

# UML Profile for NIEM 3 (NIEM-UML-3)

*FTF Beta 2*

**OMG Document Number:** dtc/2016-05-02

**Normative Machine Consumable Files:**

---

<http://www.omg.org/spec/NIEM-UML/20150201/NIEM-UML-Profile.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/XmlPrimitiveTypes.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-adapters-edxl-cap.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-adapters-edxl-de.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-adapters-edxl-have.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-adapters-geospatial.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-ansi\\_d20.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-ansi_d20.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-apco\\_event.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-apco_event.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-atf.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-canada\\_post.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-canada_post.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-cbrncl.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-census\\_commodity.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-census_commodity.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-census\\_uscounty.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-census_uscounty.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-core\\_misc.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-core_misc.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dea\\_ctsub.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dea_ctsub.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dod\\_jcs-pub2.0.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dod_jcs-pub2.0.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dol\\_soc.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dol_soc.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dot\\_hazmat.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-dot_hazmat.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-edxl\\_have.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-edxl_have.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-edxl\\_rm.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-edxl_rm.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi\\_ncic.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi_ncic.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi\\_ndex.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi_ndex.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi\\_ucr.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fbi_ucr.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips\\_10-4.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips_10-4.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips\\_5-2.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips_5-2.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips\\_6-4.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-fips_6-4.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-hl7.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso\\_3166-1.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso_3166-1.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso\\_4217.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso_4217.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso\\_639-3.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-iso_639-3.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-it\\_codes.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-it_codes.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-mmucc.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga\\_datum.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga_datum.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga\\_genc.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga_genc.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga\\_vdatum.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nga_vdatum.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-nlets.xmi>  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-occ\\_facility.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-occ_facility.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-pmise\\_sar.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-pmise_sar.xmi)  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-unece\\_rec20.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-unece_rec20.xmi)

[http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-usps\\_states.xmi](http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-usps_states.xmi)  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-codes-xCard.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-domains-biometrics.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-domains-emergencyManagement.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-domains-infrastructureProtection.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-domains-screening.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-external-cap.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-external-de.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-external-have.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-external-ogc.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-external-xml.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMReference/NIEM-Reference-niem-core.xmi>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMpim2psm.qvto>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMpsm2xsd.qvto>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMmpdmodel2artifact.qvto>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMmpdartifact2model.qvto>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMglobals.qvto>  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMplatformBinding.qvto>

---

Copyright © 2015 Data Access Technologies (A Division of Model Driven Solutions)

Copyright © 2012 Georgia Tech Research Institute (GTRI)

Copyright © 2012 Microsoft

Copyright © 2015 Object Management Group (OMG)

Copyright © 2012 Visumpoint

#### USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

#### LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

#### PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

#### GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

#### DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE

COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

#### RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227- 7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

#### TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, IMMT™, MOF™, OMG Interface Definition Language (IDL)™, and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

#### COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

# Table of Contents

Preface .....	1
<b>1 Scope.....</b>	<b>3</b>
1.1 NIEM-UML Background .....	3
1.2 Intended Users of NIEM-UML.....	3
1.3 NIEM-UML Profiles .....	4
1.4 NIEM-UML Transformations.....	5
1.5 NIEM-UML Libraries .....	5
<b>2 Conformance .....</b>	<b>6</b>
2.1 Conformance Points.....	6
2.2 NIEM Platform Independent Model (PIM) .....	6
2.3 NIEM Platform Specific Model (PSM) .....	6
2.4 NIEM Model Package Description (MPD) Model .....	7
2.5 NIEM PIM to NIEM PSM Transform .....	7
2.6 NIEM PSM to NIEM-Conforming XML Schema Transform .....	7
2.7 NIEM MPD Model to NIEM MPD Artifact Transform .....	7
2.8 NIEM MPD Artifact to NIEM MPD Model Transform .....	7
2.9 Tool Conformance .....	7
<b>3 Normative References.....</b>	<b>9</b>
<b>4 Terms and Definitions.....</b>	<b>10</b>
4.1 Definitions .....	10
4.2 Acronyms.....	14
<b>5 Symbols .....</b>	<b>15</b>
<b>6 Additional Information.....</b>	<b>16</b>
6.1 Acknowledgements.....	16
6.2 Proof of Concept .....	16
6.3 NIEM-UML Introduction and Concepts.....	16
6.3.1 Background.....	16
6.3.2 NIEM-UML Goals.....	17
6.3.3 Understanding NIEM-UML and Model Driven Architecture (MDA).....	17
<b>7 NIEM-UML Modeling Guide.....</b>	<b>20</b>
7.1 Overview .....	20
7.1.1 Introduction.....	20
7.1.2 Platform Independent Perspective .....	21
7.1.3 Platform Specific Perspective .....	26
7.1.4 Model Packaging Perspective .....	30
7.2 Modeling Namespaces .....	35
7.2.1 Namespaces .....	35
7.2.2 NIEM Names .....	38
7.2.3 Local Vocabularies .....	39
7.3 Modeling Complex Types .....	40
7.3.1 Complex Types .....	40
7.3.2 Object Types .....	43
7.3.3 Role Types .....	45
7.3.4 Association Types.....	49
7.3.5 Metadata Types.....	52
7.3.6 Augmentation Types.....	55
7.3.7 Adapter Types.....	58
7.4 Modeling Simple Types .....	60
7.4.1 Simple Types .....	60
7.4.2 Primitive Types.....	62

7.4.3	Code Types .....	66
7.4.4	Unions.....	69
7.4.5	Lists .....	71
7.5	Modeling Properties.....	73
7.5.1	Properties .....	73
7.5.2	Property Holders and Property References .....	77
7.5.3	Substitution Groups .....	81
7.5.4	Representations.....	83
7.5.5	Choice Groups .....	85
7.6	Packaging Models.....	86
7.6.1	Reference and Subset Models .....	86
7.6.2	Model Package Descriptions.....	91
7.7	Detailed Modeling Design Rules .....	93
7.7.1	Design Rules Rationale.....	93
7.7.2	Simple Restrictions.....	93
7.7.3	Complex Restrictions.....	94
7.7.4	Business Rules .....	95
<b>8</b>	<b>NIEM-UML Profile Reference.....</b>	<b>97</b>
8.1	Overview .....	97
8.2	Profile : NIEM_Common_Profile.....	98
8.2.1	Overview .....	98
8.2.2	<Stereotype> AdapterType .....	98
8.2.3	<Stereotype> AssociationType.....	100
8.2.4	<Stereotype> AugmentationType .....	101
8.2.5	<Stereotype> Choice .....	102
8.2.6	<Stereotype> Deprecated.....	103
8.2.7	<Stereotype> Documentation .....	103
8.2.8	<Stereotype> List.....	104
8.2.9	<Stereotype> LocalTerm .....	106
8.2.10	<Stereotype> LocalVocabulary .....	107
8.2.11	<Stereotype> MetadataApplication .....	107
8.2.12	<Stereotype> MetadataType .....	108
8.2.13	<Stereotype> Namespace .....	109
8.2.14	<Stereotype> NIEMType .....	119
8.2.15	<Stereotype> ObjectType .....	123
8.2.16	<Stereotype> PropertyHolder .....	131
8.2.17	<Stereotype> References .....	131
8.2.18	<Stereotype> Representation .....	132
8.2.19	<Stereotype> Restriction .....	134
8.2.20	<Stereotype> Union .....	134
8.2.21	<Stereotype> UnionOf.....	138
8.3	Profile : NIEM_PIM_Profile .....	139
8.3.1	Overview .....	139
8.3.2	<Stereotype> Augments .....	139
8.3.3	<Stereotype> InformationModel .....	140
8.3.4	<Stereotype> ReferenceName .....	161
8.3.5	<Stereotype> RoleOf .....	161
8.3.6	<Stereotype> RolePlayedBy .....	162
8.3.7	<Stereotype> Subsets.....	163
8.3.8	<Enumeration> DefaultPurposeCode .....	164
8.4	Profile : NIEM_PSM_Profile .....	165
8.4.1	Overview .....	165
8.4.2	<Stereotype> XSDAnyProperty .....	165
8.4.3	<Stereotype> XSDDeclaration .....	166
8.4.4	<Stereotype> XSDProperty .....	166

8.4.5 <Stereotype> XSDRepresentationRestriction.....	195
8.4.6 <Stereotype> XSDSimpleContent .....	195
8.4.7 <Enumeration> XSDProcessContentsCode.....	196
8.4.8 <Enumeration> XSDPropertyKindCode .....	196
8.4.9 <Enumeration> XSDWhiteSpaceCode.....	196
8.5 Profile : Model_Package_Description_Profile .....	198
8.5.1 Overview .....	198
8.5.2 <Stereotype> ApplicationInfo .....	200
8.5.3 <Stereotype> BusinessRulesArtifact .....	200
8.5.4 <Stereotype> ChangeInformationType .....	201
8.5.5 <Stereotype> ChangeLogType .....	201
8.5.6 <Stereotype> ConformanceAssertion .....	202
8.5.7 <Stereotype> ConformanceReport .....	202
8.5.8 <Stereotype> Documentation .....	202
8.5.9 <Stereotype> ExtensionSchemaDocument.....	202
8.5.10 <Stereotype> ExternalSchemaDocument .....	203
8.5.11 <Stereotype> File.....	203
8.5.12 <Stereotype> FileType .....	203
8.5.13 <Stereotype> IEPSSampleXMLDocument .....	204
8.5.14 <Stereotype> ModelPackageDescriptionRelationship.....	204
8.5.15 <Stereotype> MPDChangeLog.....	204
8.5.16 <Stereotype> qualifiedName .....	204
8.5.17 <Stereotype> ReadMe .....	205
8.5.18 <Stereotype> ReferenceSchemaDocument.....	206
8.5.19 <Stereotype> RelaxNGSchema .....	206
8.5.20 <Stereotype> RequiredFile .....	206
8.5.21 <Stereotype> SchematronSchema .....	206
8.5.22 <Stereotype> SubsetSchemaDocument .....	206
8.5.23 <Stereotype> Wantlist .....	207
8.5.24 <Stereotype> XMLCatalog.....	207
8.5.25 <Stereotype> XMLSchemaDocument .....	207
8.5.26 <Artifact> ArtifactOrArtifactSet .....	207
8.5.27 <Artifact> ConformanceTargetType .....	207
8.5.28 <Artifact> ConstraintSchemaDocumentSet.....	208
8.5.29 <Artifact> ContactInformationType .....	208
8.5.30 <Artifact> DescribedType .....	208
8.5.31 <Artifact> EntityRepresentation .....	209
8.5.32 <Artifact> EXIXMLSchemaType .....	209
8.5.33 <Artifact> FileSet .....	209
8.5.34 <Artifact> FileSetType .....	209
8.5.35 <Artifact> IEPConformanceTargetType .....	210
8.5.36 <Artifact> ModelPackageDescription .....	210
8.5.37 <Artifact> OrganizationType .....	223
8.5.38 <Artifact> PersonType .....	223
8.5.39 <Artifact> QualifiedNamesType .....	223
8.5.40 <Artifact> RelaxNGValidationType .....	223
8.5.41 <Artifact> SchemaDocumentSet .....	224
8.5.42 <Artifact> SchemaDocumentSetType .....	224
8.5.43 <Artifact> SchematronValidationType .....	224
8.5.44 <Artifact> TextRuleType .....	225
8.5.45 <Artifact> ValidityConstraintType.....	225
8.5.46 <Artifact> ValidityConstraintWithContextType .....	225
8.5.47 <Artifact> ValidityContextType.....	225
8.5.48 <Artifact> XMLSchemaType.....	226
8.5.49 <Artifact> XPathType .....	226
8.5.50 <Enumeration> ChangeCodeSimpleType .....	226

8.5.51 <Enumeration> ModelPackageDescriptionClassCode .....	227
8.5.52 <Enumeration> RelationshipCode.....	228
<b>9 NIEM-UML Transformation Reference .....</b>	<b>230</b>
9.1 Introduction .....	230
9.1.1 NIEM Provisioning Context .....	230
9.1.2 Transformation Notation.....	232
9.1.3 Platform Binding .....	234
9.2 NIEM PIM to NIEM PSM .....	235
9.3 NIEM PSM to NIEM-Conforming XML Schema.....	245
9.4 NIEM MPD Model to NIEM MPD Artifact.....	259
9.5 NIEM MPD Artifact to NIEM MPD Model.....	261
<b>10 NIEM-UML PIM Example (informative) .....</b>	<b>281</b>
10.1 Example Description .....	281
10.2 Organization of NIEM Information Models and Classes .....	281
10.3 High-Level Design.....	282
10.4 Documenting Elements.....	282
10.5 UML Associations Defining NIEM Properties.....	283
10.6 UML Enumerations Defining NIEM Code Types .....	283
10.7 Properties of Pet.....	284
10.8 Properties Using Classes as Their Types .....	284
10.9 Finding Classes in Reference Namespaces .....	285
10.10 Defining a subset namespace with «Subsets» .....	286
10.11 Reusing Person.....	288
10.12 Reusing Person Name .....	289
10.13 Contact Information .....	290
10.14 Augmenting Telephone Number.....	292
10.15 Using a NIEM Association for Contact Information .....	292
10.16 Pet Adoptions as a Kind of Activity.....	293
10.17 Pet Adoption Centers as a Role of an Organization .....	295
10.18 Putting Together the High-Level Picture .....	296
10.19 Exchange Message .....	297
10.20 Primitive types .....	297
Annex A 10.21 The Pet Adoption IEPD Model .....	298
<b>Machine Readable Artifacts.....</b>	<b>300</b>
A.1 Normative .....	300
A.2 Informative .....	302
A.3 Ancillary .....	Error! Bookmark not defined.

# Preface

## 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/IOP
- Data Distribution Services
- Specialized CORBA IDL/Language Mapping Specifications

#### Modeling and Metadata Specifications

- UML, MOF, CWM, XMI
- UML Profile 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 Specifications Catalog cited above or by contacting the Object Management Group, Inc. at: OMG Headquarters 140 Kendrick Street Building A, Suite 300 Needham, MA 02494 USA Tel: +1-781-444-0404 Fax: +1-781-444-0320 Email: [pubs@omg.org](mailto: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 section 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](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 which support mission data sharing and exchange requirements. By providing a common vocabulary and mature framework to facilitate information exchange, NIEM enables communities to "speak the same language" as they share, exchange, accept, and translate information efficiently.

NIEM is currently defined in terms of the eXtensible Markup Language (XML), XML Schema (XSD) and the normative NIEM platform specifications which include the NIEM Naming and Design Rules (NDR) and the NIEM Model Package Description (MPD) Specification. 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/>.

In 2013 the NIEM-PMO instituted the NIEM-3 program, intended as the next major revision of NIEM. In 2014 NIEM-3 was defined and approved. NIEM-UML-3 updates the NIEM-UML specification for NIEM-3 conformance. More resources and information relating to NIEM-UML may be found on the NIEM GITHUB site at <https://github.com/NIEM/NIEM-UML/>.

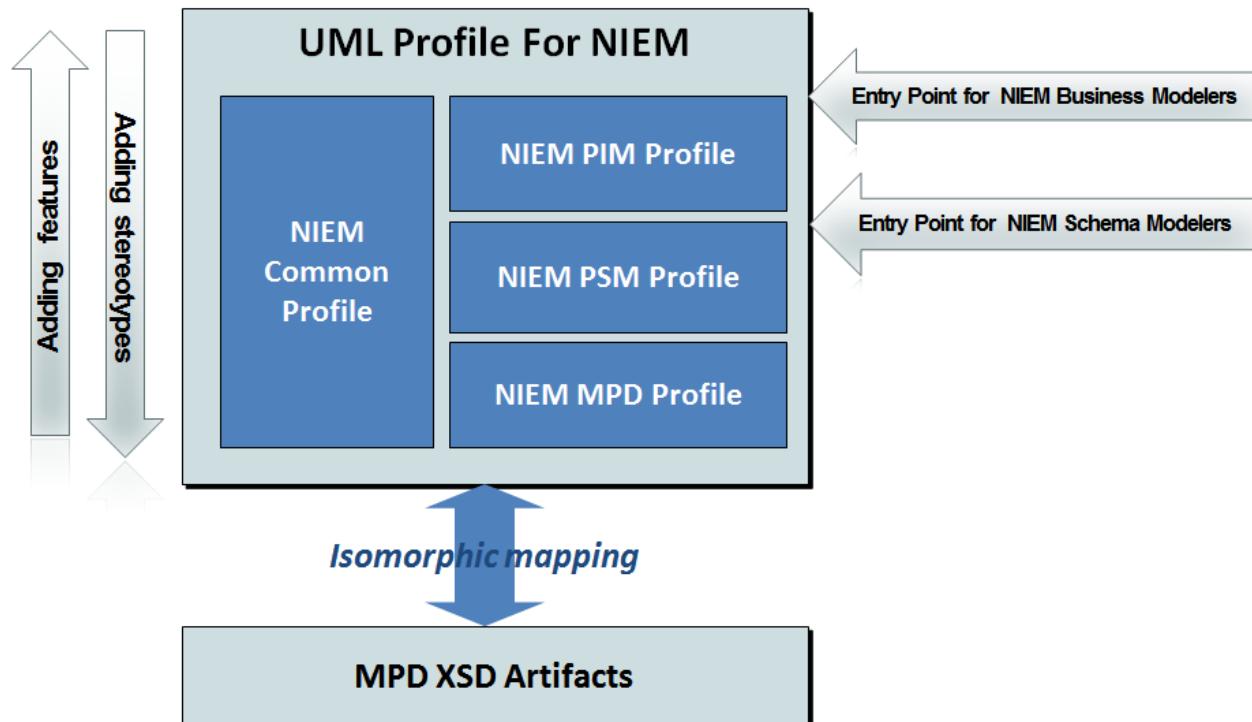
## 1.2 Intended Users of NIEM-UML

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 2.4.1 constructs that are extended by UML stereotypes. The subset identifies those NIEM 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 that have no analogous representation in UML. All NIEM-UML models use the standard XMI exchange format specified for UML 2.4.1 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 3.0 Release. The current version of this model library is the normative representation for the NIEM 3.0 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 (not to be confused with NIEM conformance targets, which are explained in [NIEM-NDR] [Section 4](#)):

- 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)

Subclause 8.3 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)

Subclause 8.4 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 2.5, Subclause 12.3.3: 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; and
- 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 Subclause 9.3 of this specification and as further discussed in Subclause 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**

Subclause 8.5 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 Subclause 2.2.
- The imported UML Packages with the NIEM PSM Profile applied and the NIEM PIM Profile not applied is a conforming NIEM PSM as defined in Subclause 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**

Subclause 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**

Subclause 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**

Subclause 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 Subclause 2.6.

## **2.8 NIEM MPD Artifact to NIEM MPD Model Transform**

Subclause 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 Subclause 2.2.

- The tool produces a NIEM PSM as described in Subclause 2.3.
- 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 Subclause 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 Subclause 2.6.

**NOTE.** The NIEM PSM to NIEM-conforming XML schema generation rules as described in Subclause 9.3 may be formalized using QVT [QVT] (see also Annex A.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 subclause.

- 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 Subclause 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 Subclause 2.8.

### 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 2.5, ( <a href="http://www.omg.org/spec/MOF/2.5">http://www.omg.org/spec/MOF/2.5</a> )
[NIEM-3]	NIEM 3 ( <a href="http://reference.niem.gov/niem/">http://reference.niem.gov/niem/</a> )
[NIEM-Conformance]	NIEM Conformance, Version 3.0 ( <a href="http://reference.niem.gov/niem/specification/conformance/3.0/">http://reference.niem.gov/niem/specification/conformance/3.0/</a> )
[NIEM-MPD]	NIEM Model Package Description Specification, Version 3.0 ( <a href="http://reference.niem.gov/niem/specification/model-package-description/3.0/">http://reference.niem.gov/niem/specification/model-package-description/3.0/</a> )
[NIEM-NDR]	NIEM Naming and Design Rules (NDR), Version 3.0 ( <a href="http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/">http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/</a> )
[OCL]	OMG Object Constraint Language (OCL), Version 2.3.1, formal/2012-01-01 ( <a href="http://www.omg.org/spec/OCL/2.3.1">http://www.omg.org/spec/OCL/2.3.1</a> )
[QVT]	Meta Object Facility (MOF) Query/View/Transformation Specification, Version 1.1, formal/2011-01-01 ( <a href="http://www.omg.org/spec/QVT/1.1">http://www.omg.org/spec/QVT/1.1</a> )
[RFC2119]	Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997 ( <a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a> )
[Schematron]	ISO Schematron ( <a href="http://www.schematron.com">http://www.schematron.com</a> )
[UML]	OMG Unified Modeling Language (OMG UML), Version 2.5, ( <a href="http://www.omg.org/spec/UML/2.5">http://www.omg.org/spec/UML/2.5</a> )
[XMI]	OMG MOF 2 XMI Mapping Specification, Version 2.5 ( <a href="http://www.omg.org/spec/XMI/2.5">http://www.omg.org/spec/XMI/2.5</a> )
[XMLNamespaces]	Namespaces in XML, World Wide Web Consortium 16 August 2006 ( <a href="http://www.w3.org/TR/2006/REC-xml-names-20060816">http://www.w3.org/TR/2006/REC-xml-names-20060816</a> )
	Namespaces in XML Errata, 6 December 2002 ( <a href="http://www.w3.org/XML/xml-names-19990114-errata">http://www.w3.org/XML/xml-names-19990114-errata</a> )
[XMLSchemaDatatypes]	XML Schema Part 2: Datatypes Second Edition, W3C Recommendation ( <a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a> )
[XMLSchemaStructures]	XML Schema Part 1: Structures Second Edition, W3C Recommendation ( <a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a> )

# 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. NIEM defines a set of conformance targets and requires that each NIEM-conformant document and schema declares which conformance targets it satisfies. See [NIEM-NDR] [Section 4](#).

### **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.

## **Schematron**

An ISO standard for making assertions about patterns found in XML documents.

## **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 a 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) 2.4.1 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
NIEM-3	National Information Exchange Model, Version 3
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 Acknowledgements

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

- Submitters:
  - Model Driven Solutions
- 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](http://www.ise.gov)
  - Office of the Secretary of Defense
- Contributors:
  - Adaptive
  - Escape Velocity
  - Everware-CBDI
  - Georgia Tech Research Institute (GTRI)
  - Hidden Symmetry Ltd
  - 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.* Use 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 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. This specification contains numerous hyperlinks to URLs within the [NIEM-NDR] and [NIEM-MPD] normative specifications. These links provide additional detail and clarity to this specification.

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 240 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 which underlie its structure and maintain its consistency. While the PIM specializes NIEM, every effort has been made to make the representation of NIEM in UML correspond to commonly accepted patterns of modeling in UML. Someone familiar with UML should be able to start modeling quickly and in many cases, with minimal modification, may be able to reuse existing models to derive NIEM PSM artifacts and ultimately NIEM artifacts.

A NIEM PIM conforms to the rules and structure of the PIM profile described in this document and, with NIEM-UML conforming tooling can 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).

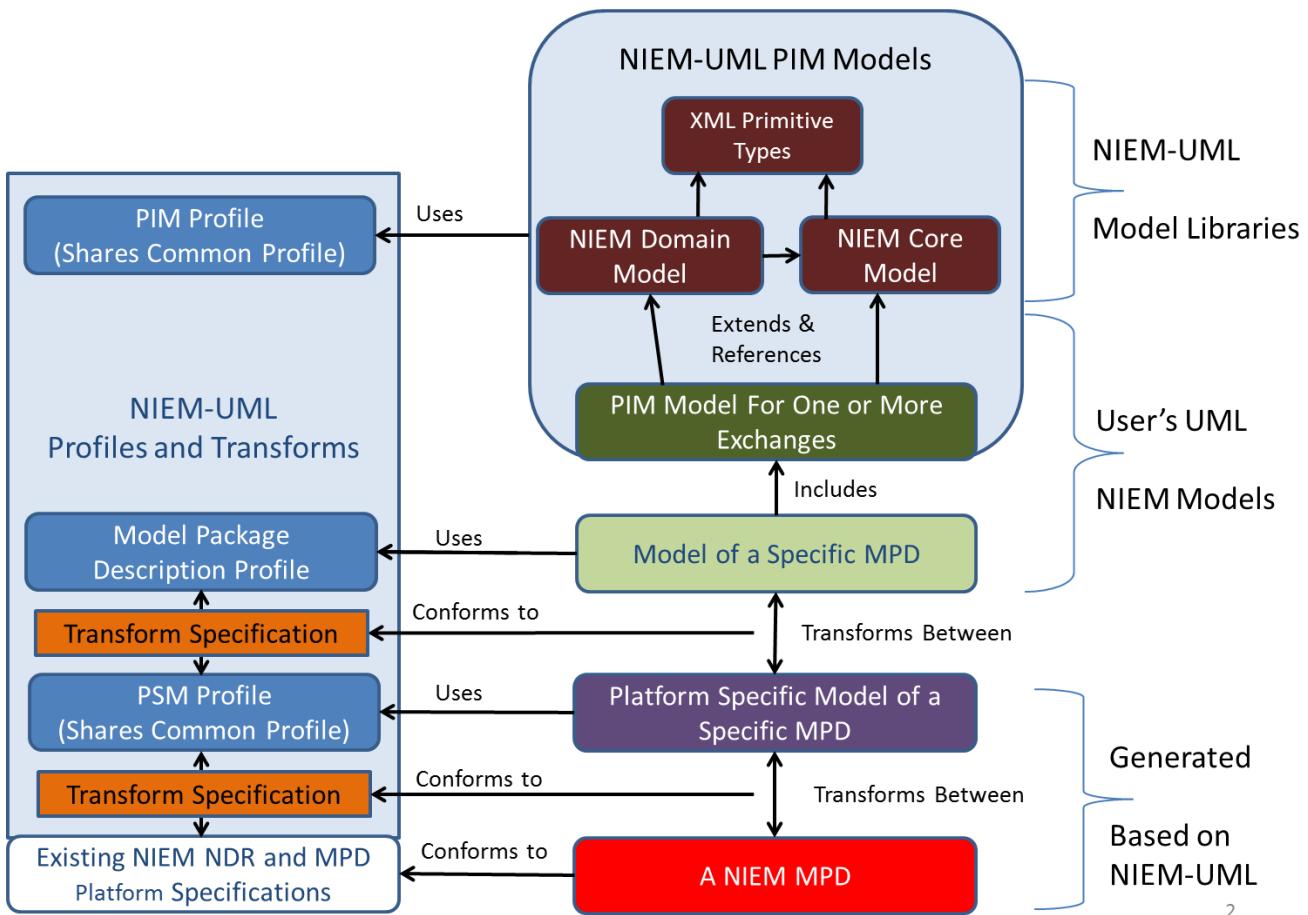
### **6.3.3.3 Intent of the PSM**

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.



2

**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

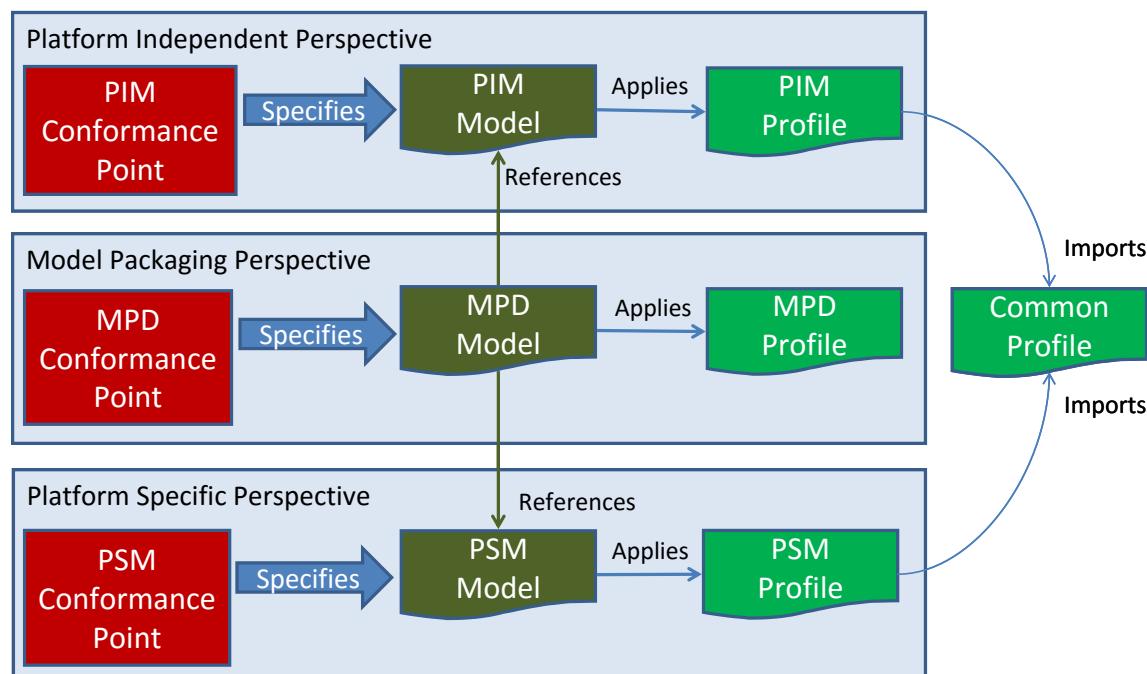
### 7.1.1 Introduction

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 subclauses in this clause then discuss how to model and transform various NIEM concepts across the three perspectives.

**Note:** to understand the PSM to Schema Mapping and XML Schema Representation aspects of these subclauses it may be helpful to refer to clause 9.1.1 which describes the context in which these schemas should be interpreted.

## 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.
	«InformationModel»	7.2.1 Namespaces	An information model is a namespace that contains a <i>default purpose</i> , such as reference, subset, extension or exchange.
<b>Types</b>			
Class	None «ObjectType»	7.3.2 Object Types	Object type is the default for UML classes.  A NIEM object type represents data about things with their own identity and lifespan that have some existence. An object may or may not represent a physical thing. It may represent something conceptual.
	See «RoleOf» Property and «RolePlayedBy» Generalization	7.3.3 Role Types	NIEM differentiates between an object and a role of the object. The term “role” is used here to mean a function or part played by some object. A class is interpreted as a role by means of a «RoleOf» association end or «RolePlayedBy» generalization.
	«AssociationType»	7.3.4 Association Types	A NIEM <i>association</i> 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).

**Table 7-1 Platform Independent Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
	«MetadataType»	7.3.5 Metadata Types	NIEM <i>metadata</i> 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.
	«AugmentationType» See also «Augments» Realization	7.3.6 Augmentation Types	A NIEM <i>augmentation type</i> 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 <i>adapter type</i> 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»	7.5.5 Choice Groups	A <i>choice</i> 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 direct instances.
Association Class	None	7.3.4 Association Types	A NIEM <i>association</i> 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 <i>Also applies to PrimitiveType and Enumeration</i>	«Union»	7.4.4 Unions	A <i>union</i> is a simple type whose values are the union of the values of one or more other simple types, which are the <i>member types</i> of the union.

**Table 7-1 Platform Independent Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
which are DataTypes	«List»	7.4.5 Lists	A <i>list</i> 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 <i>item type</i> 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 <i>primitive type</i> is a simple type defined in terms of a predefined set of atomic values such as strings and numbers.
Enumeration	None	7.4.3 Code Types	A UML enumeration with literals represents a NIEM <i>code simple type</i> , 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.  Because NIEM properties represented as elements must have complex types, the PIM-PSM mapping also generates a NIEM <i>code type</i> , which is a complex type that extends the <i>code simple type</i> .
	«LocalVocabulary»	7.2.3 Local Vocabularies	A local vocabulary defines terms and abbreviations meaningful within a community.
Enumeration Literal	«LocalTerm»	7.2.3 Local Vocabularies	A term within a local vocabulary.
Relations			
Aggregation (Property / Association End)	None		Aggregation may be used in NIEM-UML models but has no impact on the generated schema or conformance. The reference/containment choice is made at the instance level in NIEM using structures:REF and structures:ID (see clause 9.1.1).

**Table 7-1 Platform Independent Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
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.
	«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	«Augments»	7.3.6 Augmentation Types	Augments specifies what type an augmentation augments.
	«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 Subsets realization establishes the relationship between a subset and reference element. Specializes «References»
	«Restriction»	7.3 Modeling Complex Types and 7.4 Modeling Simple Types	A Restriction realization represents a relationship between two type definitions: the first is derived by restriction from the second.
Usage dependency	«UnionOf»	7.4.4 Unions	A UnionOf establishes the relationship between a «Union» and one of its members.
	«Metadata Application»	7.3.5 Metadata Types	Metadata application is a relation between a «MetadataType» class and a class or property. It restricts the types and elements that may have the metadata.
<b>Properties</b>			
Property / Association End	None	7.5.1 Properties	A <i>property</i> relates a NIEM object (the <i>subject</i> ) to another object or to a value (the <i>object</i> ). 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.

**Table 7-1 Platform Independent Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
	«Representation»	7.5.4 Representations	NIEM defines a Representation pattern that enables multiple representations for a single concept. The representations are in the substitution group for the representation element, represented in UML as a typeless property stereotyped as «Representation».
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.
<b>General</b>			
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.
	«Deprecated»		A deprecated component is one whose use is not recommended.
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.

### 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
<b>Types</b>			
Class	«ObjectType»	7.3.2 Object Types	A NIEM <i>object type</i> 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 <i>role type</i> 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 <i>association</i> 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 <i>metadata</i> 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 <i>augmentation type</i> is a complex type that provides a reusable block of data that may be added to object types or association types.

**Table 7-2 Platform Specific Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
	«AdapterType»	7.3.7 Adapter Types	An <i>adapter type</i> 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»	7.5.5 Choice Groups	A <i>choice</i> 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 <i>Also applies to PrimitiveType and Enumeration which are DataTypes</i>	«Union»	7.4.4 Unions	A <i>union</i> is a simple type whose values are the union of the values of one or more other simple types, which are the <i>member types</i> of the union.
	«List»	7.4.5 Lists	A <i>list</i> 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 <i>item type</i> 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 <i>primitive type</i> is a simple type defined in terms of a predefined set of atomic values such as strings or numbers.

**Table 7-2 Platform Specific Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
Enumeration	None	7.4.3 Code Types	A UML enumeration with literals represents a NIEM <i>code simple type</i> , 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.
Enumeration	«LocalVocabulary»	7.2.3 Local Vocabularies	A local vocabulary defines terms and abbreviations meaningful within a community.
Enumeration Literal	«LocalTerm»	7.2.3 Local Vocabularies	A term within a local vocabulary.
<b>Relations</b>			
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.
Realization	«References»	7.5.2 Property Holders and Property References	A References realization reuses class and property definitions from another class or namespace.
	«XSDDeclaration»	7.5.2 Property Holders and Property References	An XSDDeclaration realization identifies the property declaration referenced by a specific property use.
	«XSDSimple Content»	7.3.2 Object Types	The «XSDSimpleContent» stereotype represents a relationship between two type definitions: the first is a complex type definition with simple content, 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 <code>xs:restriction</code> to the other complex type.  Otherwise, the relationship is implemented in XML Schema through base attribute on the <code>xs:extension</code> element of the first type definition, the actual value of which resolves to the second type definition.

**Table 7-2 Platform Specific Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
	«Restriction»	7.3 Modeling Complex Types and 7.4 Modeling Simple Types	A Restriction realization represents a relationship between two type definitions: the first is derived by restriction from the second.
Usage dependency	«UnionOf»	7.4.4 Unions	A UnionOf establishes the relationship between a «Union» and one of its members.
	«Metadata Application»	7.3.5 Metadata Types	Metadata application is a relation between a «MetadataType» class and a class or property. It restricts the types and elements that may have the metadata.
<b>Properties</b>			
Property	«XSDProperty»	7.5.1 Properties	A <i>property</i> relates a NIEM object (the <i>subject</i> ) to another object or to a value (the <i>object</i> ). Property data describes an object as having a characteristic with a specific value or a particular relationship to another object.
	«XSDAnyProperty»	7.5.1 Properties	An «XSDAnyProperty» property may have a value of any type. It is implemented in XML schema as an xs:any.
	«Representation»	7.5.4 Representations	NIEM defines a Representation pattern that enables multiple representations for a single concept. The representations are in the substitution group for the representation element, represented in UML as a typeless property stereotyped as «Representation».
Multiplicity (Property)	None	7.5.1 Properties	Multiplicity constrains how many values a property or association end may have.
Subsets (Property)	None	7.5.3 Substitution Groups	Subset defines a property as being substitutable for another property. This expresses the NIEM substitution group concept.
Derived Union (Property)	None	7.5.3 Substitution Groups	Derived union defines a property whose values are entirely derived as the union of the values of properties that subset it. This expresses the NIEM concept of an abstract property.

**Table 7-2 Platform Specific Perspective Modeling Summary**

UML Element	Stereotype	NIEM Concept Reference	Note
<b>General</b>			
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.
Element	«Deprecated»		A deprecated component is one whose use is not recommended.
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. Such a package is a structure of files constituted within a ZIP file that together provide the complete definition of a NIEM MPD. The core concept in a Model Package Description is the mpd-catalog.xml file, which contains metadata to enable the identification, location and navigation of the various artifacts within the packaged MPD.

In NIEM-UML, most artifacts within an MPD are modeled by instantiating an Artifact specified by the profile; the profile provides an Artifact type for each type of artifact that may appear in an MPD. Most relationships within an MPD are modeled using stereotyped Dependencies, where the different stereotypes represented different kinds of MPD relationship.

The UML artifacts shown in Table 7-3 and stereotyped elements shown in Table 7-4 have an interpretation in a NIEM-UML MPD model and are supported by the normative mapping from a NIEM-UML model to NIEM conformant MPD Catalog components via the mappings specified in Clause 9. These tables refer to the MPD Catalog XML Schema document, in which the prefix c: refers to “<http://reference.niem.gov/niem/resource/mpd/catalog/3.0/>” and nc: refers to “<http://release.niem.gov/niem/niem-core/3.0/>”. Note that the MPD catalog imports niem-core and uses it for some elements relating to contact information. Those catalog components whose name ends “Type” are xs:complexTypes, those whose name ends “SimpleType” are xs:simpleTypes, others are xs:elements or (where specified) xs:attributes. UML Artifacts whose type names are in *italics* are abstract.

**Table 7-3 Model Packaging Perspective Modeling Summary - Artifacts**

UML Element	Type	MPD Catalog component	Note
Artifact	<i>ArtifactOrArtifactSet</i>		A data concept for a file or file set in an MPD.
	ConformanceTargetType	c:ConformanceTargetType	A data type for identifying and describing a conformance

		target.
ConstraintSchemaDocumentSet	c:ConstraintSchemaDocument Set	An MPD artifact set of constraint schema documents and other supporting artifacts.
ContactInformationType	nc>ContactInformationType	A data type for how to contact a person or an organization.
<i>DescribedType</i>		Common supertype for NIEM MPD Catalog types which have descriptionText.
<i>EntityRepresentation</i>		A data concept for a person, organization, or thing capable of bearing legal rights and responsibilities.
EXIXMLSchemaType	c:EXIXMLSchema	An XML Schema to be used for EXI serialization of an IEP Class
FileSet	c:FileSet	A generic MPD artifact set; used to group artifacts that are not accounted for by other set classifiers.
<i>FileSet</i> <i>Type</i>		A data type for a set of MPD file artifacts.
IEPConformanceTargetType	c:IEPConformanceTarget Type	A data type for a class or category of IEP, which has a set of validity constraints and a unique identifier.
ModelPackageDescription	"http://reference.niem.gov/niem/resource/mpd/catalog/3.0/"	The key artifact of the MPD model; represents the mpd-catalog.
OrganizationType	nc:OrganizationType	A data type for a body of people organized for a particular purpose.
PersonType	nc:PersonType	A data type for a human being.
QualifiedNamesType	c:QualifiedNamesType	A data type for a set of qualified names.
RelaxNGValidationType	c:RelaxNGValidationType	A data type for a RelaxNG validation constraint, indicating a RelaxNG schema document against which an artifact may be validated, as

		well as a description of the validation roots for assessment of validity.
SchemaDocumentSet	c:SchemaDocumentSet	An MPD artifact set that may include subset schema documents, extension and external schema documents, and other supporting artifacts.
<i>SchemaDocumentSetType</i>		A data type for an MPD artifact set that may include subset schema documents, extension schema documents, and external schema documents or constraint schema documents.
SchematronValidationType	c:SchematronValidationType	A validity constraint that indicates that an artifact must be valid against the rules carried by a Schematron file, starting with the identified validation roots.
TextRuleType	c:TextRuleType	A data type for a rule drafted in a human language.
<i>ValidityConstraintType</i>		A data concept for a rule or instructions for validating an IEP candidate.
<i>ValidityConstraintWithContextType</i>		A data concept for a rule or instructions for validating an IEP candidate (XML document) using some context within that XML document.
ValidityContextType	c:ValidityContextType	A data type for a rule or instructions for validating an IEP candidate within context defined by an XPath expression.
XMLSchemaType	c:XMLSchemaType	A data type for a validity constraint that indicates an XML Schema against which an artifact may be validated, or which can be used for other purposes. c:XMLSchemaDocument identifies the root or starting XML schema document.

	XPathType	c:XPathType	A data type for an XPath expression.
Enumeration	ChangeCodeSimpleType		No mapping to MPD catalog
	ModelPackageDescriptionClass Code	c:mpdClassURIList	The MPD catalog c:mpdClassURIList attribute declares a list of conformance target identifiers, identifying the conformance targets to which the MPD claims to conform.
	RelationshipCode	c:RelationshipCodeSimple Type	A data type for a classification of the relationship between MPDs.

**Table 7-4 Model Packaging Perspective Modeling Summary - Stereotypes**

UML Element	Stereotype	MPD Catalog Component	Note
Package	«ChangeInformationType»		No mapping to MPD catalog
	«ChangeLogType»		No mapping to MPD catalog
Dependency	«ModelPackageDescription Relationship»	c:Relationship	A relationship between MPDs.
Usage	«ApplicationInfo»	c:ApplicationInfo	An MPD artifact that is used by a software tool (e.g., import, export, input, output, etc.).
	«BusinessRulesArtifact»	c:BusinessRulesArtifact	An MPD artifact that contains business rules and constraints on exchange content.
	«ConformanceAssertion»	c:ConformanceAssertion	An MPD artifact that is a signed declaration that a NIEM IEPD or EIEM is NIEM-conformant
	«ConformanceReport»	c:ConformanceReport	An MPD artifact either auto-generated by a NIEM-aware software tool or manually prepared that checks NIEM conformance and/or quality and renders a detailed report of results. This report may also be an auto-generated and manually prepared hybrid

		artifact.
«Documentation»	c:Documentation	An MPD artifact that is a form of explanatory documentation.
«ExtensionSchemaDocument»	c:ExtensionSchemaDocument	An MPD artifact that is a NIEM extension schema document.
«ExternalSchemaDocument»	c:ExternalSchemaDocument	An MPD artifact that is a schema document external to NIEM.
«File»	c:File	A generic electronic file artifact in an MPD; a file stored on a computer system.
«FileType»	c:FileType	A data type for an MPD file artifact.
«IEPSampleXMLDocument»	c:IEPSampleXMLDocument	An example MPD instance XML document or IEP artifact.
«MPDChangeLog»	c:MPDChangeLog	An MPD artifact that contains a record of the MPD changes.
«qualifiedName»	c:qualifiedNameList attribute	A list of qualified names.
«ReadMe»	c:ReadMe	An MPD read-me artifact.
«ReferenceSchemaDocument»	c:ReferenceSchemaDocument	An MPD artifact that is a reference schema document (from a release, domain update, or core update).
«RelaxNGSchema»	c:RelaxNGSchema	A RelaxNG schema.
«RequiredFile»	c:RequiredFile	An MPD file artifact that another artifact depends on and should not be separated from.
«SchematronSchema»	c:SchematronSchema	A Schematron schema document.
«SubsetSchemaDocument»	c:SubsetSchemaDocument	An MPD artifact that is a subset schema document.
«Wantlist»	c:Wantlist	An MPD artifact that represents a NIEM schema subset and is used as an import or export for the

		NIEM Schema Subset Generator Tool.
«XMLCatalog»	c:XMLCatalog	An MPD artifact that is an OASIS XML catalog.
«XMLSchemaDocument»	c:XMLSchemaDocument	An MPD artifact that is an XML schema document (i.e., an XSD that is not necessarily a NIEM subset, extension, or reference schema).

## 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 in all the symbol spaces of a namespace [NIEM-NDR] [Section 10](#).

#### 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, and the schema version provided as the value of the version attribute. Table 7-5 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 7-5 NIEM Components included in a NIEM Namespace**

NIEM Component	UML Representation
Complex Type	Class (see Subclause 7.3)
Simple Type	Data Type (see Subclause 7.4)
Property Declaration	Property (see Subclause 7.5)
LocalVocabulary	Local Vocabulary (See Subclause 7.2.3)

A «Namespace» package is used to group those model elements of the kinds shown in Table 7-5 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.

The `isConformant` property indicates whether a «Namespace» package is NIEM-conformant. The targets it conforms to are specified by the `defaultPurpose` of the related «InformationModel», and optionally by the «Namespace» package's `conformanceTargets` attribute.

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

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` property. 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 or extension). If no other purpose is specified when an «InformationModel» package is referenced in an MPD model, then the default purpose is used (see Subclause 7.6.2). This allows for the identification of information models in a PIM that are, e.g., specifically intended to be subsets of references models, extensions of such subsets, etc.

An «InformationModel» package provides the logical scoping for the NIEM naming of model elements representing NIEM components (see Subclause 7.2.2). This includes UML properties representing NIEM properties, even though a UML property is not a packageable element. The UML namespace for a property is the UML classifier that owns the property. However, in NIEM every property *declaration* is considered to be “top level”, and, so, the NIEM property names are included in the NIEM namespace. (This is discussed further in Subclause 7.5.2.)

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

## PSM

A «Namespace» package represents an XML schema. The target namespace for the schema is supplied as the value of the `targetNamespace` attribute of the stereotype. The elements in the package (or any subpackage, to any level of nesting) representing NIEM components (as indicated in Table 7-5) are implemented as components of the XML schema represented by package.

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.
- 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 7-6 and elements in the package mapped per Table 7-5.

**Table 7-6 Mapping of a «Namespace» package to an xs:schema**

Stereotype Attributes	Schema Property
Namespace::targetNamespace	xs:schema/@targetNamespace
Namespace::version	xs:schema/@version

- 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 xs:schema/xs:annotation/xs: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 classes (not shown in the diagram) that represents NIEM object types (see Subclause 7.3.2). The properties of these classes represent both the declaration of NIEM properties and the use of those properties in the context of the object types represented by the classes (see Subclause 7.5.2). Therefore, the names of the object types 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 various attributes and elements elided):

```

<xs:schema
    targetNamespace="http://release.niem.gov/niem/domains/cbrn/3.0/" version="1"
    ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and-
    design-rules/3.0/#ReferenceSchemaDocument"
    xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/" ...>

```

```

<xs:annotation>
  <xs:documentation>
    Chemical, Biological, Radiological, and Nuclear Domain
  </xs:documentation>
</xs:annotation>
...
</xs: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 [NIEM-NDR] [Section 10](#). 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 7-5 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. However, 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 as specified in the default PSM mapping rules in subsequent subclauses covering each kind of item.

**NOTE.** The rules for constructing NIEM names are intended to produce names that are syntactically valid according to the rules for required name prefixes and/or suffixes. It is still the responsibility of the modeler to provide UML names for model elements representing NIEM components that have semantically appropriate object-class, property and qualifier terms (per [NIEM-NDR] sections [10.8.4](#), [10.8.5](#) and [10.8.6](#)), so that the constructed NIEM names are fully conformant.

The name of a 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 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.

- 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.

#### 7.2.2.4 Example

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

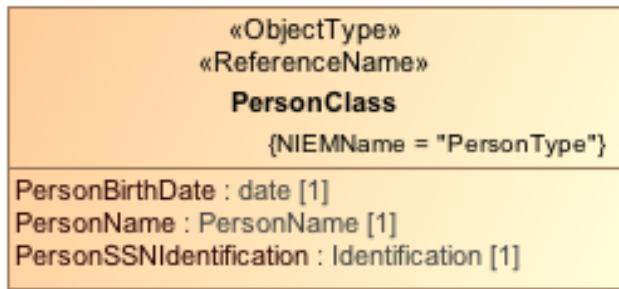


Figure 7-3 Specification of a NIEM name using the «ReferenceName» stereotype

### 7.2.3 Local Vocabularies

#### 7.2.3.1 Background

A NIEM namespace may define a “local vocabulary”. A local vocabulary defines a set of domain specific local terms or abbreviations that then may be used in NIEM names and definitions. See [NIEM-NDR] section [10.8.2](#).

#### 7.2.3.2 Representation

##### Common

The local vocabulary is defined as a stereotype of enumeration where each enumeration literal is stereotyped as a local term. The enumeration literal’s UML name corresponds with the domain specific abbreviation; the literal property gives the meaning of the local term; the definition property is a dictionary-type description of the meaning of the local term. The sourceURIs property value holds the NIEM sourceURIs. The value of ownedComment.body gives the NIEM SourceText terms.

#### 7.2.3.3 Example

##### PIM Representation

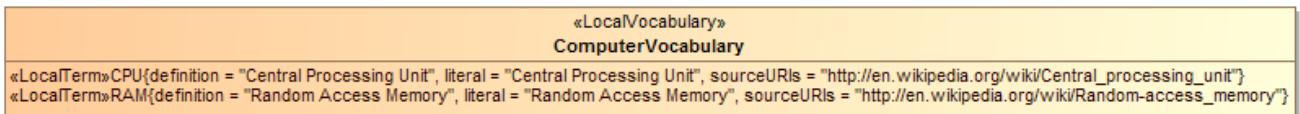


Figure 7-4 A local vocabulary

This figure shows a local vocabulary defining the terms “CPU” and “RAM”.

##### PSM Representation

The PSM representation is the same.

## **XMLSchema Representation**

The schema representation of the local terms is in the `xs:schema/xs:annotation/xs:appinfo`.

```
<xs:annotation>
  <xs:documentation>Computer Vocabulary</xs:documentation>
  <xs:appinfo>
    <term:LocalTerm
      term="CPU"
      literal="Central Processing Unit"
      definition="Central Processing Unit"
      sourceURIss="http://en.wikipedia.org/wiki/Central_processing_unit" />
    <term:LocalTerm
      term="RAM"
      literal="Random Access Memory"
      definition="Random Access Memory "
      sourceURIss="http://en.wikipedia.org/wiki/Random-access_memory "/>
  </xs:appinfo>
</xs:annotation>
```

## **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] [Section 10.1](#). 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 7-7. All of the stereotypes cited in the table that apply to UML::Class are specializations of the abstract «NIEMType» stereotype. Subsequent subclauses provide the details on how to model each kind of complex type.

**Table 7-7 Complex Type Representation**

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

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

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

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

## PIM

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

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

In a PIM, additional notations using generalization are allowed in the modeling of role types (see Subclause 7.3.3). However, a «NIEMType» class may be the special class in at most one generalization that is not marked as a «RolePlayedBy» stereotype, 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 «XSDSimpleContent» realization. The supplier of the «XSDSimpleContent» is a DataType. The class must not be both the special class in a generalization and the client of an «XSDSimpleContent» realization. If the class is the client of a «Restriction» realization, then the supplier DataType defines the constraining facets of the Complex Type's `xs:restriction`. If the class is not a client of a «Restriction» realization, then the supplier DataType is the base of the Complex Type's `xs: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.
- All object types with complex content and association types – *augmentable types* - will generate a NIEM *augmentation point* as the last property based on the NIEM naming conventions for augmentation points. See subclause 7.3.6 and [NIEM-NDR] [Section 10.4](#).

#### PSM to XML Schema Mapping

- A class in a PSM with a «NIEMType» stereotype (i.e., one of the stereotypes listed inTable 7-7) applied shall map to a corresponding complex type definition with the `xs: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 «XSDSimpleContent» realization, then the base type definition for the complex type definition shall be structures: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 `xs:complexType/xs:annotation/xs:documentation` for the complex type definition.
- If the class is a client of a realization with the «XSDSimpleContent» 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 «XSDSimpleContent» stereotype applied, and is also the client of a «Restriction» realization, then the content of the `xs:restriction` will include the constraining facets defined by the supplier Data Type of the «XSDSimpleContent» 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] [Section 10.2](#).

### 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 Subclause 7.5.

#### PSM

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

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

### 7.3.2.3 Mapping Summary

#### PIM to PSM Mapping

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

## PSM to XML Schema Mapping

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

### 7.3.2.4 Examples

#### PIM Representation

Figure 7-5 shows an example of a Person object type represented as a class in a PIM. The identification of the class as representing an object type is implicit, since it has no stereotype.

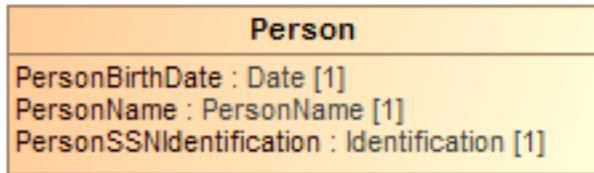
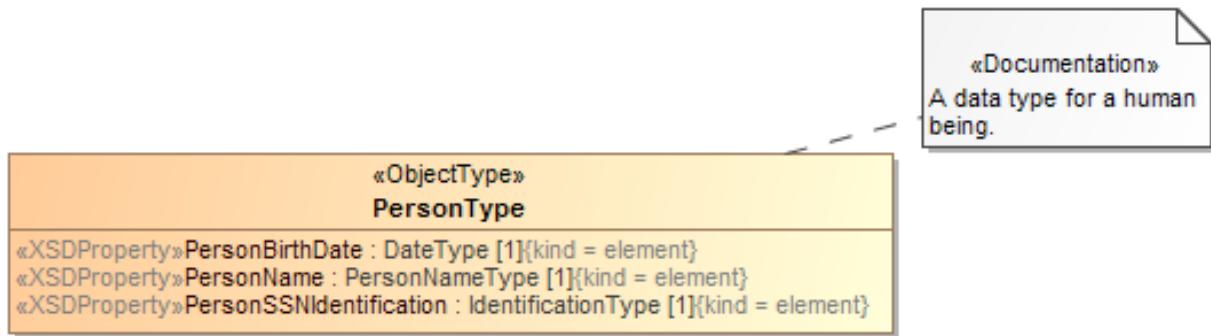


Figure 7-5 Representation of a NIEM object type as a UML class in a PIM

#### PSM Representation

Figure 7-6 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] [Section 11.1](#). Note also the attached «Documentation» comment.



**Figure 7-6 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 content elided – the representation of properties is discussed in subclause 7.5):

```

<xs:complexType name="PersonType">
  <xs:annotation>
    <xs:documentation>A data type for a human being.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      ...
    </xs:extension>
  </xs:complexContent>
</xs: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] [Section 10.2.2](#).

### 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 Subclause 7.5 on the modeling of properties.)

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

#### PIM

In a PIM, a role type may be defined either by explicitly modeling a role-of property or by modeling the role type with a generalization to the base type. If an explicit role-of property is modeled, then it is identified by applying the

«RoleOf» stereotype to the UML property representing it. If a generalization is used, then this is identified by applying the «RolePlayedBy» stereotype to it.

An explicit «RoleOf» property of a role type may be the opposite end of an association between the role type and its base type (note that it is the association *end* that is stereotyped, not the association). If the «RoleOf» property is an association end, then the multiplicity of the near end of the association may be used to distinguish between two semantic 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 when 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 be 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”. A role type is otherwise implemented exactly as for any other object type. (Note that this means the interpretation, above, can’t be explicitly represented in a PSM.)

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

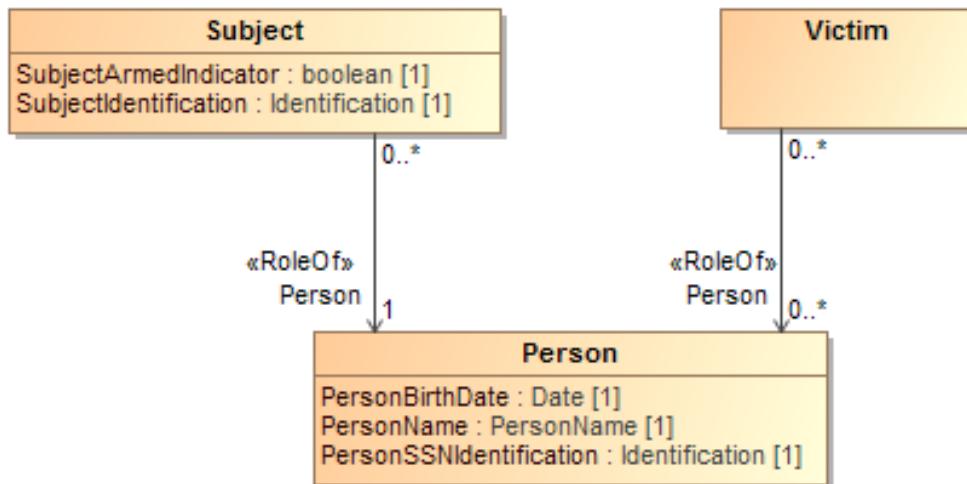


Figure 7-7 Representation of role types using role-of properties in a PIM

Figure 7-8 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.

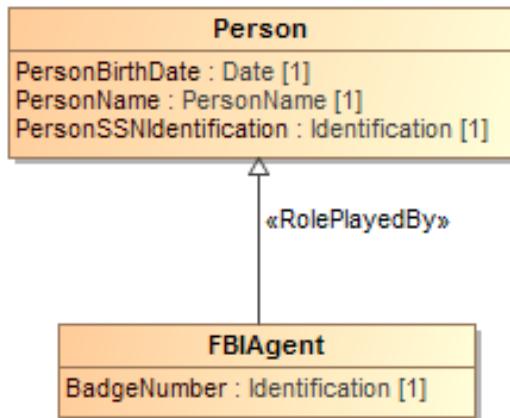
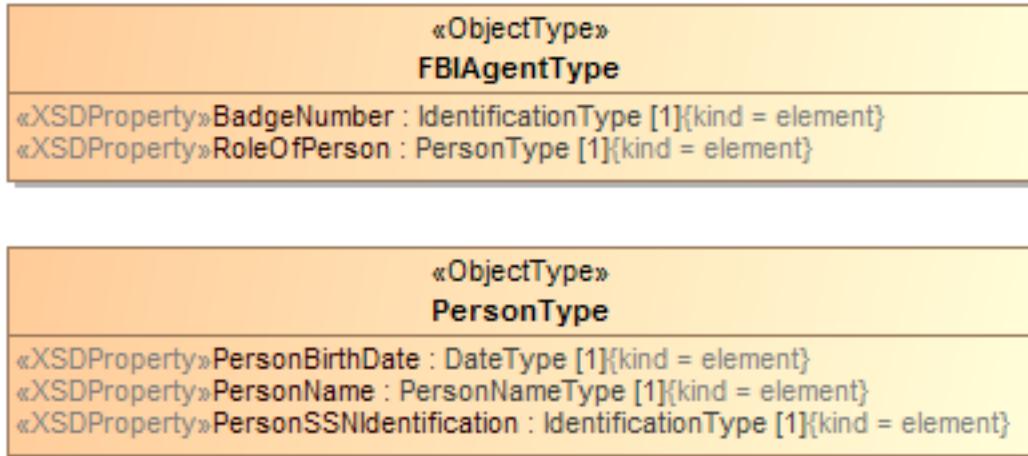


Figure 7-8 Representation of a role type using a generalization in a PIM

#### PSM Representation

Figure 7-9 shows the FBI Agent role type shown in Figure 7-8 as represented in the PSM. Note the required naming of the role-of property.



**Figure 7-9 Representation of a role type in a PSM**

## XML Schema Representation

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

```

<xs:complexType name="SubjectType">
  <xs:annotation>
    <xs:documentation>A data type for a person who is involved or suspected of being
involved in an incident or criminal activity.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="1" ref="nc:RoleOfPerson"/>
        <xs:element maxOccurs="1" minOccurs="1" ref="j:SubjectArmedIndicator"/>
        <xs:element maxOccurs="1" minOccurs="1" ref="j:SubjectIdentification"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="j:SubjectAugmentationPoint"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="VictimType">
  <xs:annotation>
    <xs:documentation>A data type for a person who suffers injury, loss, or death as a
result of an incident.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="nc:RoleOfPerson"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="j:VictimAugmentationPoint"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs: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] [Section 10.3.1](#).

### 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.

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

#### PIM

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

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

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

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

#### PSM

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

### 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 corresponding association class end.

## 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.

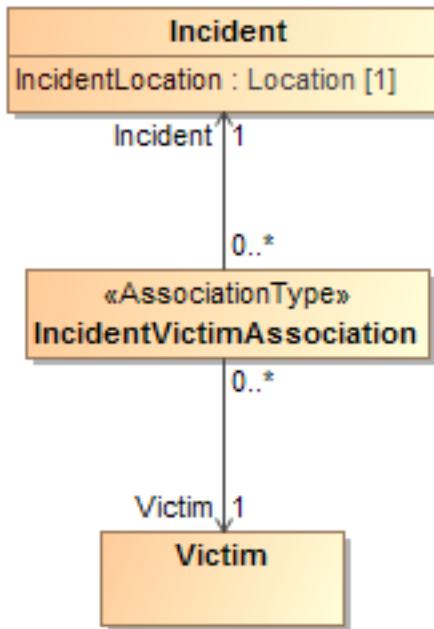
## PSM to XML Schema Mapping

- 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 structures:AssociationType.

### 7.3.4.4 Example

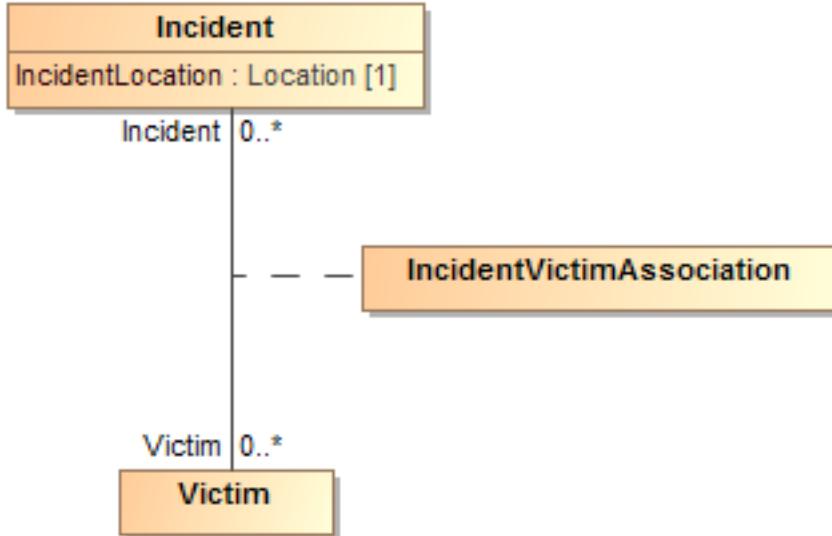
#### PIM Representation

Figure 7-10 represents a NIEM association between incidents and victims. Each association is a relationship between exactly one incident and one victim. Since the properties of the IncidentVictimAssociation association type are modeled as UML associations, multiplicities may be shown on the near ends of the associations. This explicitly models that a victim can be a victim in any number of incidents and an incident may have any number of victims. (More restricted multiplicities may also be used, modeling additional constraints in the PIM, even though these cannot be carried forward to the PSM – see Subclause 7.5.1.)



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

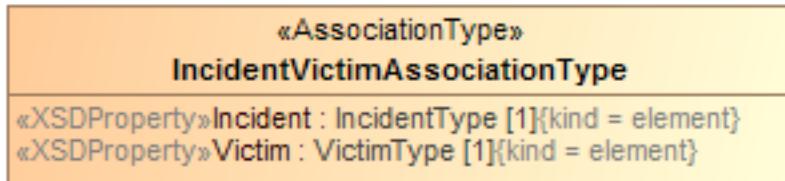
Figure 7-11 represents the same NIEM association between incidents and victims using a UML association class. The multiplicities of the association ends signifies that a victim can be a victim in any number of incidents and an incident may have any number of victims.



**Figure 7-11 Representation of a NIEM association type as a UML association class**

## PSM Representation

Figure 7-12 shows the PSM representation of the IncidentVictim association type.



**Figure 7-12 Representation of a NIEM association type in a NIEM PSM**

### XML Schema Representation

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

```

<xs:complexType name="IncidentVictimAssociationType">
    <xs:annotation>
        <xs:documentation>A data type for a relationship between an incident and a person who is a victim as a result of the incident.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="nc:AssociationType">
            <xs:sequence>
                <xs:element maxOccurs="1" minOccurs="1" ref="nc:Incident"/>
                <xs:element maxOccurs="1" minOccurs="1" ref="j:Victim"/>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                           ref="j:IncidentVictimAssociationAugmentationPoint" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="Victim" nillable="false" type="j:VictimType">
    <xs:annotation>
        <xs:documentation>A person, organization, or other entity who suffers injury, loss, or death as a result of an incident.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="Incident" nillable="false" type="j:IncidentType">
    <xs:annotation>
        <xs:documentation>An occurrence or an event that may require a response.</xs:documentation>
    </xs:annotation>
</xs: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] [Section 10.5.1](#).

