|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | 75 | Date: | October 2014 | |
| Archetype Modeling Language (AML)  Version: 0.5  **OMG Document Number: formal/2014-10-21**  **Standard document URL: http://www.omg.org/spec/AML/1.0**  Original File: N/A |

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&apos;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 logo©, XMI Logo©, CWM©, CWM Logo©, IIOP© , MOF© , OMG Interface Definition Language (IDL)© , and 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.

**OMG's Issue Reporting Procedure**

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

**Acknowledgements**

The following individuals submitted parts of this specification and/or have assisted the AML team in the development of the specification:

Table of Contents

1 5

2 6

2.1 6

2.2 6

2.2.1 6

2.2.2 6

**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. A Specifications Catalog is available from the OMG website at:

*http://www.omg.org/technology/documents/spec\_catalog.htm*

Specifications within the Catalog are organized by the following categories:

**OMG Modeling Specifications**

• UML  
• MOF  
• XMI  
• CWM  
• Profile specifications

**OMG Middleware Specifications**

• CORBA/IIOP  
• IDL/Language Mappings  
• Specialized CORBA specifications  
• CORBA Component Model (CCM)

**Platform Specific Model and Interface Specifications**

• CORBAservices  
• CORBAfacilities  
• OMG Domain specifications  
• OMG Embedded Intelligence specifications  
• OMG Security specifications

OMG Headquarters   
 109 Highland Ave,   
 Needham, MA 02494 USA  
 USA   
   
 Tel: +1-781-444-0404   
 Fax: +1-781-444-0320   
 Email: pubs@omg.org  
   
Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

**Typographical Conventions**

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or 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.

# Scope

# Conformance

# Normative References

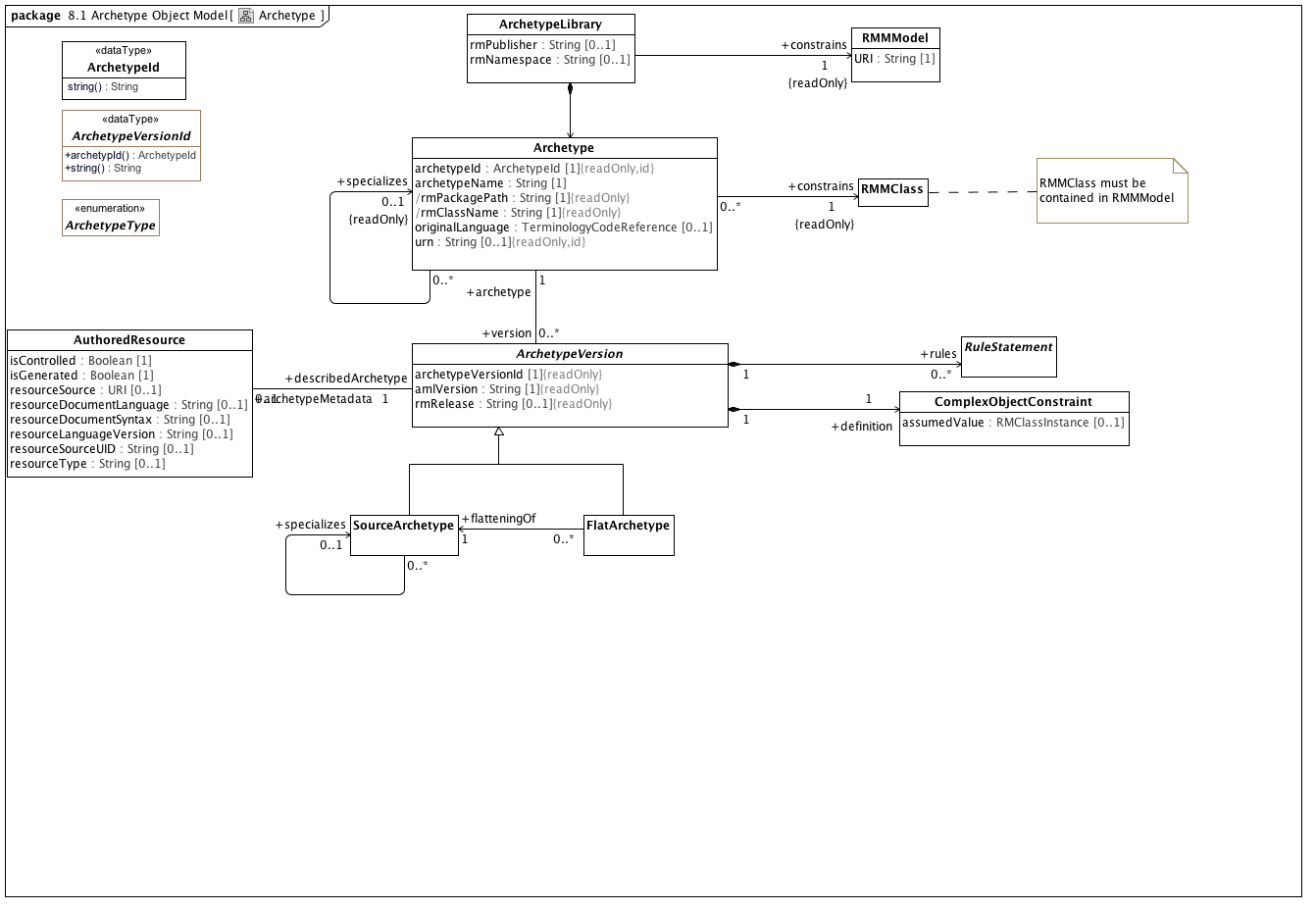
# Terms and Definitions

# Symbols

# Additional Information

# AML Object Model

## <Package> Archetype Object Model



**Archetype**

The Archetype package, showing the compositions rules, definition, archetypeId, and terminology; the concrete kinds of archetypes; and lineage by the recursive parent relationship.

### <DataType> ArchetypeId

**Description**

An artifact that uniquely identifies an *Archetype* within a given community of use. The actual syntax and structure of the *ArchetypId* type should be established by a community of use, but all *ArchetypeId* implementations must support a *String* representation.

**Operations**

• string () : [String](#_e8a6ce315d976318da3ab784a645ea44)

The stringified representation of the identifier.

### <DataType> ArchetypeVersionId

**Operations**

• public archetypId () : [ArchetypeId](#_abe68de6d7b599f5e4ea361caee12c81)

• public string () : [String](#_e8a6ce315d976318da3ab784a645ea44)

### <Class> Archetype

**Attributes**

• public archetypeId : [ArchetypeId](#_abe68de6d7b599f5e4ea361caee12c81) [1]

The unique archetype identifier. However constructed, this uniquely identifies the archetype across its entire life cycle. No other archetype can have this identifier and a different identifier designates a different Archetype.

• public archetypeName : String [1]

The human readable name of the Archetype. This is typically derived from the other archetype details (See: openEHR Knowledge Artefact Identification - Revision 0.7.0 for an example). It is possible for this identifier to change over the life of an Archetype.

• public rmPackagePath : String [1]

The qualifiedName of a package in the target reference model that has the root *rmClass* as a visible member (there can be more than one possibility in a reference model).

• public rmClassName : String [1]

Name of the root class of this archetype. *rmClass* must match the *visibleName* of the class referenced by the *ComplexObjectConstraint* target of the *Archetype definition* as well as the visibleName of the RMMClass instance that it constrains.

• public originalLanguage : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

The original language in which the resource was authored (essential for evaluating natural language quality)

• public urn : String [0..1]

**Associations**

• public rmUMLModel : [RMMModel](#_fc116c5fcb379006ed51eb855a1dae57) [1]

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

• public rmPackage : [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4) [1]

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

• public version : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

• public specializes : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• uri

[OCL]

rmURI = rmUMLModel.URI

• package

[OCL]

rmPackagePath = rmPackage.qualifiedName

• class

[OCL]

rmClassName = constrains.name

• classpackage

[OCL]

rmPackage.member->exists(c|c=constrains)

• differentArchetype

An archetype cannot specialize itself or any of its descendants (note - only self is included in formal OCL)

[OCL2.0]

not (self = self.specializes)

### <Class> ArchetypeLibrary

**Description**

A collection of archetypes that apply to the same reference model.

**Attributes**

• public rmPublisher : String [0..1]

• public rmNamespace : String [0..1]

**Associations**

• public : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75)

• public constrains : [RMMModel](#_fc116c5fcb379006ed51eb855a1dae57) [1]

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

### <Class> ArchetypeVersion

**Description**

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Direct Known Subclasses (Specialization)**

[FlatArchetype](#_166addccec3fe63279b3f9eeb9930ab8), [SourceArchetype](#_dcaf9716e9bc2255b93c20393e8712f2)

**Attributes**

• public archetypeVersionId [1]

The specific version of this *Archetype*. *archetypeVersion* does not impact archetype identity. If an archetype undergoes non-backwards compatible changes, it becomes a new archetype with a new identifier.

• public amlVersion : String [1]

The URI of the modeling language and version used to construct this Archetype, if derived from a serialized representation

• public rmRelease : String [0..1]

The specific version of the reference model that was constrained. Depending on the context and workflow model, it may be possible to update a reference model in a backwards-compatible fashion that doesn't require the referencing archetypes to be revised. *rmVersion* exists to support this particular situation and records the specific RM version that the archetype was built to constrain.

**Associations**

• public definition : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134) [1]

A constraint on a complex object, which will typically consist of other constraints

• public rules : [RuleStatement](#_f8740e8d27529166da46265bd8521c94) [0..\*]

Abstract parent of all statement types

• public archetypeMetadata : [AuthoredResource](#_47dea9d0676ad6870be946fa52e870ad) [0..1]

*AuthoredResource* carries a minimal set of information about the source and origin of an *Archetype*. Its intent is to be a "connection point" to attach additional workflow and other provenance information to the target *Archetype.*

• public archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

### <Class> AuthoredResource

**Description**

*AuthoredResource* carries a minimal set of information about the source and origin of an *Archetype*. Its intent is to be a "connection point" to attach additional workflow and other provenance information to the target *Archetype.*

**Attributes**

• public isControlled : Boolean [1]

A flag indicating whether the archetype is change-controlled or not can be included after the version. Archetypes that include the “controlled” flag should have the revision history section included, while those with the “uncontrolled” flag, or no flag at all, may omit the revision history. This enables archetypes to be privately edited in an early development phase without generating large revision histories of little or no value

• public isGenerated : Boolean [1]

A flag indicating whether the archetype was generated or authored. This marker is used to support the migration to differential archetype representation introduced in ADL 1.5, to enable proper representation of specialised archetypes.

• public resourceSource : [URI](#_887928f30f99c8a1ca89ed7a082356aa) [0..1]

A URI that references the source document (if any) from which the original resource was derived.

• public resourceDocumentLanguage : String [0..1]

The language (e.g. AOM, CEM, ...) of the source of the constraints, if any.

• public resourceDocumentSyntax : String [0..1]

The syntax of the resource document (ADL, XML, XMI, ...)

• public resourceLanguageVersion : String [0..1]

The version of the resourceDocumentLanguage (e.g. ADL 1.5, XMI 2.1, etc)

• public resourceSourceUID : String [0..1]

An external identifier that uniquely identifies this *Archetype*. The format and structure of this identifier are determined by the rules of the *resourceDocumentLanguage* and/or *resourceDocumentSyntax.* This identifier cannot be used as an identifier within AML itself as it may not always be present. It must be preserved, however, for export to external resources.

• resourceType : String [0..1]

The artefact type. This field does not impact the semantics of the representation is can be used to record specific types or classifications according to the source entity. As an example, ADL specifies artefact types of "archetype", "template", "template\_overlay" and "operational\_archetype" with an optional "flat" keyword.

**Associations**

• public describedArchetype : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [1]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

### <Class> ComplexObjectConstraint

**Description**

A constraint on a complex object, which will typically consist of other constraints

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Attributes**

• public assumedValue : [RMClassInstance](#_935cd7de4b22d47dd2c6aef93bed5c7a) [0..1]

Value to be assumed in instances in which no value is provided

**Associations**

• public attributeTuple : [AttributeTupleConstraint](#_f6da15c71717330ae1b56f8b41e3dd51) [0..\*]

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

• private targetObject : [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36) [0..\*]

A constraint defined by reference to a node defined elsewhere in the same archetype

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

**Constraints**

• instanceOfConstraint

[English]

If assumedValue exists, assumedValue.classifier must be equal to or a specialization of self.parent

### <Class> FlatArchetype

**Description**

A *FlatArchetype* is generated from one or more *SourceArchetypes* via the flattening process. The flattening operation:

Replaces *ComplexObjectConstraintProxies* with *ComplexObjectConstraints* that contain copies of the subtrees to which they point.

Applies *SourceArchetype* overlays to the parent structure resulting in a full archetype structure.

**Direct Known Superclasses (Generalization)**

[ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99)

**Associations**

• public flatteningOf : [SourceArchetype](#_dcaf9716e9bc2255b93c20393e8712f2) [1]

The source form of an archetype, potentially including references to other archetypes whose contents are not explicitly reproduced in the source form

**Constraints**

• noSpecialization

A flat archetype cannot specialize another archetype

[OCL2.0]

not exists(archetype.specializes)

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMModel

**Description**

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

**Direct Known Superclasses (Generalization)**

[RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Attributes**

• public URI : String [1]

### <Class> RuleStatement

**Description**

Abstract parent of all statement types

**Direct Known Superclasses (Generalization)**

[RuleElement](#_aca03c36f52ec3dbae79ae94604d6df9)

**Direct Known Subclasses (Specialization)**

[Assertion](#_f9e7c553caf3e674732fe386e3d45466), [VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015)

### <Class> SourceArchetype

**Description**

The source form of an archetype, potentially including references to other archetypes whose contents are not explicitly reproduced in the source form

**Direct Known Superclasses (Generalization)**

[ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99)

**Associations**

• public specializes : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

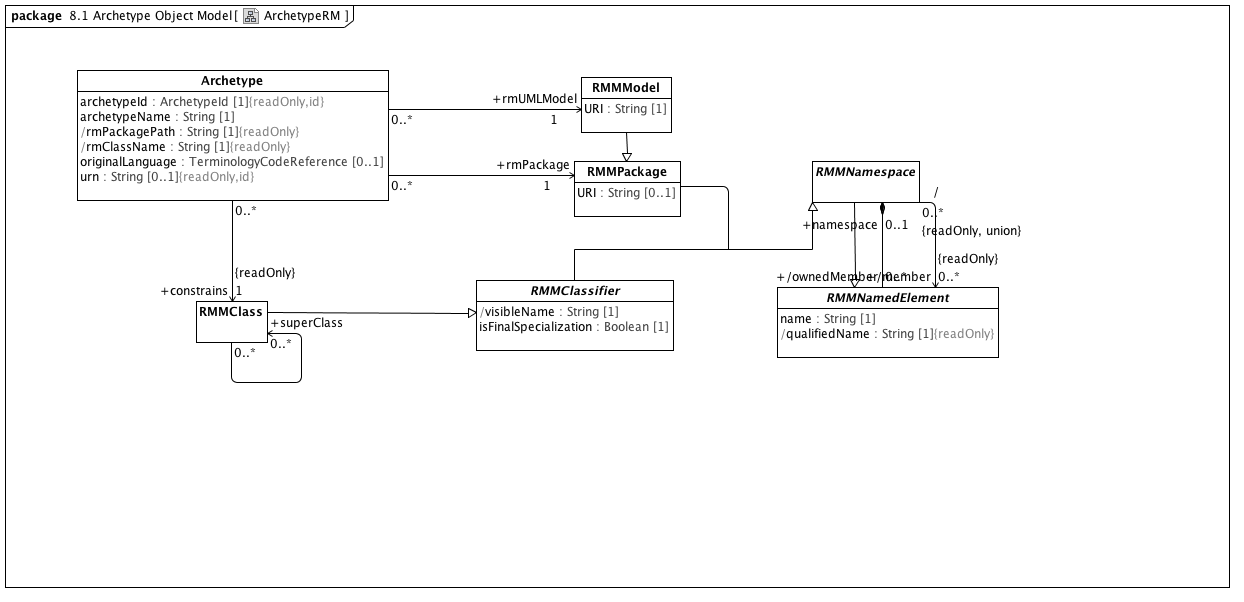
**Constraints**

• sameArchetypes

[OCL2.0]

exists(self.specializes) implies self.archetype.specializes = self.specializes.archetype.specializes

### <Enumeration> ArchetypeType



**ArchetypeRM**

An Archetype references (or constrains) a single class in a UML Reference Model. The rmURI identifies the particular model, the rmPackage names the path to the (a) package that has the constrained class as a member, and the rmClassName identifies the particular class being constrained.

### <Class> Archetype

**Attributes**

• public archetypeId : [ArchetypeId](#_abe68de6d7b599f5e4ea361caee12c81) [1]

The unique archetype identifier. However constructed, this uniquely identifies the archetype across its entire life cycle. No other archetype can have this identifier and a different identifier designates a different Archetype.

• public archetypeName : String [1]

The human readable name of the Archetype. This is typically derived from the other archetype details (See: openEHR Knowledge Artefact Identification - Revision 0.7.0 for an example). It is possible for this identifier to change over the life of an Archetype.

• public rmPackagePath : String [1]

The qualifiedName of a package in the target reference model that has the root *rmClass* as a visible member (there can be more than one possibility in a reference model).

• public rmClassName : String [1]

Name of the root class of this archetype. *rmClass* must match the *visibleName* of the class referenced by the *ComplexObjectConstraint* target of the *Archetype definition* as well as the visibleName of the RMMClass instance that it constrains.

• public originalLanguage : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

The original language in which the resource was authored (essential for evaluating natural language quality)

• public urn : String [0..1]

**Associations**

• public rmUMLModel : [RMMModel](#_fc116c5fcb379006ed51eb855a1dae57) [1]

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

• public rmPackage : [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4) [1]

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

• public version : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

• public specializes : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• uri

[OCL]

rmURI = rmUMLModel.URI

• package

[OCL]

rmPackagePath = rmPackage.qualifiedName

• class

[OCL]

rmClassName = constrains.name

• classpackage

[OCL]

rmPackage.member->exists(c|c=constrains)

• differentArchetype

An archetype cannot specialize itself or any of its descendants (note - only self is included in formal OCL)

[OCL2.0]

not (self = self.specializes)

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMModel

**Description**

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

**Direct Known Superclasses (Generalization)**

[RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Attributes**

• public URI : String [1]

### <Class> RMMNamedElement

**Description**

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

**Direct Known Subclasses (Specialization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22), [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b), [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

**Attributes**

• public name : String [1]

The name of the Reference Model element. Name must be unique within the context of the owning *namespace*

• public qualifiedName : String [1]

**Associations**

• public namespace : [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b) [0..1]

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

### <Class> RMMNamespace

**Description**

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873), [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Associations**

• public member : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

• public ownedMember : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

### <Class> RMMPackage

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMModel](#_fc116c5fcb379006ed51eb855a1dae57)

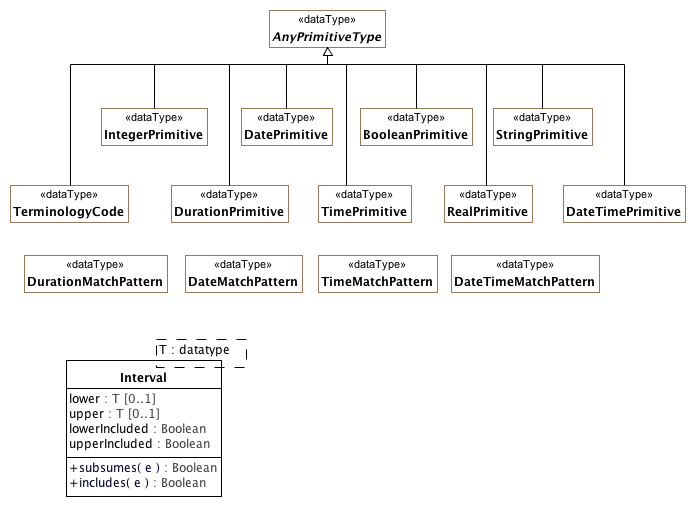
**Attributes**

• public URI : String [0..1]

## <Package> Reference Object Model

### <Package> Primitive Data Types

In the AML/ADL context, the term "primitive data type" is used to indicate "leaf nodes" -- data elements that are treated as being atomic and are only constrained in terms of their possible value ranges. AML data types, like UML data types, are "model Types whose instances are distinguished only by their value" but, unlike the UML definition of "Primitive Type", AML primitive types can embody the notion of substructure. When a AML profile is applied to a UML Reference Model, it may be necessary to map one or more of the AML types to corresponding types in the target model.



**PrimitiveDataTypes**

The set of primitive data types that can appear on a reference model and can be constrained using AML primitive constraints.

### <DataType> AnyPrimitiveType

**Description**

An abstract supertype of data types. This type usually maps to a type like "Any" or "Object" in an object system; it is defined here to provide the value and reference equality semantics.

**Direct Known Subclasses (Specialization)**

[BooleanPrimitive](#_19737cdaaaee2179c50553b52361808b), [DatePrimitive](#_db5d020506d0af7a330f0b4fe1cb870a), [DateTimePrimitive](#_8347879bb381db17040637cc3ba0a25c), [DurationPrimitive](#_64b0026498682fb721ccdb38c186bba2), [IntegerPrimitive](#_89e2b2b9de405e6d05c4c5259fc8ffd6), [RealPrimitive](#_c596f10fb93cb4f697f2f1b0b64b43ce), [StringPrimitive](#_6a90be7cfa784ea4b4e8cad8f4a47e82), [TerminologyCode](#_3828b4174a837b16c5b4b5dcadf76a23), [TimePrimitive](#_05e0f2a221f0733d693058e9253b0017)

### <DataType> BooleanPrimitive

**Description**

A primitive type with two values supporting binary logic

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> DateMatchPattern

**Description**

A pattern supporting the specification of dates by partial match

### <DataType> DatePrimitive

**Description**

A primitive type specifying a calendar date

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> DateTimeMatchPattern

**Description**

A pattern supporting the specification of dateTimes by partial match

### <DataType> DateTimePrimitive

**Description**

A primitive type specifying a character string

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> DurationMatchPattern

**Description**

A pattern supporting the specification of durations by partial match

### <DataType> DurationPrimitive

**Description**

A primitive type specifying a quantity of time

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> IntegerPrimitive

**Description**

A primitive type specifying an integer

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> RealPrimitive

**Description**

A primitive type specifying a real number

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> StringPrimitive

**Description**

A primitive type specifying a character string

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> TerminologyCode

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <DataType> TimeMatchPattern

**Description**

A pattern supporting the specification of times by partial match

### <DataType> TimePrimitive

**Description**

A primitive type specifying a point in time

**Direct Known Superclasses (Generalization)**

[AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12)

### <Class> Interval

**Attributes**

• public lower : [T](#_e36b5e306e1664a3e0bc600785109705) [0..1]

• public upper : [T](#_e36b5e306e1664a3e0bc600785109705) [0..1]

• lowerIncluded : Boolean

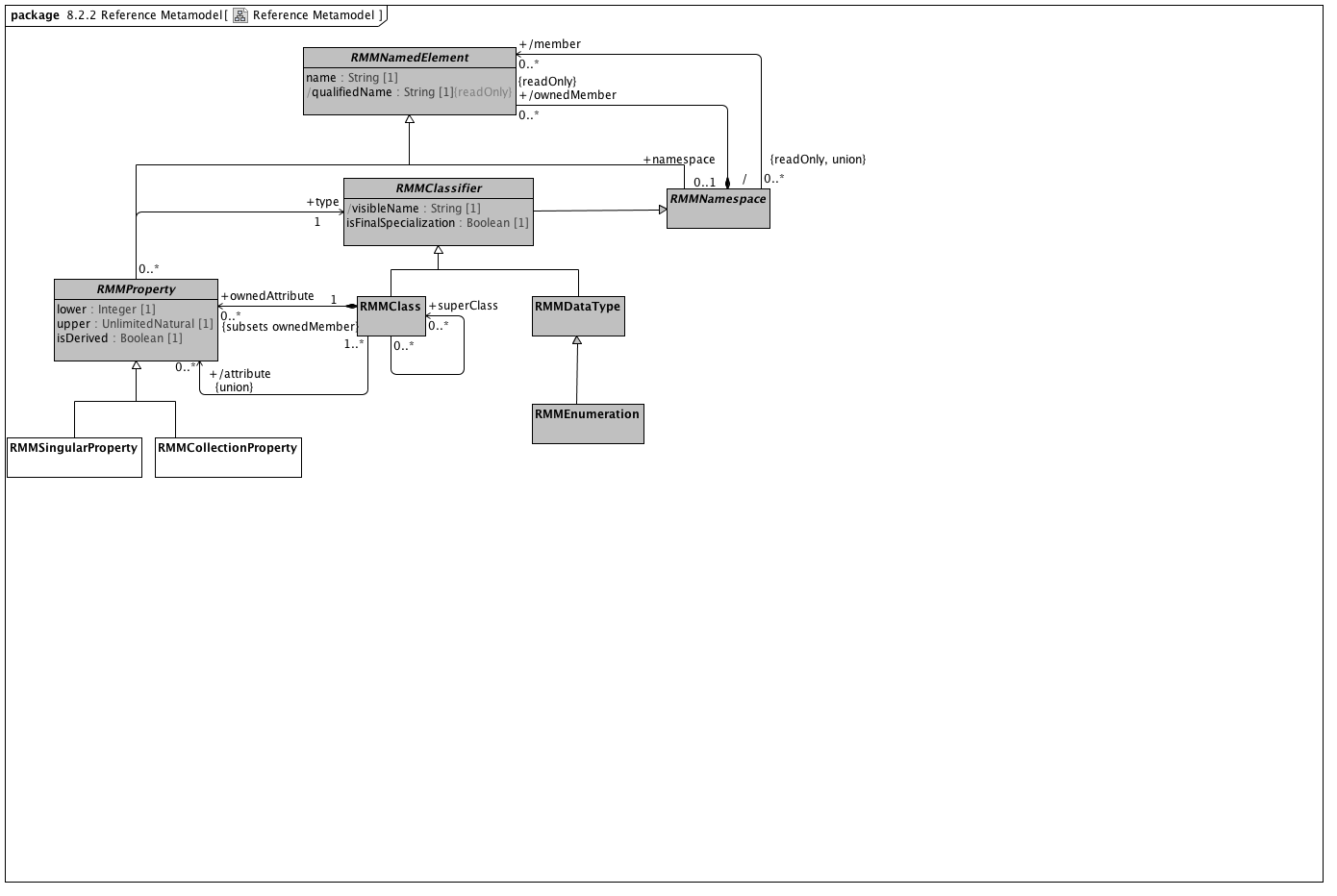
• public upperIncluded : Boolean

**Operations**

• public subsumes (e) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

• public includes (e) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

### <Package> Reference Metamodel



**Reference Metamodel**

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMCollectionProperty

**Description**

RMMCollectionProperty represents the subset of RMMProperty instances that can occur more than one time. An RMMCollectionProperty instance is viewed by AML as a collection of objects of a given type that possess two separate characteristics:

The collection as a whole may be required, optional or prohibited.

The cardinality of the collection may be constrained.

This combination allows a number of useful constructs, including:

requiring that a list be present but that it have no members, which can be used to assert a relationship between an object and an empty set of objects

making an attribute optional, but, if present, requiring that it have a minimum number of members

**Direct Known Superclasses (Generalization)**

[RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

### <Class> RMMDataType

**Description**

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Direct Known Subclasses (Specialization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

### <Class> RMMEnumeration

**Description**

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Direct Known Superclasses (Generalization)**

[RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51)

**Associations**

• public ownedLiteral : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Constraints**

• enumerateValueDomain

[]

### <Class> RMMNamedElement

**Description**

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

**Direct Known Subclasses (Specialization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22), [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b), [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

**Attributes**

• public name : String [1]

The name of the Reference Model element. Name must be unique within the context of the owning *namespace*

• public qualifiedName : String [1]

**Associations**

• public namespace : [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b) [0..1]

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

### <Class> RMMNamespace

**Description**

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873), [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Associations**

• public member : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

• public ownedMember : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

### <Class> RMMProperty

**Description**

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMCollectionProperty](#_702e30cd0381b6e9726bcc6fe779a70f), [RMMSingularProperty](#_5917d2795b1a9ae4f33929e6edb8af81)

**Attributes**

• private lower : Integer [1]

• private upper : UnlimitedNatural [1]

• public isDerived : Boolean [1]

**Associations**

• public type : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Constraints**

• nonDerived

[OCL]

isDerived=false

### <Class> RMMSingularProperty

**Description**

*RMMSingularProperty* represents the subset of *RMMProperty* instances having an upper bound of 1. The AML treats RMMSingular properties as single values (vs. collections) of attributes that can be required, optional or prohibited.

