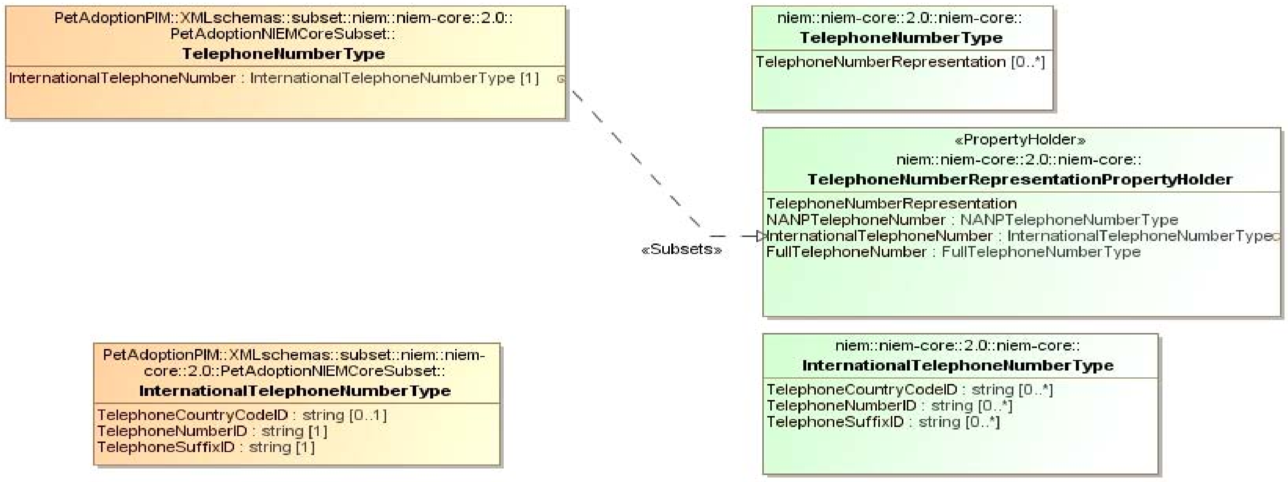
There are two considerations for contact information with respect to Person - defining the contact information as well as creating an association between the contact information and person. First let's look at our definition of contact information; you should understand most of this picture now in Figure A.16. We will also leave off the implied <<Subsets>> as you now understand the relationship between the subset and NIEM-Core.



PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ContactInformationType
ContactMeans [1]
«PropertyHolder»
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ContactMeansPropertyHolder
ContactMeans
ContactEmailID : string
ContactTelephoneNumber : TelephoneNumberType
ContactMailingAddress : AddressType
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
AddressType
AddressFullText : TextType [0..*]
niem::niem-core::2.0::niem-core::
ContactInformationType
ContactMeans [0..*]
ContactEntityDescriptionText : TextType [0..*]
ContactInformationDescriptionText : TextType [0..*]
«PropertyHolder»
niem::niem-core::2.0::niem-core::
ContactMeansPropertyHolder
ContactMeans
ContactEmailID : string
ContactMailingAddress : AddressType
ContactWebsiteURI : anyURI
ContactFaxNumber : TelephoneNumberType
ContactRadio : ContactRadioType
ContactPagerNumber : TelephoneNumberType
ContactInstantMessenger : InstantMessengerType
ContactTelephoneNumber : TelephoneNumberType
ContactMobileTelephoneNumber : TelephoneNumberType
niem::niem-core::2.0::niem-core::
AddressType
AddressRepresentation [0..*]
«PropertyHolder»
niem::niem-core::2.0::niem-core::
AddressRepresentationPropertyHolder
AddressRepresentation
AddressFullText : TextType
StructuredAddress : StructuredAddressType

Figure A.16 - Referencing Contact Information

As you can see in Figure A.16, this is a very general idea of contact information that can have any number of various kinds of contacts. Note that “ContactInformation” subsets “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, ContactTelephoneNumber, 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.17 we will just show you this model fragment as it uses the same pattern of referencing.



PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
TelephoneNumberType
InternationalTelephoneNumber : InternationalTelephoneNumberType [1]
niem::niem-core::2.0::niem-core::
TelephoneNumberType
TelephoneNumberRepresentation [0..*]
«PropertyHolder»
niem::niem-core::2.0::niem-core::
TelephoneNumberRepresentationPropertyHolder
TelephoneNumberRepresentation
NANPTelephoneNumber : NANPTelephoneNumberType
InternationalTelephoneNumber : InternationalTelephoneNumberType
FullTelephoneNumber : FullTelephoneNumberType
«Subsets»
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
InternationalTelephoneNumberType
TelephoneCountryCodeID : string [0..1]
TelephoneNumberID : string [1]
TelephoneSuffixID : string [1]
niem::niem-core::2.0::niem-core::
InternationalTelephoneNumberType
TelephoneCountryCodeID : string [0..*]
TelephoneNumberID : string [0..*]
TelephoneSuffixID : string [0..*]

Figure A.17 - Telephone Number

A.15 Augmenting Telephone Number

Consider that we may want some additional information about telephone numbers and want 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.18 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 by using a “type=” in the XML instance.

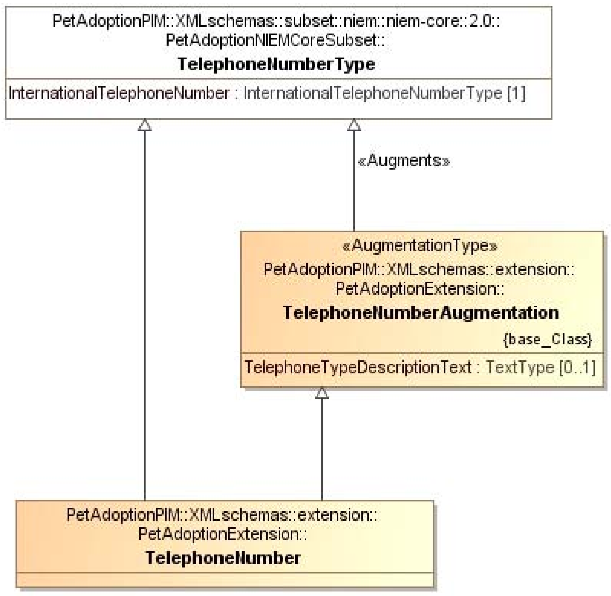

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::
PetAdoptionNIEMCoreSubset::
TelephoneNumberType
InternationalTelephoneNumber : InternationalTelephoneNumberType [1]
«Augments»
«AugmentationType»
PetAdoptionPIM::XMLschemas::extension::
PetAdoptionExtension::
TelephoneNumberAugmentation
{base_Class}
TelephoneTypeDescriptionText : TextType [0..1]
PetAdoptionPIM::XMLschemas::extension::
PetAdoptionExtension::
TelephoneNumber

Figure A.18 - Augmenting Telephone Number

A.16 Using a NIEM Association for Contact Information

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 Figure A.19 (which is still a subset of all the information) we put more of the pieces together and see a NIEM association "PersonContactInformationAssociationType" that is defined in NIEM core and then reused in our example. This NIEM association 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.

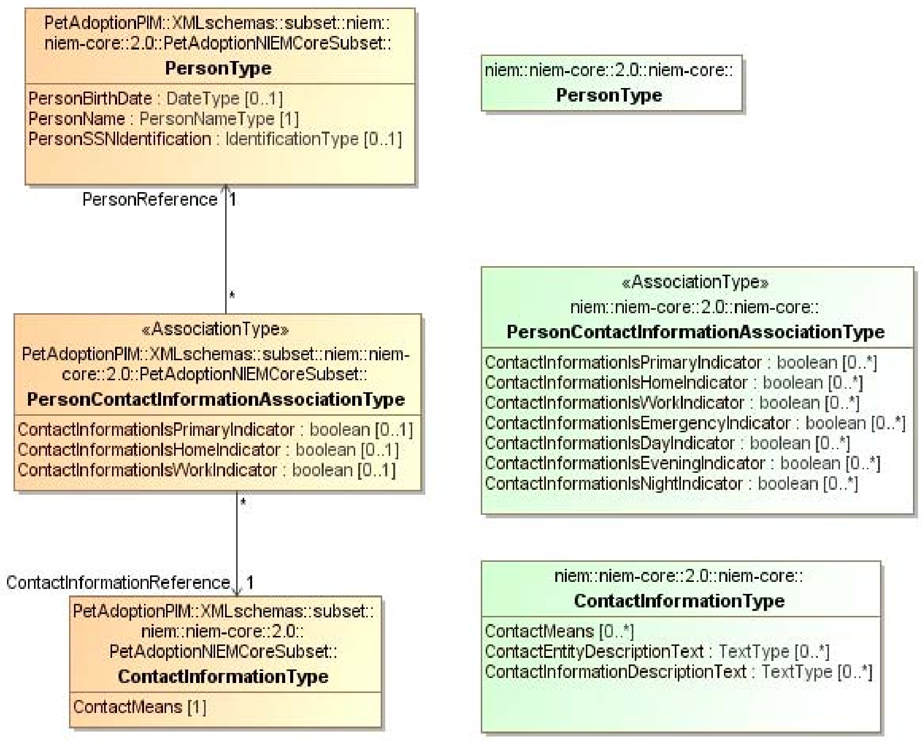

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
PersonType
PersonBirthDate : DateType [0..1]
PersonName : PersonNameType [1]
PersonSSNIdentification : IdentificationType [0..1]
PersonReference
1
*
niem::niem-core::2.0::niem-core::
PersonType
«AssociationType»
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
PersonContactInformationAssociationType
ContactInformationIsPrimaryIndicator : boolean [0..1]
ContactInformationIsHomeIndicator : boolean [0..1]
ContactInformationIsWorkIndicator : boolean [0..1]
«AssociationType»
niem::niem-core::2.0::niem-core::
PersonContactInformationAssociationType
ContactInformationIsPrimaryIndicator : boolean [0..*]
ContactInformationIsHomeIndicator : boolean [0..*]
ContactInformationIsWorkIndicator : boolean [0..*]
ContactInformationIsEmergencyIndicator : boolean [0..*]
ContactInformationIsDayIndicator : boolean [0..*]
ContactInformationIsEveningIndicator : boolean [0..*]
ContactInformationIsNightIndicator : boolean [0..*]
*
ContactInformationReference
1
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ContactInformationType
ContactMeans [1]
niem::niem-core::2.0::niem-core::
ContactInformationType
ContactMeans [0..*]
ContactEntityDescriptionText : TextType [0..*]
ContactInformationDescriptionText : TextType [0..*]

Figure A.19 - Contact Information Association

A.17 Pet Adoptions as a Kind of Activity

There are 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.

This pattern of reuse in Figure A.20 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.

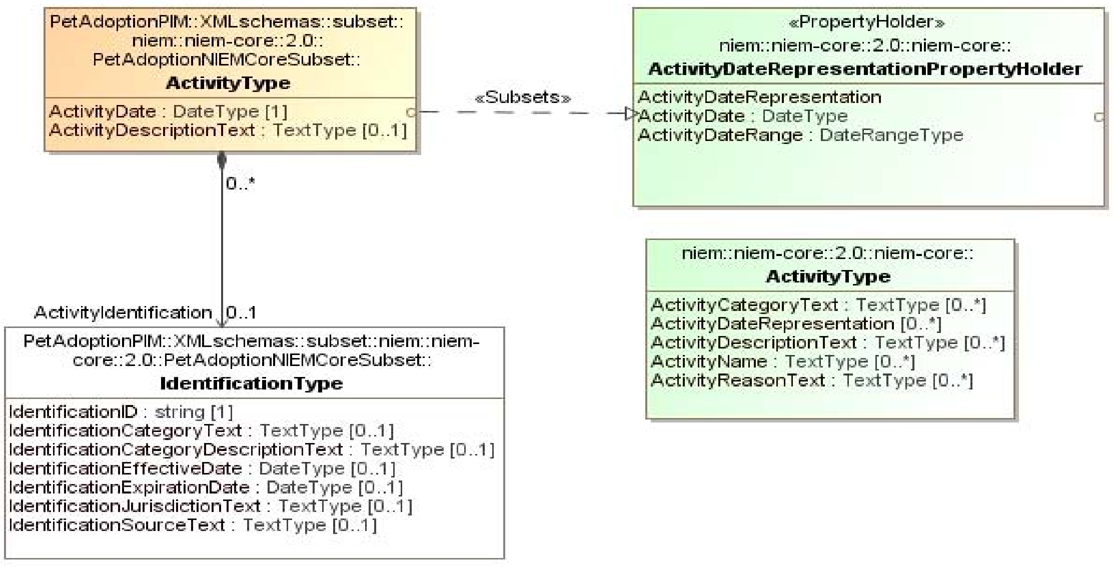

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ActivityType
ActivityDate : DateType [1]
ActivityDescriptionText : TextType [0..1]
«Subsets»
«PropertyHolder»
niem::niem-core::2.0::niem-core::
ActivityDateRepresentationPropertyHolder
ActivityDateRepresentation
ActivityDate : DateType
ActivityDateRange : DateRangeType
0..*
ActivityIdentification
0..1
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
IdentificationType
IdentificationID : string [1]
IdentificationCategoryText : TextType [0..1]
IdentificationCategoryDescriptionText : TextType [0..1]
IdentificationEffectiveDate : DateType [0..1]
IdentificationExpirationDate : DateType [0..1]
IdentificationJurisdictionText : TextType [0..1]
IdentificationSourceText : TextType [0..1]
niem::niem-core::2.0::niem-core::
ActivityType
ActivityCategoryText : TextType [0..*]
ActivityDateRepresentation [0..*]
ActivityDescriptionText : TextType [0..*]
ActivityName : TextType [0..*]
ActivityReasonText : TextType [0..*]

Figure A.20 - Defining Activity from NIEM-Core

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

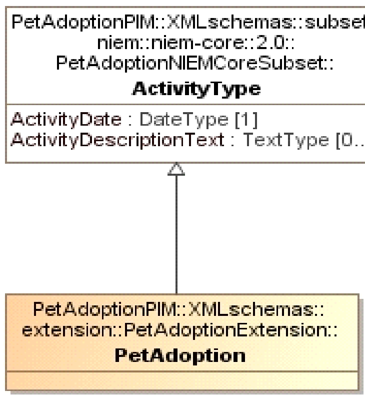

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ActivityType
ActivityDate : DateType [1]
ActivityDescriptionText : TextType [0..1]
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoption

Figure A.21 - PetAdoption as a kind of 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 sub clause 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

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 that we can reuse. Figure A.22 uses the same «Subsets» 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.

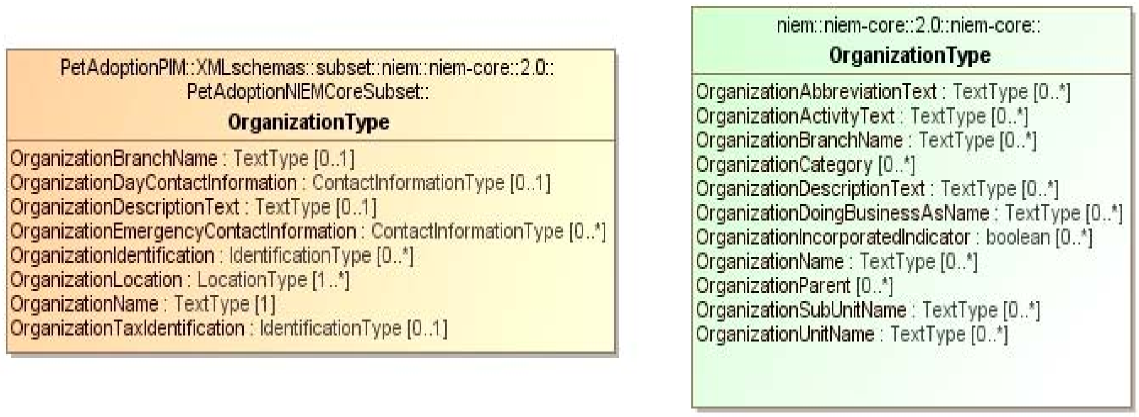

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::
PetAdoptionNIEMCoreSubset::
OrganizationType
OrganizationBranchName : TextType [0..1]
OrganizationDayContactInformation : ContactInformationType [0..1]
OrganizationDescriptionText : TextType [0..1]
OrganizationEmergencyContactInformation : ContactInformationType [0..*]
OrganizationIdentification : IdentificationType [0..*]
OrganizationLocation : LocationType [1..*]
OrganizationName : TextType [1]
OrganizationTaxIdentification : IdentificationType [0..1]
niem::niem-core::2.0::niem-core::
OrganizationType
OrganizationAbbreviationText : TextType [0..*]
OrganizationActivityText : TextType [0..*]
OrganizationBranchName : TextType [0..*]
OrganizationCategory [0..*]
OrganizationDescriptionText : TextType [0..*]
OrganizationDoingBusinessAsName : TextType [0..*]
OrganizationIncorporatedIndicator : boolean [0..*]
OrganizationName : TextType [0..*]
OrganizationParent [0..*]
OrganizationSubUnitName : TextType [0..*]
OrganizationUnitName : TextType [0..*]

Figure A.22 - Using Organization from Core

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.23 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).