### 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 or a property. The former models the restriction of the application of the metadata to NIEM objects represented as instances of the supplier class; the latter models the restriction of the application of the metadata to NIEM elements corresponding to the supplier property or any of its (transitive) substitutions. A «MetadataType» class with no «MetadataApplication» dependency represents metadata that may be applied to any NIEM object.

#### PIM

As for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of a «MetadataType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.5.

#### 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 classes and supplier classes/properties mapped from the PIM.

#### PSM to XML Schema Mapping

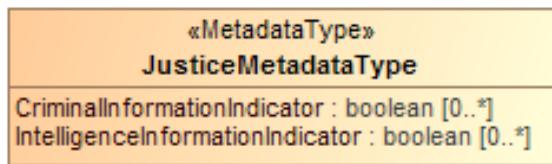
- A class in a PSM with the «MetadataType» stereotype applied shall map to a complex type definition mapped with complex content and:
  - The properties of the class shall map to corresponding property references (XSD attribute uses and element particles) in the complex content of the complex type definition 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 `structures:MetadataType`.

- If a «MetadataType» class in a PSM is the client of a «MetadataApplication» usage dependency and the supplier is a class, then the complex type mapped from the supplier of the dependency shall be referenced in the `xs:complexType/@appInfo:appliesToTypes` attribute for the complex type definition mapped from the «MetadataType» class.
- If a «MetadataType» class in a PSM is the client of a «MetadataApplication» usage dependency and the supplier is a property, then the complex type mapped from the supplier of the dependency shall be referenced in the `xs:complexType/@appInfo:appliesToElements` attribute for the complex type definition mapped from the «MetadataType» class.

### 7.3.5.4 Examples

#### PIM Representation

Figure 7-13 shows a class that represents a metadata type. Since the class has no «MetadataApplication» dependency, the metadata modeled by the class can be applied to any NIEM object. The only difference in the PSM would be the stereotypes on property.



**Figure 7-13 Representation of a metadata type as a UML class in a PIM**

Figure 7-14 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 7-14 Representation of a metadata application constraint as a UML dependency in a PSM or PIM**

#### XML Schema Representation

The **MeasureMetadataType** modeled in Figure 7-14 is represented in XML Schema as follows:

```

<xs:complexType name="MeasureMetadataType" appinfo:appliesToTypes="MeasureType">
    <xs:annotation>
        <xs:documentation>
            A data type for metadata about a measurement.
        </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:MetadataType">
            <xs:sequence>
                ...
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
  
```

## 7.3.6 Augmentation Types

### 7.3.6.1 Background

An *augmentable type* is any object type with complex content (excluding adapter types) or any association type. As noted in Subclause 7.3.1.3, all augmentable types contain an *augmentation point* property. An *augmentation type* is a complex type that provides a reusable block of data that is designed to be added to augmentable types. An *augmentation* is an element that substitutes for an augmentation point element. Most augmentations are typed by augmentation types; however a type that is not an augmentation type may be used as an augmentation. See [NIEM-NDR] [Section 10.4.4](#).

### 7.3.6.2 Representation

#### Common

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

#### PIM

An augmentable type in a PIM may have its augmentation point property omitted.

**NOTE.** Just as for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of an «AugmentationType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.5.

An augmentation is represented in a PIM using a realization with the «Augments» stereotype applied, where the augmenting type (usually but not necessarily an «AugmentationType») is the special class and the augmented type is represented by the general class.

#### PSM

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

An augmentation point is represented by an abstract property (see subclause 7.5.3.1) with name ending `AugmentationPoint`.

An augmentation is represented by a property that subsets the augmentation property, using a similar approach to that described in subclause 7.5.3.

### 7.3.6.3 Mapping Summary

#### 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.

- Every class in a PIM that represents an augmentable type (i.e. association type or object type with complex content) shall map to a corresponding class in the PSM with an abstract AugmentationPoint property added as the last property (if not already there).
  - The name of the property shall be the NIEM name of the type with the suffix “Type” removed and the suffix “AugmentationPoint” added.
- Every class in a PIM that represents an augmentable type shall also map to a corresponding PropertyHolder class in the PSM with the AugmentationPoint property.
- A type that «Augments» another type shall generate a property in the namespace of the «Augments» realization.
  - The name of the property shall be the name of the Realization, or if it has no name, the name of the augmenting type with the suffix “Type” removed.
  - That property shall subset the augmentation point of the target of the “Augments” relation. If the augmenting property and the augmented type are in the same namespace, that subsetting will occur within the PropertyHolder corresponding to the augmentation point. Otherwise a new PropertyHolder will be generated, specializing the augmentation point PropertyHolder, to contain the subsetting property.

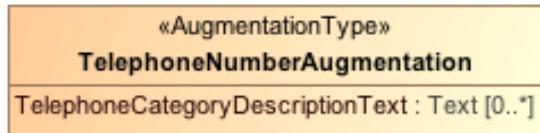
## **PSM to XML Schema Mapping**

- 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 `structures:AugmentationType`.

### **7.3.6.4 Examples**

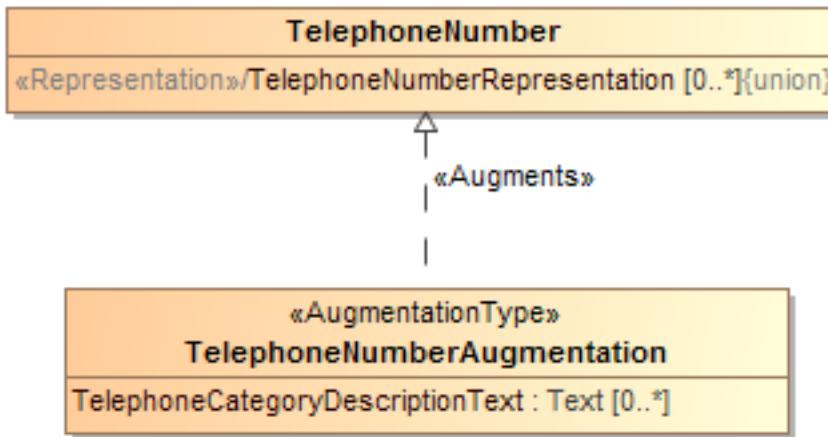
#### **PIM Representation**

Figure 7-15 shows an augmentation type represented as a UML class with the «AugmentationType» stereotype.



**Figure 7-15 Representation of an augmentation type as a UML class in a PIM**

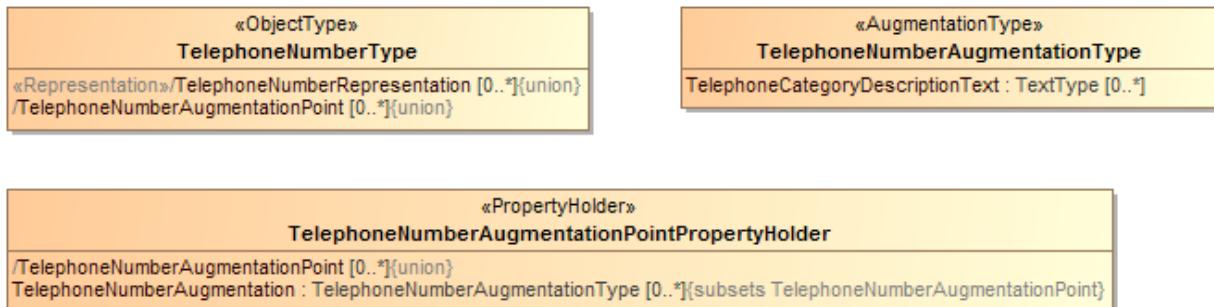
Figure 7-16 shows the augmentation of TelephoneNumber by TelephoneNumberAugmentation using an «Augments» realization.



**Figure 7-16 Representation of augmentation in a PIM**

## PSM Representation

Figure 7-17 shows the PSM representation of the same model. Notice the explicit representation of augmentation points, and how the augmentation represented by «Augments» in the PIM generates an augmentation property subsetting the augmentation point in the PSM.



**Figure 7-17 Representation of augmentation in a PSM**

## XML Schema Representation

The definition of the TelephoneNumberAugmentation type is represented in XML schema as follows:

```

<xs:complexType name="TelephoneNumberAugmentationType">
    <xs:annotation>
        <xs:documentation>A data type that supplements TelephoneNumber.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:AugmentationType">
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="tns:TelephoneCategoryDescriptionText"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
    ...
</xs:complexType>

```

Its use as an augmentation is represented as follows:

```
<xs:element name="TelephoneNumberAugmentation"
    nillable="false"
    substitutionGroup="nc:TelephoneNumberAugmentationPoint"
    type="tns:TelephoneNumberAugmentationType">
    <xs:annotation>
        <xs:documentation>Additional information about a telephone
number</xs:documentation>
    </xs:annotation>
</xs: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] [Section 10.2.3](#).

### 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 «References» realizations to elements defined outside of a «Namespace» package marked as isConformant=true.

#### PIM

As for the representation of an object type in a PIM (see Subclause 7.3.2.2), the properties of an «AdapterType» class may be represented either as attributes of the class or opposite ends of associations in which the class participates. The modeling of properties is discussed further in Subclause 7.5.

#### 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.

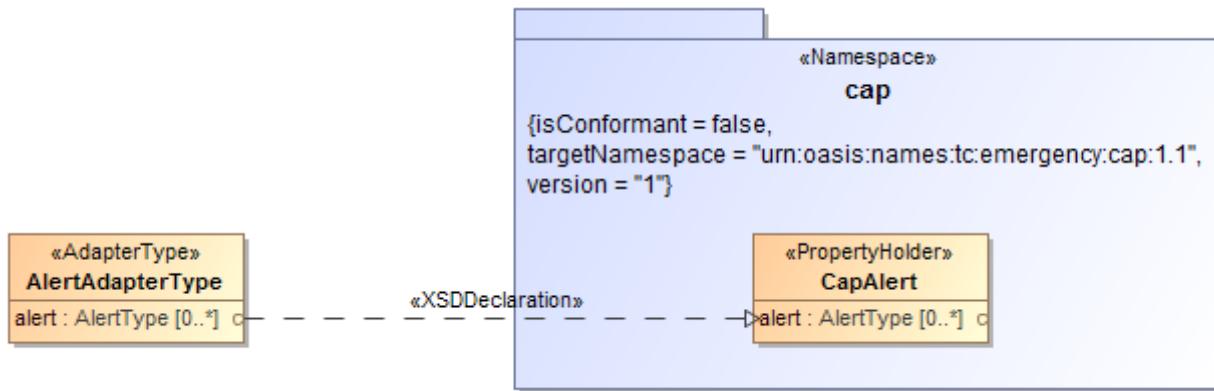
## PSM to XML Schema Mapping

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

### 7.3.7.4 Example

#### PSM Representation

Figure 7-18 shows the PSM representation of the AlertAdapterType class. The namespace cap represents an external schema that does not conform to NIEM.



**Figure 7-18 Representation of an adapter type as a UML class**

#### XML Schema Representation

The AlertAdapterType modeled in Figure 7-18 is represented in XML schema as follows:

```

<xs:complexType name="AlertAdapterType" appinfo:externalAdapterTypeIndicator="true">
  <xs:annotation>
    <xs: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.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="cap:alert"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
  
```

```

    </xs:extension>
  </xs:complexContent>
</xs: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] [Section 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 7-8 summarizes the representation of the various kinds of simple types, as detailed in subsequent subclauses.

**Table 7-8 Simple Type Representation**

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

A simple type may also be defined as having a value space that is a restriction of the value space of another simple type. This is represented by a UML data type that is a client of a «Restriction» realization to another UML data type representing the simple type being restricted. The restricted type may then have the «ValueRestriction» applied, the attributes of which may be used to specify various restriction *facets*. Note that not all facets are applicable to all kinds of simple type.

##### PIM

Simple types that are not enumerations may not be used in a PIM as the types of properties. This is because the PIM-PSM-schema mapping translates all PIM properties to XS element definitions, and NIEM does not permit elements to have simple types.

Enumerations representing code types may be used in a PIM as the types of properties. The PIM-PSM mapping for simple enumerations generates both a code simple type and a complex code type that extends the simple type. XSD elements that correspond to properties typed in a PIM by an enumeration will be typed in the schema by the extending complex type.

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

otherwise it has complex content. Such a specialized data type may not be the general type for any «ValueRestriction» data type.

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

## PSM

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

Generalization is not used with data types in a PSM.

A data type in a PSM that is the client of a «Restriction» realization may also have the «XSDRepresentationRestriction» stereotype applied. This models the restriction on the representation of the literals denoting values of the data type in an XML schema. Specifically, the whiteSpace attribute of «XSDRepresentationRestriction» is implemented as the `xs: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 below).
- 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.
- A DataType stereotyped by «ValueRestriction», and populated with the facet tag values defined on the PIM «ValueRestriction».
- A 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 `xs:simpleType/@name` given by the data type name.
- If a data type in a PSM is the client of a realization stereotyped as «Restriction», then it shall map to a simple type definition that is a restriction whose base type is the supplier type of the realization. If the data type has the «ValueRestriction» stereotype applied, then the attribute values of the stereotype shall map to corresponding restriction facets.
- If a data type in a PSM has the «XSDRepresentationRestriction» stereotype applied, then the simple type definition mapped from the data type shall include a `xs:restriction/xs: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 `xs:simpleType/xs:annotation/xs: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 `xs:restriction`, as described in clause 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 A) 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 A). All primitive types used in a NIEM-UML model shall be either a predefined primitive type from the Primitive Type Library or a primitive type that is a direct or indirect specialization of a predefined primitive type.

## PIM

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

Only primitive types that map to complex types may be used as the types of properties in a PIM.

## 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.

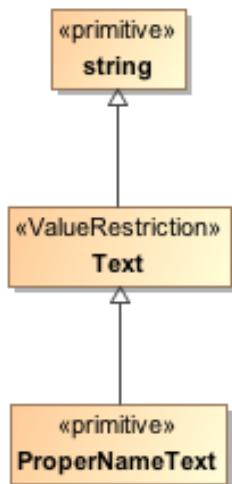
#### PSM to XML Schema Mapping

- 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

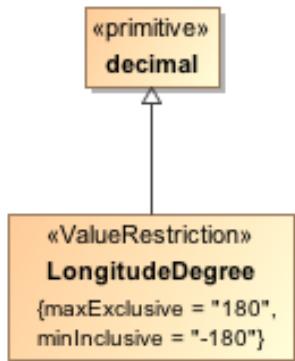
#### PIM Representation

Figure 7-19 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 7-19 Representation of primitive types in a PIM**

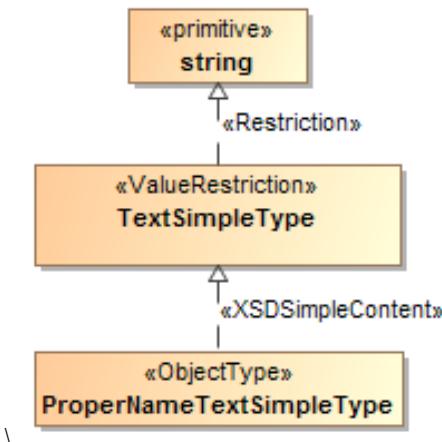
The Text primitive type in Figure 7-19 is stereotyped as a «ValueRestriction», but it does not have any restriction facets specified. Figure 7-20 shows an example of a primitive type defined as a «ValueRestriction» with restriction facets.



**Figure 7-20 Representation of a primitive type with a value restriction in a PIM**

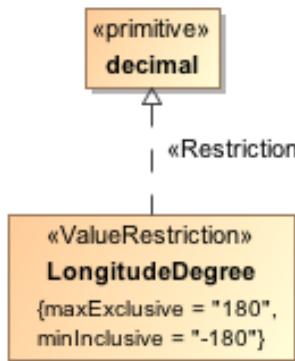
## PSM Representation

Figure 7-21 shows the PSM representation of the primitive types modeled in Figure 7-19. A primitive types in a PSM must be stereotyped as a «ValueRestriction», so the ProperNameTextType becomes an «ObjectType» class in the PSM.



**Figure 7-21 Representation of primitive types in a PSM**

Figure 7-22 shows the PSM representation of the primitive type shown in Figure 7-20, which uses a «Restriction» realization instead of a generalization.



**Figure 7-22 Representation of a primitive type with a value restriction in a PSM**

## XML Schema Representation

The primitive types shown in Figure 7-21 are represented in XML schema as follows:

```

<xs:simpleType name="TextSimpleType">
    <xs:annotation>
        <xs:documentation>A data type for text</xs:documentation>
    </xs:annotation>
    <xs:restriction base=" xs:string"/>
</xs:simpleType>
<xs:complexType name="ProperNameTextSimpleType">
    <xs:annotation>
        <xs:documentation>A data type for proper name text</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="tns:TextSimpleType">
            <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

```

The primitive type shown in Figure 7-22 is represented in XML schema as:

```
<xs:simpleType name="LongitudeDegreeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for longitude degrees</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="-180"/>
    <xs:maxExclusive value="180"/>
  </xs:restriction>
</xs:simpleType>
```

## 7.4.3 Code Types

### 7.4.3.1 Background

A *code type* is a 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] [Section 10.2.4](#).

### 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 enumeration is the client.

#### PIM

As noted in 7.4.1.2, the PIM-PSM-schema mapping translates all PIM properties to XS element definitions, and NIEM does not permit elements to have simple types.

One pattern for addressing this limitation is to define a simple type “wrapped” by a complex type. This may be accomplished in a PIM by defining an enumeration with literals for the simple type, and another empty enumeration that specializes the simple type to represent the complex type. A specialized enumeration without the stereotype application is actually mapped to the PSM as a complex type (as specified for data types in general in Subclause 7.4.1.2), so can be used as the type of a property.

However, simple enumerations defined in a PIM may also be used as the types of properties, as long as they:

- Are non-empty;
- Have no derived types;
- Are not referenced by a Union or List.

Such enumerations generate a pair of types in the target schema, so that the element representing the property is typed by a complex type.

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.

## **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 an atomic simple type definition that is a restriction of the identified base type using multiple `xs: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 that is not empty, has no derived types, and is not referenced by a Union or List shall map to a corresponding pair of types in the PSM, an enumeration for the CodeSimpleType and an object type for the CodeType. The CodeSimpleType has the corresponding enumeration literals. The CodeType shall be related to the CodeSimpleType by an «XSDSimpleContent» realization.
- 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” then the NIEM name of the PSM enumeration shall be the PIM enumeration name, and the NIEM name of the object type shall be the PIM enumeration name with “Simple” removed.
  - If the PIM enumeration name ends in “CodeType” then the NIEM name of the PSM enumeration shall be the PIM enumeration name with “Simple” added between “Code” and “Type”, and the NIEM name of the object type shall be the PIM enumeration name.
  - If the PIM enumeration name ends in “CodeSimple” then the NIEM name of the PSM enumeration shall be the PIM enumeration name with “Type” appended, and the NIEM name of the object type shall be the PIM enumeration name with “Simple” removed and “Type” appended.
  - If the PIM enumeration name ends in “Code” then the NIEM name of the PSM enumeration shall be the PIM enumeration name with “SimpleType” appended, and the NIEM name of the object type shall be the PIM enumeration name with “Type” appended.
  - Otherwise, the NIEM name of the PSM enumeration shall be the PIM enumeration name with “CodeSimpleType” appended, and the NIEM name of the object type shall be the PIM enumeration name with “CodeType” 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 7-23 shows the definition of the SupervisionLevel code type as a UML enumeration.

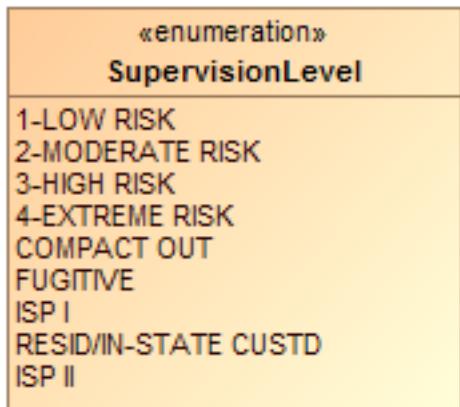
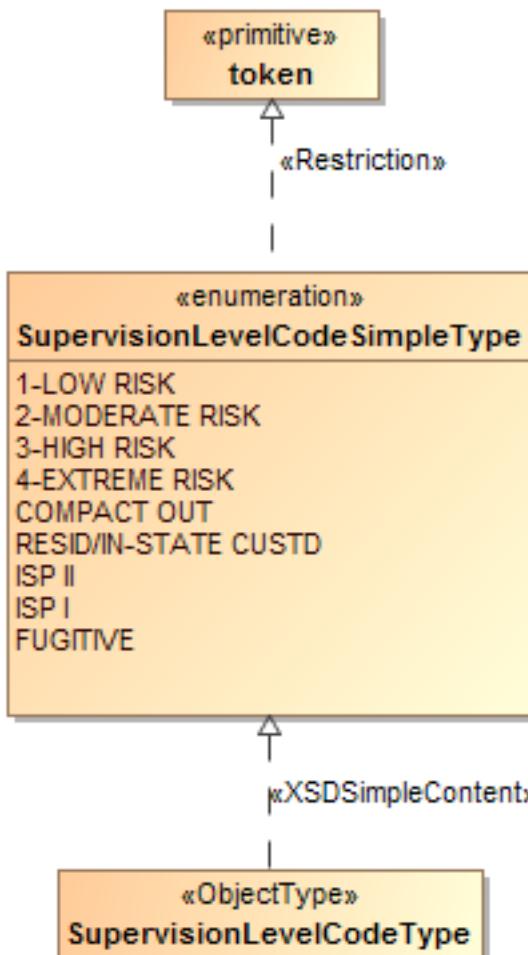


Figure 7-23 A code type represented as a UML enumeration in a PIM

### PSM Representation

Figure 7-24 shows the PSM representation of the model shown in Figure 7-23 , with an explicit «Restriction» realization from the code simple type to the XSD token primitive type, and an XSDSimpleContent from the code type represented an as object type.



**Figure 7-24 A code type represented as a restriction in a PSM**

## XML Schema Representation

The XML Schema representation for the code simple type shown in Figure 7-24 is:

```
<xs:simpleType name="SupervisionLevelCodeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for levels of supervision required for a person.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:enumeration value="1-LOW RISK"/>
    <xs:enumeration value="2-MODERATE RISK"/>
    <xs:enumeration value="3-HIGH RISK"/>
    <xs:enumeration value="4-EXTREME RISK"/>
    <xs:enumeration value="COMPACT OUT"/>
    <xs:enumeration value="FUGITIVE"/>
    <xs:enumeration value="ISP I"/>
    <xs:enumeration value="ISP II"/>
    <xs:enumeration value="RESID/IN-STATE CUSTD"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="SupervisionLevelCodeType">
  <xs:annotation>
    <xs:documentation>A data type for levels of supervision required for a person.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="core_misc:SupervisionLevelCodeSimpleType">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## 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 `xs:union/@xs: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 7-25 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.

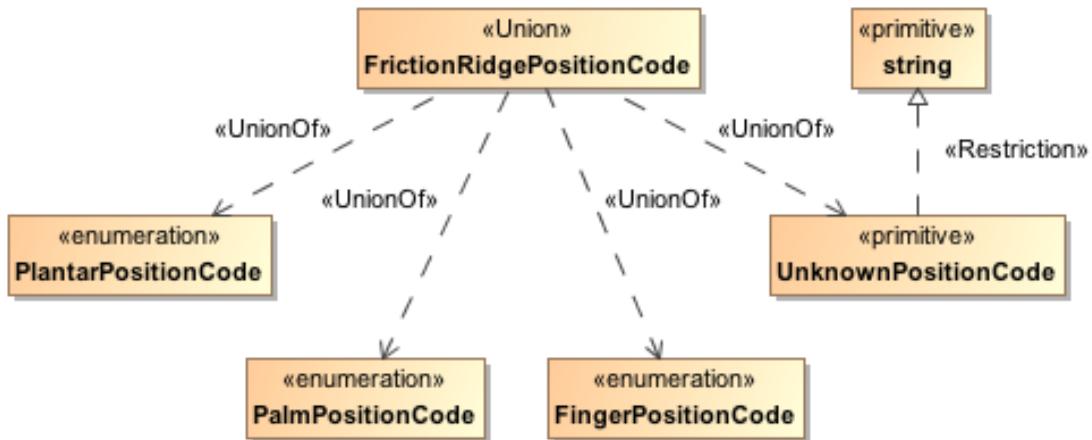


Figure 7-25 Representation of a union as a UML data type

## PSM Representation

The PSM representation for the types shown in Figure 7-25 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 7-25 is implemented in XML Schema as follows:

```

<xs:simpleType name="FrictionRidgePositionCodeSimpleType">
    <xs:annotation>
        <xs:documentation>
            A data type for a friction ridge image position
        </xs:documentation>
    </xs:annotation>
    <xs:union memberTypes="biom:FingerPositionCodeSimpleType
        biom:PalmPositionCodeSimpleType biom:PlantarPositionCodeSimpleType
        biom:UnknownPositionCodeSimpleType"/>
</xs: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.

### 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 7-26 shows the PIM representation of a simple type that is a list of Boolean values. Note that the required property of the «List» data type is represented using an association (see also Section 7.5.1.2)



Figure 7-26 Representation of a list in a PIM

#### PSM Representation

Figure 7-27 shows the PSM representation of the «List» data type shown in Figure 7-26. Note that, in the PSM, the required property of the «List» data type is represented as an attribute.



**Figure 7-27 Representation of a list in a PSM**

## XML Schema Representation

The «List» data type shown in Figure 7-27 is implemented in XML Schema as follows:

```
<xs:simpleType name="BooleanListSimpleType">
    <xs:annotation>
        <xs:documentation>
            A data type for a white space-delimited list of boolean.
        </xs:documentation>
    </xs:annotation>
    <xs:list itemType="xs:boolean"/>
</xs: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] [Section 10.6](#).

#### 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 aggregation may be used in a UML model but it has no meaning in the mapping to NIEM-XML; it is considered a comment.

##### PIM

A property in a PIM must be typed by a complex type or by an enumeration representing a code type.

A property may optionally be represented as a classifier-owned 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 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.

A UML association used to represent a NIEM property (or two NIEM properties) may not be an association class.

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

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

## 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 maximum multiplicity must be 1 and its type must be a data type representing a simple type.

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

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

There are significant differences between the UML representation and XML Schema implementation of a NIEM property. Sections 9.2.1 and 9.2.3 of [NIEM-NDR], Rule 9-35 and Rule 9-47 require that an attribute or element declaration be a top-level declaration; however, Section 7.3.44 of [UML] requires that a Property be the ownedAttribute of a Classifier. Thus in the UML representation, only one Classifier may reference a Property, while in the XML Schema implementation, more than one type definition may reference the same attribute or element declaration.

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

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

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

### 7.5.1.3 Mapping Summary

#### PIM Representation Mapping

- A UML property that is a classifier-owned 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 owned by a class does not have the stereotype «ReferenceName» applied and is not the client of a «References» realization, then its 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, which shall have the simple or complex type mapped from the type of the properties, all of which shall have the same type.
  - 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 `xs:attribute/@name` given by the property name and the `xs: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 `xs: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 `xs: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`.

#### 7.5.1.4 Example

##### PIM Representation

Figure 7-28 shows a set of three NIEM properties represented as attributes of a Person class. The complex type represented by this class is thus also modeled as being the subject of these properties.

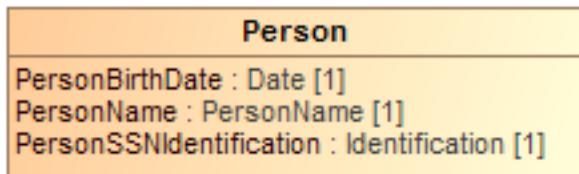


Figure 7-28 Representation of NIEM properties as UML properties in a PIM

Figure 7-29 shows an example of the alternative representation of a NIEM property as an association end.

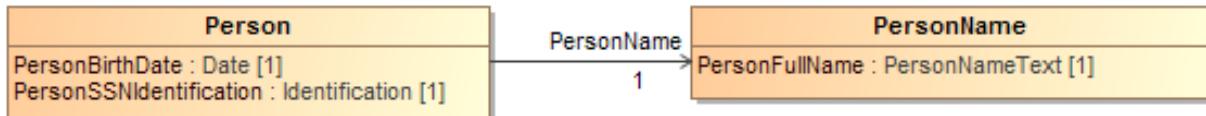


Figure 7-29 Representation of a NIEM property as a UML association end

##### PSM Representation

Figure 7-30 shows the PSM representation of the class modeled in Figure 7-28, with the «XSDProperty» stereotype applied to all properties.

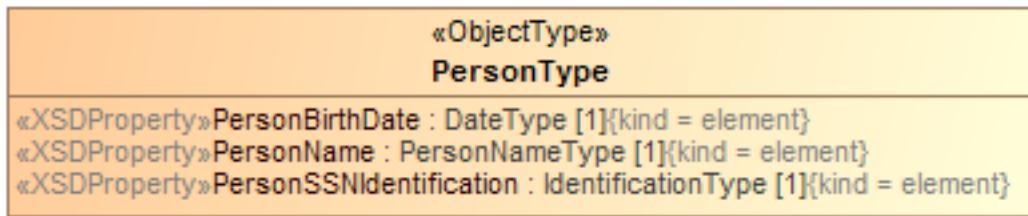


Figure 7-30 Representation of NIEM properties as UML properties in a PSM

##### XML Schema Representation

The class shown in Figure 7-30 is represented in XML schema as follows:

```
<xs:complexType name="PersonType">
    <xs:annotation>
        <xs:documentation>A data type for a human being.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ComplexObjectType">
            <xs:sequence>
                <xs:element maxOccurs="1" minOccurs="1" ref="nc:PersonBirthDate"/>
                <xs:element maxOccurs="1" minOccurs="1" ref="nc:PersonName"/>
                <xs:element maxOccurs="1" minOccurs="1"
                    ref="nc:PersonSSNIdentification"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```

        </xs:sequence>
    </xs:extension>
</xs:complexContent>
</xs:complexType>

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

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

<xs: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 Subclause 7.2.1). However, a UML property that is the client of a «References» realization does not represent a property declaration and thus has the same NIEM name as the supplier of the realization.

## PIM

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

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

## PSM

A property declaration within a PropertyHolder 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 «XSDDeclaration» realization must always be between one property and another property or between a property and a «Namespace» package. In the latter 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 Subclause 7.5.1.4. The «PropertyHolder» class shall have no representation in the XSD.
- A property in a PSM that is the client of a «References» or «XSDDeclaration» realization whose supplier has a different target namespace shall be mapped as an attribute use or element particle, as described in Subclause 7.5.1.4, but shall *not* be mapped as an attribute or element declaration. The attribute use or element particle shall have its `ref` attribute set to the attribute or element declaration mapped from the supplier of the realization.

### 7.5.2.4 Example

#### PIM Representation

Figure 7-31 shows two properties of the Payload class being defined by reference to properties of the same name defined in NIEM Core. The OrganizationItemAssociation and OrganizationContactInformationAssociation property declarations are modeled as properties of «PropertyHolder» classes, independently of their use in the definition of Payload or any other complex type. (This representation may also be used in a PSM.)

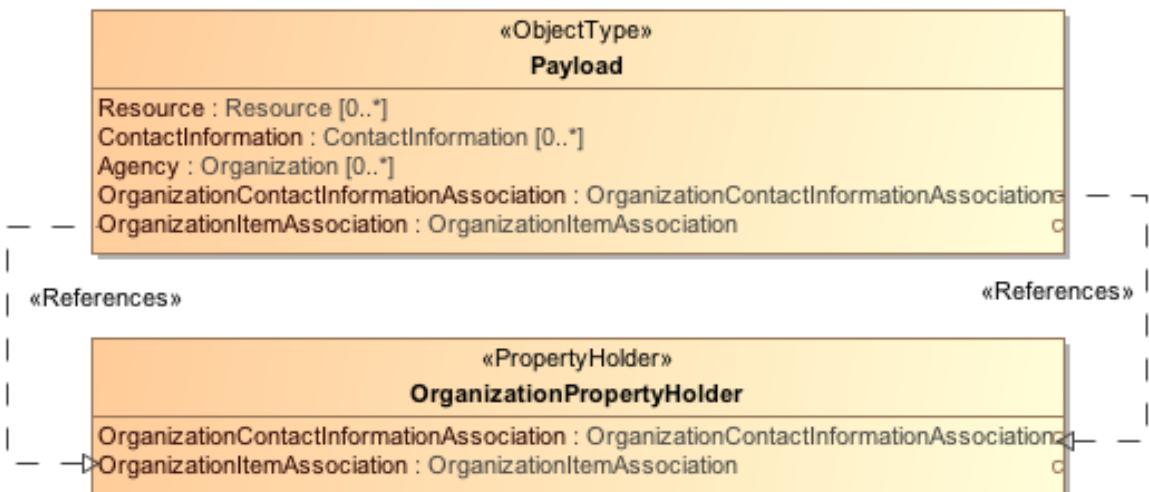
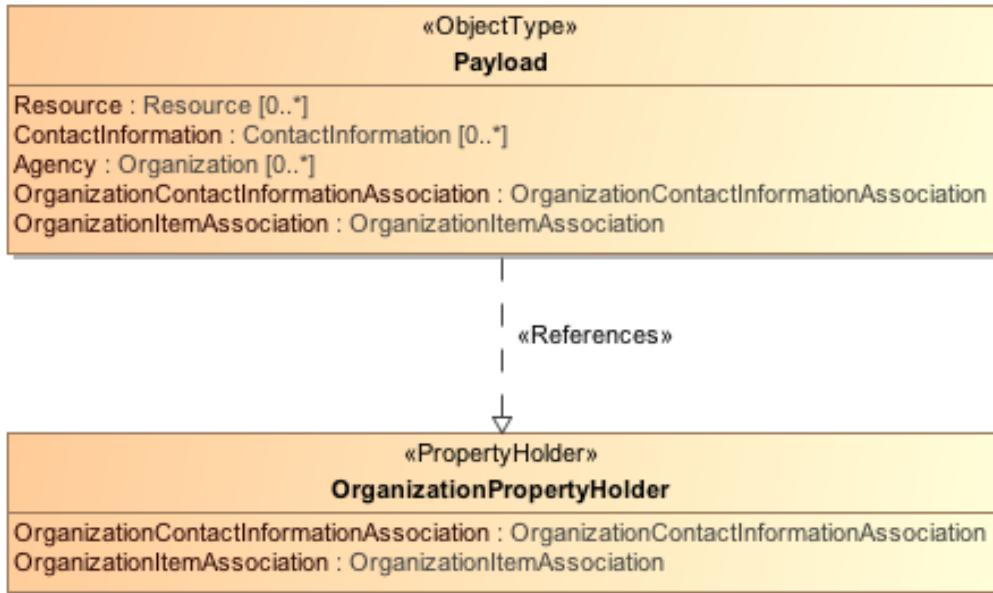


Figure 7-31 Representation of property references using «References» realizations

Figure 7-32 shows an alternative representation of the model shown in Figure 7-31, using a single «Reference» realization between the two classes. Since both of the properties OrganizationItemAssociation and OrganizationContactInformationAssociation in the Payload match the names of properties of the referenced «PropertyHolder» class, these are both considered to be defined by reference. However, the properties Resource, ContactInformation and Agency are defined in the context of their use in the Payload class.



**Figure 7-32 Alternative representation using «References» realizations between classes**

## XML Schema Representation

The property references modeled in Figure 7-31 are represented in XML Schema as follows:

```

<xs:complexType name="PayloadType">
    <xs:annotation>
        <xs:documentation>A data type for describing a payload</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ComplexObjectType">
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="tns:Resource"/>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="tns:ContactInformation"/>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="tns:Agency"/>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="nc:OrganizationContactInformationAssociation"/>
                <xs:element maxOccurs="unbounded" minOccurs="0"
                    ref="nc:OrganizationItemAssociation"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs: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 Subclause 7.2.1) that directly or indirectly contains the owning class of the head property. Members of the substitution 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 *subsetting property*. The well-formedness rules of UML require that a subsetting property be owned either in the same class or a direct or indirect subclass of any subsetted property (see [UML 7.3.45]). However, a «PropertyHolder» class may be used to define substitution group properties independently of any complex type definition (see Subclause 7.5.2).

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

#### PIM

There is no further representation for a PIM.

#### PSM

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

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

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

### 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

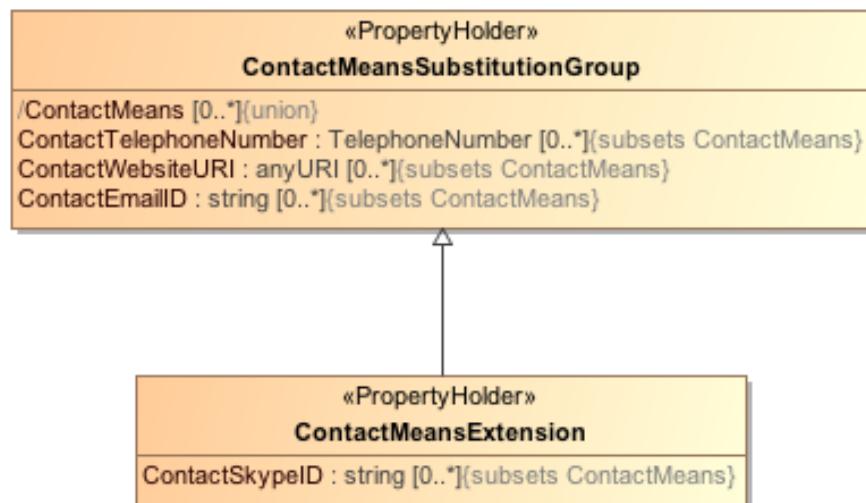
#### PIM Representation

Figure 7-33 shows an example of a substitution group defined in a «PropertyHolder» class as a set of properties that subset the head property ContactMeans. Since ContactMeans is a derived union, it represents an abstract property. The ContactMeans property of the ContactInformation «ObjectType» class is defined by reference to the head property ContactMeans, meaning that any of the properties in the substitution group for ContactMeans is substitutable for ContactMeans in ContactInformation.



**Figure 7-33 Representation of a substitution group using UML suberset properties in a PIM**

Figure 7-34 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.



**Figure 7-34 Extending a substitution group in a PIM or PSM**

#### XML Schema Representation

The substitution group modeled in Figure 7-33 is represented in XML schema as follows:

```

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

<xs:element name="ContactWebsiteURI" nillable="true"
    substitutionGroup="nc:ContactMeans" type="niem-xs:anyURI">
    <xs:annotation>
        <xs:documentation>A website address by which a person or organization may be
        contacted.</xs:documentation>
    </xs:annotation>

```

```

</xs:element>

<xs:element name="ContactTelephoneNumber" nillable="true"
    substitutionGroup="nc>ContactMeans" type="nc:TelephoneNumberType">
    <xs:annotation>
        <xs:documentation>A telephone number for a telecommunication device by which a
person or organization may be contacted.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="ContactEmailID" nillable="true"
    substitutionGroup="nc>ContactMeans" type="niem-xs:string">
    <xs:annotation>
        <xs:documentation>An electronic mailing address by which a person or
organization may be contacted.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:complexType name="ContactInformationType">
    <xs:annotation>
        <xs:documentation>A data type for how to contact a person or an
organization.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ComplexObjectType">
            <xs:sequence>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="nc>ContactMeans"/>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="nc>ContactEntity"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

## 7.5.4 Representations

### 7.5.4.1 Background

NIEM 3 introduces the “Representation” pattern, which is a way to provide multiple representations for a single concept. A NIEM type may contain a representation element, whose name ends with “Representation”; its various representations are in its substitution group. See [NIEM-NDR] [Section 10.7](#).

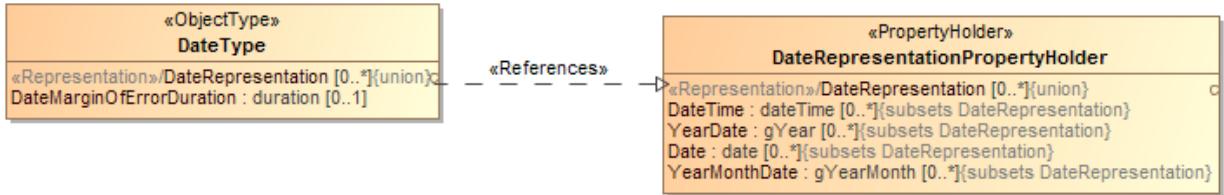
### 7.5.4.2 Representation

A representation element is represented as a UML derived union property stereotyped as «Representation». This is a special case of the substitution group concept described in subclause 7.5.3. All of the constraints and mappings in subclause 7.5.3 apply to Representation properties.

### 7.5.4.3 Example

#### PIM Representation

Figure 7-35 shows a representation property in the type DateType, together with its substitution group defined in a property holder.



**Figure 7-35 «Representation» property**

## XML Schema Representation

The example modeled in Figure 7-35 is represented in XML schema as follows:

```

<xs:complexType name="DateType">
    <xs:annotation>
        <xs:documentation>A data type for a calendar date.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:DateRepresentation" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="nc:DateMarginOfErrorDuration" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="nc:DateAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="DateTime" type="niem-xs:dateTime"
    substitutionGroup="nc:DateRepresentation" nillable="true">
    <xs:annotation>
        <xs:documentation>A full date and time.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="YearDate" type="niem-xs:gYear"
    substitutionGroup="nc:DateRepresentation" nillable="true">
    <xs:annotation>
        <xs:documentation>A year.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="Date" type="niem-xs:date"
    substitutionGroup="nc:DateRepresentation" nillable="true">
    <xs:annotation>
        <xs:documentation>A full date.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:element name="YearMonthDate" type="niem-xs:gYearMonth"
    substitutionGroup="nc:DateRepresentation" nillable="true">

```

```

<xs:annotation>
  <xs:documentation>A year and month.</xs:documentation>
</xs:annotation>
</xs:element>

```

## 7.5.5 Choice Groups

### 7.5.5.1 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.) Choice groups may not be used in NIEM references schemas: they may only be used in extension schemas.

### 7.5.5.2 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 have multiplicity 0..1. 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 `xs: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.

### 7.5.5.3 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 `xs:choice` model group. The property multiplicity gives the occurrence bounds for the group. The properties of the «Choice» class map as properties (see Subclause 7.5.1) to members of the model group. (Note that the «Choice» class does not itself map to a type in the XML schema.)

### 7.5.5.4 Example

#### PIM Representation

Figure 7-36 shows an example of a choice group in which only one of Date or DateTime may have a value. The property DateChoice models the inclusion of the choice group in the complex type represented by the DateType. Note that the names of the «Choice» class and the property that uses it are arbitrary. (The representation in a PSM is similar.)



**Figure 7-36 Representation of a choice group**

## XML Schema Representation

The choice group modeled in Figure 7-36 is represented in XML schema as follows:

```

<xs:complexType name="DateType">
    <xs:annotation>
        <xs:documentation>A data type for a calendar date.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ComplexObjectType">
            <xs:sequence>
                <xs:choice>
                    <xs:element ref="nc:Date" minOccurs="0" maxOccurs="1"/>
                    <xs:element ref="nc:DateTime" minOccurs="0" maxOccurs="1"/>
                </xs:choice>
                <xs:element ref="nc:DateAccuracyCode" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="nc:DateMarginOfErrorDuration" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs: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.

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.

### 7.6.1.2 Representation

#### Common

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

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

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 which 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 which is not nullable in the base model must not be nullable in the subset model. A Property which is nullable in the base model may be declared not nullable in the subset model.
- Complex types within a subset model must conform to the base model. This includes:
  - A property may be removed only if the base property has a cardinality range which 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.
- 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 which 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 which 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» which 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 “subset”, otherwise the defaultPurpose must be the same for both models.

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.

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.

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.

In NIEM 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 and Schematron constraint that enforces the more restrictive subtype. This is the only use of constraint schema in NIEM-UML.

## 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.

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.

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.

### 7.6.1.3 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 a (implicit) constraint schema set and Schematron constraint 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 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».

#### 7.6.1.4 Example

Figure 7-37 shows an example of a small subset model with two classes with properties defined by «Subsets» realizations to corresponding classes in the Reference Model. Figure 7-38 shows an alternative representation of the same model using a «Subsets» realization between the two packages.

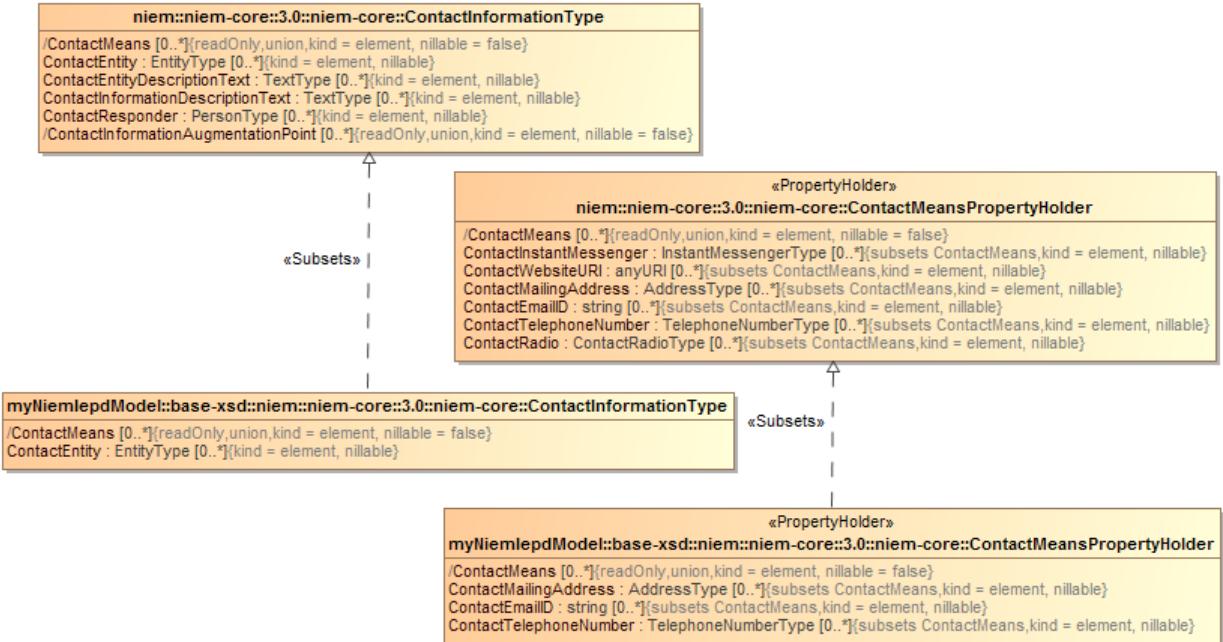


Figure 7-37 Representation of a subset model using «Subsets» realizations between classes

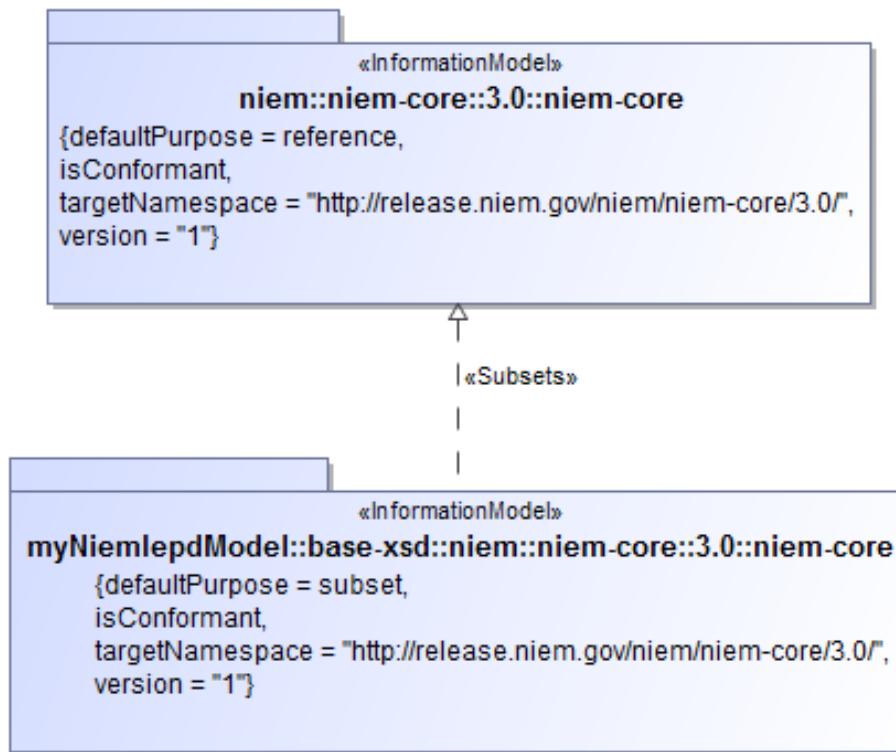


Figure 7-38 Alternative Representation using «Subsets» realization between packages

## 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 its content. See [NIEM-MPD] [Section 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

Most artifacts included in an MPD are modeled as UML InstanceSpecifications classified by one of a number of UML Artifacts included in the NIEM Model Package Description Profile, listed in Table 7-3. Most relationships between MPD artifacts are modeled as stereotyped UML relationships using one of the stereotypes in the NIEM Model Package Description Profile, listed in Table 7-4.

*Note: In this subclause, the term *Artifact* (upper case) is used to mean a UML InstanceSpecification classified by a UML Artifact from the profile. The term *artifact* (lower case) is used to mean a file included in the physical representation of the MPD.*

A MPD is modeled as a ModelPackageDescription Artifact. The slots of the Artifact are used to set the various properties of the MPD. The slot mpdClassCode is set to the class of the MPD, and in the case of an IEPD will be set to ModelPackageDescriptionClassCode::iepd. The ModelPackageDescription Artifact corresponds to the mpd-catalog.xml file that contains metadata describing the complete contents of the physical representation of the MPD.

Each MPD is associated with one or more Conformance Targets (see [NIEM-MPD] [Section 3.2](#)). For an IEPD there is normally an IEP Conformance Target (see [NIEM-MPD] [Section 3.2.3](#) that specifies how Information Exchange Packages (IEPs) will be validated. The IEP conformance target is modeled as an IEPConformanceType Artifact referenced by the IEPConformanceTarget slot of the ModelPackageDescription Artifact. An XMLSchemaType Artifact signifies that the IEP is to be validated by means of an XML Schema, as is normally the case for a NIEM-conformant IEPD. This XMLSchemaType Artifact is referenced by the ValidityConstraintWithContext slot of the IEPConformanceTargetType Artifact, which corresponds to the c:ValidityConstraintWithContext element within the c:IEPConformanceTarget element within the MPD catalog document.

XML Schema artifacts to be included in an MPD are normally modeled by NIEM «InformationModel» packages (see Subclause 7.2.1). Depending on the purpose of the «InformationModel» the artifact's inclusion in the MPD is represented by a UML Usage stereotyped as «ExtensionSchemaDocument», «ReferenceSchemaDocument» or «SubsetSchemaDocument».

Other artifacts to be included in the MPD are represented by Artifacts of corresponding types, related to the elements that correspond to their parents in the MPD catalog by means of links or appropriately stereotyped Usages.

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

### 7.6.2.3 Example

Figure 7-39 is an example of the representation of an MPD together with elements representing a change log, an IEPConformanceTarget, and an AuthoritativeSource element represented as an instance of OrganizationType, linked to an instance of ContactInformationType.

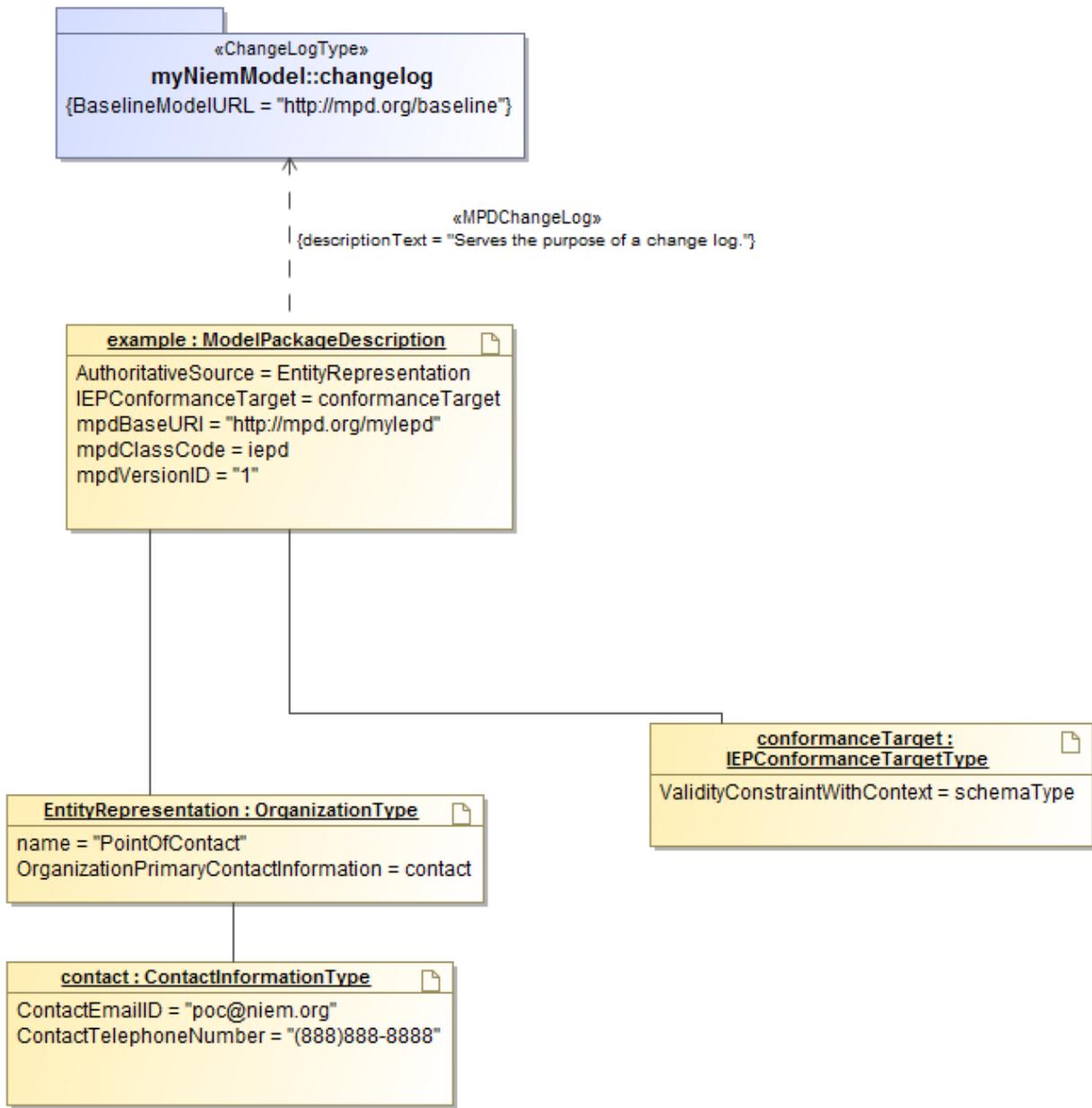


Figure 7-39 Representation of a NIEM MPD

Figure 7-40 is another part of the same example, showing the conformance target instance linked to an XMLSchemaType instance that refers to an extension «InformationModel».

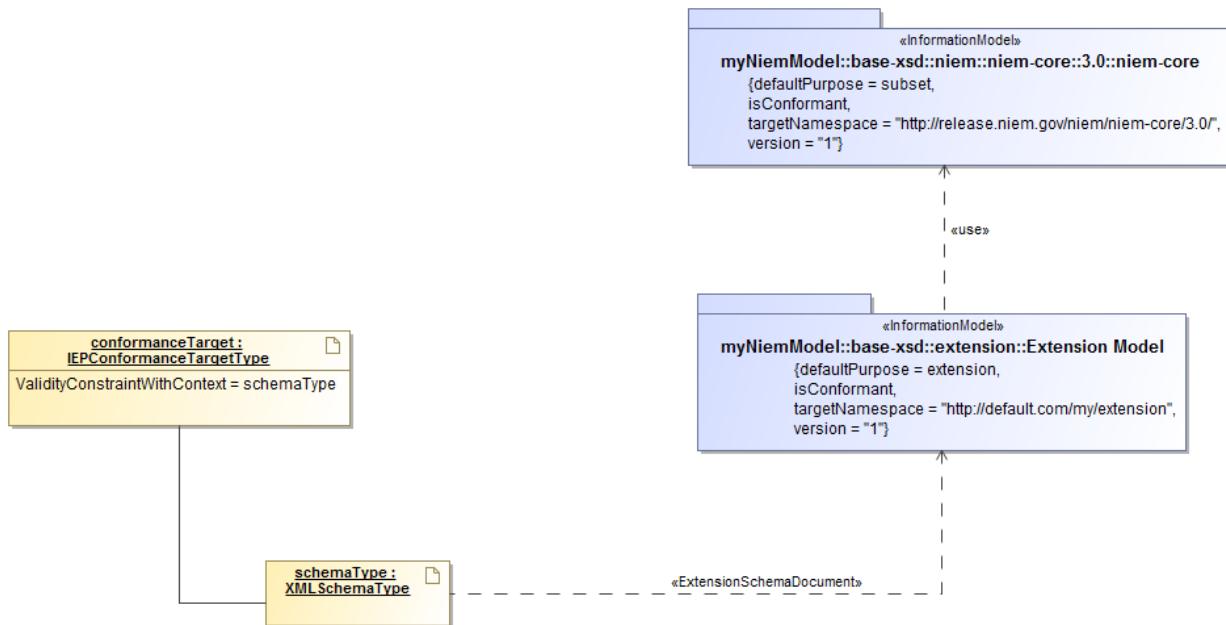


Figure 7-40 An MPD referring to an extension model

## 7.7 Detailed Modeling Design Rules

### 7.7.1 Design Rules Rationale

This non-normative section 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:SimpleObjectTypeAttributeGroup.

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:

- the whiteSpace facet is represented via a tag on «XSDRepresentationRestriction», which may be applied to a DataType.
- enumeration facets are represented as EnumerationLiterals on an Enumeration.
- 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 clauses 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 clauses 7.3 Modeling Complex Types and 7.4 Modeling Simple Types.

### **PSM**

PSM representations for modeling simple restrictions are described in clauses 7.3 Modeling Complex Types and 7.4 Modeling Simple Types.

## **7.7.2.3 Mapping Summary**

### **PIM to PSM Mapping**

PIM to PSM mappings are described in clauses 7.3 Modeling Complex Types and 7.4 Modeling Simple Types.

### **PSM to XML Schema Mapping**

PSM to XML Schema mappings are described in clauses 7.3 Modeling Complex Types and 7.4 Modeling Simple Types.

## **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 which 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 clause 7.3 Modeling Complex Types.

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 clause 7.3 Modeling Complex Types.

#### **PSM**

PSM representations for modeling complex restrictions is described in clause 7.3 Modeling Complex Types.

### **7.7.3.3 Mapping Summary**

#### **PIM to PSM Mapping**

PIM to PSM mappings is described in clause 7.3 Modeling Complex Types.

#### **PSM to XML Schema Mapping**

PSM to XML Schema mappings is described in clause 7.3 Modeling Complex Types.

## **7.7.4 Business Rules**

### **7.7.4.1 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 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.

## **7.7.4.2 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 as Constraints.

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 Schematron, can be represented as a MOF model and consequently be the target artifacts for QVT. In the case of 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.

## **7.7.4.3 Mapping Summary**

### **PIM to PSM Mapping**

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

### **PSM to XML Schema Mapping**

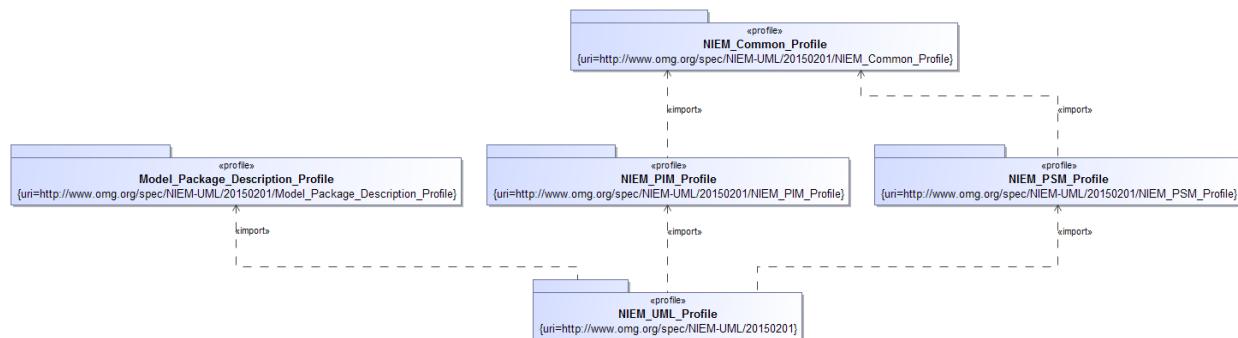
Due to lack of normative specifications within NIEM for business-rules, there are currently no identified artifacts to be targeted during transformation from PSM to MPD.

# 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.



**Figure 8-1 NIEM UML Profiles**

The **NIEM\_Common\_Profile** contains only UML Stereotypes. The **NIEM\_PIM\_Profile** and **NIEM\_PSM\_Profile** contain UML Stereotypes and Enumerations. The **Model\_Package\_Description\_Profile** contains UML Stereotypes, Enumerations and Artifacts, as explained in clause 7.1.4. Each element is documented by a Description; a list of the elements that it generalizes or (for Stereotypes) extends; a list of its properties or (for Enumerations) literals; and a list of its constraints.

Each constraint that corresponds to a rule in the [NIEM-NDR] or [NIEM-MPD] specification is documented with links to that rule and its explanation, and those links should be followed for clarification.

Constraints are specified using OCL or English text. Where the English states that the rule is definitional or non-computable it means that the rule cannot be expressed in OCL. Where it states that the rule is satisfied by provisioning it means that the constraint is satisfied by virtue of the transformation process that generates NIEM artifacts from NIEM-UML models. Where it states that the rule is deferred it means that it may be possible to express the constraint in OCL but that work has not been done in this version of the specification.

## 8.2 Profile : 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.

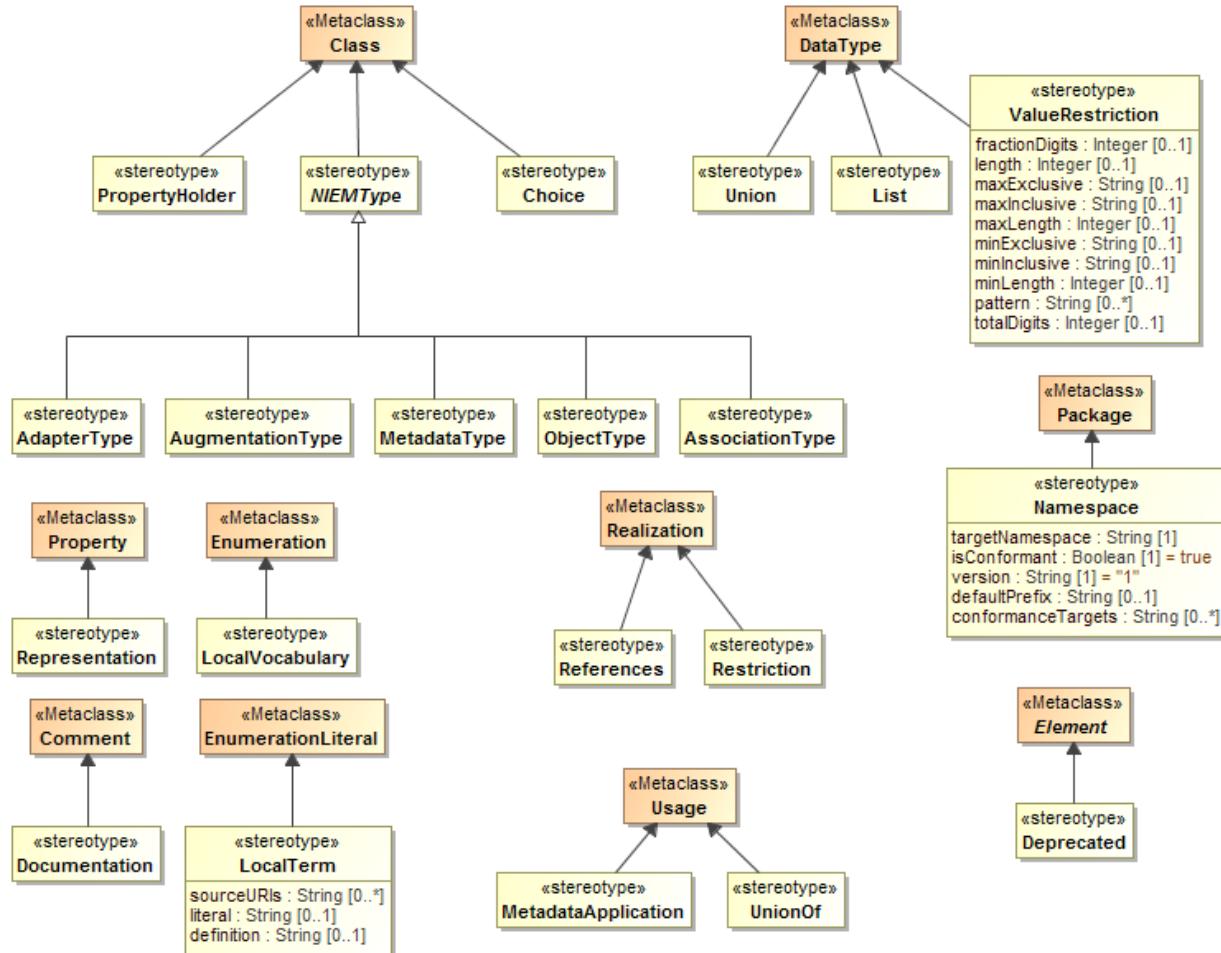


Figure 8-2 NIEM Common Profile

### 8.2.2 <Stereotype> AdapterType

#### 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. See [NIEM NDR] [Section 10.2.3.2, External adapter types](#).

## Generalization

[NIEMType](#)

## Constraints

### NDR3 [Rule 10-11] (REF,EXT). External adapter type not a base type

[Rule 10-11](#), External adapter type not a base type (REF, EXT): [Section 10.2.3.2](#), External adapter types

[OCL] context AdapterType inv:

```
self.base_Class._directedRelationshipOfTarget-  
>select(t|t.oclIsKindOf(Generalization) or t.stereotypedBy('Restriction'))-  
>size()=0
```

### NDR3 [Rule 10-12] (SET). External adapter type not a base type

[Rule 10-12](#), External adapter type not a base type (SET): [Section 10.2.3.2](#), External adapter types

[OCL] context AdapterType inv:

```
self.base_Class._directedRelationshipOfTarget-  
>select(t|t.oclIsKindOf(Generalization) or t.stereotypedBy('Restriction'))-  
>size()=0
```

### NDR3 [Rule 10-69] (REF). External adapter type indicator annotates complex type

[Rule 10-69](#), External adapter type indicator annotates complex type (REF): [Section 10.9.1](#), The NIEM appinfo namespace

[English]

This constraint realized by provisioning:

A Class stereotyped as AdapterType will result in production of appinfo:externalAdapterTypeIndicator attribute on the xs:complexType representing the AdapterType.

### NDR3 [Rule 10-8] (REF,EXT). External adapter type has indicator

[Rule 10-8](#), External adapter type has indicator (REF, EXT): [Section 10.2.3.2](#), External adapter types.

[English]

The constraint is resolved during provisioning:

An AdapterType and only an AdapterType has the appinfo:externalAdapterTypeIndicator set to a value of true.

### NDR3 [Rule 10-9] (REF,EXT). Structure of external adapter type definition follows pattern

[Rule 10-9](#), Structure of external adapter type definition follows pattern (REF, EXT): [Section 10.2.3.2](#), External adapter types.

[OCL] context AdapterType inv:

```

self.base_Class.general->isEmpty()
and
self.base_Class.clientDependency->select(d|d.stereotypedBy('Restriction'))-
>isEmpty()

```

## 8.2.3 <Stereotype> AssociationType

### 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. See [NIEM-NDR] [Section 10.3.1, Association types](#).

### Generalization

[NIEMType](#)

### Constraints

#### NDR3 [Rule 10-19] (REF,EXT). Association types is derived from association type

[Rule 10-19](#), Association types is derived from association type (REF, EXT): [Section 10.3.1, Association types](#)

[OCL] context AssociationType inv:

```

(
  (
    self.stereotypedBy('AssociationType') or
    self.oclisKindOf(AssociationClass)
      and not(self.namespace.oclisUndefined())
      and self.namespace.stereotypedBy('Namespace')
      and
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIE
    M_Common_Profile::Namespace).isConformant
  )
  implies
  (
    self.general
    ->union(self.clientDependency-
    >select(d|d.stereotypedBy('Restriction')).supplier-
    >select(s|s.oclisKindOf(Classifier)).oclAsType(Classifier))
      ->forAll(c|c.stereotypedBy('AssociationType') or
      c.oclisKindOf(AssociationClass))
  )
)

```

```

and
(
    not(self.name.oclIsUndefined())
    and self.niemName().endsWith('AssociationType')
    and not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('Namespace')
    and
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
    (self.stereotypeBy('AssociationType') or
    self.oclIsKindOf(AssociationClass))
)

```

## 8.2.4 <Stereotype> AugmentationType

### 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 «Augments» Realization. AugmentationType is implemented in XML Schema as a complex type definition with complex content. See [NIEM-NDR] [Section 10.4, Augmentations](#).

### Generalization

[NIEMType](#)

### Constraints

#### NDR3 [Rule 10-30] (INS). Element within instance of augmentation type modifies base

[Rule 10-30](#), Element within instance of augmentation type modifies base (INS): [Section 10.4.4, Augmentation types](#) [English]

The instance rule is outside the scope of the NIEM-UML model.

#### NDR3 [Rule 10-31] (REF,EXT). Only an augmentation type name ends in "AugmentationType"

[Rule 10-31](#), Only an augmentation type name ends in AugmentationType (REF, EXT): [Section 10.4.4, Augmentation types](#)

[OCL] context AugmentationType inv:

```

self.niemName().endsWith('AugmentationType') =
self.stereotypeBy('AugmentationType')

```

**NDR3 [Rule 10-32] (REF,EXT). Schema component with name ending in "AugmentationType" is an augmentation type**

[Rule 10-32](#), Schema component with name ending in AugmentationType is an augmentation type (REF, EXT): [Section 10.4.4](#), Augmentation types

[OCL] context AugmentationType inv:

```
(  
    self.stereotypedBy('AugmentationType')  
    or self.niemName().endsWith('AugmentationType'))  
)  
implies  
    self.general  
    ->union(self.clientDependency-  
>select(d|d.stereotypedBy('Restriction')).supplier-  
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))  
    ->forAll(g|g.stereotypedBy('AugmentationType') or  
g.niemName().endsWith('AugmentationType'))
```

**NDR3 [Rule 10-33] (REF,EXT). Type derived from augmentation type is an augmentation type**

[Rule 10-33](#), Type derived from augmentation type is an augmentation type (REF, EXT): [Section 10.4.4](#), Augmentation types

[OCL] context AugmentationType inv:

```
self.stereotypedBy('AugmentationType')  
implies  
self._directedRelationshipOfTarget-  
>select(d|d.oclIsKindOf(Generalization)).oclAsType(Generalization).specific  
->union(self.supplierDependency-  
>select(d|d.stereotypedBy('Restriction')).client-  
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))  
->forAll(g|g.stereotypedBy('AugmentationType'))
```

## 8.2.5 <Stereotype> Choice

### 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. See [NIEM-NDR] Sections [9.3.1.2](#), *Choice* and [9.3.2.2](#), *Choice cardinality*.

### Extends

UML::Class

### Constraints

#### Choice

The ownedAttributes of a Choice class shall have multiplicity 0..1. A Choice Class shall not participate in any Generalizations, either as the general or the special Classifier.

**[OCL] context Choice inv:**

```
(  
    not (self.base_Class.namespace.oclIsUndefined() or  
self.base_Class.namespace.namespace.oclIsUndefined())  
    and self.base_Class.namespace.namespace.stereotypeBy('Namespace')  
    and  
self.base_Class.namespace.namespace.appliedStereotype('Namespace').oclAsType(  
NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies(  
    self.base_Class.attribute->forAll(a| (a.lower=0) and (a.upper=1))  
    and self.base_Class.generalization->isEmpty()  
    and self.base_Class._directedRelationshipOfTarget-  
>select(d|d.oclIsKindOf(Generalization))->isEmpty()  
)
```

## 8.2.6 <Stereotype> Deprecated

### Description

A deprecated component is one whose use is not recommended. A deprecated component may be kept in a schema for support of older versions but should not be used in new efforts. A deprecated component may be removed, replaced, or renamed in a later version of a namespace. See [NIEM-NDR] [Section 10.9.1.1](#), *Deprecation*.

### Extends

UML::Element

### Constraints

#### NDR3 [Rule 10-66] (REF,EXT). Component marked as deprecated is deprecated component

[Rule 10-66](#), Component marked as deprecated is deprecated component (REF, EXT): [Section 10.9.1.1](#), Deprecation [English]

Rule is informative. Provisioning ensure that an appinfo:deprecated maps to a model element stereotyped by Deprecated.

## 8.2.7 <Stereotype> Documentation

### 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.

## Extends

UML::Comment

## Constraints

### Documentation

The owner of a Documentation Comment must have no other Documentation Comments.

#### [OCL] context Documentation inv:

```
self.base_Comment.annotatedElement->notEmpty() and
    self.base_Comment.annotatedElement-
    >forAll(e|e=self.base_Comment.owningElement) and
        (self.base_Comment.owningElement.ownedComment-
    >select(c|c.stereotypeBy('Documentation'))->size()=1)
```

## 8.2.8 <Stereotype> List

### 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 xs: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. See [NIEM-NDR] Sections [9.1.2.1](#), *Simple types prohibited as list item types* and [11.1.2.1](#), *Derivation by list*.

## Extends

UML::DataType

## Constraints

### List

A List DataType shall have a single ownedAttribute with multiplicity 0..\* whose type is also a DataType.

#### [OCL] context List inv:

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

### **NDR3 [Rule 11-6] (REF,EXT). Use lists only when data is uniform**

[Rule 11-6](#), Use lists only when data is uniform (REF, EXT): [Section 11.1.2.1](#), Derivation by list

[English]

Not currently expressed in OCL.

### **NDR3 [Rule 11-7] (REF,EXT). List item type defined by conformant schemas**

[Rule 11-7](#), List item type defined by conformant schemas (REF, EXT): [Section 11.1.2.1](#), Derivation by list

**[OCL] context List inv:**

```
self.base_DataType.attribute.type  
->select(t|t.owner->forAll(p|  
    (p.stereotype('Namespace') and  
    p.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Prof  
ile::Namespace).isConformant)  
    or( p.name='XMLPrimitiveTypes'  
    ))  
)->size()=1
```

### **NDR3 [Rule 9-13](REF,EXT). No list item type of xs:ID**

[Rule 9-13](#), No list item type of xs:ID (REF, EXT): [Section 9.1.2.1](#), Simple types prohibited as list item types

**[OCL] context List inv:**

```
self.base_DataType.attribute.type-  
>exists(t|not((t.name='ID') and(t._'package'.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-14] (REF,EXT). No list item type of xs:IDREF**

[Rule 9-14](#), No list item type of xs:IDREF (REF, EXT): [Section 9.1.2.1](#), Simple types prohibited as list item types

**[OCL] context List inv:**

```
self.base_DataType.attribute.type-  
>exists(t|not((t.name='IDREF') and(t._'package'.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-15] (REF,EXT). No list item type of xs:anySimpleType**

[Rule 9-15](#), No list item type of xs:anySimpleType (REF, EXT): [Section 9.1.2.1](#), Simple types prohibited as list item types

**[OCL] context List inv:**

```
self.base_DataType.attribute.type-  
>exists(t|not((t.name='anySimpleType') and(t._'package'.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-16] (REF,EXT). No list item type of xs:ENTITY**

[Rule 9-16](#), No list item type of xs:ENTITY (REF, EXT): [Section 9.1.2.1](#), Simple types prohibited as list item types

[OCL] context List inv:

```
self.base_DataType.attribute.type->exists(t|not((t.name='ENTITY') and (t._'package'.name='XMLPrimitiveTypes')))
```

## **8.2.9 <Stereotype> LocalTerm**

### **Description**

The LocalTerm stereotype defines a domain-specific word, phrase, acronym, or other string of characters used in a LocalVocabulary. It may occur as a term within the name of a schema component within the schema document. The domain-specific term is represented by the EnumerationLiteral's name. NDR SourceText is represented as UML ownedComment.body. See [NIEM-NDR] [Section 10.8.2.1](#), *Use of Acronyms, Initialisms, Abbreviations, and Jargon*

### **Extends**

UML::EnumerationLiteral

### **Properties**

#### **definition : String [0..1]**

The value of definition is a dictionary-style description of the meaning of the local term.

#### **literal : String [0..1]**

The value of literal is the meaning of the local term, provided as a full, plain-text form of the term. This may be useful when a local term is an abbreviation, acronym, or diminutive form of a longer term.

#### **sourceURIs : String [0..\*]**

The value of sourceURIs is a list of URIs, each of which is an identifier or locator for an originating or authoritative document defining the term.

### **Constraints**

### **NDR3 [Rule 10-74] (REF,EXT). term:LocalTerm annotates schema**

[Rule 10-74](#), term:LocalTerm annotates schema (REF, EXT): [Section 10.9.2](#), The NIEM local terminology namespace

[OCL] context LocalTerm inv:

```
not(self.base_EnumerationLiteral.namespace.oclisUndefined())
and not(self.base_EnumerationLiteral.namespace.namespace.oclisUndefined())
and
self.base_EnumerationLiteral.namespace.namespace.stereotypedBy('Namespace')
```

### **NDR3 [Rule 10-75] (REF,EXT). term:LocalTerm has literal or definition**

[Rule 10-75](#), term:LocalTerm has literal or definition (REF, EXT): [Section 10.9.2](#), The NIEM local terminology namespace

[OCL] context LocalTerm inv:

```
not(self.literal.oclIsUndefined() or self.literal='') or  
not(self.definition.oclIsUndefined() or self.definition='')
```

## **8.2.10 <Stereotype> LocalVocabulary**

### **Description**

Local vocabulary defines a set of domain specific terms or abbreviations that then may be used in NIEM names and definitions. The local vocabulary is defined as a stereotype of Enumeration where each EnumerationLiteral is a vocabulary term represented by the «LocalTerm» stereotype. See [NIEM-NDR] [Section 10.8.2.1, Use of Acronyms, Initialisms, Abbreviations, and Jargon](#).

### **Extends**

UML::Enumeration

## **8.2.11 <Stereotype> MetadataApplication**

### **Description**

The «MetadataApplication» stereotype applies to a Usage between a «MetadataType» Class and either another «MetadataType» Class or a Property. It represents a constraint on a NIEM «MetadataType» that limits the application of the NIEM «MetadataType» to specific schema types or schema elements. 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 «MetadataApplication»s. If a «MetadataType» is not a client of any «MetadataApplication», then it applies to any type. If a Property is the supplier of a «MetadataApplication» Usage, then the allowable elements referencing the «MetadataType» are restricted to the indicator supplier Property and any of its (transitive) substitutions. A «MetadataApplication» Usage with a Class supplier is implemented in NIEM as appinfo:appliesToTypes. A «MetadataApplication» Usage with a Property supplier is implemented in NIEM as appinfo:appliesToElements. See [NIEM-NDR] [Section 10.9.1.2, appinfo:appliesToTypes annotation](#), and [Section 10.9.1.3, appinfo:appliesToElements annotation](#).

### **Extends**

UML::Usage

### **Constraints**

### **NDR3 [Rule 10-70] (REF,EXT). appinfo:appliesToTypes annotates metadata element**

[Rule 10-70](#), appinfo:appliesToTypes annotates metadata element (REF, EXT): [Section 10.9.1.2](#),  
appinfo:appliesToTypes annotation

**[OCL] context** MetadataApplication **inv:**

```
self.base_Usage.client->forAll(g|g.oclIsKindOf(Property) and  
g.oclaType(Property).type.stereotypedBy('MetadataType'))
```

### NDR3 [Rule 10-71] (SET). appinfo:appliesToTypes references types

[Rule 10-71](#), appinfo:appliesToTypes references types (SET): [Section 10.9.1.2](#), appinfo:appliesToTypes annotation

**[OCL] context** MetadataApplication **inv:**

```
self.base_Usage.supplier->forAll(g|g.oclIsKindOf(Classifier) or  
g.oclIsKindOf(Property))
```

### NDR3 [Rule 10-72] (REF,EXT). appinfo:appliesToElements annotates metadata element

[Rule 10-72](#), appinfo:appliesToElements annotates metadata element (REF, EXT): [Section 10.9.1.3](#),  
appinfo:appliesToElements annotation

**[OCL] context** MetadataApplication **inv:**

```
self.base_Usage.client->forAll(g|g.oclIsKindOf(Property) and  
g.oclaType(Property).type.stereotypedBy('MetadataType'))
```

### NDR3 [Rule 10-73] (SET). appinfo:appliesToElements references elements

[Rule 10-73](#), appinfo:appliesToElements references elements (SET): [Section 10.9.1.3](#), appinfo:appliesToElements  
annotation

**[OCL] context** MetadataApplication **inv:**

```
self.base_Usage.supplier->forAll(g|g.oclIsKindOf(Classifier) or  
g.oclIsKindOf(Property))
```

## 8.2.12 <Stereotype> **MetadataType**

### 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 and properties representing the applicable types and elements.

MetadataType is implemented in XML Schema as a complex type definition with complex content. See [NIEM-NDR] [Section 10.5.1](#), *Metadata types*.

## Generalization

[NIEMType](#)

## Constraints

### NDR3 [Rule 10-36] (REF,EXT). Metadata type has data about data

[Rule 10-36](#), Metadata type has data about data (REF, EXT): [Section 10.5.1](#), Metadata types

[English]

Rule is definitional.

### NDR3 [Rule 10-37] (REF,EXT). Metadata type derived from structures:MetadataType

[Rule 10-37](#), Metadata type derived from structures:MetadataType (REF, EXT): [Section 10.5.1](#), Metadata types

[English]

NTAC removed constraint

### NDR3 [Rule 10-38] (REF,EXT). Metadata types are derived from metadata types

[Rule 10-38](#), Metadata types are derived from metadata types (REF, EXT): [Section 10.5.1](#), Metadata types

[OCL] context MetadataType inv:

```
self.base_Class.general->forAll(g|g.stereotypeBy('MetadataType'))  
and  
self.base_Class.clientDependency-  
>select(d|d.stereotypeBy('Restriction')).supplier-  
>forAll(g|g.stereotypeBy('MetadataType'))
```

## 8.2.13 <Stereotype> Namespace

### 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.

### Extends

UML::Package

## Properties

### conformanceTargets : String [0..\*]

The Conformance Targets Attribute Specification defines an attribute that, when it appears in an XML document, claims the document conforms to one or more conformance targets. This pattern and specification was developed to overcome shortcomings in the NIEM 2 `ConformantIndicator` element, and to provide needed capabilities in future specifications. The attribute is a claim of conformance, and not a statement that should be trusted by a validating system. A validator would use this claim to identify to which conformance rules a document should be validated. The attribute's value is a list of internationalized resource identifiers (IRIs). A later specification may define an IRI for its conformance target, and when an XML document has that IRI in its conformance target attribute, the document is claiming to conform to that conformance target. The *effective conformance targets attribute* of a conformant document is the first occurrence of the attribute

{<http://release.niem.gov/niem/conformanceTargets/3.0/>} `conformanceTargets`, in document order.

The only conformance target explicitly defined in NIEM 3 are

<http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/#ReferenceSchemaDocument>

and

<http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/#ExtensionSchemaDocument>

For NIEM-3 UML, the above formally defined conformance targets are implicitly associated with a NIEM target schema based on the `defaultPurpose` of an «InformationModel». Thus, the "conformanceTargets" tag need be populated only with domain-specific conformance target values.

### defaultPrefix : String [0..1]

The default prefix for the namespace, used to represent common NIEM prefixes. This prefix should be used on all XML and/or XML Schema serializations using that 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 : Boolean [1]

Indicates whether the namespace is NIEM-conformant. The targets it conforms to are specified by the `defaultPurpose` of the related «InformationModel», and by its `conformanceTargets` attribute.

### targetNamespace : String [1]

The target namespace URI for this NIEM namespace. It is implemented in XML Schema as the value of the `targetNamespace` attribute on the `xs:schema` document element. Per Rules [9-82](#) and [9-83](#) of [NIEM-NDR], the value of the `targetNamespace` attribute must be present and must be an absolute URI.

### version : String [1]

The version of the NIEM namespace. It is implemented in XML Schema as the value of the `version` attribute on the `xs:schema` document element. Per [Rule 9-84](#) of [NIEM-NDR], the value of the `version` attribute must be present and must not be the empty string. Default is "1".

## Constraints

### Namespace. Can not be contained by Namespace

A «Namespace» package may not be contained, directly or indirectly, in any other «Namespace» package.

**[OCL] context Namespace inv:**

```
not(self.base_Package.nestingPackageoclIsUndefined())
implies
self.base_Package.nestingPackage.nearestNiemNamespace().oclIsUndefined()
```

### **NDR3 [Rule 10-7] (REF,EXT). Import of external namespace has data definition**

[Rule 10-7](#), Import of external namespace has data definition (REF, EXT): [Section 10.2.3.1](#), Import of external namespace.

[English]

This constraint resolved during provisioning;  
in UML, all InformationModels must be documented;  
an xs:import is generated if content of the InformationModel refers to another InformationModel;  
generated xs:import elements from a external namespace will include xs:documentation obtained from their  
documented "external" InformationModels.

### **NDR3 [Rule 7-2] (REF,EXT,INS). Document uses XML namespaces properly**

[Rule 7-2](#), Document uses XML namespaces properly (REF, EXT, INS): [Section 7.2](#), Conformance to XML Namespaces

[English]

self.targetNamespace is namespace-well-formed and namespace-valid.

namespace-valid conformance is enforced during provisioning by ensuring that the production of names is conformant with NDR naming rules and XML Namespaces Specification.  
namespace-well-formed conformance is enforced during provisioning by ensuring that the target xml document is well formed with respected to the XML Namespaces Specification.

### **NDR3 [Rule 7-3] (REF,EXT). Document is a schema document**

[Rule 7-3](#), Document is a schema document (REF, EXT): [Section 7.3](#), Conformance to XML Schema

[English]

Enforced during provisioning to Schema from Namespace Package.

### **NDR3 [Rule 7-4] (REF,EXT). Document element is xs:schema**

[Rule 7-4](#), Document element is xs:schema (REF, EXT): [Section 7.3](#), Conformance to XML Schema

[English]

Enforced during provisioning to Schema from Namespace Package.

### **NDR3 [Rule 7-5] (REF,EXT). Component name follows ISO 11179 Part 5 Annex A**

[Rule 7-5](#), Component name follows ISO 11179 Part 5 Annex A (REF, EXT): [Section 7.5](#), ISO 11179 Part 5

[English]

The default normative naming rules based on ISO 11179-5 are not easily computable, so are not represented as an executable OCL Constraint.

### **NDR3 [Rule 9-10] (REF,EXT). Simple type definition is top-level**

[Rule 9-10](#), Simple type definition is top-level (REF, EXT): [Section 9.1.2](#), Simple type definition

**[OCL] context Namespace inv:**

```
self.isConformant
implies
self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Class)).oclAsType(Class).nestedClassifier->isEmpty()
```

### **NDR3 [Rule 9-1] (REF,EXT). No base type in the XML namespace**

[Rule 9-1](#), No base type in the XML namespace (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies
self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
    c.general->union(c.clientDependency-
        >select(d|d.stereotypeBy('Restriction')).supplier-
            >select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
        ->forAll(g|g._'package'.name<>'xml')
)
```

### **NDR3 [Rule 9-26] (REF,EXT). No mixed content on complex type**

[Rule 9-26](#), No mixed content on complex type (REF, EXT): [Section 9.1.3.1](#), No mixed content

[English]

There is no option in NIEM-UML to specify mixed content, consequently there is no mixed content produced during provisioning of target schemas.

### **NDR3 [Rule 9-27] (REF,EXT). No mixed content on complex content**

[Rule 9-27](#), No mixed content on complex content (REF, EXT): [Section 9.1.3.1](#), No mixed content

[English]

There is no option in NIEM-UML to specify mixed content, consequently there is no mixed content produced during provisioning of target schemas.

### **NDR3 [Rule 9-28] (REF,EXT). Complex type content is explicitly simple or complex**

[Rule 9-28](#), Complex type content is explicitly simple or complex (REF, EXT): [Section 9.1.3](#), Complex type definition

[English]

Complex type content is always enforced to be simple or complex based on defined provisioning.

### **NDR3 [Rule 9-2] (REF,EXT). No base type of xs:ID**

[Rule 9-2](#), No base type of xs : ID (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
  c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
  ->select(t| (t.name='ID') and (t._'package'.name='XMLPrimitiveTypes'))-
>size()=0
)
```

### **NDR3 [Rule 9-3] (REF,EXT). No base type of xs:IDREF**

[Rule 9-3](#), No base type of xs : IDREF (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```

self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
    c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
    ->select(t|(t.name='IDREF') and (t._'package'.name='XMLPrimitiveTypes'))-
>size()=0
)

```

### **NDR3 [Rule 9-44] (REF,EXT). No element default value**

[Rule 9-44](#), No element default value (REF, EXT): [Section 9.2.1.1](#), No element value constraints

[English]

Constraint is realized via Provisioning, which does not create any @default attributes.

### **NDR3 [Rule 9-45] (REF,EXT). No element fixed value**

[Rule 9-45](#), No element fixed value (REF, EXT): [Section 9.2.1.1](#), No element value constraints

[English]

Constraint is realized via Provisioning, which does not create any @fixed attributes.

### **NDR3 [Rule 9-4] (REF,EXT). No base type of xs:IDREFS**

[Rule 9-4](#), No base type of xs:IDREFS (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```

self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
    c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
    ->select(t|(t.name='IDREFS') and
(t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)

```

### **NDR3 [Rule 9-58] (REF,EXT). No use of element xs:notation**

[Rule 9-58](#), No use of element xs : notation (REF, EXT): [Section 9.2.4](#), Notation declaration

[English]

This constraint enforced by provisioning, which does not produce any xs:notation schema components.

### **NDR3 [Rule 9-59] (EXT). Model group does not affect meaning**

[Rule 9-59](#), Model group does not affect meaning (EXT): [Section 9.3.1](#), Model group

[English]

This constraint is not computable.

### **NDR3 [Rule 9-5] (REF,EXT). No base type of xs:anyType**

[Rule 9-5](#), No base type of xs : anyType (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
  c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
  ->select(t| (t.name='anyType') and
  (t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)
```

### **NDR3 [Rule 9-60] (REF,EXT). No xs:all**

[Rule 9-60](#), No xs : all (REF, EXT): [Section 9.3.1](#), Model group

[English]

This constraint enforced by provisioning, there is not model representation for xs:all and no production of an xs:all model group.

### **NDR3 [Rule 9-6] (REF,EXT). No base type of xs:anySimpleType**

[Rule 9-6](#), No base type of xs : anySimpleType (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
    c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
    ->select(t| (t.name='anySimpleType') and
(t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)
```

### NDR3 [Rule 9-7] (REF,EXT). No base type of xs:NOTATION

[Rule 9-7](#), No base type of xs :NOTATION (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
    c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
    ->select(t| (t.name='NOTATION') and
(t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)
```

### NDR3 [Rule 9-82] (REF,EXT). Schema document defines target namespace

[Rule 9-82](#), Schema document defines target namespace (REF, EXT): [Section 9.7](#), Schema as a whole

**[OCL] context Namespace inv:**

```
self.isConformant
implies
(
not(self.targetNamespace.oclIsUndefined()) and
    (self.targetNamespace<>'')
)
```

### NDR3 [Rule 9-83] (REF,EXT). Target namespace is absolute URI

[Rule 9-83](#), Target namespace is absolute URI (REF, EXT): [Section 9.7](#), Schema as a whole

[English]

Specification of this constraint in OCL has been deferred.

### NDR3 [Rule 9-84] (REF,EXT). Schema has version

[Rule 9-84](#), Schema has version (REF, EXT): [Section 9.7](#), Schema as a whole

**[OCL] context Namespace inv:**

```
self.isConformant
implies
(
not(self.version.oclIsUndefined())
and
self.version<>''
)
```

### NDR3 [Rule 9-89] (REF,EXT). xs:import must have namespace

[Rule 9-89](#), xs:import must have namespace (REF, EXT): [Section 9.8](#), Schema assembly

[English]

This constraint resolved during provisioning; all xs:import declarations are implicitly specified by the relationships between Information Model elements and will include the namespace specified in the targetNamespace of the referenced Information Model.

### NDR3 [Rule 9-8] (REF,EXT). No base type of xs:ENTITY

[Rule 9-8](#), No base type of xs:ENTITY (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies

self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
  c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
  ->select(t| (t.name='ENTITY') and
  (t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)
```

### NDR3 [Rule 9-90] (SET). XML Schema document set must be complete

[Rule 9-90](#), XML Schema document set must be complete (SET): [Section 9.8](#), Schema assembly

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

#### **NDR3 [Rule 9-91] (REF,EXT). Namespace referenced by attribute type is imported**

[Rule 9-91](#), Namespace referenced by attribute type is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

#### **NDR3 [Rule 9-92] (REF,EXT). Namespace referenced by attribute base is imported**

[Rule 9-92](#), Namespace referenced by attribute base is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

#### **NDR3 [Rule 9-93] (REF,EXT). Namespace referenced by attribute itemType is imported**

[Rule 9-93](#), Namespace referenced by attribute itemType is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

#### **NDR3 [Rule 9-94] (REF,EXT). Namespaces referenced by attribute memberTypes is imported**

[Rule 9-94](#), Namespaces referenced by attribute memberTypes is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

### **NDR3 [Rule 9-95] (REF,EXT). Namespace referenced by attribute ref is imported**

[Rule 9-95](#), Namespace referenced by attribute `ref` is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

### **NDR3 [Rule 9-96] (REF,EXT). Namespace referenced by attribute substitutionGroup is imported**

[Rule 9-96](#), Namespace referenced by attribute `substitutionGroup` is imported (REF, EXT): [Section 9.8.1](#), Namespaces for referenced components are imported

[English]

This constraint resolved during provisioning; the transitive closure of all schemas referenced will be in the schema document set for an MPD.

### **NDR3 [Rule 9-9] (REF,EXT). No base type of xs:ENTITIES**

[Rule 9-9](#), No base type of `xs:ENTITIES` (REF, EXT): [Section 9.1.1.1](#), Types prohibited as base types

**[OCL] context Namespace inv:**

```
self.isConformant
implies
self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|
  c.general->union(c.clientDependency-
>select(d|d.stereotypeBy('Restriction')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
  ->select(t| (t.name='ENTITIES') and
  (t._'package'.name='XMLPrimitiveTypes'))->size ()=0
)
```

## **8.2.14 <Stereotype> NIEMType**

### **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.

### **Extends**

UML::Class

## Constraints

**NDR3 [Rule 10-48] (REF,EXT).** Name of schema component other than attribute begins with upper case letter

[Rule 10-48](#), Name of schema component other than attribute begins with upper case letter (REF, EXT): [Section 10.8.1](#), Character case

[OCL] context NIEMType inv:

```
(  
    not (self.oclAsType(Classifier).name.oclIsUndefined() or  
(self.oclAsType(Classifier).niemName()=='')  
        and not (self.oclIsKindOf(Association) and not (self.oclIsKindOf(Class)))  
        and not (self.oclAsType(Classifier).namespace.oclIsUndefined())  
        and self.oclAsType(Classifier).namespace.stereotypeBy('Namespace')  
        and not (self.oclAsType(Classifier).stereotypeBy('PropertyHolder') or  
self.oclAsType(Classifier).stereotypeBy('LocalVocabulary'))  
        and  
self.oclAsType(Classifier).namespace.appliedStereotype('Namespace').oclAsType  
(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
(self.oclAsType(Classifier).niemName().firstToUpper()=self.oclAsType(Classifier).niemName())
```

**NDR3 [Rule 10-49] (REF,EXT).** Names use common abbreviations

[Rule 10-49](#), Names use common abbreviations (REF, EXT): [Section 10.8.2](#), Use of acronyms and abbreviations

[OCL] context NIEMType inv:

```
(  
    not (self.oclAsType(Classifier).name.oclIsUndefined() or  
self.oclAsType(Classifier).namespace.oclIsUndefined())  
        and not (self.oclIsKindOf(Association) and not (self.oclIsKindOf(Class)))  
        and self.oclAsType(Classifier).namespace.stereotypeBy('Namespace')  
        and  
self.oclAsType(Classifier).namespace.appliedStereotype('Namespace').oclAsType  
(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
        and not (self.oclAsType(Classifier).stereotypeBy('PropertyHolder'))  
)  
implies  
not (self.oclAsType(Classifier).niemName().match('.*Identifier.*') or  
self.oclAsType(Classifier).niemName().match('.*UniformResourceIdentifier.*'))
```

**NDR3 [Rule 11-36] (SET).** Reference schema imports reference schema

[Rule 11-36](#), Reference schema imports reference schema (SET): [Section 11.8](#), Schema assembly

**[OCL] context NIEMType inv:**

```
(  
    not (self.namespace.oclIsUndefined())  
    and self.namespace.stereotypedBy('InformationModel')  
    and  
    self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profil  
e::NIEM_PIM_Profile::InformationModel).isConformant  
    and  
    self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profil  
e::NIEM_PIM_Profile::InformationModel).defaultPurpose->exists(purpose|  
  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)  
        or  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)  
    )  
)  
implies  
    self.oclAsType(Classifier).general.namespace.oclAsType(NamedElement)->  
    >asSet()  
        ->union(self.clientDependency-  
    >select(d|d.stereotypedBy('Restriction')).supplier-  
    >select(s|s.oclIsKindOf(Classifier)).namespace.oclAsType(NamedElement)->  
    >asSet())  
        ->forAll(p|  
            (p.stereotypedBy('InformationModel')  
                and  
                p.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_<br>Profile::InformationModel).defaultPurpose->exists(purpose|  
  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)  
                or  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)  
            )  
        )  
        or (p.name='XMLPrimitiveTypes')  
)
```

**NDR3 [Rule 9-25] (REF,EXT). Complex type has data definition**

[Rule 9-25](#), Complex type has data definition (REF, EXT): [Section 9.1.3](#), Complex type definition

**[OCL] context NIEMType inv:**

```
(  
    not (self.namespace.oclIsUndefined())  
    and self.namespace.stereotypedBy('Namespace')  
    and  
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIE  
M_Common_Profile::Namespace).isConformant  
        and not(self.stereotypedBy('PropertyHolder'))  
)  
implies
```

```

    self.ownedComment->exists(c|not(c._'body'.oclIsUndefined()) and
(c._'body'<>''))

```

### NDR3 [Rule 9-29] (REF). Complex content uses extension

[Rule 9-29](#), Complex content uses extension (REF): [Section 9.1.3.2](#), Complex content

#### [OCL] context NIEMType inv:

```

(
    not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('InformationModel')
    and
self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi
le::NIEM_PIM_Profile::InformationModel).isConformant

    and
self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi
le::NIEM_PIM_Profile::InformationModel).defaultPurpose->exists(purpose|
((purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)o
r(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset))
)
    and not(self.oclIsKindOf(Enumeration) and
self.oclAsType(Enumeration).ownedLiteral->notEmpty())
)
implies
    self.oclAsType(Classifier).clientDependency-
>select(d|d.stereotypeBy('Restriction'))->isEmpty()

```

### NDR3 [Rule 9-32] (REF). Simple content uses extension

[Rule 9-32](#), Simple content uses extension (REF): [Section 9.1.3.3](#), Simple content

#### [OCL] context NIEMType inv:

```

(
    not(self.oclAsType(Classifier).namespace.oclIsUndefined())
    and
self.oclAsType(Classifier).namespace.stereotypeBy('InformationModel')
    and
self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi
le::NIEM_PIM_Profile::InformationModel).isConformant
    and
self.oclAsType(Classifier).namespace.appliedStereotype('InformationModel').oc
lAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose-
>exists(purpose|

```

```

(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference) or
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)
)
    and not(self.oclIsKindOf(Enumeration) and
self.oclAsType(Enumeration).ownedLiteral->notEmpty())
)
implies
self.oclAsType(Classifier).clientDependency-
>select(d|d.stereotypeBy('Restriction'))->isEmpty()

```

## 8.2.15 <Stereotype> ObjectType

### 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. See [NIEM-NDR] [Section 10.2.1](#), *General object types*.

### Generalization

[NIEMType](#)

### Constraints

#### NDR3 [Rule 10-18] (REF,EXT). Proxy type has designated structure

[Rule 10-18](#), Proxy type has designated structure (REF, EXT): [Section 10.2.5](#), Proxy types

#### [OCL] context ObjectType inv:

```

(
    self.general-
>exists(g| (g.namespace.name='XMLPrimitiveTypes') and (g.name=self.name) )
    or
    self.clientDependency-
>select(d|d.stereotypeBy('XSDSimpleContent')).supplier
        ->select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)
        -
>exists(g| (g.namespace.name='XMLPrimitiveTypes') and (g.name=self.name) )
)
implies
self.attribute->isEmpty()

```

#### NDR3 [Rule 10-21] (REF). Augmentable type has augmentation point element

[Rule 10-21](#), Augmentable type has augmentation point element (REF): [Section 10.4.1](#), Augmentable types

[English]

The constraint is enforced by provisioning; if an AugmentationPoint is not defined in the model, then it is created

### **NDR3 [Rule 10-22] (REF,EXT). Augmentable type has at most one augmentation point element**

[Rule 10-22](#), Augmentable type has at most one augmentation point element (REF, EXT): [Section 10.4.1](#), Augmentable types

**[OCL] context ObjectType inv:**

```
self.attribute->select(a|self.niemName().replace('Type', 'AugmentationPoint')=a.niemName())->size()<=1
```

### **NDR3 [Rule 10-23] (REF,EXT). Augmentation point corresponds to augmentable type**

[Rule 10-23](#), Augmentation point corresponds to augmentable type (REF, EXT): [Section 10.4.2](#), Augmentation point element declarations

**[OCL] context ObjectType inv:**

```
self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))->forAll(a|self._'package'.ownedType.niemName()->exists(n|n.replace('Type', 'AugmentationPoint')=a.niemName()))
```

### **NDR3 [Rule 10-24] (REF,EXT). An augmentation point has no type**

[Rule 10-24](#), An augmentation point has no type (REF, EXT): [Section 10.4.2](#), Augmentation point element declarations

**[OCL] context ObjectType inv:**

```
self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))->forAll(a|a.type.oclIsUndefined())
```

### **NDR3 [Rule 10-25] (REF,EXT). An augmentation point has no substitution group**

[Rule 10-25](#), An augmentation point has no substitution group (REF, EXT): [Section 10.4.2](#), Augmentation point element declarations

**[OCL] context ObjectType inv:**

```
self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))->forAll(a|a.subsettedProperty->isEmpty())
```

### **NDR3 [Rule 10-26] (REF,EXT). Augmentation point element may only be referenced by its type**

[Rule 10-26](#), Augmentation point element may only be referenced by its type (REF, EXT): [Section 10.4.3](#), Augmentation point element use

**[OCL] context** ObjectType **inv:**

```
not(self.stereotypeBy('PropertyHolder'))  
implies  
self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))  
->forAll(a|a.niemName().replace('AugmentationPoint', 'Type')=self.niemName())
```

### NDR3 [Rule 10-27] (REF). Augmentation point reference is optional

[Rule 10-27](#), Augmentation point reference is optional (REF): [Section 10.4.3](#), Augmentation point element use

**[OCL] context** ObjectType **inv:**

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    self.namespace.stereotypeBy('InformationModel')  
    and  
  
    self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi  
le::NIEM_PIM_Profile::InformationModel)  
    -  
    >forAll(im| (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPur  
poseCode::subset) or (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::Defa  
ultPurposeCode::reference))  
)  
implies  
self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))  
->forAll(a|a.lower=0)
```

### NDR3 [Rule 10-28] (REF). Augmentation point reference is unbounded

[Rule 10-28](#), Augmentation point reference is unbounded (REF): [Section 10.4.3](#), Augmentation point element use

**[OCL] context** ObjectType **inv:**

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    self.namespace.stereotypeBy('InformationModel')  
    and  
  
    self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi  
le::NIEM_PIM_Profile::InformationModel)  
    ->forAll(im|  
  
        (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::su  
bset) or  
  
        (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::re  
ference))  
)  
implies
```

```

self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))
->forAll(a|a.upper<0)

```

### **NDR3 [Rule 10-29] (REF). Augmentation point reference must be last particle**

[Rule 10-29](#), Augmentation point reference must be last particle (REF, EXT): [Section 10.4.3](#), Augmentation point element use

[OCL] context ObjectType inv:

```

(
    not (self.namespace.oclIsUndefined())
    and
    self.namespace.stereotypeBy('informationModel')
    and

    self.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profi
    le::NIEM_PIM_Profile::InformationModel)
        ->forAll(im|
            (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::su
            bset) or

            (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::re
            ference))
        )
    implies
    self.attribute->select(a|a.niemName().endsWith('AugmentationPoint'))
    ->forAll(a|self.attribute->last()=a)

```

### **NDR3 [Rule 11-11] (REF,EXT). Complex type with simple content has structures:SimpleObjectAttributeGroup**

[Rule 11-11](#), Complex type with simple content has structures:SimpleObjectAttributeGroup (REF, EXT): [Section 11.1.3](#), Complex type definition

[English]

This constraint realized during provisioning;

The structures:SimpleObjectAttributeGroup is not in the UML-NIEM model, it is produced as required during construction of a complex type with simple content

### **NDR3 [Rule 11-1] (REF,EXT). Name of type ends in "Type"**

[Rule 11-1](#), Name of type ends in Type (REF, EXT): [Section 11.1](#), Type definition components

[OCL] context ObjectType inv:

```

(
    not (self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('Namespace')

```

```

        and
self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_
M_Common_Profile::Namespace).isConformant
    and not(self.stereotypedBy('PropertyHolder'))
    and not(self.stereotypedBy('LocalVocabulary'))
    and self.general-
>select(g| (g.name=self.name) and (g.namespace.name='XMLPrimitiveTypes'))-
>isEmpty()
)
implies
self.niemName().endsWith('Type')

```

### **NDR3 [Rule 11-2] (REF,EXT). Name of type other than proxy type is in upper camel case**

[Rule 11-2](#), Name of type other than proxy type is in upper camel case (REF, EXT): [Section 11.1](#), Type definition components

**[OCL] context** ObjectType **inv:**

```

(
    not(self.namespace.oclisUndefined())
    and not(self.oclisKindOf(Association) and not(self.oclisKindOf(Class)))
    and self.namespace.stereotypedBy('InformationModel')
        and not(self.stereotypedBy('PropertyHolder'))
        and not(self.stereotypedBy('LocalVocabulary'))
        and self.general-
>select(g| (g.name=self.name) and (g.namespace.name='XMLPrimitiveTypes'))-
>isEmpty()
)
implies
self.niemName().match('^([A-Z][A-Za-z0-9\\-]*)+$')
```

### **NDR3 [Rule 11-32] (REF,EXT). Standard opening phrase for complex type**

[Rule 11-32](#), Standard opening phrase for complex type (REF, EXT): [Section 11.6.1.1](#), Data definition opening phrases

**[OCL] context** ObjectType **inv:**

```

(
    not(self.stereotypedBy('LocalVocabulary'))
    and
    not(self.stereotypedBy('PropertyHolder'))
    and not(self.name.oclisUndefined())
    and not(self.namespace.oclisUndefined())
    and self.namespace.stereotypedBy('Namespace')
    and
self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_
M_Common_Profile::Namespace).isConformant
)
```

```

implies
if (self.niemName().endsWith('AssociationType')) then
self.ownedComment._'body'.toLower().normalizeSpace()->exists(b|b.match('a
data type for (a relationship|an association).*'))
else if (self.niemName().endsWith('AugmentationType')) then
self.ownedComment._'body'.toLower().normalizeSpace()->exists(b|b.match('a
data type (that supplements|for additional information about).*'))
else if (self.niemName().endsWith('MetadataType')) then
self.ownedComment._'body'.toLower().normalizeSpace()->exists(b|b.match('a
data type for (metadata about|information that further qualifies).*'))
else self.ownedComment._'body'.toLower().normalizeSpace()-
>exists(b|b.match('a data type.*'))
endif
endif
endif

```

### **NDR3 [Rule 11-3] REF,EXT. Base type definition defined by conformant schema**

[Rule 11-3](#), Base type definition defined by conformant schema (REF, EXT): [Section 11.1.1](#), Type definition hierarchy

**[OCL] context** ObjectType **inv:**

```

(
    not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('Namespace')
    and
self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_
Common_Profile::Namespace).isConformant
)
implies
self.generalization->select(x|not(x.stereotypeBy('RolePlayedBy') or
x.stereotypeBy('Augments'))).general
->union(self.clientDependency->select(d|d.stereotypeBy('Restriction') or
d.stereotypeBy('XSDSimpleContent')).supplier-
>select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier))
->select(g|not(g.namespace.oclIsUndefined()) and
not(g.namespace.name='XMLPrimitiveTypes')).namespace
->forAll(g|
    g.stereotypeBy('Namespace')

and(g.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_
Profile::Namespace).isConformant)
)
```

### **NDR3 [Rule 11-4] (REF,EXT). Name of simple type ends in "SimpleType"**

[Rule 11-4](#), Name of simple type ends in SimpleType (REF, EXT): [Section 11.1.2](#), Simple type definition

**[OCL] context** ObjectType **inv:**

```

(
    not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('Namespace')
    and
self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
and
(
    (self.oclIsKindOf(Enumeration) and self.oclAsType
(Enumeration).ownedLiteral->notEmpty() and
self._directedRelationshipOfTarget->select(d|d.oclIsKindOf(Generalization))->notEmpty())
    or
    self.stereotypeBy('List')
    or
    self.stereotypeBy('Union')
    or
    self.stereotypeBy('ValueRestriction')
    or
    self.stereotypeBy('XSDRepresentationRestriction')
    or
    self.supplierDependency->exists(c|c.stereotypeBy('XSDSimpleContent'))
)
implies
self.niemName().endsWith('SimpleType')

```

### **NDR3 [Rule 11-5] (REF,EXT). Name of simple type is upper camel case**

[Rule 11-5](#), Name of simple type is upper camel case (REF, EXT): [Section 11.1.2](#), Simple type definition.

**[OCL] context ObjectType inv:**

```

(
    not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypeBy('Namespace')
    and
self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
and
(
    (self.oclIsKindOf(Enumeration) and self.oclAsType
(Enumeration).ownedLiteral->notEmpty())
    or
    self.stereotypeBy('List')
    or
    self.stereotypeBy('Union')
    or
    self.stereotypeBy('ValueRestriction')
    or
    self.stereotypeBy('XSDRepresentationRestriction')
    or
    self.supplierDependency->exists(c|c.stereotypeBy('XSDSimpleContent'))
)
```

```

implies
self.niemName().match('^([A-Z][A-Za-z0-9\-\-]*+$')

```

### **NDR3 [Rule 9-24] (REF,EXT). Complex type definitions is top-level**

[Rule 9-24](#), Complex type definitions is top-level (REF, EXT): [Section 9.1.3](#), Complex type definition

#### **[OCL] context ObjectType inv:**

```

(
    not(self.nearestNiemNamespace().oclIsUndefined())
    and
    self.nearestNiemNamespace().appliedStereotype('Namespace').oclAsType(NIEM_UML
    _Profile::NIEM_Common_Profile::Namespace).isConformant
    and (self.oclIsKindOf(Class) or self.oclIsKindOf(DataType))
)
implies
(
    self.namespace.stereotypedBy('Namespace')
    and
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIE
    M_Common_Profile::Namespace).isConformant
)

```

### **ObjectType type name unique in namespace**

Each type name must be unique within the schema type symbol space.

#### **[OCL] context ObjectType inv:**

```

(
    not(self.stereotypedBy('LocalVocabulary'))
    and not(self.stereotypedBy('PropertyHolder'))
    and not(self.name.oclIsUndefined())
    and not(self.namespace.oclIsUndefined())
    and self.namespace.stereotypedBy('Namespace')
    and
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIE
    M_Common_Profile::Namespace).isConformant
)
implies
    self.namespace.oclaType(Package).ownedType
    ->select(t| t<>self) and not(self.stereotypedBy('LocalVocabulary'))
and not(self.stereotypedBy('PropertyHolder')))
->forAll(t| self.niemName()<>t.niemName())

```

## **8.2.16 <Stereotype> PropertyHolder**

### **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 [9.2.1](#) and [9.2.3](#) of [NIEM-NDR], Rule [9-35](#) and Rule [9-47](#), require that an attribute or element declaration be a top-level declaration, but [NIEM-NDR] 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.

### **Extends**

UML::Class

## **8.2.17 <Stereotype> References**

### **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.

### **Extends**

UML::Realization

### **Constraints**

### **References**

References may only be between packages, classifiers or properties and the metatype of the client must be the same as the metatype of the supplier.

NIEM subsets may omit elements with zero cardinality and adjust the cardinality of elements in reference schemas from which they are derived, as long as the subset property is maintained.

**[OCL] context References inv:**

```
self.base_Realization.client->forAll(c|self.base_Realization.supplier-
>forAll(s|s.oclType()=c.oclType()))
and
(
  ( 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.niemName()=supplierAttribute.niemName())
      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.niemName()=supplierAttribute.niemName
()) )
  )
)
```

## 8.2.18 <Stereotype> Representation

### Description

One need frequently faced by schema developers is for multiple representations for a single concept. To handle this need, NIEM has adopted the Representation pattern, in which a type may contain a representation element, and the various representations for that element type are in the substitution group for that representation element. In NIEM-3 UML, the Representation concept may be expressed as an abstract type-less Property whose name has a suffix of

"Representation". Alternatively, an abstract type-less Property Stereotyped by «Representation» may be used to represent the Representation concept, in which case the NIEM naming rule for Representation elements will be implicitly applied during transformation to the target schema element. See [NIEM-NDR] [Section 10.7](#), *The "Representation" pattern*.

## Extends

UML::Property

## Constraints

### NDR3 [Rule 10-41] (REF,EXT). Name of element that ends in "Representation" is abstract

[Rule 10-41](#), Name of element that ends in Representation is abstract (REF, EXT): [Section 10.7](#), The Representation pattern

[OCL] context Representation inv:

```
(  
  (not (self.namespace.oclIsUndefined()))  
  and not (self.namespace.namespace.oclIsUndefined())  
  and self.namespace.namespace.stereotypedBy('InformationModel')  
  and  
  self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
  and not (self.name.oclIsUndefined())  
  and self.niemName().endsWith('Representation')  
  )  
  implies  
  self.stereotypedBy('Representation')  
)  
and  
(  
  self.stereotypedBy('Representation')  
  implies  
  self.type.oclIsUndefined()  
)
```

### NDR3 [Rule 10-42] (REF,EXT). A substitution for a representation element declaration is a value for a type

[Rule 10-42](#), A substitution for a representation element declaration is a value for a type (REF, EXT): [Section 10](#), Rules for NIEM modeling, by NIEM concept

[English]

Rule is definitional.

## 8.2.19 <Stereotype> Restriction

### 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 `xs: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. NIEM does not support the use of complex type restriction in reference schemas, because the use of restriction in a reference schema would reduce the ability for that schema to be reused. Restriction may be used in extension schemas. Section 3.4 and 3.14 of [XML Schema Structures](#) addresses the use of restriction in XML Schema.

### Extends

UML::Realization

### Constraints

#### Restriction

If the general Classifier is a DataType, the specific Classifier must be a DataType.

If the general Classifier is a NIEMType that is not the client of an «XSDSimpleContent» Realization, the specific Classifier must be a NIEMType that is not the client of an «XSDSimpleContent» Realization.

If the general Classifier is a NIEMType that is the client of an «XSDSimpleContent» Realization, the specific Classifier must be a NIEMType that is the client of a «XSDSimpleContent» Realization.

#### [OCL] context Restriction inv:

```
(self.base_Realization.supplier->forAll(o|o.oclIsKindOf(DataType)) implies  
self.base_Realization.client->forAll(o|o.oclIsKindOf(DataType)))  
and (  
    self.base_Realization.supplier.clientDependency-  
>select(d|d.stereotypedBy('NIEMSsimpleContent'))->isEmpty()  
    =self.base_Realization.client.clientDependency-  
>select(d|d.stereotypedBy('NIEMSsimpleContent'))->isEmpty()  
)
```

## 8.2.20 <Stereotype> Union

### 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 `xs: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.

## Extends

UML::DataType

## Constraints

### NDR3 [Rule 11-8] (REF,EXT). Union member types defined by conformant schemas

[Rule 11-8](#), Union member types defined by conformant schemas (REF, EXT): [Section 11.1.2.2](#), Derivation by union

[OCL] context Union inv:

```
(  
    not (self.base_DataType.namespace.oclIsUndefined())  
    and self.base_DataType.namespace.stereotypeBy('InformationModel')  
    and  
    self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(  
        NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant  
)  
implies  
    self.base_DataType.clientDependency-  
    >select (d|d.stereotypeBy('UnionOf')).supplier-  
    >select (s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)  
    -  
    >forAll (t| (t._'package'.name='XMLPrimitiveTypes') or (t._'package'.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant))
```

### NDR3 [Rule 9-17] (REF,EXT). No union member types of xs:ID

[Rule 9-17](#), No union member types of `xs:ID` (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

[OCL] context Union inv:

```
(  
    not (self.base_DataType.namespace.oclIsUndefined())  
    and self.base_DataType.namespace.stereotypeBy('InformationModel')  
    and  
    self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(  
        NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant  
)  
implies  
    self.base_DataType.clientDependency  
    ->select (d|d.stereotypeBy('UnionOf')).supplier  
    ->select (s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)  
    ->forAll (t| (t.name<>'ID') and (t._'package'.name<>'XMLPrimitiveTypes'))
```

### **NDR3 [Rule 9-18] (REF,EXT). No union member types of xs:IDREF**

[Rule 9-18](#), No union member types of xs:IDREF (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

#### **[OCL] context Union inv:**

```
(  
    not (self.base_DataType.namespace.oclIsUndefined())  
    and self.base_DataType.namespace.stereotypeBy('InformationModel')  
    and  
    self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(  
        NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant  
)  
implies  
    self.base_DataType.clientDependency  
        ->select (d|d.stereotypeBy('UnionOf')).supplier  
        ->select (s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)  
->forAll (t| (t.name<>'IDREF') and (t._'package'.name<>'XMLPrimitiveTypes'))
```

### **NDR3 [Rule 9-19] (REF,EXT). No union member types of xs:IDREFS**

[Rule 9-19](#), No union member types of xs:IDREFS (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

#### **[OCL] context Union inv:**

```
(  
    not (self.base_DataType.namespace.oclIsUndefined())  
    and self.base_DataType.namespace.stereotypeBy('InformationModel')  
    and  
    self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(  
        NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant  
)  
implies  
    self.base_DataType.clientDependency  
        ->select (d|d.stereotypeBy('UnionOf')).supplier  
        ->select (s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)  
->forAll (t| (t.name<>'IDREFS') and (t._'package'.name<>'XMLPrimitiveTypes'))
```

### **NDR3 [Rule 9-20] (REF,EXT). No union member types of xs:anySimpleType**

[Rule 9-20](#), No union member types of xs:anySimpleType (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

#### **[OCL] context Union inv:**

```

(
    not(self.base_DataType.namespace.oclIsUndefined())
    and self.base_DataType.namespace.stereotypeBy('InformationModel')
    and
self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(
NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant
)
implies
    self.base_DataType.clientDependency
    ->select(d|d.stereotypeBy('UnionOf')).supplier
    ->select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)
-
>forAll(t| (t.name<>'anySimpleType') and(t._'package'.name<>'XMLPrimitiveTypes'
))

```

### **NDR3 [Rule 9-21] (REF,EXT). No union member types of xs:ENTITY**

[Rule 9-21](#), No union member types of xs : ENTITY (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

#### **[OCL] context Union inv:**

```

(
    not(self.base_DataType.namespace.oclIsUndefined())
    and self.base_DataType.namespace.stereotypeBy('InformationModel')
    and
self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(
NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant
)
implies
    self.base_DataType.clientDependency
    ->select(d|d.stereotypeBy('UnionOf')).supplier
    ->select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)
->forAll(t| (t.name<>'ENTITY') and(t._'package'.name<>'XMLPrimitiveTypes'))

```

### **NDR3 [Rule 9-22] (REF,EXT). No union member types of xs:ENTITIES**

[Rule 9-22](#), No union member types of xs : ENTITIES (REF, EXT): [Section 9.1.2.2](#), Simple types prohibited as union member types

#### **[OCL] context Union inv:**

```

(
    not(self.base_DataType.namespace.oclIsUndefined())
    and self.base_DataType.namespace.stereotypeBy('InformationModel')
    and
self.base_DataType.namespace.appliedStereotype('InformationModel').oclAsType(
NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).isConformant
)
```

```

implies
self.base_DataType.clientDependency
->select(d|d.stereotypeBy('UnionOf')).supplier
->select(s|s.oclIsKindOf(Classifier)).oclAsType(Classifier)
->forAll(t| (t.name<>'ENTITIES') and(t._'package'.name<>'XMLPrimitiveTypes'))

```

## **Union**

A Union shall not have any ownedAttributes. A Union shall not have any generalizations.

### **[OCL] context Union inv:**

```

self.base_DataType.attribute->isEmpty()
and
self.base_DataType.generalization->isEmpty()

```

## **8.2.21 <Stereotype> UnionOf**

### **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.

### **Extends**

UML::Usage

### **Constraints**

#### **UnionOf**

The supplier must be a DataType that represents a legal union member type. The client must be a union DataType.

### **[OCL] context UnionOf inv:**

```

self.base_Usage.supplier->forAll(s|s.oclIsKindOf(DataType))
and
self.base_Usage.client->forAll(c|c.stereotypeBy('Union'))

```

## 8.3 Profile : 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.

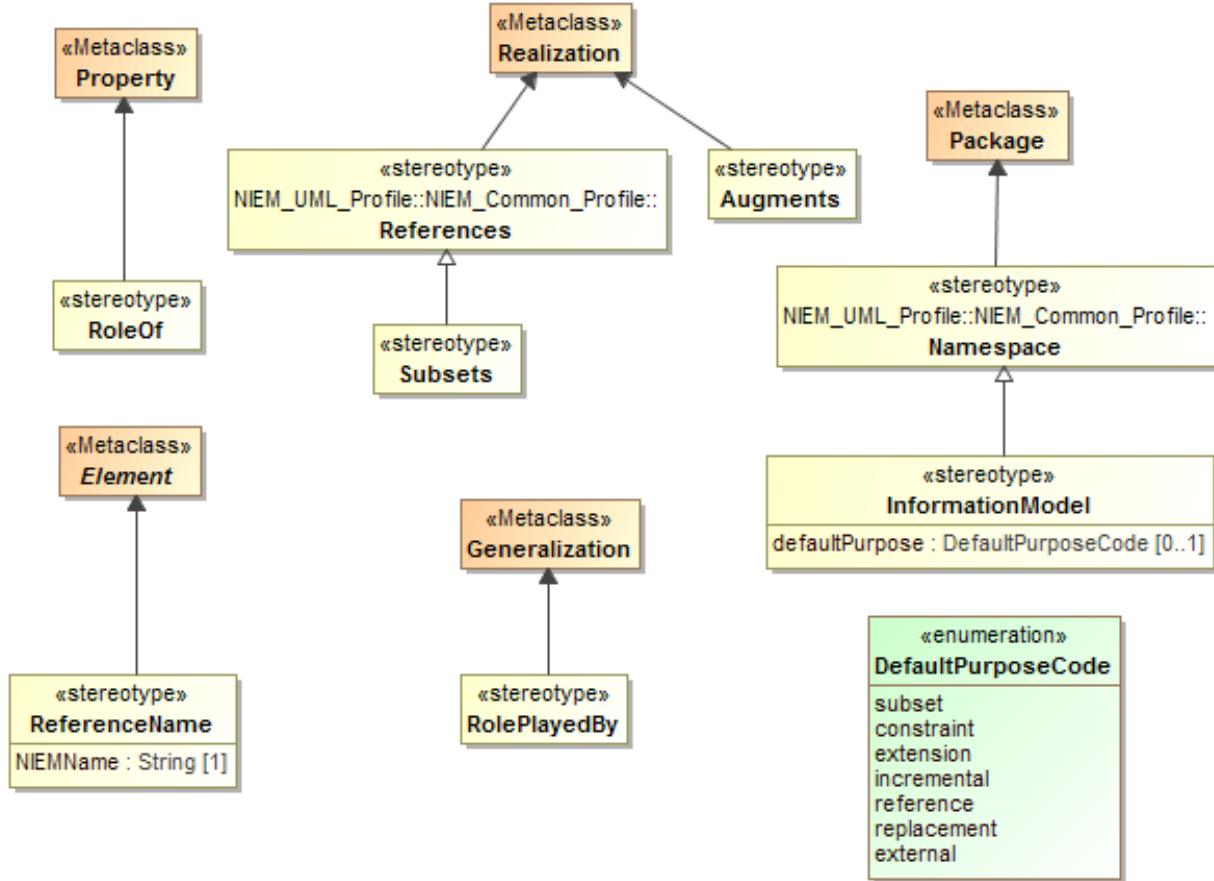


Figure 8-3 NIEM PIM Profile

### 8.3.2 <Stereotype> Augments

#### Description

An Augments Realization specifies that the client Class is augmented by the supplier Class, which may or may not be an «AugmentationType». The «Augments» Realization corresponds to an XSD Element and is the equivalent of an explicitly specified augmentation element. If there exists an explicitly specified augmentation element, it may be used to override characteristics of the «Augments» Realization. The «Augments» Realization may optionally be used to specify implementation detail as follows:

- The name. NDR rules and conventions are used to produce a default name. If the target type is an «AugmentationType», then the name will be the same as the «AugmentationType», less the "Type"

suffix. In general, the name will be the same as the name of the Type, less the "Type" suffix. If the default name is not the desired name of the element, the UML name of the «Augments» Realization may be set to override the default.

- Documentation. By default, the ownedComment is used to create documentation associated with the provisioned element. If unspecified, an NDR-compliant documentation will be provisioned for the target XSD Element.
- The namespace of the XSD Element is the «InformationModel» which owns the «Augments» Realization. This enables easy and direct re-use of Reference Model Types by an Extension Model, with minimal modeling overhead.

An explicit declaration of an augmentation element and/or the \*AugmentationPoint element may be used to extend/override the characteristics of an «Augments» Realization. The declaration extends the default behavior of the «Augments» Realization with the additional capabilities:

- The element may be used explicitly in the particle decomposition/redefinition of an \*AugmentationPoint. This enables the same level of particle decomposition for an \*AugmentationPoint which exists for any other element.
- An explicit element enables specification/override of the default 0..\* cardinality.
- An explicit element enables reuse of the element by other types within an extension schema, independent of the augmentation pattern.
- An explicit element is required if the augmentation element declaration is intended to be abstract. An abstract element in the \*AugmentationPoint substitutionGroup hierarchy provides additional flexibility in organizing element constraints.