**Direct Known Superclasses (Generalization)**

[RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

**Constraints**

• singular

[OCL]

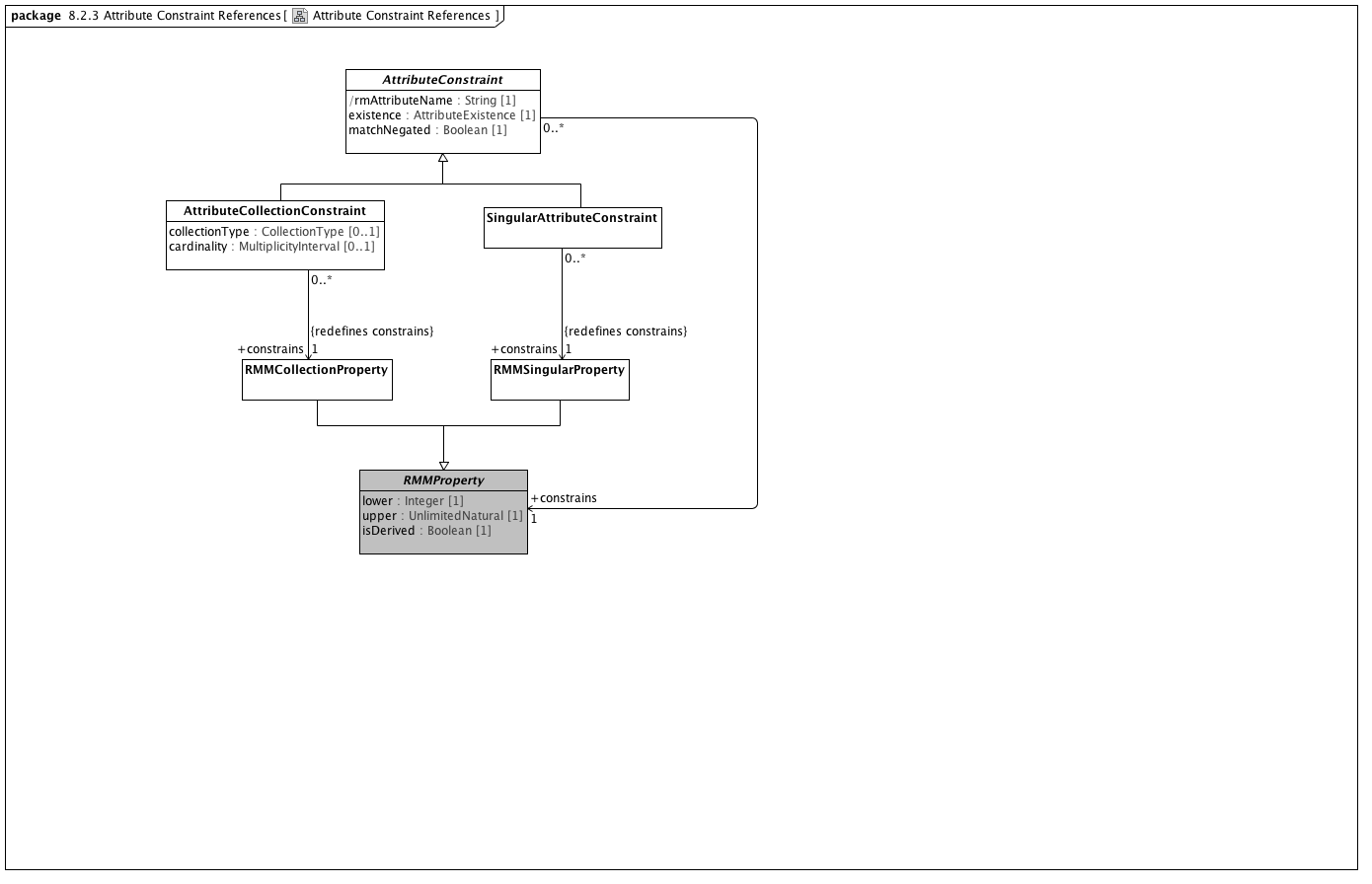
upper = 1

• collection

[OCL]

upper > 1 or upper.isUnlimited()

### <Package> Attribute Constraint References



**Attribute Constraint References**

### <Class> AttributeCollectionConstraint

**Description**

A constraint on a set of objects contained in an attribute

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05)

**Attributes**

• public collectionType : [CollectionType](#_ac8c7771bf7f68d6747022e6924749ca) [0..1]

A classification of collection indicating whether its members must be unique or ordered

• public cardinality : [MultiplicityInterval](#_c810ec7fa381fa249b7a7d9fecae85b6) [0..1]

The range of quantities of members that can be included in an attribute

**Associations**

• public member : [AttributeCollectionMember](#_177e37623ae3f5642980fd445bf78af1) [1..\*]

An association that matches members of a collection of attributes with specific ObjectConstraints

• public constrains : [RMMCollectionProperty](#_702e30cd0381b6e9726bcc6fe779a70f) [1]

RMMCollectionProperty represents the subset of RMMProperty instances that can occur more than one time. An RMMCollectionProperty instance is viewed by AML as a collection of objects of a given type that possess two separate characteristics:

The collection as a whole may be required, optional or prohibited.

The cardinality of the collection may be constrained.

This combination allows a number of useful constructs, including:

requiring that a list be present but that it have no members, which can be used to assert a relationship between an object and an empty set of objects

making an attribute optional, but, if present, requiring that it have a minimum number of members

**Constraints**

• collectionType

[OCL]

parentProperty.isOrdered implies collectionType = CollectionType::LIST and parentProperty.isUnique implies collectionType = CollectionType.SET

• cardinality

[OCL]

cardinality.minimum >= parent.lower and (cardinality.maximum = unlimitedValue() or parent.upper = unlimitedValue() or cardinality.maximum <= parent.upper)

### <Class> AttributeConstraint

**Description**

A constraint on a reference model attribute

**Direct Known Subclasses (Specialization)**

[AttributeCollectionConstraint](#_5eefba8eca7402f09bd5619804038771), [SingularAttributeConstraint](#_48ee2586ffa14e5bb1cf8ad893969da7)

**Attributes**

• public rmAttributeName : String [1]

Name of attribute within the reference model that is constrained by this node

• public existence : [AttributeExistence](#_4f99fbfcf9617d7ad55eca111d84fb67) [1]

Strength of requirement that the attribute instance be present

• public matchNegated : Boolean [1]

Whether the match operator is to be inverted so that the constraint specifies anything except what is represented

**Associations**

• public attribute : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134)

A constraint on a complex object, which will typically consist of other constraints

• public constrains : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public parent : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public object : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Constraints**

• name

[OCL]

rmAttributeName = parentProperty.name

### <Class> RMMCollectionProperty

**Description**

RMMCollectionProperty represents the subset of RMMProperty instances that can occur more than one time. An RMMCollectionProperty instance is viewed by AML as a collection of objects of a given type that possess two separate characteristics:

The collection as a whole may be required, optional or prohibited.

The cardinality of the collection may be constrained.

This combination allows a number of useful constructs, including:

requiring that a list be present but that it have no members, which can be used to assert a relationship between an object and an empty set of objects

making an attribute optional, but, if present, requiring that it have a minimum number of members

**Direct Known Superclasses (Generalization)**

[RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

### <Class> RMMProperty

**Description**

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMCollectionProperty](#_702e30cd0381b6e9726bcc6fe779a70f), [RMMSingularProperty](#_5917d2795b1a9ae4f33929e6edb8af81)

**Attributes**

• private lower : Integer [1]

• private upper : UnlimitedNatural [1]

• public isDerived : Boolean [1]

**Associations**

• public type : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Constraints**

• nonDerived

[OCL]

isDerived=false

### <Class> RMMSingularProperty

**Description**

*RMMSingularProperty* represents the subset of *RMMProperty* instances having an upper bound of 1. The AML treats RMMSingular properties as single values (vs. collections) of attributes that can be required, optional or prohibited.

**Direct Known Superclasses (Generalization)**

[RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

**Constraints**

• singular

[OCL]

upper = 1

• collection

[OCL]

upper > 1 or upper.isUnlimited()

### <Class> SingularAttributeConstraint

**Description**

An AttributeConstraint that identifies valid values for a single value instance

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05)

**Associations**

• public alternative : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

• public excludes : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

• public constrains : [RMMSingularProperty](#_5917d2795b1a9ae4f33929e6edb8af81) [1]

*RMMSingularProperty* represents the subset of *RMMProperty* instances having an upper bound of 1. The AML treats RMMSingular properties as single values (vs. collections) of attributes that can be required, optional or prohibited.

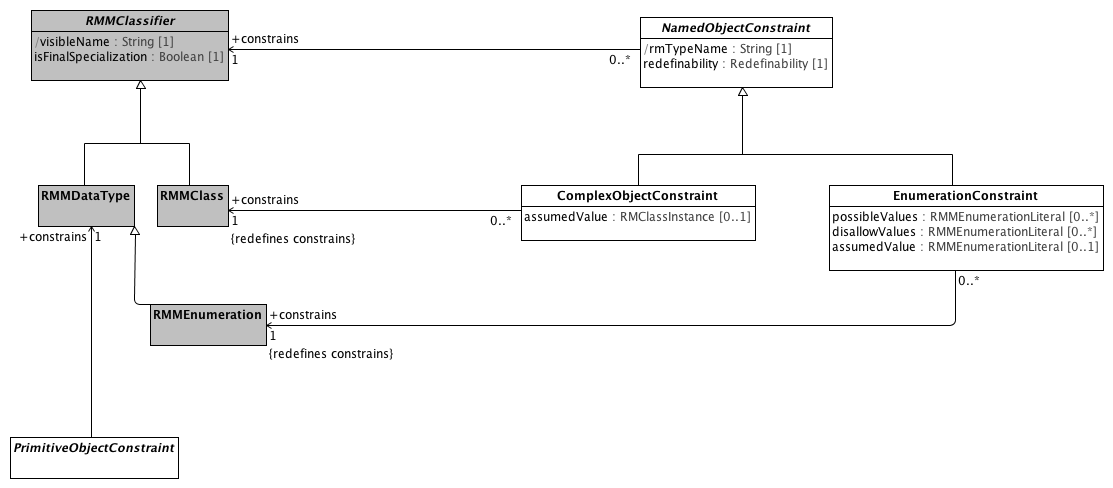
**Constraints**

• existence

[OCL]

parentProperty.lower = 1 implies existence = AttributeExistence::REQUIRED

### <Package> Object Constraint References



**Object Constraint References**

### <Class> ComplexObjectConstraint

**Description**

A constraint on a complex object, which will typically consist of other constraints

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Attributes**

• public assumedValue : [RMClassInstance](#_935cd7de4b22d47dd2c6aef93bed5c7a) [0..1]

Value to be assumed in instances in which no value is provided

**Associations**

• public attributeTuple : [AttributeTupleConstraint](#_f6da15c71717330ae1b56f8b41e3dd51) [0..\*]

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

• private targetObject : [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36) [0..\*]

A constraint defined by reference to a node defined elsewhere in the same archetype

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

**Constraints**

• instanceOfConstraint

[English]

If assumedValue exists, assumedValue.classifier must be equal to or a specialization of self.parent

### <Class> EnumerationConstraint

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[LocalEnumerationConstraint](#_fac1fd23e79b0fc3d709d92006b38e40)

**Attributes**

• public possibleValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of possible enumeration literals that are valid in the constrained instance. If *possibleValues* is empty, all literals not referenced in *disallowValues* are valid.

• public disallowValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of enumeration literals that can't appear in this constrained instance.

• public assumedValue : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..1]

**Associations**

• public constrains : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e) [1]

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Constraints**

• pvValues

The list of possible values must be a subset of the set of possible enumeration literals.

[OCL2.0]

self.possibleValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• dvValues

The list of disallowed values must be in the set of enumeration literals.

[OCL2.0]

self.disallowValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• pORd

An constraint may either specify possible values or disallow values but not both.

[OCL2.0]

possibleValues->size() = 0 or disallowValues->size() = 0

### <Class> NamedObjectConstraint

**Description**

Abstract model of constraint on any kind of object node

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootProxy](#_12e58855caae51d65fb43e2837534f63), [ArchetypeSlot](#_e518b2b75b6f66417345772b8440e6f2), [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134), [EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f)

**Attributes**

• public rmTypeName : String [1]

Reference model type that this node constrains

• public redefinability : [Redefinability](#_45bc3b03e253b26272fb450b2c34f5f2) [1]

Whether this node can be further constrained or elaborated in specializations

**Associations**

• public constrains : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

### <Class> PrimitiveObjectConstraint

**Description**

A constraint on an instance of a primitive data type (see: [Primitive Data Types package](platform:/resource/metamodel/am.emx#_-pgJIByrEeONZZvjZFK4_A)) a Terminology Code Reference (See: [Core package of the Terminology Services module](platform:/resource/metamodel/am.emx#_FkmfQJgMEeOEysZ5-LoitA)) or an RMMEnumeration as defined in the [Enumeration Metamodel](platform:/resource/metamodel/am.emx#_BgPdsJW2EeOEysZ5-LoitA).

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[BooleanConstraint](#_72963a51b64d10f1a0ab72b99a7d95f7), [DateConstraint](#_05406539ea2335c7e43c699988cdb385), [DateTimeConstraint](#_2e2f2a2affb8ef26918289abc068cad2), [DurationConstraint](#_ac453484c1a5116843147a38bd4020c0), [IntegerConstraint](#_05ad521706377b116680c20824646d90), [RealConstraint](#_c4e09895097a057d7e6ce4b0d10c8967), [StringConstraint](#_e78c0feb207cbea2ca9911ec94e2a83e), [TimeConstraint](#_20b17cf4d1a1f7228a809f6ed68b3a0c)

**Associations**

• public constrains : [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310) [1]

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMDataType

**Description**

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Direct Known Subclasses (Specialization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

### <Class> RMMEnumeration

**Description**

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Direct Known Superclasses (Generalization)**

[RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51)

**Associations**

• public ownedLiteral : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

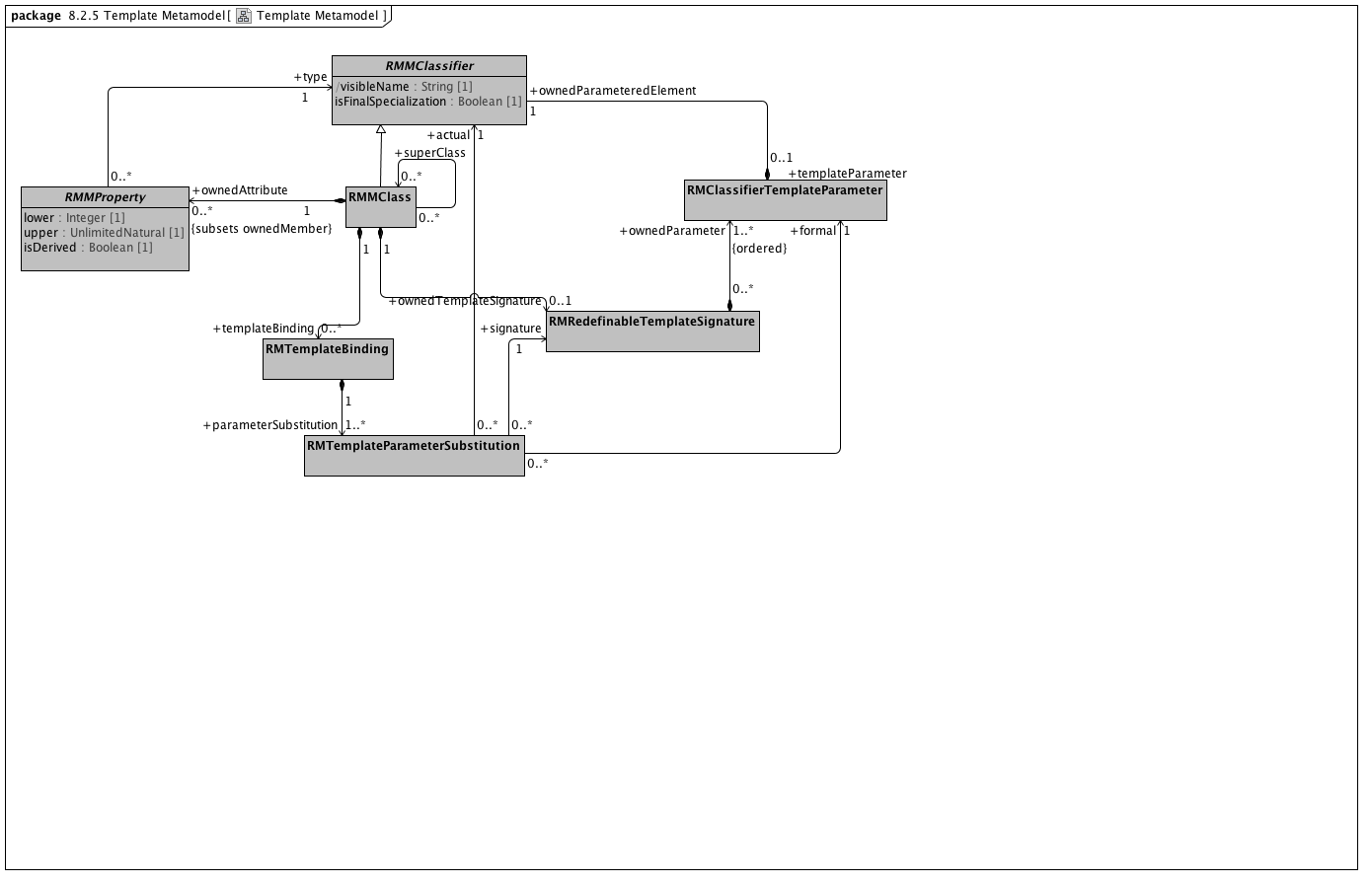
A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Constraints**

• enumerateValueDomain

[]

### <Package> Template Metamodel



**Template Metamodel**

### <Class> RMClassifierTemplateParameter

**Associations**

• public ownedParameteredElement : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMProperty

**Description**

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMCollectionProperty](#_702e30cd0381b6e9726bcc6fe779a70f), [RMMSingularProperty](#_5917d2795b1a9ae4f33929e6edb8af81)

**Attributes**

• private lower : Integer [1]

• private upper : UnlimitedNatural [1]

• public isDerived : Boolean [1]

**Associations**

• public type : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Constraints**

• nonDerived

[OCL]

isDerived=false

### <Class> RMRedefinableTemplateSignature

**Description**

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

**Associations**

• public ownedParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [1..\*]

### <Class> RMTemplateBinding

**Description**

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

**Associations**

• public parameterSubstitution : [RMTemplateParameterSubstitution](#_c2a122fef357367888fcb3768852586c) [1..\*]

A *RMTemplateParameterSubstitution* indicates that the *actual RMClass* or *RMPrimitiveType* is to be substituted as the *type* for the *ownedParameteredElement* owned by the *formal* *RMClassifierTemplateParameter.*

### <Class> RMTemplateParameterSubstitution

**Description**

A *RMTemplateParameterSubstitution* indicates that the *actual RMClass* or *RMPrimitiveType* is to be substituted as the *type* for the *ownedParameteredElement* owned by the *formal* *RMClassifierTemplateParameter.*

**Associations**

• public formal : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [1]

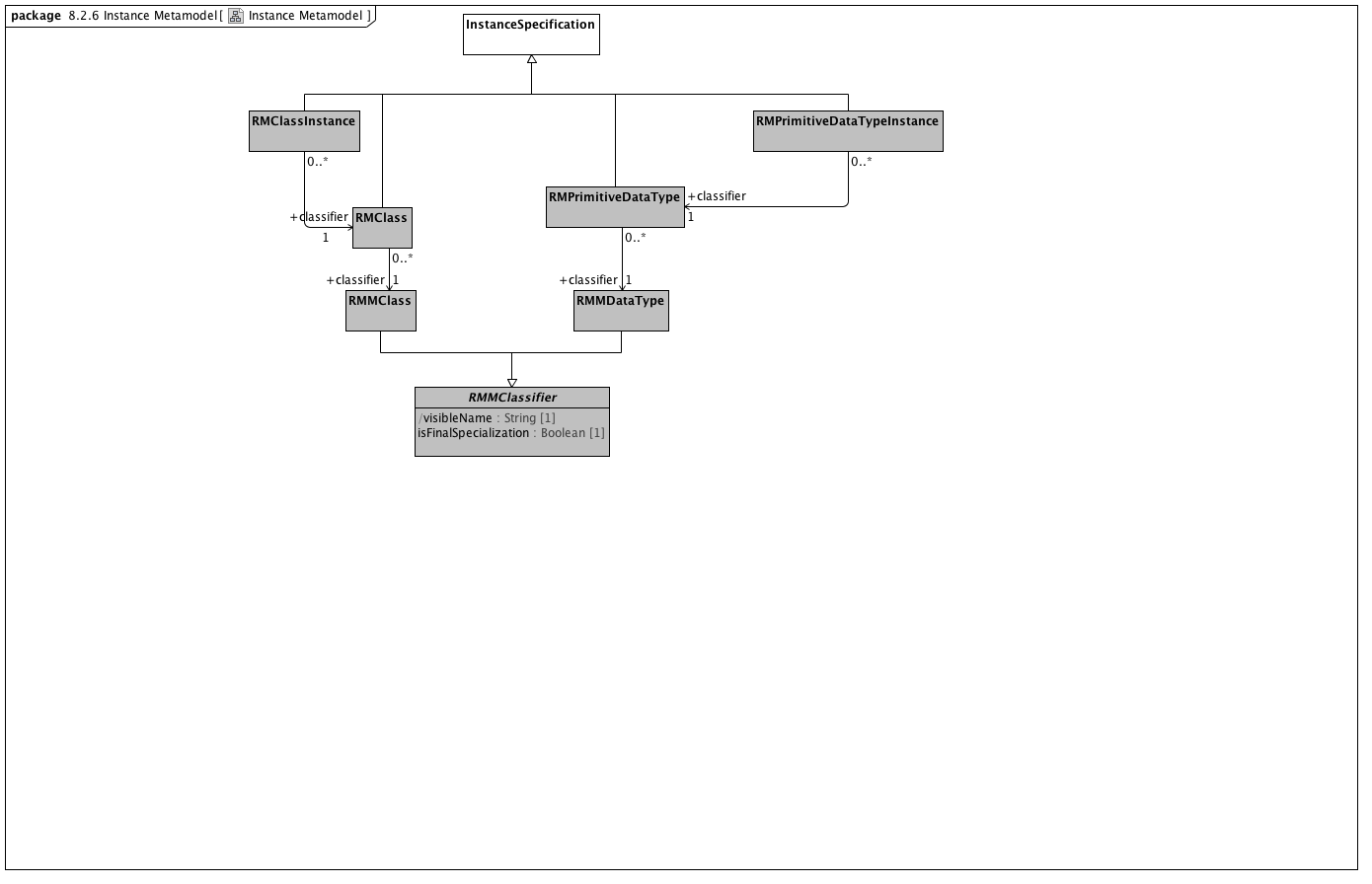
• public signature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public actual : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

### <Package> Instance Metamodel



**Instance Metamodel**

### <Class> InstanceSpecification

**Direct Known Subclasses (Specialization)**

[RMClass](#_977fccbc34231ad0bea8fc5ff3c8addc), [RMClassInstance](#_935cd7de4b22d47dd2c6aef93bed5c7a), [RMPrimitiveDataType](#_2fd719f1d4a4fafd390224b10cd96510), [RMPrimitiveDataTypeInstance](#_44fd267adf8202d91aafd96398da0a13)

### <Class> RMClass

**Description**

An instance of *an instance* of an *RMClass.* As an example, if the RMClass "Automobile" were an instance of an RMClass, with the ownedAttributes "model" and "year", an RMClassInstance might be named "Ford", with the model attribute set to "Fairlane" and the year to "1965".

**Direct Known Superclasses (Generalization)**

[InstanceSpecification](#_14aa28a97b7753c26b32de25fe333e41)

**Associations**

• public classifier : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

### <Class> RMClassInstance

**Direct Known Superclasses (Generalization)**

[InstanceSpecification](#_14aa28a97b7753c26b32de25fe333e41)

**Associations**

• public classifier : [RMClass](#_977fccbc34231ad0bea8fc5ff3c8addc) [1]

An instance of *an instance* of an *RMClass.* As an example, if the RMClass "Automobile" were an instance of an RMClass, with the ownedAttributes "model" and "year", an RMClassInstance might be named "Ford", with the model attribute set to "Fairlane" and the year to "1965".

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMDataType

**Description**

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Direct Known Subclasses (Specialization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

### <Class> RMPrimitiveDataType

**Direct Known Superclasses (Generalization)**

[InstanceSpecification](#_14aa28a97b7753c26b32de25fe333e41)

**Associations**

• public classifier : [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310) [1]

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

### <Class> RMPrimitiveDataTypeInstance

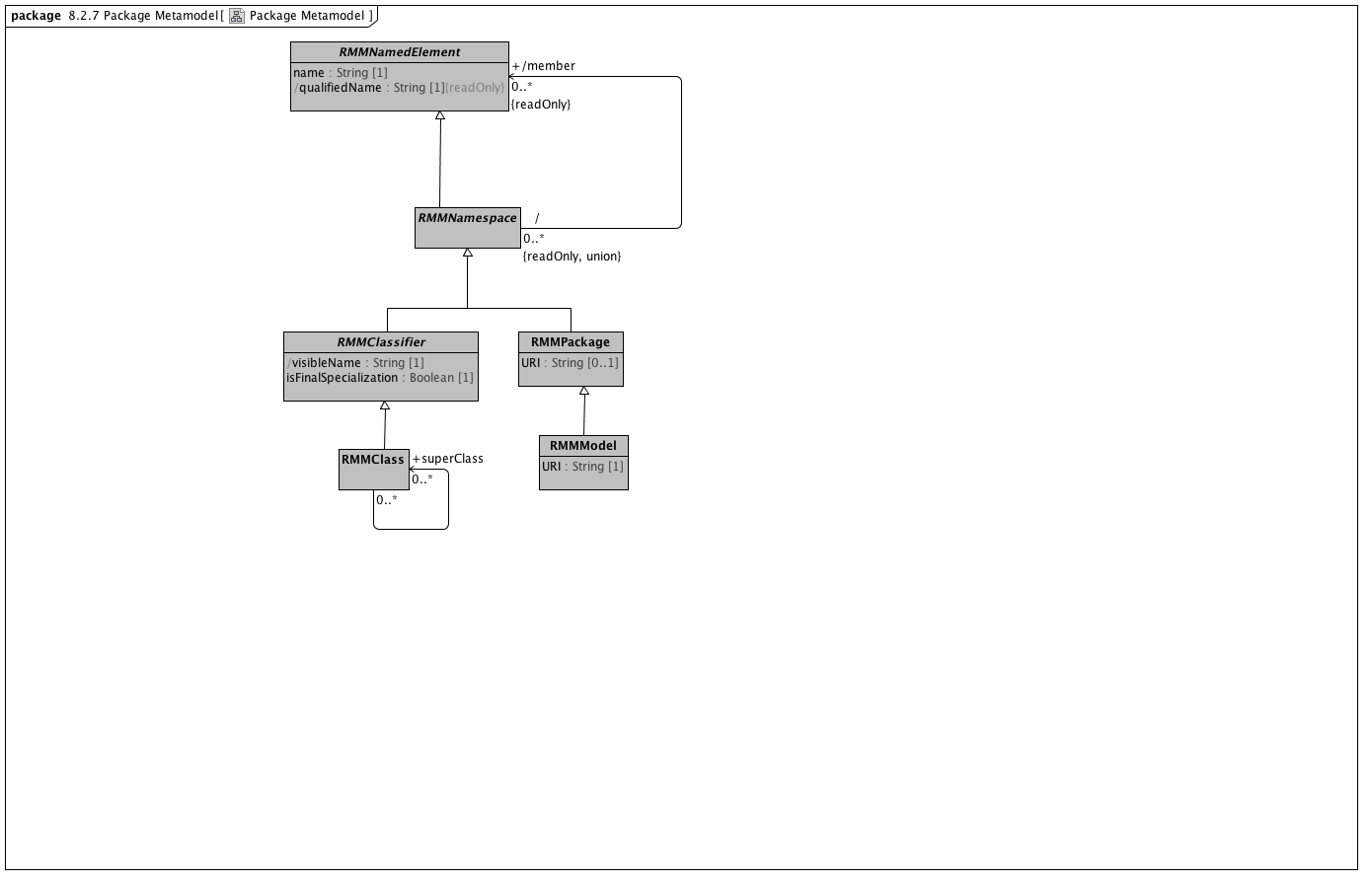
**Direct Known Superclasses (Generalization)**

[InstanceSpecification](#_14aa28a97b7753c26b32de25fe333e41)

**Associations**

• public classifier : [RMPrimitiveDataType](#_2fd719f1d4a4fafd390224b10cd96510) [1]

### <Package> Package Metamodel



**Package Metamodel**

### <Class> RMMClass

**Description**

A class, in the object-oriented sense

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Associations**

• public ownedAttribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public ownedTemplateSignature : [RMRedefinableTemplateSignature](#_6afdd25f5589999ef5ae78a4eab8563d) [0..1]

A *RMRedefinableTemplateSignature* associates an ordered list of *RMClassifierTemplateParameters* with an owning *RMClass.* The owning *RMClass* typically, but not always has one or more ownedAttributes that reference one of the ownedParameters of the *RMRedefinableTemplateSignature.*

• public templateBinding : [RMTemplateBinding](#_039ec0a61521832e985575d3d9688234) [0..\*]

*RMTemplateBinding* is a subtype of the UML::TemplateBinding class. It represents a set of parameter substitutions that are to be applied to a *RMRedefinableTemplateSignature* defined by a parent or ancestor *superClass.* A *RMTemplateBinding* contains one or more parameter substitutions to be applied to one or more types referenced by an *ownedAttribute* of the parent or ancestor class.

• public attribute : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public superClass : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [0..\*]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

### <Class> RMMClassifier

**Description**

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

**Direct Known Subclasses (Specialization)**

[RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3), [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Attributes**

• public visibleName : String [1]

The minimal qualification necessary to render the RMClass or RMDataType name distinguishable from all other names within the containing namespace

• public isFinalSpecialization : Boolean [1]

If true, the *RMMClassifier* instance cannot be constrained within an archetype.

**Associations**

• public templateParameter : [RMClassifierTemplateParameter](#_3d9b09fe9052c8305d90ab92bc37d26b) [0..1]

### <Class> RMMModel

**Description**

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

**Direct Known Superclasses (Generalization)**

[RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Attributes**

• public URI : String [1]

### <Class> RMMNamedElement

**Description**

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

**Direct Known Subclasses (Specialization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22), [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b), [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9)

**Attributes**

• public name : String [1]

The name of the Reference Model element. Name must be unique within the context of the owning *namespace*

• public qualifiedName : String [1]

**Associations**

• public namespace : [RMMNamespace](#_f762e4ef59f1948849a49d421126c16b) [0..1]

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

### <Class> RMMNamespace

**Description**

An element in a model that owns and/or imports a set of NamedElements that can be identified by name

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873), [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4)

**Associations**

• public member : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

• public ownedMember : [RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816) [0..\*]

*RMNamedElement* is the superclass of all named elements in the Reference Model, and represents the subset of UML::NamedElements that are referenced by the AML profile. While a Reference Model may contain UML::NamedElements without names, Archetypes can only constrain those that have *names* and are of type *RMPrimitiveDataType*, *RMClass* or *RMProperty*.