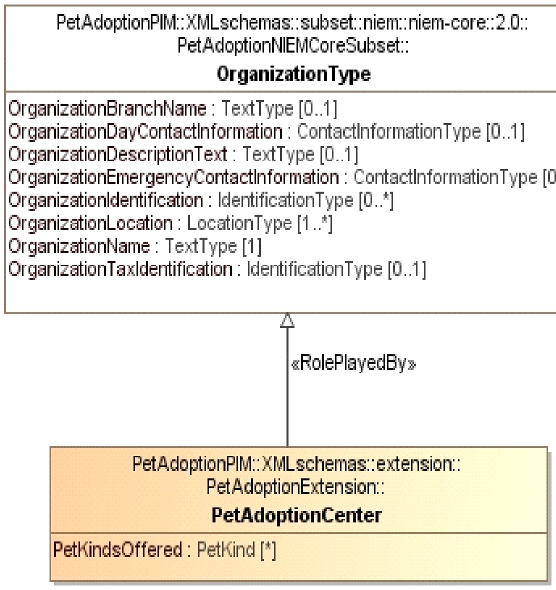
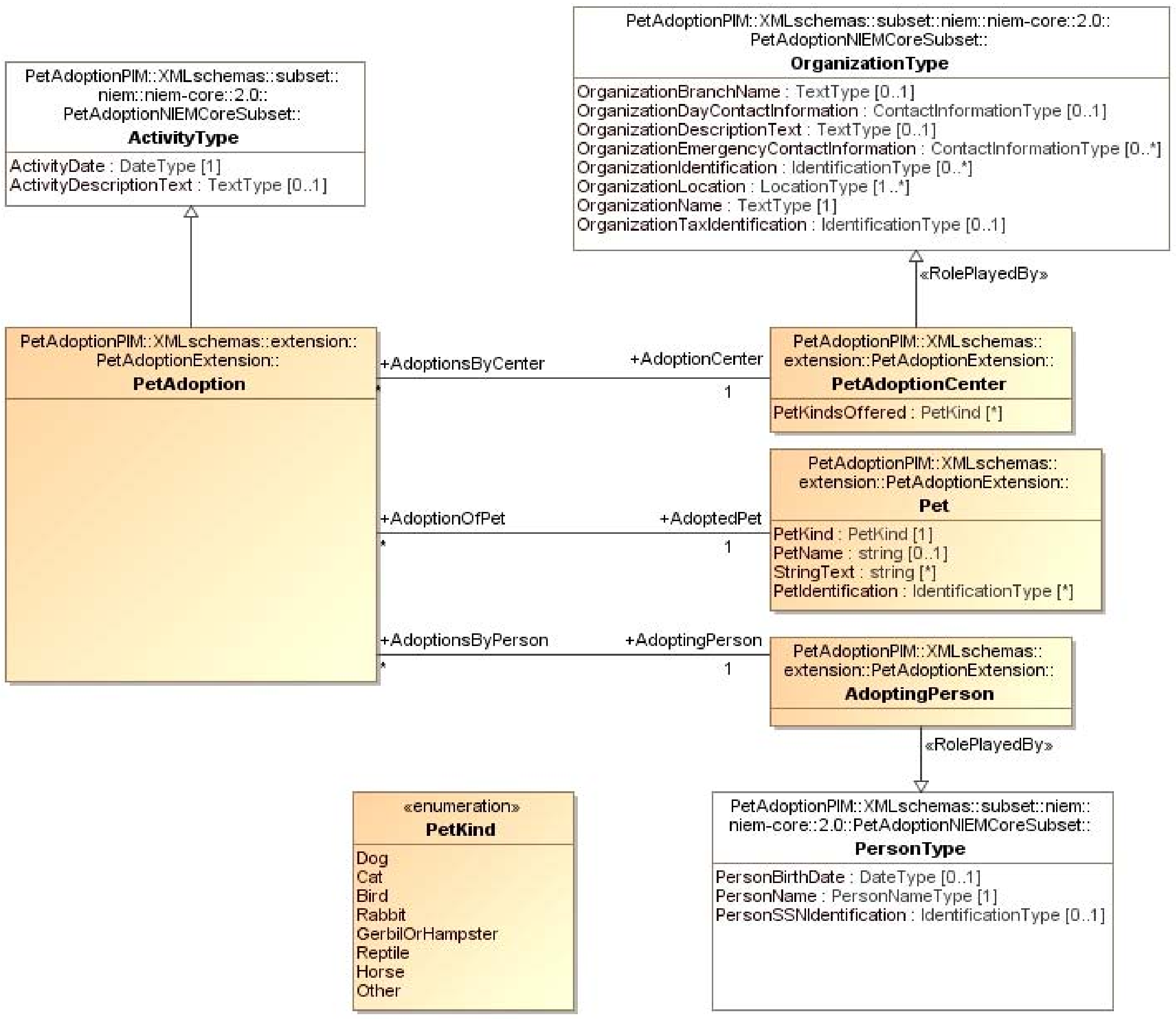

PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::
PetAdoptionNIEMCoreSubset::
OrganizationType
OrganizationBranchName : TextType [0..1]
OrganizationDayContactInformation : ContactInformationType [0..1]
OrganizationDescriptionText : TextType [0..1]
OrganizationEmergencyContactInformation : ContactInformationType [0..*]
OrganizationIdentification : IdentificationType [0..*]
OrganizationLocation : LocationType [1..*]
OrganizationName : TextType [1]
OrganizationTaxIdentification : IdentificationType [0..1]
«RolePlayedBy»
PetAdoptionPIM::XMLschemas::extension::
PetAdoptionExtension::
PetAdoptionCenter
PetKindsOffered : PetKind [*]

Figure A.23 - Adoption Centers as a Role of an Organization

A.19 Putting Together the 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.24, 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.



PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
ActivityType
ActivityDate : DateType [1]
ActivityDescriptionText : TextType [0..1]
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
OrganizationType
OrganizationBranchName : TextType [0..1]
OrganizationDayContactInformation : ContactInformationType [0..1]
OrganizationDescriptionText : TextType [0..1]
OrganizationEmergencyContactInformation : ContactInformationType [0..*]
OrganizationIdentification : IdentificationType [0..*]
OrganizationLocation : LocationType [1..*]
OrganizationName : TextType [1]
OrganizationTaxIdentification : IdentificationType [0..1]
«RolePlayedBy»
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoption
+AdoptionsByCenter
*
+AdoptionCenter
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoptionCenter
PetKindsOffered : PetKind [*]
+AdoptionOfPet
*
+AdoptedPet
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
Pet
PetKind : PetKind [1]
PetName : string [0..1]
StringText : string [*]
PetIdentification : IdentificationType [*]
+AdoptionsByPerson
*
+AdoptingPerson
1
PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
AdoptingPerson
«RolePlayedBy»
«enumeration»
PetKind
Dog
Cat
Bird
Rabbit
GerbilOrHampster
Reptile
Horse
Other
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
PersonType
PersonBirthDate : DateType [0..1]
PersonName : PersonNameType [1]
PersonSSNIdentification : IdentificationType [0..1]

Figure A.24 - Completed High Level Picture

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.

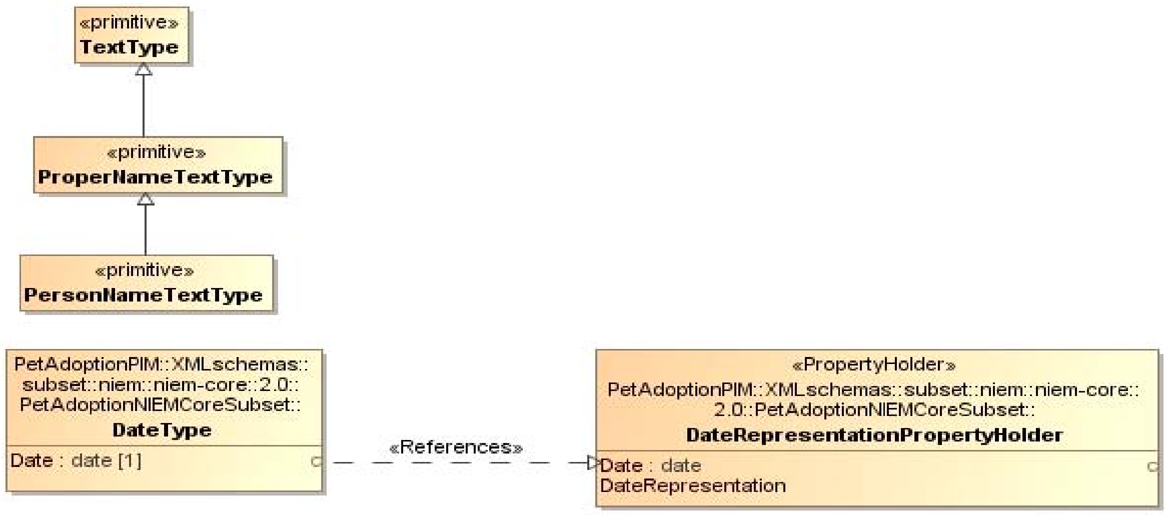

«primitive»
TextType
«primitive»
ProperNameTextType
«primitive»
PersonNameTextType
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
DateType
Date : date [1]
«References»
«PropertyHolder»
PetAdoptionPIM::XMLschemas::subset::niem::niem-core::2.0::PetAdoptionNIEMCoreSubset::
DateRepresentationPropertyHolder
Date : date
DateRepresentation

Figure A.25 - Primitive Types

The above are shown for clarity - the primitive types are included automatically. Date type does require some attention, to establish the date representation required. Any type that is referenced is automatically 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.

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 "PetAdoptionExchangeType" in Figure A.26 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.

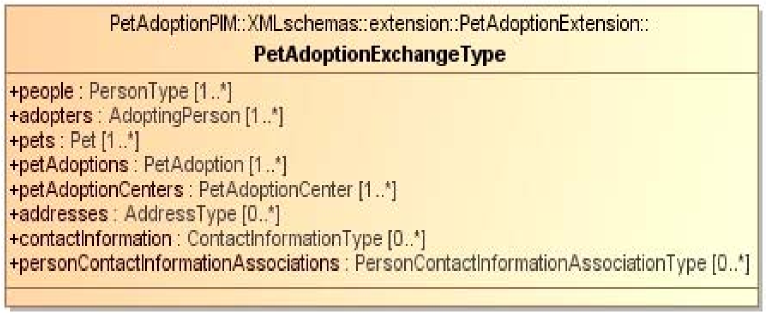

PetAdoptionPIM::XMLschemas::extension::PetAdoptionExtension::
PetAdoptionExchangeType
+people : PersonType [1..*]
+adopters : AdoptingPerson [1..*]
+pets : Pet [1..*]
+petAdoptions : PetAdoption [1..*]
+petAdoptionCenters : PetAdoptionCenter [1..*]
+addresses : AddressType [0..*]
+contactInformation : ContactInformationType [0..*]
+personContactInformationAssociations : PersonContactInformationAssociationType [0..*]

Figure A.26 - Pet Adoption Exchange Type

For each such exchange type we need to add a property to the "Exchange Schema," this is done by just making one property for each exchange type in a property holder.

Our exchange schema looks like this:

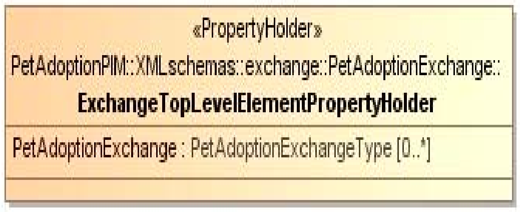

«PropertyHolder»
PetAdoptionPIM::XMLschemas::exchange::PetAdoptionExchange::
ExchangeTopLevelElementPropertyHolder
PetAdoptionExchange : PetAdoptionExchangeType [0..*]

Figure A.27 - Exchange Schema

A.22 Using Classes by Default

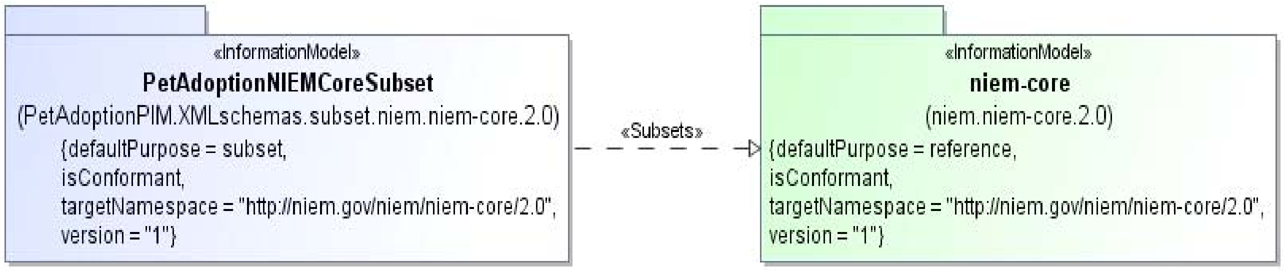

«InformationModel»
PetAdoptionNIEMCoreSubset
(PetAdoptionPIM.XMLschemas.subset.niem.niem-core.2.0)
{defaultPurpose = subset,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}
«Subsets»
«InformationModel»
niem-core
(niem.niem-core.2.0)
{defaultPurpose = reference,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}

Figure A.28 - Referencing NIEM Core

Figure A.28 shows that the entire “PetAdoptionNIEMCoreSubset” model «subsets» the “niem-core” model. By referencing the entire package any NEIM-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 automatically be redefined to reference the corresponding element in PetAdoptionNIEMCoreSubset. It is good practice to inspect the types and properties automatically included to make sure you are not including more than is necessary - this may be done in the PSM model or some tools may provide features for doing so.

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 itself. 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.29 is the IEPD model for our example.

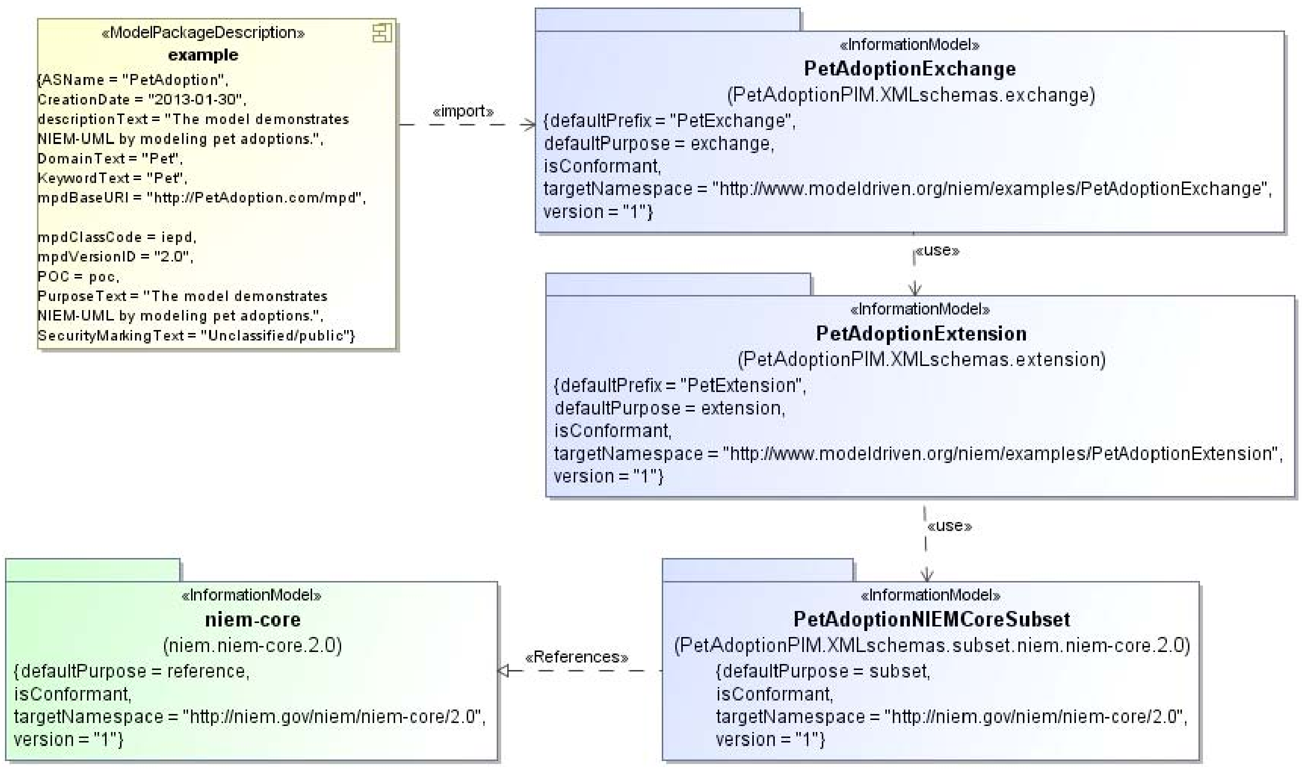

«ModelPackageDescription»
example
{ASName = "PetAdoption",
CreationDate = "2013-01-30",
descriptionText = "The model demonstrates NIEM-UML by modeling pet adoptions.",
DomainText = "Pet",
KeywordText = "Pet",
mpdBaseURI = "http://PetAdoption.com/mpd",
mpdClassCode = iepd,
mpdVersionID = "2.0",
POC = poc,
PurposeText = "The model demonstrates NIEM-UML by modeling pet adoptions.",
SecurityMarkingText = "Unclassified/public"}
«import»
«InformationModel»
PetAdoptionExchange
(PetAdoptionPIM.XMLschemas.exchange)
{defaultPrefix = "PetExchange",
defaultPurpose = exchange,
isConformant,
targetNamespace = "http://www.modeldriven.org/niem/examples/PetAdoptionExchange",
version = "1"}
«use»
«InformationModel»
PetAdoptionExtension
(PetAdoptionPIM.XMLschemas.extension)
{defaultPrefix = "PetExtension",
defaultPurpose = extension,
isConformant,
targetNamespace = "http://www.modeldriven.org/niem/examples/PetAdoptionExtension",
version = "1"}
«use»
«InformationModel»
niem-core
(niem.niem-core.2.0)
{defaultPurpose = reference,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}
«References»
«InformationModel»
PetAdoptionNIEMCoreSubset
(PetAdoptionPIM.XMLschemas.subset.niem.niem-core.2.0)
{defaultPurpose = subset,
isConformant,
targetNamespace = "http://niem.gov/niem/niem-core/2.0",
version = "1"}

Figure A.29 - 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 «subsets» and configures those concepts for our use in pet adoption.

Note that each information model has a defaultPurpose that corresponds to the NIEM schema types. This purpose will be used unless that purpose is overridden using the «ModelPackageDescriptionFile» 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
Structured English Mapping Specifications

(normative)

B.1 Mapping Between a NIEM-conforming Platform Specific Model and an MPD Catalog XML Instance

B.1.1 Scope

The following sub clause describes a mapping between a NIEM-conforming Platform Specific Model and an MPD Catalog XML Instance.

B.1.2 Conventions

This sub clause employs the following conventions.

[Convention: To reference a specification]

If the sub clause references a specification, the sub clause represents the reference as follows:

1. the prefix "[Reference:"
2. the title of the specification
3. the character "]" and on the following line
4. the word to indicate the specification and the URI for the specification indented from the "[Reference:" prefix

If this sub clause were to reference the specification with the title "XML Information Set (Second Edition)" and the URI "http://www.w3.org/TR/xml-infoset," and if it were to associate the specification with the word "infoset," the sub clause would represent the reference as follows:

[Reference: XML Information Set (Second Edition)]

The word "infoset" indicates a term from this specification.

http://www.w3.org/TR/xml-infoset

[Convention: To indicate a term from another specification]

If the sub clause employs a term from another specification, or a term derived from another specification, the sub clause represents the term as follows:

1. the character "{"
2. a word to indicate the specification
3. the character ":"
4. the term itself and
5. the character "}"

If this sub clause were to employ the term “attribute information item” from the specification “XML Information Set (Second Edition),” and presuming the word to indicate the specification is “infoset,” the representation of that term would be {infoset:attribute}.

[Convention: To indicate a rule]

If this sub clause specifies a rule, the sub clause represents the rule as follows:

1. the prefix “[Rule ”
2. the character “:”
3. the rule name
4. the character “]” and on the following line
5. the rule itself indented from the “[Rule ” prefix

If this sub clause were to specify a rule for which the name is “Example Rule” and for which rule itself is “This is an example rule.”, the sub clause would represent the definition as follows:

[Rule: Example Rule]

This is an example rule.

B.1.3 References

This sub clause references the following specifications.

[Reference: OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.4.1]

The word “uml” indicates a term from this specification. Example: {uml:Class}

http://www.omg.org/spec/UML/2.4/Superstructure

[Reference: Namespaces in XML 1.0 (Third Edition)]

The word “namespace” indicates a term from this specification. Example: {namespace:qualified name}

http://www.w3.org/TR/xml-names/

[Reference: XML Information Set (Second Edition)]

The word “infoset” indicates a term from this specification. Example: {infoset:attribute}

http://www.w3.org/TR/xml-infoset

[Reference: NIEM Model Package Description Specification, Version 1.0]

The word “mpd” indicates a term from this specification. Example: {mpd:artifact}

http://reference.niem.gov/niem/specification/model-package-description/1.0/model-package-description-1.0.pdf

B.1.4 Terminology

B.1.4.1 {uml} Terminology

The following definitions derive from {uml}.

[Definition: *attribute-name* attribute of the *{uml:metaclass}*]

This phrase refers to the attribute of the *{uml:metaclass}* with the name *attribute-name*. Example: the *name* attribute of the *{uml:Property}*

[Definition: *end-name {uml:end-type}* of the *{uml:metaclass}*]

This phrase refers to the opposite association end of the *{uml:metaclass}* with the name *end-name,* the type of which is *{uml:end-type}*. Example: the *subsettedProperty* of the *{uml:Property}*

[Definition: *end-name* of the *{uml:metaclass}*]

This phrase refers to the opposite association end of the *metaclass* with the name *end-name*. Unlike the above definition, the type of the association end is not specified. Example: the type of the {uml:Property}

B.1.4.2 {infoset} Terminology

The following definitions derive from {infoset}.

