|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | 75 | Date: | February 2015 | |
| Archetype Modeling Language (AML)    Version: 1.0  **OMG Document Number: health/2014-10-01**  **Standard document URL: http://www.omg.org/spec/AML/1.0**  Original File: N/A |

Copyright © 2014, Mayo Clinic  
Copyright © 2014, Object Management Group, Inc.  
Copyright © 2014, Visumpoint, LLC

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

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

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

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

RESTRICTED RIGHTS LEGEND

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

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , IMM™ , 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>.)

Table of Contents

1. Scope 9

1.1. Archetype Modeling Language (AML) Background 9

1.2. AML Intended Users 10

1.3. AML Profiles 10

2. Conformance 11

2.1. Conformance Points 11

2.2. AML Reference Model Profile 11

2.3. AML Terminology Binding Profile 11

2.4. AML Constraint Model Profile 11

3. Normative References 11

4. Terms and Definitions 12

5. Symbols 16

5.1. Graphical Symbols 16

5.2. Abbreviations 16

6. Additional Information 17

6.1. Changes to Adopted OMG Specifications 17

6.2. Acknowledgements 17

7. The AOM and the AML Metamodel 17

8. Profiles 18

8.1. Dependencies 19

8.2. Reference Model Profile 19

8.2.1. Reference Model Stereotypes 20

8.2.2. AML Data Types 20

8.2.3. Profile Elements 21

8.2.3.1. <Class> DV\_Date 21

8.2.3.2. <Class> DV\_DateTime 22

8.2.3.3. <Class> DV\_Duration 22

8.2.3.4. <Class> DV\_Time 23

8.2.3.5. <Class> TerminologyCode 23

8.2.3.6. <Stereotype> DataBinding 24

8.2.3.7. <Stereotype> Infrastructure 25

8.2.3.8. <Stereotype> MappedDataType 25

8.2.3.9. <Stereotype> ReferenceModel 26

8.2.3.10. <Stereotype> Runtime 26

8.3. Terminology Profile 27

8.3.1. Identification and Designation 27

8.3.2. Resource References 28

8.3.2.1. Resource References 28

8.3.2.2. Resource Reference Relationships 29

8.3.3. Enumerated Value Domains 30

8.3.4. Profile Elements 30

8.3.4.1. <Class> DV\_Date 30

8.3.4.2. <Class> DV\_DateTime 30

8.3.4.3. <Class> DV\_Duration 31

8.3.4.4. <Class> DV\_Time 31

8.3.4.5. <Class> TerminologyCode 32

8.3.4.6. <Stereotype> ArchetypeType 33

8.3.4.7. <Stereotype> AssumedValue 33

8.3.4.8. <Stereotype> CodeSystemReference 34

8.3.4.9. <Stereotype> CodeSystemReferenceInstance 34

8.3.4.10. <Stereotype> CodeSystemVersionReference 34

8.3.4.11. <Stereotype> ConceptReference 35

8.3.4.12. <Stereotype> ConceptReferenceInstance 35

8.3.4.13. <Stereotype> DataBinding 36

8.3.4.14. <Stereotype> DescribedItem 37

8.3.4.15. <Stereotype> DesignatableItem 38

8.3.4.16. <Stereotype> EnumeratedValueDomain 38

8.3.4.17. <Stereotype> IdentifiedItem 39

8.3.4.18. <Stereotype> Infrastructure 40

8.3.4.19. <Stereotype> KnownNamespace 40

8.3.4.20. <Stereotype> Language 41

8.3.4.21. <Stereotype> MappedDataType 41

8.3.4.22. <Stereotype> NamespaceInstance 41

8.3.4.23. <Stereotype> PermissibleValue 42

8.3.4.24. <Stereotype> PossibleValue 42

8.3.4.25. <Stereotype> ReferenceModel 43

8.3.4.26. <Stereotype> ResourceReference 43

8.3.4.27. <Stereotype> ResourceReferenceInstance 44

8.3.4.28. <Stereotype> Runtime 45

8.3.4.29. <Stereotype> ScopedIdentifier 45

8.3.4.30. <Stereotype> ValueSetDefinitionReference 45

8.3.4.31. <Stereotype> ValueSetReference 46

8.3.4.32. <Stereotype> ValueSetReferenceInstance 46

8.4. Constraint Profile 47

8.4.1. Archetypes 47

8.4.1.1. Archetype Libraries 47

8.4.1.2. Archetype Metadata 48

8.4.2. Data Type Constraints 49

8.4.2.1. Primitive Type Constraints 49

8.4.2.2. Date and Time Match Types 49

8.4.2.3. Intervals 50

8.4.3. Object and Property Constraints 50

8.4.3.1. Object Constraints 50

8.4.3.2. Attribute Constraints 51

8.4.3.3. Enumeration Constraints 51

8.4.3.4. Constraint Proxies 51

8.4.4. Terminology Constraints 52

8.4.5. Profile Elements 52

8.4.5.1. <Class> DateInterval 52

8.4.5.2. <Class> DateTimeInterval 53

8.4.5.3. <Class> DurationInterval 53

8.4.5.4. <Class> DV\_Date 54

8.4.5.5. <Class> DV\_DateTime 54

8.4.5.6. <Class> DV\_Duration 55

8.4.5.7. <Class> DV\_Time 55

8.4.5.8. <Class> IntegerInterval 55

8.4.5.9. <Class> Interval 56

8.4.5.10. <Class> RealInterval 56

8.4.5.11. <Class> TerminologyCode 57

8.4.5.12. <Class> TimeInterval 58

8.4.5.13. <Stereotype> AMLType 59

8.4.5.14. <Stereotype> Archetype 59

8.4.5.15. <Stereotype> ArchetypeCurrentVersion 60

8.4.5.16. <Stereotype> ArchetypeId 61

8.4.5.17. <Stereotype> ArchetypeLibrary 61

8.4.5.18. <Stereotype> ArchetypeRootConstraint 62

8.4.5.19. <Stereotype> ArchetypeRootProxy 62

8.4.5.20. <Stereotype> ArchetypeType 63

8.4.5.21. <Stereotype> ArchetypeType 63

8.4.5.22. <Stereotype> ArchetypeVersion 63

8.4.5.23. <Stereotype> ArchetypeVersionId 64

8.4.5.24. <Stereotype> AssumedValue 64

8.4.5.25. <Stereotype> AttributeCollectionConstraint 65

8.4.5.26. <Stereotype> AttributeConstraint 65

8.4.5.27. <Stereotype> AuthoredResource 66

8.4.5.28. <Stereotype> BooleanConstraint 67

8.4.5.29. <Stereotype> CodeSystemReference 67