### <Class> RMMPackage

**Direct Known Superclasses (Generalization)**

[RMMNamespace](#_f762e4ef59f1948849a49d421126c16b)

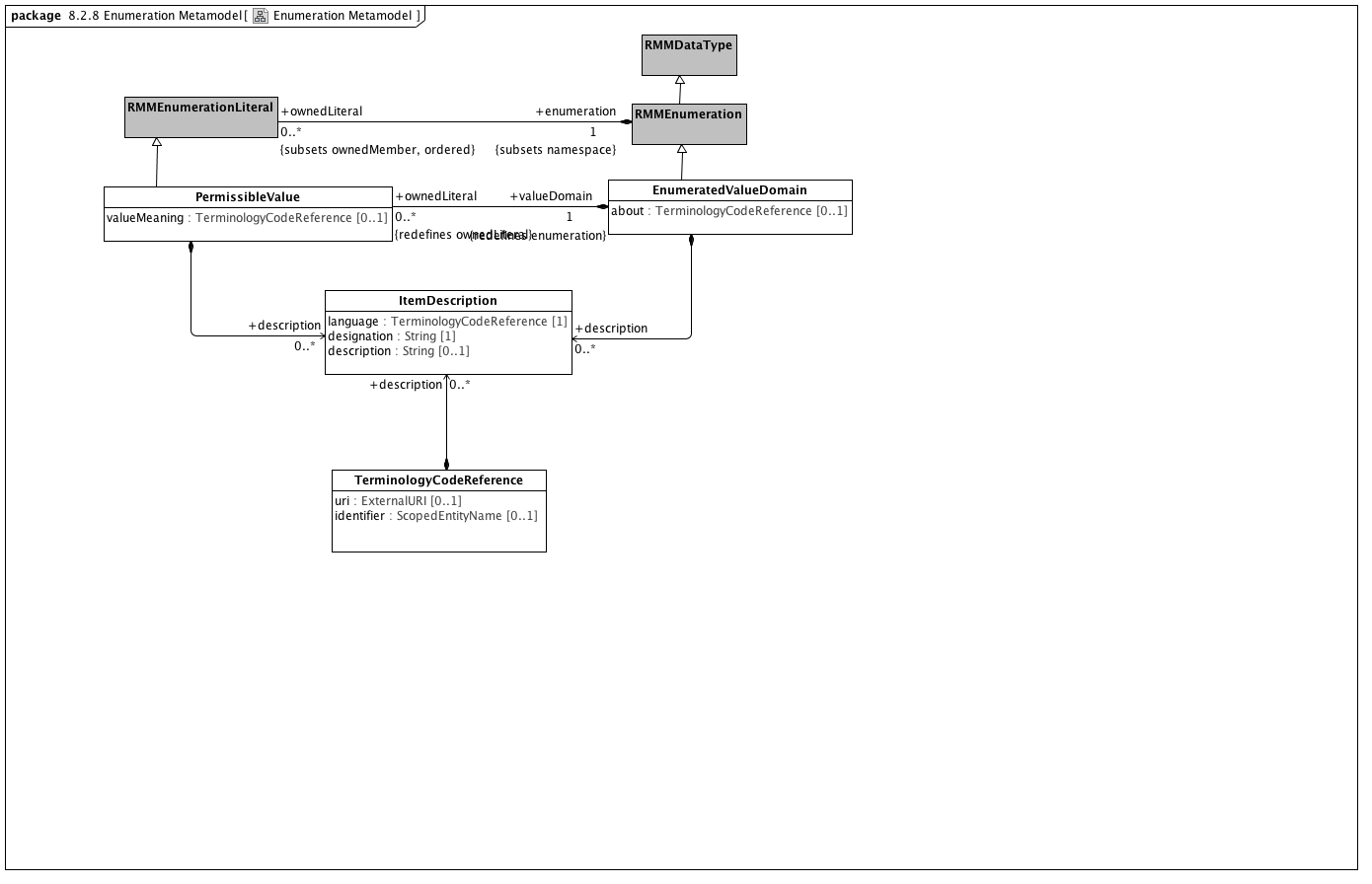
**Direct Known Subclasses (Specialization)**

[RMMModel](#_fc116c5fcb379006ed51eb855a1dae57)

**Attributes**

• public URI : String [0..1]

### <Package> Enumeration Metamodel



**Enumeration Metamodel**

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> ItemDescription

**Description**

A human readable designation and optional description of an object model entity in a specified language.

**Attributes**

• public language : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]

The language in which the term is defined

• public designation : String [1]

description of the meaning of the term

• public description : String [0..1]

description of the meaning of the term

**Associations**

• public : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> RMMDataType

**Description**

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

**Direct Known Superclasses (Generalization)**

[RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873)

**Direct Known Subclasses (Specialization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

### <Class> RMMEnumeration

**Description**

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Direct Known Superclasses (Generalization)**

[RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51)

**Associations**

• public ownedLiteral : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Constraints**

• enumerateValueDomain

[]

### <Class> RMMEnumerationLiteral

**Description**

A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12)

**Associations**

• public enumeration : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e) [1]

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

• public classifier : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

### <Class> TerminologyCodeReference

**Description**

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

**Direct Known Subclasses (Specialization)**

[AttributeName](#_e2995e77bf0a43587adc571de3d5131b)

**Attributes**

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

A URI that resolves to the full EntityDescription represented by this resource.

• public identifier : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [0..1]

A namespace/name combination that uniquely represents the entity. This can be the primary entityID, as determined by the service or any valid alternateId. Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering URIAndEntityName instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the “fully specified name” or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

**Constraints**

• identifierOrURI

A TerminologyCodeReference must either have a *uri*, an *identifier*, or both.

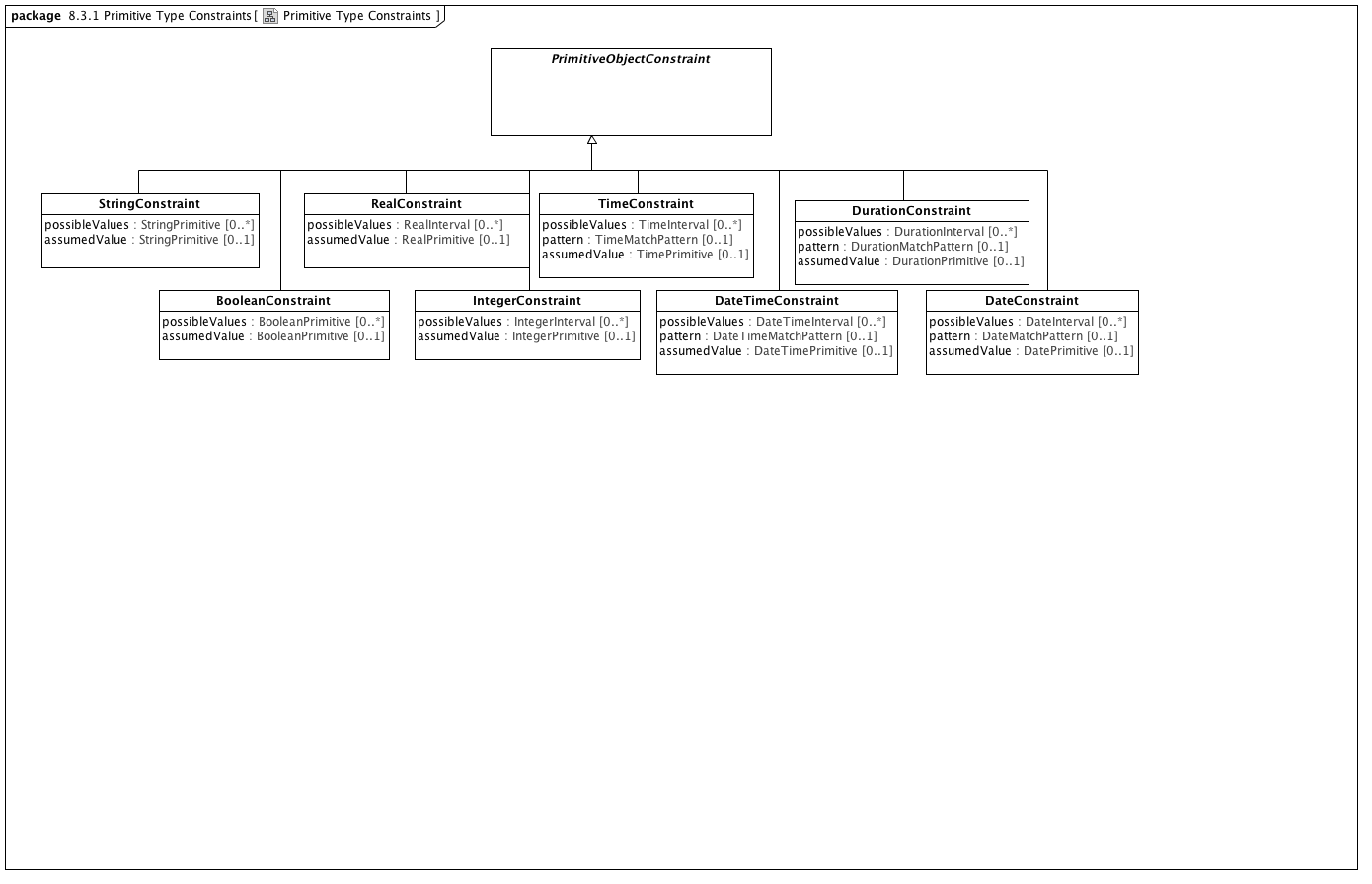
[OCL]

uri->notEmpty() or identifier->notEmpty()

## <Package> Constraint Object Model

Constraint Model Package Overview The constraint model is the core of the archetype design. It illustrates how constraints are defined, showing the object-attribute-object pattern characteristic of object constraints. ComplexObjectConstraint. Because objects are composed of properties (attributes and relationships), and properties consist of objects, the archetype definition consists of alternate layers of ArchetypeRootConstraint, but rather than a single archetype, it defines a set of archetypes. It can be thought of like a keyhole, into which few or many keys might fit, depending on how specific its shape is. Logically it has the same semantics as a ComplexObjectConstraint, except that the constraints are expressed in another archetype, not the current one. ComplexObjectConstraints PrimitiveObjectConstraints AttributeConstraints SingularAttributeConstraint class. Where a SingularAttributeConstraint is associated with more than one ObjectConstraint, the ObjectConstraints are alternatives. AttributeCollectionConstraint, which differentiates between unique and repeatable and between ordered and unordered collections. In addition, while the AttributeConstraint determines whether a property may exist, the quantity of a repeating element is defined in the AttributeCollectionConstraint’s cardinality property. AttributeCollectionConstraint, with its defined cardinality, there may be different sets of sibling members with different constraints, and the number of each of these subsets is specified as the AttributeCollectionMember’s occurrences property. In an organization, for instance, the cardinality for the member property may be “two or more,” but within that set of members, we may have two constraints. One type of member, the leader (indicated by an ObjectConstraint on the person type or role), may be required to occur exactly once, whereas other types may have multiple occurrences. AttributeTupleConstraints AttributeTuple would be defined for each pair of the two values (unit code and numeric ceiling), and these tuples would be grouped into an AttributeTupleConstraint, defining an array of acceptable sets of values.

### <Package> Primitive Type Constraints



**Primitive Type Constraints**

### <Class> BooleanConstraint

**Description**

A *BooleanConstraint* restricts the possible values of a target *BooleanPrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [BooleanPrimitive](#_19737cdaaaee2179c50553b52361808b) [0..\*]

List of valid Boolean values for this constraint

• public assumedValue : [BooleanPrimitive](#_19737cdaaaee2179c50553b52361808b) [0..1]

### <Class> DateConstraint

**Description**

A *DateConstraint* restricts the possible values of a target *DatePrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [DateInterval](#_69ee53d558bd7c80a52df2f5ae7c8ab0) [0..\*]

• public pattern : [DateMatchPattern](#_3cae9b111208dfafe95b548c45720f8e) [0..1]

• public assumedValue : [DatePrimitive](#_db5d020506d0af7a330f0b4fe1cb870a) [0..1]

### <Class> DateTimeConstraint

**Description**

A *DateTimeConstraint* restricts the possible values of a target *DateTimePrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [DateTimeInterval](#_3145967a9ac97f161c2d4ff9e296537e) [0..\*]

Range of valid DateTime values

• public pattern : [DateTimeMatchPattern](#_b501b0aa7bbcd9ac89d6ddbf9f0bf3d9) [0..1]

A DateTimeMatchPattern to constrain valid instances of DateTime

• assumedValue : [DateTimePrimitive](#_8347879bb381db17040637cc3ba0a25c) [0..1]

### <Class> DurationConstraint

**Description**

A *DurationConstraint* restricts the possible values of a target *DurationPrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [DurationInterval](#_8370a7823f617661d9ddb083556dfa3f) [0..\*]

Range of valid Duration values

• private pattern : [DurationMatchPattern](#_baaa358403d3312c0779bc256e1050bd) [0..1]

A DurationMatchPattern to constrain valid instances of Duration

• public assumedValue : [DurationPrimitive](#_64b0026498682fb721ccdb38c186bba2) [0..1]

### <Class> IntegerConstraint

**Description**

An *IntegerConstraint* restricts the possible values of a target *IntegerPrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• possibleValues : [IntegerInterval](#_5218d9b9782bc6f925ed66b3e2bc529b) [0..\*]

• public assumedValue : [IntegerPrimitive](#_89e2b2b9de405e6d05c4c5259fc8ffd6) [0..1]

### <Class> PrimitiveObjectConstraint

**Description**

A constraint on an instance of a primitive data type (see: [Primitive Data Types package](platform:/resource/metamodel/am.emx#_-pgJIByrEeONZZvjZFK4_A)) a Terminology Code Reference (See: [Core package of the Terminology Services module](platform:/resource/metamodel/am.emx#_FkmfQJgMEeOEysZ5-LoitA)) or an RMMEnumeration as defined in the [Enumeration Metamodel](platform:/resource/metamodel/am.emx#_BgPdsJW2EeOEysZ5-LoitA).

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[BooleanConstraint](#_72963a51b64d10f1a0ab72b99a7d95f7), [DateConstraint](#_05406539ea2335c7e43c699988cdb385), [DateTimeConstraint](#_2e2f2a2affb8ef26918289abc068cad2), [DurationConstraint](#_ac453484c1a5116843147a38bd4020c0), [IntegerConstraint](#_05ad521706377b116680c20824646d90), [RealConstraint](#_c4e09895097a057d7e6ce4b0d10c8967), [StringConstraint](#_e78c0feb207cbea2ca9911ec94e2a83e), [TimeConstraint](#_20b17cf4d1a1f7228a809f6ed68b3a0c)

**Associations**

• public constrains : [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310) [1]

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

### <Class> RealConstraint

**Description**

A *RealConstraint* restricts the possible values of a target *RealPrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [RealInterval](#_27fcdc4a73f595dca8b4e4fa28fd59a6) [0..\*]

• public assumedValue : [RealPrimitive](#_c596f10fb93cb4f697f2f1b0b64b43ce) [0..1]

### <Class> StringConstraint

**Description**

A *StringConstraint* restricts the possible values of a target *StringPrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [StringPrimitive](#_6a90be7cfa784ea4b4e8cad8f4a47e82) [0..\*]

A list of valid String instances

• public assumedValue : [StringPrimitive](#_6a90be7cfa784ea4b4e8cad8f4a47e82) [0..1]

### <Class> TimeConstraint

**Description**

A *TimeConstraint* restricts the possible values of a target *TimePrimitive* data type.

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da)

**Attributes**

• public possibleValues : [TimeInterval](#_31d7d28b74d99d489c8e65e4a57e9ca9) [0..\*]

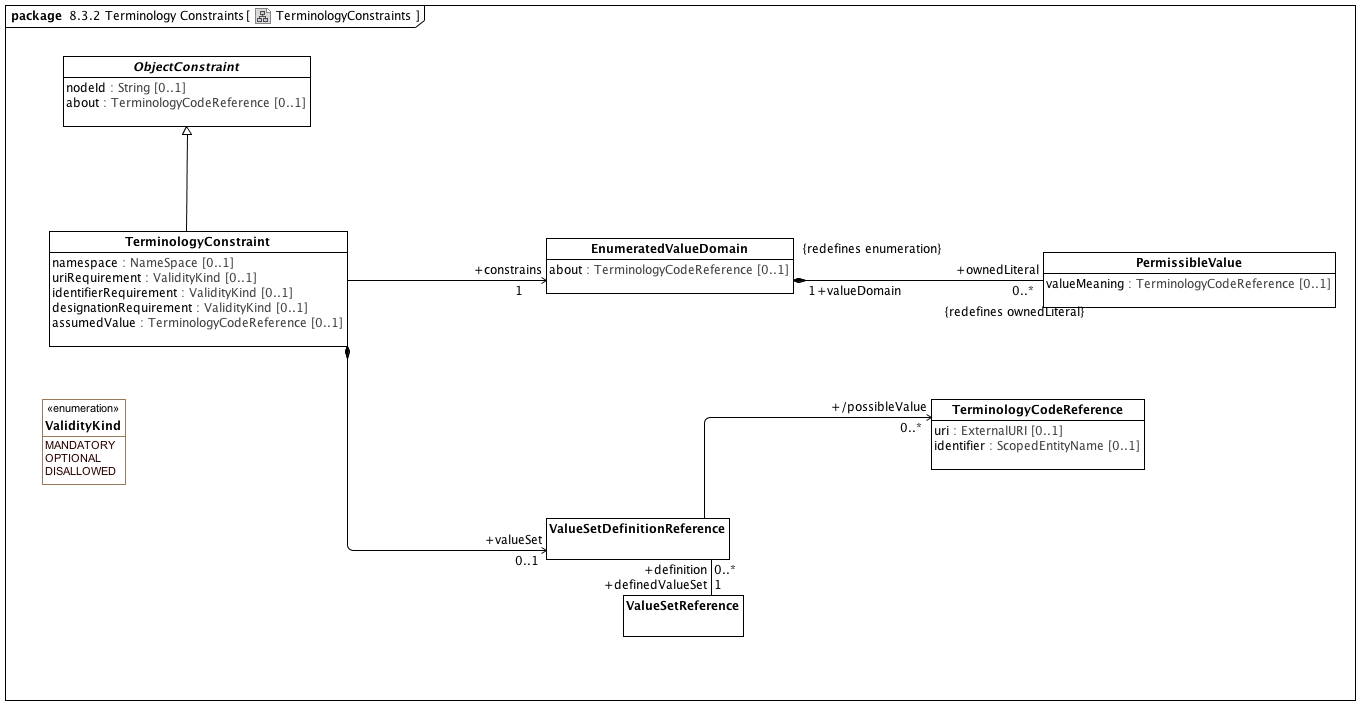
Range of valid Time values

• public pattern : [TimeMatchPattern](#_a28995c5bf2253d86854a7a97e55ba1d) [0..1]

A TimeMatchPattern to constrain valid instances of Time

• assumedValue : [TimePrimitive](#_05e0f2a221f0733d693058e9253b0017) [0..1]

### <Package> Terminology Constraints



**TerminologyConstraints**

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> TerminologyCodeReference

**Description**

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

**Direct Known Subclasses (Specialization)**

[AttributeName](#_e2995e77bf0a43587adc571de3d5131b)

**Attributes**

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

A URI that resolves to the full EntityDescription represented by this resource.

• public identifier : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [0..1]

A namespace/name combination that uniquely represents the entity. This can be the primary entityID, as determined by the service or any valid alternateId. Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering URIAndEntityName instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the “fully specified name” or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

**Constraints**

• identifierOrURI

A TerminologyCodeReference must either have a *uri*, an *identifier*, or both.

[OCL]

uri->notEmpty() or identifier->notEmpty()

### <Class> TerminologyConstraint

**Description**

A constraint on instances of the reference model Terminology type

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Attributes**

• public namespace : [NameSpace](#_b448a755fcce2af9da8221a55b6b53ff) [0..1]

• public uriRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a URI is required

• public identifierRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a concept identifier is required

• public designationRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a human-readable designation is required

• public assumedValue : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public valueSet : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..1]

• public constrains : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> ValueSetDefinitionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Associations**

• public possibleValue : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..\*]

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

• public definedValueSet : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7) [1]

The URI, identifier and name of a collection of TerminologyCodeReferences

### <Class> ValueSetReference

**Description**

The URI, identifier and name of a collection of TerminologyCodeReferences

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Associations**

• public binding : [ValueSetBinding](#_fdb7a97d4c43d7f09387cdc69dd2f65a) [0..1]

An externally specified set of coded values

• public definition : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..\*]

### <Enumeration> ValidityKind

**Description**

Enumeration of classes of permissibility

**Enumeration Literals**

* **DISALLOWED**

The property must not be present in a conformant instance

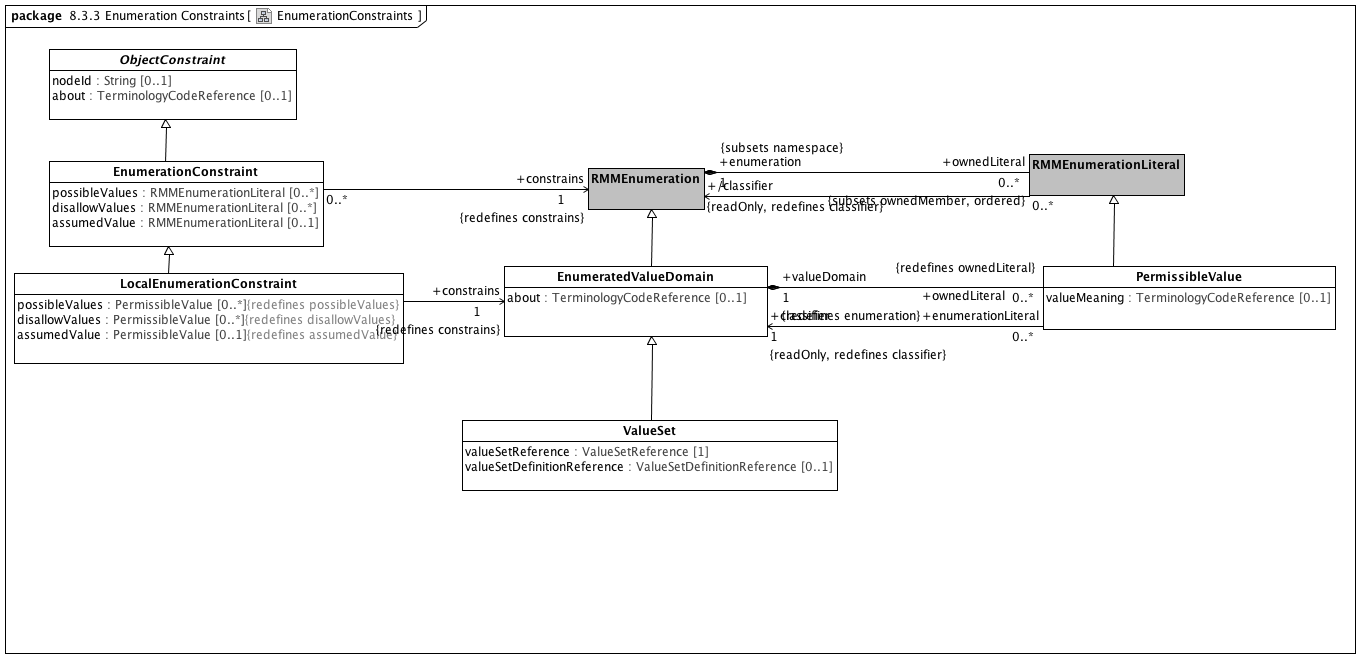
* **MANDATORY**

The property must be present in a conformant instance

* **OPTIONAL**

The property may be present in a conformant instance

### <Package> Enumeration Constraints



**EnumerationConstraints**

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> EnumerationConstraint

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[LocalEnumerationConstraint](#_fac1fd23e79b0fc3d709d92006b38e40)

**Attributes**

• public possibleValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of possible enumeration literals that are valid in the constrained instance. If *possibleValues* is empty, all literals not referenced in *disallowValues* are valid.

• public disallowValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of enumeration literals that can't appear in this constrained instance.

• public assumedValue : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..1]

**Associations**

• public constrains : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e) [1]

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Constraints**

• pvValues

The list of possible values must be a subset of the set of possible enumeration literals.

[OCL2.0]

self.possibleValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• dvValues

The list of disallowed values must be in the set of enumeration literals.

[OCL2.0]

self.disallowValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• pORd

An constraint may either specify possible values or disallow values but not both.

[OCL2.0]

possibleValues->size() = 0 or disallowValues->size() = 0

### <Class> LocalEnumerationConstraint

**Direct Known Superclasses (Generalization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f)

**Attributes**

• public possibleValues : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

• public disallowValues : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

• public assumedValue : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..1]

**Associations**

• public constrains : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> RMMEnumeration

**Description**

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Direct Known Superclasses (Generalization)**

[RMMDataType](#_d5914eb0da42172989bbe57f23fc4310)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51)

**Associations**

• public ownedLiteral : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Constraints**

• enumerateValueDomain

[]

### <Class> RMMEnumerationLiteral

**Description**

A RMMEnumerationLiteral is a specialization UML::EnumerationLiteral. The only characteristic that is significant from the AML model perspective is the *RMMEnumerationLiteral* name, which is unique within the context of the RMMEnumeration namespace. An *RMMEnumerationLiteral* returns its *name* as the *PermissibleValue* value().

**Direct Known Superclasses (Generalization)**

[RMMNamedElement](#_527fd9eb1e787c36a3748854a9431816)

**Direct Known Subclasses (Specialization)**

[PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12)

**Associations**

• public enumeration : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e) [1]

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

• public classifier : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

### <Class> ValueSet

**Direct Known Superclasses (Generalization)**

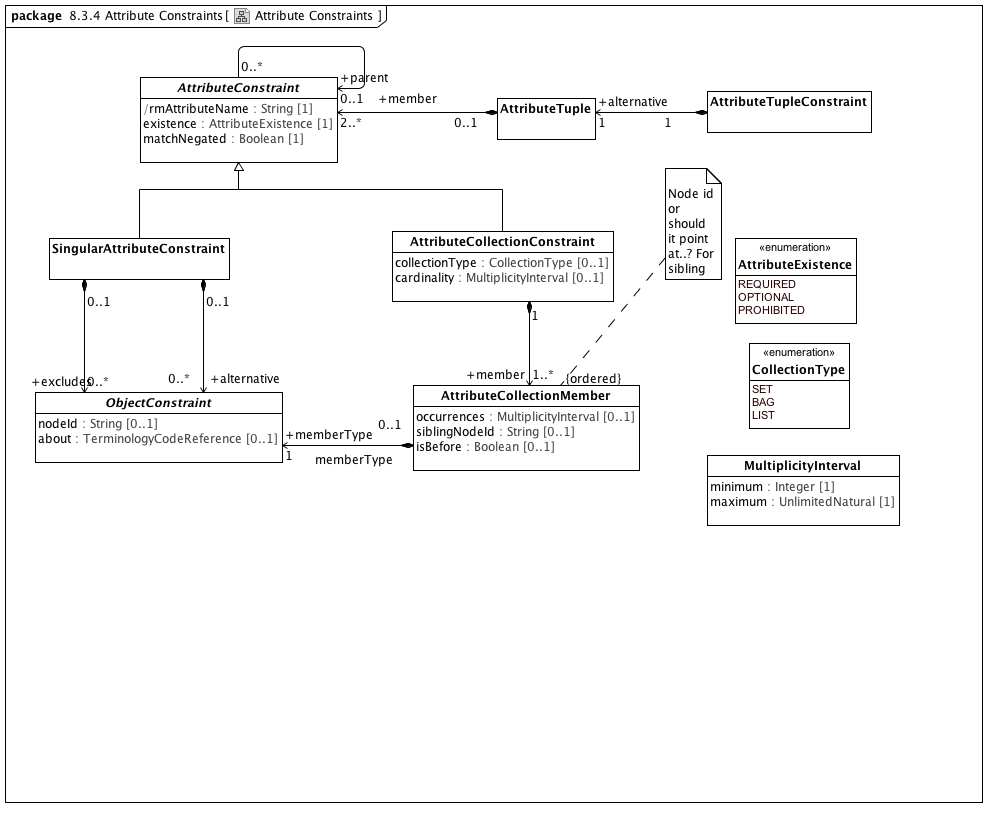
[EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51)

**Attributes**

• public valueSetReference : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7) [1]

• public valueSetDefinitionReference : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..1]

### <Package> Attribute Constraints



**Attribute Constraints**

### <Class> AttributeCollectionConstraint

**Description**

A constraint on a set of objects contained in an attribute

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05)

**Attributes**

• public collectionType : [CollectionType](#_ac8c7771bf7f68d6747022e6924749ca) [0..1]

A classification of collection indicating whether its members must be unique or ordered

• public cardinality : [MultiplicityInterval](#_c810ec7fa381fa249b7a7d9fecae85b6) [0..1]

The range of quantities of members that can be included in an attribute

**Associations**

• public member : [AttributeCollectionMember](#_177e37623ae3f5642980fd445bf78af1) [1..\*]

An association that matches members of a collection of attributes with specific ObjectConstraints

• public constrains : [RMMCollectionProperty](#_702e30cd0381b6e9726bcc6fe779a70f) [1]

RMMCollectionProperty represents the subset of RMMProperty instances that can occur more than one time. An RMMCollectionProperty instance is viewed by AML as a collection of objects of a given type that possess two separate characteristics:

The collection as a whole may be required, optional or prohibited.

The cardinality of the collection may be constrained.

This combination allows a number of useful constructs, including:

requiring that a list be present but that it have no members, which can be used to assert a relationship between an object and an empty set of objects

making an attribute optional, but, if present, requiring that it have a minimum number of members

**Constraints**

• collectionType

[OCL]

parentProperty.isOrdered implies collectionType = CollectionType::LIST and parentProperty.isUnique implies collectionType = CollectionType.SET

• cardinality

[OCL]

cardinality.minimum >= parent.lower and (cardinality.maximum = unlimitedValue() or parent.upper = unlimitedValue() or cardinality.maximum <= parent.upper)

### <Class> AttributeCollectionMember

**Description**

An association that matches members of a collection of attributes with specific ObjectConstraints

**Attributes**

• public occurrences : [MultiplicityInterval](#_c810ec7fa381fa249b7a7d9fecae85b6) [0..1]

Number of times the element described by this constraint can repeat

• public siblingNodeId : String [0..1]

Identification of another node under this AttributeConstraint from which *isBefore* is evaluted

• public isBefore : Boolean [0..1]

Whether an object [immediately] precedes the object named in siblingNodeId

**Associations**

• public memberType : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [1]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

### <Class> AttributeConstraint

**Description**

A constraint on a reference model attribute

**Direct Known Subclasses (Specialization)**

[AttributeCollectionConstraint](#_5eefba8eca7402f09bd5619804038771), [SingularAttributeConstraint](#_48ee2586ffa14e5bb1cf8ad893969da7)

**Attributes**

• public rmAttributeName : String [1]

Name of attribute within the reference model that is constrained by this node

• public existence : [AttributeExistence](#_4f99fbfcf9617d7ad55eca111d84fb67) [1]

Strength of requirement that the attribute instance be present

• public matchNegated : Boolean [1]

Whether the match operator is to be inverted so that the constraint specifies anything except what is represented

**Associations**

• public attribute : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134)

A constraint on a complex object, which will typically consist of other constraints

• public constrains : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public parent : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public object : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Constraints**

• name

[OCL]

rmAttributeName = parentProperty.name

### <Class> AttributeTuple

**Description**

A set of constraints on related attributes to be used to differentiate scenarios where the value of one attribute affects the valid values of another

**Associations**

• public member : [AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05) [2..\*]

A constraint on a reference model attribute

### <Class> AttributeTupleConstraint

**Description**

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

**Associations**

• public alternative : [AttributeTuple](#_6d5bfb351e19f61e0327587b0ff5fd4f) [1]

A set of constraints on related attributes to be used to differentiate scenarios where the value of one attribute affects the valid values of another

### <Class> MultiplicityInterval

**Description**

Range of quantities

**Attributes**

• public minimum : Integer [1]

The smallest value allowed

• public maximum : UnlimitedNatural [1]

The largest value allowed

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [AttributeTuple](#_6d5bfb351e19f61e0327587b0ff5fd4f) [1]

A set of constraints on related attributes to be used to differentiate scenarios where the value of one attribute affects the valid values of another

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> SingularAttributeConstraint

**Description**

An AttributeConstraint that identifies valid values for a single value instance

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05)

**Associations**

• public alternative : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

• public excludes : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

• public constrains : [RMMSingularProperty](#_5917d2795b1a9ae4f33929e6edb8af81) [1]

*RMMSingularProperty* represents the subset of *RMMProperty* instances having an upper bound of 1. The AML treats RMMSingular properties as single values (vs. collections) of attributes that can be required, optional or prohibited.