[Definition: {infoset:element}]

Same as {infoset:element information item}

[Definition: {infoset:attribute}]

Same as {infoset:attribute information item}

[Definition: "*prefix*:*local-name*" {infoset:attribute}]

This phrase references an {infoset:attribute}

1. for which the value of the namespace name property is the namespace name associated by this sub clause with the namespace prefix "*prefix*",
2. and for which the value of the local name property is "*local-name*".

[Definition: "*prefix*:*local-name*" {infoset:element}]

This phrase references an {infoset:element}

1. for which the value of the namespace name property is the namespace name associated by this sub clause with the namespace prefix "*prefix*",
2. and for which the value of the local name property is "*local-name*".

[Definition: content of the "prefix:local-name" {infoset:element}]

This phrase references the character information items among the children property of the referenced {infoset:element}.

B.1.4.3 {mpd} Terminology

The following definitions derive from {infoset}.

[Definition: catalog namespace]

The catalog namespace is the namespace for which the namespace name is "http://reference.niem.gov/niem/resource/mpd/catalog/1.0/." This sub clause associates the namespace name with the namespace prefix "ca."

B.1.5 Mapping

B.1.5.1 Mapping ca:Catalog {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescription} and a ca:Catalog {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescription} and a ca:Catalog {infoset:element} if and only if the following are true.

1. Given the {infoset:attribute}s among the attributes of the ca:Catalog {infoset:element}
   a. The value of the mpdVersionID attribute of the {stereotype:ModelPackageDescription} must equal the normalized value of the ca:mpdVersionID {infoset:attribute}.
   b. The concatenation of the value of the mpdBaseURI and the value of the mpdVersionID attributes of the {stereotype:ModelPackageDescription} must equal the normalized value of the ca:mpdURI {infoset:attribute}.
   c. The value of the mpdClassCode attribute of the {stereotype:ModelPackageDescription} must equal the normalized value of the ca:mpdClassCode {infoset:attribute}.
   d. Exactly one of the following must be true:
      i. the value of the descriptionText attribute of the {stereotype:ModelPackageDescription} must be present and must equal the normalized value of the ca:descriptionText {infoset:attribute}; or
      ii. the value of the descriptionText attribute of the {stereotype:ModelPackageDescription} must be absent and the ca:descriptionText {infoset:attribute} must be absent.
2. For each {stereotype:ModelPackageDescriptionFile} for which the {stereotype:ModelPackageDescription} is a client {uml:NamedElement}, a mapping must exist between the {stereotype:ModelPackageDescriptionFile} and a ca:File {infoset:element} among the children of the ca:Catalog {infoset:element}.
3. For each {uml:Usage} for which the {stereotype:ModelPackageDescription} is a client {uml:NamedElement}, a mapping must exist between the supplier {uml:NamedElement} of the {uml:Usage} and a ca:FileSet {infoset:element} among the children of the ca:Catalog {infoset:element}.
4. A mapping must exist between the {stereotype:ModelPackageDescription} and the ca:Metadata {infoset:element} among the children of the ca:Catalog {infoset:element}.

B.1.5.2 Mapping ca:Metadata {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescription} and a ca:Metadata {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescription} and a ca:Metadata {infoset:element} only if the following are true.

1. The value of the SecurityMarkingText attribute of the {stereotype:ModelPackageDescription} must equal the content of the ca:SecurityMarkingText {infoset:element} among the children of the ca:Metadata {infoset:element}.
2. The value of the CreationDate attribute of the {stereotype:ModelPackageDescription} must equal the content of the ca:CreationDate {infoset:element} among the children of the ca:Metadata {infoset:element}.
3. Exactly one of the following must be true:
   a. the LastRevisionDate attribute of the {stereotype:ModelPackageDescription} must be present and the value of the attribute must equal the content of the ca:LastRevisionDate {infoset:element} among the children of the ca:Metadata {infoset:element}; or

   b. the LastRevisionDate attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:LastRevisionDate {infoset:element} must be absent from the children of the ca:Metadata {infoset:element}.

4. Exactly one of the following must be true:
   a. the NextRevisionDate attribute of the {stereotype:ModelPackageDescription} must be present and the value of the attribute must equal the content of the ca:NextRevisionDate {infoset:element} among the children of the ca:Metadata {infoset:element}; or
   b. the NextRevisionDate attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:NextRevisionDate {infoset:element} must be absent from the children of the ca:Metadata {infoset:element}.
5. Exactly one of the following must be true:
   a. The StatusText attribute of the {stereotype:ModelPackageDescription} must be present and the value of the attribute must equal the content of the ca:StatusText {infoset:element} among the children of the ca:Metadata {infoset:element}; or
   b. the StatusText attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:StatusText {infoset:element} must be absent from the children of the ca:Metadata {infoset:element}.
6. For each value of the KeywordText attribute of the {stereotype:ModelPackageDescription}, a ca:KeywordText {infoset:element} must exist among the children of the ca:Metadata {infoset:element}, and the content of the ca:KeywordText {infoset:element} must equal that value of the KeywordText attribute of the {stereotype:ModelPackageDescription}.
7. For each value of the DomainText attribute of the {stereotype:ModelPackageDescription}, a ca:DomainText {infoset:element} must exist among the children of the ca:Metadata {infoset:element}, and the content of the ca:DomainText {infoset:element} must equal that value of the DomainText attribute of the {stereotype:ModelPackageDescription}.
8. For each value of the PurposeText attribute of the {stereotype:ModelPackageDescription}, a ca:PurposeText {infoset:element} must exist among the children of the ca:Metadata {infoset:element}, and the content of the ca:PurposeText {infoset:element} must equal that value of the PurposeText attribute of the {stereotype:ModelPackageDescription}.
9. For each value of the ExchangePatternText attribute of the {stereotype:ModelPackageDescription}, a ca:ExchangePatternText {infoset:element} must exist among the children of the ca:Metadata {infoset:element}, and the content of the ca:ExchangePatternText {infoset:element} must equal that value of the ExchangePatternText attribute of the {stereotype:ModelPackageDescription}.
10. For each {stereotype:ModelPackageDescriptionRelationship} for which the {stereotype:ModelPackageDescription} is the client {uml:NamedElement}, a mapping must exist between the {stereotype:ModelPackageDescriptionRelationship} and a ca:Relationship {infoset:element} among the children of the ca:Metadata {infoset:element}.
11. A mapping must exist between the {stereotype:ModelPackageDescription} and the ca:AuthoritativeSource {infoset:element} among the children of the ca:Metadata {infoset:element}.

B.1.5.3 Mapping ca:AuthoritativeSource {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescription} and a ca:AuthoritativeSource {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescription} and a ca:AuthoritativeSource {infoset:element} only if the following are true.

1. The value of the ASName attribute of the {stereotype:ModelPackageDescription} must equal the content of the ca:ASName {infoset:element} among the children of the ca:AuthoritativeSource {infoset:element}.
2. Exactly one of the following must be true:
   a. the ASAddressText attribute of the {stereotype:ModelPackageDescription} must be present and the value of the attribute must equal the content of the ca:ASAddressText {infoset:element} among the children of the ca:AuthoritativeSource {infoset:element}; or
   b. the ASAddressText attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:ASAddressText {infoset:element} must be absent from the children of the ca:AuthoritativeSource {infoset:element}.
3. Exactly one of the following must be true:
   a. The ASWebSiteURL attribute of the {stereotype:ModelPackageDescription} must be present and the value of the attribute must equal the content of the ca:ASWebSiteURL {infoset:element} among the children of the ca:AuthoritativeSource {infoset:element}; or
   b. the ASWebSiteURL attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:ASWebSiteURL {infoset:element} must be absent from the children of the ca:AuthoritativeSource {infoset:element}.
4. For each value of the POC attribute of the {stereotype:ModelPackageDescription}, a mapping must exist between an instance of a POCType and a ca:POC {infoset:element} among the children of the ca:AuthoritativeSource {infoset:element}.

B.1.5.4 Mapping ca:POC {infoset:element}

[Rule: Mapping between for an instance of POCType and a ca:POC {infoset:element}]

A mapping shall exist between an instance of POCType and a ca:POC {infoset:element} only if the following are true.

1. The value of the POCName attribute of the instance of POCType must equal the content of the ca:POCName {infoset:element} among the children of the ca:POC {infoset:element}.
2. For each value of the POCEmail attribute of the instance of POCType, a ca:POCEmail {infoset:element} must exist among the children of the ca:POC {infoset:element} and the content of the ca:POCEmail {infoset:element} must equal the value of that POCEmail attribute of the instance of POCType.
3. For each value of the POCTelephone attribute of the instance of POCType, a ca:POCTelephone {infoset:element} must exist among the children of the ca:POC {infoset:element} and the content of the ca:POCTelephone {infoset:element} must equal the value of that POCTelephone attribute of the instance of POCType.

B.1.5.5 Mapping ca:Relationship {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescriptionRelationship} and a ca:Relationshop {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescriptionRelationship} and a ca:Relationship {infoset:element} only if the following are true.

1. The value of the relationshipCode attribute of the {stereotype:ModelPackageDescriptionRelationship} must equal the normalized value of the ca:relationshipCode {infoset:attribute} of the ca:Relationship {infoset:element}.
2. Exactly one of the following must be true:

   a. the value of the descriptionText attribute of the {stereotype:ModelPackageDescriptionRelationship} must be present the value of the attribute and must equal the normalized value of the ca:descriptionText {infoset:attribute} of the ca:Relationship {infoset:element}; or
   b. the value of the descriptionText attribute of the {stereotype:ModelPackageDescription} must be absent, and the ca:descriptionText {infoset:attribute} of the ca:Relationship {infoset:element} must be absent.
3. The value of the mpdURI attribute of the {stereotype:ModelPackageDescription} that is the supplier {uml:NamedElement} of the {uml:ModelPackageDescriptionRelationship} must equal the normalized value of the ca:resourceURI {infoset:attribute} of the Relationship {infoset:element}.

B.1.5.6 Mapping ca:FileSet {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescriptionFileSet} and a ca:FileSet {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescriptionFileSet} and a ca:FileSet {infoset:element} only if the following are true.

1. Exactly one of the following must be true:
   a. the value of the externalURI attribute of the {stereotype:ModelPackageDescriptionFileSet} must be present and the value of the attribute must equal the normalized value of the ca:externalURI {infoset:attribute} of the ca:FileSet {infoset:element}; or
   b. the value of the externalURI attribute of the {stereotype:ModelPackageDescriptionFileSet} must be absent, and the ca:externalURI {infoset:attribute} of the ca:FileSet {infoset:element} must be absent.
2. The concatenation of “http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/nature#” and the value of the natureCode attribute of the {stereotype:ModelPackageDescriptionFileSet} must equal the normalized value of the ca:natureURI {infoset:attribute} of the ca:FileSet {infoset:element}.
3. The concatenation of “http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#” and the purposeCode attribute of the {stereotype:ModelPackageDescriptionFileSet} must equal the normalized value of the ca:purposeURI {infoset:attribute} of the ca:FileSet {infoset:element}.
4. Exactly one of the following must be true:
   a. the value of the descriptionText attribute of the {stereotype:ModelPackageDescriptionFileSet} must be present and the value of the attribute must equal the normalized value of the ca:descriptionText {infoset:attribute} of the ca:FileSet {infoset:element}; or
   b. the value of the descriptionText attribute of the {stereotype:ModelPackageDescriptionFileSet} must be absent, and the ca:descriptionText {infoset:attribute} of the ca:FileSet {infoset:element} must be absent.
5. For each {stereotype:ModelPackageDescriptionFile} for which the {stereotype:ModelPackageDescriptionFileSet} is a client {uml:NamedElement}, a mapping must exist between the {stereotype:ModelPackageDescriptionFile} and a ca:File {infoset:element} among the children of the ca:FileSet {infoset:element}.

B.1.5.7 Mapping ca:File {infoset:element}

[Rule: Mapping between a {stereotype:ModelPackageDescriptionFile} and a ca:File {infoset:element}]

A mapping shall exist between a {stereotype:ModelPackageDescriptionFile} and a ca:File {infoset:element} only if the following are true.

1. Exactly one of the following must be true:

   a. the value of the externalURI attribute of the {stereotype:ModelPackageDescriptionFile} must be present and the value of the attribute must equal the normalized value of the ca:externalURI {infoset:attribute} of the ca:File {infoset:element}; or
   b. the value of the externalURI attribute of the {stereotype:ModelPackageDescriptionFile} must be absent, and the ca:externalURI {infoset:attribute} of the ca:File {infoset:element} must be absent.
2. The value of the relativePathName attribute of the {stereotype:ModelPackageDescriptionFile} must equal the normalized value of the ca:relativePathName {infoset:attribute} of the ca:File {infoset:element}.
3. The concatenation of "http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/nature#" and the value of the natureCode attribute of the {stereotype:ModelPackageDescriptionFile} must equal the normalized value of the ca:natureURI {infoset:attribute} of the ca:File {infoset:element}.
4. The concatenation of "http://reference.niem.gov/niem/resource/mpd/lexicon/1.0/purpose#" and the purposeCode attribute of the {stereotype:ModelPackageDescriptionFile} must equal the normalized value of the ca:purposeURI {infoset:attribute} of the ca:File {infoset:element}.
5. Exactly one of the following must be true:
   a. the value of the descriptionText attribute of the {stereotype:ModelPackageDescriptionFile} must be present and the value of the attribute must equal the normalized value of the ca:descriptionText {infoset:attribute} of the ca:File {infoset:element}; or
   b. the value of the descriptionText attribute of the {stereotype:ModelPackageDescriptionFile} must be absent, and the ca:descriptionText {infoset:attribute} of the ca:File {infoset:element} must be absent.

B.2 Mapping Between NIEM-conformant XML Schema and NIEM-conforming Platform Specific Model

B.2.1 Scope

This sub clause specifies the mapping between a NIEM-conformant XML Schema and a NIEM-conforming Platform Specific Model.

B.2.2 Conventions

This sub clause employs the following conventions.

[Convention: To reference a specification]

If the sub clause references a specification, the sub clause represents the reference as follows:

1. the prefix "[Reference: "
2. the title of the specification
3. the character "]", and on the following line
4. the word to indicate the specification and the URI for the specification indented from the "[Reference:" prefix

If this sub clause were to reference the specification with the title "XML Information Set (Second Edition)" and the URI "http://www.w3.org/TR/xml-infoset," and if it were to associate the specification with the word "infoset," the sub clause would represent the reference as follows.

[Reference: XML Information Set (Second Edition)]

The word “infoset” indicates a term from this specification.

http://www.w3.org/TR/xml-infoset

[Convention: To indicate a term from another specification]

If the sub clause employs a term from another specification, or a term derived from another specification, the sub clause represents the term as follows:

1. the character “{”
2. a word to indicate the specification
3. the character “:”
4. the term itself and
5. the character “}”

If this sub clause were to employ the term “attribute information item” from the specification “XML Information Set (Second Edition),” and presuming the word to indicate the specification is “infoset,” the representation of that term would be {infoset:attribute}.

This convention serves to distinguish between same term employed by different specifications for different purposes. For example, the term “component” occurs in both “XML Schema Part 1: Structures Second Edition” and “OMG Unified Modeling Language (TM) (OMG UML), Superstructure Version 2.4.1.” In the former, the term “component” refers to any schema component; in the latter, the term “Component” refers to a particular metaclass. Rather than leave the reader to infer the intended meaning from the context of the sentence, this sub clause distinguishes between the two: the former is {schema:component} and the latter is {uml:Component}.

[Convention: To indicate a definition]

If this sub clause defines a term, the sub clause represents the definition as follows:

1. the prefix “[Definition: ”
2. the term
3. the character “]” and on the following line
4. the definition itself indented from the “[Definition:” prefix

If the term includes *formatted text*, the use of the term will instantiate the *formatted text*.

If this sub clause were to define a term “Example Definition” for which the definition is “This is an example definition.”, the sub clause would represent the definition as follows.

[Definition: Example Definition]

This is an example definition.

[Convention: To indicate a rule]

If this sub clause specifies a rule, the sub clause represents the rule as follows:

1. the prefix “[Rule ”
2. the character “:”

3. the rule name
4. the character “]” and on the following line
5. the rule itself, indented from the “[Rule ” prefix

If this sub clause were to specify a rule for which the name is “Example Rule” and for which rule itself is “This is an example rule.”, the sub clause would represent the definition as follows:

[Rule: Example Rule]

This is an example rule.

B.2.3 References

This sub clause references the following specifications.

[Reference: OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.4.1]

The word “uml” indicates a term from this specification. Example: {uml:Class}

http://www.omg.org/spec/UML/2.4/Superstructure

[Reference: Namespaces in XML 1.0 (Third Edition)]

The word “namespace” indicates a term from this specification. Example: {namespace:qualified name}

http://www.w3.org/TR/xml-names/

[Reference: XML Information Set (Second Edition)]

The word “infoset” indicates a term from this specification. Example: {infoset:attribute}

http://www.w3.org/TR/xml-infoset

[Reference: XML Schema Part 1: Structures Second Edition]

The word “schema” indicates a term from this specification. Example: {schema:complex type definition}

http://www.w3.org/TR/xmlschema-1/

[Reference: XML Schema Part 2: Datatypes Second Edition]

The word “schema” also indicates a term from this specification. Example: {schema:simple type definition}

http://www.w3.org/TR/xmlschema-2/

[Reference: NIEM Naming and Design Rules Version 1.3]

The word “niem” indicates a term from this specification. Example: {niem:object type}

http://reference.niem.gov/niem/specification/naming-and-design-rules/1.3/niem-ndr-1.3.pdf

B.2.4 Terminology

B.2.4.1 {uml} Terminology

The following definitions derive from {uml}.

[Definition: *attribute-name* attribute of the *{uml:metaclass}*]

This phrase refers to the attribute of the *{uml:metaclass}* with the name *attribute-name*. Example: The *name* attribute of the *{uml:Property}*

[Definition: *end-name {uml:end-type}* of the *{uml:metaclass}*]

This phrase refers to the opposite association end of the *{uml:metaclass}* with the name *end-name,* the type of which is *{uml:end-type}*. Example: the *subsettedProperty* of the *{uml:Property}*

[Definition: *end-name* of the *{uml:metaclass}*]

This phrase refers to the opposite association end of the *metaclass* with the name *end-name*. Unlike the above definition, the type of the association end is not specified. Example: The type of the {uml:Property}

B.2.4.2 {infoset} Terminology

The following definitions derive from {infoset}.

[Definition: {infoset:element}]

Same as {infoset:element information item}

[Definition: {infoset:attribute}]

Same as {infoset:attribute information item}

[Definition: "*prefix*:*local-name*" {infoset:attribute}]

This phrase references an {infoset:attribute}

1. for which the value of the namespace name property is the namespace name associated by this sub clause with the namespace prefix "*prefix*", and
2. for which the value of the local name property is "*local-name*".

[Definition: "*prefix*:*local-name*" {infoset:element}]

This phrase references an {infoset:element}

1. for which the value of the namespace name property is the namespace name associated by this sub clause with the namespace prefix "*prefix*", and
2. for which the value of the local name property is "*local-name*".

B.2.4.3 {schema} Terminology

The following definitions derive from {schema}.

[Definition: XML Schema namespace]

The XML Schema namespace is the namespace for which the namespace name is "http://www.w3.org/2001/XMLSchema." This sub clause associates the namespace name with the namespace prefix "xsd."