8.4.5.30. <Stereotype> CodeSystemReferenceInstance 68

8.4.5.31. <Stereotype> CodeSystemVersionReference 68

8.4.5.32. <Stereotype> ComplexObjectConstraint 69

8.4.5.33. <Stereotype> ConceptReference 69

8.4.5.34. <Stereotype> ConceptReferenceInstance 70

8.4.5.35. <Stereotype> Constrains 70

8.4.5.36. <Stereotype> DataBinding 71

8.4.5.37. <Stereotype> DateConstraint 72

8.4.5.38. <Stereotype> DateTimeConstraint 73

8.4.5.39. <Stereotype> DescribedItem 73

8.4.5.40. <Stereotype> DesignatableItem 74

8.4.5.41. <Stereotype> DurationConstraint 75

8.4.5.42. <Stereotype> EnumeratedValueDomain 76

8.4.5.43. <Stereotype> EnumeratedValueDomainConstraint 76

8.4.5.44. <Stereotype> EnumerationConstraint 77

8.4.5.45. <Stereotype> IdentifiedItem 78

8.4.5.46. <Stereotype> Infrastructure 78

8.4.5.47. <Stereotype> IntegerConstraint 79

8.4.5.48. <Stereotype> KnownNamespace 79

8.4.5.49. <Stereotype> Language 80

8.4.5.50. <Stereotype> MappedDataType 80

8.4.5.51. <Stereotype> NamespaceInstance 81

8.4.5.52. <Stereotype> ObjectConstraint 81

8.4.5.53. <Stereotype> ObjectConstraintProxy 82

8.4.5.54. <Stereotype> PermissibleValue 83

8.4.5.55. <Stereotype> PossibleValue 83

8.4.5.56. <Stereotype> PrimitiveObjectConstraint 83

8.4.5.57. <Stereotype> RealConstraint 84

8.4.5.58. <Stereotype> ReferenceModel 85

8.4.5.59. <Stereotype> ReferenceModel 85

8.4.5.60. <Stereotype> ReferenceModelImport 86

8.4.5.61. <Stereotype> ResourceDescription 86

8.4.5.62. <Stereotype> ResourceReference 87

8.4.5.63. <Stereotype> ResourceReferenceInstance 87

8.4.5.64. <Stereotype> ResourceTranslation 88

8.4.5.65. <Stereotype> Runtime 88

8.4.5.66. <Stereotype> ScopedIdentifier 89

8.4.5.67. <Stereotype> SingularAttributeConstraint 89

8.4.5.68. <Stereotype> StringConstraint 90

8.4.5.69. <Stereotype> TargetConstraint 90

8.4.5.70. <Stereotype> TerminologyCodeConstraint 91

8.4.5.71. <Stereotype> TimeConstraint 92

8.4.5.72. <Stereotype> TranslationDetails 92

8.4.5.73. <Stereotype> ValueSetDefinitionReference 93

8.4.5.74. <Stereotype> ValueSetReference 93

8.4.5.75. <Stereotype> ValueSetReferenceInstance 94

9. Appendix A: AML Meta Model 94

9.1. AArchetype Meta Model 95

9.1.1. Archetype Libraries 95

9.1.2. Archetypes and the UML Reference Model 95

9.1.3. ADL Archetype Metadata 96

9.2. AReference Model Meta Model 97

9.2.1. Primitive Data Types 97

9.2.2. Reference Metamodel 99

9.2.3. Attribute Constraint References 100

9.2.4. Object Constraint References 100

9.2.5. Template Metamodel 101

9.2.6. Instance Metamodel 101

9.2.7. Package Metamodel 103

9.2.8. Enumeration Metamodel 103

9.3. AConstraint Meta Model 104

9.3.1. Atomic Data Type Constraints 104

9.3.2. Terminology Constraints 104

9.3.3. Enumeration Constraints 106

9.3.4. Object Constraints 106

9.3.4.1. Named Object Constraints 107

9.3.5. Attribute Constraints 108

9.4. ATerminology Object Model 108

9.4.1. Common Terminology Services Components 108

9.4.2. ISO 11179 Model Components 109

9.4.2.1. Designation, Definition and Identification 109

9.4.2.2. Describable Items in AML 110

9.4.3. AML Described Items 112

10. B Reference Model Profile Examples 112

10.1. BDataBinding Example 112

10.1.1. BSample Reference Model 112

10.1.2. BSample DataBinding 113

10.1.3. BSample Constraint 113

Appendix A: AML MetaModel 114

**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

## Archetype Modeling Language (AML) Background

This specification defines the Archetype Modeling Language (AML). The AML defines a standard means for modeling Archetype Models (AMs) to support the representation of Clinical Information Modeling Initiative (CIMI) artifacts using modeling profiles as defined in the UML. Archetype Models are Platform Independent Models (PIMs) and are developed as a set of constraints on a specific Reference Model (RM).

The CIMI RM is the underlying RM on which CIMI’s clinical information models are defined. The reference model defines a rigorous and stable set of modeling patterns that include a set of structural patterns, complex data types, and demographic classes. All CIMI clinical models will be defined by constraining the CIMI reference model. Each instance of a CIMI Clinical Model will be a constrained instance of the CIMI reference model conforming to the constraints defined by the associated clinical model.

The motivation for including a reference model in the CIMI clinical modeling architecture is to provide a consistent computational framework upon which model authoring and translation tools can be based. The reference model is the ‘common language’ used to describe all clinical models. It provides a single information model that can be used to represent instances of all clinical models and upon which further constraints can be applied to represent the specific information requirements of all clinical model. This information model represents the core artifact implemented in software; it provides the physical structure of the clinical models and its example instances. Existing implementation experience has shown this increases the computational capabilities of the resulting modeling and translation tools.