**Constraints**

• existence

[OCL]

parentProperty.lower = 1 implies existence = AttributeExistence::REQUIRED

### <Enumeration> AttributeExistence

**Description**

Strengths of requirement for the existence of an attribute

**Enumeration Literals**

* **OPTIONAL**

The attribute may be present in a conforming instance

* **PROHIBITED**

The attribute must not be present in a conforming instance

* **REQUIRED**

The attribute must be present in a conforming instance

### <Enumeration> CollectionType

**Description**

Classification of collections

**Enumeration Literals**

* **BAG**

An collection of elements, neither ordered nor necessarily unique

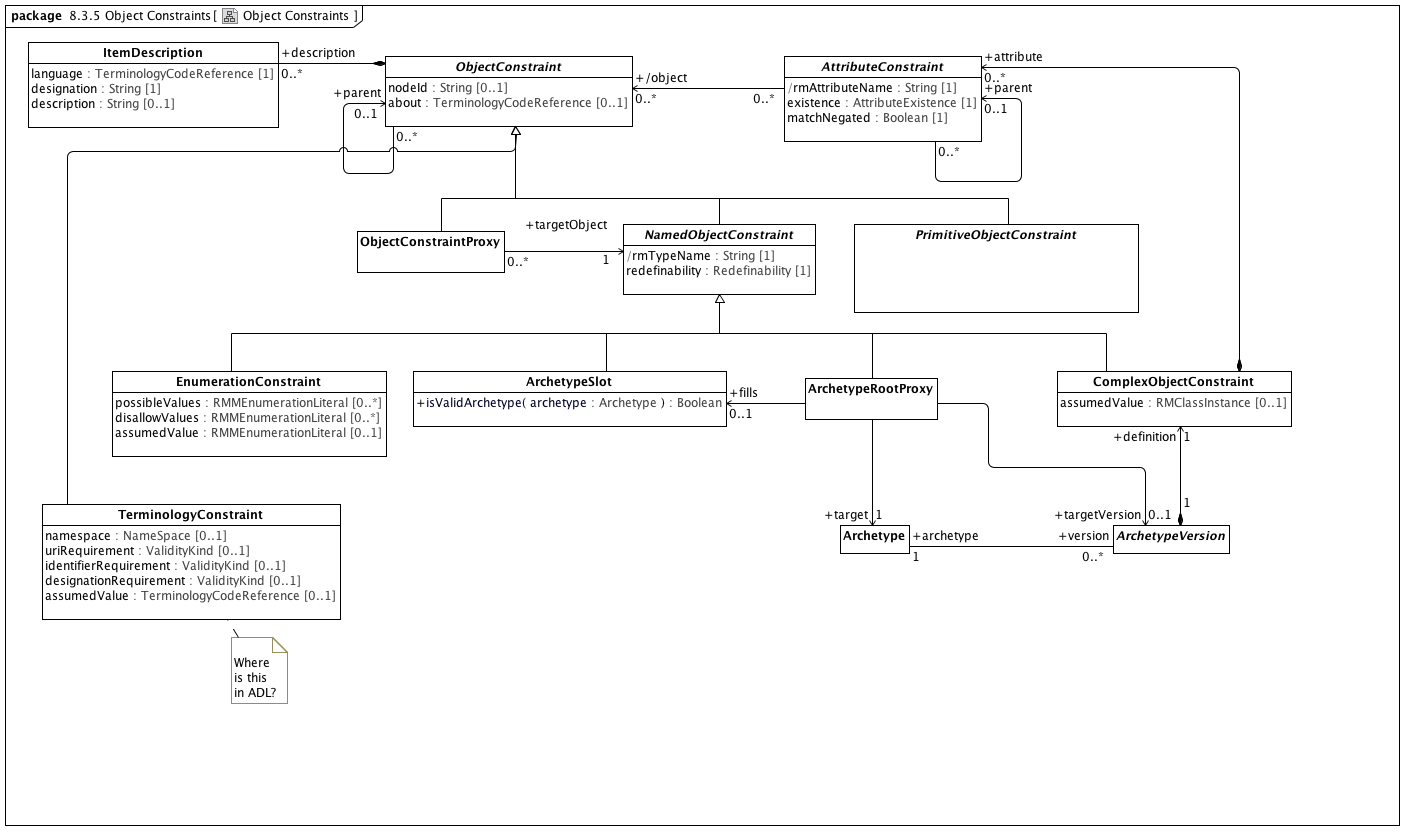
* **LIST**

An ordered collection of elements, not necessarily unique

* **SET**

An collection of unique elements, not necessarily ordered

### <Package> Object Constraints



**Object Constraints**

### <Class> Archetype

**Attributes**

• public archetypeId : [ArchetypeId](#_abe68de6d7b599f5e4ea361caee12c81) [1]

The unique archetype identifier. However constructed, this uniquely identifies the archetype across its entire life cycle. No other archetype can have this identifier and a different identifier designates a different Archetype.

• public archetypeName : String [1]

The human readable name of the Archetype. This is typically derived from the other archetype details (See: openEHR Knowledge Artefact Identification - Revision 0.7.0 for an example). It is possible for this identifier to change over the life of an Archetype.

• public rmPackagePath : String [1]

The qualifiedName of a package in the target reference model that has the root *rmClass* as a visible member (there can be more than one possibility in a reference model).

• public rmClassName : String [1]

Name of the root class of this archetype. *rmClass* must match the *visibleName* of the class referenced by the *ComplexObjectConstraint* target of the *Archetype definition* as well as the visibleName of the RMMClass instance that it constrains.

• public originalLanguage : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

The original language in which the resource was authored (essential for evaluating natural language quality)

• public urn : String [0..1]

**Associations**

• public rmUMLModel : [RMMModel](#_fc116c5fcb379006ed51eb855a1dae57) [1]

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

• public rmPackage : [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4) [1]

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

• public version : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

• public specializes : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• uri

[OCL]

rmURI = rmUMLModel.URI

• package

[OCL]

rmPackagePath = rmPackage.qualifiedName

• class

[OCL]

rmClassName = constrains.name

• classpackage

[OCL]

rmPackage.member->exists(c|c=constrains)

• differentArchetype

An archetype cannot specialize itself or any of its descendants (note - only self is included in formal OCL)

[OCL2.0]

not (self = self.specializes)

### <Class> ArchetypeRootProxy

**Description**

A specialization of ComplexObjectConstraint whose node\_id attribute is an archetype identifier rather than an internal node code. Used to reference external archetypes to be included in a composite archetype.

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Associations**

• public target : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

• public fills : [ArchetypeSlot](#_e518b2b75b6f66417345772b8440e6f2) [0..1]

A classifier that describes the set of archetypes that may be used to validate instances

• public targetVersion : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..1]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• validSlot

[Analysis]

If there is a target fills ArchetypeSlot, the ArchetypeRootProxy archetypeId must be a member of one or more includes value sets and not be a member of one or more excludes value sets as identified in the ArchetypeSlot or any of its parents or ancestors

• validTarget

[OCL]

target.archetypeId = archetypeId and target.archetypeName = archetypeName -- Also

### <Class> ArchetypeSlot

**Description**

A classifier that describes the set of archetypes that may be used to validate instances

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Operations**

• public isValidArchetype (archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75)) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

### <Class> ArchetypeVersion

**Description**

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Direct Known Subclasses (Specialization)**

[FlatArchetype](#_166addccec3fe63279b3f9eeb9930ab8), [SourceArchetype](#_dcaf9716e9bc2255b93c20393e8712f2)

**Attributes**

• public archetypeVersionId [1]

The specific version of this *Archetype*. *archetypeVersion* does not impact archetype identity. If an archetype undergoes non-backwards compatible changes, it becomes a new archetype with a new identifier.

• public amlVersion : String [1]

The URI of the modeling language and version used to construct this Archetype, if derived from a serialized representation

• public rmRelease : String [0..1]

The specific version of the reference model that was constrained. Depending on the context and workflow model, it may be possible to update a reference model in a backwards-compatible fashion that doesn't require the referencing archetypes to be revised. *rmVersion* exists to support this particular situation and records the specific RM version that the archetype was built to constrain.

**Associations**

• public definition : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134) [1]

A constraint on a complex object, which will typically consist of other constraints

• public rules : [RuleStatement](#_f8740e8d27529166da46265bd8521c94) [0..\*]

Abstract parent of all statement types

• public archetypeMetadata : [AuthoredResource](#_47dea9d0676ad6870be946fa52e870ad) [0..1]

*AuthoredResource* carries a minimal set of information about the source and origin of an *Archetype*. Its intent is to be a "connection point" to attach additional workflow and other provenance information to the target *Archetype.*

• public archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

### <Class> AttributeConstraint

**Description**

A constraint on a reference model attribute

**Direct Known Subclasses (Specialization)**

[AttributeCollectionConstraint](#_5eefba8eca7402f09bd5619804038771), [SingularAttributeConstraint](#_48ee2586ffa14e5bb1cf8ad893969da7)

**Attributes**

• public rmAttributeName : String [1]

Name of attribute within the reference model that is constrained by this node

• public existence : [AttributeExistence](#_4f99fbfcf9617d7ad55eca111d84fb67) [1]

Strength of requirement that the attribute instance be present

• public matchNegated : Boolean [1]

Whether the match operator is to be inverted so that the constraint specifies anything except what is represented

**Associations**

• public attribute : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134)

A constraint on a complex object, which will typically consist of other constraints

• public constrains : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public parent : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public object : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Constraints**

• name

[OCL]

rmAttributeName = parentProperty.name

### <Class> ComplexObjectConstraint

**Description**

A constraint on a complex object, which will typically consist of other constraints

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Attributes**

• public assumedValue : [RMClassInstance](#_935cd7de4b22d47dd2c6aef93bed5c7a) [0..1]

Value to be assumed in instances in which no value is provided

**Associations**

• public attributeTuple : [AttributeTupleConstraint](#_f6da15c71717330ae1b56f8b41e3dd51) [0..\*]

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

• private targetObject : [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36) [0..\*]

A constraint defined by reference to a node defined elsewhere in the same archetype

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

**Constraints**

• instanceOfConstraint

[English]

If assumedValue exists, assumedValue.classifier must be equal to or a specialization of self.parent

### <Class> EnumerationConstraint

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[LocalEnumerationConstraint](#_fac1fd23e79b0fc3d709d92006b38e40)

**Attributes**

• public possibleValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of possible enumeration literals that are valid in the constrained instance. If *possibleValues* is empty, all literals not referenced in *disallowValues* are valid.

• public disallowValues : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..\*]

The set of enumeration literals that can't appear in this constrained instance.

• public assumedValue : [RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22) [0..1]

**Associations**

• public constrains : [RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e) [1]

A subset of the UML::Enumeration data type. While UML::Enumeration data types can have both ownedAttributes and ownedOperations, these aspects are ignored from the AML perspective. The only aspects of an RMMEnumeration that are visible in the AML model is the package name.

**Constraints**

• pvValues

The list of possible values must be a subset of the set of possible enumeration literals.

[OCL2.0]

self.possibleValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• dvValues

The list of disallowed values must be in the set of enumeration literals.

[OCL2.0]

self.disallowValues->asSet()->forAll(d | self.parentClass.ownedLiteral->exists(p | p=d))

• pORd

An constraint may either specify possible values or disallow values but not both.

[OCL2.0]

possibleValues->size() = 0 or disallowValues->size() = 0

### <Class> ItemDescription

**Description**

A human readable designation and optional description of an object model entity in a specified language.

**Attributes**

• public language : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]

The language in which the term is defined

• public designation : String [1]

description of the meaning of the term

• public description : String [0..1]

description of the meaning of the term

**Associations**

• public : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

### <Class> NamedObjectConstraint

**Description**

Abstract model of constraint on any kind of object node

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootProxy](#_12e58855caae51d65fb43e2837534f63), [ArchetypeSlot](#_e518b2b75b6f66417345772b8440e6f2), [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134), [EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f)

**Attributes**

• public rmTypeName : String [1]

Reference model type that this node constrains

• public redefinability : [Redefinability](#_45bc3b03e253b26272fb450b2c34f5f2) [1]

Whether this node can be further constrained or elaborated in specializations

**Associations**

• public constrains : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> ObjectConstraintProxy

**Description**

A constraint defined by reference to a node defined elsewhere in the same archetype

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

### <Class> PrimitiveObjectConstraint

**Description**

A constraint on an instance of a primitive data type (see: [Primitive Data Types package](platform:/resource/metamodel/am.emx#_-pgJIByrEeONZZvjZFK4_A)) a Terminology Code Reference (See: [Core package of the Terminology Services module](platform:/resource/metamodel/am.emx#_FkmfQJgMEeOEysZ5-LoitA)) or an RMMEnumeration as defined in the [Enumeration Metamodel](platform:/resource/metamodel/am.emx#_BgPdsJW2EeOEysZ5-LoitA).

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[BooleanConstraint](#_72963a51b64d10f1a0ab72b99a7d95f7), [DateConstraint](#_05406539ea2335c7e43c699988cdb385), [DateTimeConstraint](#_2e2f2a2affb8ef26918289abc068cad2), [DurationConstraint](#_ac453484c1a5116843147a38bd4020c0), [IntegerConstraint](#_05ad521706377b116680c20824646d90), [RealConstraint](#_c4e09895097a057d7e6ce4b0d10c8967), [StringConstraint](#_e78c0feb207cbea2ca9911ec94e2a83e), [TimeConstraint](#_20b17cf4d1a1f7228a809f6ed68b3a0c)

**Associations**

• public constrains : [RMMDataType](#_d5914eb0da42172989bbe57f23fc4310) [1]

*RMMDataTypes*, like UML::DataTypes "model Types whose instances are distinguished only by their value". *RMMDataTypes* form the leaf nodes of any AML constraint model -- they are the places where actual atomic value instances are recorded.

While not formally represented in this model (because we don't know how to create a generalization set), the three subclasses of *RMMDataType* (*RMMEnumeration*, *CompoundRMMDataType* and *UMLPrimitiveType*) are both disjoint and covering.

### <Class> TerminologyConstraint

**Description**

A constraint on instances of the reference model Terminology type

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Attributes**

• public namespace : [NameSpace](#_b448a755fcce2af9da8221a55b6b53ff) [0..1]

• public uriRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a URI is required

• public identifierRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a concept identifier is required

• public designationRequirement : [ValidityKind](#_411de44f0fb7bb89a16c1f6c35fcd7d9) [0..1]

Whether a human-readable designation is required

• public assumedValue : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

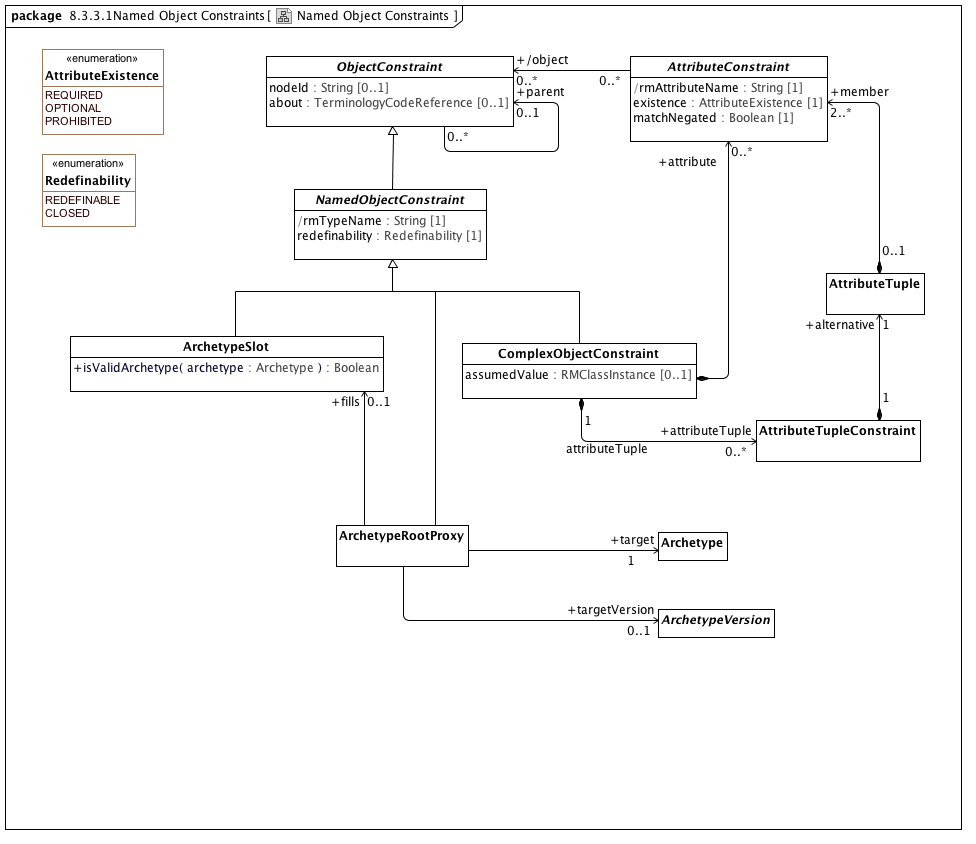
**Associations**

• public valueSet : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..1]

• public constrains : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

#### <Package> Named Object Constraints



**Named Object Constraints**

### <Class> Archetype

**Attributes**

• public archetypeId : [ArchetypeId](#_abe68de6d7b599f5e4ea361caee12c81) [1]

The unique archetype identifier. However constructed, this uniquely identifies the archetype across its entire life cycle. No other archetype can have this identifier and a different identifier designates a different Archetype.

• public archetypeName : String [1]

The human readable name of the Archetype. This is typically derived from the other archetype details (See: openEHR Knowledge Artefact Identification - Revision 0.7.0 for an example). It is possible for this identifier to change over the life of an Archetype.

• public rmPackagePath : String [1]

The qualifiedName of a package in the target reference model that has the root *rmClass* as a visible member (there can be more than one possibility in a reference model).

• public rmClassName : String [1]

Name of the root class of this archetype. *rmClass* must match the *visibleName* of the class referenced by the *ComplexObjectConstraint* target of the *Archetype definition* as well as the visibleName of the RMMClass instance that it constrains.

• public originalLanguage : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

The original language in which the resource was authored (essential for evaluating natural language quality)

• public urn : String [0..1]

**Associations**

• public rmUMLModel : [RMMModel](#_fc116c5fcb379006ed51eb855a1dae57) [1]

Quoting the UML 2.5 specification, "A Model is a description of a system, where ‘system’ is meant in the broadest sense and may include not only software and

hardware but organizations and processes. It describes the system from a certain viewpoint (or vantage point) for a certain

category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction. A Model is

complete in the sense that it covers the whole system, although only those aspects relevant to its purpose (i.e., within the given

level of abstraction and viewpoint) are represented in the Model."

From the AML perspective, the "aspects relative to [the model's] purpose" consist of a collection of packages which in turn contain a set of RMMClass definitions. To be used in AML, a model *must* be identified by a unique URI.

• public rmPackage : [RMMPackage](#_a0a843d7d41881592e31e887cebd6da4) [1]

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

• public version : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

• public specializes : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..\*]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• uri

[OCL]

rmURI = rmUMLModel.URI

• package

[OCL]

rmPackagePath = rmPackage.qualifiedName

• class

[OCL]

rmClassName = constrains.name

• classpackage

[OCL]

rmPackage.member->exists(c|c=constrains)

• differentArchetype

An archetype cannot specialize itself or any of its descendants (note - only self is included in formal OCL)

[OCL2.0]

not (self = self.specializes)

### <Class> ArchetypeRootProxy

**Description**

A specialization of ComplexObjectConstraint whose node\_id attribute is an archetype identifier rather than an internal node code. Used to reference external archetypes to be included in a composite archetype.

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Associations**

• public target : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

• public fills : [ArchetypeSlot](#_e518b2b75b6f66417345772b8440e6f2) [0..1]

A classifier that describes the set of archetypes that may be used to validate instances

• public targetVersion : [ArchetypeVersion](#_1de96fa71501cf96b27b14f3f9f1bb99) [0..1]

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Constraints**

• validSlot

[Analysis]

If there is a target fills ArchetypeSlot, the ArchetypeRootProxy archetypeId must be a member of one or more includes value sets and not be a member of one or more excludes value sets as identified in the ArchetypeSlot or any of its parents or ancestors

• validTarget

[OCL]

target.archetypeId = archetypeId and target.archetypeName = archetypeName -- Also

### <Class> ArchetypeSlot

**Description**

A classifier that describes the set of archetypes that may be used to validate instances

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Operations**

• public isValidArchetype (archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75)) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

### <Class> ArchetypeVersion

**Description**

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Direct Known Subclasses (Specialization)**

[FlatArchetype](#_166addccec3fe63279b3f9eeb9930ab8), [SourceArchetype](#_dcaf9716e9bc2255b93c20393e8712f2)

**Attributes**

• public archetypeVersionId [1]

The specific version of this *Archetype*. *archetypeVersion* does not impact archetype identity. If an archetype undergoes non-backwards compatible changes, it becomes a new archetype with a new identifier.

• public amlVersion : String [1]

The URI of the modeling language and version used to construct this Archetype, if derived from a serialized representation

• public rmRelease : String [0..1]

The specific version of the reference model that was constrained. Depending on the context and workflow model, it may be possible to update a reference model in a backwards-compatible fashion that doesn't require the referencing archetypes to be revised. *rmVersion* exists to support this particular situation and records the specific RM version that the archetype was built to constrain.

**Associations**

• public definition : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134) [1]

A constraint on a complex object, which will typically consist of other constraints

• public rules : [RuleStatement](#_f8740e8d27529166da46265bd8521c94) [0..\*]

Abstract parent of all statement types

• public archetypeMetadata : [AuthoredResource](#_47dea9d0676ad6870be946fa52e870ad) [0..1]

*AuthoredResource* carries a minimal set of information about the source and origin of an *Archetype*. Its intent is to be a "connection point" to attach additional workflow and other provenance information to the target *Archetype.*

• public archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

### <Class> AttributeConstraint

**Description**

A constraint on a reference model attribute

**Direct Known Subclasses (Specialization)**

[AttributeCollectionConstraint](#_5eefba8eca7402f09bd5619804038771), [SingularAttributeConstraint](#_48ee2586ffa14e5bb1cf8ad893969da7)

**Attributes**

• public rmAttributeName : String [1]

Name of attribute within the reference model that is constrained by this node

• public existence : [AttributeExistence](#_4f99fbfcf9617d7ad55eca111d84fb67) [1]

Strength of requirement that the attribute instance be present

• public matchNegated : Boolean [1]

Whether the match operator is to be inverted so that the constraint specifies anything except what is represented

**Associations**

• public attribute : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134)

A constraint on a complex object, which will typically consist of other constraints

• public constrains : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public parent : [RMMProperty](#_652433ba6af347b5ab3d4b0b4b2931c9) [1]

A Reference Model Property (RMProperty) is a proper subset of a UML property. The aspects of a property that can be addressed by archetype includes:

The property name. Only named properties may be constrained within a reference model. Referring to section 7.4.3 in the ptc/2013-09-05, a non-hidden *RMProperty* is always referred to in its unqualified form. If it is necessary to reference hidden elements within an archetype, the qualified name (N::x) form should be used. Qualification should be the minimum that sufficient to render the name unique.

The lower and upper bounds. The UML MultiplicityElement shows lower and upper as derived properties. The AML specification assumes that, if present, these properties have been computed and it is up to the implementer to correctly interpret MultiplicityElement lowerValue and upperValue properties to determine these results.

Derived properties cannot be constrained using AML -- the rationale being that (a) derived properties may or may not be present in object instances and (b) the primary constraints need to be applied to the parameters of the derivation rather than the result.

UML::Associations are not used in AML -- the model is traversed via the *ownedAttribute* association. AssociationClasses are not differentiated from any other RMMClass in the model, and may be referenced and traversed via. whatever *ownedAttribute*/type links that are available.

While default values can be specified in the Reference Model, they are ignored in AML. Note, however, that AML can specify default values (with tighter semantics) in an archetype.

All other UML::Property links, including aggregation, isComposite, isID, association, qualifier, opposite, defaultVelue, redefines, subsettedProperty, and interface are ignored within the AML profile.

• public object : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855) [0..\*]

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Constraints**

• name

[OCL]

rmAttributeName = parentProperty.name

### <Class> AttributeTuple

**Description**

A set of constraints on related attributes to be used to differentiate scenarios where the value of one attribute affects the valid values of another

**Associations**

• public member : [AttributeConstraint](#_11f887fb6f19248bf7193bca31772c05) [2..\*]

A constraint on a reference model attribute

### <Class> AttributeTupleConstraint

**Description**

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

**Associations**

• public alternative : [AttributeTuple](#_6d5bfb351e19f61e0327587b0ff5fd4f) [1]

A set of constraints on related attributes to be used to differentiate scenarios where the value of one attribute affects the valid values of another

### <Class> ComplexObjectConstraint

**Description**

A constraint on a complex object, which will typically consist of other constraints

**Direct Known Superclasses (Generalization)**

[NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9)

**Attributes**

• public assumedValue : [RMClassInstance](#_935cd7de4b22d47dd2c6aef93bed5c7a) [0..1]

Value to be assumed in instances in which no value is provided

**Associations**

• public attributeTuple : [AttributeTupleConstraint](#_f6da15c71717330ae1b56f8b41e3dd51) [0..\*]

An AttributeTupleConstraint presents a set of two or more alternative tuples, each of which consists of two or more attributes. The containing ComplexObjectConstraint is satisfied when all of the constraints in one of the AttributeTuples are satisfied.

• private targetObject : [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36) [0..\*]

A constraint defined by reference to a node defined elsewhere in the same archetype

• public constrains : [RMMClass](#_a75c06fc93e516ccf92a1e38e18c46f3) [1]

A class, in the object-oriented sense

**Constraints**

• instanceOfConstraint

[English]

If assumedValue exists, assumedValue.classifier must be equal to or a specialization of self.parent

### <Class> NamedObjectConstraint

**Description**

Abstract model of constraint on any kind of object node

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootProxy](#_12e58855caae51d65fb43e2837534f63), [ArchetypeSlot](#_e518b2b75b6f66417345772b8440e6f2), [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134), [EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f)

**Attributes**

• public rmTypeName : String [1]

Reference model type that this node constrains

• public redefinability : [Redefinability](#_45bc3b03e253b26272fb450b2c34f5f2) [1]

Whether this node can be further constrained or elaborated in specializations

**Associations**

• public constrains : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [RMMClassifier](#_31f3bed9860f1a34043799bd12ffe873) [1]

A generalization of *RMDataType* and *RMClass*, both of which have *RMProperty*s

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Enumeration> AttributeExistence

**Description**

Strengths of requirement for the existence of an attribute

**Enumeration Literals**

* **OPTIONAL**

The attribute may be present in a conforming instance

* **PROHIBITED**

The attribute must not be present in a conforming instance

* **REQUIRED**

The attribute must be present in a conforming instance

### <Enumeration> Redefinability

**Description**

Whether a node can be further constrained or elaborated in specializations

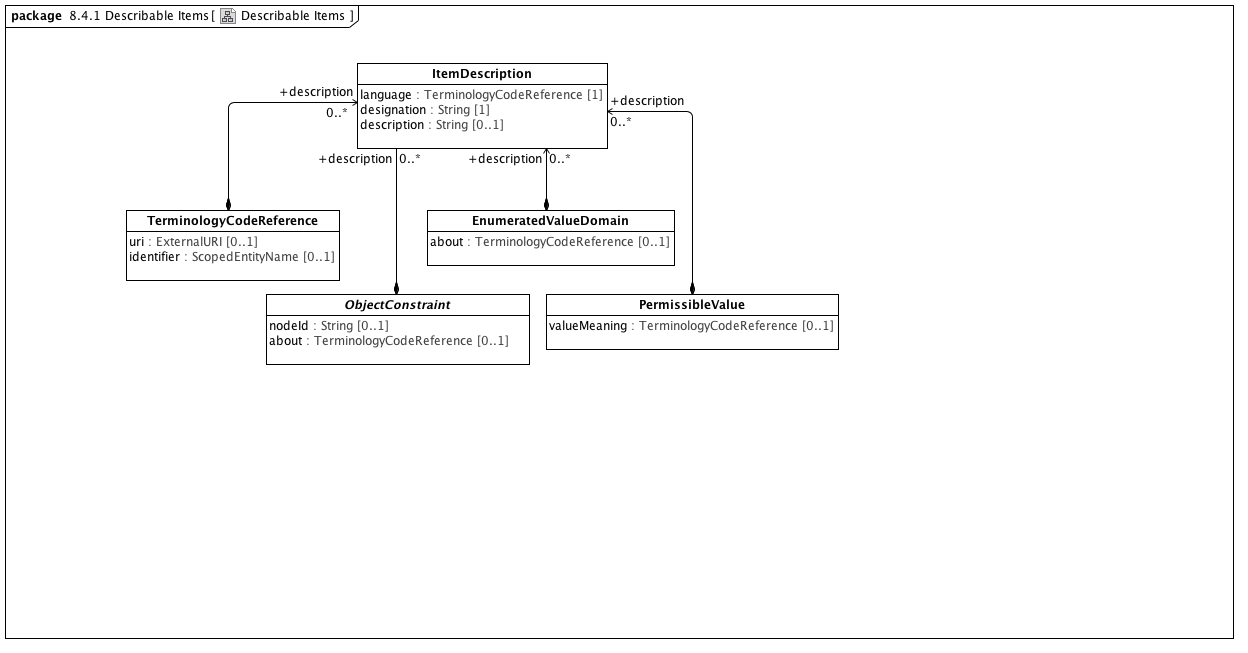
**Enumeration Literals**

* **CLOSED**

* **REDEFINABLE**

## <Package> Terminology Object Model

### <Package> Describable Items



**Describable Items**

This diagram shows the classes that may be associated with one or more Item Descriptions. Each of these entries corresponds to an entry in the ADL 1.5 terminology section:

* ObjectConstraint - this represents the "id" codes in ADL -- the "term definitions" of object constraints
* EnumeratedValueDomain - this represents the "ac" codes in ADL -- the "term definitions" of internal and external value sets
* PermissibleValue - this represents the "at" codes in ADL -- the "term definition" of individual codes

In addition, terminology code references may be accompanied by one or more Item Descriptions that represent the intent of the code reference on a strictly informative basis.