The above definition serves to associate the namespace prefix "xsd" with the namespace name "http://www.w3.org/2001/XMLSchema."

[Definition: explicit members of the facets property of a {schema:simple type definition}]

> The XML representation of a {schema:simple type definition} is an `xsd:simpleType` {infoset:element}, the children property of which includes a `xsd:restriction` {infoset:element}, the children property of which may include a set of {infoset:element}.
>
> A subset of these {infoset:element}s represent a set of {schema:facet}s. The explicit members of the facets property of a {schema:simple type definition} are these {schema:facet}s.

The above definition distinguishes between the explicit members of the facets property of a {schema:simple type definition} and all members of the facets property, which also include the members of the facets property of the {schema:base type definition} for the {schema:simple type definition}.

[Definition: explicit members of the member type definitions property of a {schema:union simple type definition}]

> The XML representation of a {schema:simple type definition} derived by union from a set of {schema:simple type definition}s is an `xsd:simpleType` {infoset:element}. The children property of this {infoset:element} includes an `xsd:union` {infoset:element}, the attributes property of which includes an `xsd:memberTypes` {infoset:attribute}.
>
> The items in the actual value of the `xsd:memberTypes` {infoset:attribute} resolve to {schema:simple type definition}s. These {schema:simple type definition}s are the explicit members of the member type definitions property of the {schema:union simple type definition}.

The above definition distinguishes between the explicit members of the member type definitions property of a {schema:union simple type definition} and the members of the member type definitions property; the members of the member type definitions property are the explicit members of the member type definitions property, with those explicit members that are {schema:union simple type definitions} with the members of their member type definitions property.

[Definition: explicit members of the attribute uses property of a {schema:complex type definition}]

> The XML representation of a {schema:complex type definition} is an `xsd:complexType` {infoset:element}. The descendants of this {infoset:element} include `xsd:attribute` {infoset:element}s and `xsd:attributeGroup` {infoset:element}s.
>
> In the case of the former, each `xsd:attribute` {infoset:element} represents an {schema:attribute use}; each of these {schema:attribute use}s is an explicit member of the attribute uses property of the {schema:complex type definition}.
>
> In the case of the latter, the `xsd:ref` {infoset:attribute} of the `xsd:attributeGroup` {infoset:element} resolves to a {schema:attribute group}. Each member of the attribute uses property of the {schema:attribute group} is an explicit member of the attribute uses property of the {schema:complex type definition}.

The above definition distinguishes between the explicit members of the attribute uses property of a {schema:complex type definition} and all members of the attribute uses property, which also include the members of the attribute uses property of the {schema:base type definition} for the {schema:complex type definition}.

[Definition: {schema:element use}]

> A {schema:element use} is a {schema:particle} for which the term property is a {schema:element declaration}.

[Definition: {schema:wildcard use}]

> A {schema:wildcard use} is a {schema:particle} for which the term property is a {schema:wildcard}.

[Definition: {schema:model group use}]

> A {schema:model group use} is a {schema:particle} for which the term property is a {schema:model group}.

The above definitions distinguish between {schema:particle}s based on the term property of the {schema:particle}.

[Definition: {schema:choice}]

A {schema:choice} is a {schema:model group use} and for which the value of the compositor property of the {schema:model group} is “choice.”

[Definition: {schema:sequence}]

A {schema:choice} is a {schema:model group use} and for which the value of the compositor property of the {schema:model group} is “sequence.”

The above definitions distinguish between {schema:model group use}s based on the value of the compositor property of the {schema:model group}.

B.2.4.4 {niem} Terminology

The following definitions derive from {niem}.

[Definition: structures namespace]

The structures namespace is the namespace for which the namespace name is “http://niem.gov/niem/structures/2.0”. This sub clause associates the namespace name with the namespace prefix “s”.

The above definition serves to associate the namespace prefix “s” with the namespace name “http://niem.gov/niem/structures/2.0”.

[Definition: appinfo namespace]

The appinfo namespace is the namespace for which the namespace name is “http://niem.gov/niem/appinfo/2.0”. This sub clause associates the namespace name with the namespace prefix “i”.

The above definition serves to associate the namespace prefix “i” with the namespace name “http://niem.gov/niem/appinfo/2.0”.

[Definition: documentation for {schema:component}]

The annotation property of a {schema:component} is a {schema:annotation}; the user information property of this {schema:annotation} is represented in XML as a sequence of {infoset:element}s. This phrase references the first `xsd:documentation` {infoset:element} in that sequence.

[Definition: documentation for *prefix:local-name* {infoset:element}]

The children property of the *prefix:local-name* {infoset:element} includes an `xsd:annotation` {infoset:element}; this phrase references the `first xsd:documentation` {infoset:element} in the children property of the `xsd:annotation` {infoset:element}.

The above definitions specify the {infoset:element} that represents the documentation for either a {schema:component} or an {infoset:element}.

[Definition: “*application-information-qname*” application information for {schema:component}]

The annotation property of a {schema:component} is a {schema:annotation}; the application information property of this {schema:annotation} is represented in XML as a sequence of {infoset:element}s. This phrase references the “*application-information-qname*” {infoset:element} in that sequence.

[Definition: "*application-information-qname*" application information for {schema:component} must indicate*prefix:local-name*]

The members of the attributes property of the referenced {infoset:element} include each of the following:

1. An "i:namespace" {infoset:attribute} for which the normalized value is the namespace name associated by this specification with the namespace prefix *prefix*.
2. An "i:name" {infoset:attribute} for which the normalized value is *local-name*.

[Definition: value of the "*application-information-qname*" application information for {schema:component} must be "*application-information-value*"]

The children property of the referenced {infoset:element} are the character information items "*application-information-value*".

B.2.5 Built-In {uml:Element}

The following definitions identify {uml:Element}s that correspond to the XML Schema namespace and to built-in {schema:simple type definition}s.

B.2.5.1 Built-In {uml:Package}

The following definition identifies a {uml:Package} that corresponds to the XML Schema namespace.

[Definition: Built-In {uml:Package}]

A built-in {uml:Package} is any {uml:Package} that corresponds to the XML Schema namespace.

A built-in {uml:Package} corresponds to the XML Schema namespace. This sub clause employs the built-in {uml:Package} as a container for any {uml:DataType} that corresponds to a built-in {schema:simple type definition}.

B.2.5.2 Built-In {uml:DataType}

The following definitions identify a {uml:DataType} that corresponds to a built-in {schema:simple type definition}.

[Definition: Built-In Atomic {uml:DataType}]

A built-in atomic {uml:DataType} is a {uml:DataType}

1. for which the namespace is the built-in {uml:Package} and
2. for which the value of the name attribute is exactly one of the following:

Name of Built-In Atomic {uml:DataType}

anyURI	base64Binary	byte	boolean
date	dateTime	decimal	double
duration	ENTITY	float	gDay
gMonth	gMonthDay	gYear	gYearMonth
hexBinary	ID	IDREF	int
integer	language	long	Name
NCName	negativeInteger	NMTOKEN	nonNegativeInteger

nonPositiveInteger	normalizedString	NOTATION	positiveInteger
QName	short	string	time
token	unsignedByte	unsignedInt	unsignedLong
unsignedShort			

A built-in atomic {uml:DataType} corresponds to a built-in {schema:atomic simple type definition}.

[Definition: Built-In List {uml:DataType}]

A built-in list {uml:DataType} is a {uml:DataType}

1. for which the namespace is the built-in {uml:Package} and
2. for which the value of the name attribute is exactly one of the following:
    a. "NMTOKENS"
    b. "IDREFS" or
    c. "ENTITIES"

A built-in list {uml:DataType} corresponds to a built-in {schema:list simple type definition}.

[Definition: Built-In {uml:DataType}]

A built-in {uml:DataType} is any {uml:DataType} that is

1. a built-in atomic {uml:DataType} or
2. a built-in list {uml:DataType}

A built-in {uml:DataType} corresponds to a built-in {schema:simple type definition}.

While it would be unwise for a modeler to represent a built-in {uml:DataType} in a way that is inconsistent with its definition in XML Schema, this sub clause does not require a built-in {uml:DataType} reflect its definition in XML Schema, other than sharing its name and its target namespace, as this is sufficient to specify a mapping.

B.2.6 Categorized {uml:Element}

The following definitions place a {uml:Element} into a category based on its context.

B.2.6.1 Categorized {stereotype:Namespace}

The following definitions place a {stereotype:Namespace} into a category based on the value of the isConformant attribute.

[Definition: Conformant {stereotype:Namespace}]

A conformant {stereotype:Namespace} is any {stereotype:Namespace}

1. that is not a built-in {uml:Package} and
2. for which the value of the isConformant attribute is "true".

[Definition: Non-Conformant {stereotype:Namespace}]

A non-conformant {stereotype:Namespace} is any {stereotype:Namespace}

1. that is not a built-in {uml:Package} and
2. for which the value of the isConformant attribute is “false”

The above corresponds to conformant and non-conformant {schema:schema}.

[Definition: Categorized {stereotype:Namespace}]

A categorized {stereotype:Namespace} is any Conformant {stereotype:Namespace} or a Non-Conformant {stereotype:Namespace}.

[Definition: Uncategorized {stereotype:Namespace}]

An uncategorized {stereotype:Namespace} is any {stereotype:Namespace} that is not a categorized {stereotype:Namespace}.

The sub clause defines a categorized {stereotype:Namespace} and uncategorized {stereotype:Namespace} for use in definitions and rules.

B.2.6.2 Categorized {uml:DataType}

The following definitions place a {uml:DataType} into a category based on its context: its relationship to a {stereotype:Restriction}, its relationship to a {stereotype:UnionOf}, and its ownedAttribute {uml:Property}.

[Definition: Category 1 {uml:DataType}]

A Category 1 {uml:DataType} is any {uml:DataType}

1. that is not a built-in {uml:DataType}
2. for which no {uml:Property} is its ownedAttribute {uml:Property}
3. that is the client {uml:NamedElement} of exactly one {stereotype:Restriction} and
4. that is not the client {uml:NamedElement} of any {stereotype:UnionOf}

A Category 1 {uml:DataType} corresponds to a user-derived {schema:simple type definition} that is derived by restriction from another {schema:simple type definition}.

[Definition: Category 2 {uml:DataType}]

A Category 2 {uml:DataType} is any {uml:DataType}

1. that is not a built-in {uml:DataType}
2. for which exactly one {uml:Property} is its ownedAttribute {uml:Property}
3. that is not the client {uml:NamedElement} of any {stereotype:Restriction} and
4. that is not the client {uml:NamedElement} of any {stereotype:UnionOf}

A Category 2 {uml:DataType} corresponds to a user-defined {schema:simple type definition} that is derived by list from another {schema:simple type definition}.

[Definition: Category 3 {uml:DataType}]

A Category 3 {uml:DataType} is any {uml:DataType}

1. that is not a built-in {uml:DataType}
2. for which no {uml:Property} is its ownedAttribute {uml:Property}

3. that is not the client {uml:NamedElement} of any {stereotype:Restriction} and
4. that is the client {uml:NamedElement} of at least one {stereotype:UnionOf}

A Category 3 {uml:DataType} corresponds to a user-defined {schema:simple type definition} that is derived by union from at least one {schema:simple type definition}.

[Definition: Categorized {uml:DataType}]

A categorized {uml:DataType} is any {uml:DataType} that is

1. a Category 1 {uml:DataType}
2. a Category 2 {uml:DataType} or
3. a Category 3 {uml:DataType}

[Definition: Uncategorized {uml:DataType}]

An uncategorized {uml:DataType} is any {uml:DataType} that is not a categorized {uml:DataType}.

The sub clause defines a categorized {uml:DataType} and uncategorized {uml:DataType} for use in definitions and rules.

The following table summarizes the characteristics of the above {uml:DataType}s.

Characteristic of {uml:DataType}

{uml:DataType}	**Category**		
	1	**2**	**3**
must not be a built-in {uml:DataType}	Y	Y	Y
must be client {uml:NamedElement} of {stereotype:Restriction}	Y		
no {uml:Property} may be its ownedAttribute {uml:Property}	Y		Y
exactly one {uml:Property} must be its ownedAttribute {uml:Property}		Y	
must not be client {uml:NamedElement} of any {stereotype:UnionOf}	Y	Y	
must be client {uml:NamedElement} of at least one {stereotype:UnionOf}			Y

The following table summarizes the characteristics of the {schema:component} that correspond to the above categories of {uml:DataType}.

Characteristic of Corresponding {schema:component}

Corresponding {schema:component}	**Category**		
	1	**2**	**3**
must be user-derived {schema:simple type definition}	Y	Y	Y
must be user-derived {schema:atomic simple type definition}	Y		
must be user-derived {schema:list simple type definition}		Y	
must be user-derived {schema:union simple type definition}			Y

B.2.6.3 Categorized {uml:Enumeration}

The following definitions place a {uml:Enumeration} into a category.

[Definition: Category 1 {uml:Enumeration}]

A Category 1 {uml:Enumeration} is any Category 1 {uml:DataType} that is also a {uml:Enumeration}.

A Category 1 {uml:Enumeration} corresponds to a {schema:simple type definition} that is derived by restriction from another {schema:simple type definition}.

[Definition: Categorized {uml:Enumeration}]

A categorized {uml:Enumeration} is any {uml:Enumeration} that is a Category 1 {uml:Enumeration}.

[Definition: Uncategorized {uml:Enumeration}]

An uncategorized {uml:Enumeration} is any {uml:Enumeration} that is not a categorized {uml:Enumeration}.

The sub clause defines a categorized {uml:Enumeration} and uncategorized {uml:Enumeration} for use in definitions and rules.

B.2.6.4 Categorized {stereotype:List}

The following definitions place a {stereotype:List} into a category based on the category of the {uml:DataType} to which the stereotype is applied.

[Definition: Category 1 {stereotype:List}]

A Category 1 {stereotype:List} is any Category 1 {uml:DataType} to which the {stereotype:List} is applied.

A Category 1 {stereotype:List} corresponds to a {schema:list simple type definition} that is derived by restriction from another {schema:list simple type definition}.

[Definition: Category 2 {stereotype:List}]

A Category 2 {stereotype:List} is any Category 2 {uml:DataType} to which the {stereotype:List} is applied.

A Category 2 {stereotype:List} corresponds to a {schema:list simple type definition} that is derived by list from a {schema:simple type definition}.

[Definition: Categorized {stereotype:List}]

A categorized {stereotype:List} is any {stereotype:List} that is

1. a Category 1 {stereotype:List} or
2. a Category 2 {stereotype:List}

[Definition: Uncategorized {stereotype:List}]

An uncategorized {stereotype:List} is any {stereotype:List} that is not a categorized {stereotype:List}.

The sub clause defines a categorized {stereotype:List} and uncategorized {stereotype:List} for use in definitions and rules.

B.2.6.5 Categorized {stereotype:Union}

The following definitions place a {stereotype:Union} into a category based on the category of the {uml:DataType} to which the stereotype is applied.

[Definition: Category 1 {stereotype:Union}]

A Category 1 {stereotype:Union} is any Category 1 {uml:DataType} to which the {stereotype:Union} is applied.

A Category 1 {stereotype:Union} corresponds to a {schema:union simple type definition} that is derived by restriction from another {schema:union simple type definition}.

[Definition: Category 3 {stereotype:Union}]

A Category 3 {stereotype:Union} is any Category 3 {uml:DataType} to which the {stereotype:Union} is applied.

A Category 3 {stereotype:Union} corresponds to a {schema:union simple type definition} that is derived by union from more than one {schema:simple type definition}.

[Definition: Categorized {stereotype:Union}]

A categorized {stereotype:Union} is any {stereotype:Union} that is

1. a Category 1 {stereotype:Union} or
2. a Category 3 {stereotype:Union}

[Definition: Uncategorized {stereotype:Union}]

An uncategorized {stereotype:Union} is any {stereotype:Union} that is not a categorized {stereotype:Union}.

The sub clause defines a categorized {stereotype:Union} and uncategorized {stereotype:Union} for use in definitions and rules.

B.2.6.6 Categorized {uml:Class}

The following definitions place a {uml:Class} into a category based on its context: its relationship to {stereotype:XSDSimpleContent}, its relationship to {stereotype:Restriction}, and its relationship to {uml:Generalization}.

[Definition: Category 1 {uml:Class}]

A Category 1 {uml:Class} is any {uml:Class}

1. that is not the client {uml:NamedElement} of any {stereotype:XSDSimpleContent}
2. that is not the client {uml:NamedElement} of any {stereotype:Restriction}
3. that is not the specific {uml:Classifier} of any {uml:Generalization}
4. that is not the supplier {uml:NamedElement} of any {stereotype:Restriction} and
5. that is not the general {uml:Classifier} of any {uml:Generalization}

A Category 1 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with complex content} that is derived by extension from exactly one of “s:ComplexObjectType,” “s:MetadataType,” or “s:AugmentationType.”

[Definition: Category 2 {uml:Class}]

A Category 2 {uml:Class} is any {uml:Class}

1. that is not the client {uml:NamedElement} of any {stereotype:XSDSimpleContent}
2. that is not the client {uml:NamedElement} of any {stereotype:Restriction}
3. that is not the specific {uml:Classifier} of any {uml:Generalization} and
4. that is exactly one of the following:
    1. the supplier {uml:NamedElement} of at least one {stereotype:Restriction} or
    2. the general {uml:Classifier} of at least one {uml:Generalization}

A Category 2 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with complex content} that is derived by extension from exactly one of "s:ComplexObjectType" or "s:AugmentationType."

[Definition: Category 3 {uml:Class}]

A Category 3 {uml:Class} is any {uml:Class}

1. that is not the client {uml:NamedElement} of any {stereotype:XSDSimpleContent},
2. that is not the client {uml:NamedElement} of any {stereotype:Restriction}, and
3. that is the specific {uml:Classifier} of exactly one {uml:Generalization} the general {uml:Classifier} of which corresponds to a {schema:complex type definition with complex content}.

A Category 3 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with complex content} that is derived by extension from a {schema:complex type definition with complex content}.

[Definition: Category 4 {uml:Class}]

A Category 4 {uml:Class} is any {uml:Class}

1. that is not the client {uml:NamedElement} of any {stereotype:XSDSimpleContent},
2. that is the client {uml:NamedElement} of exactly one {stereotype:Restriction}, and
3. that is not the specific {uml:Classifier} of any {uml:Generalization}.

A Category 4 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with complex content} that is derived by restriction from a {schema:complex type definition with complex content}.

[Definition: Category 5 {uml:Class}]

A Category 5 {uml:Class} is any {uml:Class}

1. that is the client {uml:NamedElement} of exactly one {stereotype:XSDSimpleContent},
2. that is not the client {uml:NamedElement} of any {stereotype:Restriction}, and
3. that is not the specific {uml:Classifier} of any {uml:Generalization}.

A Category 5 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with simple content} that is derived by extension from a {schema:simple type definition}.

[Definition: Category 6 {uml:Class}]

A Category 6 {uml:Class} is a {uml:Class}

1. that is the client {uml:NamedElement} of a {stereotype:XSDSimpleContent},
2. that is the client {uml:NamedElement} of a {stereotype:Restriction}, and
3. that is not the specific {uml:Classifier} of any {uml:Generalization}.

A Category 6 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with simple content} that is derived by restriction from a {schema:complex type definition with simple content}.