Development of the AML specification was guided by:

1. The need for a means to accurately and usefully represent AMs in accordance with the openEHR Foundation’s Archetype Definition Language (ADL) and Archetype Object Model (AOM) version 2.0 specifications;
2. Compatibility with the Object Management Group (OMG) *Common Terminology Service 2 (CTS2)* specification; and
3. Where possible, being informed by and faithful to the *ISO/IEC 11179, Information Technology, -- Metadata registries*, specification.

In the AML RFP, the version of the openEHR Foundation’s ADL and AOM specifications cited for coverage by the OMG AML specification was version 1.5. In the process of producing the AML specification, however, a number of inconsistencies were discovered in the openEHR specifications, as well as opportunities for improvements. These were reported to the openEHR Foundation. In response, the openEHR Foundation revised the specifications. This resulted in a set of changes to the specifications that were not backward compatible with version 1.5. As a consequence, the revised specifications were released as version 2.0, subsuming the requirements found in version 1.5, now made consistent in version 2.0, and forming the updated requirements basis for AML coverage.

## AML Intended Users

The AML is primarily intended to support two clinical modeling communities of users:

* Those having subject matter expertise regarding clinical model domains and currently using ADL-based tools to develop such models, and
* Those familiar with modeling using the UML, though not necessarily familiar with clinical modeling domains or current methods employed to represent them.

Clause 7 of this specification, *AML Meta Model*, provides an informational meta model of the openEHR AOM as an aid to bridging between these communities.

While the AML specification targets CIMI clinical modeling practitioners, the modeling approach defined in the profiles is intended to be generalizable for use with other reference models and application in other domain areas.

## AML Profiles

The AML is specified by three UML profilescollectively meeting the requirements of archetype modeling. These are the:

* *Reference Model Profile (RMP)*: Enables the specification of reference models upon which archetypes can be based;
* *Constraint Model Profile (CMP)*: Supports the specification of constraints on a given reference model to enable the development of archetypes including Clinical Information Models (CIMs); and
* *Terminology Binding Profile (TBP)*: Supports the binding of information models to terminology. Terminology bindings include:
  1. *Value Bindings*: Support linking the data model to value domains that restrict the valid value of an attribute to a set of values corresponding to a set of meanings recorded in an external terminology;
  2. *Semantic Bindings:* Define the meaning of model elements using concepts in an external terminology; and
  3. *Constraint Bindings:* Specify constraints on the information model using concepts and relationships defined in an external terminology.

This set of UML profiles enables the specification of CIMI clinical model content (using the CIMI Reference Model) and the generation of CIMI clinical model artifacts, such as ones represented by the openEHR Foundation’s ADL. (The ADL is a serialization of the openEHR Foundation’s AOM.) While the transformation of AML models to an instance of the AOM was an optional requirement for the AML specification, the AML profile supports the representation of sufficient information in an AM to enable such a transformation.

# Conformance

## Conformance Points

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

* AML Reference Model Profile
* AML Terminology Binding Profile
* AML Constraint Model Profile

## AML Reference Model Profile

Sub clause 8.1 of this specification defines the AML Reference Model Profile.

## AML Terminology Binding Profile

Sub clause 8.2 of this specification defines the AML Terminology Binding Profile. The Terminology Binding Profile imports the Reference Model Profile.

## AML Constraint Model Profile

Sub clause 8.3 of this specification defines the AML Constraint Model Profile. The Constraint Model Profile imports both the Reference Model Profile and Terminology Binding Profile.

# Normative References

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

[ADL] openEHR *Archetype Definition Language: ADL2*, <http://www.openehr.org/releases/trunk/architecture/am/adl2.pdf>

[AOM] *openEHR Archetype Object Model* (AOM), <http://www.openehr.org/releases/trunk/architecture/am/aom2.pdf>

[AOMT] openEHR *openEHR Templates* (supersedes *openEHR Archetype Templates*), <http://www.openehr.org/releases/trunk/architecture/am/tom.pdf>

[ARCH] *openEHR Archetypes: Constraint-based Domain Models for Future-proof Information Systems*, <http://www.openehr.org/publications/archetypes/archetypes_beale_oopsla_2002.pdf>

[CIMI] CIMI Reference Model Requirements, <http://informatics.mayo.edu/CIMI/index.php/CIMI_Reference_Model_Requirements>

[CTS2] OMG *Common Terminology Service 2 (CTS2)*, [http://www.omg.org/spec/CTS2/1.1/](http://www.omg.org/spec/CTS2/1.1/" \o "http://www.omg.org/spec/CTS2/1.1/)

[HLV7v3] *HL7 Version 3 Standard: Core Principles and Properties of Version 3 Models*, <http://www.hl7.org/implement/standards/product_brief.cfm?product_id=58>

[MDMI] OMG *Model Driven Message Interoperability (MDMI), Version 1.0*, <http://www.omg.org/spec/MDMI/1.0/>

[MDR] *ISO/IEC 11179, Information Technology, -- Metadata registries*, [http://metadata-standards.org/11179/](http://metadata-standards.org/11179/" \o "http://metadata-standards.org/11179/)

[NIEM] OMG *UML Profile for NIEM Version 1.0*, [http://www.omg.org/spec/NIEM-UML/1.0/](http://www.omg.org/spec/NIEM-UML/1.0/" \o "http://www.omg.org/spec/NIEM-UML/1.0/)

[OCL] OMG *Object Constraint Language (OCL), Version 2.4*, <http://www.omg.org/spec/OCL/2.4/>

[ODM] OMG *Ontology Definition Metamodel (ODM) Version 1.1*, <http://www.omg.org/spec/ODM/1.1/>

[QVT] OMG *Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.2 (Beta)*, <http://www.omg.org/spec/QVT/1.2/Beta/>

[UML] OMG *Unified Modeling Language (UML) Version 2.5 – Beta 2*, <http://www.omg.org/spec/UML/2.5/Beta2/>

# Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Archetype

An archetype is a re-usable formal definition of domain level information defined in terms of constraints on an information model. The key feature of the archetype approach to computing is a complete separation of information models (such as object models of software or models of database schemas) from domain models.