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> ItemDescription

**Description**

A human readable designation and optional description of an object model entity in a specified language.

**Attributes**

• public language : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]

The language in which the term is defined

• public designation : String [1]

description of the meaning of the term

• public description : String [0..1]

description of the meaning of the term

**Associations**

• public : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

### <Class> ObjectConstraint

**Description**

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

**Direct Known Subclasses (Specialization)**

[EnumerationConstraint](#_42c2e4f902eddd2a1629a431a96cd94f), [NamedObjectConstraint](#_ab5b3b01964560abb1047edd9efa4eb9), [ObjectConstraintProxy](#_6da4a9bc7db41a2b89064f79f0c4ed36), [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da), [TerminologyConstraint](#_b2d4edbc24f651e5a3d756933fff1326)

**Attributes**

• public nodeId : String [0..1]

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public parent : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> TerminologyCodeReference

**Description**

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

**Direct Known Subclasses (Specialization)**

[AttributeName](#_e2995e77bf0a43587adc571de3d5131b)

**Attributes**

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

A URI that resolves to the full EntityDescription represented by this resource.

• public identifier : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [0..1]

A namespace/name combination that uniquely represents the entity. This can be the primary entityID, as determined by the service or any valid alternateId. Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering URIAndEntityName instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the “fully specified name” or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

**Constraints**

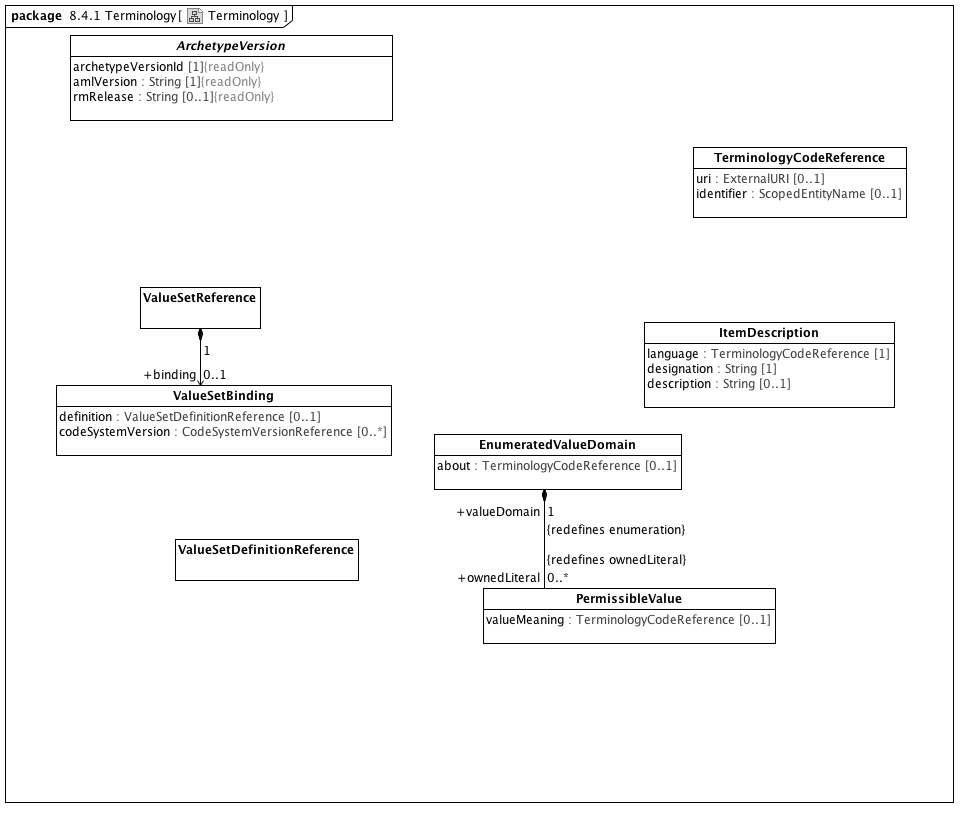
• identifierOrURI

A TerminologyCodeReference must either have a *uri*, an *identifier*, or both.

[OCL]

uri->notEmpty() or identifier->notEmpty()

### <Package> Terminology



**Terminology**

### <Class> ArchetypeVersion

**Description**

An Archetype is a set of constraints that can be applied as a predicate against instances of the Reference Model class constrained by the *ComplexObjectConstraint* *definition.*

**Direct Known Subclasses (Specialization)**

[FlatArchetype](#_166addccec3fe63279b3f9eeb9930ab8), [SourceArchetype](#_dcaf9716e9bc2255b93c20393e8712f2)

**Attributes**

• public archetypeVersionId [1]

The specific version of this *Archetype*. *archetypeVersion* does not impact archetype identity. If an archetype undergoes non-backwards compatible changes, it becomes a new archetype with a new identifier.

• public amlVersion : String [1]

The URI of the modeling language and version used to construct this Archetype, if derived from a serialized representation

• public rmRelease : String [0..1]

The specific version of the reference model that was constrained. Depending on the context and workflow model, it may be possible to update a reference model in a backwards-compatible fashion that doesn't require the referencing archetypes to be revised. *rmVersion* exists to support this particular situation and records the specific RM version that the archetype was built to constrain.

**Associations**

• public definition : [ComplexObjectConstraint](#_abfab8c8e983a73b4981f6fcfdd16134) [1]

A constraint on a complex object, which will typically consist of other constraints

• public rules : [RuleStatement](#_f8740e8d27529166da46265bd8521c94) [0..\*]

Abstract parent of all statement types

• public archetypeMetadata : [AuthoredResource](#_47dea9d0676ad6870be946fa52e870ad) [0..1]

*AuthoredResource* carries a minimal set of information about the source and origin of an *Archetype*. Its intent is to be a "connection point" to attach additional workflow and other provenance information to the target *Archetype.*

• public archetype : [Archetype](#_f45a7b68ecac449e953ff8a65d6eff75) [1]

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> ItemDescription

**Description**

A human readable designation and optional description of an object model entity in a specified language.

**Attributes**

• public language : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]

The language in which the term is defined

• public designation : String [1]

description of the meaning of the term

• public description : String [0..1]

description of the meaning of the term

**Associations**

• public : [ObjectConstraint](#_aa52f11e5760ad2f47030803962bb855)

ObjectConstraint represents the properties and associations that are common to all types of object constraints:

Every object constraint may directly specialize at most one parent ObjectConstraint

Every AttributeConstraint is owned by exactly one ObjectConstraint

Every ObjectConstraint is referenced by exactly one AttributeConstraint, with the exception of the root Archetype definition ComplexObjectConstraint that is not owned by any referencing attribute.

The subtypes of ObjectConstraint include:

ObjectConstraintProxy - a reference to an existing NamedObjectConstraint. ObjectConstraintProxys only exist in SourceArchetypes and are replaced by a copy of their targetObject during the flattening process.

NamedObjectConstraint - the set of ObjectConstraints that reference a Reference Model Class and have node identifiers

EnumerationConstraint - constraints on the Reference Model Enumeration class

ArchetypeSlot - identifies a (constrained) slot to be filled by a separate archetype

ArchetypeRootProxy - references an archetype that constraints the type and/or attributes of a Reference Model Class and optionally fills an ArchetypeSlot defined in a parent Archetype

ComplexObjectConstraint - a constraint on the type and/or attributes of a Reference Model Class

TerminologyConstraint - constraints on the TerminologyCodeReference type

PrimitiveObjectConstraint - constraints on the set of primitive data types supplied in the reference model

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> TerminologyCodeReference

**Description**

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

**Direct Known Subclasses (Specialization)**

[AttributeName](#_e2995e77bf0a43587adc571de3d5131b)

**Attributes**

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

A URI that resolves to the full EntityDescription represented by this resource.

• public identifier : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [0..1]

A namespace/name combination that uniquely represents the entity. This can be the primary entityID, as determined by the service or any valid alternateId. Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering URIAndEntityName instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the “fully specified name” or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

**Constraints**

• identifierOrURI

A TerminologyCodeReference must either have a *uri*, an *identifier*, or both.

[OCL]

uri->notEmpty() or identifier->notEmpty()

### <Class> ValueSetBinding

**Description**

An externally specified set of coded values

**Attributes**

• public definition : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..1]

The version of the value set definition

• public codeSystemVersion : [CodeSystemVersionReference](#_6cc1b578ac4ab07a6712e0f4fa94db8b) [0..\*]

The version of the controlled terminology from which the values are selected

### <Class> ValueSetDefinitionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Associations**

• public possibleValue : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..\*]

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

• public definedValueSet : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7) [1]

The URI, identifier and name of a collection of TerminologyCodeReferences

### <Class> ValueSetReference

**Description**

The URI, identifier and name of a collection of TerminologyCodeReferences

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Associations**

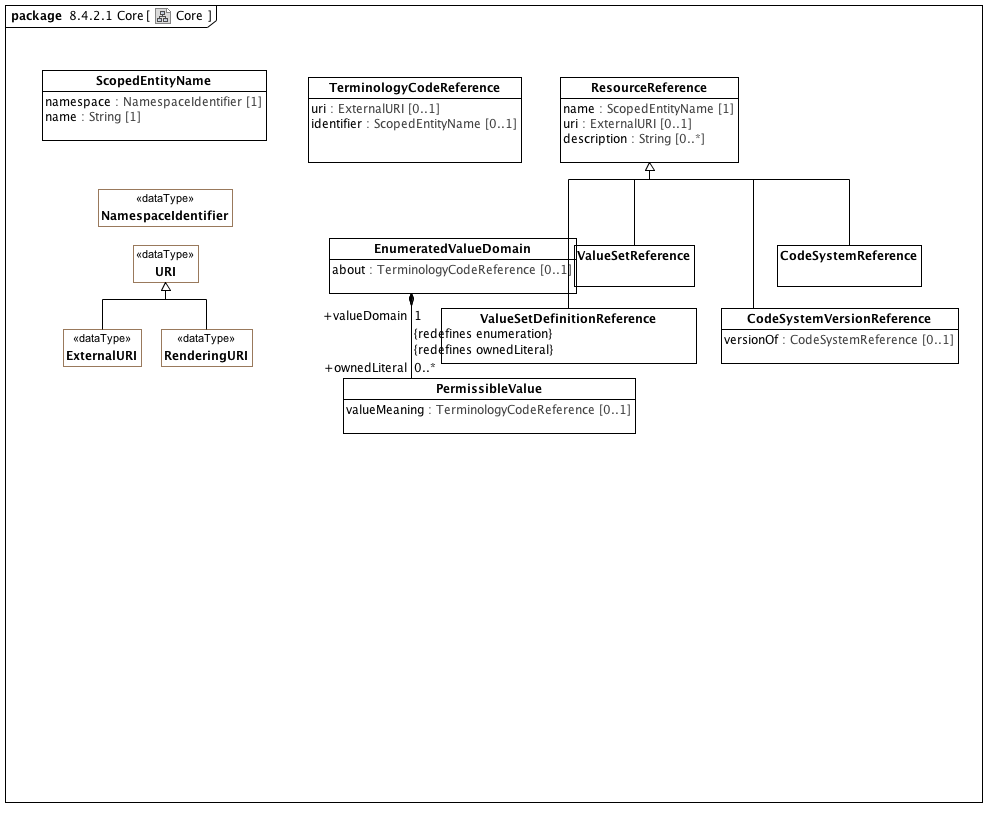
• public binding : [ValueSetBinding](#_fdb7a97d4c43d7f09387cdc69dd2f65a) [0..1]

An externally specified set of coded values

• public definition : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..\*]

### <Package> Terminology Services

#### <Package> Core



**Core**

### <DataType> ExternalURI

**Description**

A URI that references an "real world" (vs. digital) entity. Examples include individual people, locations, organizations as well as abstract concepts or classes. "Well behaved" External URI's do not directly reference a digital resource, although they may result in a redirection to a *RenderingURI* that resolves to a description of the actual target.

**Direct Known Superclasses (Generalization)**

[URI](#_887928f30f99c8a1ca89ed7a082356aa)

### <DataType> NamespaceIdentifier

**Description**

An identifier that uniquely references the scoping namespace of an Entity (class, role or individual) within a the context of a service. *NamespaceIdentifier* syntax must match the [PNAME\_NS](http://www.w3.org/TR/rdf-sparql-query/#rPNAME_NS) production as defined in the [SPARQL Query Specification](http://www.w3.org/TR/rdf-sparql-query/).

### <DataType> RenderingURI

**Description**

A URI that represents a digital resource, such as a page in a REST service, an online document or other digital artifact.

**Direct Known Superclasses (Generalization)**

[URI](#_887928f30f99c8a1ca89ed7a082356aa)

### <DataType> URI

**Description**

A Universal Resource Identifier (URI) as defined in [IETF 3986](http://www.ietf.org/rfc/rfc3986.txt) . Implementations are encouraged to consider implementing this data type using the IRI ([RFC3987](http://www.ietf.org/rfc/rfc3987.txt)) specification.

**Direct Known Subclasses (Specialization)**

[ExternalURI](#_de932b9629138c166e8cfb00efa65177), [RenderingURI](#_821273fdc1c3295a17225200782229ea)

### <Class> CodeSystemReference

**Description**

The URI, identifier and name of a code system, as defined in the CTS2 specification.

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

### <Class> CodeSystemVersionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Attributes**

• public versionOf : [CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7) [0..1]

### <Class> EnumeratedValueDomain

**Description**

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

**Direct Known Superclasses (Generalization)**

[RMMEnumeration](#_190e24bd48f094ad9ad981ac0b4eb47e)

**Direct Known Subclasses (Specialization)**

[ValueSet](#_73d0fab5bddf198ab14a77c3fed1636a)

**Attributes**

• public about : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public ownedLiteral : [PermissibleValue](#_66976d5fcaf3eff9df49b6e5dab4ad12) [0..\*]

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

### <Class> PermissibleValue

**Description**

A permissible value within the context of a value domain. While permissible values may be represented as integers, strings or simply as named data type instances (as is the case in UML), all permissible values need to have a mechanism for providing a String representation of the represented value. The String returned by the value function must be unique within the context of the containing domain.

**Direct Known Superclasses (Generalization)**

[RMMEnumerationLiteral](#_41faf6a7041d7068bbbf4f9ff3924d22)

**Attributes**

• valueMeaning : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..1]

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

• public valueDomain : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

• public classifier : [EnumeratedValueDomain](#_ad639ee3d4cd535b2d3e55238d69cc51) [1]

An EnumeratedValueDomain represents a discrete set of possible values for a particular field or data element. Each permissible value represents an intended meaning that, while sometimes determinable from the string itself or its accompanying documentation, can only be fully fixed by connecting it to an official "value meaning" reference in an external terminological resource.

### <Class> ResourceReference

**Description**

**Direct Known Subclasses (Specialization)**

[CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7), [CodeSystemVersionReference](#_6cc1b578ac4ab07a6712e0f4fa94db8b), [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821), [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7)

**Attributes**

• public name : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [1]

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

• public description : String [0..\*]

A textual description of the resource in a human readable language.

### <Class> ScopedEntityName

**Description**

The combination of a *namespace* identifier and a local *name*. Scoped entity names are not portable - they only work within the context of a single service instance, as different services may assign different *namespace* identifiers to the same namespace and different services may make different choices of the appropriate local identifier to use to represent an entity. As an example, one service may choose to use the entity code while a second may use another designation that is known to be unique.

**Attributes**

• public namespace : [NamespaceIdentifier](#_94cb7eefb9b55dbc722d53bf1ec0f163) [1]

An identifier that references a unique namespace URI within the context of a service

• public name : String [1]

The local entity name within the context of the namespace. What is chosen for the entity name is service specific.

### <Class> TerminologyCodeReference

**Description**

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

**Direct Known Subclasses (Specialization)**

[AttributeName](#_e2995e77bf0a43587adc571de3d5131b)

**Attributes**

• public uri : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]

A URI that resolves to the full EntityDescription represented by this resource.

• public identifier : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [0..1]

A namespace/name combination that uniquely represents the entity. This can be the primary entityID, as determined by the service or any valid alternateId. Service implementers are encouraged to develop mechanisms that will allow clients to choose an appropriate namespace for rendering URIAndEntityName instances. As an example, it should be possible to view SNOMED-CT entity references by either the SctId, the “fully specified name” or, where appropriate, the CTV3ID or SNOMED-3 identifier. Similar mechanisms would apply to ontologies that have both id and label fields.

**Associations**

• public description : [ItemDescription](#_47e163911a910ae4a0de27029dcdf5dd) [0..\*]

A human readable designation and optional description of an object model entity in a specified language.

**Constraints**

• identifierOrURI

A TerminologyCodeReference must either have a *uri*, an *identifier*, or both.

[OCL]

uri->notEmpty() or identifier->notEmpty()

### <Class> ValueSetDefinitionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

**Associations**

• public possibleValue : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..\*]

A *TerminologyCodeReference* (alias: URIAndEntityName) consists of a local identifier that references a unique meaning within the context of a given domain in a terminology service instance and a globally unique *URI* that identifies the intended meaning of the identifier.

• public definedValueSet : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7) [1]

The URI, identifier and name of a collection of TerminologyCodeReferences

### <Class> ValueSetReference

**Description**

The URI, identifier and name of a collection of TerminologyCodeReferences

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_c92f346cfe19d59bae8a98c0ac6f9d71)

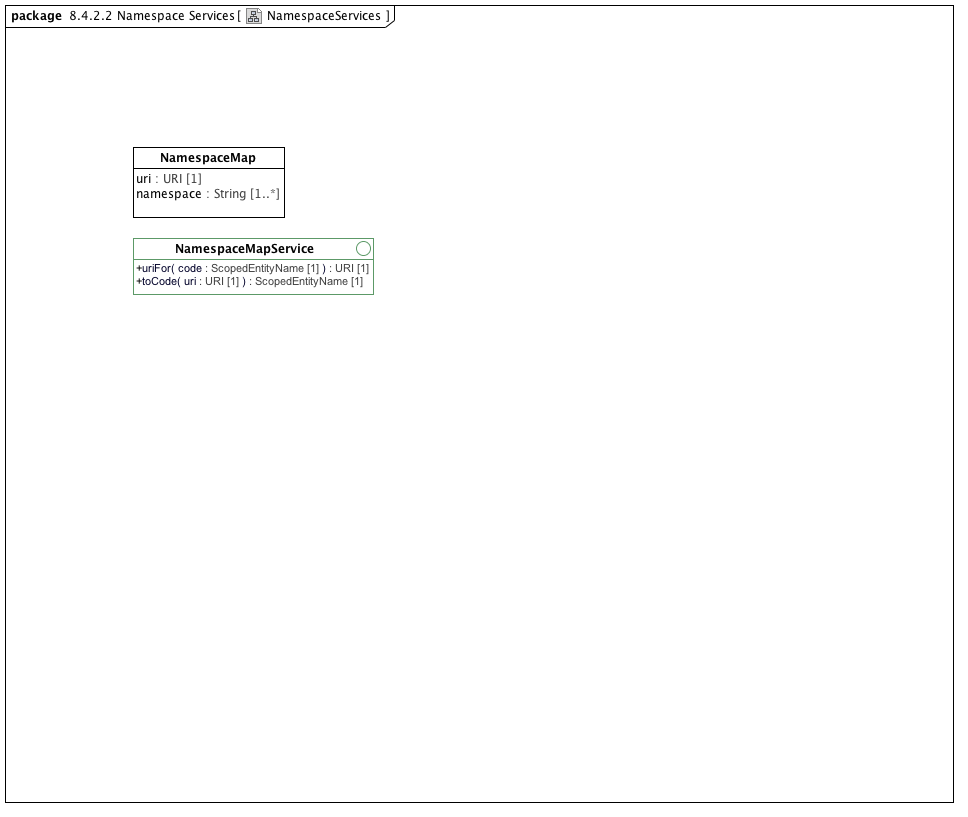
**Associations**

• public binding : [ValueSetBinding](#_fdb7a97d4c43d7f09387cdc69dd2f65a) [0..1]

An externally specified set of coded values

• public definition : [ValueSetDefinitionReference](#_be2600754ff104c3bebcfa73ab768821) [0..\*]

#### <Package> Namespace Services



**NamespaceServices**

### <Interface> NamespaceMapService

**Description**

A service that provides access to the collection of namespace maps that are used in a given service instance.

**Attributes**

• public knownMaps : [NamespaceMap](#_4b44c903b99cddb901d1475d3111286d) [1..\*]

The list of namespace maps that are known to the service.

**Operations**

• public uriFor (code : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [1]) : [URI](#_887928f30f99c8a1ca89ed7a082356aa)

Return a URI for the supplied namespace and name. Returns an empty string if none are known.

• public toCode (uri : [URI](#_887928f30f99c8a1ca89ed7a082356aa) [1]) : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6)

Return a namespace and name for the supplied URI. Both namespace and name are blank if the URI isn't recognized.

### <Class> NamespaceMap

**Description**

A URI and the set of namespace identifiers that represent it. Note that, as in the XML specification, it is possible for more than one namespace to reference the same URI

**Attributes**

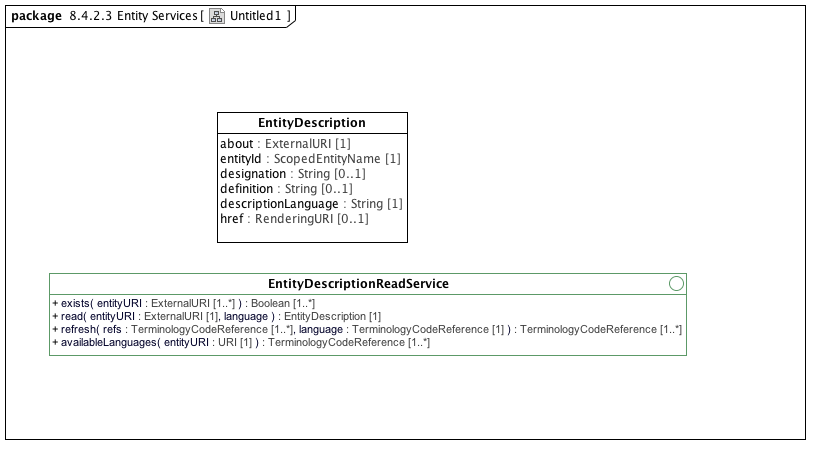
• public uri : [URI](#_887928f30f99c8a1ca89ed7a082356aa) [1]

A URI that represents a namespace.

• public namespace : String [1..\*]

One or more local identifiers that represent the associated *uri*.

#### <Package> Entity Services



**Untitled1**

### <Interface> EntityDescriptionReadService

**Description**

The EntityDescriptionReadService provides a mechanism to determine whether a given terminology code URI is known to the service, to retrieve a set of entity descriptions in a given language and to refresh the contents of set of terminologyCodeReferences.

**Attributes**

• public supportedLanguages : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1..\*]

The languages that are recognized by the read service. The presence of a language code indicates that at least some of the resources within the service are accessible in a given language. *supportedLanguage* returns *TerminologyCodeReferences* to allow the language designation to be supplied in the native language of the user. Note that, in many implementations, it should be possible to ignore the URI and use the *identifier.name* as a language code.

**Operations**

• public exists (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1..\*]) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

Determine whether the supplied set of entityURI's are known to the service.

• public read (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], language [1]) : [EntityDescription](#_c59b63ad054463f448827db0b03882ed)

Return a complete EntityDescription for the supplied entityURI and language code.

• public refresh (refs : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1..\*], language : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]) : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4)

Refresh the list of TerminologyCodeReferences to carry designations, if known, in the supplied language. The namespace and name part of the ScopedEntityName may also be updated to represent the services local namespace identifiers and/or preferred names.

• public availableLanguages (entityURI : [URI](#_887928f30f99c8a1ca89ed7a082356aa) [1]) : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4)

Return a list of the languages that are available for the specific entity.

### <Class> EntityDescription

**Description**

A short description of a terminology code.

**Attributes**

• public about : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1]

The URI of the entity being described.

• public entityId : [ScopedEntityName](#_bf3eeb4d95f5d93bbd59440cca5ed9d6) [1]

A local namespace and name for the referenced entity.

• public designation : String [0..1]

A name or designation of the entity that is considered compatible with the description language. If not present, no useful designation is available for that language

• public definition : String [0..1]

A textual description or definition of the entity, if any, compatible with the description language. If not present, no known definition is available for the supplied language

• public descriptionLanguage : String [1]

The language of the designation and/or definition.

• private href : [RenderingURI](#_821273fdc1c3295a17225200782229ea) [0..1]

A URI that, when dereferenced using an HTTP service, should provide additional information about the entity. In AML implementations that are based on the CTS2 specification, this URI will return a CTS2 EntityDescription of the referenced entity.

#### <Package> Association Services



**Association Services**

### <Interface> AssociationQueryService

**Operations**

• public specializationsOf (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], direct [1]) : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4)

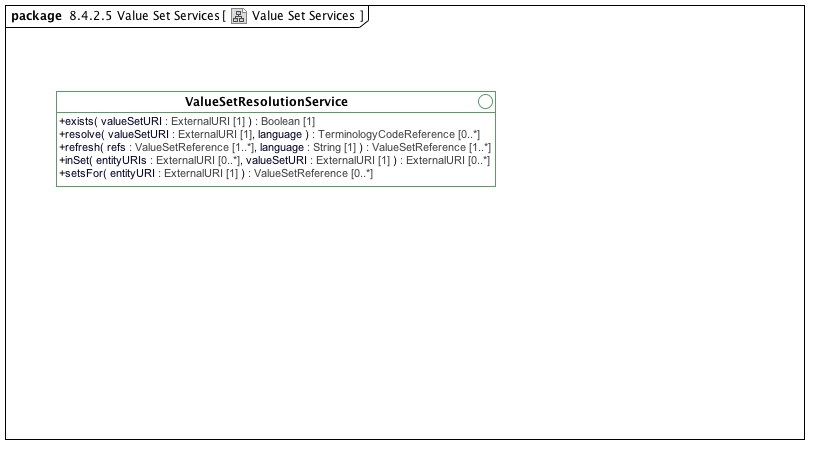
Return the set of identifiers that have a specialization relationship with the supplied entityURI in the known set of AML models.

• public specializes (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], direct : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1]) : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4)

Return the set of identifiers that the referenced entityURI specializes within the context of the known AML models.

• public isSpecializationOf (parent : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], child : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], direct : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1] = true)

#### <Package> Value Set Services



**Value Set Services**

### <Interface> ValueSetResolutionService

**Attributes**

• public supportedLanguages : String [1..\*]

The set of languages that are known to the value set resolution service. Used to assign the appropriate designation in the *resolve* method as well as the description string of the *refresh* method

**Operations**

• public exists (valueSetURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1]) : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

Determine whether the supplied value set URI can be resolved by the service.

• public resolve (valueSetURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1], language [1]) : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4)

Resolve the supplied value set URI returning the set of terminology codes in the value set. The list is determined by external parameters -- the specific versions of the code system(s) and definition versions are determined externally. A full CTS2 ValueSetResolution service can be used to control and/or access finer grained parameters.

• public refresh (refs : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7) [1..\*], language : [String](#_e8a6ce315d976318da3ab784a645ea44) [1]) : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7)

Refresh the textual descriptions of the value set to reflect the supplied language

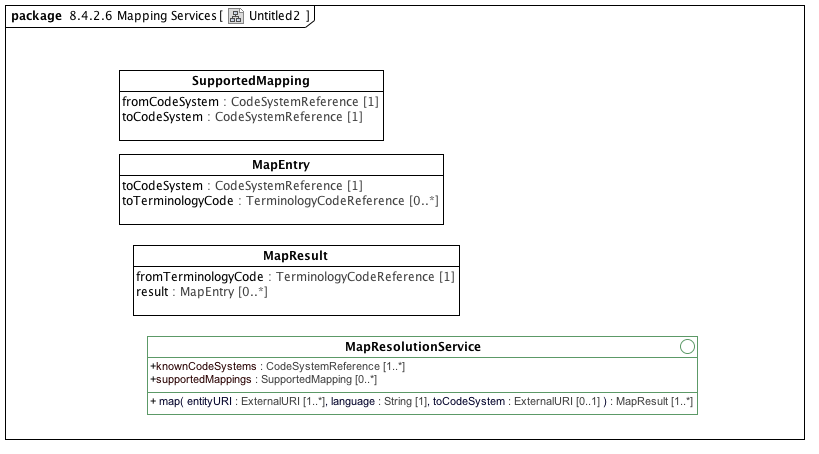
• public inSet (entityURIs : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..\*], valueSetURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1]) : [ExternalURI](#_de932b9629138c166e8cfb00efa65177)

Determine which the set of entity URI's are in the supplied value set.

• public setsFor (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1]) : [ValueSetReference](#_53376ea1584b6547b15f0e1392fc93e7)

Return the list of value sets whose resolution includes the supplied entity reference.

#### <Package> Mapping Services



**Untitled2**

### <Interface> MapResolutionService

**Attributes**

• public knownCodeSystems : [CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7) [1..\*]

Return the list of external code systems known to the map resolution service.

• public supportedMappings : [SupportedMapping](#_17b99b0497deaac3ae6ae3c36da9bf16) [0..\*]

Return the list of partial or full mappings supported by the service.

**Operations**

• public map (entityURI : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [1..\*], language : [String](#_e8a6ce315d976318da3ab784a645ea44) [1], toCodeSystem : [ExternalURI](#_de932b9629138c166e8cfb00efa65177) [0..1]) : [MapResult](#_6661810602be647d1d24466ce9ac73b7)

Provide the mappings for the supplied list of entityURI's to the target code system, or return all known maps if no code system is provided. The return MapResults are in the same order as the supplied terminology code references

### <Class> MapEntry

**Description**

A mapping for a terminology code in a given code system.

**Attributes**

• public toCodeSystem : [CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7) [1]

The target code system.

• public toTerminologyCode : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [0..\*]

The set of terminology codes that the from code maps to in the target code system. Note that the absence of a *toTerminologyCode* indicates the mapping has positively asserted that no mapping for the source code exists.