[Definition: Category 7 {uml:Class}]

A Category 7 {uml:Class} is a {uml:Class}

1. that is not the client {uml:NamedElement} of a {stereotype:XSDSimpleContent},
2. that is not the client {uml:NamedElement} of a {stereotype:Restriction}, and
3. that is the specific {uml:Classifier} of exactly one {uml:Generalization}, the general {uml:Classifier} of which corresponds to a {schema:complex type definition with simple content}.

A Category 7 {uml:Class} is a {uml:Class} that corresponds to a {schema:complex type definition with simple content} that is derived by extension from a {schema:complex type definition with simple content}.

[Definition: Categorized {uml:Class}]

A categorized {uml:Class} is any {uml:Class} that is a

1. Category 1 {uml:Class}
2. Category 2 {uml:Class}
3. Category 3 {uml:Class}
4. Category 4 {uml:Class}
5. Category 5 {uml:Class}
6. Category 6 {uml:Class} or a
7. Category 7 {uml:Class}

[Definition: Uncategorized {uml:Class}]

An uncategorized {uml:Class} is any {uml:Class} that is not a categorized {uml:Class}.

The sub clause defines a categorized {uml:Class} and uncategorized {uml:Class} for use in definitions and rules.

The following table summarizes the characteristics of the above categories of {uml:Class}.

Characteristic of {uml:Class}							
{uml:Class}	**Category**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
must be client {uml:NamedElement} of {stereotype:XSDSimpleContent}					Y	Y	
must be client {uml:NamedElement} of {stereotype:Restriction}				Y		Y	
must be specific {uml:Classifier} of {uml:Generalization}			Y				Y
may be supplier {uml:NamedElement} of {stereotype:Restriction}		Y	Y	Y	Y	Y	Y
may be general {uml:Classifier} of {uml:Generalization}		Y	Y	Y	Y	Y	Y

The following table summarizes the characteristics of the {schema:component} that correspond to the above categories of {uml:Class}.

Characteristic of Corresponding {schema:component}							
Corresponding {schema:component}	**Category**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
{schema:complex type definition with complex content}	Y	Y	Y	Y			
{schema:complex type definition with simple content}					Y	Y	Y
may be derived by extension from “s:AugmentationType”	Y	Y					
may be derived by extension from “s:ComplexObjectType”	Y	Y					
may be derived by extension from “s:MetadataType”	Y						
may be derived by extension from {schema:ct with cc}			Y				
may be derived by restriction from {schema:ct with cc}				Y			
may be derived by extension from {schema:simple type definition}					Y		
may be derived by restriction from {schema:ct with sc}						Y	
may be derived by extension from {schema:ct with sc}							Y

B.2.6.7 Categorized {stereotype:AdapterType}

The following definitions place a {stereotype:AdapterType} into a category based on the category of the {uml:Class} to which the stereotype is applied.

[Definition: Category 1 {stereotype:AdapterType}]

A Category 1 {stereotype:AdapterType} is any {stereotype:AdapterType} that is applied to a Category 1 {uml:Class}.

A {stereotype:AdapterType} corresponds to a {niem:adapter type}. It may only be applied to a Category 1 {uml:Class}, as a {niem:adapter type} must be a {schema:complex type definition with complex content} and must not be the base type definition of any {schema:complex type definition}.

[Definition: Categorized {stereotype:AdapterType}]

A categorized {stereotype:AdapterType} is any {stereotype:AdapterType} that is a Category 1 {stereotype:AdapterType}.

[Definition: Uncategorized {stereotype:AdapterType}]

An uncategorized {stereotype:AdapterType} is any {stereotype:AdapterType} that is not a categorized {stereotype:AdapterType}.

The sub clause defines a categorized {stereotype:AdapterType} and uncategorized {stereotype:AdapterType} for use in definitions and rules.

B.2.6.8 Categorized {stereotype:AssociationType}

The following definitions place a {stereotype:AssociationType} into a category based on the category of the {uml:Class} to which the stereotype is applied.

[Definition: Category 1 {stereotype:AssociationType}]

A Category 1 {stereotype:AssociationType} is any {stereotype:AssociationType} that is applied to a Category 1 {uml:Class}.

[Definition: Category 2 {stereotype:AssociationType}]

A Category 2 {stereotype:AssociationType} is any {stereotype:AssociationType} that is applied to a Category 2 {uml:Class}.

[Definition: Category 3 {stereotype:AssociationType}]

A Category 3 {stereotype:AssociationType} is any {stereotype:AssociationType} that is applied to a Category 3 {uml:Class}.

[Definition: Category 4 {stereotype:AssociationType}]

A Category 4 {stereotype:AssociationType} is any {stereotype:AssociationType} that is applied to a Category 4 {uml:Class}.

A {stereotype:AssociationType} corresponds to a {niem:association type}. It may be applied to any categorized {uml:Class} except a Category 5 {uml:Class} and a Category 6 {uml:Class}, as a {niem:association type} must be a {schema:complex type definition with complex content}.

[Definition: Categorized {stereotype:AssociationType}]

A categorized {stereotype:AssociationType} is any {stereotype:AssociationType} that is a

1. Category 1 {stereotype:AssociationType}
2. Category 2 {stereotype:AssociationType}
3. Category 3 {stereotype:AssociationType} or a
4. Category 4 {stereotype:AssociationType}

[Definition: Uncategorized {stereotype:AssociationType}]

An uncategorized {stereotype:AssociationType} is any {stereotype:AssociationType} that is not a categorized {stereotype:AssociationType}.

The sub clause defines a categorized {stereotype:AssociationType} and uncategorized {stereotype:AssociationType} for use in definitions and rules.

B.2.6.9 Categorized {stereotype:AugmentationType}

The following definitions place a {stereotype:AugmentationType} into a category based on the category of the {uml:Class} to which the stereotype is applied.

[Definition: Category 1 {stereotype:AugmentationType}]

A Category 1 {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is applied to a Category 1 {uml:Class}.

[Definition: Category 2 {stereotype:AugmentationType}]

A Category 2 {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is applied to a Category 2 {uml:Class}.

[Definition: Category 3 {stereotype:AugmentationType}]

A Category 3 {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is applied to a Category 3 {uml:Class}.

[Definition: Category 4 {stereotype:AugmentationType}]

A Category 4 {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is applied to a Category 4 {uml:Class}.

A {stereotype:AugmentationType} corresponds to an augmentation type. It may be applied to any categorized {uml:Class} except a Category 5 {uml:Class} and a Category 6 {uml:Class}, as a {niem:augmentation type} must be a {schema:complex type definition with complex content}.

[Definition: Categorized {stereotype:AugmentationType}]

A categorized {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is a

1. Category 1 {stereotype:AugmentationType}
2. Category 2 {stereotype:AugmentationType}
3. Category 3 {stereotype:AugmentationType} or a
4. Category 4 {stereotype:AugmentationType}

[Definition: Uncategorized {stereotype:AugmentationType}]

An uncategorized {stereotype:AugmentationType} is any {stereotype:AugmentationType} that is not a categorized {stereotype:AugmentationType}.

The sub clause defines a categorized {stereotype:AugmentationType} and uncategorized {stereotype:AugmentationType} for use in definitions and rules.

B.2.6.10 Categorized {stereotype:MetadataType}

The following definitions place a {stereotype:MetadataType} into a category based on the category of the {uml:Class} to which the stereotype is applied.

[Definition: Category 1 {stereotype:MetadataType}]

A Category 1 {stereotype:MetadataType} is any {stereotype:MetadataType} that is applied to a Category 1 {uml:Class}.

A {stereotype:MetadataType} corresponds to a {niem:metadata type}. It may only be applied to a Category 1 {uml:Class}, as a {niem:metadata type} must be a {schema:complex type definition with complex content} and must not be the base type definition of any {schema:complex type definition}.

[Definition: Categorized {stereotype:MetadataType}]

A categorized {stereotype:MetadataType} is any {stereotype:MetadataType} that is a Category 1 {stereotype:MetadataType}.

[Definition: Uncategorized {stereotype:MetadataType}]

An uncategorized {stereotype:MetadataType} is any {stereotype:MetadataType} that is not a categorized {stereotype:MetadataType}.

The sub clause defines a categorized {stereotype:MetadataType} and uncategorized {stereotype:MetadataType} for use in definitions and rules.

B.2.6.11 Categorized {stereotype:ObjectType}

The following definitions place a {stereotype:ObjectType} into a category based on the category of the {uml:Class} to which the stereotype is applied.

[Definition: Category 1 {stereotype:ObjectType}]

A Category 1 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 1 {uml:Class}.

[Definition: Category 2 {stereotype:ObjectType}]

A Category 2 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 2 {uml:Class}.

[Definition: Category 3 {stereotype:ObjectType}]

A Category 3 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 3 {uml:Class}.

[Definition: Category 4 {stereotype:ObjectType}]

A Category 4 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 4 {uml:Class}.

[Definition: Category 5 {stereotype:ObjectType}]

A Category 5 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 5 {uml:Class}.

[Definition: Category 6 {stereotype:ObjectType}]

A Category 6 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 6 {uml:Class}.

[Definition: Category 7 {stereotype:ObjectType}]

A Category 7 {stereotype:ObjectType} is any {stereotype:ObjectType} that is applied to a Category 7 {uml:Class}.

A {stereotype:ObjectType} corresponds to a {niem:object type}. It may be applied to any categorized {uml:Class}.

[Definition: Categorized {stereotype:ObjectType}]

A categorized {stereotype:ObjectType} is any {stereotype:ObjectType} that is a

1. Category 1 {stereotype:ObjectType}
2. Category 2 {stereotype:ObjectType}
3. Category 3 {stereotype:ObjectType}
4. Category 4 {stereotype:ObjectType}
5. Category 5 {stereotype:ObjectType}
6. Category 6 {stereotype:ObjectType} or a
7. Category 7 {stereotype:ObjectType}

[Definition: Uncategorized {stereotype:ObjectType}]

An uncategorized {stereotype:ObjectType} is any {stereotype:ObjectType} that is not a categorized {stereotype:ObjectType}.

The sub clause defines a categorized {stereotype:ObjectType} and uncategorized {stereotype:ObjectType} for use in definitions and rules.

B.2.6.12 Categorized {uml:Property}

The following definitions place a {uml:Property} into a category based on its context: its class, its type, and whether it is the client {uml:NamedElement} of a {stereotype:References}.

[Definition: Category 1 {uml:Property}]

A Category 1 {uml:Property} is any {uml:Property}

1. for which the class is a {stereotype:PropertyHolder},
2. for which the type is not a {stereotype:Choice}, and
3. that is not the client {uml:NamedElement} of any {stereotype:References}.

A Category 1 {uml:Property} corresponds to either a {schema:element declaration} or a {schema:attribute declaration}.

[Definition: Category 2 {uml:Property}]

A Category 2 {uml:Property} is any {uml:Property}

1. for which the class is a categorized {uml:Class},
2. for which the type is not a {stereotype:Choice}, and
3. that is not the client {uml:NamedElement} of any {stereotype:References}.

A Category 2 {uml:Property} corresponds to exactly one of the following:

1. both a {schema:element declaration} and a {schema:element use} that is an item in the particles property of a {schema:sequence};
2. both a {schema:attribute declaration} and a {schema:attribute use} that is an explicit member of the attribute uses property of a {schema:complex type definition}; or
3. a {schema:wildcard use} that is an item in the particles property of a {schema:sequence}.

[Definition: Category 3 {uml:Property}]

A Category 3 {uml:Property} is any {uml:Property}

1. for which the class is a categorized {uml:Class},
2. for which the type is not a {stereotype:Choice}, and
3. that is the client {uml:NamedElement} of exactly one {stereotype:References}.

A Category 3 {uml:Property} corresponds to exactly one of the following:

1. a {schema:element use} that is an item in the particles property of a {schema:sequence}; or
2. a {schema:attribute use} that is an explicit member of the attribute uses property of a {schema:complex type definition}.

[Definition: Category 4 {uml:Property}]

A Category 4 {uml:Property} is any {uml:Property}

1. for which the class is a {stereotype:Choice},
2. for which the type is not a {stereotype:Choice}, and

3. that is not the client {uml:NamedElement} of any {stereotype:References}.

A Category 4 {uml:Property} corresponds to exactly one of the following:

1. both a {schema:element declaration} and a {schema:element use} that is an item in the particles property of a {schema:choice}; or
2. a {schema:wildcard use} that is an item in the particles property of a {schema:choice}.

[Definition: Category 5 {uml:Property}]

A Category 5 {uml:Property} is any {uml:Property}

1. for which the class is a {stereotype:Choice},
2. for which the type is not a {stereotype:Choice}, and
3. that is the client {uml:NamedElement} of exactly one {stereotype:References}.

A Category 5 {uml:Property} corresponds to a {schema:element use} that is an item in the particles property of a {schema:choice}.

[Definition: Category 6 {uml:Property}]

A Category 6 {uml:Property} is any {uml:Property}

1. for which the class is a categorized {uml:Class},
2. for which the type is a {stereotype:Choice}, and
3. that is not the client {uml:NamedElement} of any {stereotype:References}.

A Category 6 {uml:Property} corresponds to a {schema:choice}.

[Definition: Categorized {uml:Property}]

A categorized {uml:Property} is any {uml:Property} that is a

1. Category 1 {uml:Property}
2. Category 2 {uml:Property}
3. Category 3 {uml:Property}
4. Category 4 {uml:Property}
5. Category 5 {uml:Property} or a
6. Category 6 {uml:Property}

[Definition: Uncategorized {uml:Property}]

An uncategorized {uml:Property} is any {uml:Property} that is not a categorized {uml:Property}.

The sub clause defines a categorized {uml:Property} and uncategorized {uml:Property} for use in definitions and rules.

The following table summarizes the characteristics of the above categories of {uml:Property}.

Characteristic of {uml:Property}						
{uml:Property}	**Category**					
	1	**2**	**3**	**4**	**5**	**6**
class must be {stereotype:PropertyHolder}	Y					
class must be categorized {uml:Class}		Y	Y			Y
class must be {stereotype:Choice}				Y	Y	
type must be {stereotype:Choice}						Y
type must not be {stereotype:Choice}	Y	Y	Y	Y	Y	
must be client {uml:NamedElement} of {stereotype:References}			Y		Y	

The following table summarizes the characteristics of the {schema:component} that correspond to the above categories of {uml:Property}.

Characteristic of Corresponding {schema:component}						
{uml:Property}	**Category**					
	1	**2**	**3**	**4**	**5**	**6**
{schema:element declaration}	Y	Y		Y		
{schema:element use} in {schema:sequence}		Y	Y			
{schema:element use} in {schema:choice}				Y	Y	
{schema:wildcard use} in {schema:sequence}		Y				
{schema:wildcard use} in {schema:choice}				Y		
{schema:attribute declaration}	Y	Y				
{schema:attribute use} in {schema:complex type definition}		Y	Y			
{schema:choice}						Y

B.2.6.13 Categorized {stereotype:XSDProperty}

The following definitions place a {stereotype:XSDProperty} into a category based on the category of the {uml:Property} to which the stereotype is applied.

[Definition: Category 1 {stereotype:XSDProperty}]

A Category 1 {stereotype:XSDProperty} is any Category 1 {uml:Property} to which the {stereotype:XSDProperty} is applied.

[Definition: Category 2 {stereotype:XSDProperty}]

A Category 2 {stereotype:XSDProperty} is any Category 2 {uml:Property} to which the {stereotype:XSDProperty} is applied.

[Definition: Category 3 {stereotype:XSDProperty}]

A Category 3 {stereotype:XSDProperty} is any Category 3 {uml:Property} to which the {stereotype:XSDProperty} is applied.

The above {stereotype:XSDProperty} correspond to a {schema:element declaration}, {schema:element use}, {schema:attribute declaration}, or {schema:attribute use}.

[Definition: Category 4 {stereotype:XSDProperty}]

A Category 4 {stereotype:XSDProperty} is any Category 4 {uml:Property} to which the {stereotype:XSDProperty} is applied.

[Definition: Category 5 {stereotype:XSDProperty}]

A Category 5 {stereotype:XSDProperty} is any Category 5 {uml:Property} to which the {stereotype:XSDProperty} is applied.

The above {stereotype:XSDProperty} correspond to a {schema:element declaration} or a {schema:element use}.

A {stereotype:XSDProperty} may be applied to any categorized {uml:Property} except a Category 6 {uml:Property}, as a {stereotype:XSDProperty} must correspond to a {schema:element declaration}, {schema:element use}, {schema:attribute declaration}, or {schema:attribute use}, while a Category 6 {uml:Property} corresponds to a {schema:choice}.

[Definition: Categorized {stereotype:XSDProperty}]

A categorized {stereotype:XSDProperty} is a {stereotype:XSDProperty} that is a

1. Category 1 {stereotype:XSDProperty}
2. Category 2 {stereotype:XSDProperty}
3. Category 3 {stereotype:XSDProperty}
4. Category 4 {stereotype:XSDProperty} or a
5. Category 5 {stereotype:XSDProperty}

[Definition: Uncategorized {stereotype:XSDProperty}]

An uncategorized {stereotype:XSDProperty} is any {stereotype:XSDProperty} that is not a categorized {stereotype:XSDProperty}.

The sub clause defines a categorized {stereotype:XSDProperty} and uncategorized {stereotype:XSDProperty} for use in definitions and rules.

B.2.6.14 Categorized {stereotype:XSDAnyProperty}

The following definitions place a {stereotype:XSDAnyProperty} into a category based on the category of the {uml:Property} to which the stereotype is applied.

[Definition: Category 2 {stereotype:XSDAnyProperty}]

A Category 2 {stereotype:XSDAnyProperty} is any Category 2 {uml:Property} to which the {stereotype:XSDAnyProperty} is applied.

[Definition: Category 4 {stereotype:XSDAnyProperty}]

> A Category 4 {stereotype:XSDAnyProperty} is any Category 4 {uml:Property} to which the {stereotype:XSDAnyProperty} is applied.

The above {stereotype:XSDAnyProperty} correspond to a {schema:wildcard use}.

[Definition: Categorized {stereotype:XSDAnyProperty}]

> A categorized {stereotype:XSDAnyProperty} is a {stereotype:XSDAnyProperty} that is a
>
> 1. Category 1 {stereotype:XSDAnyProperty} or
> 2. Category 2 {stereotype:XSDAnyProperty}

[Definition: Uncategorized {stereotype:XSDAnyProperty}]

> An uncategorized {stereotype:XSDAnyProperty} is any {stereotype:XSDAnyProperty} that is not a categorized {stereotype:XSDAnyProperty}.

The sub clause defines a categorized {stereotype:XSDAnyProperty} and uncategorized {stereotype:XSDAnyProperty} for use in definitions and rules.

B.2.6.15 Categorized {stereotype:SequenceID}

The following definitions place a {stereotype:SequenceID} into a category based on the category of the {uml:Property} to which the stereotype is applied.

[Definition: Category 2 {stereotype:SequenceID}]

> A Category 2 {stereotype:SequenceID} is any Category 2 {uml:Property} to which the {stereotype:SequenceID} is applied.

The above {stereotype:SequenceID} corresponds to a {schema:attribute use}.

[Definition: Categorized {stereotype:SequenceID}]

> A categorized {stereotype:SequenceID} is a {stereotype:SequenceID} that is a Category 2 {stereotype:SequenceID}.

[Definition: Uncategorized {stereotype:SequenceID}]

> An uncategorized {stereotype:SequenceID} is any {stereotype:SequenceID} that is not a categorized {stereotype:SequenceID}.

The sub clause defines a categorized {stereotype:SequenceID} and uncategorized {stereotype:SequenceID} for use in definitions and rules.