Archetype Definition Language (ADL)

ADL is a formal language for expressing archetypes. It provides a formal, textual syntax for describing constraints on any domain entity whose data is described by an information model (also known as the 'underlying reference model'). The ADL syntax is semantically equivalent to the AOM and represents one possible serialization of the AOM. The current version of ADL is known as 'ADL 2'.

Archetype Instance

An archetype instance is a single instantiation of data conforming to a specific archetype. In the context of CIMI this data will typically be clinical.

Archetype Model (AM)

An AM is a re-usable, formal model of an archetype expressed as a computable set of constraint statements on an underlying reference model (URM). Concepts that can be modeled using archetypes include weight measurement, blood pressure, microbiology results, discharge referral, prescription, or diagnosis. CIMI archetypes will be represented as an instance of the ‘Archetype Object Model’.

Archetype Object Model (AOM)

The AOM is the definitive expression of archetype semantics and is independent of any particular syntax. It is defined as an object model using a UML class diagram. It is a generic model, meaning it can be used to express archetypes for any reference model in a standard way. Version 1.4 of the AOM was standardized in ISO-13606:2. The current version is known as 'AOM 2'.

Archetype Query Language (AQL)

The AQL is a declarative query language developed specifically for expressing queries used for searching and retrieving the clinical data found in archetype-based EHRs. AQL expresses queries at the archetype level, i.e. semantic level, and not at the data instance level. This is key to achieving shared queries across system or enterprise boundaries.

Clinical Data Repository (CDR)

A CDR is a data store holding and managing clinical data collected from service encounters at the point-of-service locations such as hospitals, clinics, etc.

Clinical Document Architecture (CDA)

A CDA is an HL7 XML-based markup standard intended to specify the encoding, structure, and semantics of clinical documents for exchange.

Clinical Information Model (CIM)

A CIM is a representation of the structured clinical information (including relationships, constraints and terminology) describing a specific clinical concept - e.g. a blood pressure observation, a Discharge Summary, or a Medication Order.

Clinical Information Modeling Initiative (CIMI)

CIMI is an initiative established to “improve the interoperability of healthcare information systems through shared implementable clinical information models.”

Clinical Information Modeling Initiative (CIMI) Reference Model (RM)

The CIMI RM is the underlying Reference Model on which CIMI's clinical models (i.e. archetypes) are defined. This reference model defines a rigorous and stable set of modeling patterns, including a set of complex data types, information patterns (e.g. data, qualifier, state), and structural patterns (e.g. composition, entry, tree). All CIMI clinical models (i.e. archetypes) will be defined by constraining the CIMI RM. The RM is intended to be instantiated with patient data which conforms to the constraints defined by the associated clinical model.

Clinical Model Governance

Clinical Model Governance is a set of policies and processes through which the high clinical quality of all clinical artifacts (including clinical models and-or archetypes) is maintained during creation, storage, verification, maintenance, and distribution, by, for, and on behalf of CIMI.

Clinical Model Repository

The Clinical Model Repository is a data store holding clinical information models and associated artifacts in an agreed sharable format.

Clinical Model Verification

Clinical Model Verification is the act of reviewing, inspecting, or testing in order to establish a clinical model specification meets appropriate clinical safety and quality standards.

Clinical Modeling Language

A Clinical Modeling Language is a modeling language defining clinical information models.

Clinical Requirement

Clinical Requirements are requirements articulating clinical needs including clinical practices, standards, guidelines, principles, and other clinical concepts.

Code System

A Code System is a managed collection of uniquely identifiable concepts with associated representations. A code system may also form an ontological system for representing a set of concepts, e.g. SNOMED-CT, LOINC, ICD-10, etc.

Common Terminology Services 2 (CTS2)

CTS2 is an OMG specification providing a standard interface to disparate terminology sources. The Information Model specifies the structural definition, attributes, and associations of resources common to structured terminologies such as Code Systems, Binding Domains, and Value Sets. The Computational Model specifies the service descriptions and interfaces needed to access and maintain structured terminologies.

Concept

In information modeling, a concept represents an “idea” as a word or phrase in order to support human understanding, but may also be represented with a concept identifier in order to bind it to a controlled terminology or ontology.

Concept Domain

A Concept Domain is a named category of like concepts bound to one or more coded elements in an information model. Concept Domains exist to constrain the intent of the coded element and are independent of any specific vocabulary, code system, or Realm. A Concept Domain provides a high level grouping for all things possible in a given domain from which value sets will be constructed.

Concept Domain Binding

A Concept Domain Binding is the association of a value set with a concept domain in a given context.

Conceptual Information Model

A Conceptual Information Model is a representation of real-world objects and their relationships and constraints as understood by domain experts. A conceptual model should include no implementation-specific details.

Conformance

Conformance is the requirement that those who participate in CIMI by contributing data components or creating and sharing ADL artifacts are following the agreed-upon procedures for doing so and that all documentation meets minimum criteria and the CIMI Naming and Design Rules where applicable.

Constraint Model

A Constraint Model is a formal specification used for describing constraints on an Underlying Reference Model. The Constraint Model is used to express clinical information models (i.e. archetypes), not to be confused with the clinical information models that are instances of the constraint model.

Detailed Clinical Model

A Detailed Clinical Model is a relatively small standalone information model designed to express a precise clinical concept in a standardized and reusable manner.

Fully Defined Concept

A Fully Defined Concept is a concept uniquely defined by a set of defining relationships.

Information Model

An Information Model is a structured representation of the information requirements of a domain including the classes of information required and their attributes, relationships, and constraints.

Node

A Node is a named part of an information model.

Ontology

An Ontology is a formal representation of knowledge as a set of concept identifiers, terms describing the concepts so identified, and the relationships among them.

Reference Model

A Reference Model is an information model defining a set of modeling patterns upon which clinical models are defined.

Reference Terminology

A Reference Terminology is a terminology designed to provide common semantics for diverse implementations.

Semantic Binding

Semantic Binding is the association of a node in an information model with a concept from a controlled terminology representing its meaning.

Terminology

A Terminology is a vocabulary of technical terms used in a particular field, subject, science, or art.