## Extends

UML::Realization

### 8.3.3 <Stereotype> InformationModel

#### 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.

#### Generalization

[Namespace](#)

#### Properties

##### **defaultPurpose : DefaultPurposeCode [0..1]**

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.

## Constraints

### NDR3 [Rule 10-1] (REF,EXT). Complex type has a category

[Rule 10-1](#), Complex type has a category (REF, EXT): [Section 10.1](#), Categories of NIEM type definitions

[English]

The constraint is satisfied during provisioning, which produce one of the NDR defined complex type categories based on explicit or implicit model specifications.

### NDR3 [Rule 10-2] (REF,EXT). Object type with complex content is derived from object type

[Rule 10-2](#), Object type with complex content is derived from object type (REF, EXT): [Section 10.2.1.1](#), Object types with complex content

[English]

The constraint is satisfied during provisioning, which produce derivation of each Object Type from another Object Type or, if not modeled explicitly, from structures:ObjectType.

### NDR3 [Rule 10-43] (REF,EXT). Schema component name composed of English words

[Rule 10-43](#), Schema component name composed of English words (REF, EXT): [Section 10.8](#), Naming rules

[English]

This rule is not readily computational.

### NDR3 [Rule 10-44] (REF,EXT). Schema component names have only specific characters

[Rule 10-44](#), Schema component names have only specific characters (REF, EXT): [Section 10.8](#), Naming rules

**[OCL] context** InformationModel inv:

```
self.isConformant
implies(
    self.base_Package.ownedType
        ->select(t|not(t.stereotypedBy('PropertyHolder') or
t.stereotypedBy('LocalVocabulary') or (t.oclIsKindOf(Association) and
not(t.oclIsKindOf(Class)))) )
    -
    >forAll(schemaComponent|schemaComponent.niemName().match('[\w\-\]*'))
        and
        self.base_Package.ownedType -
    >select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier).attribute
    -
    >forAll(schemaComponent|schemaComponent.niemName().match('[\w\-\]*'))
)
```

### NDR3 [Rule 10-45] (REF,EXT). Hyphen in component name is a separator

[Rule 10-45](#), Hyphen in component name is a separator (REF, EXT): [Section 10.8](#), Naming rules  
[English]

Rule is definitional.

### **NDR3 [Rule 10-46](REF,EXT). Names use camel case**

[Rule 10-46](#), Names use camel case (REF, EXT): [Section 10.8.1](#), Character case

[English]

Rule is not reliably computational.

### **NDR3 [Rule 10-4] (REF,EXT). Only object type has RoleOf element**

[Rule 10-4](#), Only object type has RoleOf element (REF, EXT): [Section 10.2.2](#), Role types and roles

**[OCL] context** InformationModel inv:

```
self.isConformant
implies
self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)
  ->select(t|t.stereotypedBy('MetadataType') or
t.stereotypedBy('AssociationType') or t.stereotypedBy('AugmentationType') or
t.oclIsKindOf(AssociationClass)).attribute
    ->forAll(a|not(a.stereotypedBy('RoleOf') or
a.niemName().startsWith('RoleOf')))
```

### **NDR3 [Rule 10-50] (REF,EXT). Local term declaration is local to its schema document**

[Rule 10-50](#), Local term declaration is local to its schema document (REF, EXT): [Section 10.8.2.1](#), Use of Acronyms, Initialisms, Abbreviations, and Jargon

[English]

Rule is definitional.

### **NDR3 [Rule 10-51] (REF,EXT). Local terminology interpretation**

[Rule 10-51](#), Local terminology interpretation (REF, EXT): [Section 10.8.2.1](#), Use of Acronyms, Initialisms, Abbreviations, and Jargon

[English]

Rule is definitional.

### **NDR3 [Rule 10-52] (REF,EXT). Singular form is preferred in name**

[Rule 10-52](#), Singular form is preferred in name (REF, EXT): [Section 10.8.3](#), Word forms  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-53] (REF,EXT). Present tense is preferred in name**

[Rule 10-53](#), Present tense is preferred in name (REF, EXT): [Section 10.8.3](#), Word forms  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-54] (REF,EXT). Name does not have nonessential words**

[Rule 10-54](#), Name does not have nonessential words (REF, EXT): [Section 10.8.3](#), Word forms  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-55] (REF,EXT). Component name follows pattern**

[Rule 10-55](#), Component name follows pattern (REF, EXT): [Section 10.8](#), Naming rules  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-56] (REF,EXT). Object-class term identifies concrete category**

[Rule 10-56](#), Object-class term identifies concrete category (REF, EXT): [Section 10.8.4](#), Object-class term  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-57] (REF,EXT). Property term describes characteristic or subpart**

[Rule 10-57](#), Property term describes characteristic or subpart (REF, EXT): [Section 10.8.5](#), Property term  
[English]

Rule is definitional.

#### **NDR3 [Rule 10-58] (REF,EXT). Name may have multiple qualifier terms**

[Rule 10-58](#), Name may have multiple qualifier terms (REF, EXT): [Section 10.8.6](#), Qualifier terms  
[English]

Rule is definitional.

### **NDR3 [Rule 10-59] (REF,EXT). Name has minimum necessary number of qualifier terms**

[Rule 10-59](#), Name has minimum necessary number of qualifier terms (REF, EXT): [Section 10.8.6](#), Qualifier terms  
[English]

Rule is definitional.

### **NDR3 [Rule 10-5] (REF,EXT,INS). RoleOf elements indicate the base types of a role type**

[Rule 10-5](#), RoleOf elements indicate the base types of a role type (REF, EXT, INS): [Section 10.2.2](#), Role types and roles

[English]

This rule is definitional.

### **NDR3 [Rule 10-60] (REF,EXT). Order of qualifies is not significant**

[Rule 10-60](#), Order of qualifies is not significant (REF, EXT): [Section 10.8.6](#), Qualifier terms  
[English]

Rule is definitional.

### **NDR3 [Rule 10-61] (REF,EXT). Redundant term in name is omitted**

[Rule 10-61](#), Redundant term in name is omitted (REF, EXT): [Section 10.8.7](#), Representation terms  
[English]

The constraint can not be expressed easily in OCL.

### **NDR3 [Rule 10-65](REF,EXT). Machine-readable annotations are valid**

[Rule 10-65](#), Machine-readable annotations are valid (REF, EXT): [Section 10.9](#), Machine-readable annotations

[English]

The constraint is realized through provisioning:  
there are no NIEM-UML constructs related specifically to machine-readable annotations;  
the production of machine-readable annotations is based on the mapping of specific NDR rules to target schema annotations.

### **NDR3 [Rule 10-67] (REF,EXT). Deprecated annotates schema component**

[Rule 10-67](#), Deprecated annotates schema component (REF, EXT): [Section 10.9.1.1](#), Deprecation

[English]

The constraint is realized through provisioning:

A NamedElement with applied Stereotype Deprecated will create the appinfo:deprecated on the target schema component.

### **NDR3 [Rule 10-68] (REF,EXT). External import indicator annotates import**

[Rule 10-68](#), External import indicator annotates import (REF, EXT): [Section 10.9.1](#), The NIEM appinfo namespace

[English]

The constraint is realized through provisioning:

A provisioned xs:import will own an appinfo:externalImportIndicator if the import InformationModel has a defaultPurpose of external.

### **NDR3 [Rule 10-6] (INS). Instance of RoleOf element indicates a role object**

[Rule 10-6](#), Instance of RoleOf element indicates a role object (INS): [Section 10.2.2](#), Role types and roles

[English]

This rule is definitional.

### **NDR3 [Rule 10-76] (REF,EXT,INS). Use structures as specified**

[Rule 10-76](#), Use structures as specified (REF, EXT, INS): [Section 10.10](#), NIEM structural facilities

[English]

The constraint is realized through provisioning:

The structures namespace is not part of the NIEM-UML model, all usages of the namespace are provisioned according to the NDR rules governing that namespace.

### **NDR3 [Rule 11-24] (REF,EXT). Schema uses only known attribute groups**

[Rule 11-24](#), Schema uses only known attribute groups (REF, EXT): [Section 11.3.3.1](#), Attribute group use

[English]

Expression of this constraint as OCL has been deferred.

### **NDR3 [Rule 11-25] (REF,EXT). Data definition does not introduce ambiguity**

[Rule 11-25](#), Data definition does not introduce ambiguity (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-26] (REF,EXT). Object class has only one meaning**

[Rule 11-26](#), Object class has only one meaning (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-27] (REF,EXT). Data definition of a part does not redefine the whole**

[Rule 11-27](#), Data definition of a part does not redefine the whole (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-28] (REF,EXT). Do not leak representation into data definition**

[Rule 11-28](#), Do not leak representation into data definition (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-29] (REF,EXT). Data definition follows 11179-4 requirements**

[Rule 11-29](#), Data definition follows 11179-4 requirements (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-30] (REF,EXT). Data definition follows 11179-4 recommendations**

[Rule 11-30](#), Data definition follows 11179-4 recommendations (REF, EXT): [Section 11.6.1](#), Human-readable documentation

[English]

Constraint is non-computable.

### **NDR3 [Rule 11-34] (REF,EXT). Same namespace means same components**

[Rule 11-34](#), Same namespace means same components (REF, EXT): [Section 11.7.1](#), xs : schema document element restrictions

[English]

Constraint expression as OCL is deferred.

### **NDR3 [Rule 11-35] (REF,EXT). Different version means different view**

[Rule 11-35](#), Different version means different view (REF, EXT): [Section 11.7.1](#), xs : schema document element restrictions

[English]

Rule is definitional.

### **NDR3 [Rule 11-37] (SET). Extension schema document imports reference or extension schema**

[Rule 11-37](#), Extension schema document imports reference or extension schema (SET): [Section 11.8](#), Schema assembly

**[OCL] context** InformationModel **inv:**

```
(self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::  
extension)  
implies  
self.base_Package.ownedType-  
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)->forAll(c|  
c.general.namespace.oclAsType(NamedElement)->asSet()  
->union(c.clientDependency-  
>select(d|d.stereotypeBy('Restriction')).supplier-  
>select(s|s.oclIsKindOf(Classifier)).namespace.oclAsType(NamedElement)-  
->asSet())
```

```

->union(c.attribute-
>select(a|not(a.type.oclIsUndefined())).type.namespace.oclAsType(NamedElement
)->asSet())
->union(c.attribute.clientDependency-
>select(d|d.stereotypeBy('References')).supplier-
>select(s|s.oclIsKindOf(Property)).namespace.namespace.oclAsType(NamedElement
)->asSet())
->select(p|p.stereotypeBy('InformationModel'))

.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose->forAll(p|
    (p=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)
    or (p=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)
    or (p=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::extension))
)

```

### **NDR3 [Rule 11-38] (REF,EXT). Structures imported as conformant**

[Rule 11-38](#), Structures imported as conformant (REF, EXT): [Section 11.8.1](#), Supporting namespaces are imported as conformant

[English]

Constraint realized by provisioning;  
there is no explicit modeling of xs:import in NIEM-UML; xs:import is produced as required, according to this and other NDR rules

### **NDR3 [Rule 11-39] (REF,EXT). XML namespace imported as conformant**

[Rule 11-39](#), XML namespace imported as conformant (REF, EXT): [Section 11.8.1](#), Supporting namespaces are imported as conformant

[English]

Constraint realized by provisioning;  
there is no explicit modeling of xs:import in NIEM-UML; xs:import is produced as required, according to this and other NDR rules

### **NDR3 [Rule 11-40] (SET). Each namespace may have only a single root schema in a schema set**

[Rule 11-40](#), Each namespace may have only a single root schema in a schema set (SET): [Section 11.8](#), Schema assembly

[English]

Expressing Constraint in OCL has been deferred

### **NDR3 [Rule 11-41] (REF,EXT). Consistently marked namespace imports**

[Rule 11-41](#), Consistently marked namespace imports (REF, EXT): [Section 11.8](#), Schema assembly  
[English]

Constraint ensured by provisioning:

xs:import is not in the NIEM-UML Model, it is created during provisioning, which consistently constructs the externalImportIndicator based on the tag values in the referenced InformationModel.

### **NDR3 [Rule 12-10] (INS). Values of structures:metadata refer to values of structures:id**

[Rule 12-10](#), Values of structures:metadata refer to values of structures:id (INS): [Section 12.3](#), Instance metadata

[English]

Constraint is realized during provisioning of XML instance documents.

### **NDR3 [Rule 12-11] (INS). Value of structures:relationshipMetadata refers to value of structures:id**

[Rule 12-11](#), Value of structures:relationshipMetadata refers to value of structures:id (INS): [Section 12.3](#), Instance metadata

[English]

Constraint realized during provisioning of XML instance documents.

### **NDR3 [Rule 12-12] (INS). structures:metadata and structures:relationshipMetadata refer to metadata elements**

[Rule 12-12](#), structures:metadata and structures:relationshipMetadata refer to metadata elements (INS): [Section 12.3](#), Instance metadata

[English]

Constraint realized by provisioning of the XML Instance Documents.

### **NDR3 [Rule 12-13] (INS). Attribute structures:metadata references metadata element**

[Rule 12-13](#), Attribute structures:metadata references metadata element (INS): [Section 12.3](#), Instance metadata

[English]

Constraint realized during provisioning of XML instance documents.

### **NDR3 [Rule 12-14] (INS). Attribute structures:relationshipMetadata references metadata element**

[Rule 12-14](#), Attribute structures:relationshipMetadata references metadata element (INS): [Section 12.3](#), Instance metadata

[English]

Constraint realized by provisioning of XML Instance Documents.

### **NDR3 [Rule 12-15] (INS). Metadata is applicable to element**

Rule 12-15, Metadata is applicable to element (INS): Section 12.3, Instance metadata

[English]

Constraint realized when provisioning XML Instance Document.

### **NDR3 [Rule 12-1] (INS). Instance must be schema-valid**

[Rule 12-1](#), Instance must be schema-valid (INS): [Section 12](#), XML instance document rules

[English]

Constraint can not be easily expressed in OCL, the constraint must be enforced by an XML Schema Document Validation tool.

### **NDR3 [Rule 12-2] (INS). Element with structures:ref does not have content**

[Rule 12-2](#), Element with structures:ref does not have content (INS): [Section 12.2](#), Reference elements

[English]

Constraint is realized during provisioning of instance documents, if any. Provisioning of an element with @structures:ref attribute will not have element content.

### **NDR3 [Rule 12-3] (INS). Attribute structures:ref must reference structures:id**

[Rule 12-3](#), Attribute structures:ref must reference structures:id (INS): [Section 12.2](#), Reference elements

[English]

Constraint is realized during provisioning of instance documents, if any. Any @structures:ref will reference an element with the same value in an @structures:id.

#### **NDR3 [Rule 12-4] (INS). Linked elements have same validation root**

[Rule 12-4](#), Linked elements have same validation root (INS): [Section 12.2](#), Reference elements

[English]

Constraint is realized during provisioning of instance documents, if any.

#### **NDR3 [Rule 12-5] (INS). Attribute structures:ref references element of correct type**

[Rule 12-5](#), Attribute structures:ref references element of correct type (INS): [Section 12.2](#), Reference elements

[English]

Constraint is realized during provisioning of instance documents, if any.

#### **NDR3 [Rule 12-6] (INS). Reference and content elements have the same meaning**

[Rule 12-6](#), Reference and content elements have the same meaning (INS): [Section 12.2.1](#), Reference and content elements have same meaning

[English]

Rule is definitional.

#### **NDR3 [Rule 12-7] (INS). Empty content has no meaning**

[Rule 12-7](#), Empty content has no meaning (INS): [Section 12](#), XML instance document rules

[English]

Rule is definitional.

#### **NDR3 [Rule 12-8] (INS). Metadata applies to referring entity**

[Rule 12-8](#), Metadata applies to referring entity (INS): [Section 12.3](#), Instance metadata

[English]

Rule is definitional.

### **NDR3 [Rule 12-9] (INS). Referent of structures:relationshipMetadata annotates relationship**

[Rule 12-9](#), Referent of structures:relationshipMetadata annotates relationship (INS): [Section 12.3](#), Instance metadata

[English]

Rule is definitional.

### **NDR3 [Rule 4-1] (SET) Schema marked as reference schema document must conform**

[Rule 4-1](#), Schema marked as reference schema document must conform (SET): [Section 4.1](#), Conformance targets defined

[English]

This constraint realized by the aggregate of constraints targeting REF schema documents.

### **NDR3 [Rule 4-2] (SET) Schema marked as extension schema document must conform**

[Rule 4-2](#), Schema marked as extension schema document must conform (SET): [Section 4.1](#), Conformance targets defined

[English]

This constraint realized by the aggregate of constraints targeting EXT schema documents.

### **NDR3 [Rule 4-3] (REF,EXT) Schema is CTAS-conformant**

[Rule 4-3](#), Schema is CTAS-conformant (REF, EXT): [Section 4.3](#), Conformance target identifiers

[English]

This constraint realized by the aggregate of constraints targeting REF and EXT schema documents.

### **NDR3 [Rule 4-4] (REF,EXT). Document element has attribute ct:conformanceTargets**

[Rule 4-4](#), Document element has attribute ct:conformanceTargets (REF, EXT): [Section 4.3](#), Conformance target identifiers

[English]

This constraint realized during provisioning of the schema associated with the InformationModel

### **NDR3 [Rule 4-5] (REF). Schema claims reference schema conformance target**

[Rule 4-5](#), Schema claims reference schema conformance target (REF): [Section 4.3](#), Conformance target identifiers

[English]

This constraint realized during provisioning of the schema associated with the InformationModel

### **NDR3 [Rule 4-6] (EXT). Schema claims extension conformance target**

[Rule 4-6](#), Schema claims extension conformance target (EXT): [Section 4.3](#), Conformance target identifiers

[English]

This constraint realized during provisioning of the schema associated with the InformationModel

### **NDR3 [Rule 7-1] (REF,EXT,INS). Document is an XML document**

[Rule 7-1](#), Document is an XML document (REF, EXT, INS): [Section 7.1](#), Conformance to XML

[English]

This constraint realized during provisioning of the schema associated with the InformationModel

### **NDR3 [Rule 9-11] (REF). No simple type disallowed derivation**

[Rule 9-11](#), No simple type disallowed derivation (REF): [Section 9.1.2](#), Simple type definition

#### **[OCL] context InformationModel inv:**

```
((self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference) or (self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset))  
implies  
self.base_Package.ownedType-
```

```
>select(t|t.oclIsKindOf(DataType)).oclAsType(DataType) -  
>forAll(dt|not(dt.isFinalSpecialization))
```

### **NDR3 [Rule 9-30] (REF,EXT). Base type of complex type with complex content must have complex content**

[Rule 9-30](#), Base type of complex type with complex content must have complex content (REF, EXT): [Section 9.1.3.2.1](#), Base type of complex type with complex content has complex content

[English]

Provisioning to target schemas ensures the base type of Complex types with complex content will have complex content.

### **NDR3 [Rule 9-31] (SET). Base type of complex type with complex content must have complex content**

[Rule 9-31](#), Base type of complex type with complex content must have complex content (SET): [Section 9.1.3.2.1](#), Base type of complex type with complex content has complex content

[English]

Provisioning to target schemas ensures the base type of Complex types with complex content will have complex content.

### **NDR3 [Rule 9-33] (REF). No complex type disallowed substitutions**

[Rule 9-33](#), No complex type disallowed substitutions (REF): [Section 9.1.3](#), Complex type definition

[English]

The concept of disallowed substitutions is currently not supported by the NIEM UML Profile. Currently, there will be no "block" provisioned for a complex type.

### **NDR3 [Rule 9-34] (REF). No complex type disallowed derivation**

[Rule 9-34](#), No complex type disallowed derivation (REF): [Section 9.1.3](#), Complex type definition

#### **[OCL] context InformationModel inv:**

```
((self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode:  
:reference) or (self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::Default  
PurposeCode::subset))  
implies
```

```
self.base_Package.ownedType-
>select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)-
>forAll(dt|not(dt.isFinalSpecialization))
```

### NDR3 [Rule 9-35] (REF,EXT). Element declaration is top-level

[Rule 9-35](#), Element declaration is top-level (REF, EXT): [Section 9.2.1](#), Element declaration  
[English]

Constraint is enforced during provisioning, top level elements are always created and referenced by non top level elements.

### NDR3 [Rule 9-39] (REF,EXT). Element type not in the XML Schema namespace

[Rule 9-39](#), Element type not in the XML Schema namespace (REF, EXT): [Section 9.2.1](#), Element declaration

[OCL] context InformationModel inv:

```
self.targetNamespace<>'http://www.w3.org/2001/XMLSchema'
```

### NDR3 [Rule 9-41] (REF,EXT). Element type is not simple type

[Rule 9-41](#), Element type is not simple type (REF, EXT): [Section 9.2.1](#), Element declaration

[English]

OCL representation of this constraint is deferred.

### NDR3 [Rule 9-42] (REF). No element disallowed substitutions

[Rule 9-42](#), No element disallowed substitutions (REF): [Section 9.2.1](#), Element declaration

[English]

The concept of disallowed substitutions (@block) is currently not supported by NIEM-UML. There will be no provisioning of the @block attribute.

### NDR3 [Rule 9-47] (REF,EXT). Attribute declaration is top-level

[Rule 9-47](#), Attribute declaration is top-level (REF, EXT): [Section 9.2.3](#), Attribute declaration  
[English]

Constraint is enforced during provisioning, top level attributes are always created and referenced by non top level elements.

### **NDR3 [Rule 9-49] (REF,EXT). Attribute declaration has type**

[Rule 9-49](#), Attribute declaration has type (REF, EXT): [Section 9.2.3](#), Attribute declaration

[English]

Specification as OCL Constraint is deferred.

### **NDR3 [Rule 9-61] (REF). xs:sequence must be child of xs:extension**

[Rule 9-61](#), xs : sequence must be child of xs : extension (REF): [Section 9.3.1.1](#), Sequence

[English]

Constraint enforced by provisioning, an xs:sequence is always produced as a child of an xs:extension.

### **NDR3 [Rule 9-62] (EXT). xs:sequence must be child of xs:extension or xs:restriction**

[Rule 9-62](#), xs : sequence must be child of xs : extension or xs : restriction (EXT): [Section 9.3.1.1](#), Sequence

[English]

Constraint enforced by provisioning, an xs:sequence is always produced as a child of an xs:extension or xs:restriction.

### **NDR3 [Rule 9-63] (REF). No xs:choice**

[Rule 9-63](#), No xs : choice (REF): [Section 9.3.1.2](#), Choice

**[OCL] context** InformationModel inv:

```
((self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)or(self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference))  
implies  
self.base_Package.ownedType  
->select(t|t.oclIsKindOf(Classifier)).oclAsType(Classifier)  
->union(self.base_Package.ownedType-  
>select(t|t.oclIsKindOf(Class)).oclAsType(Class).nestedClassifier)  
->select(t|t.stereotypeBy('Choice'))->isEmpty()
```

### **NDR3 [Rule 9-64] (EXT). xs:choice must be child of xs:sequence**

[Rule 9-64](#), xs : choice must be child of xs : sequence (EXT): [Section 9.3.1.2](#), Choice

[English]

Constraint enforced by provisioning, an xs:choice is always produced as a child of an xs:sequence.

### **NDR3 [Rule 9-65] (REF,EXT). Sequence has minimum cardinality 1**

[Rule 9-65](#), Sequence has minimum cardinality 1 (REF, EXT): [Section 9.3.2.1](#), Sequence cardinality

[English]

Constraint enforced by provisioning, an xs:sequence is always produced with @minOccurs=1.

### **NDR3 [Rule 9-66] (REF,EXT). Sequence has maximum cardinality 1**

[Rule 9-66](#), Sequence has maximum cardinality 1 (REF, EXT): [Section 9.3.2.1](#), Sequence cardinality

[English]

Constraint enforced by provisioning, an xs:sequence is always produced with @maxOccurs=1.

### **NDR3 [Rule 9-67] (EXT). Choice has minimum cardinality 1**

[Rule 9-67](#), Choice has minimum cardinality 1 (EXT): [Section 9.3.2.2](#), Choice cardinality

[English]

Constraint enforced by provisioning, an xs:choice is always produced with @minOccurs=1.

### **NDR3 [Rule 9-68] (EXT). Choice has maximum cardinality 1**

[Rule 9-68](#), Choice has maximum cardinality 1 (EXT): [Section 9.3.2.2](#), Choice cardinality

[English]

Constraint enforced by provisioning, an xs:choice is always produced with @maxOccurs=1.

### **NDR3 [Rule 9-69] (REF). No use of xs:any**

[Rule 9-69](#), No use of xs : any (REF): [Section 9.3.4](#), Wildcard

**[OCL] context** InformationModel **inv:**

```
((self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference) or (self.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset))
```

```
implies
self.base_Package.ownedType-
>forAll(dt|not(dt.stereotypeBy('XSDAnyProperty')))
```

### NDR3 [Rule 9-70] (REF). No use of xs:anyAttribute

[Rule 9-70](#), No use of xs:anyAttribute (REF): [Section 9.3.4](#), Wildcard

[English]

Specification of this constraint as OCL has been deferred.

### NDR3 [Rule 9-71] (REF,EXT). No use of xs:unique

[Rule 9-71](#), No use of xs:unique (REF, EXT): [Section 9.4](#), Identity-constraint definition components

[English]

Constraint enforced by provisioning, an xs:unique can not be modeled nor is it produced in a target schema.

### NDR3 [Rule 9-72] (REF,EXT). No use of xs:key

[Rule 9-72](#), No use of xs:key (REF, EXT): [Section 9.4](#), Identity-constraint definition components

[English]

Constraint enforced by provisioning, an xs:key can not be modeled nor is it produced in a target schema.

### NDR3 [Rule 9-73] (REF,EXT). No use of xs:keyref

[Rule 9-73](#), No use of xs:keyref (REF, EXT): [Section 9.4](#), Identity-constraint definition components

[English]

Constraint enforced by provisioning, an xs:keyref can not be modeled nor is it produced in a target schema.

### NDR3 [Rule 9-74] (REF,EXT). No use of xs:group

[Rule 9-74](#), No use of xs:group (REF, EXT): [Section 9.5.1](#), Model group definition

[English]

Constraint enforced by provisioning, an xs:group can not be modeled nor is it produced in a target schema.

### **NDR3 [Rule 9-75] (REF,EXT). No definition of attribute groups**

[Rule 9-75](#), No definition of attribute groups (REF, EXT): [Section 9.5.2](#), Attribute group definition

[English]

Constraint enforced by provisioning, an xs:attributeGroup can not be modeled nor is it produced in a target schema.

### **NDR3 [Rule 9-76] (REF,EXT). Comment is not recommended**

[Rule 9-76](#), Comment is not recommended (REF, EXT): [Section 9.6](#), Annotation components

[English]

Constraint enforced by provisioning, an XML comment can not be modeled nor is it produced in a target schema.

### **NDR3 [Rule 9-77] (REF,EXT). Documentation element has no element children**

[Rule 9-77](#), Documentation element has no element children (REF, EXT): [Section 9.6](#), Annotation components

[English]

Constraint enforced by provisioning, the xs:documentation is populated by a UML Comment body, which is a String (possibly escaped to ensure no nested xml elements are present).

### **NDR3 [Rule 9-78] (REF,EXT). xs:appinfo children are comments, elements, or whitespace**

[Rule 9-78](#), xs:appinfo children are comments, elements, or whitespace (REF, EXT): [Section 9.6.1](#), Application information annotation

[English]

Constraint enforced by provisioning; the xs:appinfo is not directly modeled, and is provisioned in accordance with NDR-specified rules associated with specific NIEM concepts. Thus, an XML element is the child of an xs:appinfo.

### **NDR3 [Rule 9-79] (REF,EXT). Appinfo child elements have namespaces**

[Rule 9-79](#), Appinfo child elements have namespaces (REF, EXT): [Section 9.6.1](#), Application information annotation

[English]

Constraint enforced by provisioning; the xs:appinfo is not directly modeled, and is provisioned in accordance with NDR-specified rules associated with specific NIEM concepts. Thus, an XML element is the child of an xs:appinfo and will have a namespace name.

### **NDR3 [Rule 9-80] (REF,EXT). Appinfo descendants are not XML Schema elements**

[Rule 9-80](#), Appinfo descendants are not XML Schema elements (REF, EXT): [Section 9.6.1](#), Application information annotation

[English]

Constraint enforced by provisioning; the xs:appinfo is not directly modeled, and is provisioned in accordance with NDR-specified rules associated with specific NIEM concepts. Thus, an XML element is the child of an xs:appinfo and will not contain elements with the schema namespace.

### **NDR3 [Rule 9-81] (REF,EXT). Schema has data definition**

[Rule 9-81](#), Schema has data definition (REF, EXT): [Section 9.7](#), Schema as a whole

**[OCL] context** InformationModel inv:

```
self.isConformant
implies
  self.base_Package.ownedComment._'body'-
>exists(doc|not(doc.oclIsUndefined())and(doc<>''))
```

### **NDR3 [Rule 9-85] (REF). No disallowed substitutions**

[Rule 9-85](#), No disallowed substitutions (REF): [Section 9.7](#), Schema as a whole

[English]

The concept of disallowed substitutions (@blockDefault) is currently not supported by NIEM-UML. There will be no provisioning of the @blockDefault attribute.

### **NDR3 [Rule 9-86] (REF). No disallowed derivations**

[Rule 9-86](#), No disallowed derivations (REF): [Section 9.7](#), Schema as a whole

[English]

The concept of disallowed derivations is currently not in the NIEM-UML model; the attribute @finalDefault will not be produced for any InformationModel schema.

### **NDR3 [Rule 9-87] (REF,EXT). No use of xs:redefine**

[Rule 9-87](#), No use of xs:redefine (REF, EXT): [Section 9.8](#), Schema assembly

[English]

The concept of xs:redefine is not in the NIEM-UML model; the schema construct xs:redefine can not be modeled and will not be produced for any InformationModel schema.

### **NDR3 [Rule 9-88] (REF,EXT). No use of xs:include**

[Rule 9-88](#), No use of xs:include (REF, EXT): [Section 9.8](#), Schema assembly

[English]

The concept of xs:include is not in the NIEM-UML model; the schema construct xs:include can not be modeled and will not be produced for any InformationModel schema.

## **8.3.4 <Stereotype> ReferenceName**

### **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.

### **Extends**

UML::Element

### **Properties**

#### **NIEMName : String [1]**

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

## **8.3.5 <Stereotype> RoleOf**

### **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 NIEM role type is a type that represents a particular function, purpose, usage, or role of an object.

## Extends

UML::Property

## Constraints

### NDR3 [Rule 10-3] (REF,EXT). RoleOf element type is an object type

[Rule 10-3](#), RoleOf element type is an object type (REF, EXT): [Section 10.2.2](#), Role types and roles

[OCL] context RoleOf inv:

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    not(self.namespace.namespace.oclIsUndefined())  
    and  
    self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and  
    not(self.name.oclIsUndefined())  
    and  
    (self.niemName().startsWith('RoleOf') or  
    self.stereotypeBy('RolePlayedBy'))  
)  
implies(  
    not(self.type.oclIsUndefined())  
    and  
    self.type->exists(t|not(t.stereotypeBy('MetadataType') or  
    t.stereotypeBy('AssociationType') or t.stereotypeBy('AugmentationType') or  
    t.oclIsKindOf(AssociationClass)))  
)
```

## 8.3.6 <Stereotype> RolePlayedBy

### 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.

## Extends

UML::Generalization

## Constraints

### NDR3 [Rule 10-3] (REF,EXT). RoleOf element type is an object type

[Rule 10-3](#), RoleOf element type is an object type (REF, EXT): [Section 10.2.2](#), Role types and roles

**[OCL] context RolePlayedBy inv:**

```
self.base_Generalization.general->
  forAll(t|not(t.stereotypedBy('MetadataType') or
t.stereotypedBy('AssociationType') or t.stereotypedBy('AugmentationType') or
t.oclisKindOf(AssociationClass) ))
```

### 8.3.7 <Stereotype> Subsets

#### 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

#### Generalization

[References](#)

#### Constraints

##### Subsets

The client and supplier of a «Subsets» Realization must have the same name.

If the supplier is a «Namespace» Package, then the client must be a «Namespace» Package with the same target namespace.

If the supplier is a Classifier, then the client must be a Classifier owned by a «Namespace» Package with the same target namespace as the supplier's owning «Namespace» Package.

If the supplier is a Property, then the client must be a Property contained by a «Namespace» Package with the same target namespace as the supplier's containing «Namespace» Package.

**[OCL] context Subsets inv:**

```
self.base_Realization->forAll(r|
  r.client->forAll(c|
    r.supplier->forAll(s|
      ( s.name=c.name)
      and (s.oclisKindOf(Package) implies
(c.oclisKindOf(Package) and c.stereotypedBy('Namespace') and
s.stereotypedBy('Namespace') and
(s.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Pro-
file::Namespace).targetNamespace=c.appliedStereotype('Namespace').oclAsType(NIEM_UML_Pro-
file::NIEM_Common_Profile::Namespace).targetNamespace))
      and (s.oclisKindOf(Classifier) implies
(c.oclisKindOf(Classifier) and c.namespace.stereotypedBy('Namespace') and
s.namespace.stereotypedBy('Namespace') and
(s.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Co-
mmon_Profile::Namespace).targetNamespace=c.namespace.appliedStereotype('Name-
space').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).targetNa-
mespace))))
```

```

        and (s.oclIsKindOf(Property) implies
(c.oclIsKindOf(Property) and c.namespace.namespace.stereotypedBy('Namespace')
and s.namespace.namespace.stereotypedBy('Namespace') and
(s.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Prof
ile::NIEM_Common_Profile::Namespace).targetNamespace=c.namespace.namespace.ap
pliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile:
	Namespace).targetNamespace)))
)
)
)
)

```

## 8.3.8 <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.

### Literals

#### **subset**

A NIEM *schema document subset* is a set of XML schema documents that constitutes a reduced set of components derived from a NIEM reference schema document or document set associated with a given numbered release or domain update. See [NIEM-MPD] [Section 4.2.1, Basic Subset Concepts](#).

#### **constraint**

See [NIEM-MPD] [Section 4.5, Constraint Schema Document Sets](#).

#### **extension**

See [NIEM-MPD] [Section 4.3, Extension Schema Documents](#).

#### **incremental**

#### **reference**

See [NIEM-MPD] [Section 2.8, Reference Schema Documents](#).

#### **replacement**

#### **external**

See [NIEM-MPD] [Section 4.4, External Schema Documents](#).

## 8.4 Profile : 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 PSM Profile imports the NIEM Common Profile and, therefore, includes all the stereotypes and metaclasses covered by that profile.

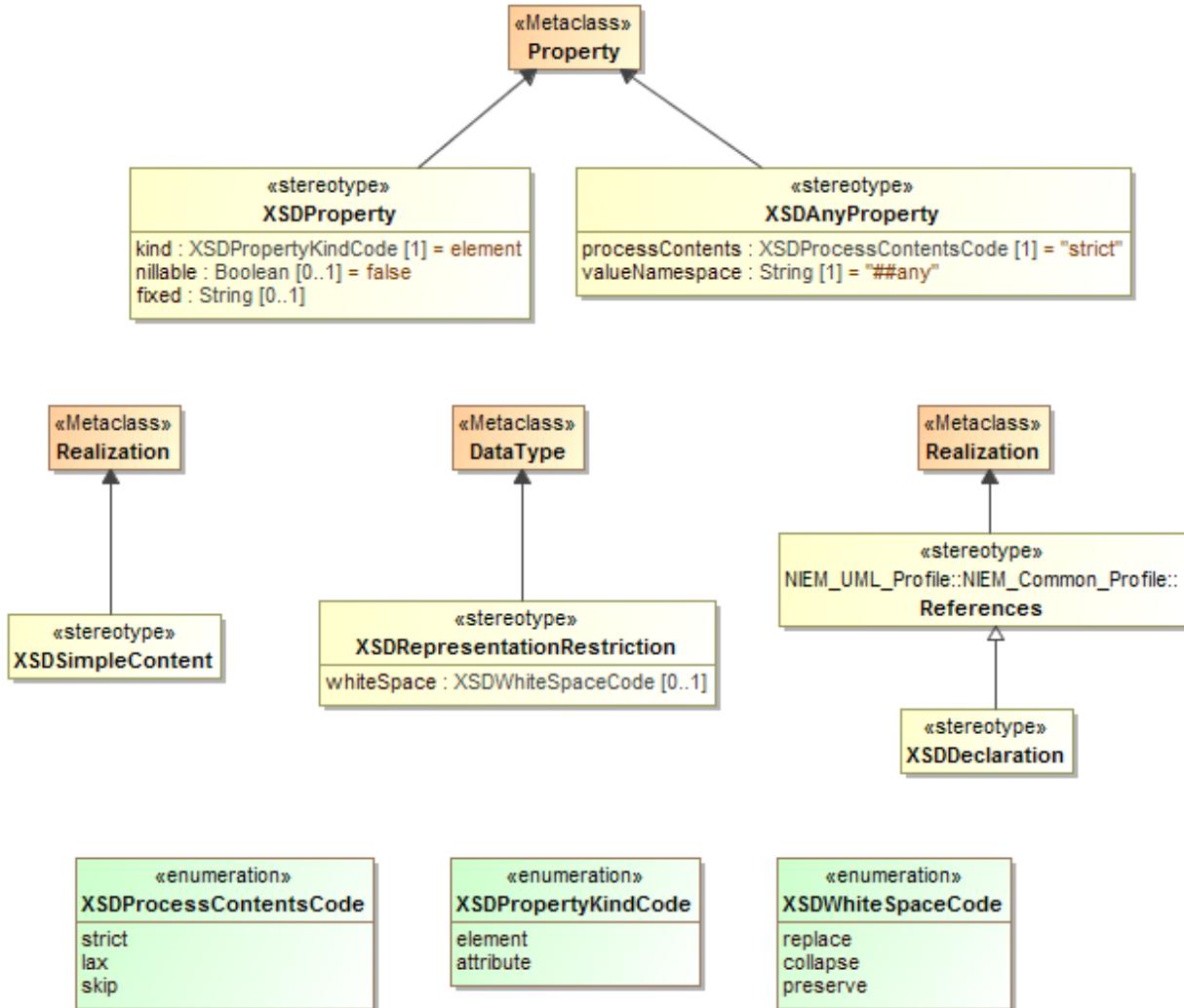


Figure 8-4 NIEM PSM Profile

### 8.4.2 <Stereotype> XSDAnyProperty

#### Description

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

## Extends

UML::Property

## Properties

### processContents : XSDProcessContentsCode [1]

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

### valueNamespace : String [1]

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

## Constraints

### XSDAnyProperty

An XSDAnyProperty must have an empty type and must not be a derived union or subset any other property.

[OCL] context XSDAnyProperty inv:

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

## 8.4.3 <Stereotype> XSDDeclaration

### 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 Namespace 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.

### Generalization

#### References

## 8.4.4 <Stereotype> XSDProperty

### 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.

## Extends

UML::Property

## Properties

### fixed : String [0..1]

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

### kind : XSDPropertyKindCode [1]

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 : Boolean [0..1]

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

## Constraints

### NDR3 [Rule 10-10] (REF,EXT). Element use from external adapter type defined by external schema documents

[Rule 10-10](#), Element use from external adapter type defined by external schema documents (REF, EXT): [Section 10.2.3.2](#), External adapter types

[OCL] context XSDProperty inv:

```
(  
    self.namespace.stereotypedBy('AdapterType')  
    and  
        (not(self.stereotypedBy('XSDProperty'))  
         or  
            (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))  
    )  
  
implies  
    self.clientDependency->select(r|r.stereotypedBy('References')).supplier  
    ->forAll(s|  
        s.namespace.namespace.stereotypedBy('Namespace')  
        and  
        not(s.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant)  
    )
```

### **NDR3 [Rule 10-13] (REF). External attribute use only in external adapter type**

[Rule 10-13](#), External attribute use only in external adapter type (REF): [Section 10.2.3.3](#), External attribute use

**[OCL] context XSDProperty inv:**

```
(  
  
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute  
)  
    and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
    and  
self.base_Property.namespace.namespace.stereotypedBy('InformationModel')  
    and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
  
    and  
  
self.base_Property.namespace.namespace.appliedStereotype('InformationModel').  
oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel)  
-  
>forAll(im| (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset) or (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference))  
  
    and  
    not(self.base_Property.namespace.stereotypedBy('AdapterType'))  
)  
implies  
self.base_Property.clientDependency-  
>select(d|d.stereotypedBy('References')).supplier.namespace.namespace.applied  
Stereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::  
InformationModel)  
-  
>forAll(typeInformationModel|typeInformationModel.defaultPurpose<>NIEM_UML_Pr  
ofile::NIEM_PIM_Profile::DefaultPurposeCode::external)
```

### **NDR3 [Rule 10-14] (REF,EXT). External attribute use has data definition**

[Rule 10-14](#), External attribute use has data definition (REF, EXT): [Section 10.2.3.3](#), External attribute use

**[OCL] context XSDProperty inv:**

```
(  
  
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute  
)  
    and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
    and self.base_Property.namespace.namespace.stereotypedBy('Namespace')  
    and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
```

```

        and
        self.base_Property.clientDependency-
>select(d|d.stereotypedBy('References')).supplier.namespace.namespace
        -
>forAll(typeInformationModel|typeInformationModel.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::external)
)
)
implies
self.base_Property.ownedComment._'body'->exists(b|not(b.oclIsUndefined()) and b<>'')

```

### NDR3 [Rule 10-15] (SET). External attribute use not an ID

[Rule 10-15](#), External attribute use not an ID (SET): [Section 10.2.3.3](#), External attribute use

[OCL] context XSDProperty inv:

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute)
    and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
    and self.base_Property.namespace.namespace.stereotypedBy('Namespace')
    and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
    and not(self.base_Property.type.oclIsUndefined())
    and not(self.base_Property.type.name.oclIsUndefined())
    and not(self.base_Property.type.namespace.oclIsUndefined())
    and
not(self.base_Property.type.namespace.oclAsType(NamedElement).name.oclIsUndefined())
)
implies
not(
    (self.base_Property.type.name='ID')
    and
    (self.base_Property.type.namespace.oclAsType(NamedElement).name='XMLPrimitiveTypes')
)

```

### NDR3 [Rule 10-16] (REF,EXT). External element use has data definition

[Rule 10-16](#), External element use has data definition (REF, EXT): [Section 10.2.3.4](#), External element use

[OCL] context XSDProperty inv:

```

(
    (not(self.stereotypedBy('XSDProperty'))           or
     (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))
        and
        not(self.namespace.oclIsUndefined())
        and
        not(self.namespace.namespace.oclIsUndefined())
        and
        self.namespace.namespace.stereotypedBy('InformationModel')
        and
        self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
        and
        self.clientDependency->select(d|d.stereotypedBy('References'))-
    >exists(d|d.supplier.namespace.namespace
            ->exists(typeInformationModel|
                typeInformationModel.stereotypedBy('InformationModel')

                and(typeInformationModel.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::external)
                    )
            )
    implies(
        self.ownedComment._'body'->exists(b|not(b.oclIsUndefined()) and b<>'')
    )
)

```

### NDR3 [Rule 10-20] (REF,EXT). Association element type is an association type

[Rule 10-20](#), Association element type is an association type (REF, EXT): [Section 10.3.2](#), Association element declarations

#### [OCL] context XSDProperty inv:

```

(
    (
        not(self.namespace.oclIsUndefined())
        and not(self.namespace.namespace.oclIsUndefined())
        and self.namespace.namespace.stereotypedBy('InformationModel')
        and
        self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
        and
        (
            not(self.stereotypedBy('XSDProperty'))
            or
            (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
        )
    )
)

```

```

and not(self.name.oclIsUndefined())
and self.niemName().endsWith('Association')
and not(self.type.oclIsUndefined())
)
implies
    (self.type.stereotypedBy('AssociationType') or
self.type.oclIsKindOf(AssociationClass))
)
and
(
(
not(self.namespace.oclIsUndefined())
and not(self.namespace.namespace.oclIsUndefined())
and self.namespace.namespace.stereotypedBy('Namespace')
and
(
    not(self.stereotypedBy('XSDProperty'))
    or

(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
)
and not(self.type.oclIsUndefined())
and (self.type.stereotypedBy('AssociationType') or
self.type.oclIsKindOf(AssociationClass))
and not(self.niemName().oclIsUndefined())
and (self.niemName()<>' ')
)
implies
    self.niemName().endsWith('Association')
)

```

### **NDR3 [Rule 10-34] (REF,EXT). Augmentation element type is an augmentation type**

[Rule 10-34](#), Augmentation element type is an augmentation type (REF, EXT): [Section 10.4.5](#), Augmentation element declarations

#### **[OCL] context XSDProperty inv:**

```

(
(
not(self.namespace.oclIsUndefined())
and self.isNavigable()
and not(self.namespace.namespace.oclIsUndefined())
and self.namespace.namespace.stereotypedBy('InformationModel')
and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
    and (not(self.stereotypedBy('XSDProperty')))      or
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
        and not(self.name.oclIsUndefined())
        and self.niemName().endsWith('Augmentation')
        and not(self.type.oclIsUndefined())

```

```

)
implies
    self.type.stereotypedBy('AugmentationType')
)
and
(
(
    not(self.namespace.oclIsUndefined())
    and self.isNavigable()
    and not(self.namespace.namespace.oclIsUndefined())
    and self.namespace.namespace.stereotypedBy('InformationModel')
    and
        (not(self.stereotypedBy('XSDProperty'))      or
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))
    and not(self.type.oclIsUndefined())
    and self.type.stereotypedBy('AugmentationType')
)
implies
    self.niemName().endsWith('Augmentation')
)

```

### NDR3 [Rule 10-35] (REF,SET). Augmentation elements are not used directly

[Rule 10-35](#), Augmentation elements are not used directly (REF, SET): [Section 10.4.5](#), Augmentation element declarations

[OCL] context XSDProperty inv:

```

(
    (not(self.stereotypedBy('XSDProperty'))      or
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))
    and
        (not(self.name.oclIsUndefined()) and
self.niemName().endsWith('Augmentation'))
    and
        not(self.namespace.oclIsUndefined())
    and
        not(self.namespace.namespace.oclIsUndefined())
    and
        self.namespace.namespace.stereotypedBy('InformationModel')
    and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies

(
    self.namespace->forAll(t|t.stereotypedBy('PropertyHolder'))
)
```

### NDR3 [Rule 10-39] (REF,EXT). Metadata element declaration type is a metadata type

[Rule 10-39](#), Metadata element declaration type is a metadata type (REF, EXT): [Section 10.5.2](#), Metadata element declarations

[OCL] context XSDProperty inv:

```
(  
  ( not(self.type.oclIsUndefined())  
    and self.isNavigable()  
    and (not(self.stereotype('XSDProperty'))  
         or  
         (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
          )  
         and not(self.name.oclIsUndefined())  
         and self.niemName().endsWith('Metadata')  
         and not(self.namespace.oclIsUndefined())  
         and not(self.namespace.namespace.oclIsUndefined())  
         and self.namespace.namespace.stereotype('InformationModel')  
         and  
         self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
      )  
      implies  
        self.type.stereotype('MetadataType')  
    )  
and  
(  
  ( self.isNavigable()  
    and (not(self.stereotype('XSDProperty'))  
         or  
         (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
          )  
         and not(self.type.oclIsUndefined())  
         and not(self.name.oclIsUndefined())  
         and self.type.stereotype('MetadataType')  
         and not(self.namespace.oclIsUndefined())  
         and not(self.namespace.namespace.oclIsUndefined())  
         and self.namespace.namespace.stereotype('InformationModel')  
      )  
      implies  
        self.niemName().endsWith('Metadata')  
    )
```

### NDR3 [Rule 10-40] (REF,EXT,SET). Metadata element has applicable elements

[Rule 10-40](#), Metadata element has applicable elements (REF, EXT, SET): [Section 10.5.2](#), Metadata element declarations

[English]

The rule is definitional.

### NDR3 [Rule 10-47] (REF,EXT). Attribute name begins with lower case letter

[Rule 10-47](#), Attribute name begins with lower case letter (REF, EXT): [Section 10.8.1](#), Character case

**[OCL] context XSDProperty inv:**

```
(  
  (self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute  
)  
    and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
    and self.base_Property.namespace.namespace.stereotypeBy('Namespace')  
    and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
(  
  self.base_Property.niemName().firstToUpper()<>self.base_Property.niemName())
```

### NDR3 [Rule 10-48] (REF,EXT). Name of schema component other than attribute begins with upper case letter

[Rule 10-48](#), Name of schema component other than attribute begins with upper case letter (REF, EXT): [Section 10.8.1](#), Character case

**[OCL] context XSDProperty inv:**

```
(  
  (  
    not(self.stereotypeBy('XSDProperty'))  
    or  
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
    )  
    and self.isNavigable()  
    and self.clientDependency-  
>select(d|d.stereotypeBy('References')).supplier-  
>select(s|s.oclIsKindOf(Property))->isEmpty()  
    and not(self.namespace.oclIsUndefined())  
    and not(self.namespace.stereotypeBy('List'))  
    and not(self.namespace.namespace.oclIsUndefined())  
    and self.namespace.namespace.stereotypeBy('Namespace')  
    and  
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
```

```

        and not(self.name.oclIsUndefined() or (self.niemName()==''))
)
implies
    (self.niemName().firstToUpper()==self.niemName())

```

### **NDR3 [Rule 10-49] (REF,EXT). Names use common abbreviations.**

[Rule 10-49](#), Names use common abbreviations (REF, EXT): [Section 10.8.2](#), Use of acronyms and abbreviations

**[OCL] context XSDProperty inv:**

```

(
    not(self.name.oclIsUndefined() or self.namespace.oclIsUndefined() or
self.namespace.namespace.oclIsUndefined())
    and self.namespace.namespace.stereotype('Namespace')
    and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
not(self.niemName().match('.*Identifier.*') or
self.niemName().match('.*UniformResourceIdentifier.*'))

```

### **NDR3 [Rule 10-62] (REF,EXT). Element with simple content has representation term**

[Rule 10-62](#), Element with simple content has representation term (REF, EXT): [Section 10.8.7](#), Representation terms

**[OCL] context XSDProperty inv:**

```

if(
    not(self.namespace.oclIsUndefined())
    and not(self.namespace.stereotype('List'))
    and not(self.namespace.namespace.oclIsUndefined())
    and self.namespace.namespace.stereotype('Namespace')
    and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
    and not(self.type.oclIsUndefined())
    and not(self.name.oclIsUndefined())
    and self.type.oclIsKindOf(DataType)
    and (
        not(self.stereotype('XSDProperty'))
        or

(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
    )
)
then (
    self.niemName().match('.*Name.*') or
    self.niemName().match('.*Text.*') or
    self.niemName().match('.*List.*') or

```

```

        self.niemName().match('.*Quantity.*') or
        self.niemName().match('.*Percent.*') or
        self.niemName().match('.*Rate.*') or
        self.niemName().match('.*Value.*') or
        self.niemName().match('.*Numeric.*') or
        self.niemName().match('.*Measure.*') or
        self.niemName().match('.*Indicator.*') or
        self.niemName().match('.*URI.*')
        or self.niemName().match('.*ID.*')
        or self.niemName().match('.*Time.*') or
        self.niemName().match('.*Date.*') or
        self.niemName().match('.*Duration.*') or
        self.niemName().match('.*DateTime.*') or
        self.niemName().match('.*Code.*') or
        self.niemName().match('.*Video.*') or
        self.niemName().match('.*Sound.*') or
        self.niemName().match('.*Picture.*') or
        self.niemName().match('.*Graphic.*') or
        self.niemName().match('.*BinaryObject.*') or
        self.niemName().match('.*Amount.*')
    )
else(true)endif

```

### **NDR3 [Rule 10-63] (REF,EXT). Name has representation term when appropriate**

[Rule 10-63](#), Name has representation term when appropriate (REF, EXT): [Section 10.8.7](#), Representation terms  
[English]

Constraint is non-computable.

### **NDR3 [Rule 10-64] (REF,EXT). Name has representation term only when appropriate**

[Rule 10-64](#), Name has representation term only when appropriate (REF, EXT): [Section 10.8.7](#), Representation terms  
[English]

Constraint is non-computable.

### **NDR3 [Rule 11-12] (REF,EXT). Element name is upper camel case**

[Rule 11-12](#), Element name is upper camel case (REF, EXT): [Section 11.2.1](#), Element declaration

#### **[OCL] context XSDProperty inv:**

```

(
    not(self.name.oclIsUndefined())
    and(self.name<>'')
    and self.clientDependency-
>select(d|d.stereotypeBy('References')).supplier-
>select(s|s.oclIsKindOf(Property))->isEmpty()
    and not(self.namespace.oclIsUndefined())

```

```

and not(self.namespace.stereotypedBy('List'))
and not(self.namespace.namespace.oclIsUndefined())
and self.namespace.namespace.stereotypedBy('InformationModel')
and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
and
(
    not(self.stereotypedBy('XSDProperty'))
    or
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
)
implies
self.niemName().match('^( [A-Z] [A-Za-z0-9\[-]* )+\$ ')

```

### **NDR3 [Rule 11-13] (REF,EXT). Element type does not have a simple type name**

[Rule 11-13](#), Element type does not have a simple type name (REF, EXT): [Section 11.2.1](#), Element declaration

#### [OCL] context XSDProperty inv:

```

(
(
    not(self.stereotypedBy('XSDProperty'))
    or
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
)
and not(self.type.oclIsUndefined())
and not(self.type.name.oclIsUndefined())
and not(self.namespace.oclIsUndefined())
and not(self.namespace.stereotypedBy('List'))
and not(self.namespace.namespace.oclIsUndefined())
and self.namespace.namespace.stereotypedBy('InformationModel')
and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
not(self.type.niemName().endsWith('SimpleType'))

```

### **NDR3 [Rule 11-14] (REF,EXT). Element type is from conformant namespace**

[Rule 11-14](#), Element type is from conformant namespace (REF, EXT): [Section 11.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    not(self.namespace.namespace.oclIsUndefined())  
    and  
    self.namespace.namespace.stereotypeBy('Namespace')  
    and  
    self.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
        and self.clientDependency-  
>select(d|d.stereotypeBy('References')).supplier-  
>select(s|s.oclIsKindOf(Property))->isEmpty()  
    and not(self.type.oclIsUndefined())  
    and self.type.namespace.stereotypeBy('Namespace')  
    and  
(  
    not(self.stereotypeBy('XSDProperty'))  
    or  
  
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
    )  
)  
implies  
self.type.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
```

**NDR3 [Rule 11-15] (REF,EXT). Name of element that ends in "Abstract" is abstract**

[Rule 11-15](#), Name of element that ends in Abstract is abstract (REF, EXT): [Section 11.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    (  
        not(self.stereotypeBy('XSDProperty'))  
        or  
  
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
        )  
        and  
    not(self.namespace.oclIsUndefined())  
        and  
    not(self.namespace.namespace.oclIsUndefined())  
        and
```

```

        self.namespace.namespace.stereotypedBy('Namespace')
        and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
        and
        self.niemName().endsWith('Abstract')
    )
implies
self.isDerivedUnion

```

**NDR3 [Rule 11-16] (REF,EXT). Name of element declaration with simple content has representation term**

[Rule 11-16](#), Name of element declaration with simple content has representation term (REF, EXT): [Section 11.2.1.1](#), Object element declarations

**[OCL] context XSDProperty inv:**

```

(
    (
        not(self.stereotypedBy('XSDProperty'))
        or

(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)
    )
    and self.isNavigable()
    and not(self.namespace.stereotypedBy('List'))
    and not(self.type.oclIsUndefined())
    and self.type.oclIsKindOf(DataType)
    and not(self.namespace.oclIsUndefined())
    and not(self.namespace.namespace.oclIsUndefined())
    and self.namespace.namespace.stereotypedBy('Namespace')
    and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies (
        self.niemName().match('.*Name.*') or
        self.niemName().match('.*Text.*') or
        self.niemName().match('.*List.*') or
        self.niemName().match('.*Quantity.*') or
        self.niemName().match('.*Percent.*') or
        self.niemName().match('.*Rate.*') or
        self.niemName().match('.*Value.*') or
        self.niemName().match('.*Numeric.*') or
        self.niemName().match('.*Measure.*') or
        self.niemName().match('.*Indicator.*') or
        self.niemName().match('.*URI.*')
        or self.niemName().match('.*ID.*')
        or self.niemName().match('.*Time.*') or

```

```

        self.niemName().match('.*Date.*') or
        self.niemName().match('.*Duration.*') or
        self.niemName().match('.*DateTime.*') or
        self.niemName().match('.*Code.*') or
        self.niemName().match('.*Video.*') or
        self.niemName().match('.*Sound.*') or
        self.niemName().match('.*Picture.*') or
        self.niemName().match('.*Graphic.*') or
        self.niemName().match('.*BinaryObject.*') or
        self.niemName().match('.*Amount.*')
    )
)

```

**NDR3 [Rule 11-17] (SET). Name of element declaration with simple content has representation term**

[Rule 11-17](#), Name of element declaration with simple content has representation term (SET): [Section 11.2.1.1](#), Object element declarations

**[OCL] context XSDProperty inv:**

```

(
    self.isNavigable()
    and
    (
        not(self.stereotypeBy('XSDProperty'))
        or

        (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKind
        Code::element)
    )
    and (not(self.type.oclIsUndefined()) and
self.type.oclIsKindOf(DataType))
    and not(self.namespace.oclIsUndefined())
    and not(self.namespace.stereotypeBy('List'))
    and not(self.namespace.namespace.oclIsUndefined())
    and self.namespace.namespace.stereotypeBy('Namespace')
    and
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies (
    self.niemName().match('.*Name.*') or
    self.niemName().match('.*Text.*') or
    self.niemName().match('.*List.*') or
    self.niemName().match('.*Quantity.*') or
    self.niemName().match('.*Percent.*') or
    self.niemName().match('.*Rate.*') or
    self.niemName().match('.*Value.*') or
    self.niemName().match('.*Numeric.*') or
    self.niemName().match('.*Measure.*') or
    self.niemName().match('.*Indicator.*') or
    self.niemName().match('.*URI.*')
    or self.niemName().match('.*ID.*')
)

```

```

        or self.niemName().match('.*Time.*') or
        self.niemName().match('.*Date.*') or
        self.niemName().match('.*Duration.*') or
        self.niemName().match('.*DateTime.*') or
        self.niemName().match('.*Code.*') or
        self.niemName().match('.*Video.*') or
        self.niemName().match('.*Sound.*') or
        self.niemName().match('.*Picture.*') or
        self.niemName().match('.*Graphic.*') or
        self.niemName().match('.*BinaryObject.*') or
        self.niemName().match('.*Amount.*')
)

```

### **NDR3 [Rule 11-18] (REF,EXT). Element substitution group defined by conformant schema**

[Rule 11-18](#), Element substitution group defined by conformant schema (REF, EXT): [Section 11.2.2](#), Element substitution group

#### [OCL] context XSDProperty inv:

```

(
(
    not(self.stereotype('XSDProperty'))
    or

(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKind
Code::element)
)
and
not(self.namespace.oclisUndefined())
and
not(self.namespace.namespace.oclisUndefined())
and
self.namespace.namespace.stereotypeBy('Namespace')
and
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.subsettedProperty.namespace.namespace->forAll(m|
    m.stereotypeBy('Namespace')
    and
m.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)

```

### NDR3 [Rule 11-19] (REF,EXT). Attribute type defined by conformant schema

[Rule 11-19](#), Attribute type defined by conformant schema (REF, EXT): [Section 11.2.3](#), Attribute declaration.

If kind=attribute, then an XSDProperty must have multiplicity 1..1 or 0..1, must not be a derived union and must not subset any other property. If the type is not empty, it must be a DataType.

[OCL] context XSDProperty inv:

```
(  
    (  
        (self.kindoclAsType(EnumerationLiteral)='attribute')  
        and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
            and  
self.base_Property.namespace.namespace.stereotypedBy('Namespace')  
            and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsTy  
pe(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
            and(not(self.base_Property.type.oclIsUndefined()))  
            and(not(self.base_Property.type.namespace.oclIsUndefined()))  
            and(not(self.base_Property.type.namespace.name.oclIsUndefined()))  
            and (  
                self.base_Property.type.namespace.stereotypedBy('Namespace')  
                or  
(self.base_Property.type.namespace.name='XMLPrimitiveTypes')  
            )  
        )  
        implies(  
            (self.base_Property.type.namespace.name='XMLPrimitiveTypes')  
            or  
  
self.base_Property.type.namespace.appliedStereotype('Namespace').oclAsType(NI  
EM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
        )  
    )  
and  
(  
    (  
        (self.kindoclAsType(EnumerationLiteral).name='attribute')  
        and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
            and  
self.base_Property.namespace.namespace.stereotypedBy('Namespace')  
            and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsTy  
pe(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    )  
    implies(  
        (self.base_Property.upper=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) )  
    )  
)
```

### NDR3 [Rule 11-20] (REF,EXT). Attribute name uses representation term

[Rule 11-20](#), Attribute name uses representation term (REF, EXT): [Section 11.2.3](#), Attribute declaration

[OCL] context XSDProperty inv:

```
( (self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute)
  and not(self.base_Property.namespace.oclIsUndefined())
  and not(self.base_Property.namespace.namespace.oclIsUndefined())
  and
  self.base_Property.namespace.namespace.stereotypedBy('InformationModel')
  and
  self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(
  NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
  and not(self.base_Property.name.oclIsUndefined())
  and (self.base_Property.niemName()<>'')
  and
  not(self.base_Property.namespace.namespace.appliedStereotype('InformationModel')
  .oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose.
  oclIsUndefined())
  and self.base_Property.clientDependency-
>select(d|d.stereotypeBy('References')).supplier
  ->select(s|s.oclIsKindOf(Property) and
  not(s.namespace.oclIsUndefined() or
  s.namespace.namespace.oclIsUndefined())).namespace.namespace
  ->forAll(a|a.stereotypeBy('InformationModel') and
  not(a.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM
  _Profile::InformationModel).defaultPurpose.oclIsUndefined()))
)
implies(
  self.base_Property.niemName().endsWith('List') or
  self.base_Property.niemName().endsWith('Name') or
  self.base_Property.niemName().endsWith('Text') or
  self.base_Property.niemName().endsWith('Quantity') or
  self.base_Property.niemName().endsWith('Percent') or
  self.base_Property.niemName().endsWith('Rate') or
  self.base_Property.niemName().endsWith('Value') or
  self.base_Property.niemName().endsWith('Numeric') or
  self.base_Property.niemName().endsWith('Measure') or
  self.base_Property.niemName().endsWith('Indicator') or
  self.base_Property.niemName().endsWith('URI') or
  self.base_Property.niemName().endsWith('ID')
  or self.base_Property.niemName().endsWith('Time') or
  self.base_Property.niemName().endsWith('Duration') or
  self.base_Property.niemName().endsWith('Date') or
  self.base_Property.niemName().endsWith('DateTime') or
  self.base_Property.niemName().endsWith('Code') or
  self.base_Property.niemName().endsWith('Video') or
  self.base_Property.niemName().endsWith('Sound') or
  self.base_Property.niemName().endsWith('Picture') or
  self.base_Property.niemName().endsWith('Graphic') or
  self.base_Property.niemName().endsWith('BinaryObject') or
  self.base_Property.niemName().endsWith('Amount')
)
```

### **NDR3 [Rule 11-21] (REF,EXT). Element or attribute declaration introduced only once into a type**

[Rule 11-21](#), Element or attribute declaration introduced only once into a type (REF, EXT): [Section 11.3.2.1](#), Element use

[English]

Satisfied by UML Constraint members\_distinguishable, where the Namespace is a Classifier and the members are Properties.