### <Class> MapResult

**Description**

The result of a mapping.

**Attributes**

• public fromTerminologyCode : [TerminologyCodeReference](#_4d1f571ab5e9384786ffe39444e822b4) [1]

The terminology code reference for the code that was mapped from.

• public result : [MapEntry](#_7e4c9687f16d9fee85e09a5f98bf4dfd) [0..\*]

The set of mappings for *fromTerminologyCode*, if any.

### <Class> SupportedMapping

**Description**

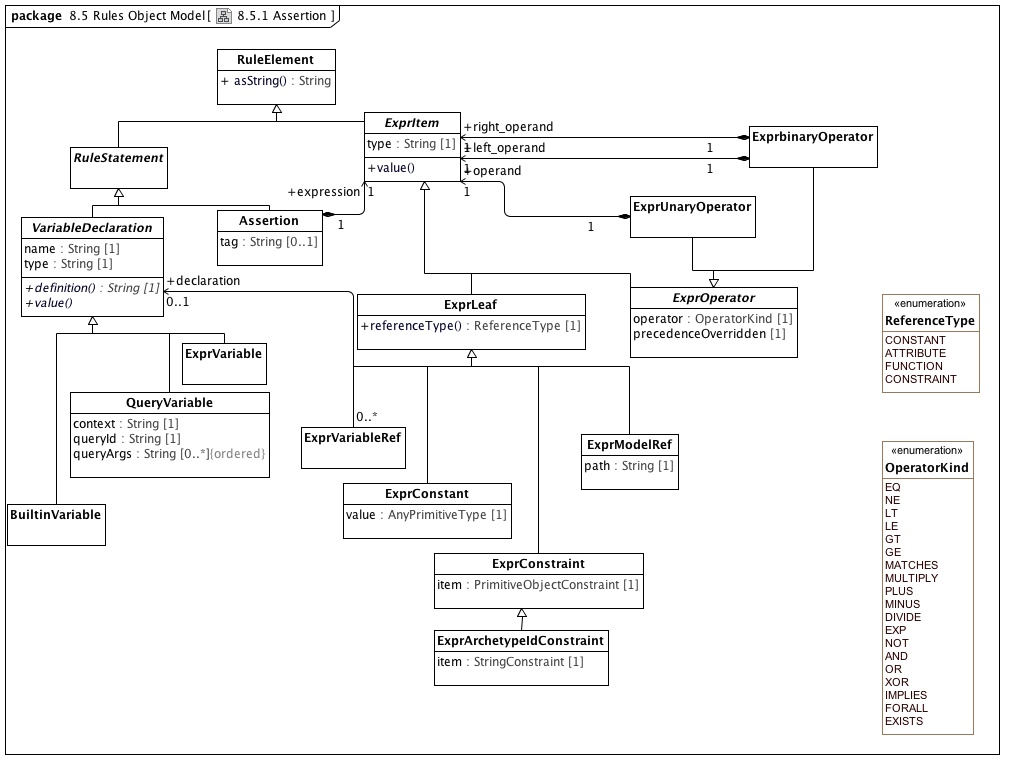
A from and to tuple that identifies the availability of a partial or complete map between codes in two different code systems.

**Attributes**

• public fromCodeSystem : [CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7) [1]

• public toCodeSystem : [CodeSystemReference](#_0ff5bf5f2f7cc9422c34b4bfd05628e7) [1]

## <Package> Rules Object Model



**8.5.1 Assertion**

### <Class> Assertion

**Description**

A first order predicate logic assertion in the form of an expression tree

**Direct Known Superclasses (Generalization)**

[RuleStatement](#_f8740e8d27529166da46265bd8521c94)

**Attributes**

• public tag : String [0..1]

Expression tag, used for differentiating multiple assertions.

**Associations**

• public expression : [ExprItem](#_6584fad2aa663f8951c117400955df67) [1]

Abstract parent of all expression items

### <Class> BuiltinVariable

**Description**

A variable with a name and definition from a small set of assumed environmental variables

**Direct Known Superclasses (Generalization)**

[VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015)

### <Class> ExprArchetypeIdConstraint

**Description**

Expression tree leaf item representing a constraint on an archetype identifier

**Direct Known Superclasses (Generalization)**

[ExprConstraint](#_85bd43994bf2970e6d28314019bb2090)

**Attributes**

• public item : [StringConstraint](#_e78c0feb207cbea2ca9911ec94e2a83e) [1]

The constraint on archetype identifiers

### <Class> ExprbinaryOperator

**Description**

Binary operator expression node

**Direct Known Superclasses (Generalization)**

[ExprOperator](#_4f284e94fe833667061a7b6b72ef6466)

**Associations**

• public left\_operand : [ExprItem](#_6584fad2aa663f8951c117400955df67) [1]

Abstract parent of all expression items

• public right\_operand : [ExprItem](#_6584fad2aa663f8951c117400955df67) [1]

Abstract parent of all expression items

### <Class> ExprConstant

**Description**

Constant expression tree leaf item

**Direct Known Superclasses (Generalization)**

[ExprLeaf](#_9560e7eb734489fe38110da26dd95cc6)

**Attributes**

• public value : [AnyPrimitiveType](#_72b7d3db018aa6b2ac9fd5dcdf816a12) [1]

The constant value

### <Class> ExprConstraint

**Description**

Expression tree leaf item representing a constraint on a primitive type

**Direct Known Superclasses (Generalization)**

[ExprLeaf](#_9560e7eb734489fe38110da26dd95cc6)

**Direct Known Subclasses (Specialization)**

[ExprArchetypeIdConstraint](#_820ea321dcb1dd82f36cbb9c8fb64676)

**Attributes**

• public item : [PrimitiveObjectConstraint](#_db9df3b10e304d809393da4afc9a91da) [1]

A constraint on a primitive type

### <Class> ExprItem

**Description**

Abstract parent of all expression items

**Direct Known Superclasses (Generalization)**

[RuleElement](#_aca03c36f52ec3dbae79ae94604d6df9)

**Direct Known Subclasses (Specialization)**

[ExprLeaf](#_9560e7eb734489fe38110da26dd95cc6), [ExprOperator](#_4f284e94fe833667061a7b6b72ef6466)

**Attributes**

• public type : String [1]

Logical type of this item

**Operations**

• public value ()

### <Class> ExprLeaf

**Description**

Non-compositional item representing a manifest constant of any primitive type; a path referring to a value in the archetype; a constraint; or a variable reference

**Direct Known Superclasses (Generalization)**

[ExprItem](#_6584fad2aa663f8951c117400955df67)

**Direct Known Subclasses (Specialization)**

[ExprConstant](#_a0128bb6e758476a358e05c37712848d), [ExprConstraint](#_85bd43994bf2970e6d28314019bb2090), [ExprModelRef](#_7d47fc8980dcc79b04b0f75cd8e3f12e), [ExprVariableRef](#_9129057e4e52fc3f3b74690cdfbfbd78)

**Operations**

• public referenceType () : [ReferenceType](#_428e9af279e2df7756c01dffd2ccc1d4)

The way the leaf item value is defined

### <Class> ExprModelRef

**Description**

Expression tree leaf item representing a reference to a value found at a location specified by a path in the archetype definition

**Direct Known Superclasses (Generalization)**

[ExprLeaf](#_9560e7eb734489fe38110da26dd95cc6)

**Attributes**

• public path : String [1]

The path

### <Class> ExprOperator

**Description**

Abstract parent of operator types

**Direct Known Superclasses (Generalization)**

[ExprItem](#_6584fad2aa663f8951c117400955df67)

**Direct Known Subclasses (Specialization)**

[ExprbinaryOperator](#_35c04239c15182fcb41266740097f888), [ExprUnaryOperator](#_2ea545d67d0574bfd64976dd80367d68)

**Attributes**

• public operator : [OperatorKind](#_ecd27b09f62f1796f70d44a561b90e90) [1]

Operator kind

• public precedenceOverridden [1]

True if the natural precedence of operators is

overridden in the expression represented by

this node of the expression tree. If True,

parentheses should be introduced around the

totality of the syntax expression corresponding

to this operator node and its operands.

### <Class> ExprUnaryOperator

**Description**

Unary operator expression node

**Direct Known Superclasses (Generalization)**

[ExprOperator](#_4f284e94fe833667061a7b6b72ef6466)

**Associations**

• public operand : [ExprItem](#_6584fad2aa663f8951c117400955df67) [1]

Abstract parent of all expression items

### <Class> ExprVariable

**Description**

A variable whose definition is an expression

**Direct Known Superclasses (Generalization)**

[VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015)

**Associations**

• public expression : [ExprItem](#_6584fad2aa663f8951c117400955df67) [1]

Abstract parent of all expression items

### <Class> ExprVariableRef

**Description**

**Direct Known Superclasses (Generalization)**

[ExprLeaf](#_9560e7eb734489fe38110da26dd95cc6)

**Associations**

• public declaration : [VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015) [0..1]

Definition of a named variable used in an assertion expression

### <Class> QueryVariable

**Description**

A variable whose value is derived from a query run on a data context in the operational environment

**Direct Known Superclasses (Generalization)**

[VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015)

**Attributes**

• public context : String [1]

Name of context

• public queryId : String [1]

Identifier of query in the external context, e.g. �date\_of\_birth�

• public queryArgs : String [0..\*]

Arguments for query

### <Class> RuleElement

**Description**

**Direct Known Subclasses (Specialization)**

[ExprItem](#_6584fad2aa663f8951c117400955df67), [RuleStatement](#_f8740e8d27529166da46265bd8521c94)

**Operations**

• public asString () : [String](#_e8a6ce315d976318da3ab784a645ea44)

A rule element in serialized form

### <Class> RuleStatement

**Description**

Abstract parent of all statement types

**Direct Known Superclasses (Generalization)**

[RuleElement](#_aca03c36f52ec3dbae79ae94604d6df9)

**Direct Known Subclasses (Specialization)**

[Assertion](#_f9e7c553caf3e674732fe386e3d45466), [VariableDeclaration](#_8a634b04f92ff4c449cdcaaae16ba015)

### <Class> VariableDeclaration

**Description**

Definition of a named variable used in an assertion expression

**Direct Known Superclasses (Generalization)**

[RuleStatement](#_f8740e8d27529166da46265bd8521c94)

**Direct Known Subclasses (Specialization)**

[BuiltinVariable](#_f8424186be59dfa59367f207253d80c9), [ExprVariable](#_9a7170f73e8949fcd1203662c43a988f), [QueryVariable](#_19604fadcc5010ea7dfd6960b3ed9095)

**Attributes**

• private name : String [1]

Name of variable

• private type : String [1]

Variable type, drawn from the reference model

**Operations**

• public definition () : [String](#_e8a6ce315d976318da3ab784a645ea44)

Formal definition of the variable

• public value ()

### <Enumeration> OperatorKind

**Description**

**Enumeration Literals**

* **AND**

* **DIVIDE**

* **EQ**

* **EXISTS**

* **EXP**

* **FORALL**

* **GE**

* **GT**

* **IMPLIES**

* **LE**

* **LT**

* **MATCHES**

* **MINUS**

* **MULTIPLY**

* **NE**

* **NOT**

* **OR**

* **PLUS**

* **XOR**

### <Enumeration> ReferenceType

**Description**

**Enumeration Literals**

* **ATTRIBUTE**

* **CONSTANT**

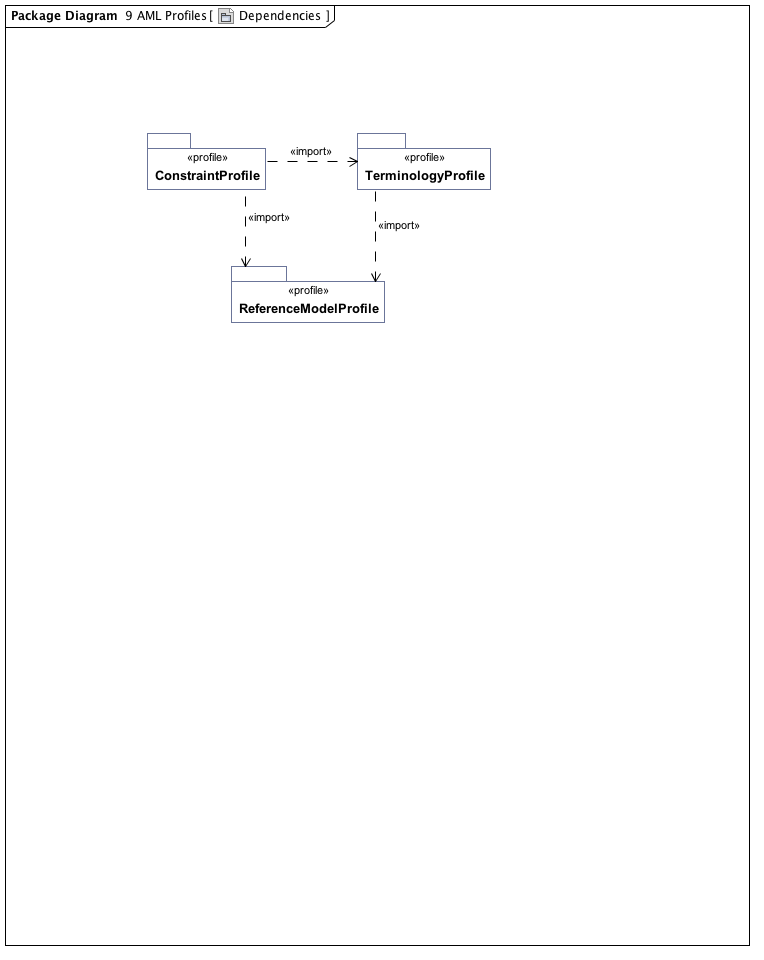
* **CONSTRAINT**

* **FUNCTION**

## <Package> Metadata Object Model

# AML Profiles

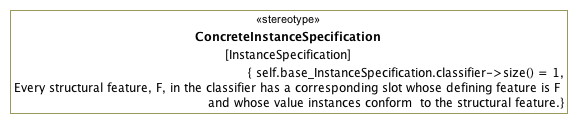
Introduction to the AML Profiles section



**Dependencies**

## <Package> Reference Model Profile

Introduction to the reference model profile section



**9.1.1 ConcreteInstance Specification**

### <Stereotype> ConcreteInstanceSpecification

**Description**

The ConcreteInstanceSpecification stereotype applies to an InstanceSpecification that represents an instance of a class or specialization thereof. Its purpose is to assure that the base InstanceSpecification has exactly one classifier and that the specification is a fully conformant instance of that classifier.

**Direct Known Subclasses (Specialization)**

[AMLTypeInstance](#_37ca1d66c752ca70d71c80306e2dfd12), [ArchetypeIdInstance](#_192fd239c67a67acded20aaf25afd72a), [ArchetypeVersionIdInstance](#_8b61c03e7d48b3e4fee906c2f8b71f55), [ResourceDescriptionInstance](#_ea4e6c2ebfb918ac64176a15c3db0e26), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d), [ResourceTranslationInstance](#_735e416f245744e0fc443c2830c8a185)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• oneClassifier

The base InstanceSpecification has exactly one classifier.

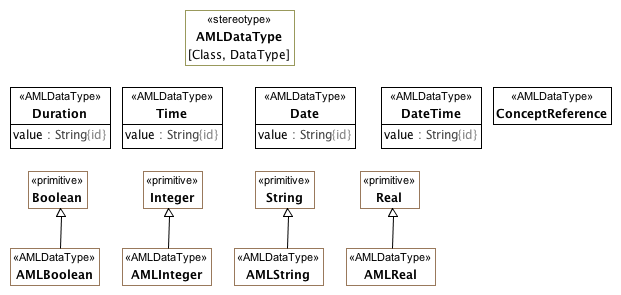
[OCL2.0]

self.base\_InstanceSpecification.classifier->size() = 1

• slotConformance

[English]

Every structural feature, F, in the classifier has a corresponding slot whose defining feature is F and whose value instances conform to the structural feature.



**9.1.2 Implementable Data Types**

The AMLDataType stereotype serves two purposes:

1. To identify the set of atomic types whose possible values can be constrained in the AML Constraint Profile
2. To identify the set of types whose value will be treated as "data types" from the AML perspective.

The AMLDataType stereotype can extend both Class and DataType elements. The target reference model may choose to represent some or all of the AML DataTypes in a different fashion. A reference model may define its own String DataType rather than using the UML Native String type directly. Similarly, it may choose to represent a Date as a complex object consisting of year, month, day, granularity, zone, etc.

One of the tasks for a reference model implementor is to create maps from the appropriate AML Data Types and the corresponding reference model types.

### <Class> ConceptReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

**Attributes**

• public describingCodeSystem : [CodeSystemReference](#_3995e3056e4230c2488360f87858c821) [0..1]

### <Class> Date

**Attributes**

• public value : String

### <Class> DateTime

**Attributes**

• public value : String

### <Class> Duration

**Attributes**

• public value : String

### <Class> Time

**Attributes**

• public value : String

### <Stereotype> AMLDataType

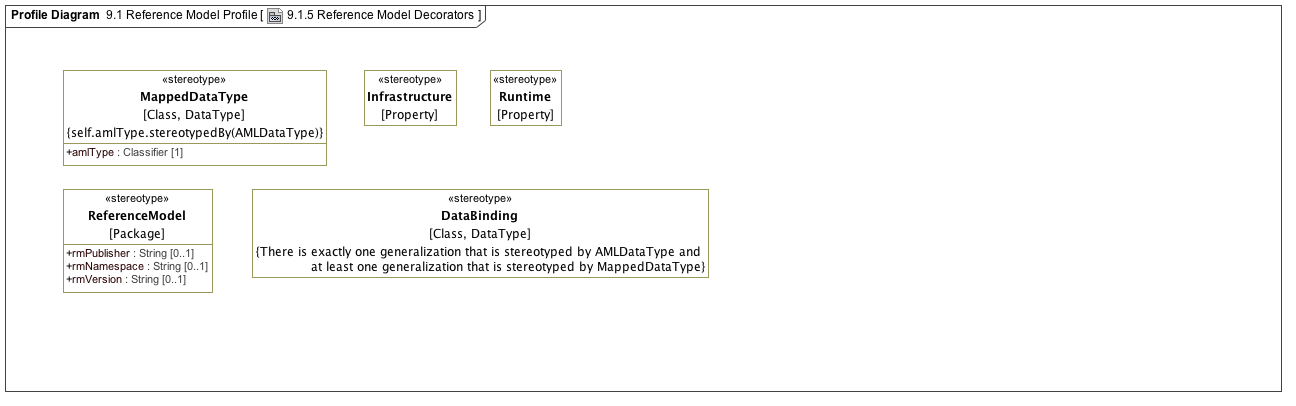
**Description**

The AMLDataType stereotype represents a built-in AML data type. Instances of AMLDataType primitive types have the same identity semantics as the UML PrimitiveType. Instances AMLDataType classes are with the identical "id" properties (property.isID = True} are considered to be identical

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

• private base\_DataType : [DataType](#_687a198ec4e5e49dcd9a605729dc1c24)



**9.1.5 Reference Model Decorators**

### <Stereotype> DataBinding

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

• private base\_DataType : [DataType](#_687a198ec4e5e49dcd9a605729dc1c24)

**Constraints**

• oneAmlProperty

[English]

There is exactly one generalization that is stereotyped by AMLDataType and at least one generalization that is stereotyped by MappedDataType

### <Stereotype> Infrastructure

**Associations**

• private base\_Property : [Property](#_9624701f3c3fcd329e6dba1fc74db836)

### <Stereotype> MappedDataType

**Attributes**

• public amlType : Classifier [1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

• private base\_DataType : [DataType](#_687a198ec4e5e49dcd9a605729dc1c24)

**Constraints**

• isAMLDataType

The amlType must reference a classifier (Class or DataType) that has a AMLDataType stereotype.

[OCL2.0]

self.amlType.stereotypedBy(AMLDataType)

### <Stereotype> ReferenceModel

**Description**

This stereotype identifies a package as a "reference model". An archetype library contains a collection of archetypes that constrain classes that are members of a target reference model. The stereotype also allows additional metadata such as the publisher, namespace, etc. to be added to the model itself.

**Attributes**

• public rmPublisher : String [0..1]

The name of the Reference Model publisher.

• public rmNamespace : String [0..1]

The owning domain name of the archetype. Corresponds to the **namespace** attribute in AOM1.5.

• public rmVersion : String [0..1]

The reference model version identifier.

**Associations**

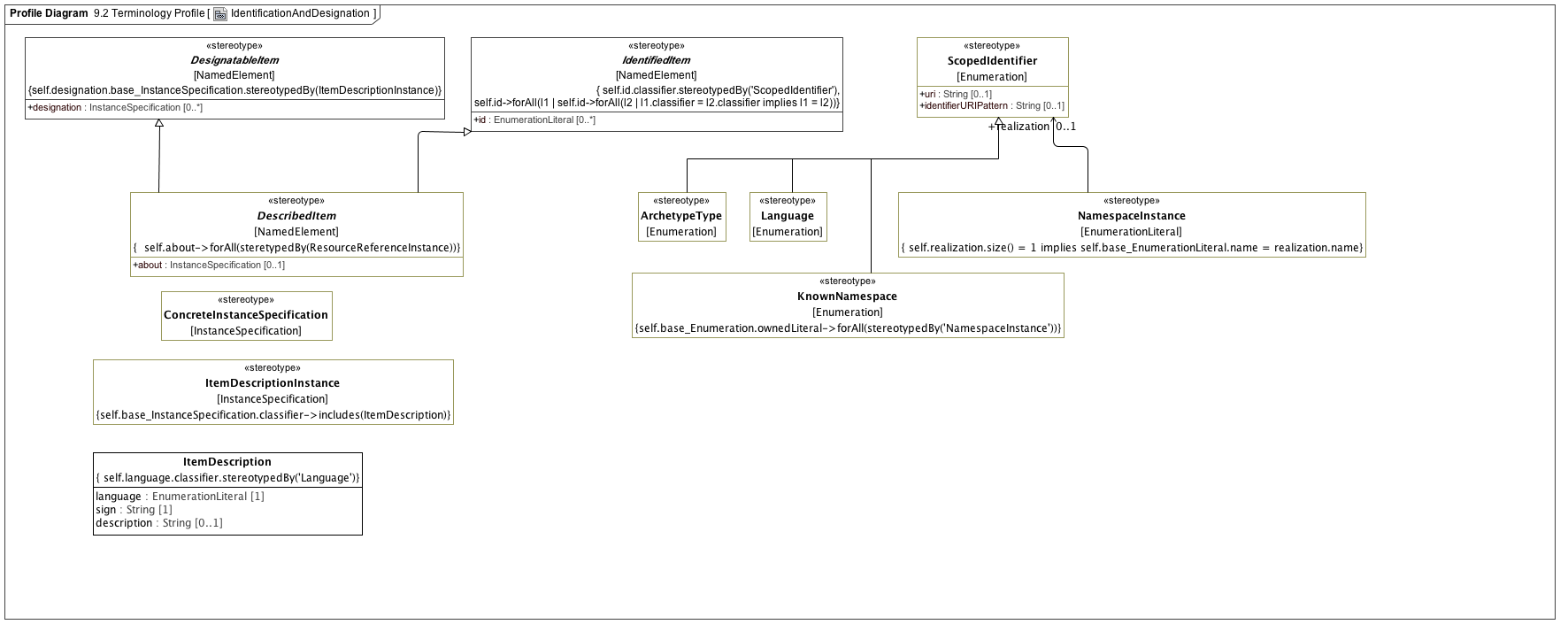
• private base\_Package : [Package](#_657662b69a91b4b82844aca2e3465cd3)

### <Stereotype> Runtime

**Associations**

• private base\_Property : [Property](#_9624701f3c3fcd329e6dba1fc74db836)

## <Package> Terminology Profile



**IdentificationAndDesignation**

### <Class> ItemDescription

**Attributes**

• public language : EnumerationLiteral [1]

• public sign : String [1]

• public description : String [0..1]

**Constraints**

• languageLiteral

[OCL2.0]

self.language.classifier.stereotypedBy('Language')

### <Stereotype> ArchetypeType

**Direct Known Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

### <Stereotype> ConcreteInstanceSpecification

**Description**

The ConcreteInstanceSpecification stereotype applies to an InstanceSpecification that represents an instance of a class or specialization thereof. Its purpose is to assure that the base InstanceSpecification has exactly one classifier and that the specification is a fully conformant instance of that classifier.

**Direct Known Subclasses (Specialization)**

[AMLTypeInstance](#_37ca1d66c752ca70d71c80306e2dfd12), [ArchetypeIdInstance](#_192fd239c67a67acded20aaf25afd72a), [ArchetypeVersionIdInstance](#_8b61c03e7d48b3e4fee906c2f8b71f55), [ResourceDescriptionInstance](#_ea4e6c2ebfb918ac64176a15c3db0e26), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d), [ResourceTranslationInstance](#_735e416f245744e0fc443c2830c8a185)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• oneClassifier

The base InstanceSpecification has exactly one classifier.

[OCL2.0]

self.base\_InstanceSpecification.classifier->size() = 1

• slotConformance

[English]

Every structural feature, F, in the classifier has a corresponding slot whose defining feature is F and whose value instances conform to the structural feature.

### <Stereotype> DescribedItem

**Direct Known Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77), [IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a), [ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2), [PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692)

**Attributes**

• public about : InstanceSpecification [0..1]

**Associations**

• private base\_NamedElement : [NamedElement](#_0907197416161d85b05eb0e9c47d90e9)

**Constraints**

• d

[OCL2.0]

self.about->forAll(steretypedBy(ResourceReferenceInstance))

### <Stereotype> DesignatableItem

**Direct Known Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

• public designation : InstanceSpecification [0..\*]

**Associations**

• private base\_NamedElement : [NamedElement](#_0907197416161d85b05eb0e9c47d90e9)

**Constraints**

• designationTypeItemDescription

Every designation instance is a valid instance of ItemDesignation

[OCL2.0]

self.designation.base\_InstanceSpecification.stereotypedBy(ItemDescriptionInstance)

### <Stereotype> IdentifiedItem

**Description**

Every identifier must come from a different namespace

**Direct Known Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

• public id : EnumerationLiteral [0..\*]

**Associations**

• private base\_NamedElement : [NamedElement](#_0907197416161d85b05eb0e9c47d90e9)

**Constraints**

• scopedIdentifierLiteral

Every *id* property is an instance of an ScopedIdentifier.

[OCL2.0]

self.id.classifier.stereotypedBy('ScopedIdentifier')

• uniqueScopes

Every id must belong to a unique instance specification classifier. An identified Item cannot have two or more identifiers drawn from the same ScopedIdentifier enumeration.

[OCL2.0]

self.id->forAll(l1 | self.id->forAll(l2 | l1.classifier = l2.classifier implies l1 = l2))

### <Stereotype> ItemDescriptionInstance

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• classifier

[OCL2.0]

self.base\_InstanceSpecification.classifier->includes(ItemDescription)

### <Stereotype> KnownNamespace

**Description**

KnownNamespace represents the set of NameSpaceInstances that are known to an AML implementation. Each NamespaceInstance uniquely names a specific namespace and an optional ScopedIdentifier that is the actual instances of that namespace.