B.2.7 Stereotyped {uml:Element}

For the purpose of this sub clause, a stereotyped {uml:Element} is not merely a {uml:Element} to which a stereotype is applied, nor even a {uml:Element} to which a stereotype from this profile is applied; it is instead a member of the set of the above categorized stereotypes.

B.2.7.1 Stereotyped {uml:DataType}

The following definitions distinguish a categorized {uml:DataType} based on whether a stereotype is applied to the {uml:DataType}.

[Definition: Stereotyped {uml:DataType}]

A stereotyped {uml:DataType} is any

1. Category 1 {stereotype:List}
2. Category 2 {stereotype:List}
3. Category 1 {stereotype:Union} or
4. Category 3 {stereotype:Union}

[Definition: Unstereotyped {uml:DataType}]

An unstereotyped {uml:DataType} is any categorized {uml:DataType} that is not a stereotyped {uml:DataType}.

B.2.7.2 Stereotyped {uml:Class}

The following definitions distinguish a categorized {uml:Class} based on whether a stereotype is applied to the {uml:Class}.

[Definition: Stereotyped {uml:Class}]

A stereotyped {uml:Class} is any

1. Category 1 {stereotype:AdapterType}
2. Category 1 {stereotype:AssociationType}
3. Category 2 {stereotype:AssociationType}
4. Category 3 {stereotype:AssociationType}
5. Category 4 {stereotype:AssociationType}
6. Category 1 {stereotype:AugmentationType}
7. Category 2 {stereotype:AugmentationType}
8. Category 3 {stereotype:AugmentationType}
9. Category 4 {stereotype:AugmentationType}
10. Category 1 {stereotype:MetadataType}
11. Category 1 {stereotype:ObjectType}
12. Category 2 {stereotype:ObjectType}
13. Category 3 {stereotype:ObjectType}
14. Category 4 {stereotype:ObjectType}
15. Category 5 {stereotype:ObjectType}
16. Category 6 {stereotype:ObjectType or any
17. Category 7 {stereotype:ObjectType}

[Definition: Unstereotyped {uml:Class}]

An unstereotyped {uml:Class} is any categorized {uml:Class} that is not a stereotyped {uml:Class}.

B.2.7.3 Stereotyped {uml:Property}

The following definitions distinguish a categorized {uml:Property} based on whether a stereotype is applied to the {uml:Property}.

[Definition: Stereotyped {uml:Property}]

A stereotyped {uml:Property} is any

1. Category 1 {stereotype:XSDProperty}
2. Category 2 {stereotype:XSDProperty}
3. Category 3 {stereotype:XSDProperty}
4. Category 4 {stereotype:XSDProperty}
5. Category 5 {stereotype:XSDProperty}
6. Category 2 {stereotype:XSDAnyProperty}
7. Category 4 {stereotype:XSDAnyProperty} or
8. Category 2 {stereotype:SequenceID}

[Definition: Unstereotyped {uml:Property}]

An unstereotyped {uml:Property} is any categorized {uml:Property} that is not a stereotyped {uml:Property}.

B.2.8 Equality for the Value of a Property of a {schema:component}

Ultimately, a mapping between a NIEM-conformant XML schema and a NIEM-conforming Platform Specific Model expands to a set of mappings between the values of the properties of the {schema:component} of the former and the values of the attributes of the {uml:Element} of the latter. Thus to define a mapping between the schema and the model, it is necessary to define a mapping between the values in the schema and the values in the model.

[Rule: Equality for instance of {uml:Boolean}]

A value that is an instance of {uml:Boolean} shall equal the actual value of an instance of the {schema:boolean datatype} if and only if the truth value of the former is equal to the truth value of the latter.

[Rule: Equality for instance of {uml:Integer}]

A value that is an instance of {uml:Integer} shall equal the actual value of an instance of the {schema:decimal datatype} if and only if the number value of the former is equal to the number value of the latter.

[Rule: Equality for instance of {uml:String}]

A value that is an instance of {uml:String} shall equal the actual value of an instance of the {schema:string datatype} if and only if the Universal Character Set encoding of the former is a lexical representation of the latter.

[Rule: Equality for instance of {uml:UnlimitedNatural}]

If the value of an instance of {uml:UnlimitedNatural} is numeric, the value shall equal the actual value of an instance of the {schema:decimal datatype} if and only if the numeric value of the former is equal to the numeric value of the latter.

If the value of an instance of {uml:UnlimitedNatural} is “*”, the value shall equal the value of a property of a {schema:component} if and only if the value of the {schema:component} is “unbounded”.

B.2.9 Mapping

B.2.9.1 Mapping the Documentation for a {schema:component}

[Definition: documentation for a {uml:Element}]

The documentation for a {uml:Element} is the ownedComment of the {uml:Element} to which the {stereotype:Documentation} is applied.

[Rule: Mapping for a {stereotype:Documentation}]

A mapping shall exist between the documentation for a {schema:component} and the documentation for a {uml:Element} if and only if exactly one of the following is true:

1. (annotation property) - The documentation for a {schema:component} is present, the documentation for a {uml:Element} is present, and the character information items of the documentation for the {schema:component} equal the value of the body attribute of the documentation for the {uml:Element}.
2. (annotation property) - The documentation for a {schema:component} is absent and the documentation for a {uml:Element} is absent.

B.2.9.2 Mapping the Facets Property of a {schema:simple type definition}

[Definition: facet for a {uml:DataType}]

A facet for a {uml:DataType} is a (name, value, documentation) tuple derived from a {uml:DataType}.

[Definition: facet set for a {uml:DataType}]

A facet set for a {uml:DataType} is a set of facets for a {uml:DataType}, derived as follows:

1. If the {stereotype:ValueRestriction} is applied to the {uml:DataType}, for each attribute of the {stereotype:ValueRestriction} that is present and has exactly one value, the facet set must include a facet
   a. for which the value of the name element is the name of the attribute and
   b. for which the value of the value element is the value of the attribute
   c. for which the value of the documentation element is absent
2. If the {stereotype:ValueRestriction} is applied to the {uml:DataType}, for each attribute of the {stereotype:ValueRestriction} that is present and has more than one value, for each value of the attribute, the facet set must include a facet
   a. for which the value of the name element is the name of the attribute and
   b. for which the value of the value element is that value of the attribute
   c. for which the value of the documentation element is absent
3. If the {stereotype:XSDRepresentationRestriction} is applied to the {uml:DataType}, for each attribute of the {stereotype:XSDRepresentationRestriction} that is present and has exactly one value, the facet set must include a facet
   a. for which the value of the name element is the name of the attribute and
   b. for which the value of the value element is the value of the attribute
   c. for which the value of the documentation element is absent

4. If the {uml:DataType} is a {uml:Enumeration}, for each ownedLiteral {uml:EnumerationLiteral}, the facet set must include a facet
   a. for which the value of the name element is “enumeration”
   b. for which the value of the value element is the value of the name attribute of the ownedLiteral {uml:EnumerationLiteral} and
   c. for which the value of the documentation element is the documentation for the ownedLiteral {uml:EnumerationLiteral}

[Rule: Mapping for the facet set for a {uml:DataType}]

A mapping shall exist between the facets property of a {schema:simple type definition} and the facet set for a {uml:DataType} if and only if each of the following is true:

1. (facets property) - For each explicit member of the facets property of the {schema:simple type definition}, a mapping must exist between the {schema:component} and exactly one facet in the facet set for the {uml:DataType}.
2. (facets property) - For each facet in the facet set for the {uml:DataType}, a mapping must exist between the facet and exactly one {schema:component} among the explicit members of the facets property of the {schema:simple type definition}.

[Rule: Mapping for a facet for a {uml:DataType}]

A mapping shall exist between a {schema:component} in the facets property of a {schema:simple type definition} and a facet of a {uml:DataType} if and only if each of the following is true:

1. (annotation property) - A mapping must exist between the documentation for the {schema:component} and the documentation element of the facet.
2. (name property) - The value of the name property of the {schema:component} must equal the value of the name element of the facet.
3. (value property) - The value of the value property of the {schema:component} must equal the value of the value element of the facet.

B.2.9.3 Mapping a {schema:simple type definition}

The following rules ensure that a relationship between two instances of {uml:DataType} correspond to the relationship between two {schema:simple type definition}.

[Rule: The supplier {uml:NamedElement} for an Unstereotyped Category 1 {uml:DataType}]

The supplier {uml:NamedElement} of the {stereotype:Restriction} for which an unstereotyped Category 1 {uml:DataType} is the client {uml:NamedElement} must be exactly one of

1. a built-in atomic {uml:DataType} or
2. an unstereotyped Category 1 {uml:DataType}

The base type definition property for a {schema:atomic simple type definition} must be a {schema:atomic simple type definition}.

[Rule: The supplier {uml:NamedElement} for a Category 1 {stereotype:List}]

The supplier {uml:NamedElement} of the {stereotype:Restriction} for which a Category 1 {stereotype:List} is the client {uml:NamedElement} must be exactly one of a

1. built-in list {uml:DataType} or a
2. categorized {stereotype:List}

The base type definition property for a {schema:list simple type definition} must be a {schema:list simple type definition}.

[Rule: The type of the {uml:Property} which is the ownedAttribute {uml:Property} of a Category 2 {stereotype:List}]

The type of the {uml:Property} which is the ownedAttribute {uml:Property} of a Category 2 {stereotype:List} must be exactly one of

1. a built-in atomic {uml:DataType}
2. an unstereotyped Category 1 {uml:DataType} or a
3. a categorized {stereotype:Union}

The item type definition property for a {schema:list simple type definition} must be exactly one of a {schema:atomic simple type definition} or a {schema:union simple type definition}.

[Rule: The supplier {uml:NamedElement} for a Category 1 {stereotype:Union}]

The supplier {uml:NamedElement} of the {stereotype:Restriction} for which a Category 1 {stereotype:Union} is the client {uml:NamedElement} must be a categorized {stereotype:Union}.

The base type definition property for a {schema:union simple type definition} must be a {schema:union simple type definition}.

[Rule: The supplier {uml:NamedElement} for a Category 3 {stereotype:Union}]

For each {stereotype:UnionOf} for which a Category 3 {stereotype:Union} is the client {uml:NamedElement}, the supplier {uml:NamedElement} of the {stereotype:UnionOf} must be exactly one of

1. a built-in {uml:DataType}
2. an unstereotyped Category 1 {uml:DataType}
3. a categorized {stereotype:List} or
4. a categorized {stereotype:Union}

The member type definitions property for a {schema:union simple type definition} must be a sequence consisting of {schema:atomic simple type definition}, {schema:list simple type definition}, or {schema:union simple type definition}.

[Rule: Mapping for a Built-In {uml:DataType}]

A mapping shall exist between a {schema:simple type definition} and a built-in {uml:DataType} if and only if each of the following is true:

1. (name property) - The value of the name property of the {schema:simple type definition} must equal the value of the name attribute of the {uml:DataType}.
2. (target namespace property) - The value of the target namespace property must be the XML Schema namespace.

[Rule: Mapping for a Categorized {uml:DataType}]

A mapping shall exist between a {schema:simple type definition} and a categorized {uml:DataType} only if each of the following is true:

1. (name property) - The value of the name property of the {schema:simple type definition} must equal the value of the name attribute of the {uml:DataType}.
2. (target namespace property) - The value of the target namespace property must equal the value of the targetNamespace attribute of the {stereotype:Namespace} that is the namespace of the {uml:DataType}.
3. (final property) - The value of the final property of the {schema:simple type definition} must be the empty set.
4. (annotation property) - A mapping must exist between the documentation for the {schema:simple type definition} the documentation for the {uml:DataType}.
5. (annotation property) - The “i:Base” application information for the {schema:simple type definition} must indicate “s:Object.”

The above rule is necessary for the following rules:

1. Mapping for an Unstereotyped Category 1 {uml:DataType}
2. Mapping for a Category 1 {stereotype:List}
3. Mapping for a Category 2 {stereotype:List}
4. Mapping for a Category 1 {stereotype:Union}
5. Mapping for a Category 3 {stereotype:Union}

- but not sufficient in itself to specify a mapping.

[Rule: Mapping for an Unstereotyped Category 1 {uml:DataType}]

A mapping shall exist between a {schema:atomic simple type definition} and an unstereotyped Category 1 {uml:DataType} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:DataType}” must hold.
2. (variety property) - The value of the variety property must be “atomic”.
3. (primitive type definition property) - The value of the primitive type definition property of the {schema:atomic simple type definition} must be the value of the primitive type definition property of the base type definition property of the {schema:atomic simple type definition}.
4. (facets property) - A mapping must exist between the explicit members of the facets property of the {schema:atomic simple type definition} and the facet set for the {uml:DataType}.
5. (base type definition property) - A mapping must exist between the base type definition property of the {schema:atomic simple type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {uml:DataType} is the client {uml:NamedElement}.

[Rule: Mapping for a Category 1 {stereotype:List}]

A mapping shall exist between a {schema:list simple type definition} and a Category 1 {stereotype:List} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:DataType}” must hold.
2. (variety property) - The value of the variety property must be “list”.
3. (item type definition property) - The value of the item type definition property of the {schema:list simple type definition} must be the value of the item type definition property of the base type definition property of the {schema:list simple type definition}.

4. (facets property) - A mapping must exist between the explicit members of the facets property of the {schema:list simple type definition} and the facet set for the {stereotype:List}.
5. (base type definition property) - A mapping must exist between the base type definition property of the {schema:list simple type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {stereotype:List} is the client {uml:NamedElement}.

[Rule: Mapping for a Category 2 {stereotype:List}]

A mapping shall exist between a {schema:list simple type definition} and a Category 2 {stereotype:List} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:DataType}” must hold.
2. (variety property) - The value of the variety property must be “list”.
3. (item type definition property) - A mapping must exist between the item type definition property of the {schema:list simple type definition} and the type of the {uml:Property} that is the ownedAttribute {uml:Property} of the {stereotype:List}.
4. (base type definition property) - The value of the base type definition property of the {schema:list simple type definition} must be the {schema:simple type definition} for {schema:anySimpleType}.

[Rule: Mapping for a Category 1 {stereotype:Union}]

A mapping shall exist between a {schema:union simple type definition} and a Category 1 {stereotype:Union} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:DataType}” must hold.
2. (variety property) - The value of the variety property must be “union”.
3. (union member type definitions property) - The value of the union member types definition property of the {schema:union simple type definition} must be the value of the union member types definition property of the base type definition property of the {schema:union simple type definition}.
4. (facets property) - A mapping must exist between the explicit members of the facets property of the {schema:union simple type definition} and the facet set for the {stereotype:Union}.
5. (base type definition property) - A mapping must exist between the base type definition property of the {schema:union simple type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {stereotype:Union} is the client {uml:NamedElement}.

[Rule: Mapping for a Category 3 {stereotype:Union}]

A mapping shall exist between a {schema:union simple type definition} and a Category 3 {stereotype:Union} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:DataType}” must hold.
2. (variety property) - The value of the variety property must be “union”.
3. (member type definitions property) - For each explicit member of the member type definitions property of the {schema:union simple type definition}, a mapping must exist between the {schema:component} and a supplier {uml:NamedElement} of a {stereotype:UnionOf} for which the {stereotype:Union} is the client {uml:NamedElement}.

4. (member type definitions property) - For each supplier {uml:NamedElement} of a {stereotype:UnionOf} for which the {stereotype:Union} is the client {uml:NamedElement}, a mapping must exist between the supplier {uml:NamedElement} and an explicit member in the member type definitions property of the {schema:union simple type definition}.
5. (base type definition property) - The value of the base type definition property of the {schema:union simple type definition} must be the {schema:simple type definition} for {schema:anySimpleType}.

B.2.9.4 Mapping the Attribute Uses Property of a {schema:complex type definition}

[Definition: Attribute Use {uml:Property}]

An attribute use {uml:Property} is a

1. Category 2 {stereotype:XSDProperty} for which the value of the kind attribute is “attribute”
2. Category 3 {stereotype:XSDProperty} for which the value of the kind attribute is “attribute” or a
3. Category 2 {stereotype:SequenceID}

[Definition: Attribute Use Set for a {uml:Class}]

The attribute use set for a {uml:Class} is the subset of ownedAttribute {uml:Property} which are attribute use {uml:Property}.

[Rule: The supplier {uml:NamedElement} for an attribute use {uml:Property}]

For the each supplier {uml:NamedElement} of a {stereotype:References} for which the client {uml:NamedElement} is an attribute use {uml:Property}, each of the following must be true:

1. the supplier {uml:NamedElement} must be an attribute declaration {uml:Property};
2. the name of the client {uml:NamedElement} must equal the name of the supplier {uml:NamedElement}; and
3. the type of the client {uml:NamedElement} must be the type of the supplier {uml:NamedElement}.

[Rule: Mapping for the Attribute Use Set for a {uml:Class}]

A mapping shall exist between the explicit members of the attribute uses property of a {schema:complex type definition} and the attribute use set for a {uml:Class} if and only if the following each of the following are true:

1. (attribute uses property) - For each {schema:attribute use} among the explicit members of the attribute uses property of the {schema:complex type definition}, a mapping must exist between the {schema:attribute use} and exactly one attribute use {uml:Property} in the attribute use set for the {uml:Class}.
2. (attribute uses property) - For each attribute use {uml:Property} in the attribute use set for the {uml:Class}, a mapping must exist between the attribute use {uml:Property} and exactly one {schema:attribute use} among the explicit members of the attribute uses property of the {schema:complex type definition}.

[Rule: Mapping for an Attribute Use {uml:Property}]

A mapping shall exist between a {schema:attribute use} and an Attribute Use {uml:Property} only if each of the following is true:

1. (required property) Exactly one of the following must be true:
   a. The lower value of the {uml:Property} must be “0,” the upper value of the {uml:Property} must be “1,” and the value of the required property must be “false”.

   b. The lower value of the {uml:Property} must be "1," the upper value of the {uml:Property} must be "1," and the value of the required property must be "true".

2. (value constraint property) - Exactly one of the following must be true:
   a. The fixed attribute of the {uml:Property} is absent and the value constraint property of the {schema:attribute declaration} is absent.
   b. The fixed attribute of the {uml:Property} is present; the value constraint property of the {schema:attribute declaration} is a pair consisting of "fixed" and a value; and that value equals the value of the fixed attribute of the {uml:Property}.

The above rule is necessary for the following rules:

1. Mapping for a Category 2 Attribute Use {stereotype:XSDProperty}
2. Mapping for a Category 3 Attribute Use {stereotype:XSDProperty}
3. Mapping for a Category 2 Attribute Use {stereotype:SequenceID}

- but not sufficient in itself to specify a mapping.

[Rule: Mapping for a Category 2 Attribute Use {stereotype:XSDProperty}]

A mapping shall exist between a {schema:attribute use} and a Category 2 Attribute Use {stereotype:XSDProperty} if and only if each of the following is true:

1. The rule "Mapping for an Attribute Use {uml:Property}" must hold.
2. (attribute declaration property) - A mapping must exist between the attribute declaration property of the {schema:attribute use} and the {stereotype:XSDProperty}.

[Rule: Mapping for a Category 3 Attribute Use {stereotype:XSDProperty}]

A mapping shall exist between a {schema:attribute use} and a Category 3 Attribute Use {stereotype:XSDProperty} if and only if each of the following is true:

1. The rule "Mapping for an Attribute Use {uml:Property}" must hold.
2. (attribute declaration property) - A mapping must exist between the attribute declaration property of the {schema:attribute use} and the supplier {uml:NamedElement} of the {stereotype:References} for the {stereotype:XSDProperty}.