### **NDR3 [Rule 11-22] (REF,EXT). Element reference defined by conformant schema**

[Rule 11-22](#), Element reference defined by conformant schema (REF, EXT): [Section 11.3.2.1](#), Element use

**[OCL] context XSDProperty inv:**

```
(  
  (   
    not(self.stereotypedBy('XSDProperty'))  
    or  
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
  )  
  and not(self.namespace.oclIsUndefined())  
  and not(self.namespace.stereotypedBy('AdapterType'))  
  and not(self.namespace.namespace.oclIsUndefined())  
  and self.namespace.namespace.stereotypedBy('Namespace')  
  and  
  self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
self.clientDependency->select(d|d.stereotypedBy('References')).supplier-  
>select(s|s.oclIsKindOf(Property)).namespace.namespace  
->forAll(m|  
  m.stereotypedBy('Namespace')  
  and  
  m.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)
```

### **NDR3 [Rule 11-23] (REF,EXT). Referenced attribute defined by conformant schemas**

[Rule 11-23](#), Referenced attribute defined by conformant schemas (REF, EXT): [Section 11.3.3](#), Attribute use

**[OCL] context XSDProperty inv:**

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
)
implies
self.base_Property.clientDependency-
>select(d|d.stereotypedBy('References')).supplier-
>select(s|s.oclisKindOf(Property)).namespace.namespace
->forAll(m|
    m.stereotypedBy('Namespace')
    and (
        m.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Prof
ile::Namespace).isConformant
        or(m.name='xml')
    )
)
)

```

### NDR3 [Rule 11-31] (REF,EXT). Standard opening phrase for element

[Rule 11-31](#), Standard opening phrase for element (REF, EXT): [Section 11.6.1.1](#), Data definition opening phrases

### [OCL] context XSDProperty inv:

```

(
(
    not(self.stereotypedBy('XSDProperty'))
    or
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_P
rofile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKind
Code::element)
)
    and self.isNavigable()
    and not(self.namespace.oclisUndefined())
    and not(self.namespace.stereotypedBy('List'))
    and not(self.namespace.namespace.oclisUndefined())
    and self.namespace.namespace.stereotypedBy('Namespace')
    and
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Pr
ofile::NIEM_Common_Profile::Namespace).isConformant
    and not(self.name.oclisUndefined())
    and (self.niemName()<>'')
    and self.clientDependency-
>select(d|d.stereotypedBy('References')).supplier-
>select(s|s.oclisKindOf(Property))->isEmpty()
)
implies

```

**NDR3 [Rule 11-36] (SET). Reference schema imports reference schema.**

[Rule 11-36](#), Reference schema imports reference schema (SET): [Section 11.8](#), Schema assembly

[OCL] context XSDProperty inv:

```
(  
    not(self.namespace.oclIsUndefined() or  
self.namespace.namespace.oclIsUndefined())  
    and not(self.namespace.stereotypeBy('AdapterType'))  
    and self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and  
self.namespace.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose->exists(purpose)|  
  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)  
    or  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)  
    )  
)  
implies  
    self->select(a|not(a.type.oclIsUndefined())).type.namespace->select(p|p.oclIsKindOf(Package)).oclAsType(NamedElement)->asSet()  
        ->union(self.clientDependency->select(d|d.stereotypeBy('References')).supplier->select(s|s.oclIsKindOf(Property)).namespace.namespace.oclAsType(NamedElement)->asSet())  
        ->forAll(p|  
            (p.name='XMLPrimitiveTypes')  
            or(p.name='xml')  
            or (p.stereotypeBy('InformationModel')  
                and  
not(p.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose.oclIsUndefined())  
                and  
p.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel).defaultPurpose->exists(purpose)|  
  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)  
    or  
(purpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)  
    )  
)
```

### NDR3 [Rule 9-36] (REF,EXT). Element declaration has data definition

[Rule 9-36](#), Element declaration has data definition (REF, EXT): [Section 9.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    self.isNavigable()  
    and not(self.namespace.oclIsUndefined())  
    and not(self.namespace.stereotypeBy('List'))  
    and not(self.namespace.namespace.oclIsUndefined())  
    and self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and  
    (not(self.stereotypeBy('XSDProperty')) or  
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
        and self.clientDependency-  
>select(d|d.stereotypeBy('References')).supplier-  
>select(s|s.oclIsKindOf(Property))->isEmpty()  
)  
implies  
    self.ownedComment->exists(c|not(c._'body'.oclIsUndefined()) and  
(c._'body'<>''))
```

**NDR3 [Rule 9-37] (REF,EXT). Untyped element is abstract**

[Rule 9-37](#), Untyped element is abstract (REF, EXT): [Section 9.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    not(self.namespace.namespace.oclIsUndefined())  
    and  
    self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and  
    (not(self.stereotypeBy('XSDProperty')) or  
(self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element)  
)  
implies  
(self.type.oclIsUndefined() implies self.isDerivedUnion)
```

**NDR3 [Rule 9-38] (REF,EXT). Element of type xs:anySimpleType is abstract**

[Rule 9-38](#), Element of type xs:anySimpleType is abstract (REF, EXT): [Section 9.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    (not(self.stereotypedBy('XSDProperty')))      or  
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))  
    and  
        not(self.namespace.oclIsUndefined())  
    and  
        not(self.namespace.namespace.oclIsUndefined())  
    and  
        self.namespace.namespace.stereotypedBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and  
        not(self.type.oclIsUndefined())  
    and  
        (self.type.name='anySimpleType')  
    and  
        (self.type._'package'.name='XMLPrimitiveTypes')  
)  
implies  
self.isDerivedUnion
```

### NDR3 [Rule 9-40] (REF,EXT). Element type not in the XML namespace

[Rule 9-40](#), Element type not in the XML namespace (REF, EXT): [Section 9.2.1](#), Element declaration

**[OCL] context XSDProperty inv:**

```
(  
    (not(self.stereotypedBy('XSDProperty'))  
     or  
    (self.appliedStereotype('XSDProperty').oclAsType(NIEM_UML_Profile::NIEM_PSM_Profile::XSDProperty).kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::element))  
    )  
    and not(self.type.oclIsUndefined() or self.namespace.oclIsUndefined() or  
    self.namespace.namespace.oclIsUndefined())  
    and self.namespace.namespace.stereotypedBy('Namespace')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
    and self.clientDependency-  
>select(d|d.stereotypedBy('References')).supplier-  
>select(s|s.oclIsKindOf(Property))->isEmpty()  
)  
implies
```

```
(self.type._'package'.name<>'xml')
```

### NDR3 [Rule 9-43] (REF). No element disallowed derivation

[Rule 9-43](#), No element disallowed derivation (REF): [Section 9.2.1](#), Element declaration

#### [OCL] context XSDProperty inv:

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    not(self.namespace.namespace.oclIsUndefined())  
    and  
    self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
(  
  
    self.namespace.namespace.appliedStereotype('InformationModel').oclAsType(NIEM_UML_Profile::NIEM_PIM_Profile::InformationModel)  
    ->forAll(im|  
  
        (im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::reference)  
  
        or(im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)  
            )  
        implies  
        not(self.isLeaf)  
)
```

### NDR3 [Rule 9-46] (REF). Element declaration is nullable

[Rule 9-46](#), Element declaration is nullable (REF): [Section 9.2.1](#), Element declaration

#### [OCL] context XSDProperty inv:

```
(  
    not(self.namespace.oclIsUndefined())  
    and  
    not(self.namespace.namespace.oclIsUndefined())  
    and  
    self.namespace.namespace.stereotypeBy('InformationModel')  
    and  
    self.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
```

```

        and
self.namespace.namespace.appliedStereotype('InformationModel').oclAsType(NIEM
_UML_Profile::NIEM_PIM_Profile::InformationModel)
->exists(im|
(im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::re
ference)
or(im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)
)
)
implies
(
self.namespace.namespace.appliedStereotype('InformationModel').oclAsType(NIEM
_UML_Profile::NIEM_PIM_Profile::InformationModel)
->forAll(im|
(im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::re
ference)
or(im.defaultPurpose=NIEM_UML_Profile::NIEM_PIM_Profile::DefaultPurposeCode::subset)
)
implies
not(self.isLeaf)
)

```

### NDR3 [Rule 9-48] (REF,EXT). Attribute declaration has data definition

[Rule 9-48](#), Attribute declaration has data definition (REF, EXT): [Section 9.2.3](#), Attribute declaration

#### [OCL] context XSDProperty inv:

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
    and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
    and self.base_Property.namespace.namespace.stereotypeBy('Namespace')
    and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsTy
pe(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.base_Property.ownedComment->exists(c|not(c._'body'.oclIsUndefined()) and
(c._'body'<>''))

```

### **NDR3 [Rule 9-50] (REF,EXT). No attribute type of xs:ID**

[Rule 9-50](#), No attribute type of xs : ID (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

**[OCL] context XSDProperty inv:**

```
(  
  
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute  
)  
    and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
    and self.base_Property.namespace.namespace.stereotypeBy('Namespace')  
        and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
self.base_Property.type-  
>forAll(t|not((t.name='ID') and (t.namespace.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-51] (REF,EXT). No attribute type of xs:IDREF**

[Rule 9-51](#), No attribute type of xs : IDREF (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

**[OCL] context XSDProperty inv:**

```
(  
  
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute  
)  
    and not(self.base_Property.namespace.oclIsUndefined() or  
self.base_Property.namespace.namespace.oclIsUndefined())  
    and self.base_Property.namespace.namespace.stereotypeBy('Namespace')  
        and  
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant  
)  
implies  
self.base_Property.type-  
>forAll(t|not((t.name='IDREF') and (t.namespace.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-52] (REF,EXT). No attribute type of xs:IDREFS**

[Rule 9-52](#), No attribute type of xs : IDREFS (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

**[OCL] context XSDProperty inv:**

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
    and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
    and self.base_Property.namespace.namespace.stereotype('Namespace')
    and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.base_Property.type-
>forAll(t|not((t.name='IDREFS') and (t.namespace.name='XMLPrimitiveTypes')))
```

### **NDR3 [Rule 9-53] (REF,EXT). No attribute type of xs:ENTITY**

[Rule 9-53](#), No attribute type of xs:ENTITY (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

#### **[OCL] context XSDProperty inv:**

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
    and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
    and self.base_Property.namespace.namespace.stereotype('Namespace')
    and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.base_Property.type-
>forAll(t|not((t.name='ENTITY') and (t.namespace.name='XMLPrimitiveTypes')) )
```

### **NDR3 [Rule 9-54] (REF,EXT). No attribute type of xs:ENTITIES**

[Rule 9-54](#), No attribute type of xs:ENTITIES (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

#### **[OCL] context XSDProperty inv:**

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
    and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
```

```

        and self.base_Property.namespace.namespace.stereotypedBy('Namespace')
        and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsTy
pe(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.base_Property.type-
>forAll(t|not((t.name='ENTITIES') and (t.namespace.name='XMLPrimitiveTypes')) )

```

### **NDR3 [Rule 9-55] (REF,EXT). No attribute type of xs:anySimpleType**

[Rule 9-55](#), No attribute type of xs:anySimpleType (REF, EXT): [Section 9.2.3.1](#), Prohibited attribute types

#### [OCL] context XSDProperty inv:

```

(
(self.kind=NIEM_UML_Profile::NIEM_PSM_Profile::XSDPropertyKindCode::attribute
)
and not(self.base_Property.namespace.oclIsUndefined() or
self.base_Property.namespace.namespace.oclIsUndefined())
and self.base_Property.namespace.namespace.stereotypedBy('Namespace')
and
self.base_Property.namespace.namespace.appliedStereotype('Namespace').oclAsTy
pe(NIEM_UML_Profile::NIEM_Common_Profile::Namespace).isConformant
)
implies
self.base_Property.type-
>forAll(t|not((t.name='anySimpleType') and (t.namespace.name='XMLPrimitiveTypes
')) )

```

### **NDR3 [Rule 9-56] (REF,EXT). No attribute default values**

[Rule 9-56](#), No attribute default values (REF, EXT): [Section 9.2.3.2](#), No attribute value constraints

[English]

This constraint enforced by provisioning, there are no @default attributes generated for an xs:attribute within a target InformationModel schema.

### **NDR3 [Rule 9-57] (REF,EXT). No attribute fixed values**

[Rule 9-57](#), No attribute fixed values (REF, EXT): [Section 9.2.3.2](#), No attribute value constraints

[English]

This constraint enforced by provisioning, there are no @fixed attributes generated for an xs:attribute within a target InformationModel schema.

## 8.4.5 <Stereotype> XSDRepresentationRestriction

### Description

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

### Extends

UML::DataType

### Properties

#### whiteSpace : XSDWhiteSpaceCode [0..1]

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 xs:whiteSpace element, the child of the xs:restriction element which is the immediate child of the xs:simpleType element.

### Constraints

#### XSDRepresentationRestriction

A DataType with an XSDRepresentationRestriction must have exactly one generalization.

[OCL] context XSDRepresentationRestriction inv:

```
( self.base_DataType.generalization->size() +self.base_DataType.clientDependency->select(d|d.stereotypeBy('Restriction')).supplier->size() )=1
```

## 8.4.6 <Stereotype> XSDSimpleContent

### 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 xs:restriction to the other complex type. Otherwise, the relationship is implemented in XML Schema through base attribute on the xs: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; [Section 9.1.3.3](#) of [NIEM-NDR] addresses simple content types in NIEM-conformant XML Schema.

## **Extends**

UML::Realization

## **Constraints**

### **XSDSimpleContent**

The client of an «XSDSimpleContent» Realization must be a Classifier owned by a «Namespace» Package.

The suppler of an «XSDSimpleContent» Realization must be a DataType.

**[OCL] context XSDSimpleContent inv:**

```
self.base_Realization.client->forAll(client|client.oclIsKindOf(Classifier)
and client.namespace.stereotypedBy('Namespace'))
and
self.base_Realization.supplier->forAll(s|s.oclIsKindOf(DataType))
```

## **8.4.7 <Enumeration> XSDProcessContentsCode**

### **Description**

XSDProcessContentsCode supports the processContents attribute of the XSDAnyProperty stereotype.

### **Literals**

**strict**

**lax**

**skip**

## **8.4.8 <Enumeration> XSDPropertyKindCode**

### **Description**

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

### **Literals**

**element**

**attribute**

## **8.4.9 <Enumeration> XSDWhiteSpaceCode**

### **Description**

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

## **Literals**

**replace**

**collapse**

**preserve**

## 8.5 Profile : Model\_Package\_Description\_Profile

### 8.5.1 Overview

The Model Package Description Profile comprises stereotypes and artifacts that are used to model NIEM MPDs. The diagrams show all the stereotypes, artifacts and enumerations defined in this profile.

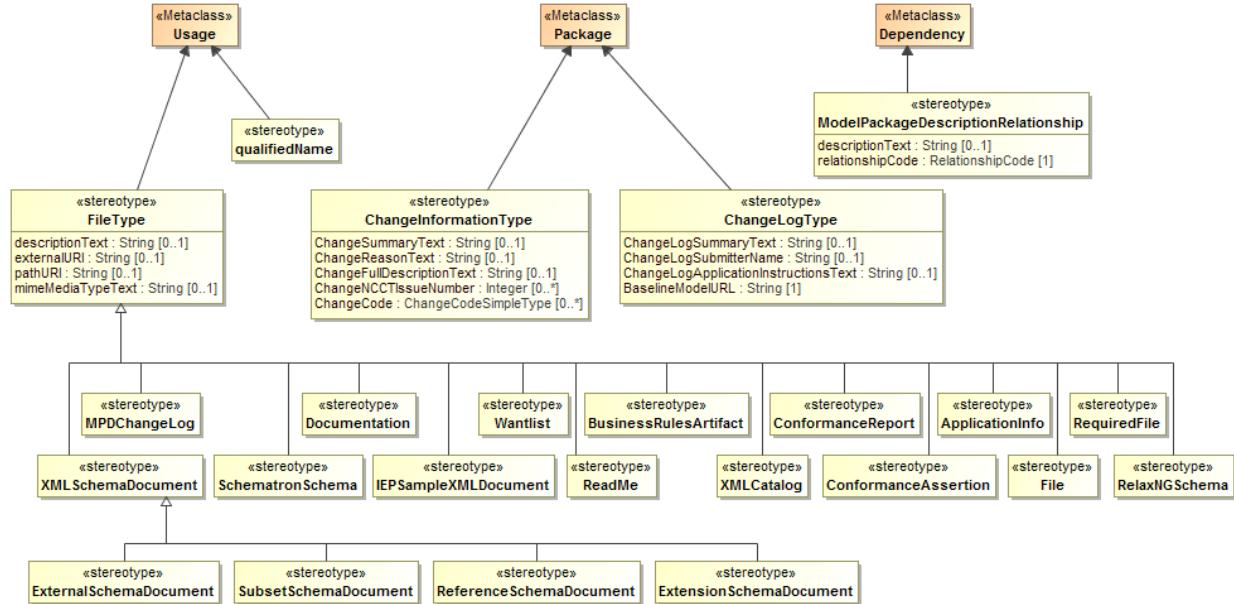


Figure 8-5 Model Package Description profile stereotypes

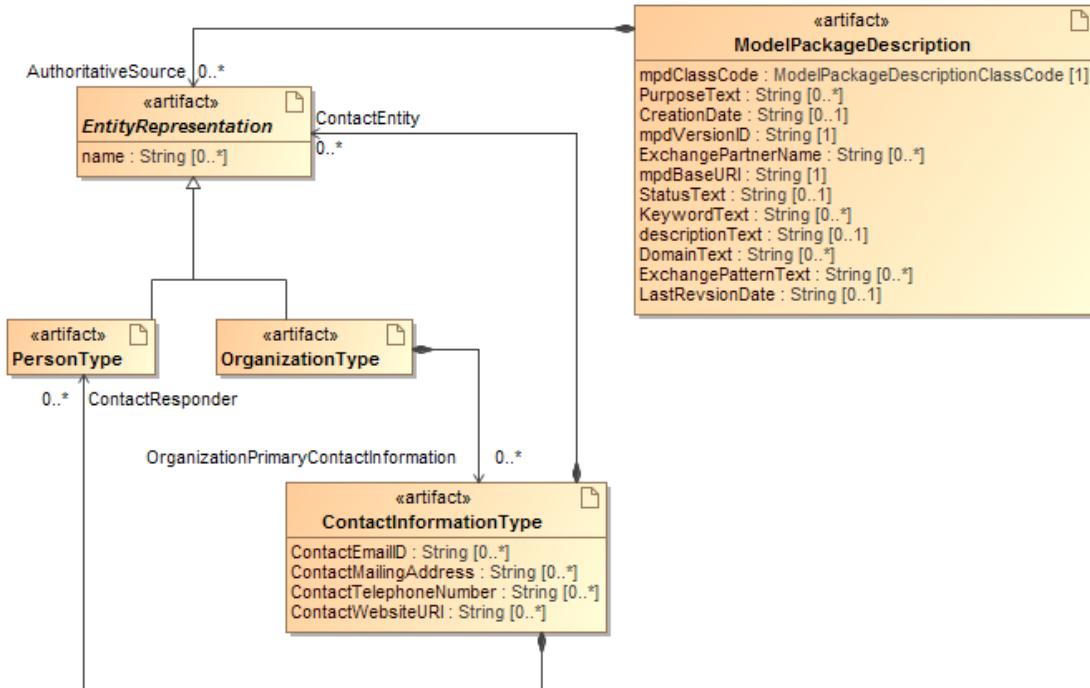


Figure 8-6 MPD Profile Artifacts for contact information

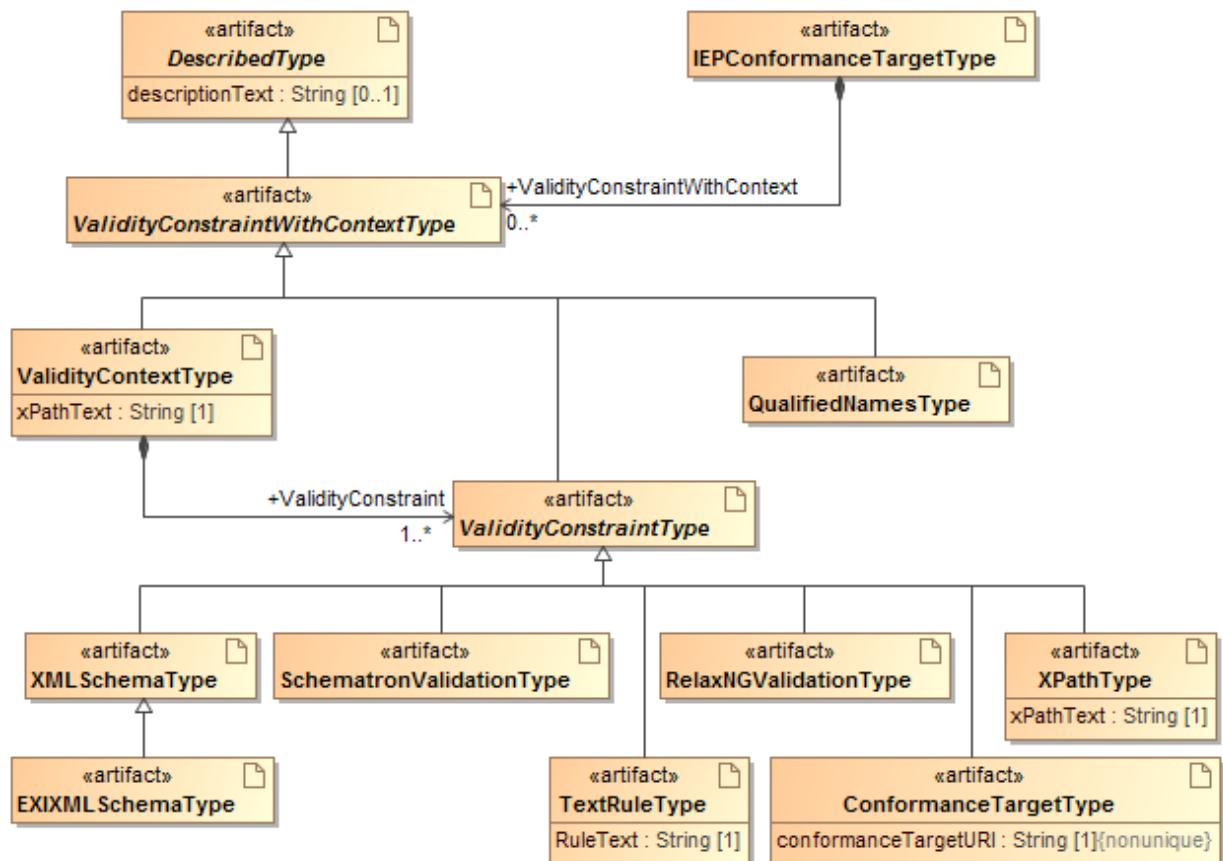


Figure 8-7 MPD Profile Artifacts for validation

«enumeration» RelationshipCode	«enumeration» ChangeCodeSimpleType	«enumeration» ModelPackageDescriptionClassCode
<ul style="list-style-type: none"> <li>updates</li> <li>conforms_to</li> <li>version_of</li> <li>specializes</li> <li>generalizes</li> <li>supersedes</li> <li>deprecates</li> <li>adapts</li> <li>derives_from</li> </ul>	<ul style="list-style-type: none"> <li>new_requirement</li> <li>bug_fix</li> <li>refactoring</li> <li>harmonization</li> <li>general_improvement</li> </ul>	<ul style="list-style-type: none"> <li>eiem</li> <li>iepd</li> <li>core_update</li> <li>release</li> <li>domain_update</li> </ul>

Figure 8-8 MPD Profile Enumerations

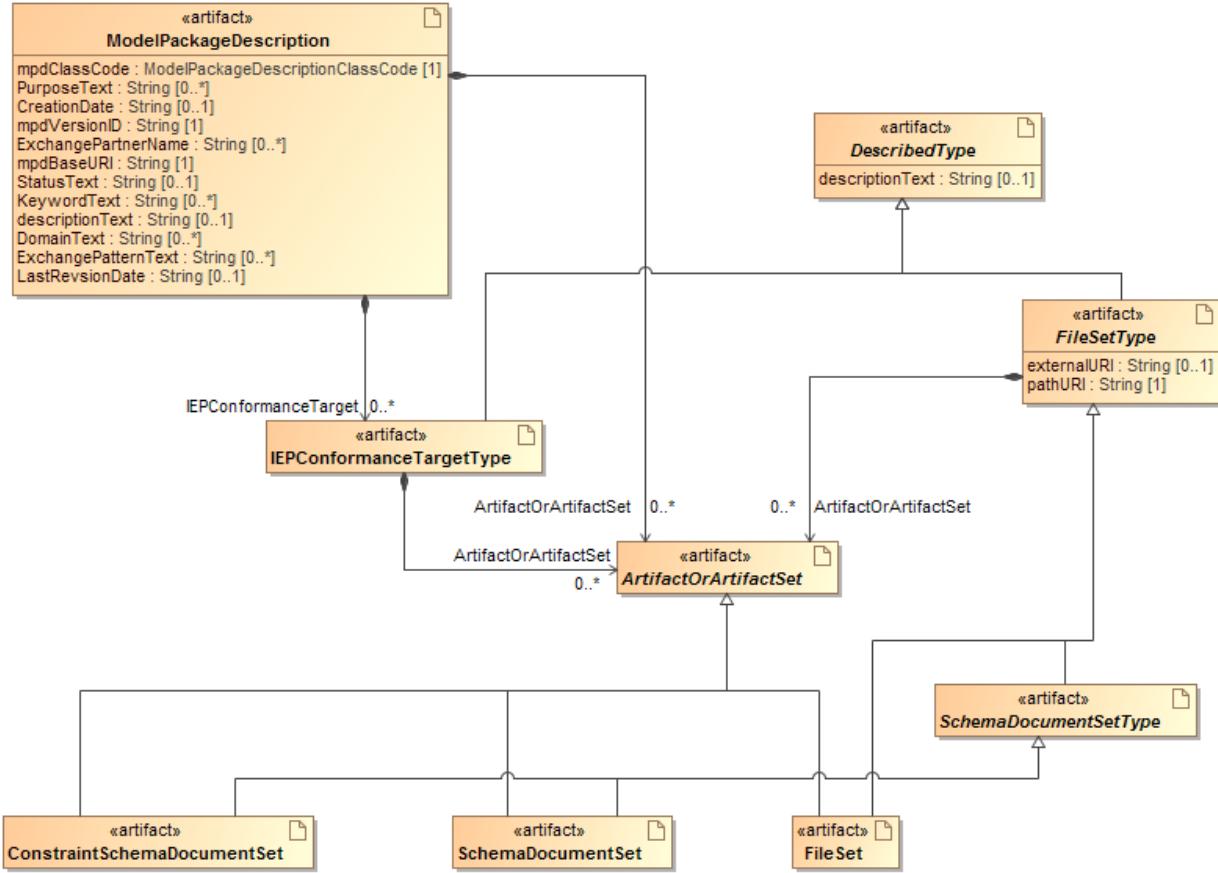


Figure 8-9 MPD Profile Artifacts for files and documents

## 8.5.2 <Stereotype> ApplicationInfo

### Description

An MPD artifact that is used by a software tool (e.g., import, export, input, output, etc.).

### Generalization

[FileType](#)

## 8.5.3 <Stereotype> BusinessRulesArtifact

### Description

An MPD artifact that contains business rules and constraints on exchange content.

### Generalization

[FileType](#)

## **8.5.4 <Stereotype> ChangeInformationType**

### **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.

### **Extends**

UML::Package

### **Properties**

#### **ChangeCode : ChangeCodeSimpleType [0..\*]**

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

#### **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.

#### **ChangeReasonText : String [0..1]**

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

#### **ChangeSummaryText : String [0..1]**

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

## **8.5.5 <Stereotype> ChangeLogType**

### **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.

### **Extends**

UML::Package

### **Properties**

#### **BaselineModelURL : String [1]**

URL of baseline model the change log applies to.

**ChangeLogApplicationInstructionsText : String [0..1]**

Descriptive text representing change log applications instructions.

**ChangeLogSubmitterName : String [0..1]**

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

**ChangeLogSummaryText : String [0..1]**

Descriptive text providing a summary of the change log.

## **8.5.6 <Stereotype> ConformanceAssertion**

**Description**

An MPD artifact that represents a declaration that a NIEM IEPD or EIEM is NIEM-conformant.

**Generalization**

[FileType](#)

## **8.5.7 <Stereotype> ConformanceReport**

**Description**

An MPD artifact either auto-generated by a NIEM-aware software tool or manually prepared that checks NIEM conformance and/or quality and renders a detailed report of results. This report may also be an auto-generated and manually prepared hybrid artifact.

**Generalization**

[FileType](#)

## **8.5.8 <Stereotype> Documentation**

**Description**

An MPD artifact that is a form of explanatory documentation.

**Generalization**

[FileType](#)

## **8.5.9 <Stereotype> ExtensionSchemaDocument**

**Description**

An MPD artifact that is a NIEM extension schema document.

**Generalization**

[XMLSchemaDocument](#)

## **8.5.10 <Stereotype> ExternalSchemaDocument**

### **Description**

An MPD artifact that is a schema document external to NIEM.

### **Generalization**

[XMLSchemaDocument](#)

## **8.5.11 <Stereotype> File**

### **Description**

A generic electronic file artifact in an MPD; a file stored on a computer system.

### **Generalization**

[FileType](#)

## **8.5.12 <Stereotype> FileType**

### **Description**

A data type for an MPD file artifact.

### **Extends**

UML::Usage

### **Properties**

#### **descriptionText : String [0..1]**

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

#### **externalURI : String [0..1]**

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.

#### **mimeMediaTypeText : String [0..1]**

A classification for an MPD file artifact from the IANA MIME media classes:

<http://www.iana.org/assignments/media-types>.

#### **pathURI : String [0..1]**

The relative path name to the file within the MPD directory structure. Implemented as the value of the pathURI attribute of the FileType type in the catalog instance.

## **8.5.13 <Stereotype> IEPSampleXMLDocument**

### **Description**

An example MPD instance XML document or IEP artifact.

### **Generalization**

[FileType](#)

## **8.5.14 <Stereotype> ModelPackageDescriptionRelationship**

### **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 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 relationshipCode values available (version\_of, specializes, generalizes, deprecates, supersedes, adapts, conforms\_to, updates, derives\_from). 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").

### **Extends**

UML::Dependency

### **Properties**

#### **descriptionText : String [0..1]**

A more detailed or specific textual explanation of the relationship between the MPDs or between an MPD and a resource (such as a specification).

#### **relationshipCode : RelationshipCode [1]**

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

## **8.5.15 <Stereotype> MPDChangeLog**

### **Description**

An MPD artifact that contains a record of the MPD changes.

### **Generalization**

[FileType](#)

## **8.5.16 <Stereotype> qualifiedName**

### **Description**

See Rule 5-42.

## Extends

UML::Usage

## Constraints

### MPD3 [Rule 5-42] (IEP). Identifying the Document Element of an IEP

[Rule 5-42](#), Identifying the Document Element of an IEP: [Section 5.6.2.4](#), c : HasDocumentElement

To identify a Document Element of an IEP in UML do the following:

Add an instance of IEPConformanceTargetType to the IEPConformanceTarget slot of the ModelPackageDescription Artifact instance.

Add a QualifiedNamesType instance to the ValidityConstraintWithContext slot of the new IEPConformanceTargetType instance.

Add a Usage with applied Stereotype «qualifiedName» where the client is the new QualifiedNamesType instance and the supplier is a Property representing an XSD Element.

The «qualifiedName» Usage will be transformed into a QName entry within the "qualifiedNameList" attribute of a "HasDocumentElement" Element within a "IEPConformanceTarget" of an MPD Catalog. By rule 5-42, the QName identifies a Document Element of an IEP. The Constraint Specification ensures that a «qualifiedName» Usage has a client of QualifiedNamesType InstanceSpecification and a supplier of a Property representing an XSD Element.

**[OCL] context qualifiedName inv:**

```
self.base_Usage.client->forAll(c|c.oclIsKindOf(InstanceSpecification) and
c.oclAsType(InstanceSpecification).classifier.name-
>exists(n|n='QualifiedNamesType'))
and
self.base_Usage.supplier->forAll(s|
  s.oclIsKindOf(Property)
  and s.oclAsType(Property).namespace.namespace.stereotypedBy('Namespace')
  and
  s.namespace.namespace.appliedStereotype('Namespace').oclAsType(NIEM_UML_Profi
le::NIEM_Common_Profile::Namespace).isConformant

  and
  (s.oclAsType(Property).niemName().substring(1,1)=s.oclAsType(Property).niemNa
me().substring(1,1).toUpperCase())
)
```

## 8.5.17 <Stereotype> ReadMe

### Description

An MPD read-me artifact.

### Generalization

[FileType](#)

## **8.5.18 <Stereotype> ReferenceSchemaDocument**

### **Description**

An MPD artifact that is a reference schema document (from a release, domain update, or core update).

### **Generalization**

[XMLSchemaDocument](#)

## **8.5.19 <Stereotype> RelaxNGSchema**

### **Description**

A RelaxNG schema.

### **Generalization**

[FileType](#)

## **8.5.20 <Stereotype> RequiredFile**

### **Description**

An MPD file artifact that another artifact depends on and should not be separated from.

### **Generalization**

[FileType](#)

## **8.5.21 <Stereotype> SchematronSchema**

### **Description**

A Schematron schema document.

### **Generalization**

[FileType](#)

## **8.5.22 <Stereotype> SubsetSchemaDocument**

### **Description**

An MPD artifact that is a subset schema document.

### **Generalization**

[XMLSchemaDocument](#)

## **8.5.23 <Stereotype> Wantlist**

### **Description**

An MPD artifact that represents a NIEM schema subset and is used as an import or export for the NIEM SSGT. See [Section 6.1](#) of [NIEM-MPD].

### **Generalization**

[FileType](#)

## **8.5.24 <Stereotype> XMLCatalog**

### **Description**

An MPD artifact that is an OASIS XML catalog.

### **Generalization**

[FileType](#)

## **8.5.25 <Stereotype> XMLSchemaDocument**

### **Description**

An MPD artifact that is an XML schema document (i.e., an XSD that is not necessarily a NIEM subset, extension, or reference schema).

### **Generalization**

[FileType](#)

## **8.5.26 <Artifact> ArtifactOrArtifactSet**

### **Description**

A data concept for a file or file set in an MPD.

## **8.5.27 <Artifact> ConformanceTargetType**

### **Description**

A data type for identifying and describing a conformance target.

### **Generalization**

[ValidityConstraintType](#)

### **Properties**

**conformanceTargetURI : String [1]**

A URI for a conformance target.

## 8.5.28 <Artifact> ConstraintSchemaDocumentSet

### Description

An MPD artifact set of constraint schema documents and other supporting artifacts.

### Generalization

[ArtifactOrArtifactSet](#) [SchemaDocumentSetType](#)

## 8.5.29 <Artifact> ContactInformationType

### Description

A data type for how to contact a person or an organization.

### Properties

#### ContactEmailID : String [0..\*]

An electronic mailing address by which a person or organization may be contacted.

#### ContactEntity : EntityRepresentation [0..\*]

An entity that may be contacted by using the given contact information.

#### ContactMailingAddress : String [0..\*]

A postal address by which a person or organization may be contacted.

#### ContactResponder : PersonType [0..\*]

A third party person who answers a call and connects or directs the caller to the intended person.

#### ContactTelephoneNumber : String [0..\*]

A telephone number for a telecommunication device by which a person or organization may be contacted.

#### ContactWebsiteURI : String [0..\*]

A website address by which a person or organization may be contacted.

## 8.5.30 <Artifact> DescribedType

### Description

Common supertype for NIEM MPD Catalog types which have descriptionText.

### Properties

#### descriptionText : String [0..1]

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

### **8.5.31 <Artifact> EntityRepresentation**

#### **Description**

A data concept for a person, organization, or thing capable of bearing legal rights and responsibilities.

#### **Properties**

**name : String [0..\*]**

A combination of names and/or titles by which an entity is known.

### **8.5.32 <Artifact> EXIXMLSchemaType**

#### **Description**

An XML Schema to be used for EXI serialization of an IEP Class.

#### **Generalization**

[ArtifactOrArtifactSet XMLSchemaType](#)

### **8.5.33 <Artifact> FileSet**

#### **Description**

A generic MPD artifact set; used to group artifacts that are not accounted for by other set classifiers.

#### **Generalization**

[ArtifactOrArtifactSet FileType](#)

### **8.5.34 <Artifact> FileType**

#### **Description**

A data type for a set of MPD file artifacts.

#### **Generalization**

[DescribedType](#)

#### **Properties**

**ArtifactOrArtifactSet : ArtifactOrArtifactSet [0..\*]**

A data concept for a file or file set in an MPD.

**externalURI : String [0..1]**

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.

#### **pathURI : String [1]**

The relative path name to the file within the MPD directory structure. Implemented as the value of the pathURI attribute of the FileType type in the catalog instance.

### **8.5.35 <Artifact> IEPConformanceTargetType**

#### **Description**

A data type for a class or category of IEP, which has a set of validity constraints and a unique identifier.

#### **Generalization**

[DescribedType](#)

#### **Properties**

##### **ArtifactOrArtifactSet : ArtifactOrArtifactSet [0..\*]**

A data concept for a file or file set in an MPD.

##### **ValidityConstraintWithContext : ValidityConstraintWithContextType [0..\*]**

A data concept for a rule or instructions for validating an IEP candidate (XML document) using some context within that XML document.

#### **Constraints**

##### **MPD3 [Rule 5-45] (IEP). Validating an IEP Sample XML Document**

[Rule 5-45.](#) Validating an IEP Sample XML Document: [Section 5.6.3.](#) IEP Sample Instance XML Documents

[English]

The constraint is enforced during provisioning of the Sample XML Document.

### **8.5.36 <Artifact> ModelPackageDescription**

#### **Description**

A ModelPackageDescription Artifact represents a NIEM Model Package Description (MPD). Specifically, it represents the information in the NIEM-3 MPD catalog, which is defined for target namespace=<http://reference.niem.gov/niem/resource/mpd/catalog/3.0/>, in the context of a NIEM-3 subset of the niem-core schema.

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 InstanceSpecification representing the MPD to the model element representing the artifact (most commonly a Namespace Package).

The attributes of the ModelPackageDescription correspond to components of the MPDType within the MPD Catalog Schema. The information model fragment corresponding to c:MPDInformation has been partially flattened from the schema containment structure into the ModelPackageDescription Attributes.

In addition to the largely isomorphic representation of the MPD Catalog as parts of the ModelPackageDescription, there are a few convenience mechanisms to simplify the UML Model. The representation of the MPD relationship between MPDs is modeled as a «ModelPackageDescriptionRelationship» Dependency from the client ModelPackageDescription Artifact Instance to the related ModelPackageDescription Artifact Instance as supplier.

Instances of ModelPackageDescription may be the client of a UML Usage to some supplying NIEM concept, such as «InformationModel». These UML Usages may be stereotyped with a sub-stereotype of «FileType». When such a Usage is defined, values of some stereotype tags may be derived, such as the pathURI (based on UML Package Structure and/or NIEM packaging structure guidelines). Note that most «FileType»s are implicit anyway, being derived by transitive closure of all «InformationModel»s referenced by used «InformationModel»s.

## Properties

### **ArtifactOrArtifactSet : ArtifactOrArtifactSet [0..\*]**

A data concept for a file or file set in an MPD.

### **AuthoritativeSource : EntityRepresentation [0..\*]**

An official sponsoring or authoring organization responsible for an MPD.

### **CreationDate : String [0..1]**

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

### **descriptionText : String [0..1]**

A description of the MPD. A statement that provides an explanation or additional detail.

Implemented as the value of the DescriptionText element of the MPDType type within the MPD Catalog document instance.

### **DomainText : String [0..\*]**

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 : String [0..\*]**

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 : String [0..\*]**

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.

### **IEPConformanceTarget : IEPConformanceTargetType [0..\*]**

A class or category of IEPs which has a set of validity constraints and a unique identifier. Every IEP is an instance of one or more IEP Conformance Targets.

### **KeywordText : String [0..\*]**

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.

#### **LastRevisionDate : String [0..1]**

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

#### **mpdBaseURI : String [1]**

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 MPD element in the catalog instance. The last segment of mpdBaseURI becomes the value of the mpdName attribute of the MPD element in the catalog instance.

Note that the relationship between mpdBaseURI, mpdURI, and mpdName are more restrictive than the rules expressed in NIEM 3 MPD, but are consistent with guidelines recommended/implied by the NIEM 3 MPD.

#### **mpdClassCode : ModelPackageDescriptionClassCode [1]**

The classification code of the MPD. Maps to the value of the mpdClassURIList attribute of the MPDType within the catalog instance. This code designates the classification or kind of the MPD.

Note that NIEM-3 MPD explicitly defines only the iepd classification code, with the other classification codes implied but not formally defined with the MPD specification.

#### **mpdVersionID : String [1]**

Many published MPDs will be periodically revised and updated; therefore, versioning is required to clearly indicate that changes have occurred. See [Section 5.2.3](#) of [NIEM-MPD]. A version number is actually part of the unique identification for an MPD. 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

Note that the value of mpdVersionID is concatenated with mpdBaseURI to form the mpdURI. This convention is more restrictive than the NIEM 3 MPD rules, but is consistent with recommended/implied MPD naming conventions.

#### **PurposeText : String [0..\*]**

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.

#### **StatusText : String [0..1]**

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.

### **Constraints**

#### **MPD3 [Rule 3-2] (MPD). MPD with MPD class of IEPD is an IEPD**

[Rule 3-2](#), MPD with MPD class of IEPD is an IEPD: [Section 3.2.2](#), IEPD Conformance Target  
[English]

Rule is definitional

### **MPD3 [Rule 3-3] (IEPD). IEPD Conformance Target Identifier**

[Rule 3-3](#), IEPD Conformance Target Identifier: [Section 3.2.2](#), IEPD Conformance Target  
[English]

Constraint realized during provisioning of MPD Catalog.

### **MPD3 [Rule 4-1] (Schema-subset). Fundamental NIEM Subset Rule**

[Rule 4-1](#), Fundamental NIEM Subset Rule: [Section 4.2.1](#), Basic Subset Concepts  
[English]

Rule is definitional.

### **MPD3 [Rule 5-10] (WF-MPD). MPD Version Number Syntax**

[Rule 5-10](#), MPD Version Number Syntax: [Section 5.2.3](#), MPD Version Numbering Scheme (`c:mpdVersionID`)  
[OCL] context ModelPackageDescription inv:  
`self.mpdVersionID.match('[0-9]+(\.\.[0-9]+)*((alpha|beta|rc|rev)[0-9]+)?')`

### **MPD3 [Rule 5-11] (WF-MPD). MPD URI Is Absolute**

[Rule 5-11](#), MPD URI Is Absolute: [Section 5.2.4.1](#), MPD URI Scheme (`c:mpdURI`)  
[English]

Expressing constraint in OCL is deferred.

### **MPD3 [Rule 5-12] (WF-MPD). MPD URI Supports Fragment**

[Rule 5-12](#), MPD URI Supports Fragment: [Section 5.2.4.2](#), URI Scheme for MPD Artifacts (`c:externalURI`)  
[English]

Constraint is definitional.

### **MPD3 [Rule 5-13] (WF-MPD). MPD URI Has No Fragment [Rule 5-13] (WF-MPD) (Constraint)**

[Rule 5-13](#), MPD URI Has No Fragment: [Section 5.2.4.2](#), URI Scheme for MPD Artifacts (`c:externalURI`)

[English]

Constraint is definitional.

#### **MPD3 [Rule 5-14] (WF-MPD). MPD Artifact URI Syntax**

[Rule 5-14](#), MPD Artifact URI Syntax: [Section 5.2.4.2](#), URI Scheme for MPD Artifacts (c:externalURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-15] (WF-MPD). c:pathURI Resolves to a Resource**

[Rule 5-15](#), c:pathURI Resolves to a Resource: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog. c:pathURI is either set implicitly to a location, or if a value is provided, the resource is moved to specified location.

#### **MPD3 [Rule 5-16] (WF-MPD). c:pathURI for c:XMLCatalog**

[Rule 5-16](#), c:pathURI for c:XMLCatalog: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-17] (WF-MPD). c:pathURI for c:MPDChangeLog**

[Rule 5-17](#), c:pathURI for c:MPDChangeLog: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog. Provisioning set values of c:pathURI based on relative location of changelog package.

#### **MPD3 [Rule 5-18] (WF-MPD). c:pathURI for c:ReadMe**

[Rule 5-18](#), c:pathURI for c:ReadMe: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-19] (WF-MPD). c:pathURI for c:IEPSampleXMLDocument**

[Rule 5-19](#), c:pathURI for c:IEPSampleXMLDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-1] (WF-MPD). MPD Has an mpd-catalog.xml in its Root Directory**

[Rule 5-1](#), MPD Has an mpd-catalog.xml in its Root Directory: [Section 5.1](#), NIEM MPD Catalog

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-20] (WF-MPD). c:pathURI for c:BusinessRulesArtifact**

[Rule 5-20](#), c:pathURI for c:BusinessRulesArtifact: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-21] (WF-MPD). c:pathURI for c:XMLSchemaDocument**

[Rule 5-21](#), c:pathURI for c:XMLSchemaDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-22] (WF-MPD). c:pathURI for c:ExternalSchemaDocument**

[Rule 5-22](#), c:pathURI for c:ExternalSchemaDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-23] (WF-MPD). c:pathURI for c:ReferenceSchemaDocument**

[Rule 5-23](#), c:pathURI for c:ReferenceSchemaDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-24] (WF-MPD). c:pathURI for c:ExtensionSchemaDocument**

[Rule 5-24](#), c:pathURI for c:ExtensionSchemaDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-25] (WF-MPD). c:pathURI for c:SubsetSchemaDocument**

[Rule 5-25](#), c:pathURI for c:SubsetSchemaDocument: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-26] (WF-MPD). c:pathURI for c:Wantlist**

[Rule 5-26](#), c:pathURI for c:Wantlist: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-27] (WF-MPD). c:pathURI for c:SchematronSchema**

[Rule 5-27](#), c:pathURI for c:SchematronSchema: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-28] (WF-MPD). c:pathURI for c:RelaxNGSchema**

[Rule 5-28](#), c:pathURI for c:RelaxNGSchema: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-29] (WF-MPD). c:pathURI for c:SchemaDocumentSet**

[Rule 5-29](#), c:pathURI for c:SchemaDocumentSet: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-2] (MPD-catalog). MPD Catalog Document Valid to mpd-catalog-3.0.xsd**

[Rule 5-2](#), MPD Catalog Document Valid to mpd-catalog-3.0.xsd: [Section 5.1](#), NIEM MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

#### **MPD3 [Rule 5-30] (WF-MPD). c:pathURI for c:ConstraintSchemaDocumentSet**

[Rule 5-30](#), c:pathURI for c:ConstraintSchemaDocumentSet: [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is realized during provisioning of MPD Catalog.

#### **MPD3 [Rule 5-31] (WF-MPD).**

[Rule 5-31](#): [Section 5.2.4.3](#), URI Scheme for Local MPD Artifacts (c:pathURI)

[English]

Constraint is definitional.

#### **MPD3 [Rule 5-32] (WF-MPD). Resolve MPD URI with Fragment**

[Rule 5-32](#), Resolve MPD URI with Fragment: [Section 5.2.4.5](#), Resolving an MPD URI with a Fragment

[English]

Constraint is definitional.

#### **MPD3 [Rule 5-33] (XML-catalog). XML Catalog uri Value Resolves to Resource**

[Rule 5-33](#), XML Catalog uri Value Resolves to Resource: [Section 5.2.4.7](#), XML Catalog URI

[English]

Constraint is realized during provisioning of XML Catalog.

### **MPD3 [Rule 5-34] (XML-catalog). XML Catalog uri Value Resolves to Resource with Correct Target Namespace**

[Rule 5-34](#), XML Catalog uri Value Resolves to Resource with Correct Target Namespace: [Section 5.2.4.7](#), XML Catalog URI

[English]

Constraint is realized during provisioning of MPD Catalog.

### **MPD3 [Rule 5-35] (IEPD). IEPD Has a Change Log**

[Rule 5-35](#), IEPD Has a Change Log: [Section 5.3.2](#), Change Log for IEPDs

**[OCL] context** ModelPackageDescription **inv:**

```
self.oclAsType(InstanceSpecification).clientDependency->select(d|d.stereotypeBy('MPDChangeLog'))->notEmpty()
```

### **MPD3 [Rule 5-36] (WF-MPD). Readme Describes Purpose, Scope, Business Value, etc.**

[Rule 5-36](#), Readme Describes Purpose, Scope, Business Value, etc.: [Section 5.4](#), ReadMe Artifact

[English]

The readme artifact provisioned includes purpose, scope, business value, exchange information, typical senders/receivers, interactions, and references to other documentation. This information is obtained from certain modeling conventions in the model.

### **MPD3 [Rule 5-37] (IEPD). IEPD Has a ReadMe Artifact**

[Rule 5-37](#), IEPD Has a ReadMe Artifact: [Section 5.4](#), ReadMe Artifact

[English]

Readme artifact content and catalog entry is created during provisioning.

### **MPD3 [Rule 5-38] (MPD-catalog). Conformance Target Identifier**

[Rule 5-38](#), Conformance Target Identifier: [Section 5.6](#), Defining Information Exchange Packages

[English]

Rule enforced during provisioning. The target identifier is synthesized.

### **MPD3 [Rule 5-39] (MPD-catalog). IEP Conformance Target Has a structures:id**

[Rule 5-39](#), IEP Conformance Target Has a structures:id: [Section 5.6](#), Defining Information Exchange Packages

[English]

Rule enforced during provisioning. The target structure:id attribute is synthesized.

### **MPD3 [Rule 5-3] (MPD-catalog). MPD Catalog Extension XML Catalog Document in Root Directory**

[Rule 5-3](#), MPD Catalog Extension XML Catalog Document in Root Directory: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-40] (IEPD). IEPD Declares One or More IEP Conformance Targets**

[Rule 5-40](#), IEPD Declares One or More IEP Conformance Targets: [Section 5.6](#), Defining Information Exchange Packages

**[OCL] context ModelPackageDescription inv:**

```
self.IEPConformanceTarget->notEmpty()
```

### **MPD3 [Rule 5-41] (MPD-catalog).**

[Rule 5-41](#): [Section 5.6.2.3](#), c:ValidityContext

[English]

Rule is definitional.

### **MPD3 [Rule 5-43] (IEP). Validating an XPath Expression**

[Rule 5-43](#), Validating an XPath Expression: [Section 5.6.2.5](#), c:ValidToXPath

[English]

Probably some model adjustment here to make the concept of validToXPath more clear.

### **MPD3 [Rule 5-44] (IEPD). IEPD Has an IEP Sample for Each c:IEPConformanceTarget**

[Rule 5-44](#), IEPD Has an IEP Sample for Each c:IEPConformanceTarget: [Section 5.6.3](#), IEP Sample Instance XML Documents

**[OCL] context ModelPackageDescription inv:**

```
(self.mpdClassCode=NIEM_UML_Profile::Model_Package_Description_Profile::Model  
PackageDescriptionClassName::iepd)  
implies  
self.IEPConformanceTarget.oclAsType(InstanceSpecification).clientDependency-  
>exists(d|d.stereotypeBy('IEPSampleXMLDocument'))
```

### **MPD3 [Rule 5-46] (IEPD). IEPD Has Conformance Assertion**

[Rule 5-46](#), IEPD Has Conformance Assertion: [Section 5.7](#), Conformance Assertion

[English]

Rule satisfied during provisioning of PSM from PIM.

### **MPD3 [Rule 5-4] (MPD-catalog). MPD Catalog Extension XML Catalog Document Name Is mpd-catalog-extension-xml-catalog.xml**

[Rule 5-4](#), MPD Catalog Extension XML Catalog Document Name Is mpd-catalog-extension-xml-catalog.xml: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-5] (MPD-catalog). MPD Catalog Extension XML Catalog Document Resolves Namespaces to URIs**

[Rule 5-5](#), MPD Catalog Extension XML Catalog Document Resolves Namespaces to URIs: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-6] (MPD-catalog). MPD Catalog Extension Schema Document Conforms to NDR Extension Rules**

[Rule 5-6](#), MPD Catalog Extension Schema Document Conforms to NDR Extension Rules: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-7] (MPD-catalog). MPD Catalog Schema and Its Extensions Conform to NDR Schema Set Rules**

[Rule 5-7](#), MPD Catalog Schema and Its Extensions Conform to NDR Schema Set Rules: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-8] (MPD-catalog). MPD Schema Document Extension Support Schemas Are Supersets of Spec Subsets**

[Rule 5-8](#), MPD Schema Document Extension Support Schemas Are Supersets of Spec Subsets: [Section 5.1.2](#), Extending an MPD Catalog

[English]

Concept and constraint not implemented in NIEM-UML.

### **MPD3 [Rule 5-9] (WF-MPD). MPD Class Determined by Conformance Target Identifier in c:mpdClassURIList**

[Rule 5-9](#), MPD Class Determined by Conformance Target Identifier in c:mpdClassURIList: [Section 5.2.2](#), MPD Class (c:mpdClassURIList)

[English]

Constraint realized during provisioning of MPD Catalog.

### **MPD3 [Rule 6-1] (WF-MPD). Wantlist Location**

[Rule 6-1](#), Wantlist Location: [Section 6.1](#), NIEM Wantlist

[English]

Location of WantList is set during provisioning to be compliant with the MPD rule.

### **MPD3 [Rule 7-1] (WF-MPD). MPD Is a ZIP File**

[Rule 7-1](#), MPD Is a ZIP File: [Section 7](#), Organization, Packaging, and Other Criteria

[English]

Rule enforcement is provided by provisioning, which produces the zip file.

### **MPD3 [Rule 7-2] (WF-MPD). XSD and XML Documents Conform to Applicable NDR Conformance Targets**

[Rule 7-2](#), XSD and XML Documents Conform to Applicable NDR Conformance Targets: [Section 7](#), Organization, Packaging, and Other Criteria

[English]

Rule enforcement is provided by provisioning and NDR/MPD rules expressed in OCL and applied to the NIEM-UML model.

### **MPD3 [Rule 7-3] (WF-MPD). MPD Archive Uncompresses to a Single Root Directory**

[Rule 7-3](#), MPD Archive Uncompresses to a Single Root Directory: [Section 7](#), Organization, Packaging, and Other Criteria

[English]

Rule enforcement is provided by provisioning.

### **MPD3 [Rule 7-5] (IEPD). IEPD File Name Syntax**

[Rule 7-5](#), IEPD File Name Syntax: [Section 7.2](#), IEPD File Name Syntax

[English]

Packaging constraints are resolved by PSM-MPD transformations.

### **MPD3 [Rule 7-6] (WF-MPD). MPD Reference to Resource Uses Common URI Scheme**

[Rule 7-6](#), MPD Reference to Resource Uses Common URI Scheme: [Section 7.3](#), Artifact Links to Other Resources

[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.

### **MPD3 [Rule 7-7] (WF-MPD). IEPD Completeness**

[Rule 7-7](#), IEPD Completeness: [Section 7.4](#), IEPD Completeness

[English]

This constraint is resolved by PSM-MPD transformations.

### **MPD3 [Rule 7-8] (WF-MPD). MPD External Schema Documents Are Local Resources**

[Rule 7-8](#), MPD External Schema Documents Are Local Resources: [Section 7.4](#), IEPD Completeness

[English]

This constraint is resolved by the NIEM-UML Model, which requires all InformationModels to be defined, and PSM-MPD transformations which enforce schemaLocations to be local.

### **MPD3 [Rule 7-9] (WF-MPD). Key MPD Resources Are Local Resources**

[Rule 7-9](#), Key MPD Resources Are Local Resources: [Section 7.4](#), IEPD Completeness

[English]

This constraint is resolved by provisioning; all generated artifacts are local to the MPD and all references between them are relative.

## **8.5.37 <Artifact> OrganizationType**

### **Description**

A data type for a body of people organized for a particular purpose.

### **Generalization**

[EntityRepresentation](#)

### **Properties**

#### **OrganizationPrimaryContactInformation : ContactInformationType [0..\*]**

A preferred means of contacting an organization.

## **8.5.38 <Artifact> PersonType**

### **Description**

Represents an AuthoritativeSource for the MPD corresponding to a niem-core:PersonType. In this case, an InstanceSpecification of PersonType is mapped to the MPD Catalog element c:MPDInformationType/c:AuthoritativeSource/niem-core:EntityPerson (whose type is niem-core:PersonType).

An InstanceSpecification of PersonType may also represent a ContactEntity within a ContactInformationType. In this case, the instance of PersonType is mapped to the MPD Catalog element .../niem-core:EntityOrganization/niem-core:OrganizationPrimaryContactInformation/niem-core:ContactEntity/niem-core:EntityPerson (whose type is niem-core:PersonType).

An InstanceSpecification of PersonType may also represent a ContactResponder within a ContactInformationType. In this case, the instance of PersonType is mapped to the MPD Catalog element .../niem-core:EntityOrganization/niem-core:OrganizationPrimaryContactInformation/niem-core:ContactResponder/niem-core:EntityPerson (whose type is niem-core:PersonType).

### **Generalization**

[EntityRepresentation](#)

## **8.5.39 <Artifact> QualifiedNamesType**

### **Description**

A data type for a set of qualified names.

### **Generalization**

[ValidityConstraintWithContextType](#)

## **8.5.40 <Artifact> RelaxNGValidationType**

### **Description**

A data type for a RelaxNG validation constraint, indicating a RelaxNG schema document against which an artifact may be validated, as well as a description of the validation roots for assessment of validity.

## **Generalization**

[ValidityConstraintType](#)

### **8.5.41 <Artifact> SchemaDocumentSet**

#### **Description**

An MPD artifact set that may include subset schema documents, extension and external schema documents, and other supporting artifacts.

#### **Generalization**

[ArtifactOrArtifactSet](#) [SchemaDocumentSetType](#)

### **8.5.42 <Artifact> SchemaDocumentSetType**

#### **Description**

A data type for an MPD artifact set that may include subset schema documents, extension schema documents, and external schema documents or constraint schema documents.

#### **Generalization**

[FileSetType](#)

#### **Constraints**

**MPD3 [Rule 7-4] (MPD-catalog). Constraint on Elements of Type c:SchemaDocumentSetType**

[Rule 7-4](#), Constraint on Elements of Type c : SchemaDocumentSetType: [Section 7.1.1](#), Constraint on Elements of Type c : SchemaDocumentSetType

**[OCL] context SchemaDocumentSetType inv:**