**Direct Known Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• namespaceInstances

[OCL2.0]

self.base\_Enumeration.ownedLiteral->forAll(stereotypedBy('NamespaceInstance'))

### <Stereotype> Language

**Direct Known Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

### <Stereotype> NamespaceInstance

**Description**

**Associations**

• private base\_EnumerationLiteral : [EnumerationLiteral](#_cba107636e03e0cb80fbb911dd86b703)

• public realization : [ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b) [0..1]

**Constraints**

• realizationId

[OCL2.0]

self.realization.size() = 1 implies self.base\_EnumerationLiteral.name = realization.name

### <Stereotype> ScopedIdentifier

**Direct Known Subclasses (Specialization)**

[ArchetypeType](#_7dc1530ae1ef855ecc3eb9bd5b555a14), [KnownNamespace](#_b9f78b93edc24bb3301ba69a57e4afc3), [Language](#_446e0591a4f825b22cd9e573c1239a72)

**Attributes**

• public uri : String [0..1]

• public identifierURIPattern : String [0..1]

**Associations**

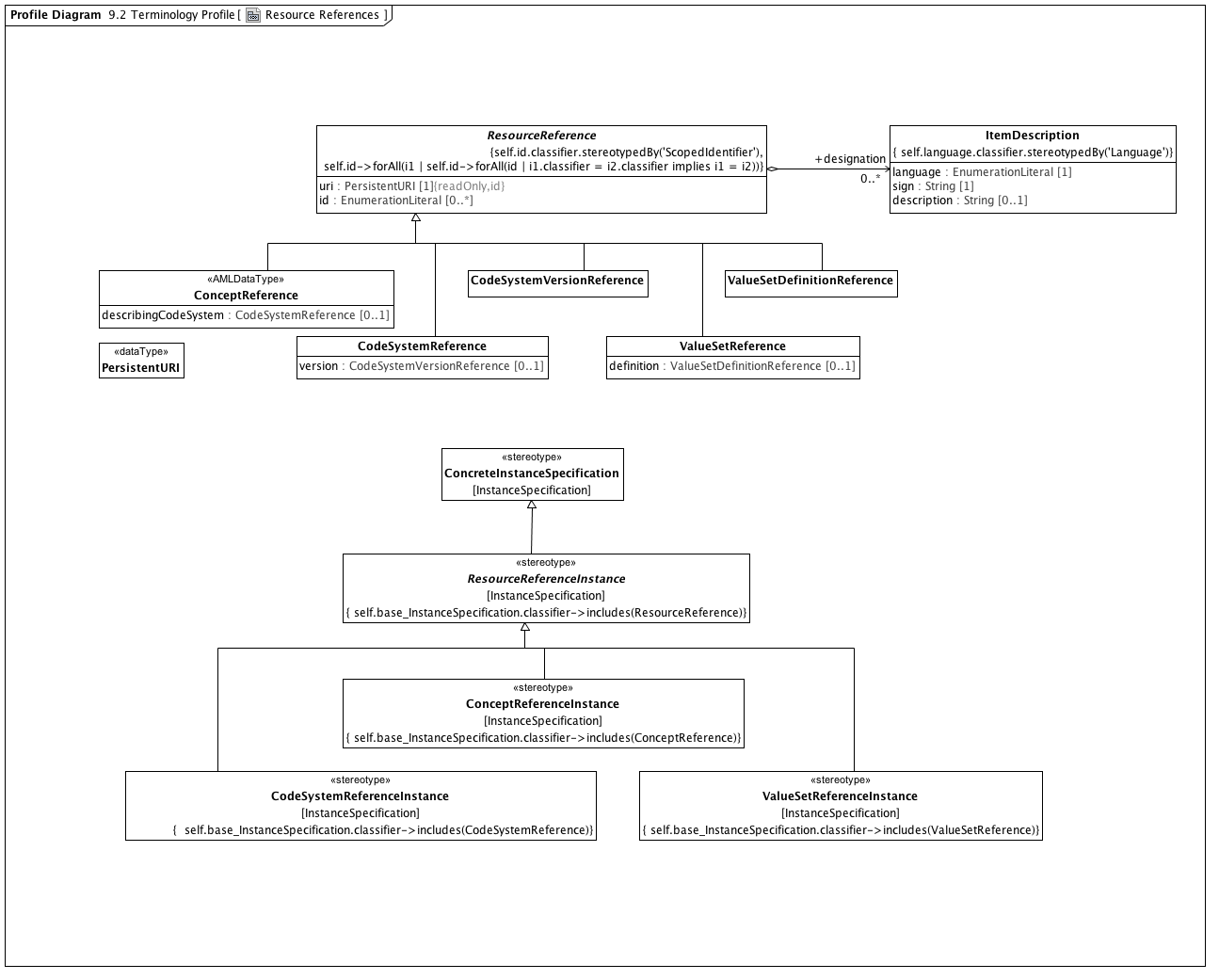
• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• hasScopedIdentifiers

[OCL2.0]

self.base\_Enumeration.ownedLiteral->forAll(stereotypedBy('ScopedIdentifier'))



**Resource References**

### <DataType> PersistentURI

### <Class> CodeSystemReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

**Attributes**

• public version : [CodeSystemVersionReference](#_488758a8e8b842f340c50a3513cf51a1) [0..1]

### <Class> CodeSystemVersionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

### <Class> ConceptReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

**Attributes**

• public describingCodeSystem : [CodeSystemReference](#_3995e3056e4230c2488360f87858c821) [0..1]

### <Class> ItemDescription

**Attributes**

• public language : EnumerationLiteral [1]

• public sign : String [1]

• public description : String [0..1]

**Constraints**

• languageLiteral

[OCL2.0]

self.language.classifier.stereotypedBy('Language')

### <Class> ResourceReference

**Direct Known Subclasses (Specialization)**

[CodeSystemReference](#_3995e3056e4230c2488360f87858c821), [CodeSystemVersionReference](#_488758a8e8b842f340c50a3513cf51a1), [ConceptReference](#_ae987614d7a9f06b4da55c1b58dd49d6), [ValueSetDefinitionReference](#_181ae8108ce58cfc25ae10446c2b674f), [ValueSetReference](#_dc7692ae0c5a409ef6e5c8c178fef068)

**Attributes**

• public uri : [PersistentURI](#_f28ad8d486d46fbf4069733a66771d1f) [1]

• public id : EnumerationLiteral [0..\*]

**Associations**

• public designation : [ItemDescription](#_71d80a66e5586709839872d696b05a07) [0..\*]

**Constraints**

• scopedIdentifier

[OCL2.0]

self.id.classifier.stereotypedBy('ScopedIdentifier')

• uniqueId

[OCL2.0]

self.id->forAll(i1 | self.id->forAll(id | i1.classifier = i2.classifier implies i1 = i2))

### <Class> ValueSetDefinitionReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

### <Class> ValueSetReference

**Direct Known Superclasses (Generalization)**

[ResourceReference](#_3011e434b91fcdb8310b1acf9765e89c)

**Attributes**

• public definition : [ValueSetDefinitionReference](#_181ae8108ce58cfc25ae10446c2b674f) [0..1]

### <Stereotype> CodeSystemReferenceInstance

**Direct Known Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isConceptReference

• isCodeSystemReference

[OCL2.0]

self.base\_InstanceSpecification.classifier->includes(CodeSystemReference)

### <Stereotype> ConceptReferenceInstance

**Direct Known Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isConceptReference

[OCL2.0]

self.base\_InstanceSpecification.classifier->includes(ConceptReference)

### <Stereotype> ConcreteInstanceSpecification

**Description**

The ConcreteInstanceSpecification stereotype applies to an InstanceSpecification that represents an instance of a class or specialization thereof. Its purpose is to assure that the base InstanceSpecification has exactly one classifier and that the specification is a fully conformant instance of that classifier.

**Direct Known Subclasses (Specialization)**

[AMLTypeInstance](#_37ca1d66c752ca70d71c80306e2dfd12), [ArchetypeIdInstance](#_192fd239c67a67acded20aaf25afd72a), [ArchetypeVersionIdInstance](#_8b61c03e7d48b3e4fee906c2f8b71f55), [ResourceDescriptionInstance](#_ea4e6c2ebfb918ac64176a15c3db0e26), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d), [ResourceTranslationInstance](#_735e416f245744e0fc443c2830c8a185)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• oneClassifier

The base InstanceSpecification has exactly one classifier.

[OCL2.0]

self.base\_InstanceSpecification.classifier->size() = 1

• slotConformance

[English]

Every structural feature, F, in the classifier has a corresponding slot whose defining feature is F and whose value instances conform to the structural feature.

### <Stereotype> ResourceReferenceInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23), [DesignatableItem](#_80448b03d480bba05b1e156796878f77), [IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Known Subclasses (Specialization)**

[CodeSystemReferenceInstance](#_e055a6cce06d0838055b62dbfbf235f2), [ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1), [ValueSetReferenceInstance](#_f3184cb0f8e704f5122c5e97fb9f130c)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isResourceReference

[OCL2.0]

self.base\_InstanceSpecification.classifier->includes(ResourceReference)

### <Stereotype> ValueSetReferenceInstance

**Direct Known Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Associations**

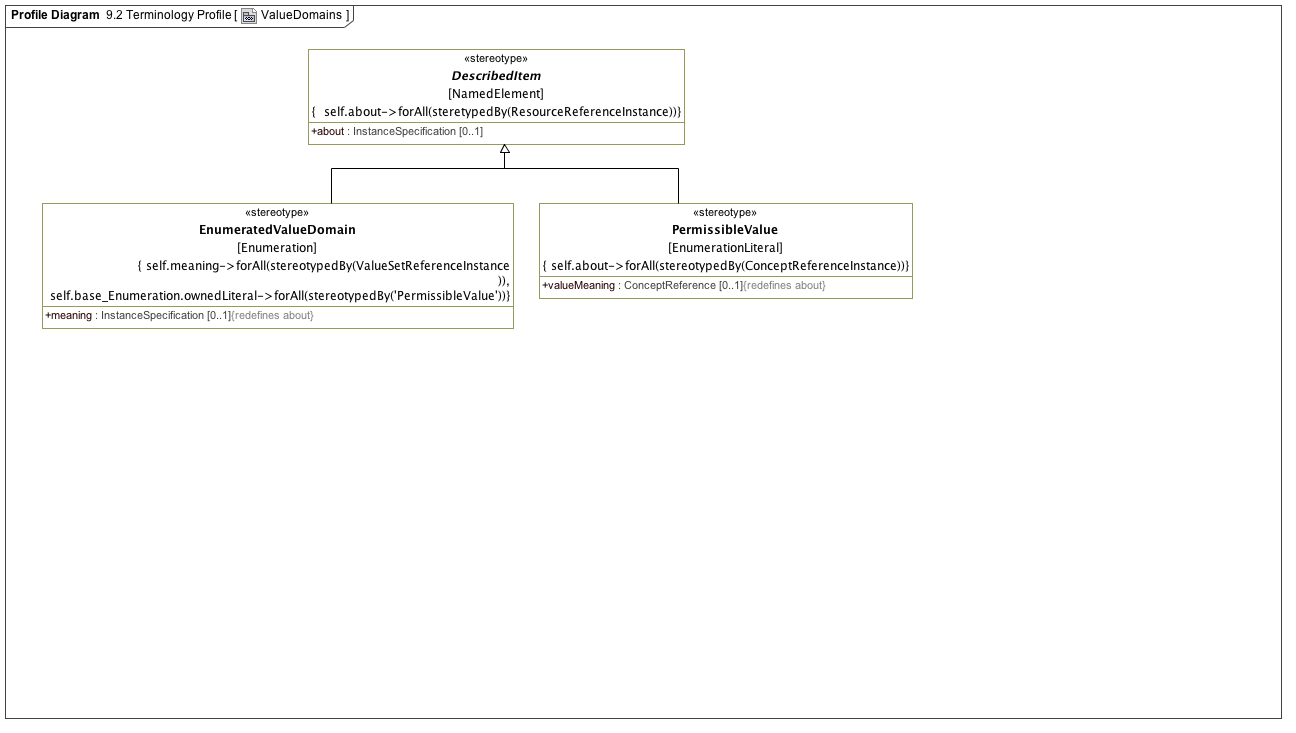
• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isValueSetReference

[OCL2.0]

self.base\_InstanceSpecification.classifier->includes(ValueSetReference)



**ValueDomains**

### <Stereotype> DescribedItem

**Direct Known Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77), [IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a), [ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2), [PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692)

**Attributes**

• public about : InstanceSpecification [0..1]

**Associations**

• private base\_NamedElement : [NamedElement](#_0907197416161d85b05eb0e9c47d90e9)

**Constraints**

• d

[OCL2.0]

self.about->forAll(steretypedBy(ResourceReferenceInstance))

### <Stereotype> EnumeratedValueDomain

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Attributes**

• public meaning : InstanceSpecification [0..1]

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• permissibleValues

[OCL2.0]

self.base\_Enumeration.ownedLiteral->forAll(stereotypedBy('PermissibleValue'))

• meaningIsConceptReference

[OCL2.0]

self.meaning->forAll(stereotypedBy(ValueSetReferenceInstance ))

### <Stereotype> PermissibleValue

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Attributes**

• public valueMeaning : [ConceptReference](#_ae987614d7a9f06b4da55c1b58dd49d6) [0..1]

**Associations**

• private base\_EnumerationLiteral : [EnumerationLiteral](#_cba107636e03e0cb80fbb911dd86b703)

**Constraints**

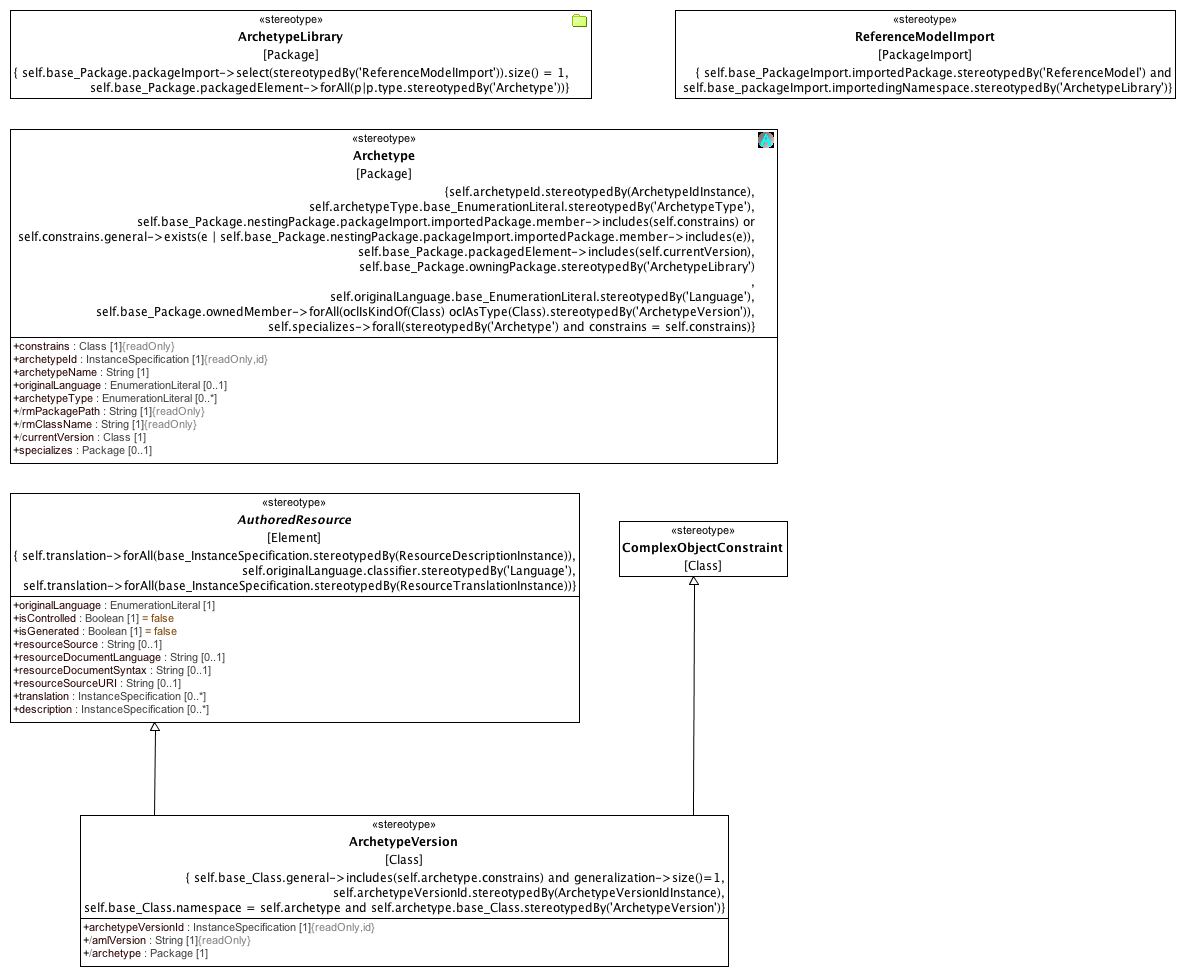
• valueMeaningIsConceptReference

[OCL2.0]

self.about->forAll(stereotypedBy(ConceptReferenceInstance))

## <Package> Constraint Profile

### <Package> Archetypes



**9.3.1.1 Archetypes**

### <Stereotype> Archetype

**Description**

An Archetype is a package that contains a versioned set of constraints on a Class that is a member of the Reference Model that is owned by the containing Archetype Library.

**Attributes**

• public constrains : Class [1]

• public archetypeId : InstanceSpecification [1]

• public archetypeName : String [1]

• public originalLanguage : EnumerationLiteral [0..1]

• public archetypeType : EnumerationLiteral [0..\*]

• public rmPackagePath : String [1]

• public rmClassName : String [1]

• public currentVersion : Class [1]

• public specializes : Package [0..1]

**Associations**

• private base\_Package : [Package](#_657662b69a91b4b82844aca2e3465cd3)

**Constraints**

• mustBeOwned

[OCL2.0]

self.base\_Package.owningPackage.stereotypedBy('ArchetypeLibrary')

• ownsVersions

[OCL2.0]

self.base\_Package.ownedMember->forAll(oclIsKindOf(Class) oclAsType(Class).stereotypedBy('ArchetypeVersion'))

• originalLanguage

[OCL2.0]

self.originalLanguage.base\_EnumerationLiteral.stereotypedBy('Language')

• archetypeType

[OCL2.0]

self.archetypeType.base\_EnumerationLiteral.stereotypedBy('ArchetypeType')

• constrainsRMElement

The Class or one of its generalization ancestors is a member of the ArchetypeLibrary Reference Model

[OCL2.0]

self.base\_Package.nestingPackage.packageImport.importedPackage.member->includes(self.constrains) or self.constrains.general->exists(e | self.base\_Package.nestingPackage.packageImport.importedPackage.member->includes(e))

• currentVersion

[OCL2.0]

self.base\_Package.packagedElement->includes(self.currentVersion)

• specializesArchetype

If an Archetype specializes another Archetype, they both constrain the same class.

[OCL2.0]

self.specializes->forall(stereotypedBy('Archetype') and constrains = self.constrains)

• archetypeIdType

[OCL2.0]

self.archetypeId.stereotypedBy(ArchetypeIdInstance)

### <Stereotype> ArchetypeLibrary

**Description**

An ArchetypeLibrary is a Package that contains a collection of archetypes that constrain classes within the same Reference Model. An ArchetypeLibrary must import exactly one reference model.

**Associations**

• private base\_Package : [Package](#_657662b69a91b4b82844aca2e3465cd3)

**Constraints**

• oneReferenceModel

[OCL2.0]

self.base\_Package.packageImport->select(stereotypedBy('ReferenceModelImport')).size() = 1

• onlyArchetypes

[OCL2.0]

self.base\_Package.packagedElement->forAll(p|p.type.stereotypedBy('Archetype'))

### <Stereotype> ArchetypeVersion

**Direct Known Superclasses (Generalization)**

[AuthoredResource](#_13ad6987e15b787d385f0b30ff25d6c9), [ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62)

**Attributes**

• public archetypeVersionId : InstanceSpecification [1]

• public amlVersion : String [1]

• public archetype : Package [1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• archetypeRoot

[OCL2.0]

self.base\_Class.general->includes(self.archetype.constrains) and generalization->size()=1

• ownedByArchetype

[OCL2.0]

self.base\_Class.namespace = self.archetype and self.archetype.base\_Class.stereotypedBy('ArchetypeVersion')

• archetypeVersionIdType

[Binary]

self.archetypeVersionId.stereotypedBy(ArchetypeVersionIdInstance)

### <Stereotype> AuthoredResource

**Direct Known Subclasses (Specialization)**

[ArchetypeVersion](#_61831f1c446a753e3069251f603bfa37)

**Attributes**

• public originalLanguage : EnumerationLiteral [1]

• public isControlled : Boolean [1] = false

• public isGenerated : Boolean [1] = false

• public resourceSource : String [0..1]

• public resourceDocumentLanguage : String [0..1]

• public resourceDocumentSyntax : String [0..1]

• public resourceSourceURI : String [0..1]

• public translation : InstanceSpecification [0..\*]

• public description : InstanceSpecification [0..\*]

**Associations**

• private base\_Element : [Element](#_5d5cfd5b8453e2037ccdadc05f608c6d)

**Constraints**

• languageEnumeration

[OCL2.0]

self.originalLanguage.classifier.stereotypedBy('Language')

• translation

[OCL2.0]

self.translation->forAll(base\_InstanceSpecification.stereotypedBy(ResourceTranslationInstance))

• description

[OCL2.0]

self.translation->forAll(base\_InstanceSpecification.stereotypedBy(ResourceDescriptionInstance))

### <Stereotype> ComplexObjectConstraint

**Description**

A ComplexObjectConstraint is a specialization of a single UML Class. A ComplexObjectConstraint may restrict the cardinality or possible values for one or more of the general class properties.

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[ArchetypeVersion](#_61831f1c446a753e3069251f603bfa37)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• singleParent

Every constraint must specialize exactly one Class

[OCL2.0]

generalization->size() = 1

• noNewAttributes

A ComplexObjectConstraint cannot introduce a new property, only subset or redefine existing ones.

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(redefinedProperty.size() > 0 or subsettedProperty.size() > 0)

• redefinesOwnedAttribute

Every owned attribute in the base class is also an attribute of type AttributeConstraint and every AttributeConstraint is also an owned attribute in the base class.

[OCL2.0]

self.attribute->forAll(a | self.base\_Class.ownedAttribute->includes(a.base\_Property)) and self.base\_Class.ownedAttribute->forAll(o | self.attribute->exists(a | a.base\_Property = o))

• unnamed1

[]

• unnamed2

[]

• allAttributeConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(stereotypedBy(AttributeConstraint)

• singularConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(upper = 1 implies stereotypedBy(SingularAttributeConstraint) and redefinedProperty.size() = 1)

•

[]

• attributeCollectionConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(upper > 1 implies stereotypedBy(AttributeCollectionConstraint) and subsettedProperty.size() > 0)

### <Stereotype> ReferenceModelImport

**Description**

ReferenceModelImport is a PackageImport where the importingNamespace is an instance of an ArchetypeLibrary and the importedPackage is an instance of a ReferenceModel.

**Associations**

• private base\_PackageImport : [PackageImport](#_1d306d6724cf35c53c9b6fb549a90ea4)

**Constraints**

• libraryOnly

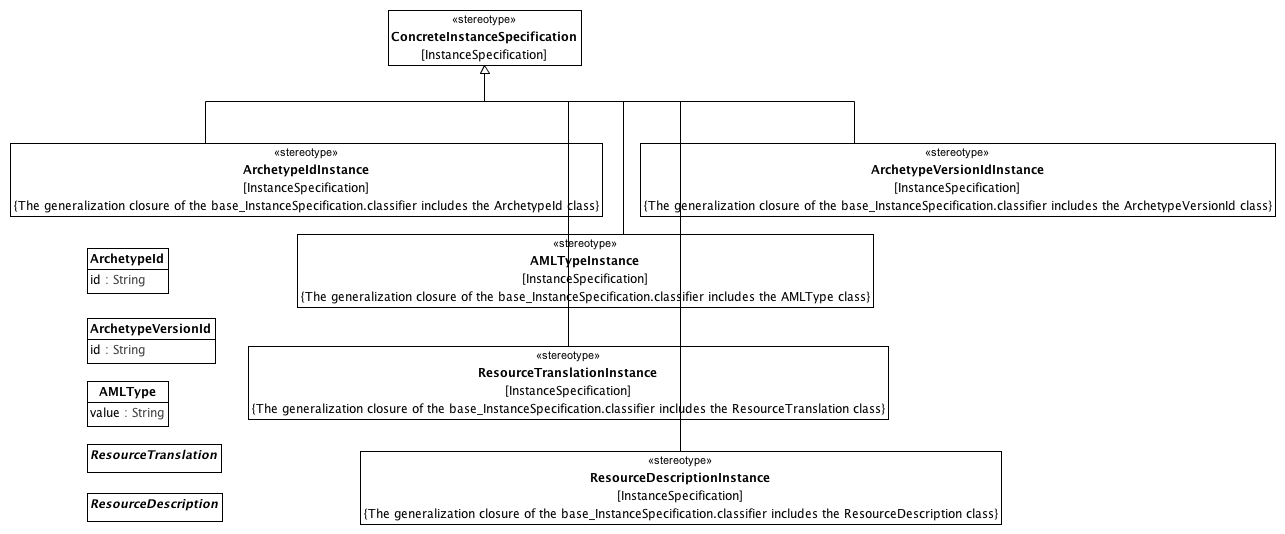
[OCL2.0]

self.base\_PackageImport.importingNamespace.stereotypedBy('ArchetypeLibrary')

• libraryReferenceModel

[OCL2.0]

self.base\_PackageImport.importedPackage.stereotypedBy('ReferenceModel') and self.base\_packageImport.importedingNamespace.stereotypedBy('ArchetypeLibrary')



**9.3.1.2 Archetype Metadata**

### <Class> AMLType

**Description**

An AMLType represents a "primitive" or "atomic" type in the AML specification. An instance of an AMLType is identified solely by its value. AMLType is not a specialization of the UML DataType because it may need to be represented as a specialization of UML Class in a Reference Model binding.

**Attributes**

• public value : String

### <Class> ArchetypeId

**Attributes**

• public id : String

### <Class> ArchetypeVersionId

**Attributes**

• public id : String

### <Class> ResourceDescription

### <Class> ResourceTranslation

### <Stereotype> AMLTypeInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isAMLType

[English]

The generalization closure of the base\_InstanceSpecification.classifier includes the AMLType class

### <Stereotype> ArchetypeIdInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isArchetypeId

[English]

The generalization closure of the base\_InstanceSpecification.classifier includes the ArchetypeId class

### <Stereotype> ArchetypeVersionIdInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isArchetypeVersionId

[English]

The generalization closure of the base\_InstanceSpecification.classifier includes the ArchetypeVersionId class

### <Stereotype> ConcreteInstanceSpecification

**Description**

The ConcreteInstanceSpecification stereotype applies to an InstanceSpecification that represents an instance of a class or specialization thereof. Its purpose is to assure that the base InstanceSpecification has exactly one classifier and that the specification is a fully conformant instance of that classifier.

**Direct Known Subclasses (Specialization)**

[AMLTypeInstance](#_37ca1d66c752ca70d71c80306e2dfd12), [ArchetypeIdInstance](#_192fd239c67a67acded20aaf25afd72a), [ArchetypeVersionIdInstance](#_8b61c03e7d48b3e4fee906c2f8b71f55), [ResourceDescriptionInstance](#_ea4e6c2ebfb918ac64176a15c3db0e26), [ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d), [ResourceTranslationInstance](#_735e416f245744e0fc443c2830c8a185)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• oneClassifier

The base InstanceSpecification has exactly one classifier.

[OCL2.0]

self.base\_InstanceSpecification.classifier->size() = 1

• slotConformance

[English]

Every structural feature, F, in the classifier has a corresponding slot whose defining feature is F and whose value instances conform to the structural feature.

### <Stereotype> ResourceDescriptionInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

**Constraints**

• isResourceDescription

[English]

The generalization closure of the base\_InstanceSpecification.classifier includes the ResourceDescription class

### <Stereotype> ResourceTranslationInstance

**Direct Known Superclasses (Generalization)**

[ConcreteInstanceSpecification](#_27a8967f003efe37629fea8e80ca6c23)

**Associations**

• private base\_InstanceSpecification : [InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

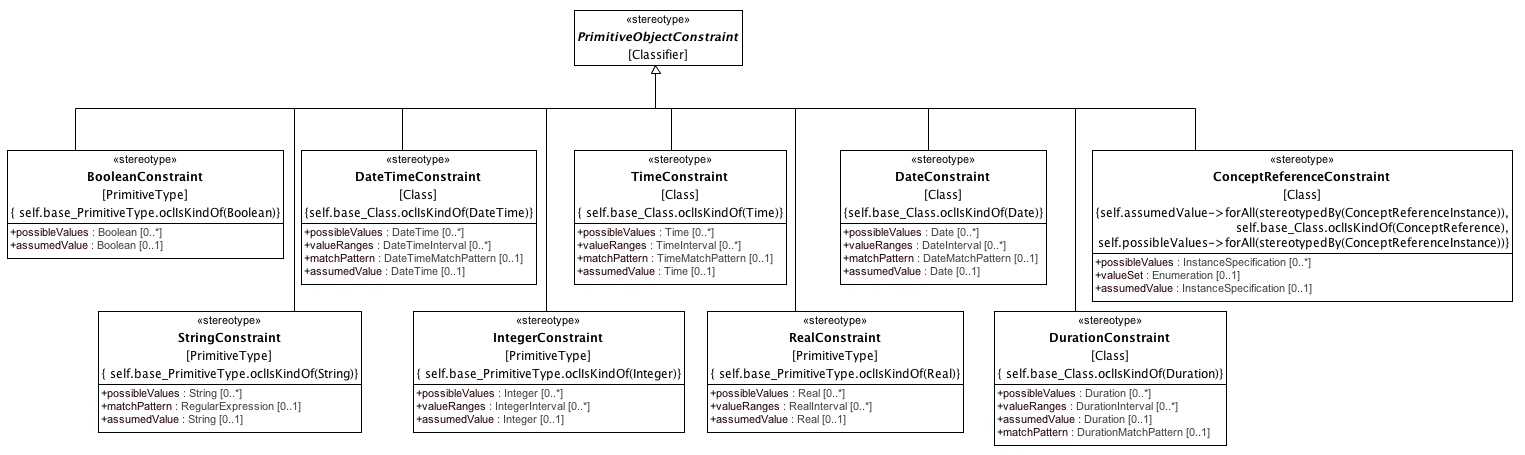
**Constraints**

• isResourceTranslation

[English]

The generalization closure of the base\_InstanceSpecification.classifier includes the ResourceTranslation class

### <Package> Data Type Constraints



**9.3.2.1 Primitive Type Constraints**

### <Stereotype> BooleanConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : Boolean [0..\*]

• public assumedValue : Boolean [0..1]

**Associations**

• private base\_PrimitiveType : [PrimitiveType](#_c88d02a0edd0a5fe4255d0ac6b9dec81)

**Constraints**

• constrainsBoolean

[OCL2.0]

self.base\_PrimitiveType.oclIsKindOf(Boolean)

### <Stereotype> ConceptReferenceConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : InstanceSpecification [0..\*]

• public valueSet : Enumeration [0..1]

• public assumedValue : InstanceSpecification [0..1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• assumedValue

[OCL2.0]

assumedValue.oclIsKindOf(ConceptReference)

• mustBeEnumeratedValueDomain

[OCL2.0]

self.valueSet->forAll(stereotypedBy('EnumeratedValueDomain'))

• constrainsConceptReference

[OCL2.0]

self.base\_Class.oclIsKindOf(ConceptReference)

• possibleValueReferences

[OCL2.0]

self.possibleValues->forAll(stereotypedBy(ConceptReferenceInstance))

• assumedValueReference

[OCL2.0]

self.assumedValue->forAll(stereotypedBy(ConceptReferenceInstance))

### <Stereotype> DateConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : [Date](#_78ee642abf9938398776ce11b2ae5595) [0..\*]

• public valueRanges : [DateInterval](#_eb65cb2938a6220d8f4a10f0d8aba136) [0..\*]

• public matchPattern : [DateMatchPattern](#_d029702cd1184b62db7fbbb690187cd5) [0..1]

• public assumedValue : [Date](#_78ee642abf9938398776ce11b2ae5595) [0..1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• constrainsDate

[OCL2.0]

self.base\_Class.oclIsKindOf(Date)

### <Stereotype> DateTimeConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : [DateTime](#_7ba7e85df09d292033e869c3e8664062) [0..\*]

• public valueRanges : [DateTimeInterval](#_956e6c028c830c0453b74cbd2204109e) [0..\*]

• public matchPattern : [DateTimeMatchPattern](#_0fe09862b0893b7f5636966c344bc986) [0..1]

• public assumedValue : [DateTime](#_7ba7e85df09d292033e869c3e8664062) [0..1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• assumedDateTime

[OCL2.0]

assumedValue.oclIsKindOf(DateTimeString)

• constrainsDateTime

[OCL2.0]

self.base\_Class.oclIsKindOf(DateTime)

### <Stereotype> DurationConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : [Duration](#_6f1e8a2b40ce6a6203e07d9c5daded71) [0..\*]

• public valueRanges : [DurationInterval](#_3d4fd0ce80d2a2e88d2f42b3cb7dbec5) [0..\*]

• public assumedValue : [Duration](#_6f1e8a2b40ce6a6203e07d9c5daded71) [0..1]

• public matchPattern : [DurationMatchPattern](#_f432daf955c8241a03bff56a5a7d8e66) [0..1]

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• constrainsDuration

[OCL2.0]

self.base\_Class.oclIsKindOf(Duration)

### <Stereotype> IntegerConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : Integer [0..\*]

• public valueRanges : [IntegerInterval](#_86618450de28d822bd6b57b67a32ab2b) [0..\*]

• public assumedValue : Integer [0..1]

**Associations**

• private base\_PrimitiveType : [PrimitiveType](#_c88d02a0edd0a5fe4255d0ac6b9dec81)