Terminology Binding

Terminology Binding is the assertion of a relationship between an information model and a terminology.

Value Binding

Value Binding is the association of a given node in a clinical model with the set of valid concepts that may populate it.

Value Set

A Value Set is a set of concept identifiers deemed valid for use in a specific context, especially to define the domain of a data element.

# Symbols

## Graphical Symbols

No AML-specific graphical symbols are defined in this specification.

## Abbreviations

ADL Archetype Definition Language

AM Archetype Model

AML Archetype Modeling Language

AOM Archetype Object Model

AQL Archetype Query Language

CDA Clinical Document Architecture

CDL Clinical Document Language

CDR Clinical Data Repository

CIM Clinical Information Model

CIMI Clinical Information Modeling Initiative

CMP Constraint Model Profile

CRM Clinical Reference Model

CTS2 Common Terminology Services 2

EHR Electronic Health Record

HL7 Health Level Seven

ICD-10 International Statistical Classification of Diseases and Related Health Problems, 10th Edition

LOINC Logical Observation Identifiers Names and Codes

MDA Model Driven Architecture

OCL Object Constraint Language

OMG Object Management Group

OpenEHR Open Electronic Health Record

PIM Platform Independent Model

PSM Platform Specific Model

RM Reference Model

RMP Reference Model Profile

SNOMED CT Systematized Nomenclature of Medicine – Clinical Terms

TBP Terminology Binding Profile

UML Unified Modeling Language

URI Uniform Resource Identifier

URM Underlying Reference Model

# Additional Information

## Changes to Adopted OMG Specifications

No changes to adopted OMG specifications are required to adopt this specification.

## Acknowledgements

The following companies submitted this specification:

1. Mayo Clinic
2. Visumpoint, LLC

The following companies supported this specification:

1. Escape Velocity, LLC

The following individuals aided the development of this specification:

|  |  |
| --- | --- |
| 1. Thomas Beale | Ocean Informatics |
| 1. Dave Carlson | Intermountain Healthcare (Consultant) |
| 1. Angelique Cortez | Accenture |
| 1. Bob Daniel | Escape Velocity LLC |
| 1. Tom Digre | Visumpoint LLC |
| 1. Stanley M. Huff, MD | Intermountain Healthcare |
| 1. Patrick Langford | Intermountain Healthcare (Consultant) |
| 1. Robert Lario | Visumpoint LLC |
| 1. Jay Lyle | Ockham Information Services LLC |
| 1. Virginia Riehl | Independent Consultant |
| 1. Deepak Kumar Sharma, M.S. | Mayo Clinic |
| 1. Harold Solbrig | Mayo Clinic |
| 1. Mason Tran | Visumpoint LLC |
| 1. Michael van der Zel | University Medical Center Groningen  Results4Care |

# The AOM and the AML Metamodel

Section 6.5.1.5 of the AML RFP states that "Submissions shall provide functional coverage for the full set of capabilities or equivalent, as defined in AOM 1.5 [cite: AOM]. Section 6.1.1 further states that the "RFP seeks a solution that will: ... Support the specification of CIMI content models in UML, such that they can be translated into AOM 1.5." and later says "... a transformation form AML models to semantically equivalent Archetype Object Models (AOM) is strongly desired, but is optional. If not provided, it must be demonstrated that this transformation is possible."

The Archetype Object Model [cite: AOM] version 1.5 was used as a reference for the creation of the corresponding AML Metamodel. While the ideal situation would have been to have used the AOM directly, a number of considerations prevented this from happening, including:

· The AOM doesn't currently exist as a machine readable UML model. It was originally created by reverse engineering the implementation data model of the AML parser and workbench and was subsequently refined directly within FrameMaker

· The AOM focused first on implementability, and only then comprehensibility. As an example, the term\_definitions attribute in the ARCHETYPE\_TERMINOLOGY section is of type "<Hash <Hash <ARCHETYPE\_TERM, String>, String> [1]". The accompanying text states that this is "a mandatory structure consisting of lists of term definitions defined local to the archetype, one list for each language of translation, as well as the original language of definition. The entries ... include: "Some or all id codes...", etc.

While these may be good instructions to an implementer, it doesn't describe the artifacts in terms of a UML model.

· Some of the critical information about the AOM was encoded in text strings. As an example, the ARCHETYPE\_HRID structure on page 17 encodes a number of model artifacts including rm\_package (the reference model package), rm\_class (the reference model class being constrained), major\_version (the identifier for the thing being versioned), minor\_version, version\_status, etc. (identifiers for specific versions of major\_version), etc.

These identifiers represent classes and relationships that need to be expressed as first class model elements if they are to be realized in a profile.

· The AOM didn't formally describe the relationship between the AOM components and the reference model metamodel. The documented alluded to the fact that there were classes, properties and types that were being constrained, but this was not made explicit. AML needed an explicit and formal description of the relationship between the AML model components and the corresponding UML metamodel artifacts being constrained.

The AOM was also very focused on the ADL use case and implementation. The detailed decomposition of an ADL archetype identifier is recorded in the ARCHETYPE\_HRID class. While this accurately reflects ADL specific identifiers, there is nothing inherent in the model that requires that exactly this identifier structure be used. The RFP specifically seeks a solution that is "... capable of being used in other domain areas, with other reference models", and, with this in mind, we chose to relax some relationships and structures that, while necessary for ADL, were not required for archetype modeling in general.

Given the considerations above, we determined that a necessary first step was to produce a *logical* model that faithfully preserved the semantics of the AOM but addressed the various issues described above. The result of this process can be found in Appendix A[[1]](#_ftn1).

[[1]](#_ftnref1) This process also required several changes and simplifications to the AOM 1.5 model. The contents of Appendix A are aligned with the AOM *2.0* version of the model rather than 1.5 as requested in the RFP.

# Profiles

The accurate and timely communication within and across health information systems of hospitals presents an important challenge and opportunity to improving the quality of health care and patient safety. Health information by its very nature is distributed across independent and disparate solutions that are often syntactically or semantically inconsistent. This impedes users from accessing information in a normalized and meaningful manner. The objective of this submission is to provide a standard for modeling Archetype Models (AMs) using UML, to support the representation of Clinical Information Modeling Initiative (CIMI) artifacts in UML.