```
self.oclAsType(InstanceSpecification).clientDependency->select(d|d.stereotypeBy('XMLSchemaDocument') or  
d.stereotypeBy('XMLCatalog'))->notEmpty()
```

### **8.5.43 <Artifact> SchematronValidationType**

#### **Description**

A data type for a Schematron validation constraint, indicating a Schematron schema document against which an artifact may be validated as well as a description of the validation roots for assessment of validity.

#### **Generalization**

[ValidityConstraintType](#)

## **8.5.44 <Artifact> TextRuleType**

### **Description**

A data type for a rule drafted in a human language.

### **Generalization**

[ValidityConstraintType](#)

### **Properties**

#### **RuleText : String [1]**

A rule written in a human language.

## **8.5.45 <Artifact> ValidityConstraintType**

### **Description**

A data concept for a rule or instructions for validating an IEP candidate.

### **Generalization**

[ValidityConstraintWithContextType](#)

## **8.5.46 <Artifact> ValidityConstraintWithContextType**

### **Description**

A data concept for a rule or instructions for validating an IEP candidate (XML document) using some context within that XML document.

### **Generalization**

[DescribedType](#)

## **8.5.47 <Artifact> ValidityContextType**

### **Description**

A data type for a rule or instructions for validating an IEP candidate within context defined by an XPath expression.

### **Generalization**

[ValidityConstraintWithContextType](#)

### **Properties**

#### **ValidityConstraint : ValidityConstraintType [1..\*]**

A data concept for a rule or instructions for validating an IEP candidate.

#### **xPathText : String [1]**

An XPath expression.

## 8.5.48 <Artifact> XMLSchemaType

### Description

A data type for a validity constraint that indicates an XML Schema against which an artifact may be validated, or which can be used for other purposes. c:XMLSchemaDocument identifies the root or starting XML schema document.

### Generalization

[ValidityConstraintType](#)

## 8.5.49 <Artifact> XPathType

### Description

A data type for an XPath expression.

### Generalization

[ValidityConstraintType](#)

### Properties

**xPathText : String [1]**

An XPath expression.

## 8.5.50 <Enumeration> ChangeCodeSimpleType

### Description

Purpose of change.

### Literals

**new\_requirement**

**bug\_fix**

**refactoring**

**harmonization**

**general\_improvement**

## **8.5.51 <Enumeration> ModelPackageDescriptionClassCode**

### **Description**

A specified classification (type or kind) of the MPD.

The MPD specification applies to all NIEM model package descriptions (MPDs). Currently, MPDs include the following:

- A NIEM information exchange package documentation (IEPD) that defines a NIEM data exchange.
- A NIEM release (including a major, minor, or micro release) [as defined in the NIEM High-Level Version Architecture 1.0].
- A NIEM domain update [as described in NIEM Domain Update Specification 1.0]. (Note these are NOT the same as a NIEM domain schema document that is a part of a NIEM release).
- A NIEM core update to a NIEM release.
- A NIEM Enterprise Information Exchange Model (EIEM) on which one or more IEPDs can be based.

Of these kinds of MPDs, the only kind which is formally specified in NIEM-3 is an IEPD. The NIEM-3 UML Models all kinds of MPD, and the kind is defined as the EnumerationLiterals of this ModelPackageDescriptionClassCode Enumeration.

The kind of MPD is reflected in the MPD Catalog c:mpdClassURLList attribute. That attribute will be provisioned with the appropriate list of URIs based on the value of this ModelPackageDescriptionClassCode Enumeration.

### **Literals**

#### **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.

#### **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.

## **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)).

## **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.

## **8.5.52 <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. Reference [Section 5.2.4.4](#) and [Appendix A](#) of [NIEM MPD].

### **Literals**

#### **updates**

A relationshipCode value for indicating that this MPD is an incremental update to the referenced resource. Used by a core or domain update to identify the domain schema in a NIEM release being incrementally updated (not replaced).

#### **conforms\_to**

A relationshipCode value for indicating that this MPD conforms to the referenced specification or standard.

#### **version\_of**

A relationshipCode value for indicating that this MPD is a different version of the referenced MPD. 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.

#### **specializes**

A relationshipCode value for indicating that this MPD is a specialization of the referenced MPD. This value is the inverse of generalizes.

#### **generalizes**

A relationshipCode value for indicating that this MPD is a generalization of the referenced MPD. This value is the inverse of specializes.

#### **supersedes**

A relationshipCode value for indicating that this MPD replaces the referenced MPD.

#### **deprecates**

A relationshipCode value for indicating that content in this MPD is preferred over content in the referenced MPD; and at some time in the future will supersede the referenced MPD.

**adapts**

A relationshipCode value for indicating that this MPD is an adaptation of the referenced MPD.

**derives\_from**

A relationshipCode value for indicating that this MPD has been derived from another; used to indicate an IEPD is derived from an EIEM (may have other uses as well).

# 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 A). 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

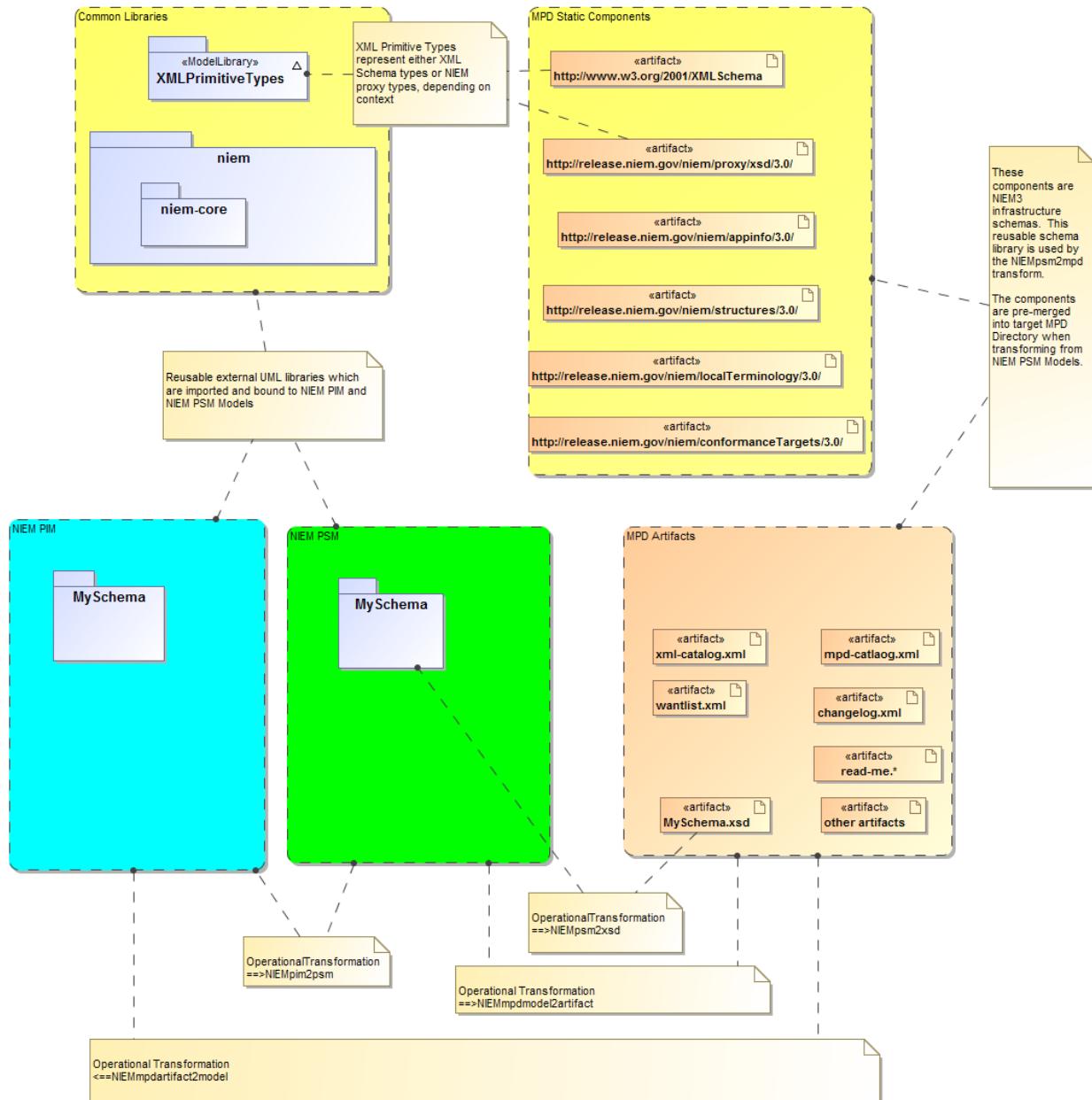
The transformations referenced in this clause are intended to constitute provisioning process that enables representation of MPD artifacts as UML Models or in their native NIEM-conformant XML format. The overall provisioning process is illustrated in Figure 9-1. The focus of this clause is to transform 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]. 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://release.niem.gov/niem/structures/3.0/>”. 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://release.niem.gov/niem/proxy/xsd/3.0/>”. This is the NIEM “proxy” schema and is used to “wrap” XML Primitive Types with XSDComplexTypeDefinitions in order to provide attributes carrying metadata.
- *appinfo*. The NIEM NDR Schema whose target namespace is “<http://release.niem.gov/niem/appinfo/3.0/>”. The [appinfo namespace](#) defines attributes which provide additional semantics for components built by NIEM-conformant schemas.
- *ct*. The NIEM NDR Schema whose target namespace is “<http://release.niem.gov/niem/conformanceTargets/3.0/>”. This schema defines the NIEM 3 Conformance Targets Attribute.
- *term*. The NIEM NDR Schema whose target namespace is “<http://release.niem.gov/niem/LocalTerminology/3.0/>”. This schema defines the components used to define NIEM 3 Local Terms.
- *XML Schema*. A representation of 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.



**Figure 9-1 NIEM Provisioning Context**

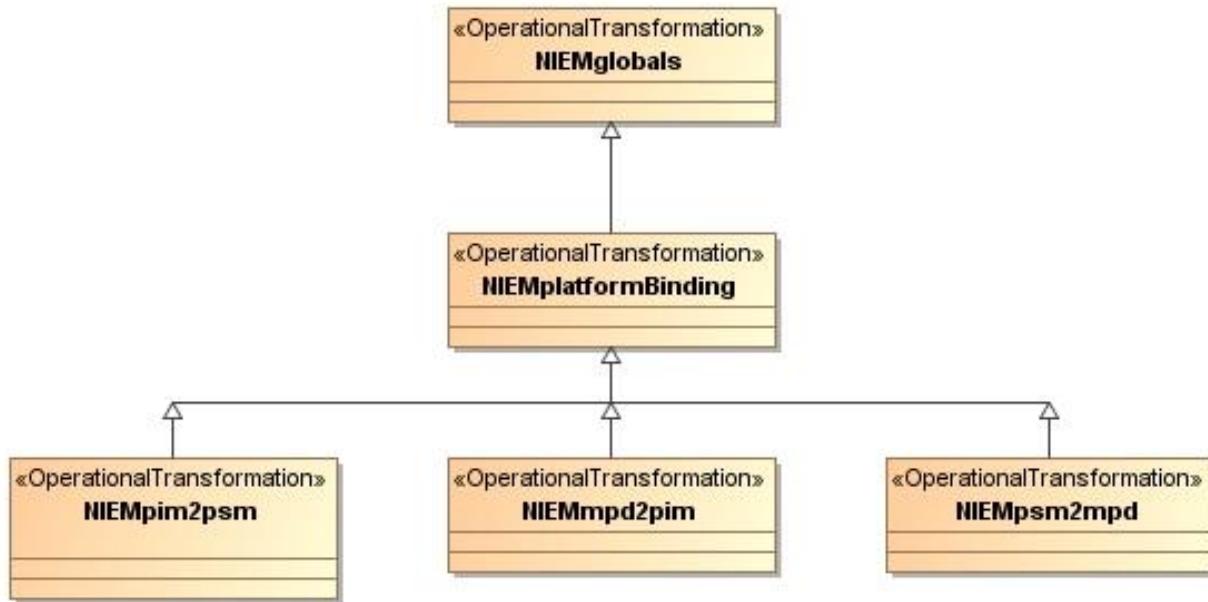
The transformations referenced in this clause include:

- **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:

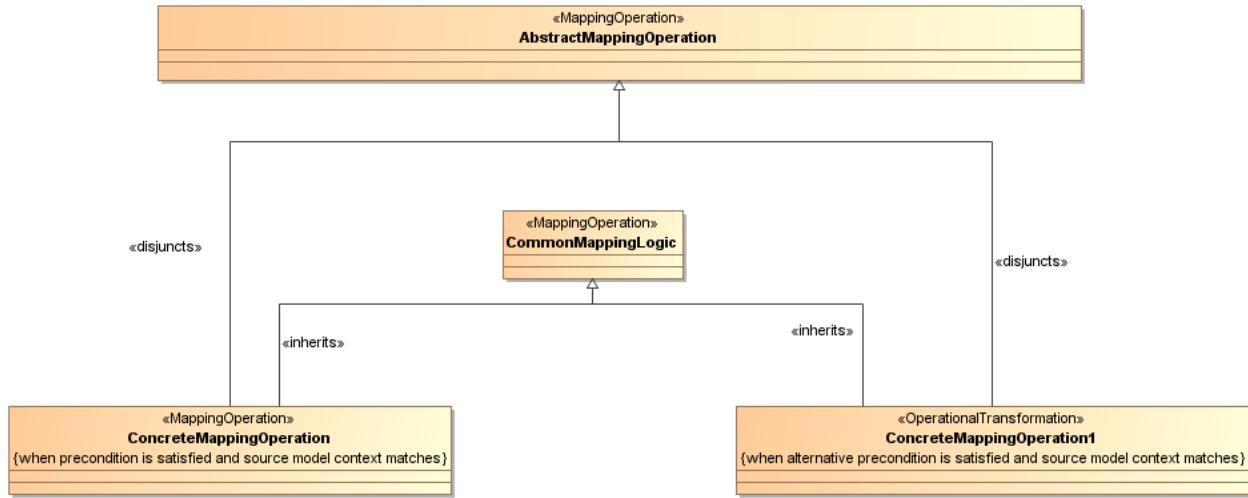
- *NIEMplatformBinding*. A set of platform-specific operations. For the purposes of this specification, these are defined as abstract operations.
- *NIEMglobals*. A set of variables initialized at the beginning of the transformation, including references to Profiles and Stereotypes from NIEM-UML, and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



**Figure 9-2 NIEM Transformations**

### 9.1.2 Transformation Notation

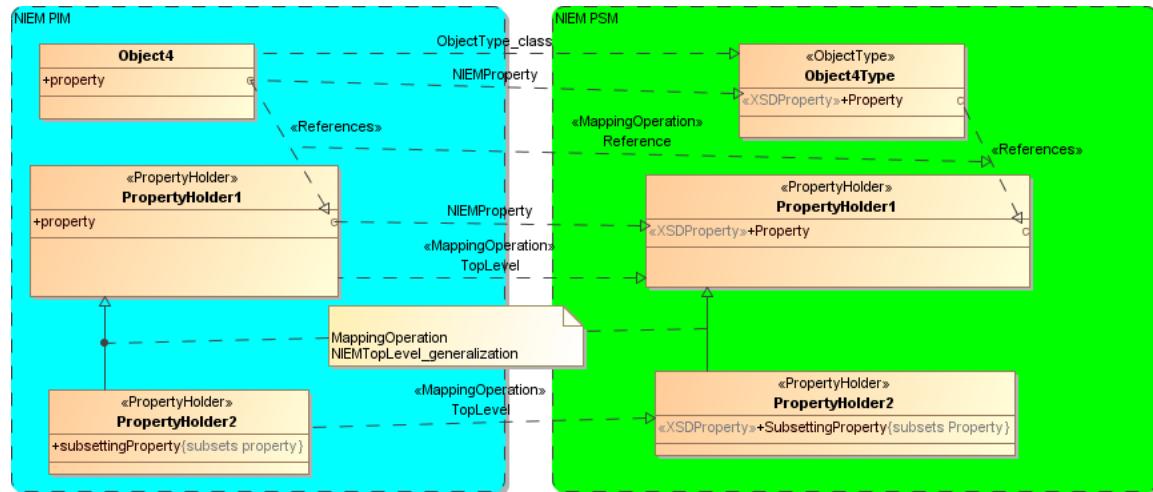
Reuse and composition facilities are associated with QVT mapping operations. Disjunction enables selecting, among the set of disjunctive mappings, the first that satisfies the when clause and then invoking it. For the NIEM transformations, disjunction is used to identify a concrete MappingOperation to 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.



**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 which 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 which have platform variations include those below. Note that Stdlib is the QVT Standard Library.

- *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. The platform-specific operation sets the lower bound of the multiplicity interval.
- *abstract helper UML::MultiplicityElement::setUpper(upper:Integer);*

Context is a UML Multiplicity Element. The platform-specific operation sets 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. The operation applies the specified stereotype to this element and returns an instance of the applied stereotype.

## Global Properties

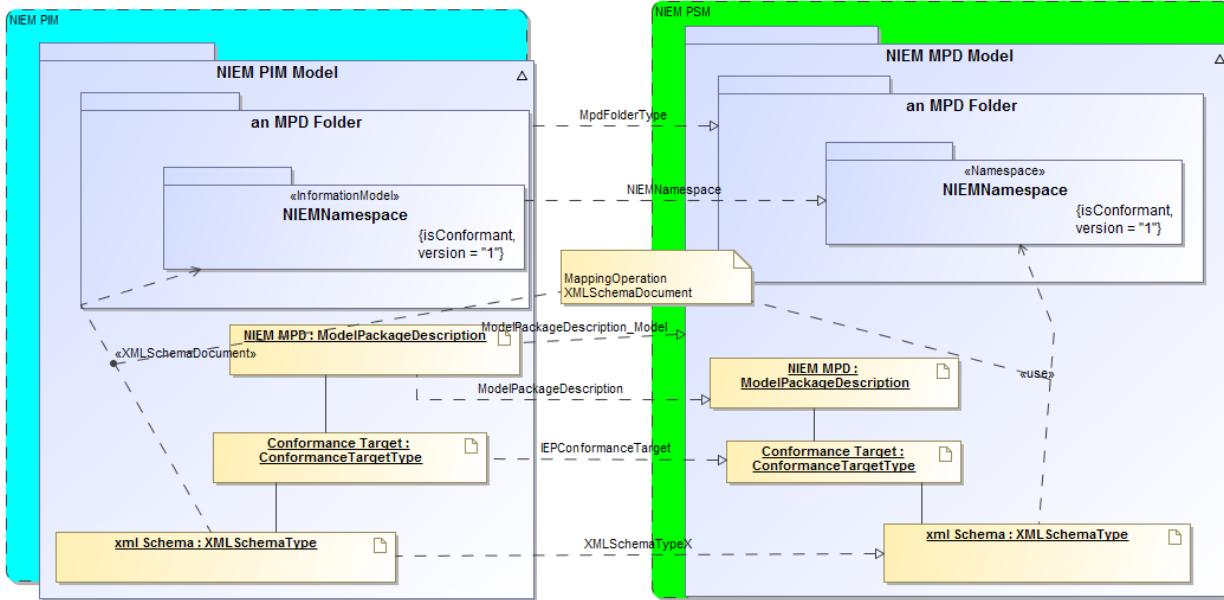
Property names are shared between the transformations. Properties may be one of the following kinds, depending upon the name syntax:

- *<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 which 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 an Instance of a NIEM PIM ModelPackageDescription Artifact. 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 an Instance of a ModelPackageDescription Artifact. 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 «XMLSchemaDocument» Usages from the NIEM PIM ModelPackageDescription Artifact and its nested Artifacts.
- 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.
- Non-NIEM\_PIM\_Profile Stereotype applications from the NIEM PIM are cloned and applied to their mapped counterparts in the NIEM PSM.



**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.
- NIEM PIM Enumeration which has literals, is not subtyped, is not a union, is not a union member, is not a list member, is not restricted, is not the type of an attribute, and is otherwise in a conformance namespace is mapped to a Class, a «XSDSimpleContent» Realization, and an Enumeration which has the same owned Literals as the PIM Enumeration. Comments and «Deprecated» elements are cloned to both Class and Enumeration. Class name is coerced to \*CodeType and Enumeration name is coerced to \*CodeSimpleType.
- 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.
- Augmentation properties subset corresponding augmentation points.

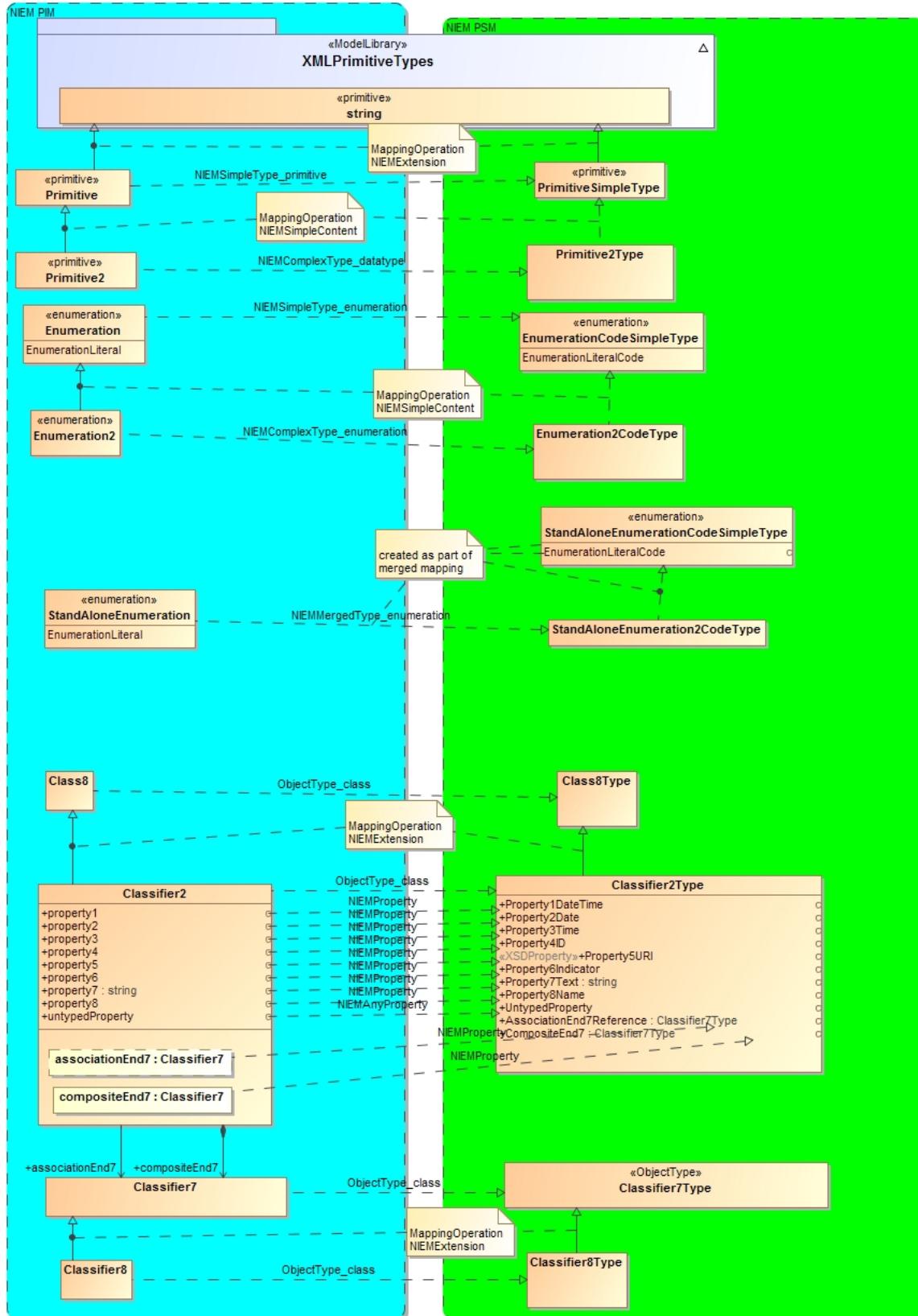
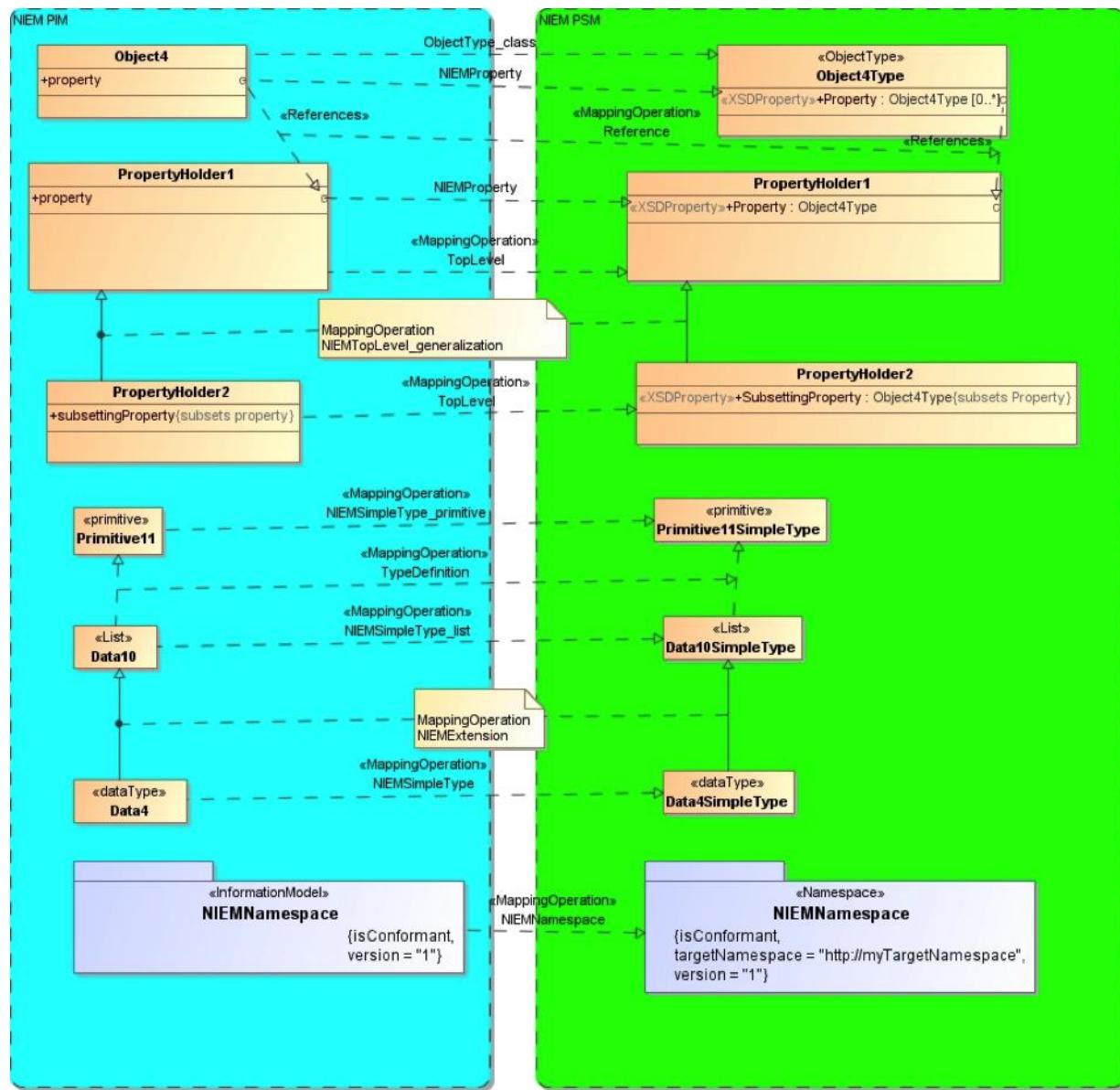


Figure 9-6 NIEM PIM to NIEM PSM - PSM Profile Mapping Overview

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”).

- (Abstract) AugmentationPoint Elements are added, if necessary, to Augmentable Types and a «References» made to their top-level declaration in a «PropertyHolder». Any «Augments» Realization whose supplier is the Augmentable Type has an Augmentation Element created (whose type is the client of the «Augments» Realization) which subsets the AugmentationPoint Element (in the «PropertyHolder»).

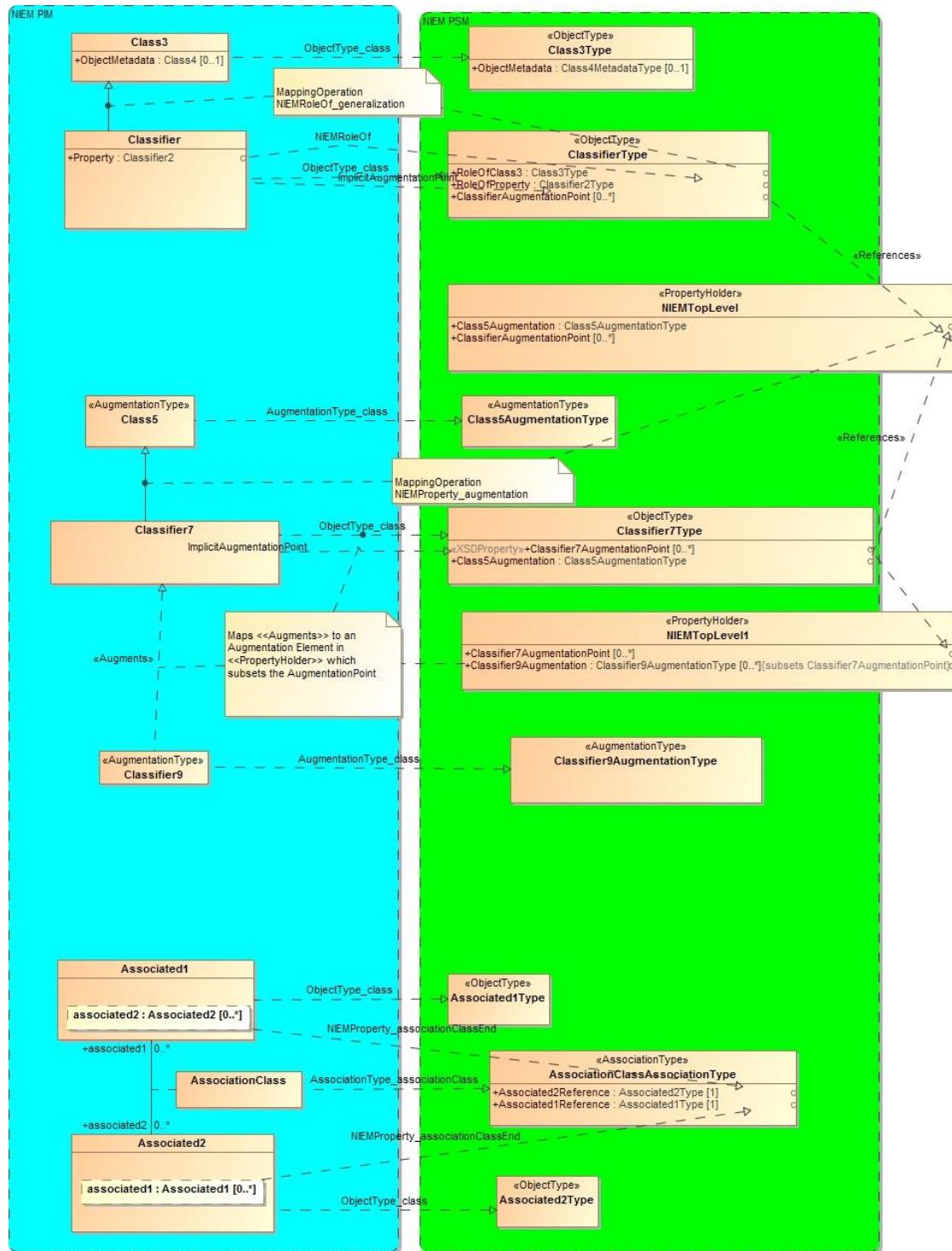
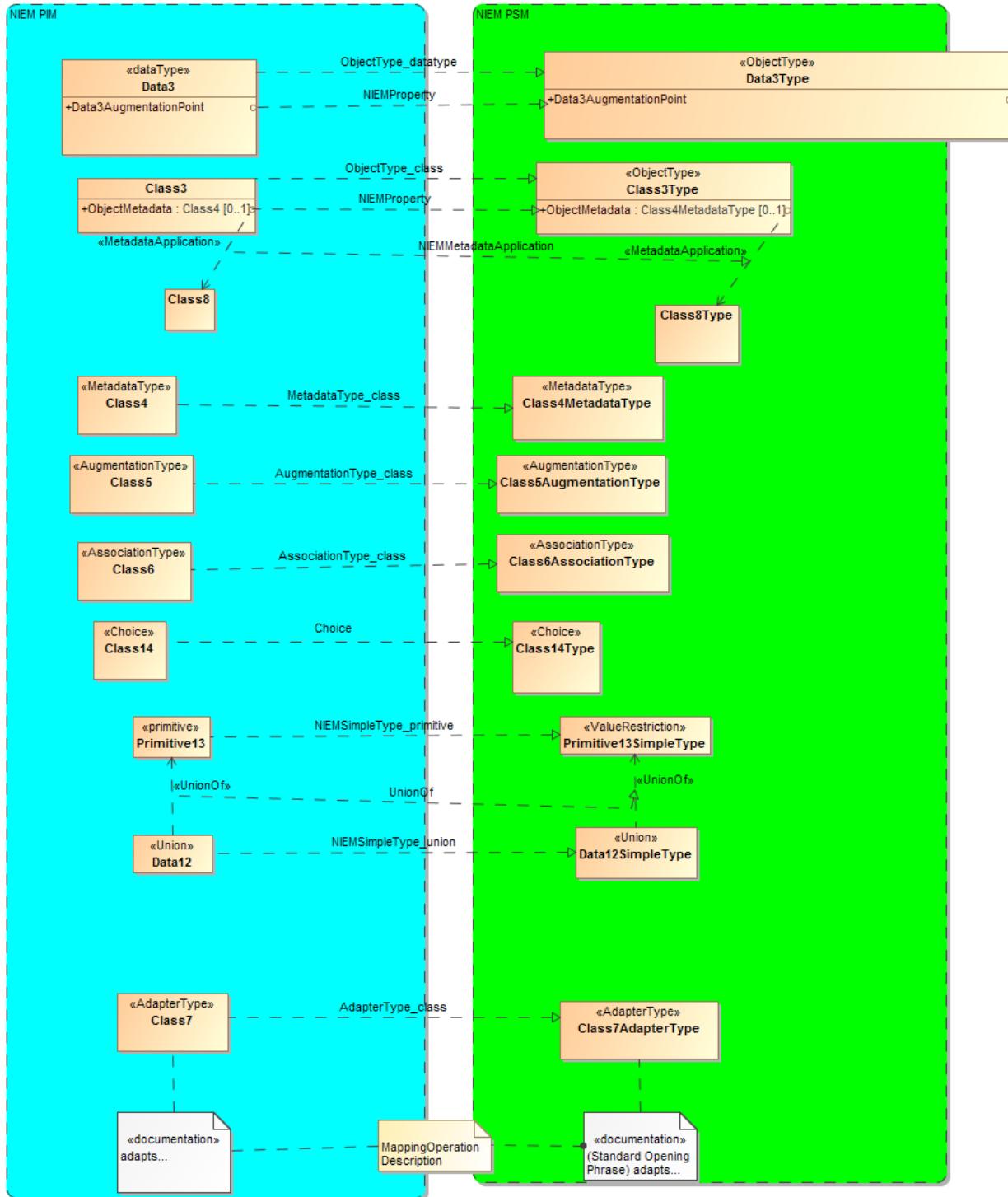


Figure 9-8 NIEM PIM to NIEM PSM - PIM Profile Mapping Overview

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.



**Figure 9-9 NIEM PIM to NIEM PSM - Common Profile Mapping Overview**

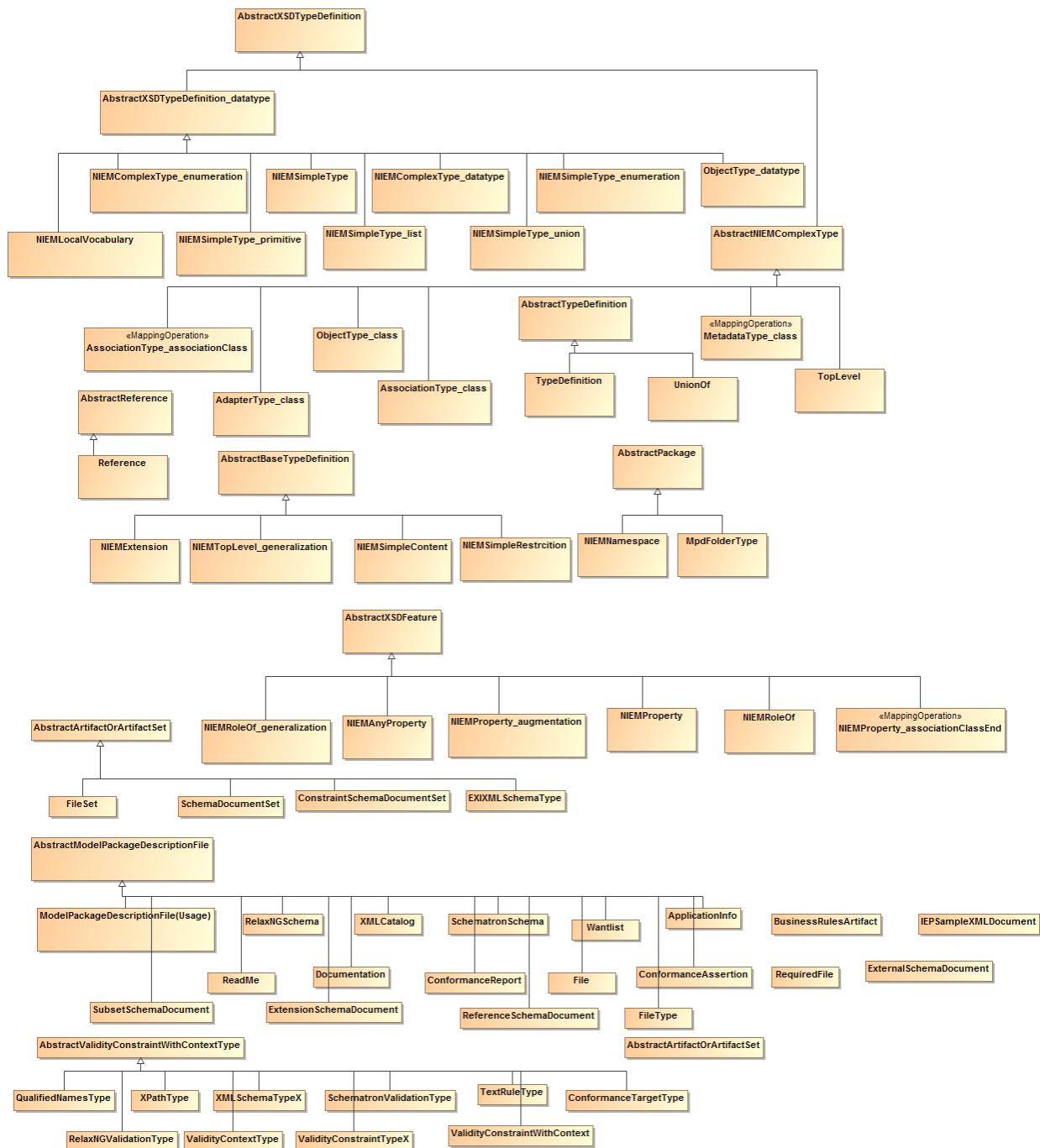
- An unstereotyped AssociationClass within NIEM PIM maps to a Class stereotyped as AssociationType in target NIEM PSM, with some cardinality adjustments.
- Use of Generalizations related to PrimitiveTypes/SimpleTypes may map to Dependencies in target NIEM PSM.

- 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 which 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.



**Figure 9-10 NIEM PIM to NIEM PSM Disjunction**

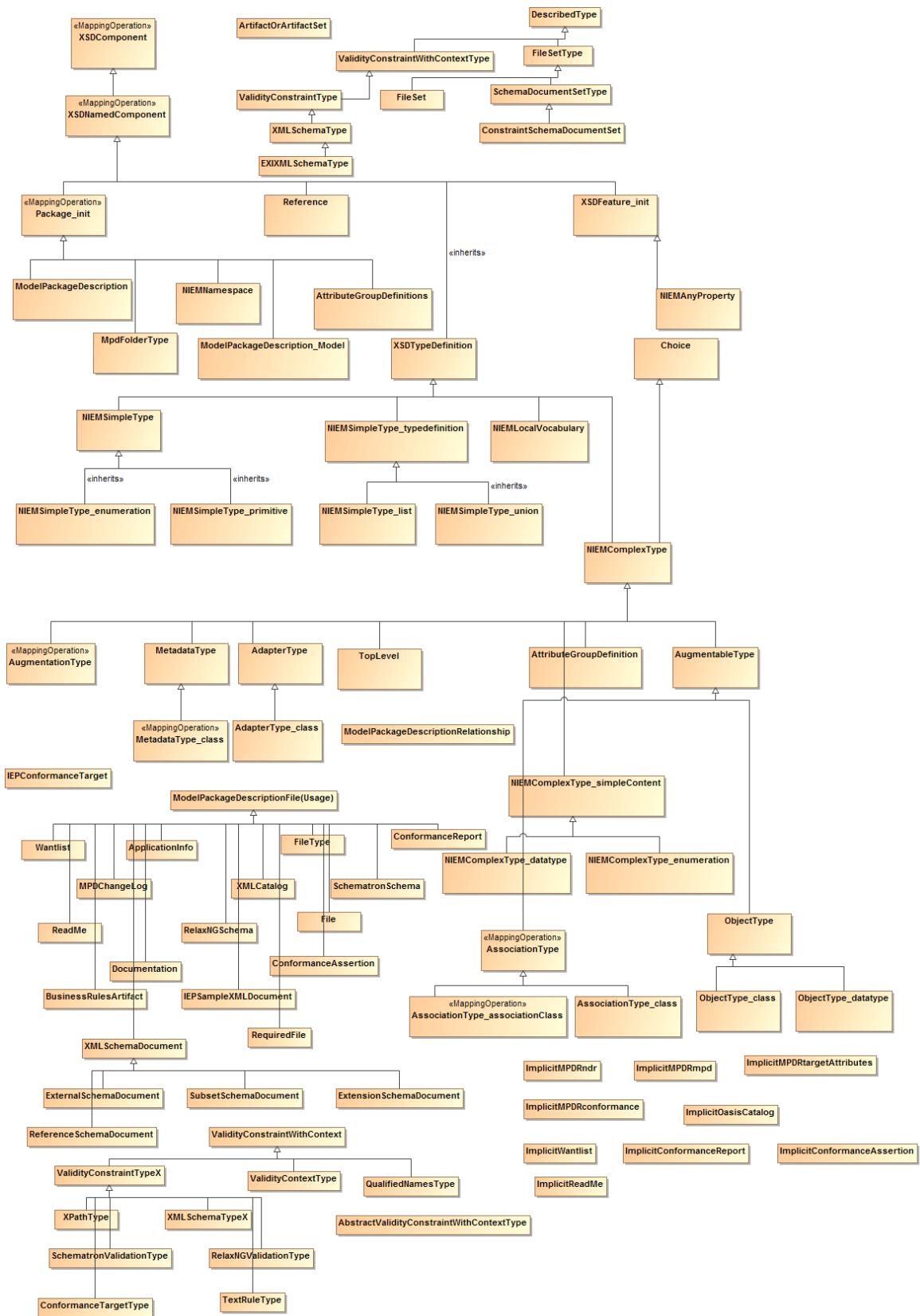


Figure 9-11 NIEM PIM to NIEM PSM Inheritance (1)

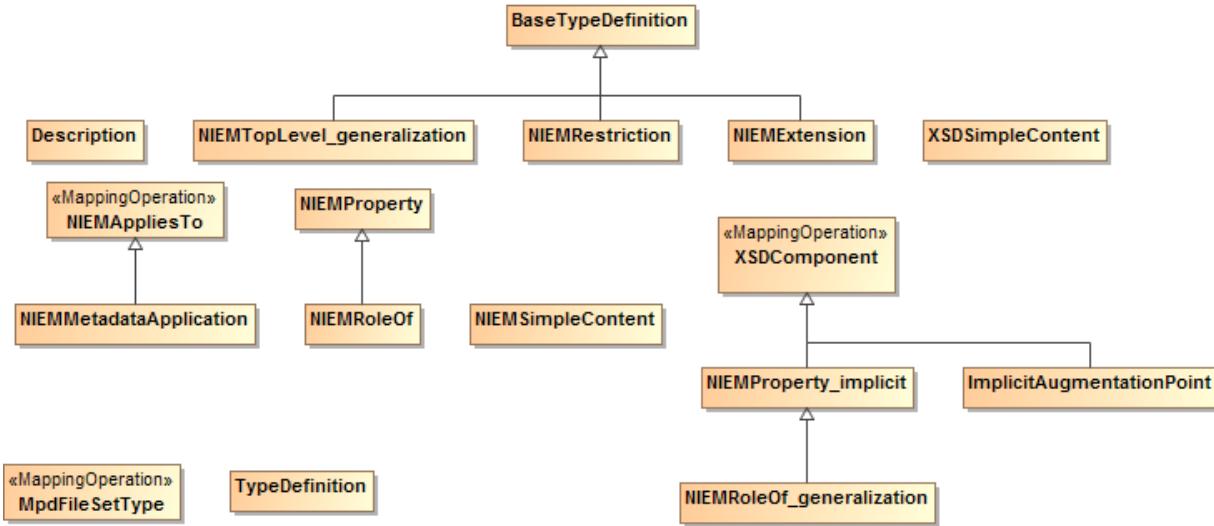


Figure 9-12 NIEM PIM to NIEM PSM Inheritance (2)

### 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 either NIEM-defined schema attributes or 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 as a schema attribute or as embedded components within 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” which contains an `xs:documentation` (DOM) Element. A UML «Documentation» Comment body is mapped to the textual value of the `xs:documentation` Element.
- An XSDAnnotation also has a property “applicationinformation” which contains an `xs:appinfo` (DOM) Element. The `xs:appinfo` Element contains (DOM) Elements defined by the NDR rules.
- `term:LocalTerm` is a NIEM defined element which has 4 attributes: term, literal, definition, and sourceURIs. The containing `xs:appinfo` element is mapped from a UML «LocalVocabulary». Each `term:LocalTerm` is mapped from «LocalTerm». The literal, definition, and sourceURIs attributes are mapped from corresponding tags in the «LocalTerm». The term attribute is mapped from the name of the underlying «LocalTerm» EnumerationLiteral.
- `appinfo:externalAdapterTypeIndicator` is a NIEM-defined (DOM) Attribute. The externalAdapterTypeIndicator attribute indicates that a complex type is an external adapter type. An external adapter type is composed of elements and attributes from non-NIEM-conformant schemas. A UML «AdapterType» is mapped to an `appinfo:externalAdapterTypeIndicator` with value “true”.
- `appinfo:externalImportIndicator` is a NIEM-defined (DOM) Attribute. The externalImportIndicator attribute is true if and only if a namespace identified via `xs:import` is expected to be non-conformant. A use

of an UML «Namespace» with isConformant=false implicitly results in an `xs:import` including an `appinfo:externalImportIndicator` with value “true”.

- `appinfo:deprecated` is a NIEM-defined (DOM) Attribute. The Deprecated attribute provides a method for identifying schema components as being deprecated. A deprecated component is one that is provided, but the use of which is not recommended. A UML «Deprecated» is mapped to an `appinfo:deprecated` attribute with value “true”.
- `appinfo:appliesToTypes` is a NIEM-defined (DOM) Attribute. The appliesToTypes attribute appears on the element declaration of a metadata element. It indicates a set of types to which the metadata element may be applied. The metadata element will also be applicable to any type that is derived from a listed type. A UML «MetadataApplication» whose supplier is a Type is mapped to an `appinfo:appliesToTypes` attribute.
- `appinfo:appliesToElements` is a NIEM-defined (DOM) Attribute. The appliesToElements attribute appears on the element declaration of a metadata element. It indicates a set of elements to which the metadata element may be applied. The metadata element will also be applicable to any element that is in the substitution group of a listed element. A UML «MetadataApplication» whose supplier is a Property is mapped to an `appinfo:appliesToElements` attribute.
- `ct:conformanceTargets` is a NIEM-defined (DOM) Attribute. The conformanceTargets attribute appears on the schema declaration of a NIEM Schema and defines the effective conformance target identifier for the document. The value of the conformanceTargets attribute is a list of URIs which will include those defined by the «Namespace» conformanceTargets tag.

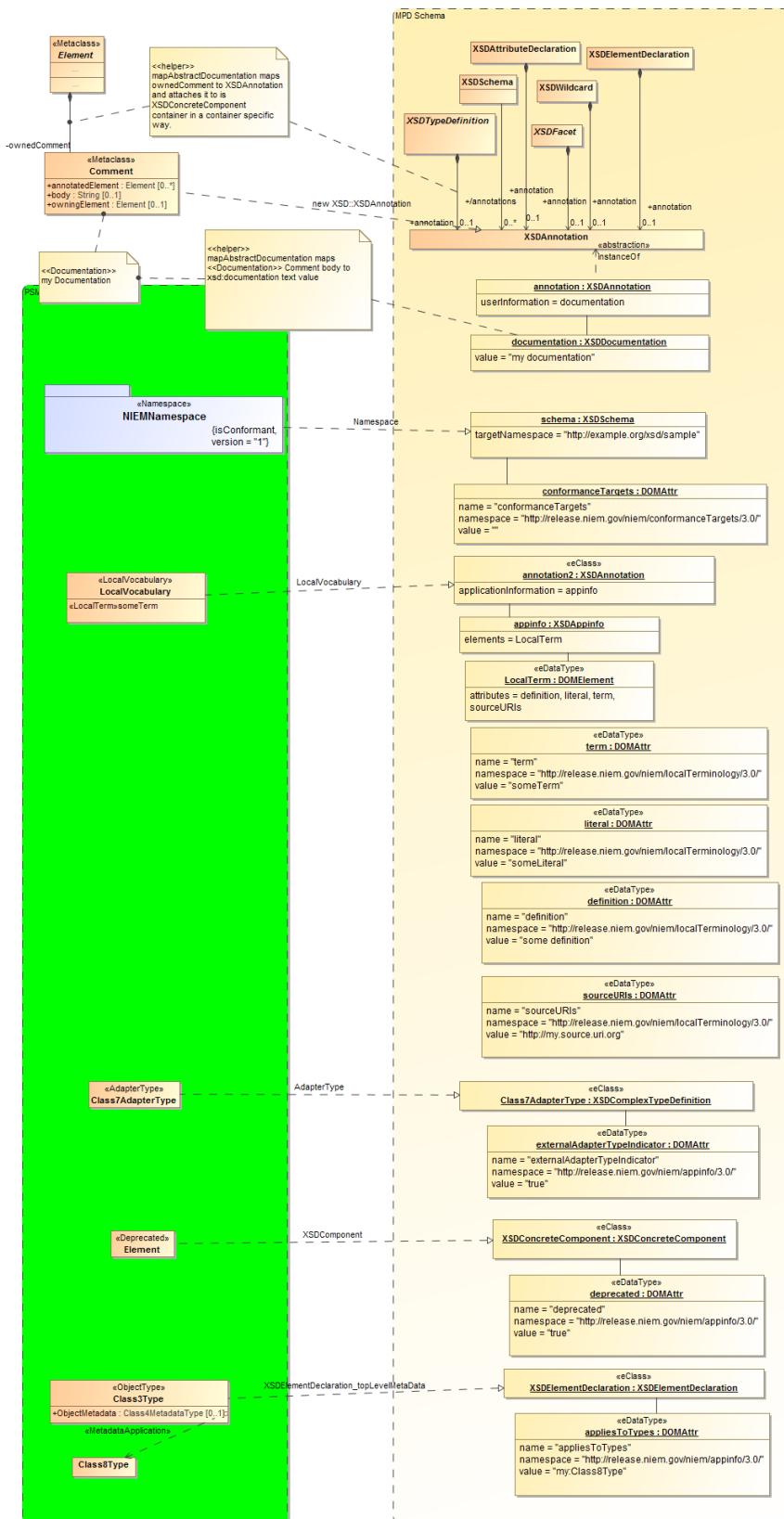
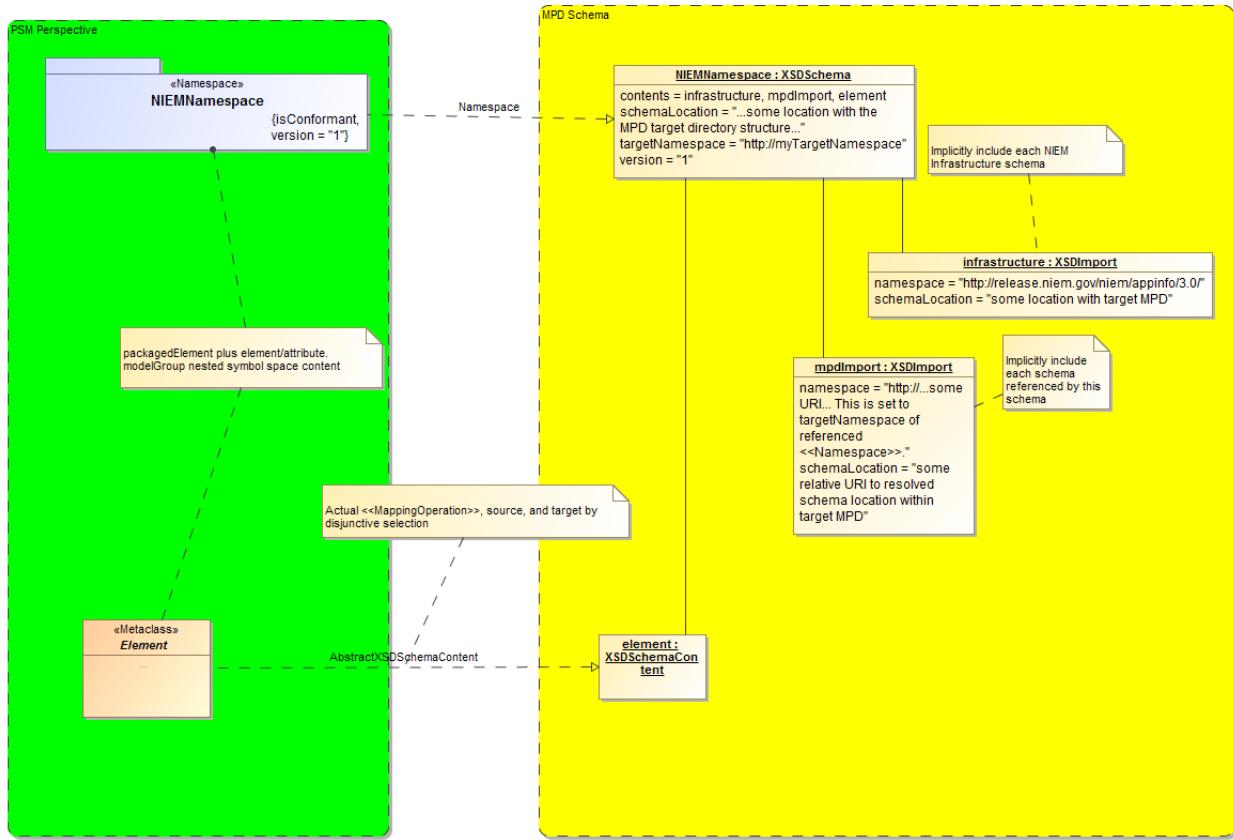


Figure 9-13 NIEM PSM to MPD Schema Artifacts – Annotation Mapping Overview

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.

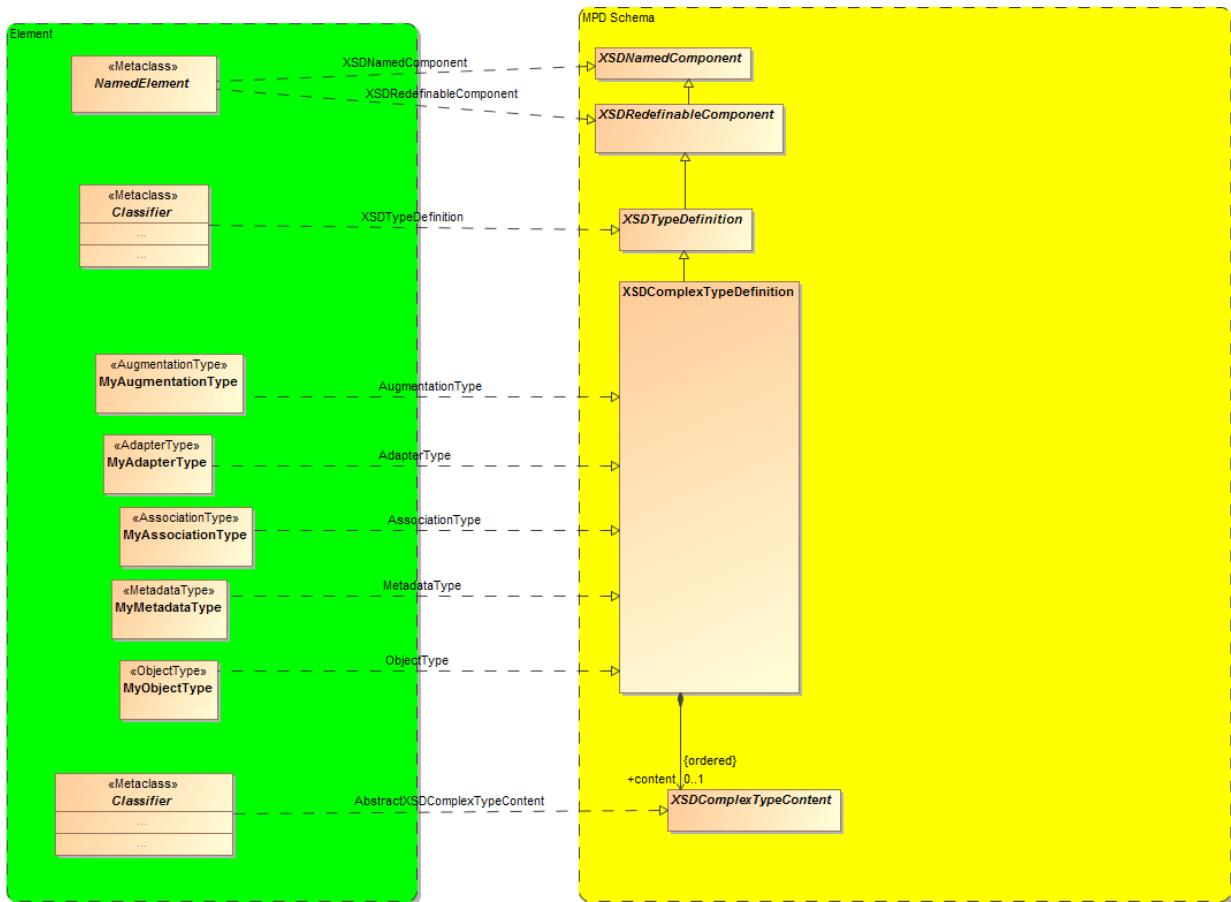


**Figure 9-14 NIEM PSM to MPD Schema Artifacts – «Namespace» Mapping Overview**

«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.

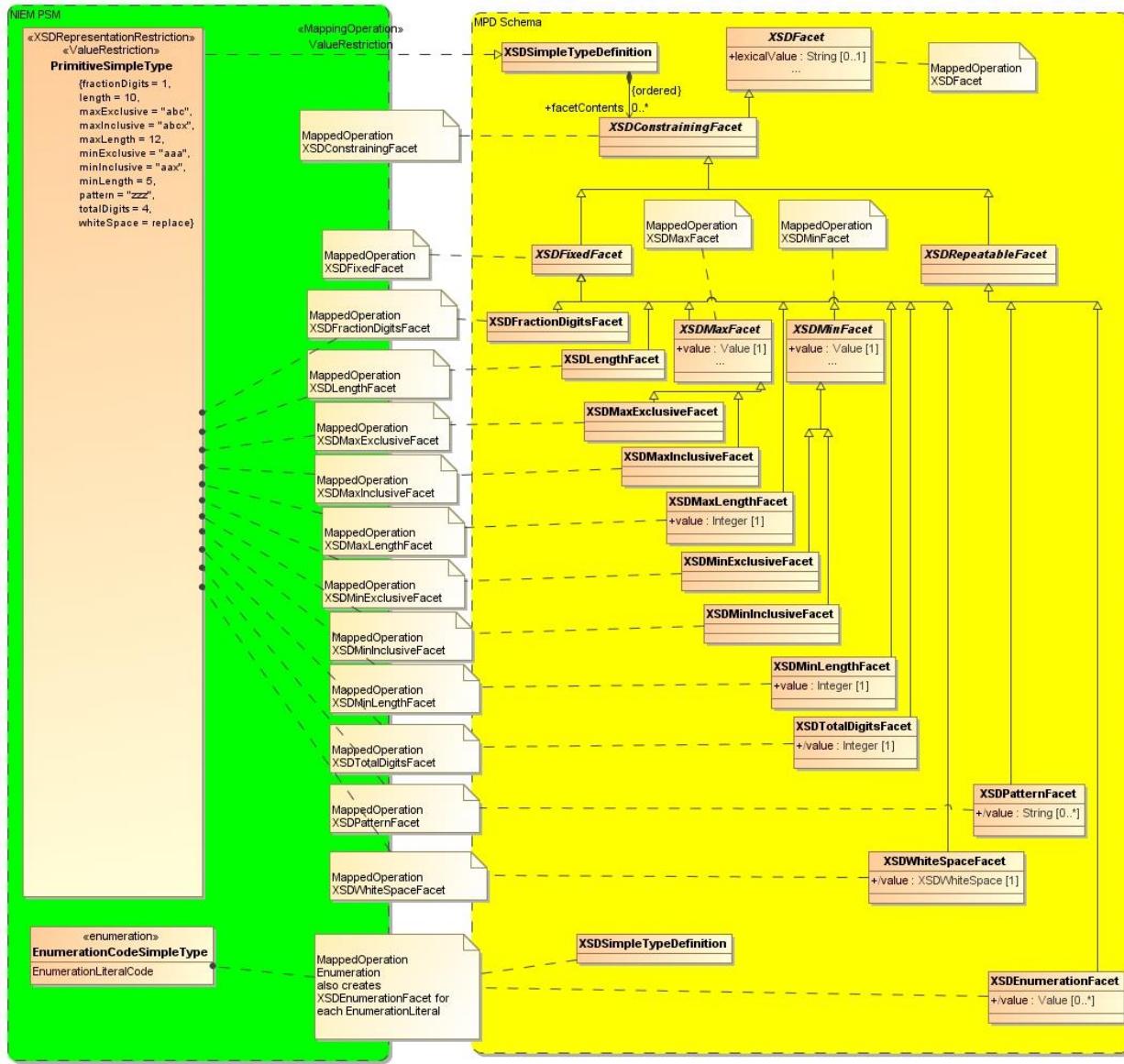
- Inheritance in the NIEM-UML model may be specified as a Generalization or as a «Restriction» Realization. In the case of «Restriction» the derivationMethod of the target XSDComplexTypeDefinition is set to *restriction*. In all other cases, the derivationMethod is set to *extension*.
- When there is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

- OwnedAttributes of «NIEMType» which 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.



**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.



**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» which 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» which 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 relationships between «PropertyHolder»s (which may be required to satisfy *subsetsProperty* reference semantics) are not mapped.

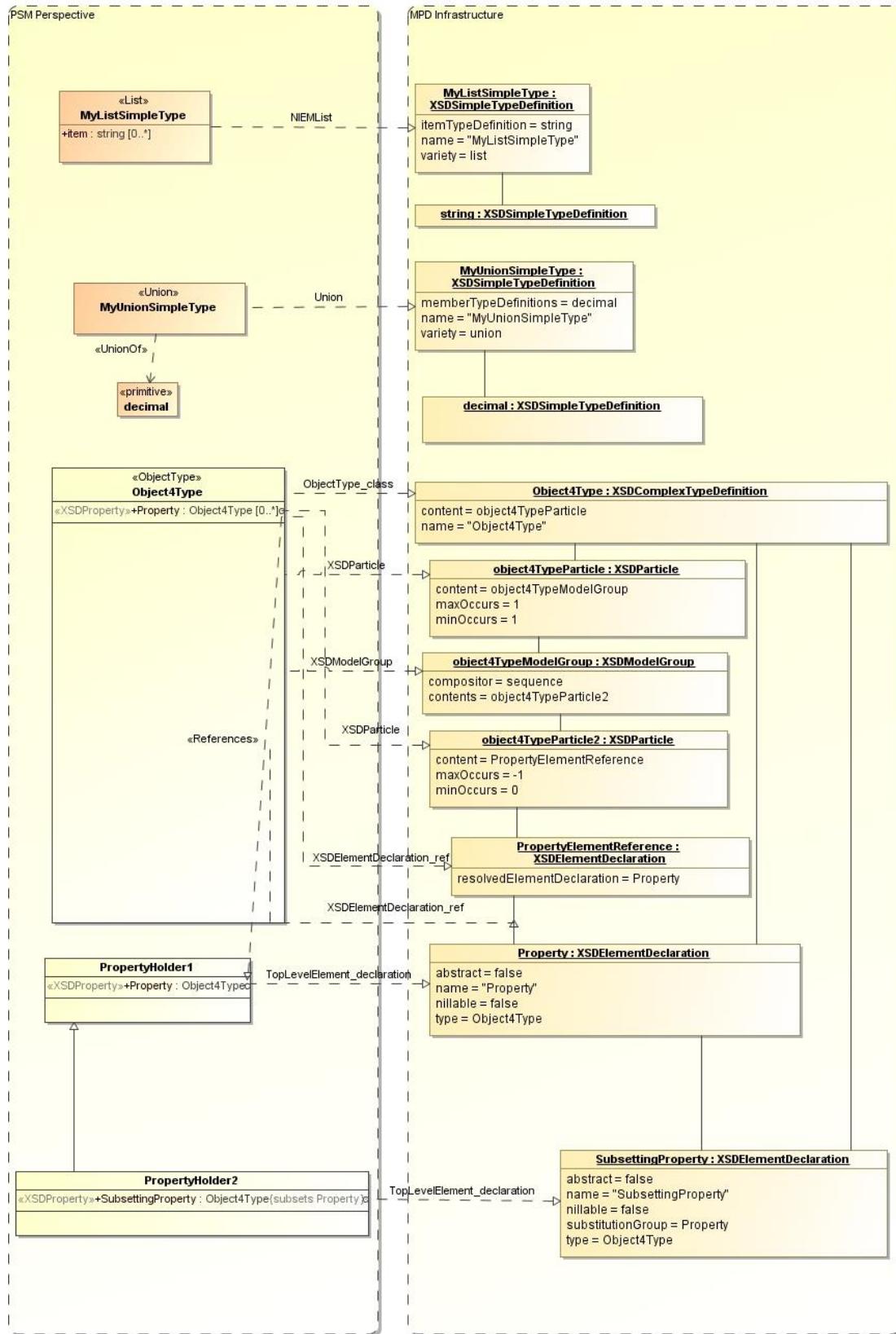
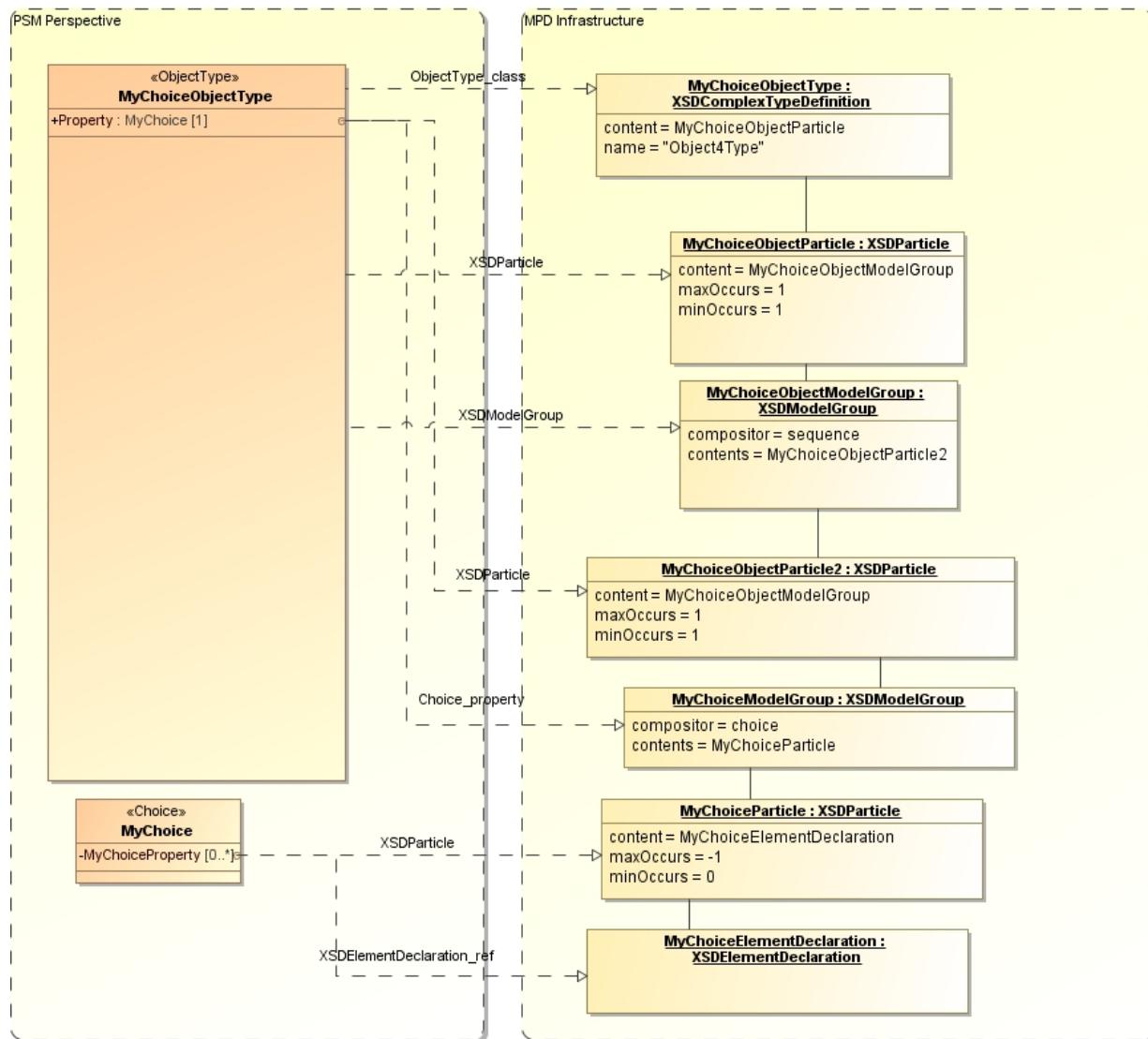


Figure 9-17 NIEM PSM to MPD Schema Artifacts - Common Profile Type Overview