[Rule: Mapping for a Category 2 Attribute Use {stereotype:SequenceID}]

A mapping shall exist between a {schema:attribute use} and a Category 2 Attribute Use {stereotype:SequenceID} if and only if each of the following is true:

1. The rule "Mapping for an Attribute Use {uml:Property}" must hold.
2. (attribute declaration property) - The attribute declaration must be s:sequenceID.

B.2.9.5 Mapping the {schema:element use} for a {schema:complex type definition}

[Definition: Element Use {uml:Property}]

An element use {uml:Property} is any

1. unstereotyped Category 2 {uml:Property}
2. unstereotyped Category 3 {uml:Property}

3. unstereotyped Category 4 {uml:Property}
4. unstereotyped Category 5 {uml:Property}
5. Category 2 {stereotype:XSDProperty} for which the value of the kind attribute is “element”
6. Category 3 {stereotype:XSDProperty} for which the value of the kind attribute is “element”
7. Category 4 {stereotype:XSDProperty} for which the value of the kind attribute is “element” or any
8. Category 5 {stereotype:XSDProperty} for which the value of the kind attribute is “element”

[Definition: Choice {uml:Property}]

A choice {uml:Property} is any Category 6 {uml:Property}.

[Definition: Wildcard {stereotype:XSDAnyProperty}]

A wildcard {uml:Property} is any

1. Category 2 {stereotype:XSDAnyProperty} or any
2. Category 4 {stereotype:XSDAnyProperty}

[Definition: Particle {uml:Property}]

A particle {uml:Property} is any element use {uml:Property}, any choice {uml:Property}, or any wildcard {stereotype:XSDAnyProperty}.

[Definition: Model Group for a {uml:Class}]

The model group for a {uml:Class} is the sublist of ownedAttribute {uml:Property} that are particle {uml:Property}.

[Rule: The supplier {uml:NamedElement} for an element use {uml:Property}]

For the each supplier {uml:NamedElement} of a {stereotype:References} for which the client {uml:NamedElement} is an element use {uml:Property}, each of the following must be true:

1. the supplier {uml:NamedElement} must be an element declaration {uml:Property};
2. the name of the client {uml:NamedElement} must equal the name of the supplier {uml:NamedElement}; and
3. the type of the client {uml:NamedElement} must be the type of the supplier {uml:NamedElement}.

[Rule: Mapping for the Model Group for a {uml:Class}]

A mapping shall exist between a {schema:model group} and the model group for a {uml:Class} if and only if each of the following are true:

1. (particles property) - For each {schema:particle} in the particles property of the {schema:model group}, a mapping must exist between the {schema:particle} and exactly one particle {uml:Property} in model group for the {uml:Class}.
2. (particles property) - For each particle {uml:Property} in model group for the {uml:Class}, a mapping must exist between the particle {uml:Property} and exactly one {schema:particle} in the particles property of the {schema:sequence}.

[Rule: Mapping for a Particle {uml:Property}]

A mapping shall exist between a {schema:particle} and a particle {uml:Property} only if each of the following is true:

1. (min occurs property) - The value of the min occurs property of the {schema:particle} must equal the value of the lower attribute of the {uml:Property}.
2. (max occurs property) - The value of the max occurs property of the {schema:particle} must equal the value of the upper attribute of the {uml:Property}.

The above rule is necessary for the following rules:

1. Mapping for a Category 2 or Category 4 Element Use {uml:Property}
2. Mapping for a Category 3 or Category 5 Element Use {uml:Property}
3. Mapping for a Choice {uml:Property}
4. Mapping for a Wildcard {stereotype:XSDAnyProperty}

- but not sufficient in itself to specify a mapping.

[Rule: Mapping for a Category 2 or Category 4 Element Use {uml:Property}]

A mapping shall exist between a {schema:element use} and a Category 2 or Category 4 Element Use {uml:Property} if and only if each of the following is true:

1. The rule "Mapping for a Particle {uml:Property}" must hold.
2. (term property) - A mapping must exist between the term property of the {schema:element use} and the {uml:Property}.

[Rule: Mapping for a Category 3 or Category 5 Element Use {uml:Property}]

A mapping shall exist between a {schema:element use} and a Category 3 or Category 5 Element Use {uml:Property} if and only if each of the following is true:

1. The rule "Mapping for a Particle {uml:Property}" must hold.
2. (term property) - A mapping must exist between the term property of the {schema:element use} and the supplier {uml:NamedElement} of the {stereotype:References} for the {uml:Property}.

[Rule: Mapping for a Choice {uml:Property}]

A mapping shall exist between a {schema:choice} and a Choice {uml:Property} if and only if each of the following is true:

1. The rule "Mapping for a Particle {uml:Property}" must hold.
2. (term property) - A mapping must exist between the term property of the {schema:choice} and the model group for the class of the {uml:Property}.

[Rule: Mapping for a Wildcard {stereotype:XSDAnyProperty}]

A mapping shall exist between a {schema:wildcard use} and a wildcard {uml:Property} if and only if each of the following is true:

1. The rule "Mapping for a Particle {uml:Property}" must hold.
2. (term property) - The term property must be a {schema:wildcard} for which each of the following is true:
   a. (namespace constraint property) - The value of the namespace constraint property of the {schema:wildcard} must equal the value of the namespace attribute of the {stereotype:XSDAnyProperty}.
   b. (process contents property) - The value of the process contents property of the {schema:wildcard} must equal the value of the processContents attribute of the {stereotype:XSDAnyProperty}.

B.2.9.6 Mapping a {schema:complex type definition}

[Rule: The general {uml:Classifier} for an Unstereotyped {uml:Class} or a categorized {stereotype:ObjectType}]

The general {uml:Classifier} of a {uml:Generalization} for which an unstereotyped {uml:Class} or categorized {stereotype:ObjectType} is the specific {uml:Classifier} must be exactly one of an unstereotyped {uml:Class} or a categorized {stereotype:ObjectType}.

[Rule: The supplier {uml:NamedElement} for an Unstereotyped {uml:Class} or a categorized {stereotype:ObjectType}]

The supplier{uml:NamedElement} of a {stereotype:Restriction} for which an unstereotyped {uml:Class} or categorized {stereotype:ObjectType} is the client{uml:NamedElement} must be exactly one of an unstereotyped {uml:Class} or a categorized {stereotype:ObjectType}.

The base type definition property of a {niem:object type} must be exactly one of “s:ComplexObjectType” and a {niem:object type}.

[Rule: The general {uml:Classifier} for a categorized {stereotype:AssociationType}]

The general {uml:Classifier} of a {uml:Generalization} for which a categorized {stereotype:AssociationType} is the specific {uml:Classifier} must be a categorized {stereotype:AssociationType}.

[Rule: The supplier {uml:NamedElement} for a categorized {stereotype:AssociationType}]

The supplier {uml:NamedElement} of a {stereotype:Restriction} for which a categorized {stereotype:AssociationType} is the client {uml:NamedElement} must be a categorized {stereotype:AssociationType}.

The base type definition property of a {niem:object type} must be exactly one of “s:ComplexObjectType” and a {niem:association type}.

[Rule: The general {uml:Classifier} for a categorized {stereotype:AugmentationType}]

The general {uml:Classifier} of a {uml:Generalization} for which a categorized {stereotype:AugmentationType} is the specific {uml:Classifier} must be a categorized {stereotype:AugmentationType}.

[Rule: The supplier {uml:NamedElement} for a categorized {stereotype:AugmentationType}]

The supplier {uml:NamedElement} of a {stereotype:Restriction} for which a categorized {stereotype:AugmentationType} is the client {uml:NamedElement} must be a categorized {stereotype: AugmentationType}.

The base type definition property of a {niem:augmentation type} must be exactly one of “s:AugmentationType” and an augmentation type.

[Rule: Mapping for a Categorized {uml:Class}]

A mapping shall exist between a categorized {uml:Class} and a {schema:complex type definition} only if each of the following is true:

1. (name property) - The value of the name property of the of the {schema:complex type definition} must equal the value of the name attribute of the {uml:Class}.
2. (target namespace property) - The value of the target namespace property must equal the value of the targetNamespace attribute of the {stereotype:Namespace} that is the namespace of the {uml:Class}.
3. (final property) - The value of the final property of the {schema:complex type definition} must be the empty set.

4. (abstract property) - The value of the abstract property of the {schema:complex type definition} must equal the value of the isAbstract attribute of the {uml:Class}.
5. (attribute wildcard property) - The attribute wildcard property of the {schema:complex type definition} must be absent.
6. (prohibited substitutions property) - The value of the prohibited substitutions property must be the empty set.
7. (annotation property) - A mapping must exist between the documentation for the {schema:complex type definition} the documentation for the {uml:Class}.

The above rule is necessary for the following rules:

1. Mapping for an Unstereotyped Category 1 or 2 {uml:Class} or a Category 1 or 2 {stereotype:ObjectType}
2. Mapping for an Unstereotyped Category 3 {uml:Class} or a Category 3 {stereotype:ObjectType}
3. Mapping for an Unstereotyped Category 4 {uml:Class} or a Category 4 {stereotype:ObjectType}
4. Mapping for an Unstereotyped Category 5 {uml:Class} or a Category 5 {stereotype:ObjectType}
5. Mapping for an Unstereotyped Category 6 {uml:Class} or a Category 6 {stereotype:ObjectType}
6. Mapping for an Unstereotyped Category 7 {uml:Class} or a Category 7 {stereotype:ObjectType}
7. Mapping for a Category 1 {stereotype:AdapterType}
8. Mapping for a Category 1 or 2 {stereotype:AssociationType}
9. Mapping for a Category 3 {stereotype:AssociationType}
10. Mapping for a Category 4 {stereotype:AssociationType}
11. Mapping for a Category 1 or 2 {stereotype:AugmentationType}
12. Mapping for a Category 3 {stereotype:AugmentationType}
13. Mapping for a Category 4 {stereotype:AugmentationType}
14. Mapping for a Category 1 {stereotype:MetadataType}

- but not sufficient in itself to specify a mapping.

[Rule: Mapping for an Unstereotyped Category 1 or 2 {uml:Class} or a Category1 or 2 {stereotype:ObjectType}]

A mapping shall exist between an unstereotyped Category 1 or 2 {uml:Class} or a Category 1 or 2 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - The base type definition property of the {schema:complex type definition} must be “s:ComplexObjectType”.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension”.
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only”. The content model must be a {schema:sequence} for which each of the following is true:
    a. (min occurs property) - the value of the min occurs property must be “1”;

   b. (max occurs property) - the value of the max occurs property must be “1”; and
   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate “s:Object”.

[Rule: Mapping for an Unstereotyped Category 3 {uml:Class} or a Category 3 {stereotype:ObjectType}]

A mapping shall exist between an unstereotyped Category 3 {uml:Class} or a Category 3 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the general {uml:Classifier} of the {uml:Generalization} for which the {uml:Class} is the specific {uml:Classifier}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension”.
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only”. The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) - the value of the min occurs property must be “1”;
   b. (max occurs property) - the value of the max occurs property must be “1”; and
   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.
6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for an Unstereotyped Category 4 {uml:Class} or a Category 4 {stereotype:ObjectType}]

A mapping shall exist between an unstereotyped Category 4 {uml:Class} or a Category 4 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {uml:Class} is the client {uml:NamedElement}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “restriction”.
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only”. The content model must be a {schema:sequence} for which each of the following is true:

   a. (min occurs property) - the value of the min occurs property must be “1”;
   b. (max occurs property) - the value of the max occurs property must be “1”; and
   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for an Unstereotyped Category 5 {uml:Class} or a Category 5 {stereotype:ObjectType}]

A mapping shall exist between a Category 5 {uml:Class} or a Category 5 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:XSDSimpleContent} for which the {uml:Class} is the client {uml:Classifier}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension.”
4. (attribute uses property) - The mapping for the attribute uses property is as follows:
   a. The XML representation of the {schema:complex type definition} is an `xsd:complexType` {infoset:element}, the children property of which includes an `xsd:simpleContent` {infoset:element}, the children property of which includes an `xsd:extension` {infoset:element}.
   b. The children property of that `xsd:extension` {infoset:element} must include an `xsd:attributeGroup` {infoset:element}, the attributes property of which must include an `xsd:ref` {infoset:attribute} for which the normalized value must indicate “s:SimpleObjectAttributeGroup.”
   c. Except for those explicit members of the attribute uses property specified in a., a mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - A mapping must exist between the content type property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:XSDSimpleContent} for which the {uml:Class} is the client {uml:Classifier}.
6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for an Unstereotyped Category 6 {uml:Class} or a Category 6 {stereotype:ObjectType}]

A mapping shall exist between an unstereotyped Category 6 {uml:Class} or a Category 6 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {uml:Class} is the client {uml:NamedElement}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “restriction”.

4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - A mapping must exist between the content type property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:XSDSimpleContent} for which the {uml:Class} is the client {uml:Classifier}.
6. (annotation property) - The "i:Base" application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for an Unstereotyped Category 7 {uml:Class} or a Category 7 {stereotype:ObjectType}]

A mapping shall exist between a Category 7 {uml:Class} or a Category 7 {stereotype:ObjectType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule "Mapping for a Categorized {uml:Class}" must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the general {uml:Classifier} of the {uml:Generalization} for which the {uml:Class} is the specific {uml:Classifier}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be "extension."
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be the content type property of the general {uml:Classifier} of the {uml:Generalization} for which the {uml:Class} is the specific {uml:Classifier}.
6. (annotation property) - The "i:Base" application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for a Category 1 {stereotype:AdapterType}]

A mapping shall exist between a Category 1 {stereotype:AdapterType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule "Mapping for a Categorized {uml:Class}" must hold.
2. (base type definition property) The base type definition property of the {schema:complex type definition} must be "s:ComplexObjectType."
3. (derivation method property) The derivation method property of the {schema:complex type definition} must be "extension."
4. (attribute uses property) A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) The content type property of the {schema:complex type definition} must be a pair consisting of a content model and "element-only." The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) the value of the min occurs property must be "1";
   b. (max occurs property) the value of the max occurs property must be "1"; and
   c. (term property) a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (annotation property) The “i:Base” application information for the {schema:complex type definition} must indicate “s:Object.”
7. (annotation property) The value of the “i:ExternalAdapterTypeIndicator” application information for the {schema:complex type definition} must be “true.”

[Rule: Mapping for a Category 1 or Category 2 {stereotype:AssociationType}]

A mapping shall exist between a Category 1 or Category 2 {stereotype:AssociationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - The base type definition property of the {schema:complex type definition} must be “s:ComplexObjectType.”
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension.”
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only.” The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) the value of the min occurs property must be “1”;
   b. (max occurs property) the value of the max occurs property must be “1”; and
   c. (term property) a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.
6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate “s:Association.”

[Rule: Mapping for a Category 3 {stereotype:AssociationType}]

A mapping shall exist between a Category 3 {stereotype:AssociationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the general {uml:Classifier} of the {uml:Generalization} for which the {stereotype:AssociationType} is the specific {uml:Classifier}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension.”
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only.” The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) the value of the min occurs property must be “1”;
   b. (max occurs property) the value of the max occurs property must be “1”; and

   c. (term property) a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for a Category 4 {stereotype:AssociationType}]

A mapping shall exist between a Category 4 {stereotype:AssociationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {stereotype:AssociationType} is the client {uml:NamedElement}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “restriction.”
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only.” The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) - the value of the min occurs property must be “1”;
   b. (max occurs property) - the value of the max occurs property must be “1”; and
   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.
6. (annotation property) The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for a Category 1 or Category 2 {stereotype:AugmentationType}]

A mapping shall exist between a Category 1 or Category 2 {stereotype:AugmentationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - The base type definition property of the {schema:complex type definition} must be “s:AugmentationType”.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension”.
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only”. The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) - the value of the min occurs property must be “1”;
   b. (max occurs property) - the value of the max occurs property must be “1”; and

c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (annotation property) - The "i:Base" application information for the {schema:complex type definition} must indicate "s:Object".

[Rule: Mapping for a Category 3 {stereotype:AugmentationType}]

A mapping shall exist between a Category 3 {stereotype:AugmentationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule "Mapping for a Categorized {uml:Class}" must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the general {uml:Classifier} of the {uml:Generalization} for which the {stereotype:AugmentationType} is the specific {uml:Classifier}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be "extension."
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and "element-only." The content model must be a {schema:sequence} for which each of the following is true:
    a. (min occurs property) - the value of the min occurs property must be "1";
    b. (max occurs property) - the value of the max occurs property must be "1"; and
    c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.
6. (annotation property) The "i:Base" application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for a Category 4 {stereotype:AugmentationType}]

A mapping shall exist between a Category 4 {stereotype:AugmentationType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule "Mapping for a Categorized {uml:Class}" must hold.
2. (base type definition property) - A mapping must exist between the base type definition property of the {schema:complex type definition} and the supplier {uml:NamedElement} of the {stereotype:Restriction} for which the {stereotype:AugmentationType} is the client {uml:NamedElement}.
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be "restriction."
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and "element-only." The content model must be a {schema:sequence} for which each of the following is true:
    a. (min occurs property) - the value of the min occurs property must be "1";
    b. (max occurs property) - the value of the max occurs property must be "1"; and

   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.

6. (prohibited substitutions property) - The value of the prohibited substitutions property must be the empty set.
7. (annotation property) - The “i:Base” application information for the {schema:complex type definition} must indicate the base type definition for the {schema:complex type definition}.

[Rule: Mapping for a Category 1 {stereotype:MetadataType}]

A mapping shall exist between a Category 1 {stereotype:MetadataType} and a {schema:complex type definition} if and only if each of the following is true:

1. The rule “Mapping for a Categorized {uml:Class}” must hold.
2. (base type definition property) - The base type definition property of the {schema:complex type definition} must be “s:MetadataType.”
3. (derivation method property) - The derivation method property of the {schema:complex type definition} must be “extension.”
4. (attribute uses property) - A mapping must exist between the explicit members of the attribute uses property of the {schema:complex type definition} and the attribute use set for the {uml:Class}.
5. (content type property) - The content type property of the {schema:complex type definition} must be a pair consisting of a content model and “element-only.” The content model must be a {schema:sequence} for which each of the following is true:
   a. (min occurs property) - the value of the min occurs property must be “1”;
   b. (max occurs property) - the value of the max occurs property must be “1”; and
   c. (term property) - a mapping must exist between the term property of the {schema:sequence} and the model group for the {uml:Class}.
6. (annotation property) The “i:Base” application information for the {schema:complex type definition} must indicate “s:Object.”
7. (annotation property) For each {stereotype:MetadataApplication} for which the {stereotype:MetadataType} is the client {uml:NamedElement},
   a. exactly one “i:AppliesTo” application information for the {schema:complex type definition} must exist, and
   b. a mapping must exist between the {schema:component} indicated by the “i:AppliesTo” application information and the supplier {uml:NamedElement}.
8. (annotation property) - For each “i:AppliesTo” application information for the {schema:complex type definition},
   a. exactly one {stereotype:MetadataApplication} for which the {stereotype:MetadataType} is the client {uml:NamedElement} must exist, and
   b. a mapping must exist between the {schema:component} indicated by the “i:AppliesTo” application information and the supplier {uml:NamedElement}.

B.2.9.7 Mapping a {schema:attribute declaration}

[Definition: Attribute Declaration {stereotype:XSDProperty}]

An attribute declaration {uml:Property} is:

1. a Category 1 {stereotype:XSDProperty} for which the value of the kind attribute is “attribute”,or

2. a Category 2 {stereotype:XSDProperty} for which the value of the kind attribute is “attribute”.