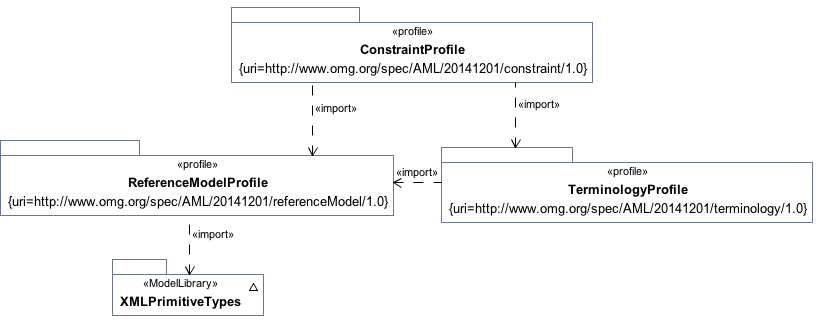
Archetypes are Platform Independent Models (PIMs), which are developed as a set of constraints on a specific Reference Model (RM). These archetypes are detailed and domain-specific definitions of concepts defined in terms of structured and constrained combinations of model elements within a reference model or parent archetype. These models refer to and represent clinical concepts such as heart rate, blood pressure, examination, etc. The ISO EN 13606 and OpenEHR communities define Archetypes utilizing the Archetype Definition Language (ADL).

The following models define the UML Profile “Archetype Modeling Language” (AML). The AML Profile was developed as an aggregation of three sub-profiles, which together meet the requirements of archetype modeling. The three sub-profiles of the AML Profile will include:

* Reference Model Profile (RMP): Enables the specification of reference models, upon which archetypes can be based.
* Constraint Model Profile (CMP): Support the specification of constraints on a given reference model, to enable the development of archetypes, including Clinical Information Models (CIMs).
* A Terminology Binding Profile (TBP): Provides support the binding of information models to terminology, with optional support for binding to CTS2. Terminology bindings will include:

1. Value Bindings: Linkage of the data model to value domains, which restrict the valid value of an attribute to a set of values that corresponds to a set of meanings recorded in an external terminology;
2. Semantic Bindings: Defining the meaning of model elements, using concepts in an external terminology; and
3. Constraint Bindings: Specifying constraints on the information model, using concepts and relationships defined in an external terminology.

## Dependencies



**Dependencies**

## Reference Model Profile

The CIMI modeling methodology utilizes a two level modeling approach. The first level, the reference model, defines the "language" of clinical modeling. A reference model is a UML model that extended using the Reference Model Profile (RMP) as defined in the following sections. A reference model defines the "maximal" set of possible data instances of and models the highest level of abstraction over which information can be usefully exchanged.

An archetype, which is expressed using the Constraint Modeling Profile (CMP) restricts or "constrains" the set of possible reference model data instances.

The Reference Model Profile defines stereotypes that

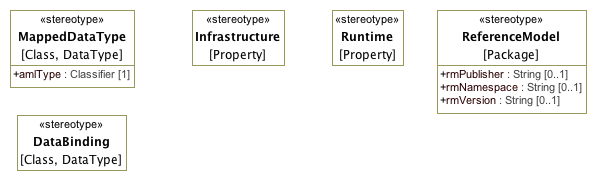
· Identify and enable annotation on the package containing the reference model

· Identify which reference model components cannot be constrained

· Identify the model elements that represent AML data types and, if necessary, identify points where a transformation process is needed.

### Reference Model Stereotypes

The AML Reference Model Profile defines the stereotypes that are used to identify the root package of the target reference model, identify the target reference model properties in the target that are not constrainable and to identify the reference model classifiers that represent built-in AML data types.



**Reference Model Stereotypes**

An AML Archetype Library [ref: AML Archetype Library] defines a set of constraints on a target reference model. The <<ReferenceModel>> stereotype is used to identify the "root" package in the target reference model to which the library constraints apply. The <<Infrastructure>> and <<Runtime>> stereotypes identify properties in the target reference model that cannot be constrained because they represent model specific infrastructure such as archetype identifiers or secondary keys or because they carry information such as provenance and timestamps that is established at runtime. <<MappedDataType>> identifies a target reference model classifier that corresponds to a built-in AML Data Type and <<DataBinding>> represents a "transformation" class -- a class that specializes both an AML Data Type and its corresponding classifier in the target reference model.

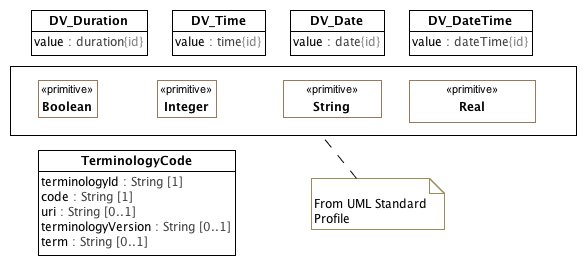
### AML Data Types

This section specifies the set of data types whose values can be constrained in an AML profile. The Constraint Model Profile defines mechanisms to specify constraints on the sizes, ranges and possible values and other characteristics of these data types.

The AML Constraint Model Profile (CMP) defines mechanisms that restricting the possible values that are allowed in a small set of built-in types, which include Boolean, Integer, String, Real, Date, Time, DateTime, Duration and coded values. In order to have a meaningful interpretation with respect to a target Reference Model, a mapping must be provided from the built-in AML Data Types to the corresponding elements in the Reference Model. If the target reference model is already based on the UML String, Integer, Real and Boolean primitive types, no further mapping is required. If, however, the target reference model represents some or all of these types in some other way, it is necessary to be able to (a) identify the reference model elements whose values are directly constrainable in AML and (b) create an intermediate classifier that represents the native AML Data Type *and* its analog in the target reference model.

The CMP also provides mechanisms for defining constraints on Date, Time, DateTime and Duration types, which we refer to collectively as "temporal types". While several models are already available for representing these types [cite: XSDUML, OMG DT, ?], the majority of the reference models that exist today either define their own temporal types or reference the HL7 Version 3 Data Types[cite: HL7V3dt] or the ISO 21090 Healthcare Data types [cite: ISO21090]. The AML Constraint Model Profile uses the XML Schema Primitive Types [cite: XSD Schema 1.1] as defined in ? [cite:?] as the "native" temporal types. It has, however, packaged these as values in classes to allow constraints to be expressed as restrictions on classes rather than data types.