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 XSMDModelGroup{compositor=choice} instead of an XSDElementDeclaration:

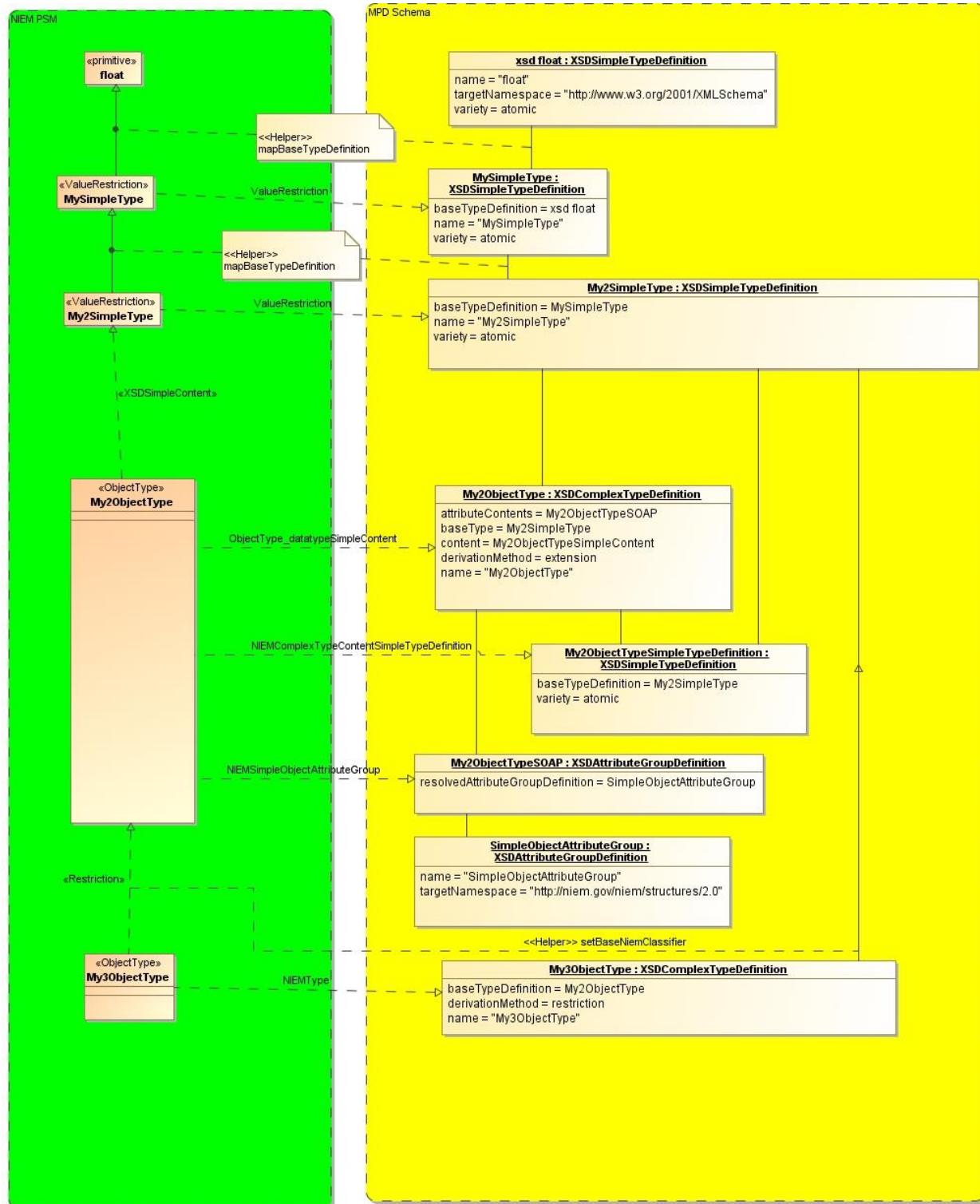
- A «NIEMType» typically maps to an XSDComplexTypeDefinition, an XSDParticle, and an XSMDModelGroup.
- A «NIEMType» may have ownedAttributes which are typed by a «Choice». As with «XSDProperty», these are mapped to an XSDParticle contained by the XSMDModelGroup just mentioned.
- For a Property typed by a «Choice», the content of the XSDParticle is an XSMDModelGroup (instead of an XSDElementDeclaration). The compositor of the XSMDModelGroup is *choice*.
- The contents of the XSMDModelGroup 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».



**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» which 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».



**Figure 9-19 NIEM PSM to MPD Schema Artifact- baseTypeDefinition Overview**

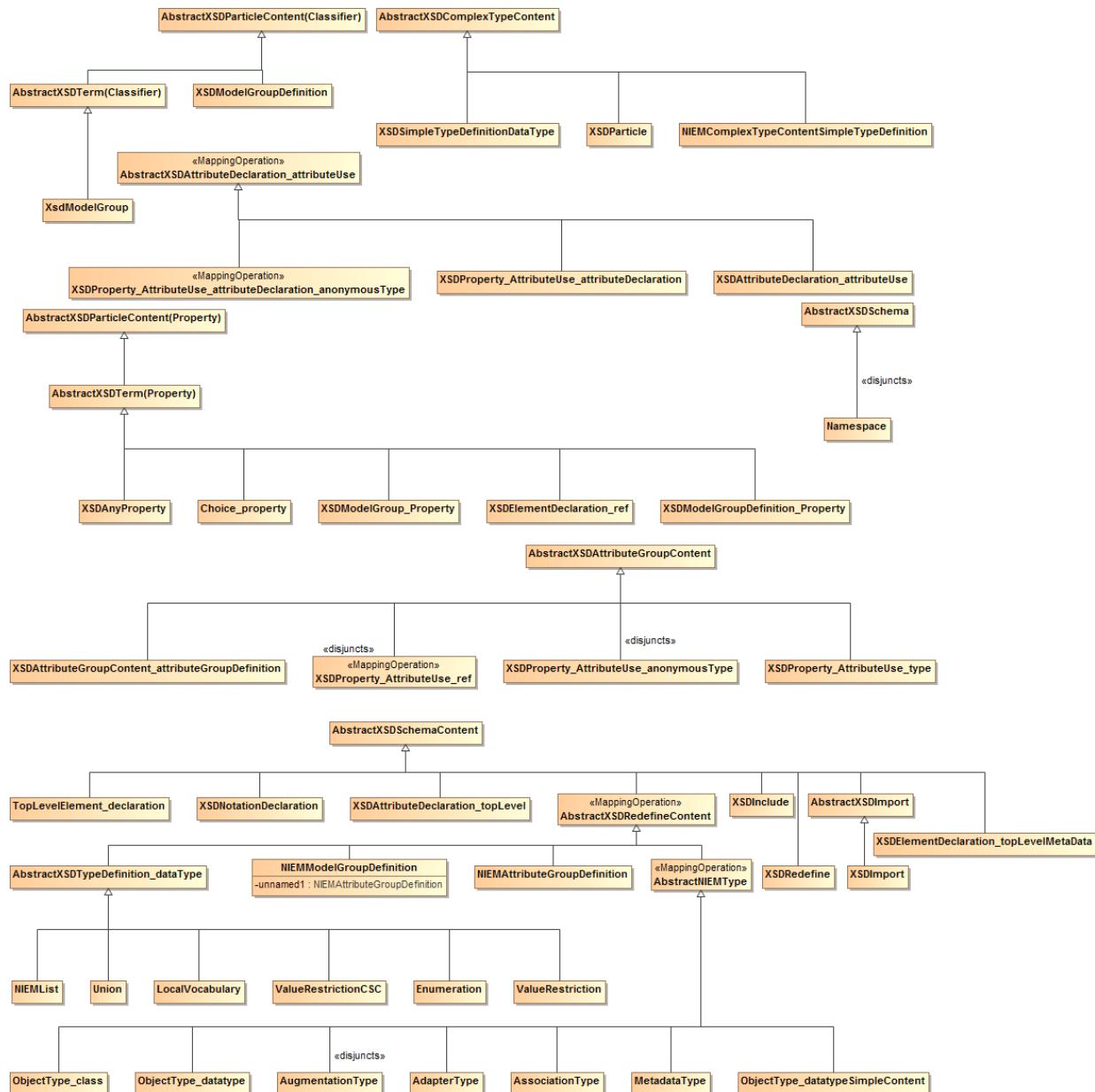
For NIEMpsm2mpd, mapping operations are generally invoked with the context of a source NIEM-UML Element and produce some form of target XSDComponent. Most mapping operations are also provided an argument which is the XSDComponent container context. The “when” condition for the MappingOperations are normally a function of the source NIEM-UML element, the type of the source NIEM-UML element'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.

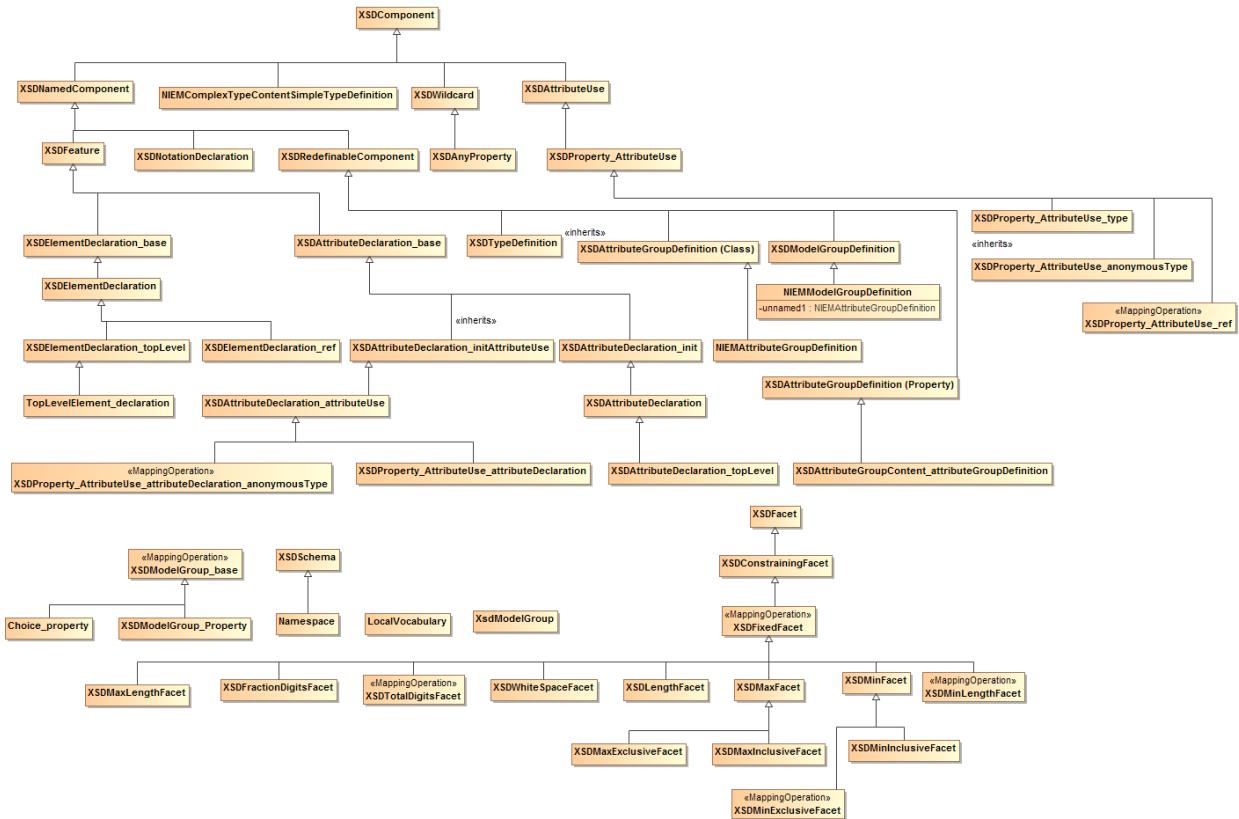
Figure 9-20 illustrates the disjunction pattern for the NIEMpsm2mpd transformation.

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.



**Figure 9-20 NIEM PSM to NIEM-Conforming XML Schema - Disjunction**



**Figure 9-21 NIEM PSM to NIEM-Conforming XML Schema - Inheritance**

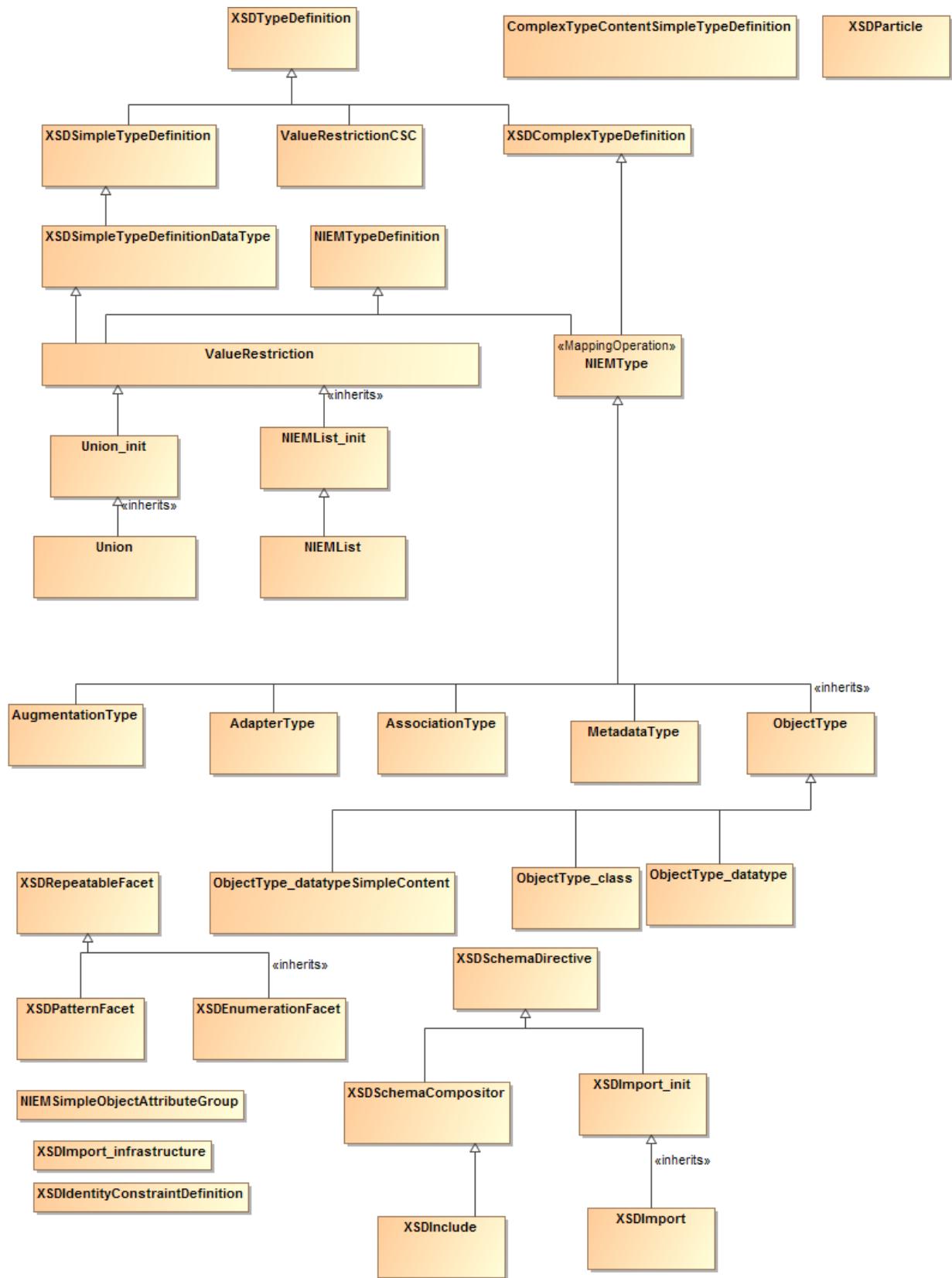


Figure 9-22 NIEM PSM to NIEM-Conforming XML Schema - Inheritance Other

## 9.4 NIEM MPD Model to NIEM MPD Artifact

Figure 9-23 NIEM MPD Model to NIEM MPD Artifact Mapping Overview illustrates the high-level packaging map between NIEM MPD Model and MPD Artifacts.

- A NIEM MPD ModelPackageDescription Artifact Instance is mapped to an MPD Catalog, and will contain all NIEM PSM packaging structure nested to any level. The Catalog includes ArtifactOrArtifactSet 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 «FileType» 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 contained by the source model ModelPackageDescription Artifact Instance maps to a Catalog RelationshipType contained by the MetadataType entry within the CatalogType.

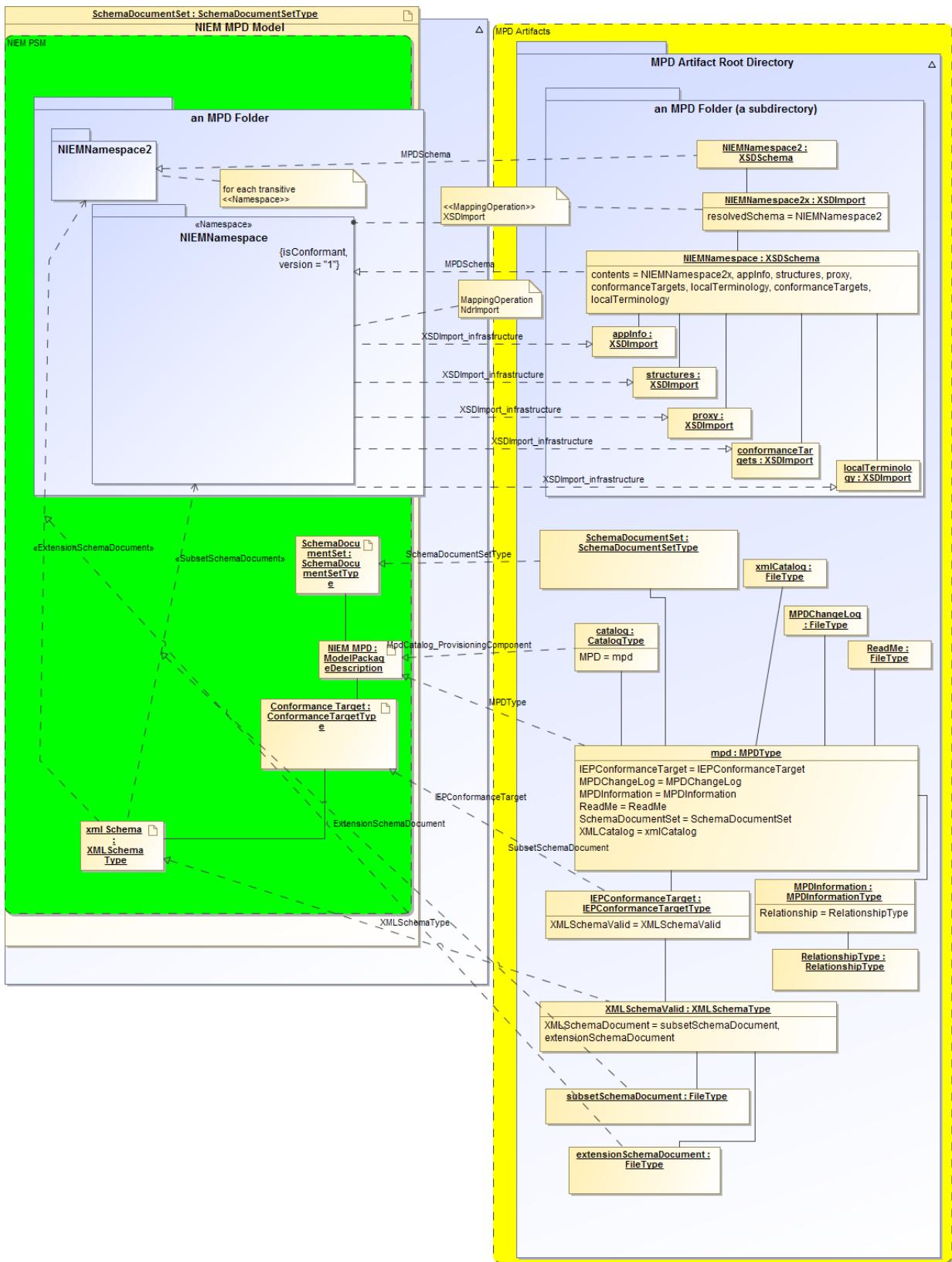


Figure 9-23 NIEM MPD Model to NIEM MPD Artifact Mapping Overview

## 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 Artifact Instance is mapped from an MPD Catalog, and will contain MPD packaging structure (as specified by Catalog ArtifactsAndArtifactSets), nested to any level. The Catalog includes ArtifactsAndArtifactSets 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.

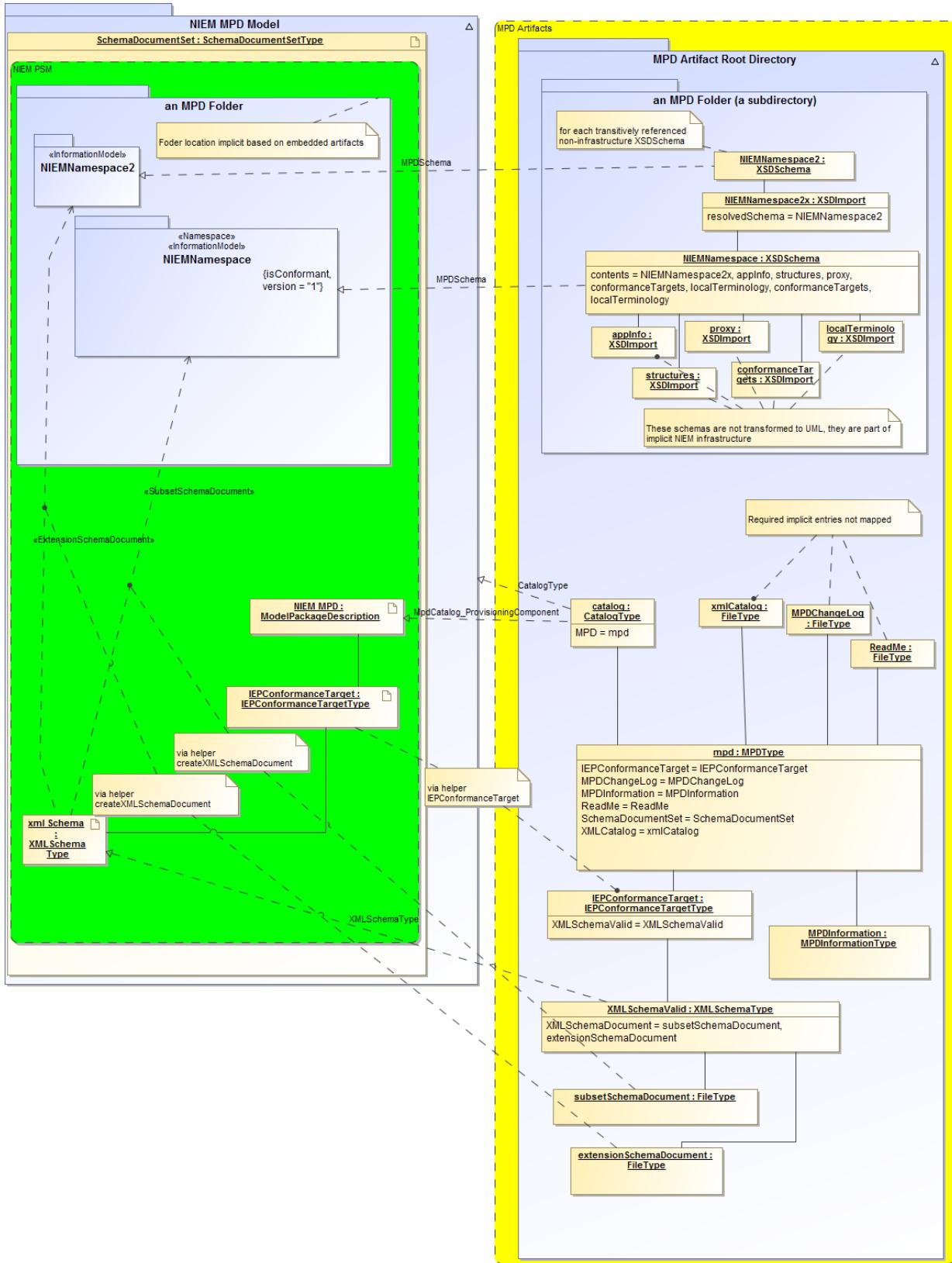


Figure 9-24 NIEM MPD Artifact to NIEM MPD Model - Overview

There are various forms of metadata embodied in the components of a NIEM conformant Schema. The metadata are represented in schemas as specific NIEM-defined XSD Attributes or 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 either a NIEM-defined Attribute or an XSDAnnotation. Figure 9-25 illustrates many specific cases of metadata usage, including those referenced in Clause 9.3.

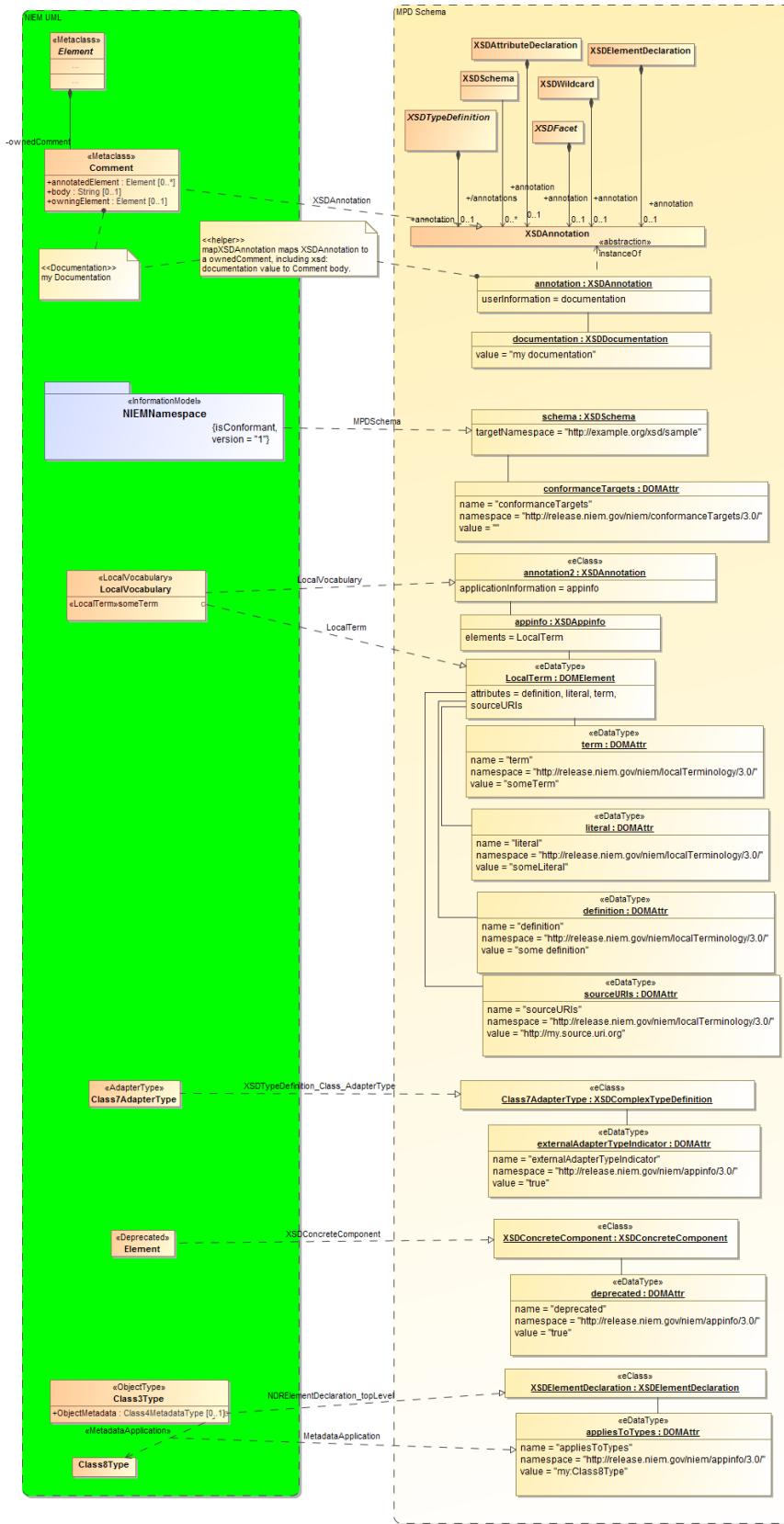
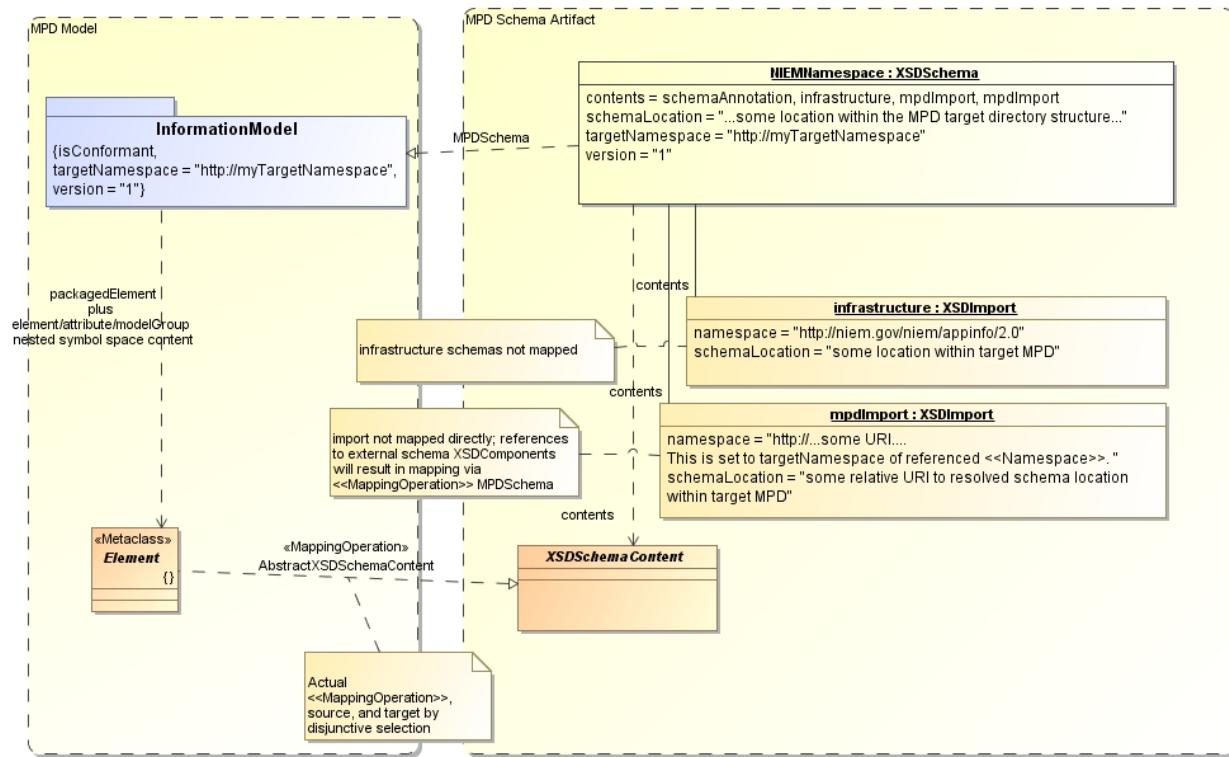


Figure 9-25 MPD Schema Artifacts to NIEM-UML MPD Model – Annotation Mapping Overview

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 NIEM Attributes/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).

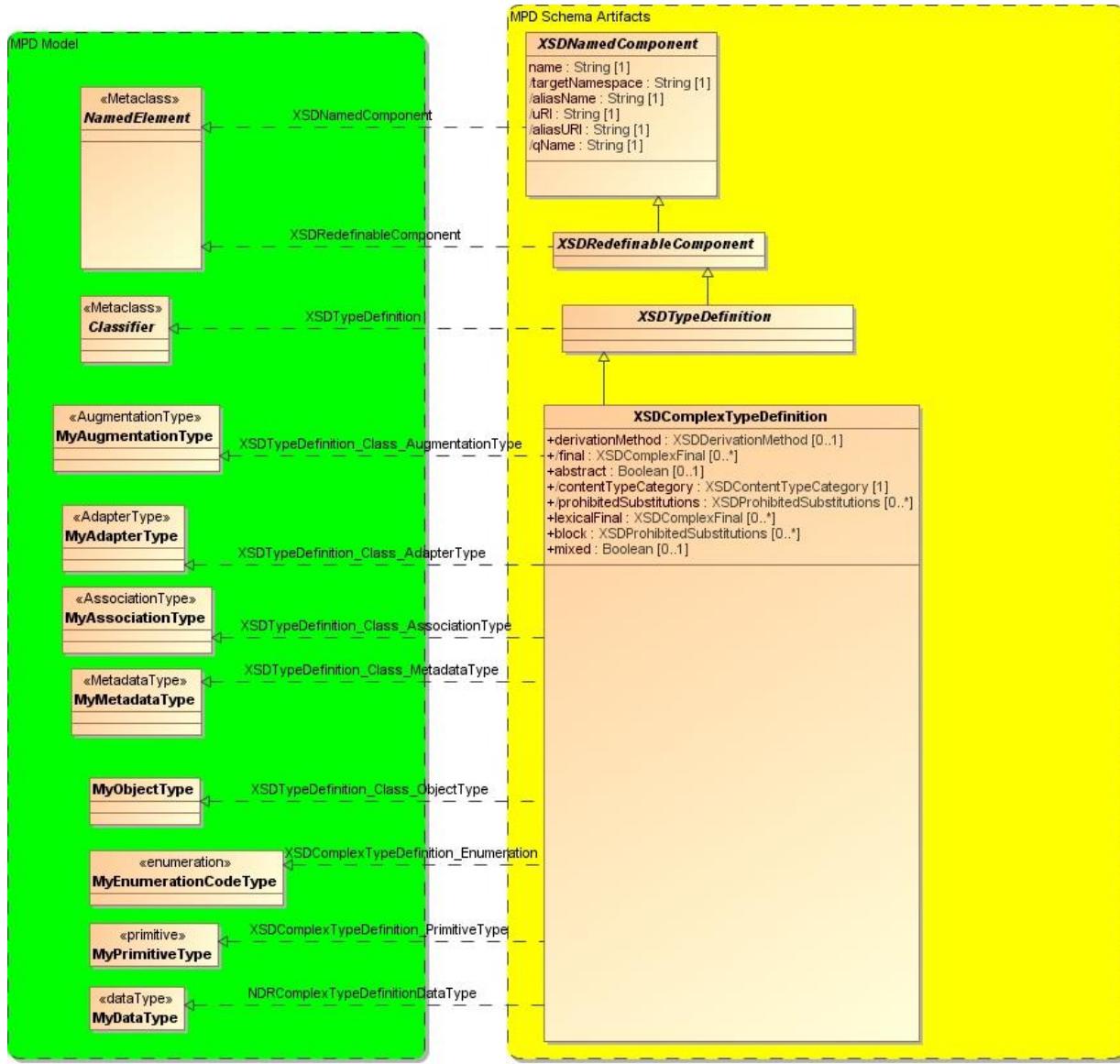


**Figure 9-26 MPD Schema Artifacts to NIEM-UML MPD Model – «InformationModel» Mapping Overview**

«NIEMType»s are mapped from XSDComplexTypeDefinitions, as illustrated in Figure 9-27. NIEM-specific meta information from NIEM Attributes/XSDAnnotations are used to help determine the stereotype to be applied, as outlined earlier.

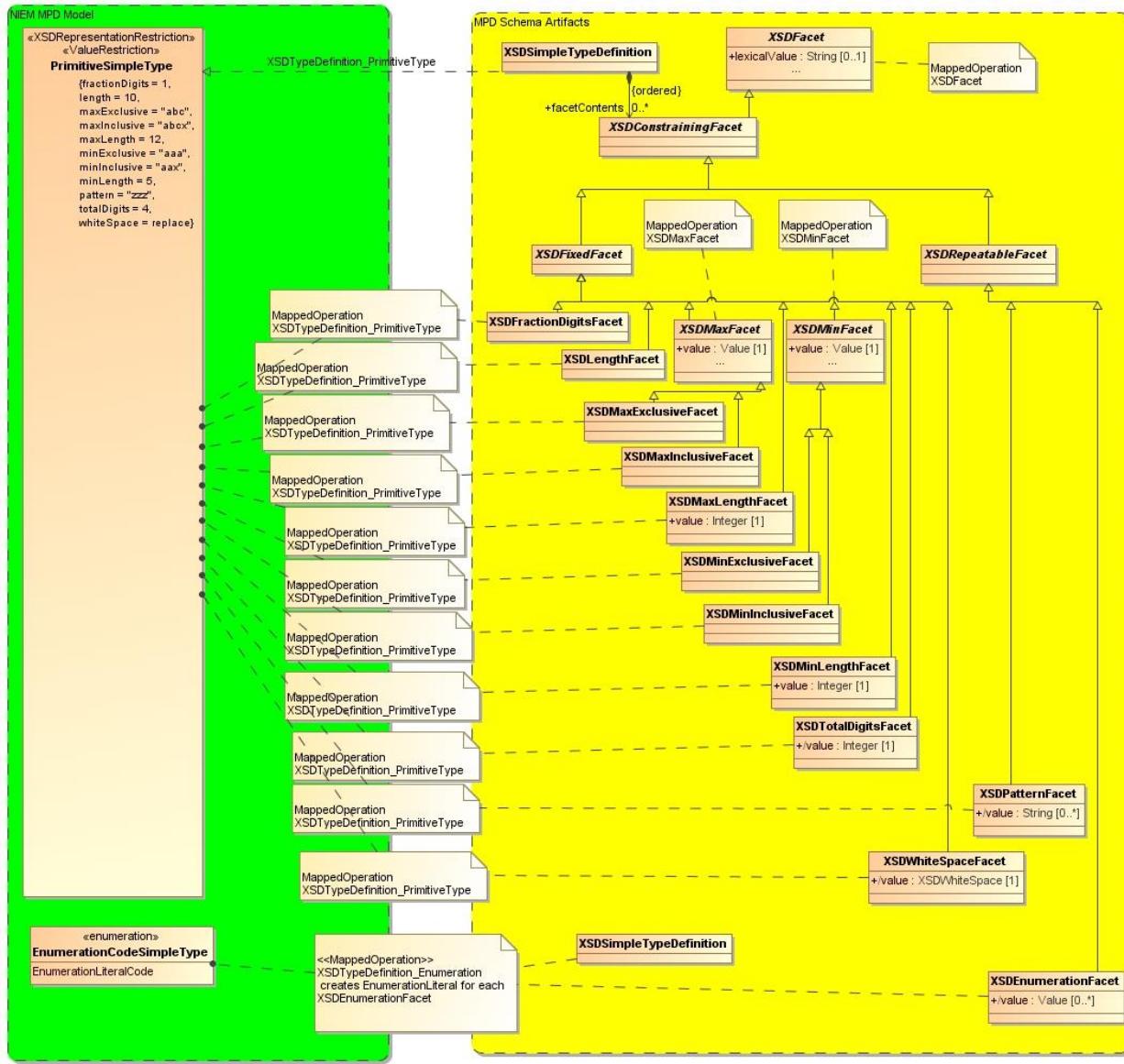
- Inheritance in the NIEM-UML model may be specified as a Generalization or as a «Restriction» Realization. «Restriction» will be used when the derivationMethod of the source XSDComplexTypeDefinition is set to *restriction*. In all other cases, Generalization is used to represent inheritance.
- When 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.



**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» which 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 which have baseTypeDefinitions residing in the structures XSDSchema become some subtype of NIEMType having no inheritance.

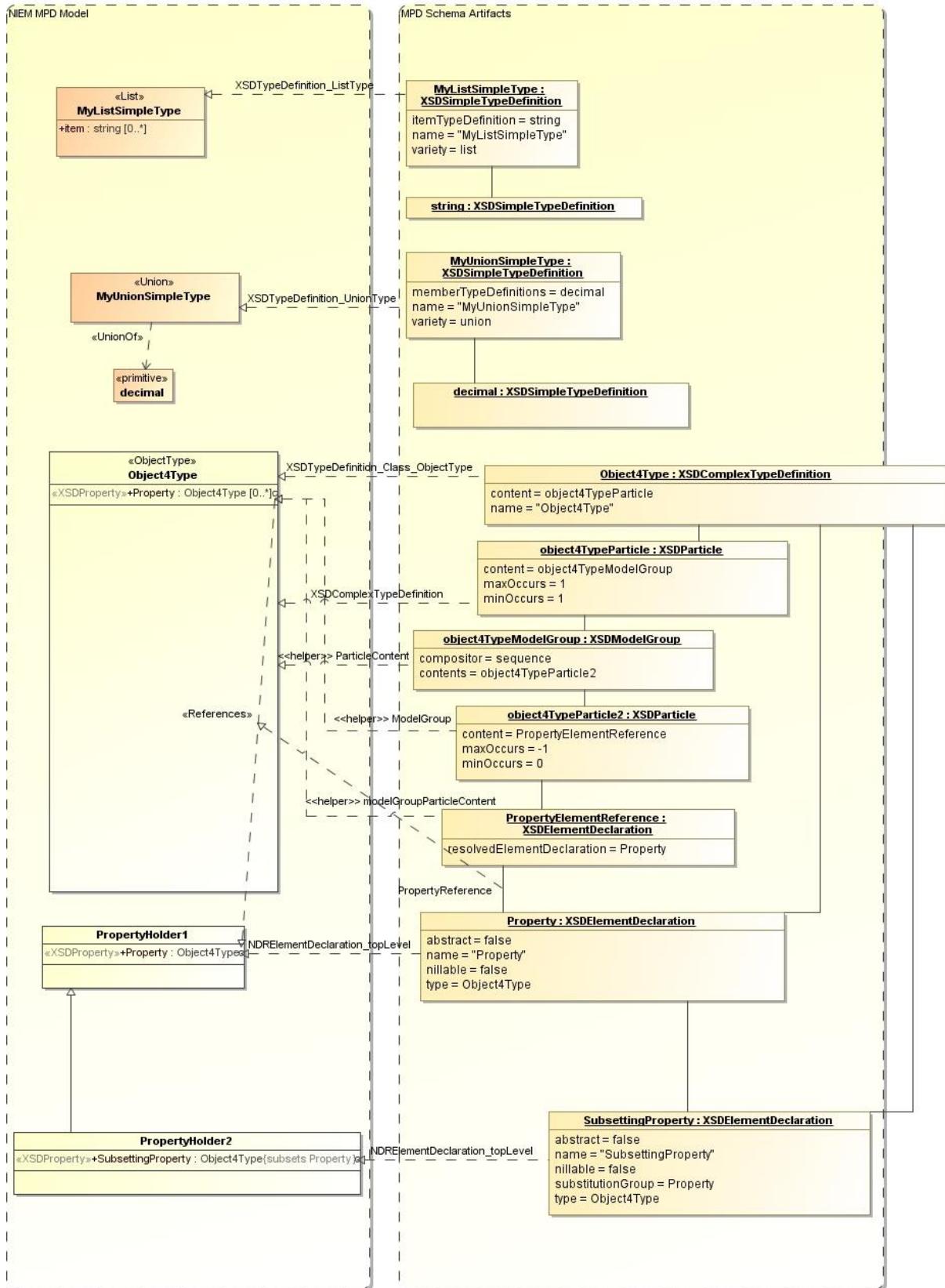


Figure 9-29 MPD Schema Artifacts to NIEM-UML MPD Model – Non-atomic Type Mapping Overview

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 which 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.

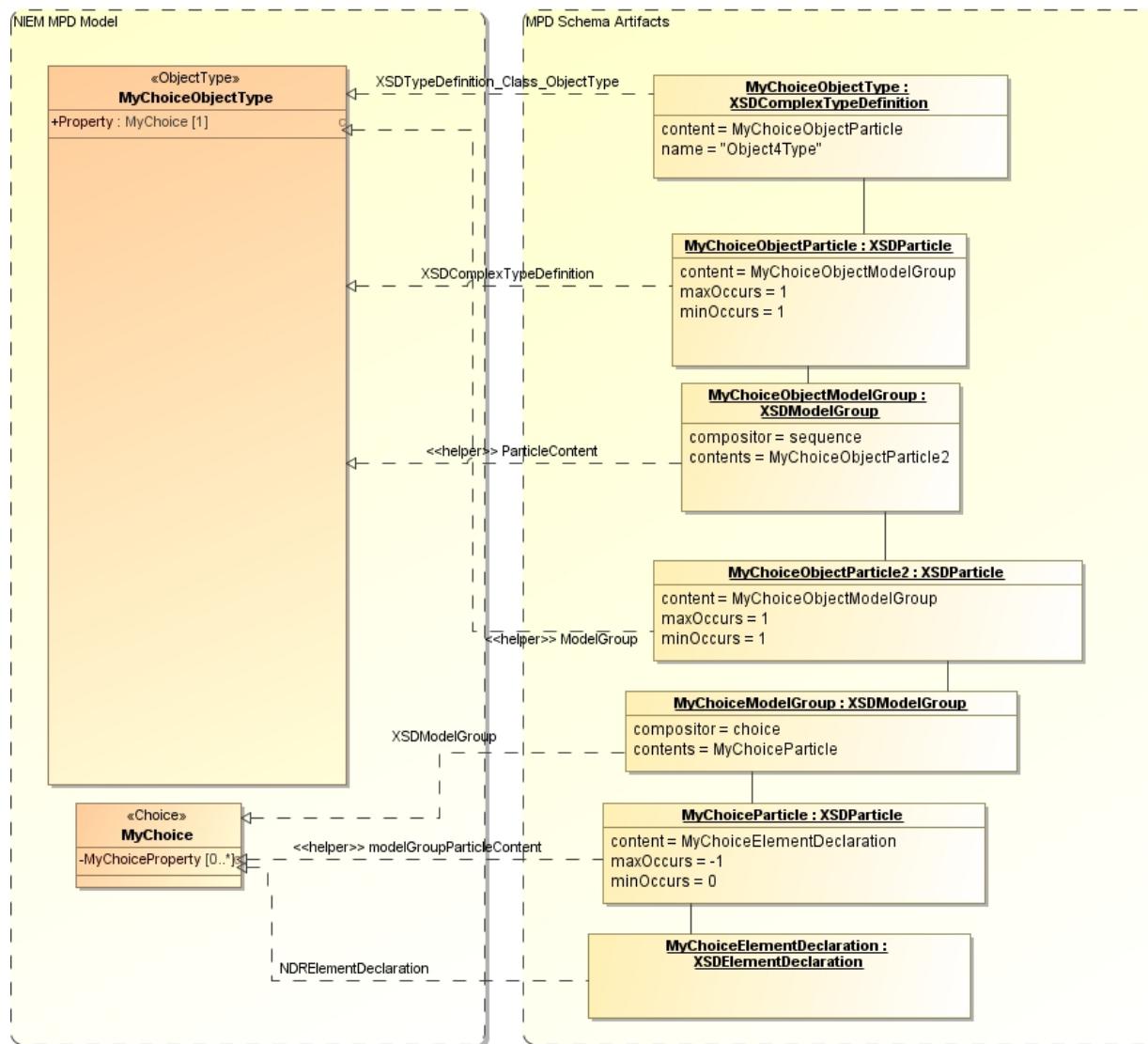
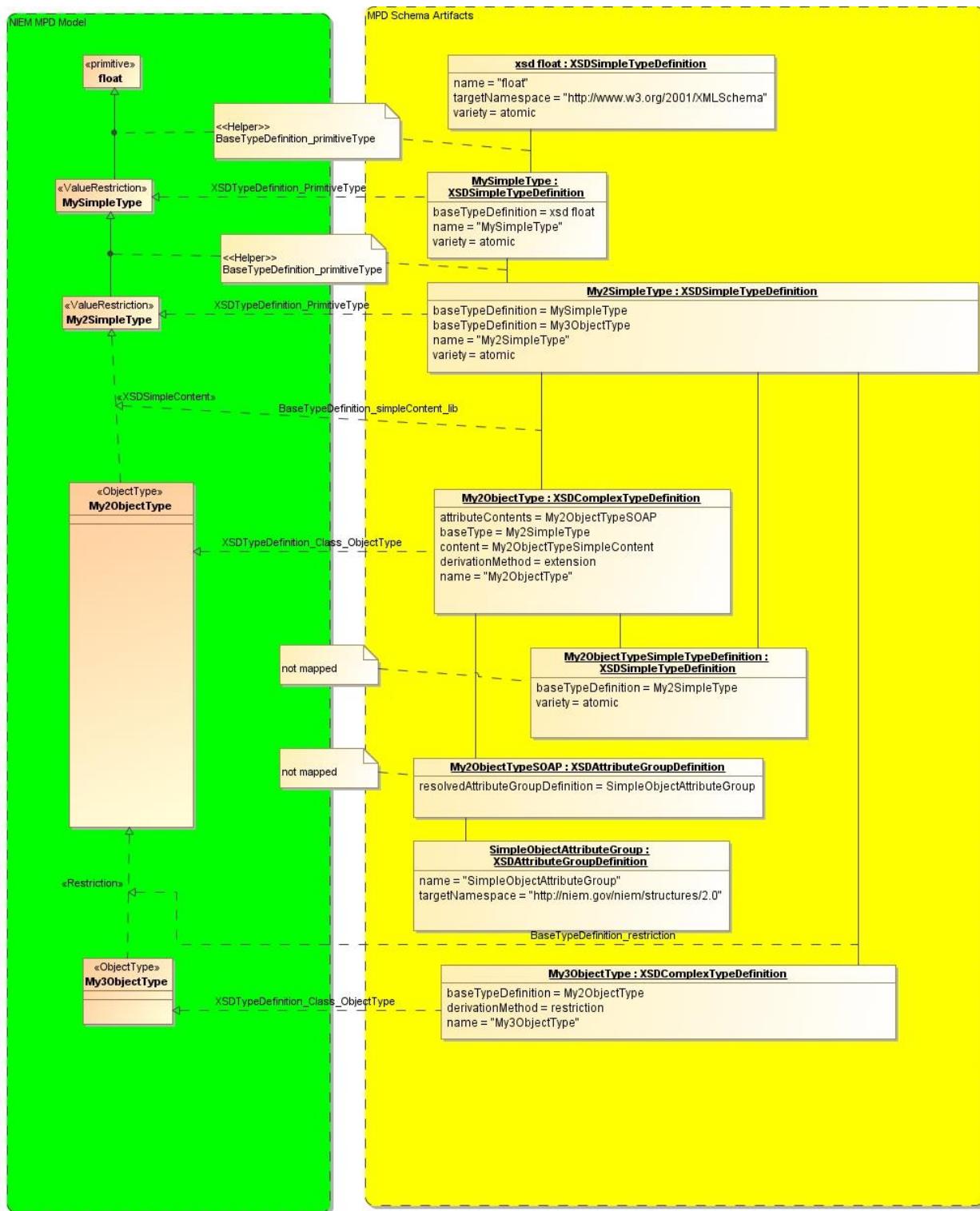


Figure 9-30 MPD Schema Artifacts to NIEM-UML MPD Model – Choice Mapping Overview

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 occurrence 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.

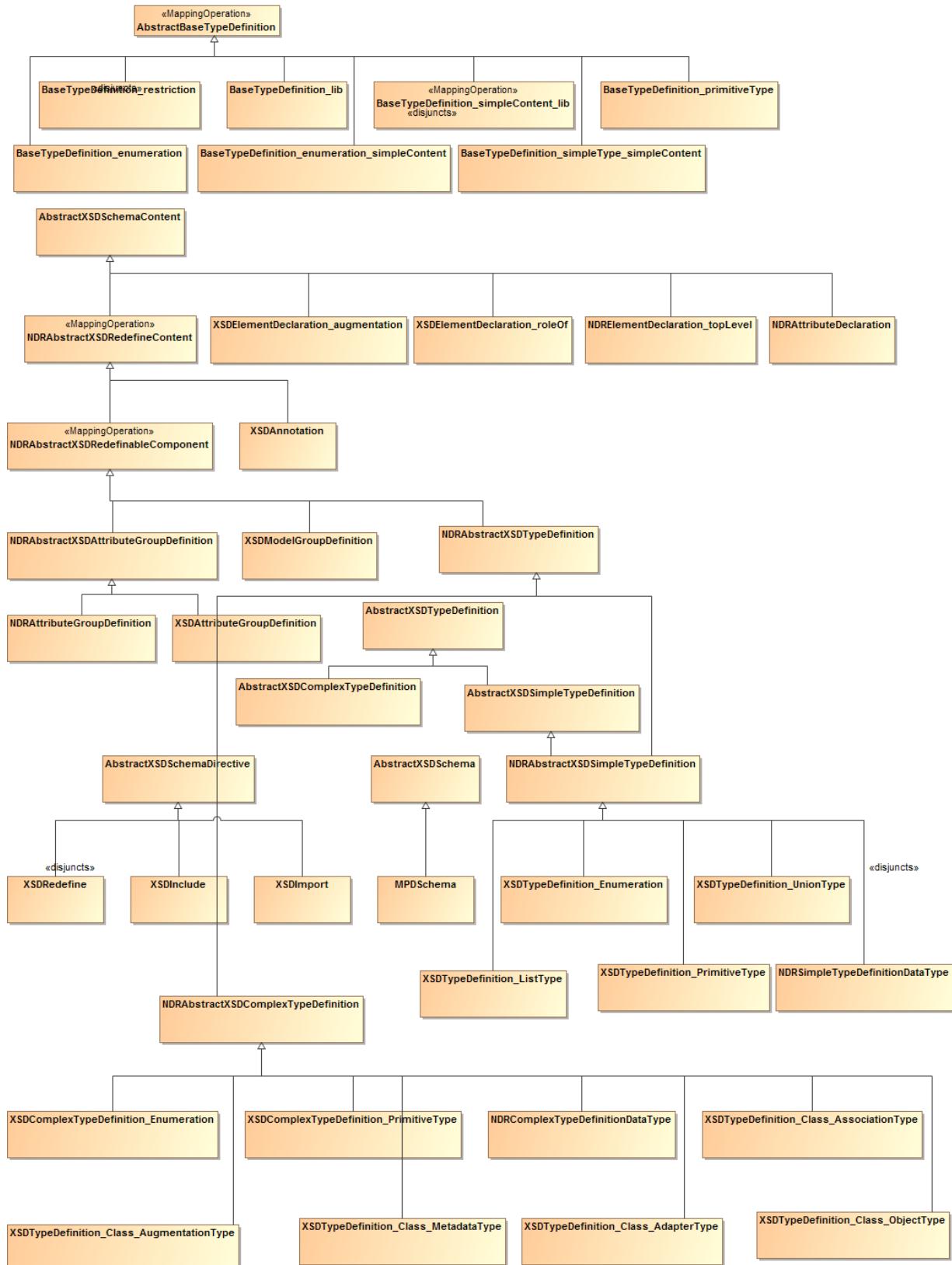


**Figure 9-31 MPD Schema Artifacts to NIEM-UML MPD Model – baseType Mapping Overview**

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

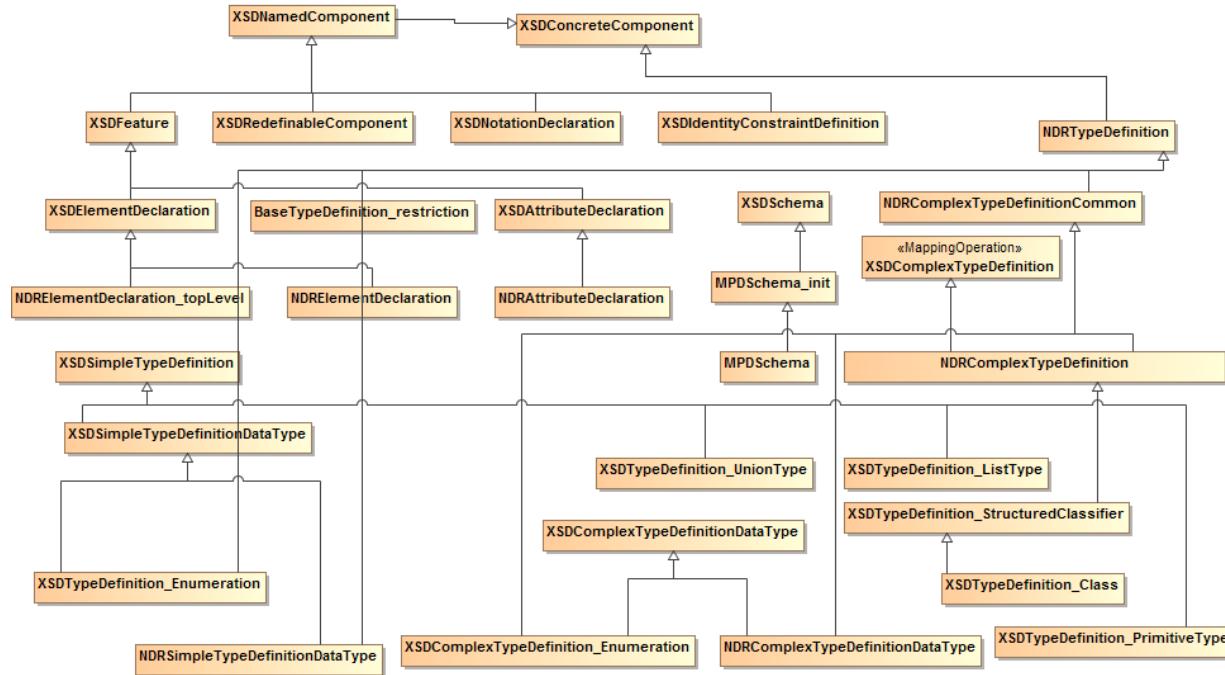
NIEM-UML container context. The mapping operation connects its target NIEM-UMLElement to its container, applies a Stereotype as appropriate, and populates the underlying UML Element properties and/or applied Stereotype tag values.

Figure 9-32 illustrates the disjunction pattern for the NIEMmpdartifact2model transformation.



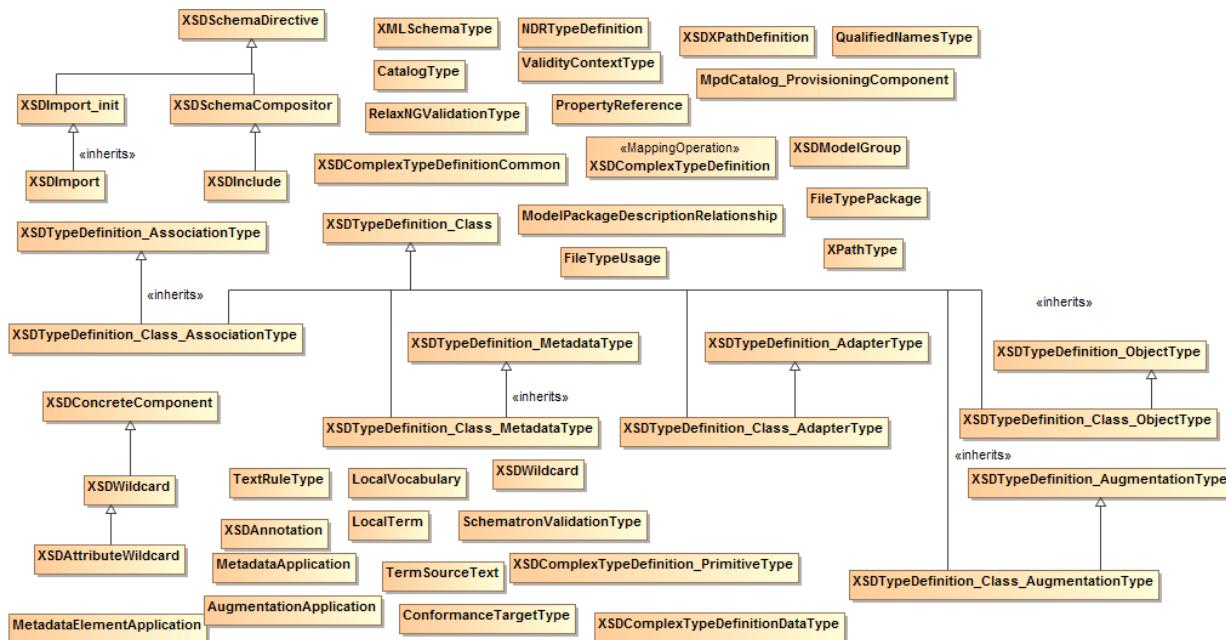
**Figure 9-32 NIEM MPD Artifact to NIEM MPD Model - Disjunction**

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.



**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**

The NIEM-3 MPD, and its specification of an MPD-Catalog, introduces a large number of concepts related to the composition of an MPD. These concepts are encapsulated within the Classifiers defined in the Model\_Package\_Description\_Profile. There is an isomorphic relationship between the Schema Components defined within the MPD-Catalog Schema and the model elements contained by the Model\_Package\_Description\_Profile, which is summarized in the following table.

**Table 9-1 MPD Catalog components**

Schema Component	UML Element
<b>Type</b>	<b>Element</b>
CatalogType	ModelPackageDescription (Artifact)
MPDType	ModelPackageDescription (Artifact)
FileType	«FileType» Usage
FileSetType	FileSetType (Artifact)
SchemaDocumentSetType	SchemaDocumentSetType (Artifact)
MPDNameSimpleType	String (PrimitiveType)
MPDVersionIDSimpleType	String (PrimitiveType)
MPDClassURIListSimpleType	String[0..*] (PrimitiveType)
MPDClassListSimpleType	String[0..*] (PrimitiveType)

RelationshipType	«ModelPackageDescriptionRelationship» Dependency
RelationshipCodeSimpleType	RelationshipCode (Enumeration)
IEPConformanceTargetType	IEPConformanceTargetType (Artifact)
ValidityContextType	ValidityContextType (Artifact)
XPathType	XPathType (Artifact)
XMLSchemaType	XMLSchemaType (Artifact)
SchematronValidationType	SchematronValidationType (Artifact)
RelaxNGValidationType	RelaxNGValidationType (Artifact)
QualifiedNamesType	QualifiedNamesType (Artifact)
ConformanceTargetType	ConformanceTargetType (Artifact)
TextRuleType	TextRuleType (Artifact)
MPDInformationType	Merged with ModelPackageDescription
<b>Element</b>	
Catalog	(Top level element, implicit in UML)
MPD	(implicit in UML)
ArtifactOrArtifactSet	ArtifactOrArtifactSet (Artifact) (unioned with many of the stereotyped Usages)
File	«File» Usage
XMLCatalog	«XMLCatalog» Usage
MPDChangeLog	«MPDChangeLog» Usage
ReadMe	«ReadMe» Usage
IEPSampleXMLDocument	«IEPSampleXMLDocument» Usage
BusinessRulesArtifact	«BusinessRulesArtifact» Usage
XMLSchemaDocument	«XMLSchemaDocument» Usage
ExternalSchemaDocument	«ExternalSchemaDocument» Usage
ExtensionSchemaDocument	«ExtensionSchemaDocument» Usage

SubsetSchemaDocument	«SubsetSchemaDocument» Usage
ReferenceSchemaDocument	«ReferenceSchemaDocument» Usage
EXIXMLSchema	Any Property whose type is ArtifactOrArtifactSet or XMLSchemaType
Wantlist	«Wantlist» Usage
ConformanceAssertion	«ConformanceAssertion» Usage
ConformanceReport	«ConformanceReport» Usage
SchematronSchema	«SchematronSchema» Usage
RelaxNGSchema	«RelaxNGSchema» Usage
Documentation	«Documentation» Usage
ApplicationInfo	«ApplicationInfo» Usage
RequiredFile	«RequiredFile» Usage
FileSet	FileSet (Artifact)
SchemaDocumentSet	SchemaDocumentSet (Artifact)
ConstraintDocumentSet	ConstraintDocumentSet (Artifact)
IEPConformanceTarget	IEPConformanceTarget
ValidityConstraintWithContext	ValidityConstraintWithContext
ValidityConstraint	ValidityConstraint
ValidityContext	(ValidityConstraintWithContext, When type is ValidityContextType)
ValidToXPath	(ValidityConstraint, when type is XPathType)
XMLSchemaValid	(ValidityConstraint, when type is XMLSchemaType)
SchematronValid	(ValidityConstraint, when type is SchematronValidationType)
RelaxNGValid	(ValidityConstraint, when type is RelaxNGValidationType)
HasDocumentElement	(ValidityConstraintWithContext, when type is QualifiedNamesType)

ConformsToConformanceTarget	(ValidityConstraint, when type is ConformanceTargetType)
ConformsToRule	(ValidityConstraint, when type is TextRuleType)
MPDInformation	(implicit in UML, structure flattened)
ExtendedInformation	(Catalog extensions not supported)
AuthoritativeSource	AuthoritativeSource
CreationDate	CreationDate
LastRevisionDate	LastRevisionDate
StatusText	StatusText
Relationship	(From «ModelPackageDescription», Refactored to relationshipCode, UML Package URI of supplier)
KeywordText	KeywordText
DomainText	DomainText
PurposeText	PurposeText
ExchangePatternText	ExchangePatternText
ExchangePartnerName	ExchangePartnerName
<b>Attribute</b>	
mpdURI	mpdBaseURI (see text for expansion to mpdURI)
mpdName	UML name (of ModelPackageDescription Artifact Instance)
mpdVersionID	mpdVersionID
mpdClassURIList	mpdClassCode
pathURI	pathURI
externalURI	externalURI
mimeMediaTypeText	mimeMediaTypeText
resourceURI	UML Package URI of «ModelPackageDescriptionResource» Supplier

relationshipCode	relationshipCode
xPathText	xPathText
qualifiedNameList	«qualifiedName» Usage
conformanceTargetURI	conformanceTargetURI

# 10 NIEM-UML PIM Example (informative)

## 10.1 Example Description

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

The business use case is for “Pet Adoption Centers” which need to share information on their pet adoptions with each other and with government agencies. The information required includes data about the pets, the people adopting the pets and the pet adoption centers. Information for sets of adoptions is defined in a NIEM exchange as part of an “IEPD” (Information Exchange Package Documentation). This and other resources, examples and information relating to NIEM-UML may be found on the NIEM GITHUB site at <https://github.com/NIEM/NIEM-UML/>.

## 10.2 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.

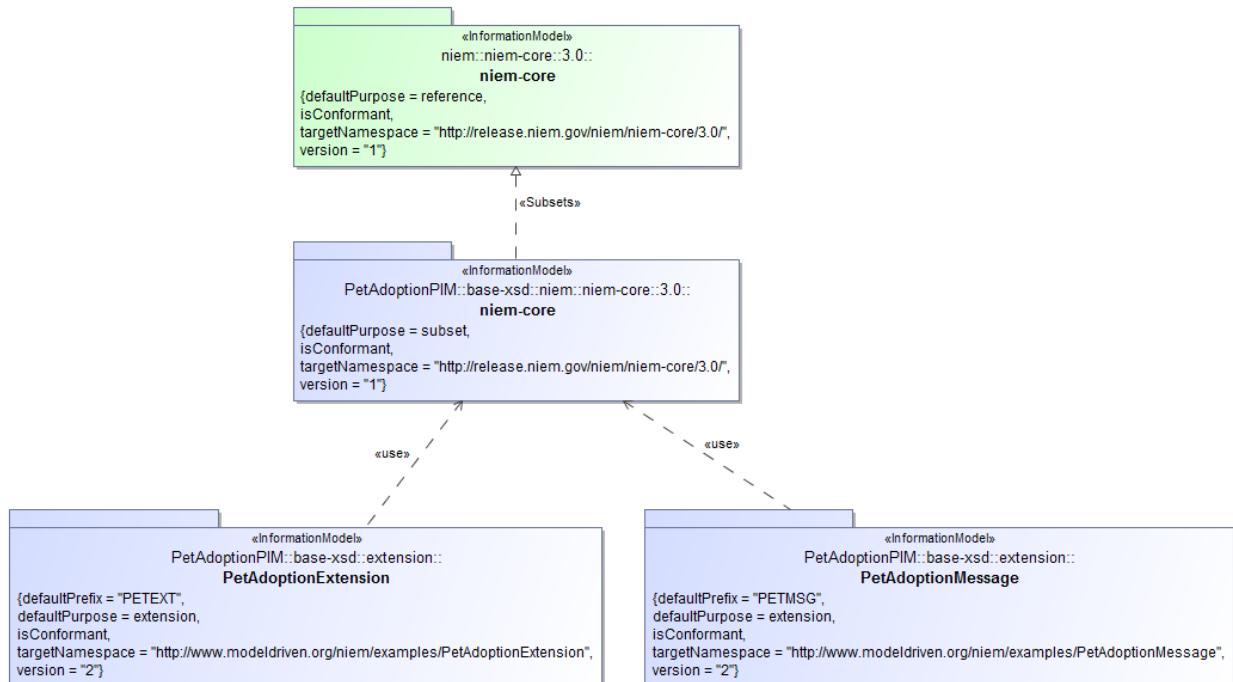


Figure 10-1 Namespace Organization

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 used in another information model which defines a schema for a particular kind of message.

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

- **PetAdoptionPIM::base-xsd::niem::niem-core::3.0::niem-core** – 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. It has the same name and namespace.
- **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.
- **PetAdoptionMessage** – this is another NIEM extension namespace and defines the types representing actual exchanges between parties. This namespace uses classes from **PetAdoptionExtension** to specify these types.
- **niem::niem-core::3.0::niem-core** – NIEM-Core is the standard namespace as supplied by NIEM, it is not unique to this model. The PetAdoptionPIM niem-core subset 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 above the class names.

Note that there is one additional package, which is used to hold the Model Package Description, defined in Subclause 10.21.

## 10.3 High-Level Design



**Figure 10-2 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 “PetAdoptionCenter” are part of the “PetAdoptionExtension” schema. An extension schema is normally where new domain concepts are defined.

## 10.4 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 10-3.

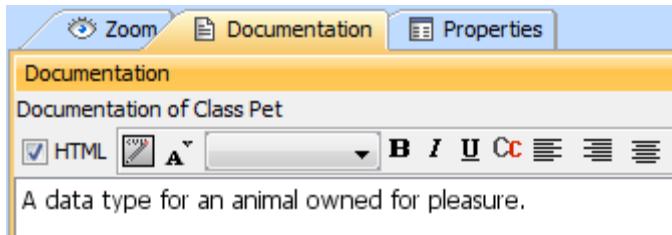


Figure 10-3 Documenting Elements

## 10.5 UML Associations Defining NIEM Properties

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

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

## 10.6 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”.

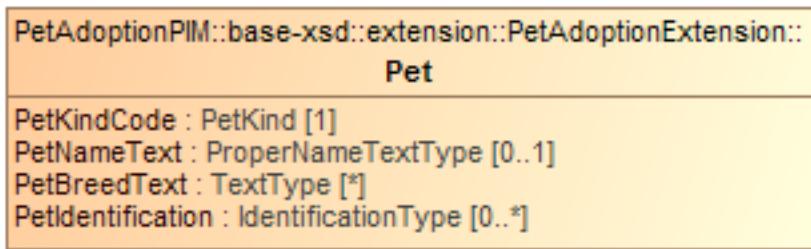
The enumeration is a type with a specific set of values. In Figure 10-4 we have defined values with tokens for each kind of pet we are concerned with; Dog, Cat, Bird, etc. The UML enumeration defines a NIEM “Code Type”.



**Figure 10-4 Enumeration**

## 10.7 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.



**Figure 10-5 Properties of Pet**

Pet has four properties:

- PetKindCode, using the same enumeration, above. A pet can only have one kind.
- PetNameText, using the type ProperNameTextType. The pet name is optional.
- PetBreedText, a TextType. To support mixed breeds pets can have multiple breeds.
- And PetIdentification, which we will explore next.

## 10.8 Properties Using Classes as Their Types

The type of the PetIdentification property is “IdentificationType”, let’s take a look at the IdentificationType class.

The IdentificationType class in Figure 10-6 is another class like Pet or Person, but it already has several properties. Note that the properties have types like “string”, “TextType” and “DateType” – these are all built-in primitive types. But where did all this come from? Note that it is in the PetAdoptionPIM subset schema. This means that IdentificationType reuses an existing class in NIEM Core.

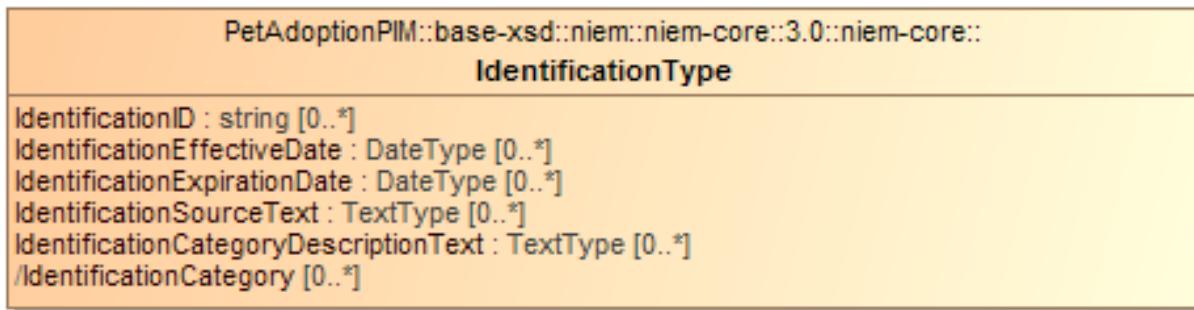


Figure 10-6 Properties Using Identification Class

## 10.9 Finding Classes in Reference Namespaces

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

You can see in Figure 10-7 that we are “in” the “niem-core”, version “3.0”. 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.



Figure 10-7 NIEM-Core Listing

Further down we see classes having to do with Identification:

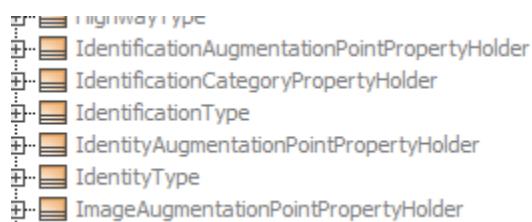


Figure 10-8 Finding the Identification Class

Pulling the IdentificationType class into a UML diagram in Figure 10-9 we see:

<b>niem::niem-core::3.0::niem-core:: IdentificationType</b>
<pre>IdentificationID : string [0..*]{kind = element, nullable} IdentificationJurisdiction : JurisdictionType [0..*]{kind = element, nullable} /IdentificationCategory [0..*]{readOnly,union,kind = element, nullable = false} IdentificationCategoryDescriptionText : TextType [0..*]{kind = element, nullable} IdentificationEffectiveDate : DateType [0..*]{kind = element, nullable} IdentificationExpirationDate : DateType [0..*]{kind = element, nullable} IdentificationSourceText : TextType [0..*]{kind = element, nullable} IdentificationStatus : StatusType [0..*]{kind = element, nullable} /IdentificationAugmentationPoint [0..*]{readOnly,union,kind = element, nullable = false}</pre>

**Figure 10-9 Identification Class Contents**

This seems to have a lot of the identification properties we need, perhaps more than we need! We also see that the property `IdentificationCategory` has no type and is marked as a `readOnly` derived union; this is a placeholder 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.

<b>«PropertyHolder» niem::niem-core::3.0::niem-core:: IdentificationCategoryPropertyHolder</b>
<pre>/IdentificationCategory [0..*]{readOnly,union,kind = element, nullable = false} IdentificationCategoryText : TextType [0..*]{subsets IdentificationCategory,kind = element, nullable}</pre>

**Figure 10-10 IdentificationCategory Substitution Group**

Note that in NIEM-Core “`IdentificationCategory`” can only have one representation (Text), however this could be expanded in other information models. 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.

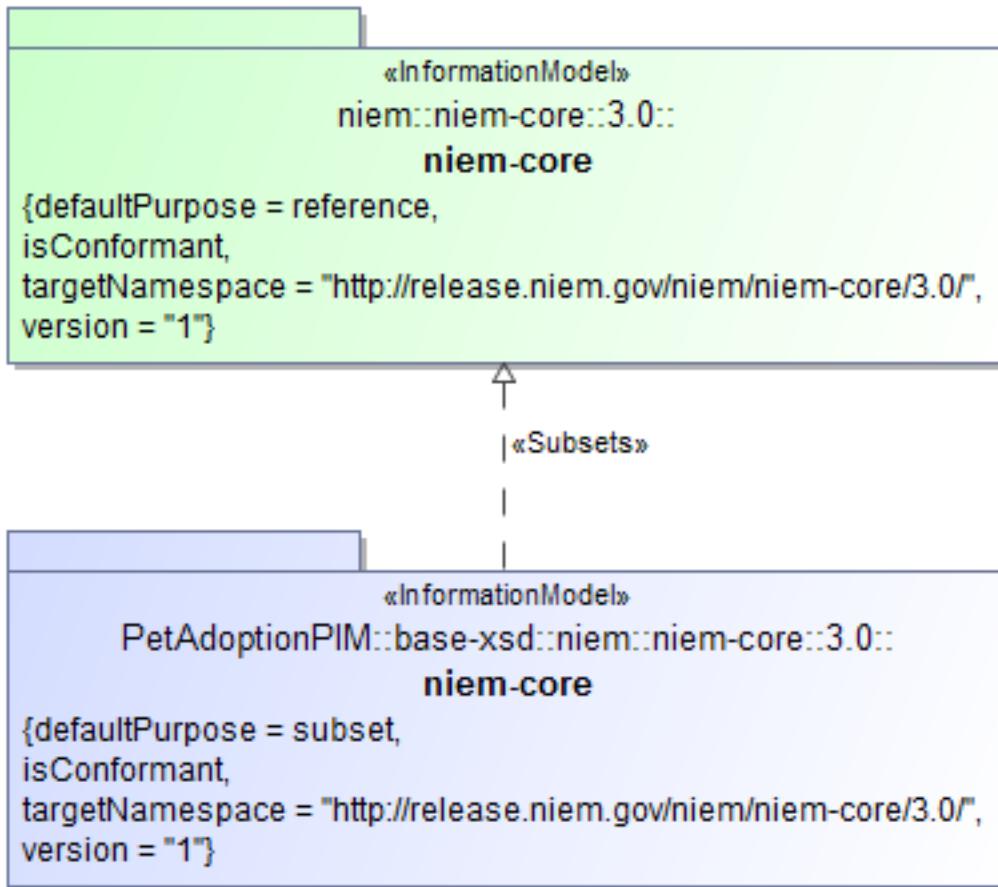