[Rule: Mapping for an Attribute Declaration {stereotype:XSDProperty}]

A mapping shall exist between a {schema:attribute declaration} and an attribute declaration {stereotype:XSDProperty} if and only if each of the following is true:

1. (name property) - The value of the name property of the {schema:attribute declaration} must equal the value of the name attribute of the {stereotype:XSDProperty}.
2. (target namespace property) - The value of the target namespace property must equal the value of the targetNamespace attribute of the {stereotype:Namespace} that is the namespace of the class of the {stereotype:XSDProperty}.
3. (type definition property) - A mapping must exist between the type definition property of the {schema:attribute declaration} and the type of the {stereotype:XSDProperty}.
4. (scope property) - The scope property of the {schema:attribute declaration} must be “global.”
5. (value constraint property) - Exactly one of the following must be true:
   a. The fixed attribute of the {stereotype:XSDProperty} is absent and the value constraint property of the {schema:attribute declaration} is absent.
   b. The fixed attribute of the {stereotype:XSDProperty} is present; the value constraint property of the {schema:attribute declaration} is a pair consisting of “fixed” and a value; and that value equals the value of the fixed attribute of the {stereotype:XSDProperty}.
6. (annotation property) - A mapping must exist between the documentation for the {schema:attribute declaration} the documentation for the {stereotype:XSDProperty}.

B.2.9.8 Mapping a {schema:element declaration}

[Definition: Element Declaration {uml:Property}]

An element declaration {uml:Property} is

1. any unstereotyped Category 1 {uml:Property}
2. any unstereotyped Category 2 {uml:Property}
3. any unstereotyped Category 4 {uml:Property}
4. a Category 1 {stereotype:XSDProperty} for which the value of the kind attribute is “element”
5. a Category 2 {stereotype:XSDProperty} for which the value of the kind attribute is “element” or
6. a Category 4 {stereotype:XSDProperty} for which the value of the kind attribute is “element”

[Definition: Reference Element Declaration {uml:Property}]

A reference element declaration {uml:Property} is any Element Declaration {uml:Property} for which the value of the name attribute ends with “Reference.”

[Definition: Content Element Declaration {uml:Property}]

A content element declaration {uml:Property} is any element declaration {uml:Property} that is not a reference element declaration {uml:Property}.

[Rule: Mapping for an Element Declaration {uml:Property}]

A mapping shall exist between a {schema:element declaration} and an element declaration {uml:Property} only if each of the following is true:

1. (name property) - The value of the name property of the {schema:element declaration} must equal the value of the name attribute of the {uml:Property}.
2. (target namespace property) - The value of the target namespace property must equal the value of the targetNamespace attribute of the {stereotype:Namespace} that is the namespace of the class of the {uml:Property}.
3. (scope property) - The scope property of the {schema:element declaration} must be “global.”
4. (identity-constraint definitions property) - The identity-constraint definitions property must be the empty set.
5. (substitution group exclusions property) - The substitution group exclusions property of the {schema:element declaration} must be the empty set.
6. (disallowed substitutions property) - The disallowed substitutions property of the {schema:element declaration} must be the empty set.
7. (abstract property) - The value of the abstract property of the {schema:element declaration} must equal the value of the isDerivedUnion attribute of the {uml:Property}.
8. (annotation property) - A mapping must exist between the documentation for the {schema:element declaration} and the documentation for the {uml:Property}.

The above rule is necessary for the following rules:

1. Mapping for an Unstereotyped Reference Element Declaration {uml:Property}
2. Mapping for a Reference Element Declaration {stereotype:XSDProperty}
3. Mapping for an Unstereotyped Content Element Declaration {uml:Property}
4. Mapping for a Content Element Declaration {stereotype:XSDProperty}

- but not sufficient in itself to specify a mapping

[Rule: Mapping for an Unstereotyped Reference Element Declaration {uml:Property}]

A mapping shall exist between a {schema:element declaration} and an unstereotyped reference element declaration {uml:Property} if and only if each of the following is true:

1. The rule “Mapping for an Element Declaration {uml:Property}” must hold.
2. (type definition property) - The type definition property of the {schema:element declaration} must be “s:ReferenceType.”
3. (value constraint property) - The value constraint property of the {schema:element declaration} must be absent.
4. (nillable property) - The value of the nillable property of the {schema:element declaration} must be false.
5. (substitution group affiliation property) - The mapping for the substitution group affiliation property is as follows:
   a. If there is exactly one subsettedProperty, a mapping must exist between the substitution group affiliation of the {schema:element declaration} and the subsettedProperty of the {uml:Property}.
   b. If there is not any subsettedProperty and if the type is a categorized {stereotype:AugmentationType}, the substitution group affiliation must be s:Augmentation.
6. (annotation property) - A mapping must exist between the {schema:component} indicated by the “i:ReferenceTarget” application information for the {schema:element declaration} and the type of the {uml:Property}.

[Rule: Mapping for a Reference Element Declaration {stereotype:XSDProperty}]

A mapping shall exist between a {schema:element declaration} and a reference element declaration {stereotype:XSDProperty} if and only if each of the following is true:

1. The rule "Mapping for an Element Declaration {uml:Property}" must hold.
2. (type definition property) - The type definition property of the {schema:element declaration} must be "s:ReferenceType."
3. (value constraint property) - The value constraint property of the {schema:element declaration} must be absent.
4. (nillable property) - The value of the nillable property of the {schema:element declaration} must equal the value of the nillable attribute of the {stereotype:XSDProperty}.
5. (substitution group affiliation property) - A mapping must exist between the substitution group affiliation of the {schema:element declaration} and the subsettedProperty of the {uml:Property}.
6. (annotation property) - A mapping must exist between the {schema:component} indicated by the "i:ReferenceTarget" application information for the {schema:element declaration} and the type of the {stereotype:XSDProperty}.

[Rule: Mapping for an Unstereotyped Content Element Declaration {uml:Property}]

A mapping shall exist between a {schema:element declaration} and an unstereotyped content element declaration {uml:Property} if and only if each of the following is true:

1. The rule "Mapping for an Element Declaration {uml:Property}" must hold.
2. (type definition property) - A mapping must exist between the type definition property of the {schema:element declaration} and the type of the {uml:Property}.
3. (value constraint property) - The value constraint property of the {schema:element declaration} must be absent.
4. (nillable property) - The value of the nillable property of the {schema:element declaration} must be false.
   a. (substitution group affiliation property) - If there exactly one subsettedProperty, a mapping must exist between the substitution group affiliation of the {schema:element declaration} and the subsettedProperty of the {uml:Property}.
   b. (substitution group affiliation property) - If there not any subsettedProperty and if the type is a categorized {stereotype:AugmentationType}, the substitution group affiliation must be s:Augmentation.
5. (annotation property) - For each {stereotype:AugmentationApplication} for which the {uml:Property} is the client {uml:NamedElement}
   a. exactly one "i:AppliesTo" application information for the {schema:element declaration} must exist; and
   b. a mapping must exist between the {schema:component} indicated by the "i:AppliesTo" application information and the supplier {uml:NamedElement}.
6. (annotation property) - For each "i:AppliesTo" application information for the {schema:element declaration}
   a. exactly one {stereotype:AugmentationApplication} for which the {uml:Property} is the client {uml:NamedElement} must exist; and
   b. a mapping must exist between the {schema:component} indicated by the "i:AppliesTo" application information and the supplier {uml:NamedElement}.

[Rule: Mapping for a Content Element Declaration {stereotype:XSDProperty}]

A mapping shall exist between a {schema:element declaration} and a content element declaration {stereotype:XSDProperty} if and only if each of the following is true:

1. The rule “Mapping for an Element Declaration {uml:Property}” must hold.
2. (type definition property) - A mapping must exist between the type definition property of the {schema:element declaration} and the type of the {uml:Property}.
3. (value constraint property) - Exactly one of the following must be true:
    a. The fixed attribute of the {stereotype:XSDProperty} is absent and the value constraint property of the {schema:element declaration} is absent.
    b. The fixed attribute of the {stereotype:XSDProperty} is present; the value constraint property of the {schema:element declaration} is a pair consisting of “fixed” and a value; and that value equals the value of the fixed attribute of the {stereotype:XSDProperty}.
4. (nillable property) - The value of the nillable property of the {schema:element declaration} must equal the value of the nillable attribute of the {stereotype:XSDProperty}.
    a. (substitution group affiliation property) - If there exactly one subsettedProperty, a mapping must exist between the substitution group affiliation of the {schema:element declaration} and the subsettedProperty of the {uml:Property}.
    b. (substitution group affiliation property) - If there not any subsettedProperty and if the type is a categorized {stereotype:AugmentationType}, the substitution group affiliation must be s:Augmentation.
5. (annotation property) - For each {stereotype:AugmentationApplication} for which the {uml:Property} is the client {uml:NamedElement}
    a. exactly one “i:AppliesTo” application information for the {schema:element declaration} must exist; and
    b. a mapping must exist between the {schema:component} indicated by the “i:AppliesTo” application information and the supplier {uml:NamedElement}.
6. (annotation property) - For each “i:AppliesTo” application information for the {schema:element declaration}
    a. exactly one {stereotype:AugmentationApplication} for which the {uml:Property} is the client {uml:NamedElement} must exist; and
    b. a mapping must exist between the {schema:component} indicated by the “i:AppliesTo” application information and the supplier {uml:NamedElement}.

B.2.9.9 Mapping for a {schema:schema}

[Definition: Import for a {uml:DataType}]

The import for a {uml:DataType} is the {stereotype:Namespace} that is the namespace of the {uml:DataType}.

[Definition: Import for a {uml:Class}]

The import for a {uml:Class} is the {stereotype:Namespace} that is the namespace of the {uml:Class}.

[Definition: Import for a {uml:Property}]

The import for a {uml:Property} is the {stereotype:Namespace} that is the namespace of the class of the {uml:Property}.

[Definition: Import set for a {stereotype:Namespace}]

An import set for a {stereotype:Namespace} (call it the importing {stereotype:Namespace}) is a set of {stereotype:Namespace} constructed as follows:

1. For each unstereotyped categorized {uml:Class}, each stereotyped {uml:Class}, each {stereotype:PropertyHolder}, and each {stereotype:Choice} in the importing {stereotype:Namespace}:
   a. For each element use {uml:Property} in the {uml:Class} that is the client {uml:NamedElement} of a {stereotype:References}:
      i. If the supplier {uml:NamedElement} of the {stereotype:References} is a {stereotype:Namespace} and if that {stereotype:Namespace} is not the importing {stereotype:Namespace}, add the {stereotype:Namespace} that is the supplier {uml:NamedElement}.
      ii. If the supplier {uml:NamedElement} of the {stereotype:References} is a element declaration {uml:Property} and if the import for that element declaration {uml:Property} is not the importing {stereotype:Namespace}, add the import for the element declaration {uml:Property}.
   b. For each attribute use {uml:Property} in the {uml:Class} that is the client {uml:NamedElement} of a {stereotype:References}:
      i. If the supplier {uml:NamedElement} of the {stereotype:References} is a {stereotype:Namespace} and if that {stereotype:Namespace} is not the importing {stereotype:Namespace}, add the {stereotype:Namespace} that is the supplier {uml:NamedElement}.
      ii. If the supplier {uml:NamedElement} of the {stereotype:References} is an attribute declaration {uml:Property} and if the import for that attribute declaration {uml:Property} is not the importing {stereotype:Namespace}, add the import for the attribute declaration {uml:Property}.
   c. For each element declaration {uml:Property} in the {uml:Class}:
      i. If the import for the type of the {uml:Property} is not the importing {stereotype:Namespace}, add the import for the type of the {uml:Property}.
   d. For each attribute declaration {uml:Property} in the {uml:Class}:
      i. If the import for the type of the {uml:Property} is not the importing {stereotype:Namespace}, add the import for the type of the {uml:Property}.

[Definition: Conformant Import]

A conformant import is an import that is also a conformant {stereotype:Namespace}.

[Definition: Non-Conformant Import]

A non-conformant import is an import that is also a non-conformant {stereotype:Namespace}.

[Rule: Mapping for a Conformant Import for a {stereotype:Namespace}]

A mapping shall exist between a conformant import for a {stereotype:Namespace} and an `xsd:import` {infoset:element} in the children property of an `xsd:schema` {infoset:element} if and only if the following is true:

1. (`xsd:namespace` {infoset:attribute}) - The value of the normalized value property of the `xsd:namespace` {infoset:attribute} must equal the value of the targetNamespace attribute of the {stereotype:Namespace}.

[Rule: Mapping for a Non-Conformant Import for a {stereotype:Namespace}]

A mapping shall exist between a non-conformant import for a {stereotype:Namespace} and an `xsd:import` {infoset:element} in the children property of the `xsd:schema` {infoset:element} if and only if each of the following is true:

1. (`xsd:namespace` {infoset:attribute}) - The value of the normalized value property of the `xsd:namespace` {infoset:attribute} must equal the value of the targetNamespace attribute of the {stereotype:Namespace}.

2. (annotation property) - A mapping must exist between the documentation for the `xsd:import` {infoset:element} and the documentation for the non-conformant import.
3. (annotation property) - The “i:ConformantIndicator” application information for the `xsd:import` {infoset:element} must be “false.”

[Rule: Mapping for the Import Set for a {stereotype:Namespace}]

A mapping shall exist between the import set for a {stereotype:Namespace} and the `xsd:import` {infoset:element}s in the children property of the `xsd:schema` {infoset:element} if any only if each of the following are true:

1. For each import in the import set for a {stereotype:Namespace}, a mapping must exist between the import and exactly one `xsd:import` {infoset:element} in the children property.
2. For each `xsd:import` {infoset:element} in the children property, a mapping must exist between the `xsd:import` {infoset:element} and exactly one import in the import set for the {stereotype:Namespace}.

[Rule: Mapping for a Conformant {stereotype:Namespace}]

A mapping shall exist between a {schema:schema} and a conformant {stereotype:Namespace} if and only if each of the following is true:

1. (type definitions property) - For each {schema:simple type definition} in the type definitions property of the {schema:schema}, a mapping must exist between the {schema:simple type definition} and exactly one {uml:DataType} for which the namespace is the {stereotype:Namespace}.
2. (type definitions property) - For each {uml:DataType} for which the namespace is the {stereotype:Namespace}, a mapping must exist between the {uml:DataType} and exactly one {schema:simple type definition} in the type definitions property of the {schema:schema}.
3. (type definitions property) - For each {schema:complex type definition} in the type definitions property of the {schema:schema}, a mapping must exist between the {schema:complex type definition} and exactly one {uml:Class} for which the namespace is the {stereotype:Namespace}.
4. (type definitions property) - For each {uml:Class} for which the namespace is the {stereotype:Namespace}, a mapping must exist between the {uml:Class} and exactly one {schema:complex type definition} in the type definitions property of the {schema:schema}.
5. (element declarations property) - For each {schema:element declaration} in the element declarations property of the {schema:schema}, a mapping must exist between the {schema:element declaration} and exactly one element declaration {uml:Property} for which the import is the {stereotype:Namespace}.
6. (element declarations property) - For each element declaration {uml:Property} for which import is the {stereotype:Namespace}, a mapping must exist between the {uml:Property} and exactly one {schema:element declaration} in the element declarations property of the {schema:schema}.
7. (attribute declarations property) - For each {schema:attribute declaration} in the attribute declarations property of the {schema:schema}, a mapping must exist between the {schema:attribute declaration} and exactly one attribute declaration {uml:Property} for which the import is the {stereotype:Namespace}.
8. (attribute declarations property) - For each attribute declaration {uml:Property} for which import is the {stereotype:Namespace}, a mapping must exist between the {uml:Property} and exactly one {schema:attribute declaration} in the attribute declarations property of the {schema:schema}.
9. (annotation property) - A mapping must exist between the documentation for the {schema:schema} and the documentation for the {stereotype:Namespace}.

10. (annotation property) - The “i:ConformantIndicator” application information for the {schema:schema} must be “true.”

11. (`xsd:import` {infoset:element}s) - In the XML representation of the {schema:schema}, a mapping must exist between the `xsd:import` {infoset:element}s in the children property of the `xsd:schema` and the import set for the {stereotype:Namespace}.

12. (`xsd:targetNamespace` {infoset:attribute}) - In the XML representation of the {schema:schema}

    a. the attributes property of the `xsd:schema` {infoset:element} must include an `xsd:targetNamespace` {infoset:attribute}; and

    b. the normalized value of the {infoset:attribute} must equal the value of the targetNamespace property of the {stereotype:Namespace}.

13. (`xsd:version` {infoset:attribute}) - In the XML representation of the {schema:schema}

    a. the attributes property of the `xsd:schema` {infoset:element} must include an `xsd:version` {infoset:attribute}; and

    b. the normalized value of the {infoset:attribute} must equal the value of the version property of the {stereotype:Namespace}.

Annex C
Machine Readable Artifacts

(normative)

C.1 Normative

NIEM-UML includes three UML models, the normative XMI for which may be referenced using the following standard URIs:

- *NIEM UML Profile*
  `http://www.omg.org/spec/NIEM-UML/20130801/NIEM-UML-Profile.xmi`
- *XML Primitive Types Library*
  `http://www.omg.org/spec/NIEM-UML/20130801/XMIPrimitiveTypes.xmi`
- *Reference Vocabulary Library* – This consists of several XMI files located in the following directory, containing NIEM-UML models of the various NIEM reference schema. They are provided in separate files to allow a user to easily access only the specific domain areas of interest.
  `http://www.omg.org/spec/NIEM-UML/20130801/NIEM-Reference/NIEM-Reference`

The NIEM UML Profile model contains the overall NIEM UML Profile and the three sub-profiles, as specified in Clause 8. Each of these have specified namespace prefixes and URIs (as recommended in section 18.3.7 of [UML]). The prefix for each profile is the same as the name of the profile and the URI is as follows:

- *NIEM_UML_Profile*

`http://www.omg.org/spec/NIEM-UML/20130801`

- *NIEM_Common_Profile*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEM_Common_Profile`

- *NIEM_PIM_Profile*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEM_PIM_Profile`

- *NIEM_PSM_Profile*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEM_PSM_Profile`

- *Model_Package_Description_Profile*

`http://www.omg.org/spec/NIEM-UML/20130801/Model_Package_Description_Profile`

NIEM-UML also includes four normative QVT transformations, as described in Clause 9, which may be found at the following URIs:

- *NIEM PIM to NIEM PSM*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEMpim2psm.qvto`

- *NIEM PSM to NIEM-Conforming XML Schema* `http://www.omg.org/spec/NIEM-UML/20130801/NIEMpsm2xsd.qvto`

- *NIEM MPD Model to NIEM MPD Artifact*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEMmpdmodel2artifact.qvto`

- *NIEM MPD Artifact to NIEM MPD Model*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEMmpdartifact2model.qvto`

These transformations in turn use the following common transformations:

- *NIEM Globals*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEMglobals.qvto`

- *NIEM Platform Binding*

`http://www.omg.org/spec/NIEM-UML/20130801/NIEMplatformBinding.qvto`

C.2 Non-Normative

The following artifacts are used in the normative QVT specification of transformations between a NIEM PSM and MPD artifacts, as discussed in Clause 9. However, they are not considered a normative part of NIEM-UML itself.

- *XML Schema Metamodel (based on Clause 10 of [XMI])*

`http://www.omg.org/spec/NIEM-UML/20130801/Nonnormative/XSD.emof`

- *MPD Catalog Metamodel*

`http://www.omg.org/spec/NIEM-UML/20130801/Nonnormative/mpd.catalog.emof`