The Constraint Model Profile also allows constraints on the Duration, Date, Time and DateTime data types as well as on "value sets" -- the allowable terminology codes that can be referenced by an enumeration or a coded field. If there are corresponding instances of any of these types in the reference model, it will be necessary to identify them and associate them with the corresponding primitive type in the Constraint Model Profile.



**AML Data Types**

The above diagram contains the list of classifiers whose values can be constrained using the AML profile. These types include the basic Boolean, Integer, String and Real types as well as Duration, Date, Time and DateTime. The AML Terminology Model Profile provides the ability to describe the set of possible value codes that are allowed for certain reference model attributes.

### Profile Elements

#### <Class> DV\_Date

**Description**

Represents an absolute point in time, as measured on the Gregorian calendar, and specified only to the day. [cite: SUPPORT\_IM, p24]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedDate](#_2a37a0f4cd69c5ee1a32f787b23de875)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::date**

ISO8601 string for date, in format **YYYYMMDD** or **YYYY-MM-DD**, or a partial invariant.

#### <Class> DV\_DateTime

**Description**

Represents an absolute point in time, specified to the second. [cite: SUPPORT\_IM, p27]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::dateTime**

A valid ISO8601 string for date/time, in format

**YYYYMMDDThhmmss[,sss][Z | ±hh[mm]]** or in extended format **YYYY-MM-DDThh:mm:ss[,sss][Z | ±hh[mm]]** or a partial variant.

Note that this class includes 2 deviations from ISO 8601:2004:

• for partial date/times, any part of the date/time up to the month may be missing, not just seconds and minutes as in the standard;

• the time 24:00:00 is not allowed, since it would mean the date was really on the next day.

#### <Class> DV\_Duration

**Description**

Represents a period of time corresponding to a difference between two time-points. See: ISO8601\_DURATION [cite: SUPPORT\_IM, p30]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::duration**

Value is a valid ISO 8601 duration, i.e. takes the form:

• **P[nnY][nnM][nnW][nnD][T[nnH][nnM][nnS]]** Where each nn represents a number of years, months, etc. nnW represents a number of 7- day weeks.

Note: allowing the **W** designator in the same expression as other designators is an exception to the published standard, but necessary in clinical information (typically for representing pregnancy duration).

#### <Class> DV\_Time

**Description**

Represents an absolute point in time from an origin usually interpreted as meaning the start of the current day, specified to the second. [cite: SUPPORT\_IM, p25]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedTime](#_6395ddb1d0c31197b0c966c1281ec5aa)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::time**

Value representation is ISO8601 string for time, i.e. in form: **hhmmss[,sss][Z|±hh[mm]]** or the extended form: **hh:mm:ss[,sss][Z|±hh[mm]]**, or a partial invariant.

A small deviation to the ISO 8601:2004 standard in this class is that the time 24:00:00 is not allowed, for consistency with **DateTime**.

#### <Class> TerminologyCode

**Description**

Represents a scoped identifier that is unique within the context of a given namespace. While a terminology code is treated as data type from the AML perspective, there multiple ways of representing the same "value" including:

* terminologyId + code -- a combination of the URI of a scoping namespace (terminologyId) and identifier (code)
* uri -- a URI that embodies both the namespace and the code
* terminologyVersion -- the URI of a terminology (code system, ontology) that contains a description of what the code represents.
* term -- a language specific string that provides a hint at the intended meaning of the code

Note that "terminologyId" and "terminologyVersion" represent very different constructs. "terminologyId" represents a namespace, not an ontology / code system. "terminologyVersion" references an collection of code descriptions.

**TerminologyCode** represents the contents of either an **EnumerationLiteral** or a **PermissibleValue**. It corresponds to the AOM **Terminology\_Code**, and its function is to gather sufficient information that, in combination with the KnownNamespace Enumeration extension, it will be possible to populate a variety of target coded types, including the ISO 21090 CD type, the openEHR DV\_CODED\_TEXT, the CIMI CODED\_TEXT, etc.

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + terminologyId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The *name* or *uri* of the *namespace* that scopes *code.*

Either the *name* of the **Enumeration** that owns the **EnumerationLiteral** represented by the *code* attribute or the *uri* tag of the **Enumeration** if it is extended by the **ScopedIdentifier** stereotype and the tag is present.

**• <property> ::= + code : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The representation of an **EnumerationLiteral.** This will usually be the name of the **EnumerationLiteral**, but it should be possible for implementations to associate other numeric or string values with literals through either the *slot* or *id* of an **IdentifiedItem** extension that will become the code value instead.

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The URI of a **ConceptReference**. This should be populated in the case where the selected value is a **PermissibleValue** and either (a) the owning **Enumeration** is stereotyped by **ScopedIdentifier** and it has a *uriPattern* attribute or (b) the **PermissibleValue** includes a *meaning* tag of type **ConceptReference**, in which case *uri* equals the *uri* of the **ConceptReference**.

If two TerminologyCodes have the same *uri*, they are considered to be identical. If a uri is absent in one or both TerminologyCodes, they are considered to be identical only if the *terminologyId* and *code* are the same.

**• <property> ::= + terminologyVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The uri of a CodeSystem or CodeSystemVersion that carried a description of the particular code at some point in time. This attribute is strictly informative and should is not part of the identity of the terminology code.

If the source **EnumerationLiteral** is stereotyped by **PermissibleValue**, the meaning tag is present and the **ConceptReference** in the meaning tag has a **CodeSystemAndVersion**, this is set to:

* The *uri* of the **CodeSystemVersionReference** if it is present
* The *uri* of the **CodeSystemReference** otherwise.

**• <property> ::= + term : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A string designating the intended target of the **TerminologyCode** in a given language and context. This is strictly informative and is not part of the part of the identity of a **TerminologyCode** as it may change as data moves through different languages or contexts.