There is one other special feature being used here, that is «`PropertyHolder`». The property holders define properties that can be used in classes but aren’t used in any class yet. The property class is kind of invisible to NIEM. The properties in a property holder are known as “global properties”. Since 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 “`IdentificationType`” in our model.

## 10.10 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.

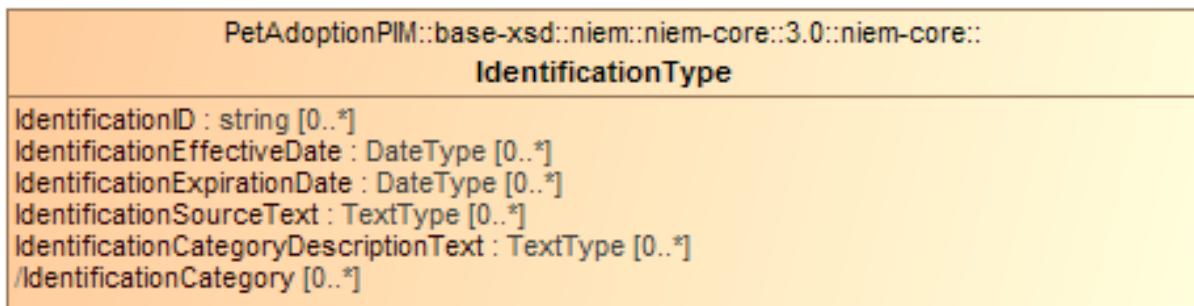
A subset information model has a «`Subsets`» to the model it subsets. Everything in that subset namespace will automatically (by name) subset the corresponding element in the reference namespace. The following figure shows that `PetAdoptionPIM::base-xsd::niem::niem-core::3.0::niem-core` subsets `niem::niem-core::3.0::niem-core`.



**Figure 10-11 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 “IdentificationType” that builds on all these parts.

The “IdentificationType” class in the PetAdoptionPIM subset «subsets» IdentificationType in in the niem-core reference model. Note that since it is in the PetAdoptionPIM subset 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.



**Figure 10-12 Subset IdentificationType**

To create the subset class shown in Figure 10-12 we copied properties from the NIEM-Core version to make the class we want for our pet model; how this is done is tool specific. We left out the properties we don’t need. Note that the types referenced by the included properties must be types in the subset model, not the reference model: in this

esee DateType and TextType are also included in the subset model, and those are the ones referenced by the IdentificationType properties.

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 in Figure 10-12 is the one we are going to use; it is the one that is the type of the pet’s identification but will also be used to identify people and adoption centers.

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!

## 10.11 Reusing Person

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

As you can see in Figure 10-14, “Person” in NIEM-Core is huge! What we did is just pick three properties that we want from 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 a corresponding subsetting type. 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 by the transformation. We have also narrowed the multiplicities, which is a legal and normal thing to do in a subset schema.

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 play may change over time.

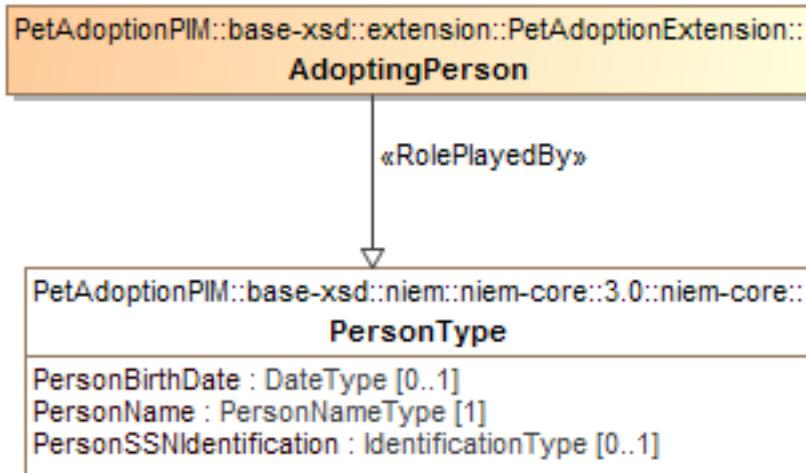
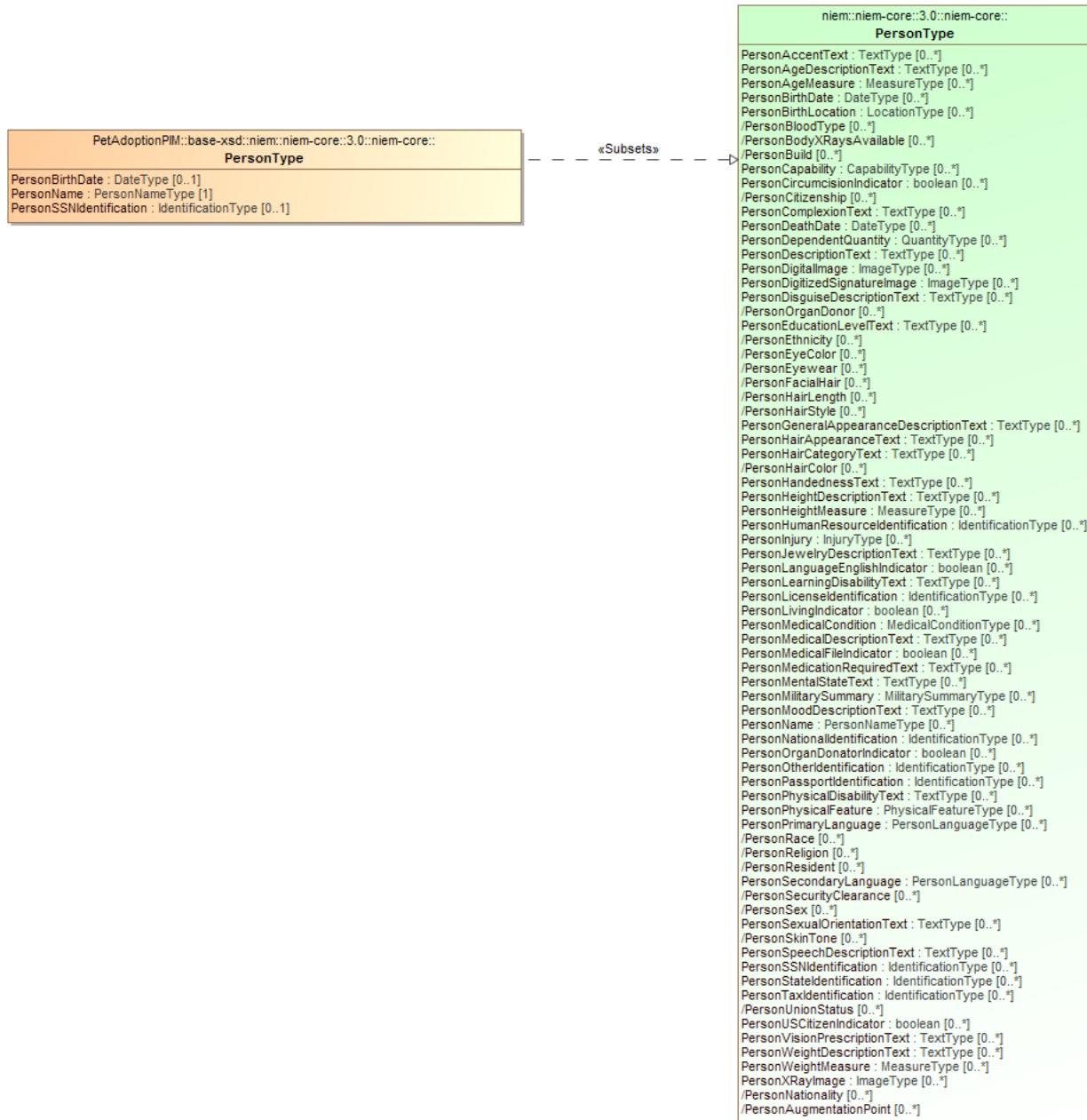


Figure 10-13 Role of Person



**Figure 10-14 Reuse of Person Class**

## 10.12 Reusing Person Name

The name of a person in NIEM core has a type of “PersonNameType”. Without much problem we find “PersonNameType” 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. So we have a person and a person’s name. We can now use the “PersonName” property already defined in NIEM-Core.

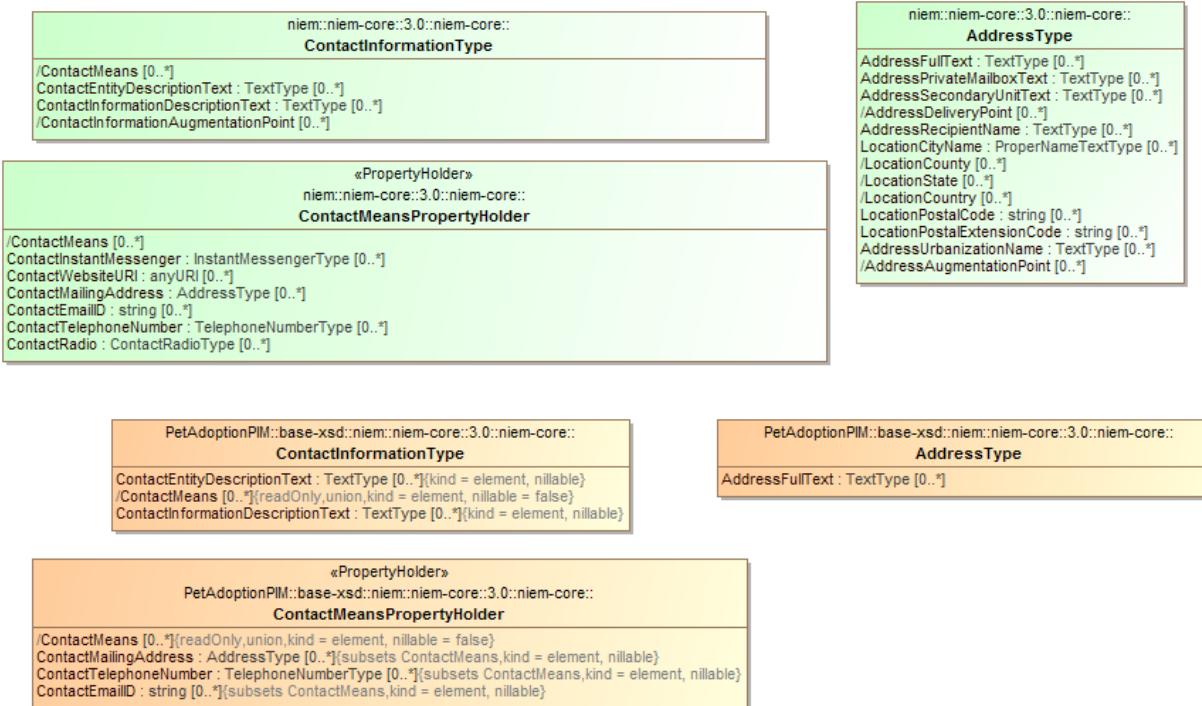


Figure 10-15 Reusing Person Name

## 10.13 Contact Information

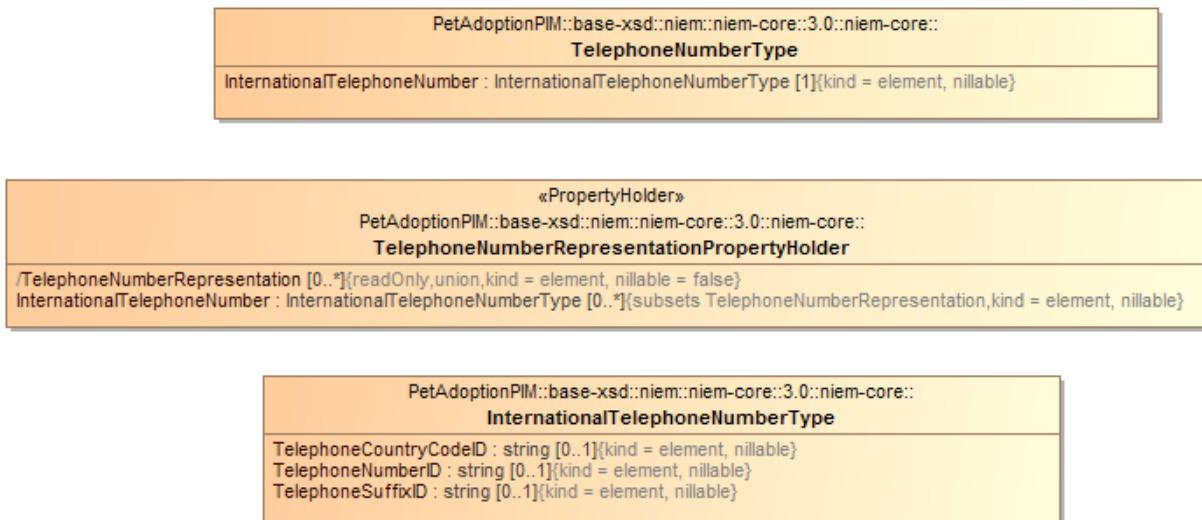
In addition to their name we are also going to want to record multiple ways to contact people (as well as adoption centers, but that is later).

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 10-16. We leave off the implied «Subsets» as you now understand the relationship between the subset and NIEM-Core.



**Figure 10-16 Referencing Contact Information**

As you can see in Figure 10-16, this is a very general idea of contact information that can have any number of various kinds of contacts. “ContactInformationType” has an untyped property “ContactMeans” which is used as the base of a substitution group with a set of properties that subset it. We have chosen to allow 3 possibilities out of the 10 pre-defined in NIEM-Core: ContactEmailID, ContactTelephoneNumber 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 could also extend the list of possible representations by subclassing the property holder in our extension model. Since we have used telephone number we have to define this as well, drawing from NIEM-Core. In Figure 10-17 we just show the subsetting model fragment.



**Figure 10-17 Telephone Number**

## 10.14 Augmenting Telephone Number

Consider that we wanted some additional information about telephone numbers and wanted to be able to “mix in” that information with a variety of telephone numbers. This is the use case for NIEM augmentations. An augmentation defines a type that can be “mixed in” with other types.

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 augments the telephone type.

The augmentation type in Figure 10-18 is stereotyped as an «AugmentationType» which «Augments» TelephoneNumberType. In the XSD an augmentation property “TelephoneNumberAugmentation” will be a substitute for “TelephoneNumberTypeAugmentationPoint” and will thus be able to augment any telephone number. Our new TelephoneNumber can now be used anywhere the NIEM-Core representation of telephone number may be used.

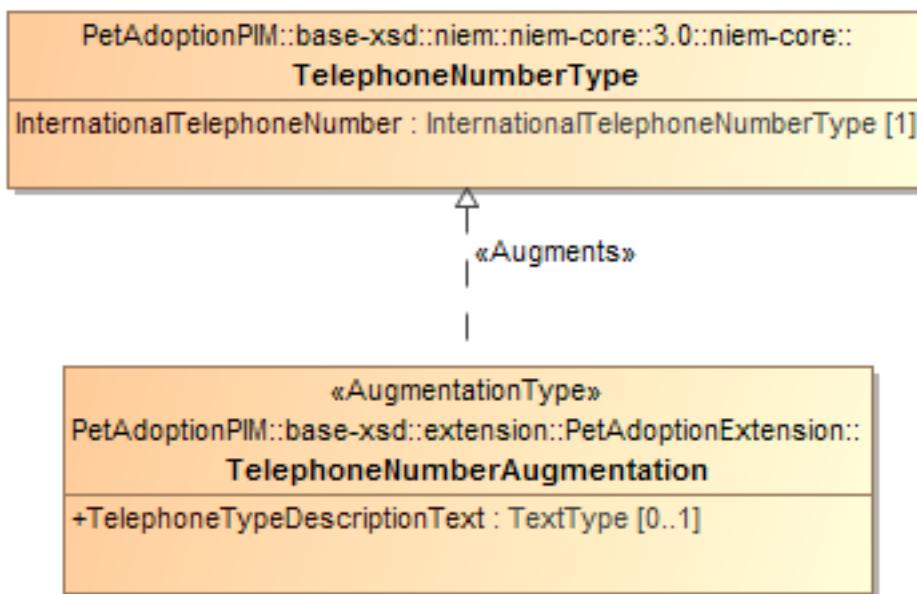


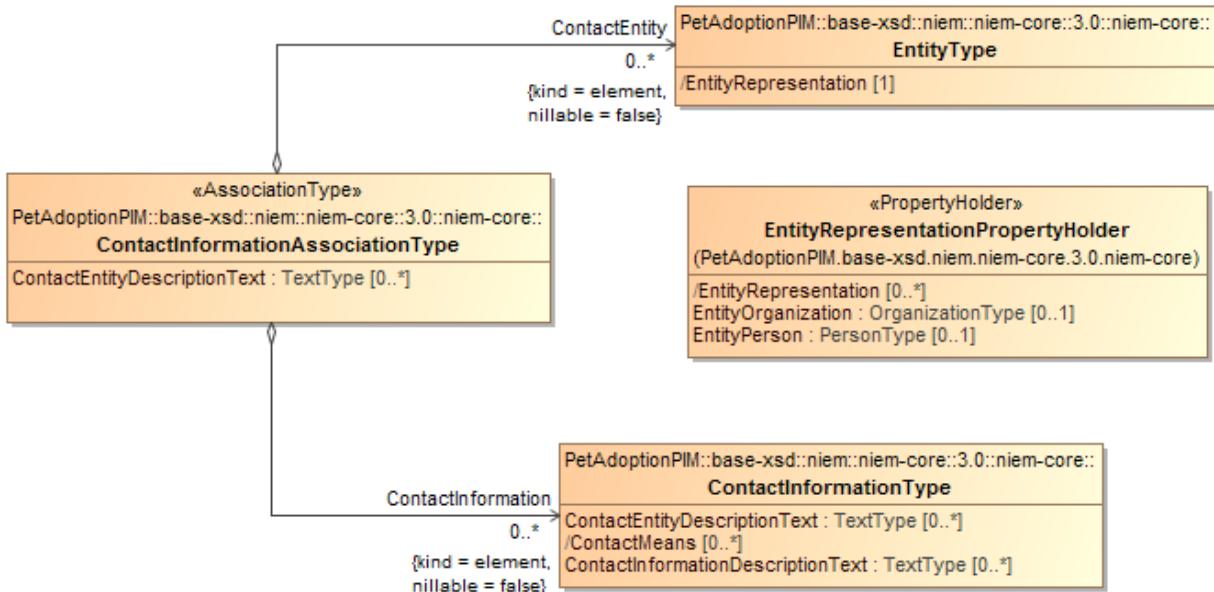
Figure 10-18 Augmenting Telephone Number

## 10.15 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 EntityType and ContactInformationType. 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 10-19 we see a NIEM association “ContactInformationAssociationType” that is defined in NIEM core and then reused in our example. This NIEM association allows us to connect any entity with any piece of contact information. Note that EntityRepresentation can be a Person or an Organization.

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.

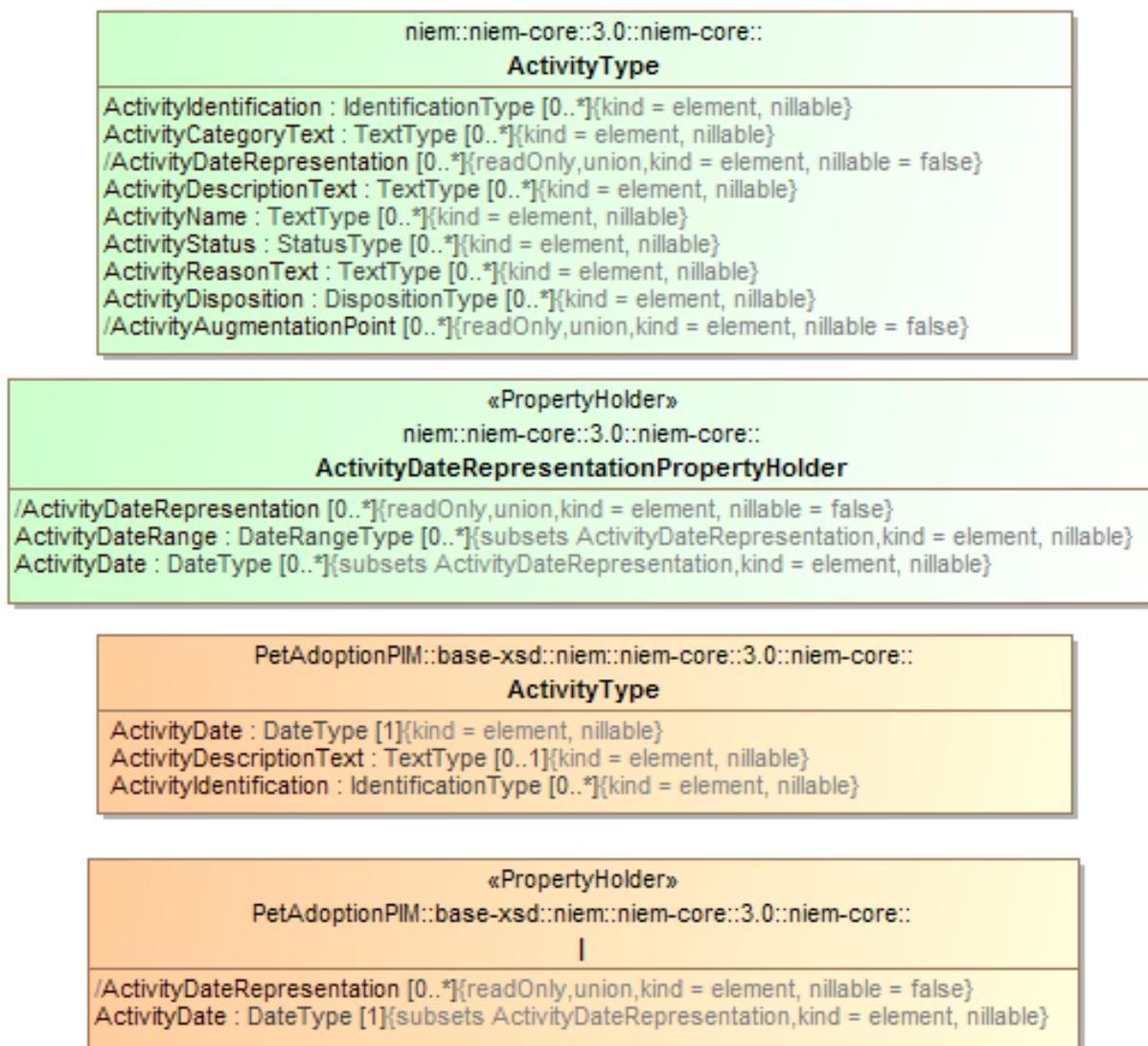


**Figure 10-19 Contact Information Association**

## 10.16 Pet Adoptions as a Kind of Activity

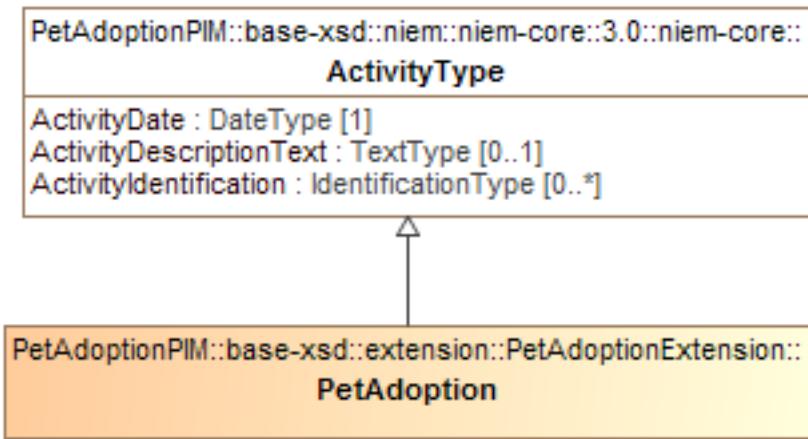
There are some more properties of a pet adoption we would like to consider. These considerations come from it being a kind of activity and activities being available in NIEM-Core.

The pattern of reuse in Figure 10-20 should look quite familiar now; we are combining the NIEM-Core concept of activity with one of the representations of that activity's date.



**Figure 10-20 Defining Activity from NIEM-Core**

Since pet adoptions are a kind of activity it makes sense for pet adoptions to be a subclass of activity. By making PetAdoption a UML subclass of ActivityType all the properties of ActivityType become available to PetAdoption – every pet adoption is an activity.



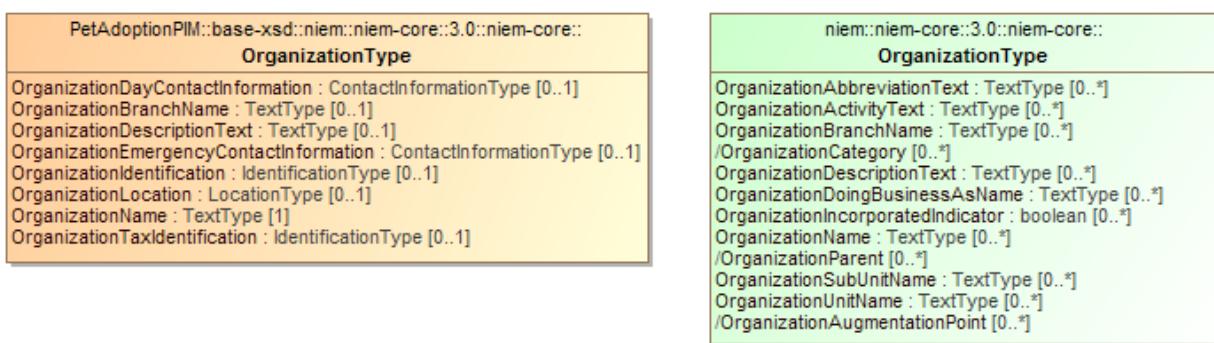
**Figure 10-21 PetAdoption as a kind of Activity**

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 section we will see how to handle other cases).

A UML subclass, or generalization, maps to an “extension” in XML Schema unless it uses RolePlayedBy (see below). So in the XML Schema, PetAdoption will extend Activity.

## 10.17 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 which we can reuse. Figure 10-22 uses the same «Subsets» pattern we have seen previously and we pick up some of the standard properties.



**Figure 10-22 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 10-23 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).

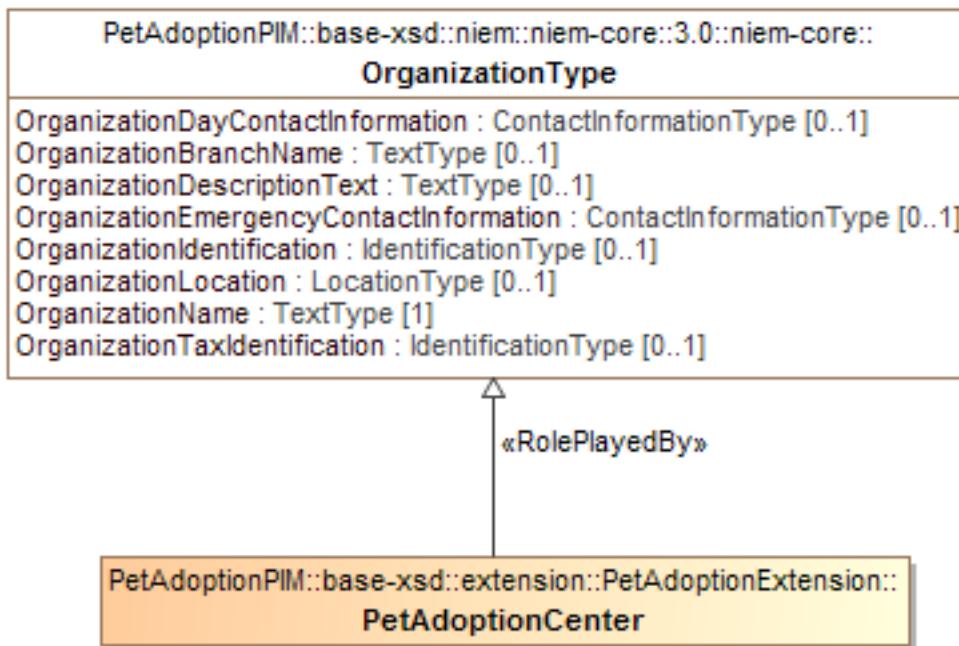


Figure 10-23 Adoption Centers as a Role of an Organization

## 10.18 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 10-24, 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.

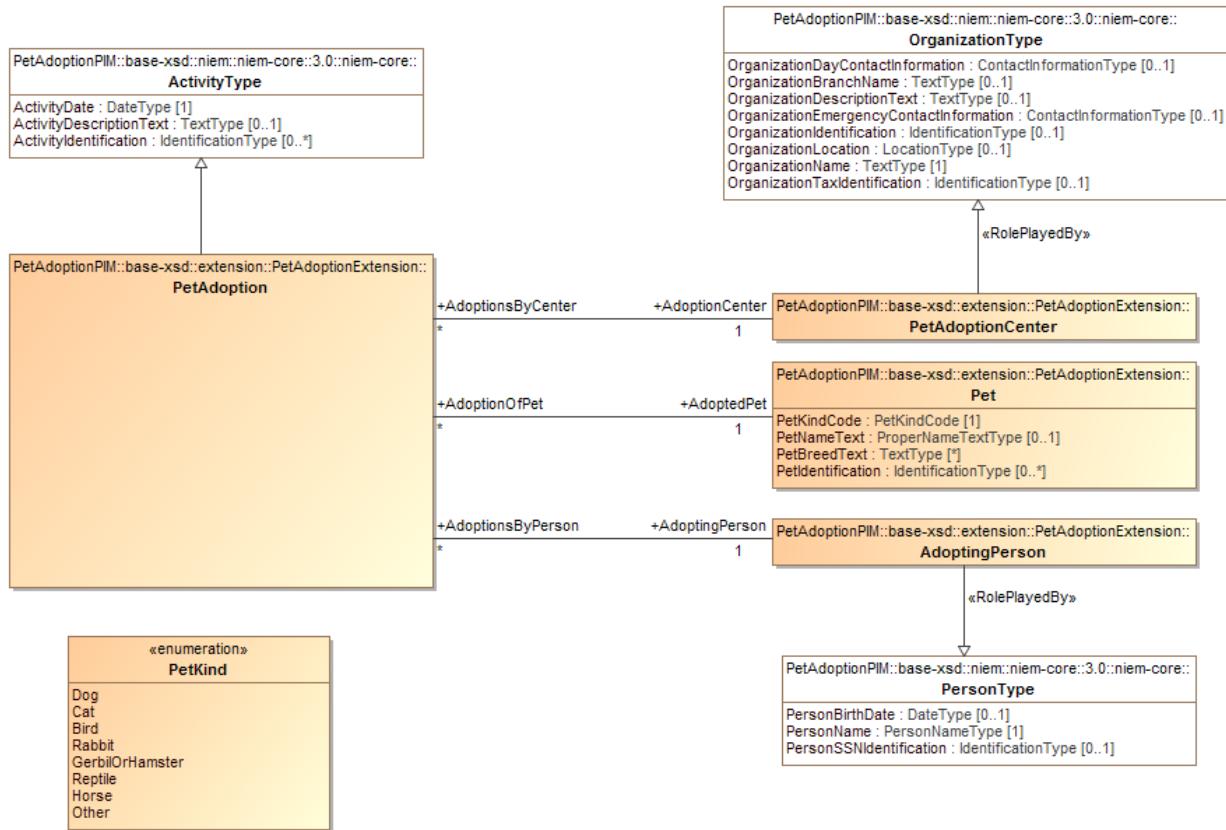


Figure 10-24 Completed High Level Picture

## 10.19 Exchange Message

In this example we gather all of the information about PetAdoptions together into a handy grab bag of pet-related data, in the expectation that this will provide a useful message to interchange between agencies. We specify this type in a different package, because its purpose – to define messages – differs from the purpose of the types defined so far. In a richer application such a package might define many different message types.

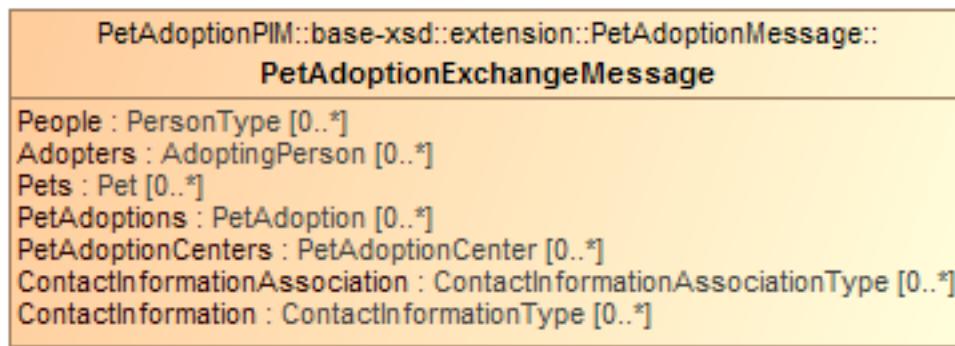
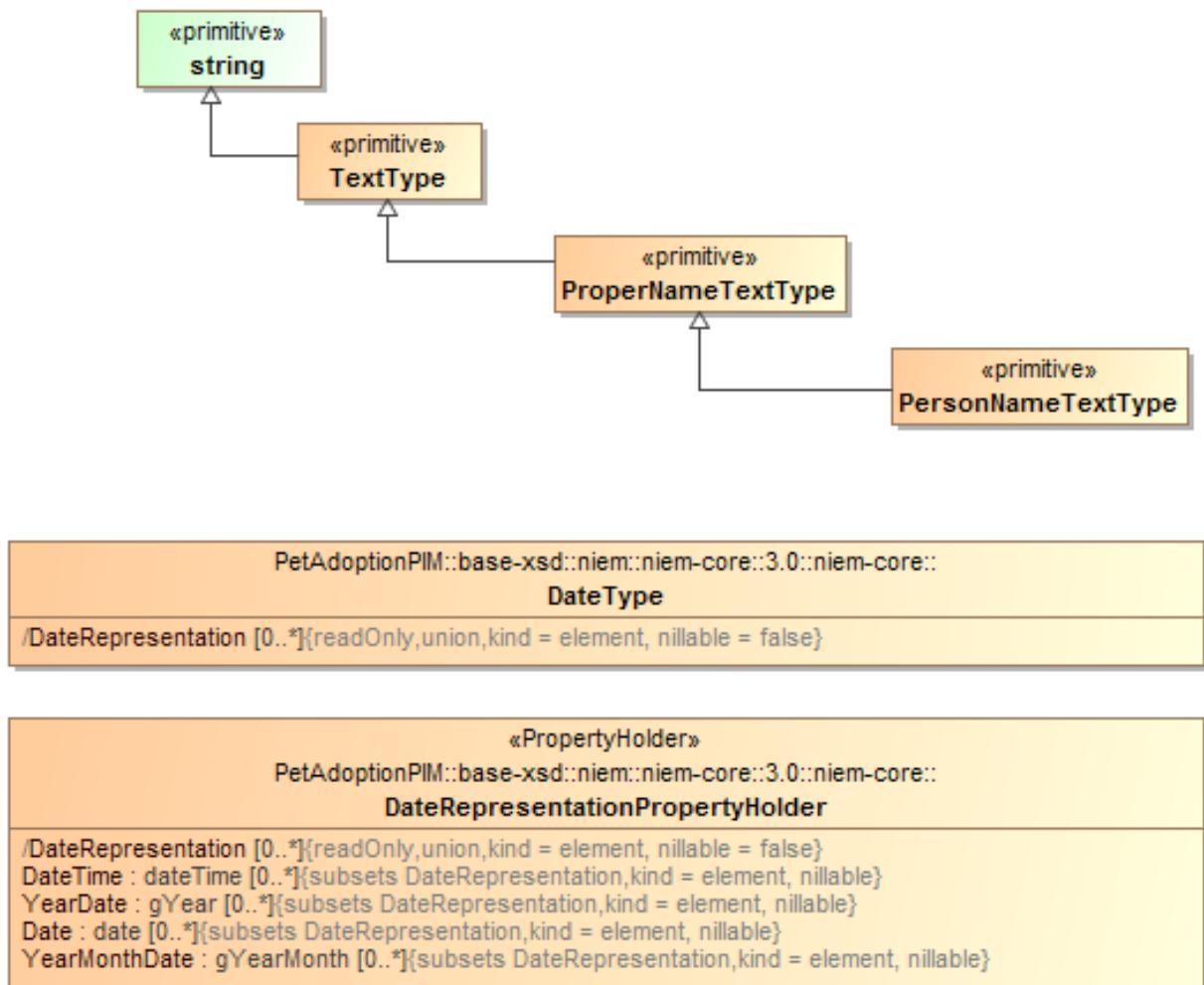


Figure 10-25 Exchange message type

## 10.20 Primitive types

We have been using properties with “primitive types” like strings numbers and dates. We have also seen some more specialized primitive types like “PersonNameText”. Where do these come from? The basic types like Strings come

from a “Primitive type library” that is standard in NIEM-UML. It is imported by using the profile and always available. More specialized primitive types are either in NIEM-Core or defined within a model as subtypes of these built-in types. The following are the primitive types used in this example model.



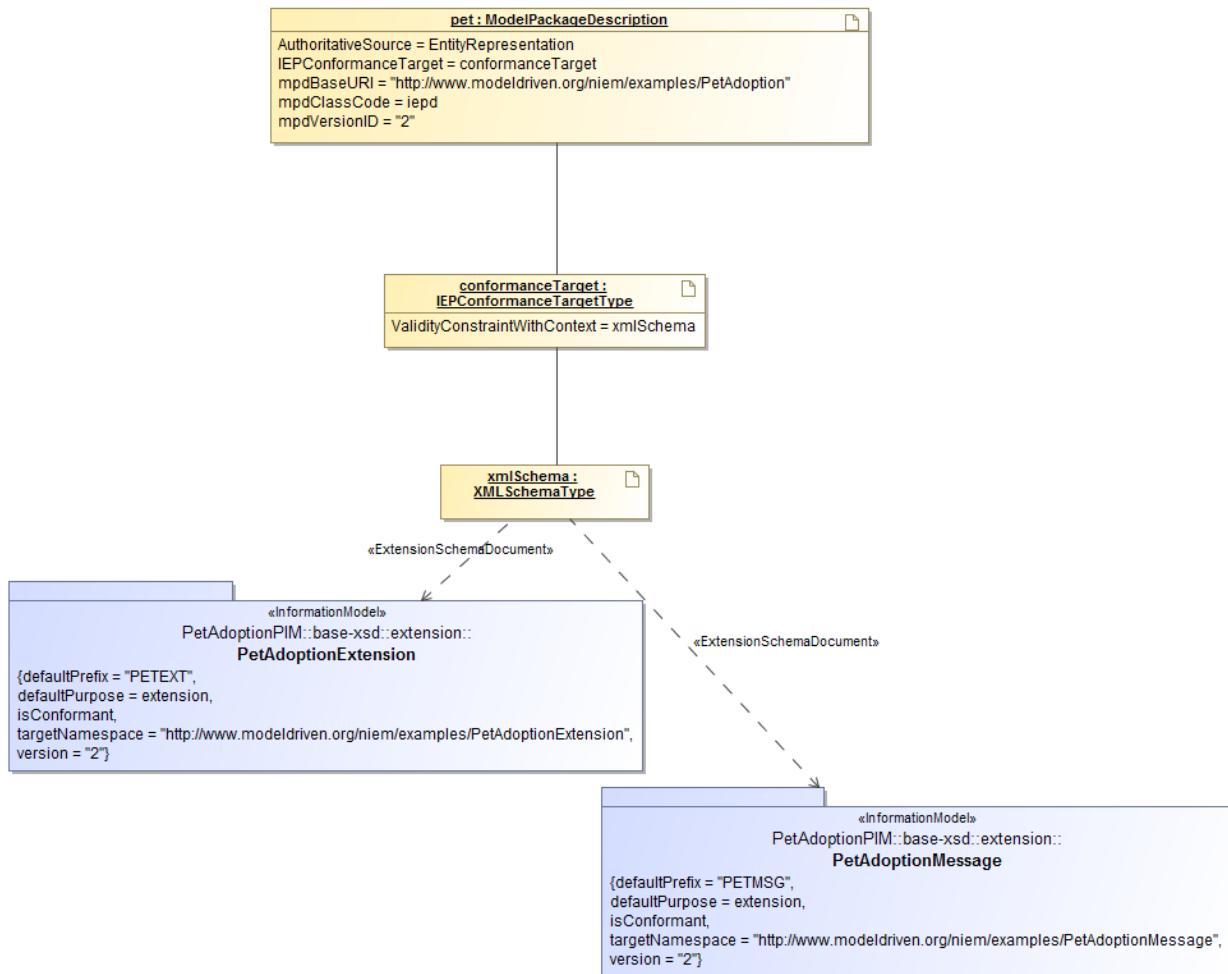
**Figure 10-26 Primitive Types**

*This concludes the platform independent part of the example.*

## 10.21 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, which generates the MPD catalog. It also references the PIM and PSM packages that contain the classes used in an exchange. Figure 10-27 is part of the IEPD model for our example.



**Figure 10-27 IEPD Specification**

The instance of the ModelPackageDescription Artifact represents the IEPD. The property values of this instance define some IEPD metadata. Associated with a ModelPackageDescription (MPD) is a “conformance target”; in this case the MPD is an IEPD and so the conformance target is an IEP conformance target. Each conformance target specifies how its exchanges will be validated; in this case - and indeed most cases – the exchanges will be XML documents validated by XML schemas; hence the instance of the XMLSchemaType Artifact.

The “meat” of an IEPD is in the information models that relate to the xml schema artifact via «ExtensionSchemaDocument» Usages. The packages referred to in this way are those that we have been building all-along in our PIM, these are:

- PetAdoptionExtension – the new concepts defined for Pet Adoption, some of which use or extend NIEM-Core concepts.
- PetAdoptionMessage – the messages which agencies can use to communicate about pet adoptions.

Note that each information model has a defaultPurpose that corresponds to the NIEM schema types.

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

Refer back to Figure 10-1 to see how these packages relate to the NIEM Core definition and its pet adoption subset.

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.

# Machine Readable Artifacts

## A.1 Normative

NIEM-UML-3 includes several UML models, the normative XMI for which may be referenced using the following standard URIs:

### Annex A

- *NIEM UML Profile*

<http://www.omg.org/spec/NIEM-UML/20150201/NIEM-UML-Profile.xmi>

- *XML Primitive Types Library*

<http://www.omg.org/spec/NIEM-UML/20150201/XMLPrimitiveTypes.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/20150201/NIEM-Reference/>

The files available at that URL are the following:

NIEM-Reference-adapters-edxl-cap.xmi  
NIEM-Reference-adapters-edxl-de.xmi  
NIEM-Reference-adapters-edxl-have.xmi  
NIEM-Reference-adapters-geospatial.xmi  
NIEM-Reference-codes-ansi\_d20.xmi  
NIEM-Reference-codes-apco\_event.xmi  
NIEM-Reference-codes-atf.xmi  
NIEM-Reference-codes-canada\_post.xmi  
NIEM-Reference-codes-cbrncl.xmi  
NIEM-Reference-codes-census\_commodity.xmi  
NIEM-Reference-codes-census\_uscounty.xmi  
NIEM-Reference-codes-core\_misc.xmi  
NIEM-Reference-codes-dea\_ctlsub.xmi  
NIEM-Reference-codes-dod\_jcs-pub2.0.xmi  
NIEM-Reference-codes-dol\_soc.xmi  
NIEM-Reference-codes-dot\_hazmat.xmi  
NIEM-Reference-codes-edxl\_have.xmi  
NIEM-Reference-codes-edxl\_rm.xmi  
NIEM-Reference-codes-fbi\_ncic.xmi  
NIEM-Reference-codes-fbi\_ndex.xmi  
NIEM-Reference-codes-fbi\_ucr.xmi  
NIEM-Reference-codes-fips\_10-4.xmi  
NIEM-Reference-codes-fips\_5-2.xmi  
NIEM-Reference-codes-fips\_6-4.xmi  
NIEM-Reference-codes-hl7.xmi  
NIEM-Reference-codes-iso\_3166-1.xmi  
NIEM-Reference-codes-iso\_4217.xmi  
NIEM-Reference-codes-iso\_639-3.xmi  
NIEM-Reference-codes-it\_codes.xmi  
NIEM-Reference-codes-mmucc.xmi  
NIEM-Reference-codes-nga\_datum.xmi  
NIEM-Reference-codes-nga\_genc.xmi  
NIEM-Reference-codes-nga\_vdatum.xmi  
NIEM-Reference-codes-nlets.xmi  
NIEM-Reference-codes-occ\_facility.xmi  
NIEM-Reference-codes-pmisse\_sar.xmi  
NIEM-Reference-codes-unece\_rec20.xmi  
NIEM-Reference-codes-usps\_states.xmi  
NIEM-Reference-codes-xCard.xmi  
NIEM-Reference-domains-biometrics.xmi

NIEM-Reference-domains-emergencyManagement.xmi  
NIEM-Reference-domains-infrastructureProtection.xmi  
NIEM-Reference-domains-screening.xmi  
NIEM-Reference-external-cap.xmi  
NIEM-Reference-external-de.xmi  
NIEM-Reference-external-have.xmi  
NIEM-Reference-external-ogc.xmi  
NIEM-Reference-external-xml.xmi  
NIEM-Reference-niem-core.xmi

The NIEM-UML-Profile.xmi file contains the overall NIEM\_UML\_Profile, the NIEM-Common\_Profile, and the three sub-profiles, as specified in Clause 8.1. Each of these have specified namespace prefixes and URIs. 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/20150201>
- *NIEM\_Common\_Profile*  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEM\\_Common\\_Profile](http://www.omg.org/spec/NIEM-UML/20150201/NIEM_Common_Profile)
- *NIEM\_PIM\_Profile*  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEM\\_PIM\\_Profile](http://www.omg.org/spec/NIEM-UML/20150201/NIEM_PIM_Profile)
- *NIEM\_PSM\_Profile*  
[http://www.omg.org/spec/NIEM-UML/20150201/NIEM\\_PSM\\_Profile](http://www.omg.org/spec/NIEM-UML/20150201/NIEM_PSM_Profile)
- *Model\_Package\_Description\_Profile*  
[http://www.omg.org/spec/NIEM-UML/20150201/Model\\_Package\\_Description\\_Profile](http://www.omg.org/spec/NIEM-UML/20150201/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/20150201/NIEMPIM2PSM.qvto>  
Dependencies: <http://www.omg.org/spec/UML/20131001>
- *NIEM PSM to NIEM-Conforming XML Schema*  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMPSM2XSD.qvto>  
Dependencies: <http://www.omg.org/spec/UML/20131001>,  
<http://www.eclipse.org/xsd/2002/XSD>
- *NIEM MPD Model to NIEM MPD Artifact*  
<http://www.omg.org/spec/NIEM-UML/20150201/NIEMMPDModel2Artifact.qvto>  
Dependencies: <http://www.omg.org/spec/UML/20131001>,  
<http://www.eclipse.org/xsd/2002/XSD>,  
<http://release.niem.gov/niem/proxy/xsd/3.0/>,  
<http://reference.niem.gov/niem/resource/mpd/catalog/3.0/>,  
<http://reference.niem.gov/niem/resource/mpd/changelog/1.1/>,  
<http://niem.gov/niem/wantlist/2.2>,  
<http://release.niem.gov/niem/niem-core/3.0/>,  
<urn:oasis:names:tc:entity:xmlns:catalog>

- *NIEM MPD Artifact to NIEM MPD Model*

<http://www.omg.org/spec/NIEM-UML/20150201/NIEMmpdartifact2model.qvto>

Dependencies: <http://www.omg.org/spec/UML/20131001>,  
<http://www.eclipse.org/xsd/2002/XSD>

These transformations in turn use the following common transformations:

- *NIEM Globals*

<http://www.omg.org/spec/NIEM-UML/20150201/NIEMglobals.qvto>

Dependencies: <http://www.omg.org/spec/UML/20131001>,  
<http://www.eclipse.org/xsd/2002/XSD>

- *NIEM Platform Binding*

<http://www.omg.org/spec/NIEM-UML/20150201/NIEMplatformBinding.qvto>

Dependencies: <http://www.omg.org/spec/UML/20131001>,  
<http://www.eclipse.org/xsd/2002/XSD>,  
<http://release.niem.gov/niem/proxy/xsd/3.0/>,  
<http://reference.niem.gov/niem/resource/mpd/catalog/3.0/>,  
<http://reference.niem.gov/niem/resource/mpd/changelog/1.1/>,  
<http://release.niem.gov/niem/niem-core/3.0/>

The namespace <http://www.eclipse.org/xsd/2002/XSD> is a published model interface for the Xml Schema InfoSet model described in clause 10 of [XMI].

Other dependencies that are not OMG specifications are all published XML Schemas. Each of these is accessed by the QVT via an implied derived MOF model that has the capability to serialize XML documents that validate against the schema. In this model each complex type corresponds by name to a Class, each simple type corresponds by name to a DataType, and each element or attribute corresponds to a Property of the corresponding type.

The NIEM schemas may be found at [NIEM-3].

The OASIS catalog schema may be found at <https://www.oasis-open.org/committees/entity/spec-2001-08-06.html>.

## A.2 Informative

The following artifacts are MagicDraw files that may help with understanding and maintenance of this specification.

- *MagicDraw implementation of NIEM-UML profile*

<http://www.omg.org/spec/NIEM-UML/20150201/NIEM-UML-Profile.mdzip>

- *MagicDraw implementation of NIEM-UML libraries*

<http://www.omg.org/spec/NIEM-UML/20150201/ModelLibraryMD.zip>

- *MagicDraw implementation of PetAdoption example*

<http://www.omg.org/spec/NIEM-UML/20150201/PetExample.mdzip>

- *MagicDraw implementation of profile and libraries with improved diagrams*

<http://www.omg.org/spec/NIEM-UML/3.0/ModelsWithImprovedDiagrams.zip>