**Constraints**

• assumedInteger

[OCL2.0]

assumedValue.oclIsKindOf(Integer)

• constrainsInteger

[OCL2.0]

self.base\_PrimitiveType.oclIsKindOf(Integer)

### <Stereotype> PrimitiveObjectConstraint

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[BooleanConstraint](#_40ee863e6fd02692437dae1d81ba12de), [ConceptReferenceConstraint](#_ef76317db67a290898f39af3c51eee9c), [DateConstraint](#_ff8930d68c378c02c221704764a5c9d4), [DateTimeConstraint](#_7dde1322feeec9c32a95df44c39d8e48), [DurationConstraint](#_384c080719f5bd1b45eae1293215b466), [IntegerConstraint](#_2219fb1dcaf5f26a0ed07de77d69cd5e), [RealConstraint](#_b921bc493035fb4e067213114372e254), [StringConstraint](#_982033c222702fafb1d4d3ed7b399317), [TimeConstraint](#_d8c772ca77efc45bee8711f1de17afc0)

**Associations**

• private base\_Classifier : [Classifier](#_5f9a87915e1e9718a1a1cc45af995a70)

### <Stereotype> RealConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : Real [0..\*]

• public valueRanges : [RealInterval](#_d4f7314ff920dd15ee0e834cfbd4e6f2) [0..\*]

• public assumedValue : Real [0..1]

**Associations**

• private base\_PrimitiveType : [PrimitiveType](#_c88d02a0edd0a5fe4255d0ac6b9dec81)

**Constraints**

• constrainsReal

[OCL2.0]

self.base\_PrimitiveType.oclIsKindOf(Real)

### <Stereotype> StringConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : String [0..\*]

• public matchPattern : [RegularExpression](#_7dba62a18c5b1da31d9d2e3df675a297) [0..1]

• public assumedValue : String [0..1]

**Associations**

• private base\_PrimitiveType : [PrimitiveType](#_c88d02a0edd0a5fe4255d0ac6b9dec81)

**Constraints**

• assumedString

The assumed value must be a String

[OCL2.0]

assumedValue.oclIsKindOf(String)

• constrainsString

[OCL2.0]

self.base\_PrimitiveType.oclIsKindOf(String)

### <Stereotype> TimeConstraint

**Direct Known Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public possibleValues : [Time](#_cba83b2c77167c96697f3caaa1886f5c) [0..\*]

• public valueRanges : [TimeInterval](#_2db4f3574d756c0312a2a6559efd3ad9) [0..\*]

• public matchPattern : [TimeMatchPattern](#_4b701856b8ef798f1dd89caa4d9efe11) [0..1]

• public assumedValue : [Time](#_cba83b2c77167c96697f3caaa1886f5c) [0..1]

**Associations**

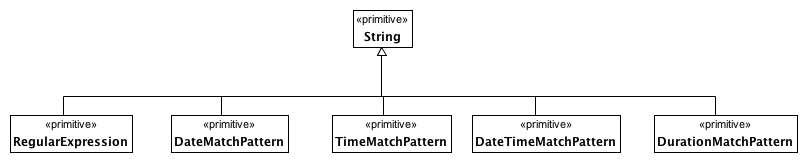
• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• constrainsTime

[OCL2.0]

self.base\_Class.oclIsKindOf(Time)



**9.3.2.2 Date and Time Match Types**



**9.3.2.3 Intervals**

### <Class> DateInterval

**Direct Known Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : [Date](#_78ee642abf9938398776ce11b2ae5595) [0..1]

• public upper : [Date](#_78ee642abf9938398776ce11b2ae5595) [0..1]

### <Class> DateTimeInterval

**Direct Known Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : [DateTime](#_7ba7e85df09d292033e869c3e8664062) [0..1]

• public upper : [DateTime](#_7ba7e85df09d292033e869c3e8664062) [0..1]

### <Class> DurationInterval

**Direct Known Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : [Duration](#_6f1e8a2b40ce6a6203e07d9c5daded71) [0..1]

• public upper : Duration [0..1]

### <Class> IntegerInterval

**Direct Known Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : Integer [0..1]

• public upper : Integer [0..1]

### <Class> Interval

**Direct Known Subclasses (Specialization)**

[DateInterval](#_eb65cb2938a6220d8f4a10f0d8aba136), [DateTimeInterval](#_956e6c028c830c0453b74cbd2204109e), [DurationInterval](#_3d4fd0ce80d2a2e88d2f42b3cb7dbec5), [IntegerInterval](#_86618450de28d822bd6b57b67a32ab2b), [RealInterval](#_d4f7314ff920dd15ee0e834cfbd4e6f2), [TimeInterval](#_2db4f3574d756c0312a2a6559efd3ad9)

**Attributes**

• public lowerIncluded : Boolean [1]

• public upperIncluded : Boolean [1]

### <Class> RealInterval

**Direct Known Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : Real [0..1]

• public upper : Real [0..1]

### <Class> TimeInterval

**Direct Known Superclasses (Generalization)**

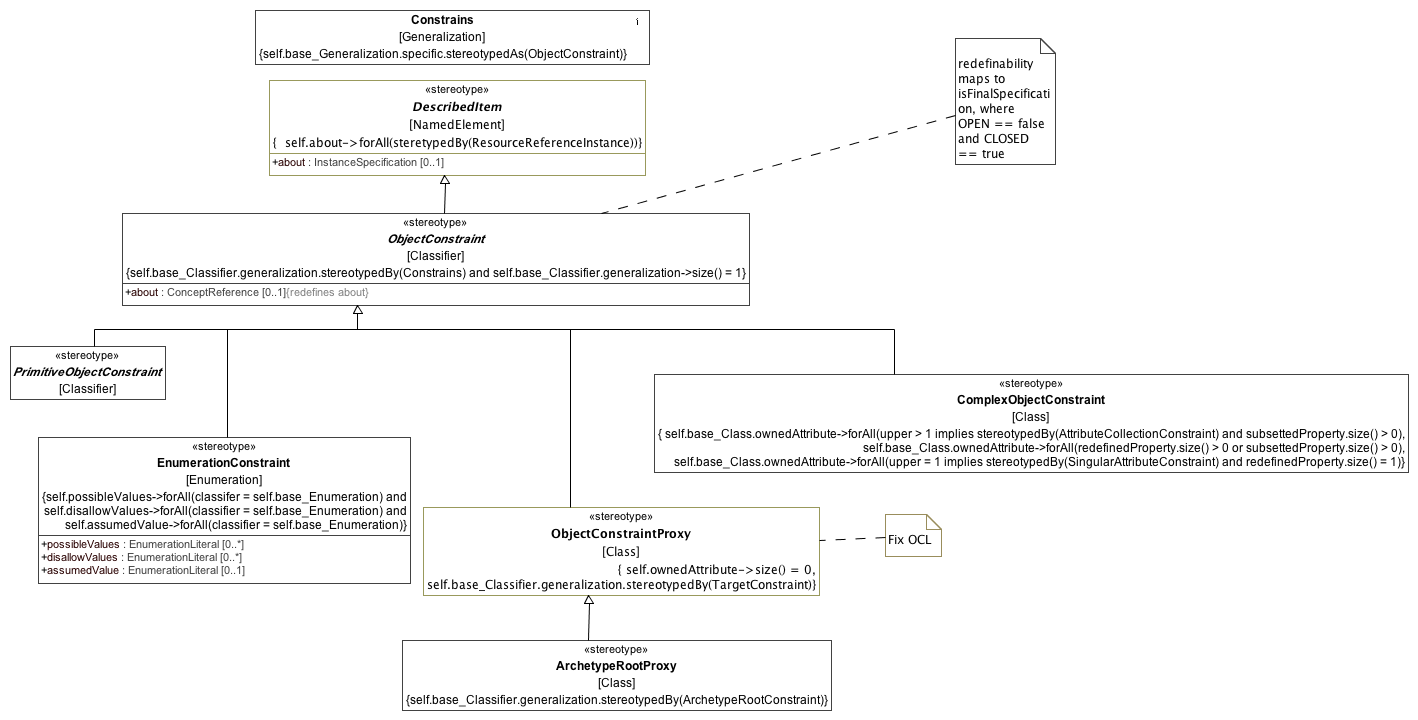
[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

• public lower : [Time](#_cba83b2c77167c96697f3caaa1886f5c) [0..1]

• public upper : [Time](#_cba83b2c77167c96697f3caaa1886f5c) [0..1]

### <Package> Object and Property Constraints



**9.3.3.1 Constraints**

### <Stereotype> ArchetypeRootProxy

**Direct Known Superclasses (Generalization)**

[ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• redefinesConstrains

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(ArchetypeRootConstraint)

### <Stereotype> ComplexObjectConstraint

**Description**

A ComplexObjectConstraint is a specialization of a single UML Class. A ComplexObjectConstraint may restrict the cardinality or possible values for one or more of the general class properties.

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[ArchetypeVersion](#_61831f1c446a753e3069251f603bfa37)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• singleParent

Every constraint must specialize exactly one Class

[OCL2.0]

generalization->size() = 1

• noNewAttributes

A ComplexObjectConstraint cannot introduce a new property, only subset or redefine existing ones.

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(redefinedProperty.size() > 0 or subsettedProperty.size() > 0)

• redefinesOwnedAttribute

Every owned attribute in the base class is also an attribute of type AttributeConstraint and every AttributeConstraint is also an owned attribute in the base class.

[OCL2.0]

self.attribute->forAll(a | self.base\_Class.ownedAttribute->includes(a.base\_Property)) and self.base\_Class.ownedAttribute->forAll(o | self.attribute->exists(a | a.base\_Property = o))

• unnamed1

[]

• unnamed2

[]

• allAttributeConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(stereotypedBy(AttributeConstraint)

• singularConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(upper = 1 implies stereotypedBy(SingularAttributeConstraint) and redefinedProperty.size() = 1)

•

[]

• attributeCollectionConstraints

[OCL2.0]

self.base\_Class.ownedAttribute->forAll(upper > 1 implies stereotypedBy(AttributeCollectionConstraint) and subsettedProperty.size() > 0)

### <Stereotype> Constrains

**Direct Known Subclasses (Specialization)**

[TargetConstraint](#_6de94cd3c6736f017766fe61020a5a13)

**Associations**

• private base\_Generalization : [Generalization](#_3596249272c463c4fc390555f5203d9e)

**Constraints**

• specificObjectConstraint

[OCL2.0]

self.base\_Generalization.specific.stereotypedAs(ObjectConstraint)

### <Stereotype> DescribedItem

**Direct Known Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77), [IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a), [ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2), [PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692)

**Attributes**

• public about : InstanceSpecification [0..1]

**Associations**

• private base\_NamedElement : [NamedElement](#_0907197416161d85b05eb0e9c47d90e9)

**Constraints**

• d

[OCL2.0]

self.about->forAll(steretypedBy(ResourceReferenceInstance))

### <Stereotype> EnumerationConstraint

**Description**

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomainConstraint](#_7b4688dbd3826f33c726c87847ae4a72)

**Attributes**

• public possibleValues : EnumerationLiteral [0..\*]

• public disallowValues : EnumerationLiteral [0..\*]

• public assumedValue : EnumerationLiteral [0..1]

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• ownedLiterals

[OCL2.0]

self.possibleValues->forAll(classifer = self.base\_Enumeration) and self.disallowValues->forAll(classifier = self.base\_Enumeration) and self.assumedValue->forAll(classifier = self.base\_Enumeration)

### <Stereotype> ObjectConstraint

**Description**

The Classifier stereotyped by an ObjectConstraint must participate in exactly one Generalization relationship, which must be stereotyped by the Constrains stereotype.

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Direct Known Subclasses (Specialization)**

[ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62), [EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9), [ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e), [PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public about : [ConceptReference](#_ae987614d7a9f06b4da55c1b58dd49d6) [0..1]

**Associations**

• private base\_Classifier : [Classifier](#_5f9a87915e1e9718a1a1cc45af995a70)

**Constraints**

• redefinesGeneralization

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(Constrains) and self.base\_Classifier.generalization->size() = 1

### <Stereotype> ObjectConstraintProxy

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootProxy](#_f5f73ce565f73d8b4808997e54e8e698)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• redefinesConstrains

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(TargetConstraint)

• noNewAttributes

[OCL2.0]

self.ownedAttribute->size() = 0

### <Stereotype> PrimitiveObjectConstraint

**Direct Known Superclasses (Generalization)**

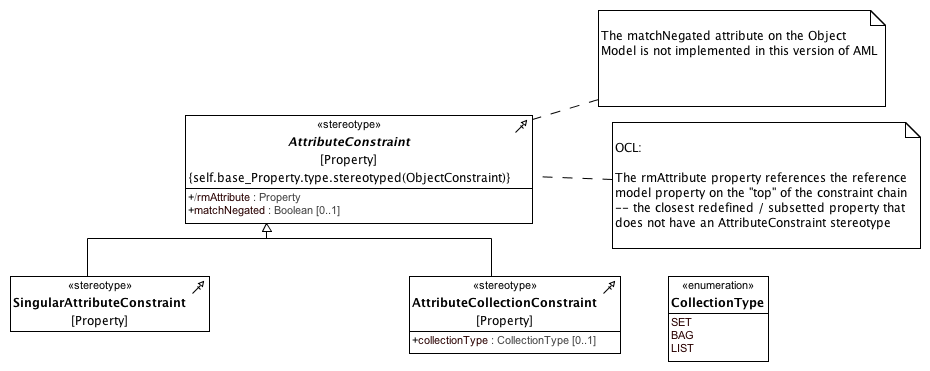
[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[BooleanConstraint](#_40ee863e6fd02692437dae1d81ba12de), [ConceptReferenceConstraint](#_ef76317db67a290898f39af3c51eee9c), [DateConstraint](#_ff8930d68c378c02c221704764a5c9d4), [DateTimeConstraint](#_7dde1322feeec9c32a95df44c39d8e48), [DurationConstraint](#_384c080719f5bd1b45eae1293215b466), [IntegerConstraint](#_2219fb1dcaf5f26a0ed07de77d69cd5e), [RealConstraint](#_b921bc493035fb4e067213114372e254), [StringConstraint](#_982033c222702fafb1d4d3ed7b399317), [TimeConstraint](#_d8c772ca77efc45bee8711f1de17afc0)

**Associations**

• private base\_Classifier : [Classifier](#_5f9a87915e1e9718a1a1cc45af995a70)



**9.3.3.2 Attribute Constraints**

### <Enumeration> CollectionType

**Enumeration Literals**

* **BAG**

* **LIST**

* **SET**

### <Stereotype> AttributeCollectionConstraint

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_1bf8a3231ae21af2dec84426b5618c38)

**Attributes**

• public collectionType : [CollectionType](#_3b22dea7c2d9e1b7b47a41449ca5085f) [0..1]

**Associations**

• private base\_Property : [Property](#_9624701f3c3fcd329e6dba1fc74db836)

### <Stereotype> AttributeConstraint

**Direct Known Subclasses (Specialization)**

[AttributeCollectionConstraint](#_4bc615eb2707782fc8254702b7e0b435), [SingularAttributeConstraint](#_2d1a6d8b2806092b50ec3fd4cd2db35b)

**Attributes**

• public rmAttribute : Property

• public matchNegated : Boolean [0..1]

**Associations**

• private base\_Property : [Property](#_9624701f3c3fcd329e6dba1fc74db836)

**Constraints**

• targetSubsetsType

[OCL2.0]

self.base\_Property.type->includes(self.target)

• objectConstraintTarget

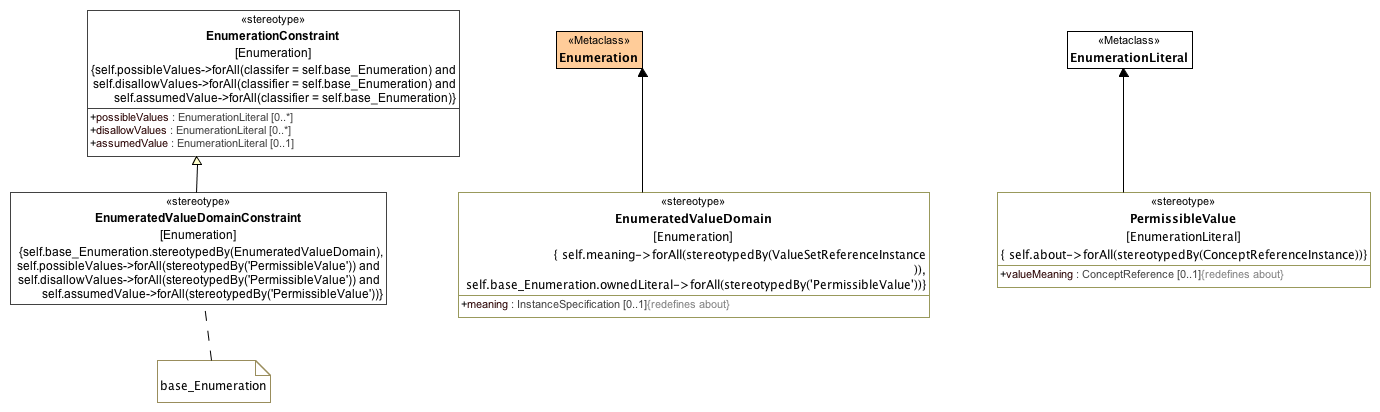
[OCL2.0]

self.base\_Property.type.stereotyped(ObjectConstraint)

### <Stereotype> SingularAttributeConstraint

**Direct Known Superclasses (Generalization)**

[AttributeConstraint](#_1bf8a3231ae21af2dec84426b5618c38)



**9.3.3.3 Enumeration Constraints**

### <Class> Enumeration

**Direct Known Superclasses (Generalization)**

[DataType](#_687a198ec4e5e49dcd9a605729dc1c24)

### <Class> EnumerationLiteral

**Direct Known Superclasses (Generalization)**

[InstanceSpecification](#_a0d9dad3458f9785e2dd80931202582d)

### <Stereotype> EnumeratedValueDomain

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Attributes**

• public meaning : InstanceSpecification [0..1]

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• permissibleValues

[OCL2.0]

self.base\_Enumeration.ownedLiteral->forAll(stereotypedBy('PermissibleValue'))

• meaningIsConceptReference

[OCL2.0]

self.meaning->forAll(stereotypedBy(ValueSetReferenceInstance ))

### <Stereotype> EnumeratedValueDomainConstraint

**Direct Known Superclasses (Generalization)**

[EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9)

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• constrainsEVD

[OCL2.0]

self.base\_Enumeration.stereotypedBy(EnumeratedValueDomain)

• possiblePermissibleValue

[OCL2.0]

self.possibleValues->forAll(stereotypedBy('PermissibleValue')) and self.disallowValues->forAll(stereotypedBy('PermissibleValue')) and self.assumedValue->forAll(stereotypedBy('PermissibleValue'))

### <Stereotype> EnumerationConstraint

**Description**

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomainConstraint](#_7b4688dbd3826f33c726c87847ae4a72)

**Attributes**

• public possibleValues : EnumerationLiteral [0..\*]

• public disallowValues : EnumerationLiteral [0..\*]

• public assumedValue : EnumerationLiteral [0..1]

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• ownedLiterals

[OCL2.0]

self.possibleValues->forAll(classifer = self.base\_Enumeration) and self.disallowValues->forAll(classifier = self.base\_Enumeration) and self.assumedValue->forAll(classifier = self.base\_Enumeration)

### <Stereotype> PermissibleValue

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Attributes**

• public valueMeaning : [ConceptReference](#_ae987614d7a9f06b4da55c1b58dd49d6) [0..1]

**Associations**

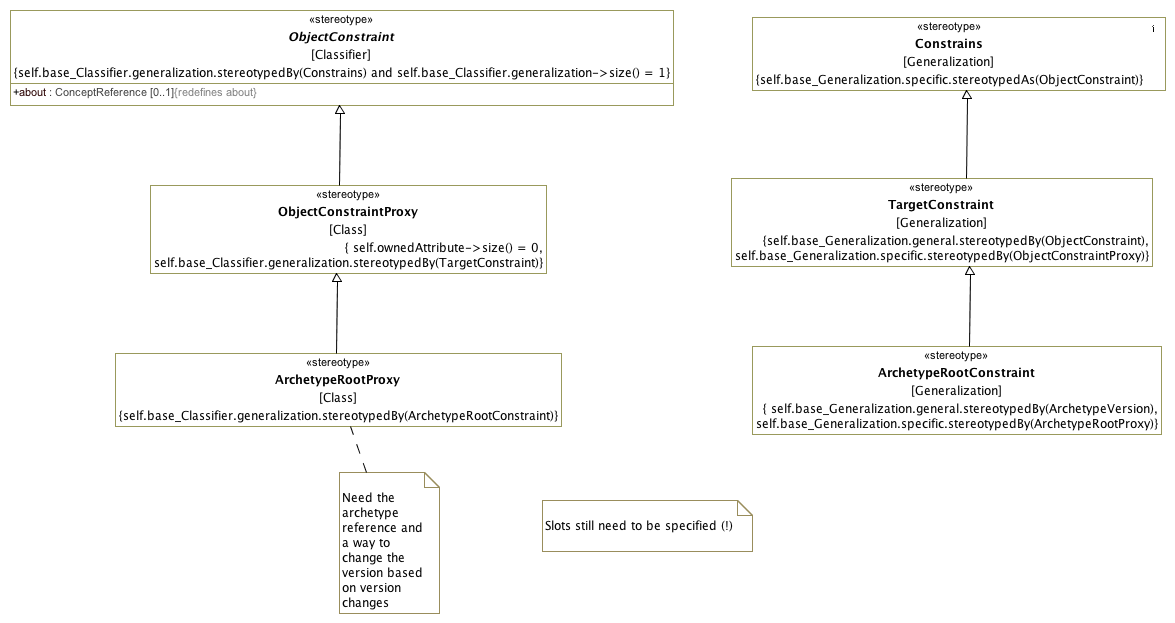
• private base\_EnumerationLiteral : [EnumerationLiteral](#_cba107636e03e0cb80fbb911dd86b703)

**Constraints**

• valueMeaningIsConceptReference

[OCL2.0]

self.about->forAll(stereotypedBy(ConceptReferenceInstance))



**9.3.3.4 Constraint Proxies**

### <Stereotype> ArchetypeRootConstraint

**Direct Known Superclasses (Generalization)**

[TargetConstraint](#_6de94cd3c6736f017766fe61020a5a13)

**Associations**

• private base\_Generalization : [Generalization](#_3596249272c463c4fc390555f5203d9e)

**Constraints**

• specificIsRootProxy

[OCL2.0]

self.base\_Generalization.specific.stereotypedBy(ArchetypeRootProxy)

• generalIsArchetypeVersion

[OCL2.0]

self.base\_Generalization.general.stereotypedBy(ArchetypeVersion)

### <Stereotype> ArchetypeRootProxy

**Direct Known Superclasses (Generalization)**

[ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• redefinesConstrains

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(ArchetypeRootConstraint)

### <Stereotype> Constrains

**Direct Known Subclasses (Specialization)**

[TargetConstraint](#_6de94cd3c6736f017766fe61020a5a13)

**Associations**

• private base\_Generalization : [Generalization](#_3596249272c463c4fc390555f5203d9e)

**Constraints**

• specificObjectConstraint

[OCL2.0]

self.base\_Generalization.specific.stereotypedAs(ObjectConstraint)

### <Stereotype> ObjectConstraint

**Description**

The Classifier stereotyped by an ObjectConstraint must participate in exactly one Generalization relationship, which must be stereotyped by the Constrains stereotype.

**Direct Known Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Direct Known Subclasses (Specialization)**

[ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62), [EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9), [ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e), [PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

• public about : [ConceptReference](#_ae987614d7a9f06b4da55c1b58dd49d6) [0..1]

**Associations**

• private base\_Classifier : [Classifier](#_5f9a87915e1e9718a1a1cc45af995a70)

**Constraints**

• redefinesGeneralization

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(Constrains) and self.base\_Classifier.generalization->size() = 1

### <Stereotype> ObjectConstraintProxy

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootProxy](#_f5f73ce565f73d8b4808997e54e8e698)

**Associations**

• private base\_Class : [Class](#_bdb071ba1da37974bd8d9639d911511b)

**Constraints**

• redefinesConstrains

[OCL2.0]

self.base\_Classifier.generalization.stereotypedBy(TargetConstraint)

• noNewAttributes

[OCL2.0]

self.ownedAttribute->size() = 0

### <Stereotype> TargetConstraint

**Direct Known Superclasses (Generalization)**

[Constrains](#_f91b532413834ad1de94d0b0af526f5b)

**Direct Known Subclasses (Specialization)**

[ArchetypeRootConstraint](#_6eafe370f24f01390e7ab79d6568ea94)

**Associations**

• private base\_Generalization : [Generalization](#_3596249272c463c4fc390555f5203d9e)

**Constraints**

• specificIsProxy

[OCL2.0]

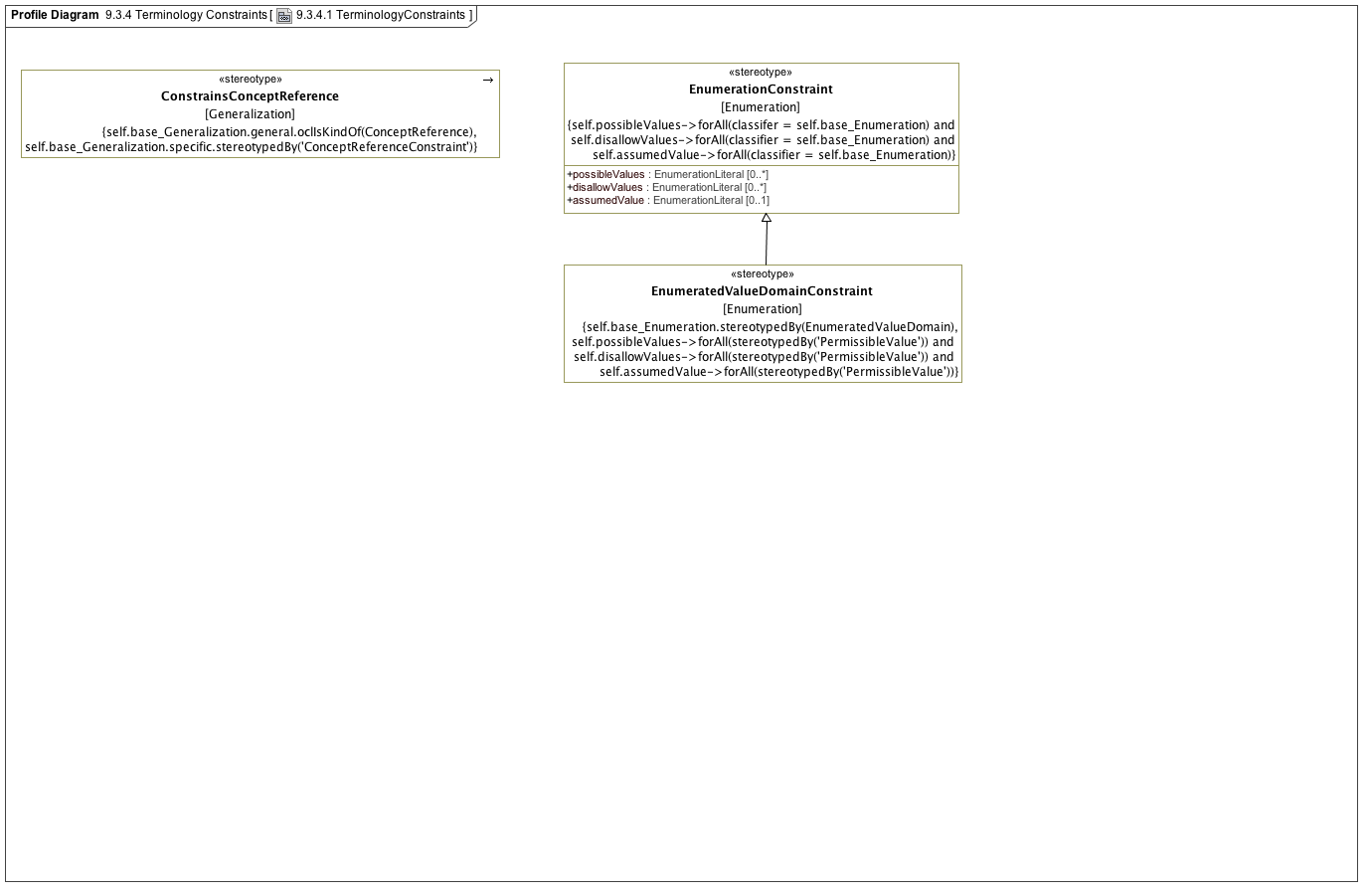
self.base\_Generalization.specific.stereotypedBy(ObjectConstraintProxy)

• generalIsObjectConstraint

[OCL2.0]

self.base\_Generalization.general.stereotypedBy(ObjectConstraint)

### <Package> Terminology Constraints



**9.3.4.1 TerminologyConstraints**

### <Stereotype> ConstrainsConceptReference

**Associations**

• private base\_Generalization : [Generalization](#_3596249272c463c4fc390555f5203d9e)

**Constraints**

• specificCRC

[OCL2.0]

self.base\_Generalization.specific.stereotypedBy('ConceptReferenceConstraint')

• generalIsConceptReference

[OCL2.0]

self.base\_Generalization.general.oclIsKindOf(ConceptReference)

### <Stereotype> EnumeratedValueDomainConstraint

**Direct Known Superclasses (Generalization)**

[EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9)

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• constrainsEVD

[OCL2.0]

self.base\_Enumeration.stereotypedBy(EnumeratedValueDomain)

• possiblePermissibleValue

[OCL2.0]

self.possibleValues->forAll(stereotypedBy('PermissibleValue')) and self.disallowValues->forAll(stereotypedBy('PermissibleValue')) and self.assumedValue->forAll(stereotypedBy('PermissibleValue'))

### <Stereotype> EnumerationConstraint

**Description**

**Direct Known Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Known Subclasses (Specialization)**

[EnumeratedValueDomainConstraint](#_7b4688dbd3826f33c726c87847ae4a72)

**Attributes**

• public possibleValues : EnumerationLiteral [0..\*]

• public disallowValues : EnumerationLiteral [0..\*]

• public assumedValue : EnumerationLiteral [0..1]

**Associations**

• private base\_Enumeration : [Enumeration](#_e3fb46f85efa560d0f6c20afc26f4f5b)

**Constraints**

• ownedLiterals

[OCL2.0]

self.possibleValues->forAll(classifer = self.base\_Enumeration) and self.disallowValues->forAll(classifier = self.base\_Enumeration) and self.assumedValue->forAll(classifier = self.base\_Enumeration)