#### <Stereotype> DataBinding

**Description**

The **DataBinding** stereotype identifies a "bridge" class providing a mapping between an internal AML data type identified by the **AMLDataType** stereotype and a corresponding Reference Model **DataType** or **Class** identified by the **MappedDataType** stereotype.

The Class or DataType extended by a DataBinding functions as the base point for AML constraints.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Constraints**

* **oneAMLDataType1**

This Class must specialized an <<AMLDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **oneAMLDataType2**

This DataType must specialize an <<AMLDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **atLeastOneMappedDataType1**

This Class must be stereotyped by <<MappedDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->exists(x|x.stereotypedBy('MappedDataType'))

* **atLeastOneMappedDataType2**

This DataType must be specialized from <<MappedDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->exists(x|x.stereotypedBy('MappedDataType'))

* **amlTypeMatch**

[English]

A data binding should specialize exactly two classifiers, one that is stereotyped by MappedDataType and a second that must be a descendant of UML Boolean, UML Integer, UML String, UML Real, Date, Time, DateTime, Duration or TerminologyCode. Furthermore, the amlType of the classifier stereotyped by MappedDataType must reference the corresponding classifier of the other generalization.

#### <Stereotype> Infrastructure

**Description**

A stereotype indicating a base Property represents an aspect of an Archetype implementation such as a specific Archetype identifier or other element. Properties with the Infrastructure stereotype cannot be constrained in AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

#### <Stereotype> MappedDataType

**Description**

Extends a Class or DataType in the Reference Model and indicates the base Class or DataType represents the classifier referenced by the *amlType* tag and is to be treated as an atomic "primitive" type that cannot be further decomposed.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + amlType : UML Standard Profile::UML2 Metamodel::Classifier**

The assumed or built-in AML Data Type corresponding to the extended Class or DataType.

**Constraints**

* **isAMLDataType**

The amlType must reference a classifier (Class or DataType) that has a AMLDataType stereotype.

[OCL]

self.amlType.stereotypedBy('AMLDataType')

* **amltypematch**

#### <Stereotype> ReferenceModel

**Description**

This stereotype identifies a package as a Reference Model -- a package which contains the collection of UML Classes that can be constrained by the Archetypes in an Archetype Library. The Reference Model stereotype also allows the specification of the publisher, namespace and version of a Reference Model in a form compatible with a modeling language such as ADL.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + rmPublisher : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The name of the Reference Model publisher. Corresponds to *rm\_publisher* in AOM 2.0

**• <property> ::= + rmNamespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The owning domain name of the archetype. Corresponds to the *namespace* attribute in AOM2.0.

**• <property> ::= + rmVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

Designates the version id of the reference model on which the archetype is based. Corresponds to *rm\_release* in AOM 2.0

#### <Stereotype> Runtime

**Description**

Indicates that the base Property represents a dynamic or "runtime" element that cannot be constrained using AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

## Terminology Profile

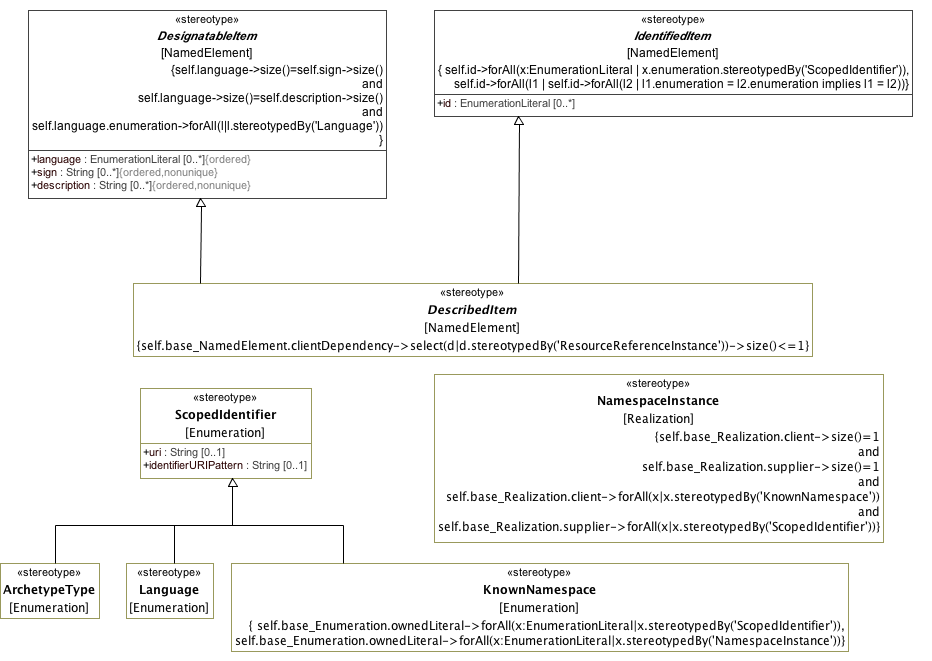
The AML Terminology Profile is the UML equivalent of the ADL 2.0 terminology section, including:

* Identifiers -- "id", "at" and "ac" identifiers may be assigned to Class constraints, EnumerationLiterals and Enumerations respectively.
* Term definitions -- multilingual designation ("text") / description tuples may be assigned to any named AML model artifact
* Term bindings
  + Class constraints may be associated with a reference to an external resource that the constraint is "about"
  + Enumeration constraints may be associated with a reference to an external value set constraining the set of possible values and value meanings
  + TerminologyCode (Permissible) values may be associated with "concept" (aka. class, category, term) references providing the meaning for the value
* Value Sets -- local enumerations may be defined that associate collections of individual codes (ADL "at" codes) with a local value set or enumeration (ADL "ac" code).

The Terminology Binding profile draws on the ISO 11179-3 model for the identification, designation, definition and value / meaning binding aspects and on the OMG Common Terminology Services 2 (CTS2) specification for the model of Concept, Code System, Code System Version, Value Set and Value Set Definition references.

### Identification and Designation

The first section in the AML Terminology Profile focuses on resource identifiers, the equivalent of which in the ADL/AOM 2.0 specification would be the "id", "at" and "ac" codes, their namespaces and their multilingual designations and descriptions.



**IdentificationAndDesignation**

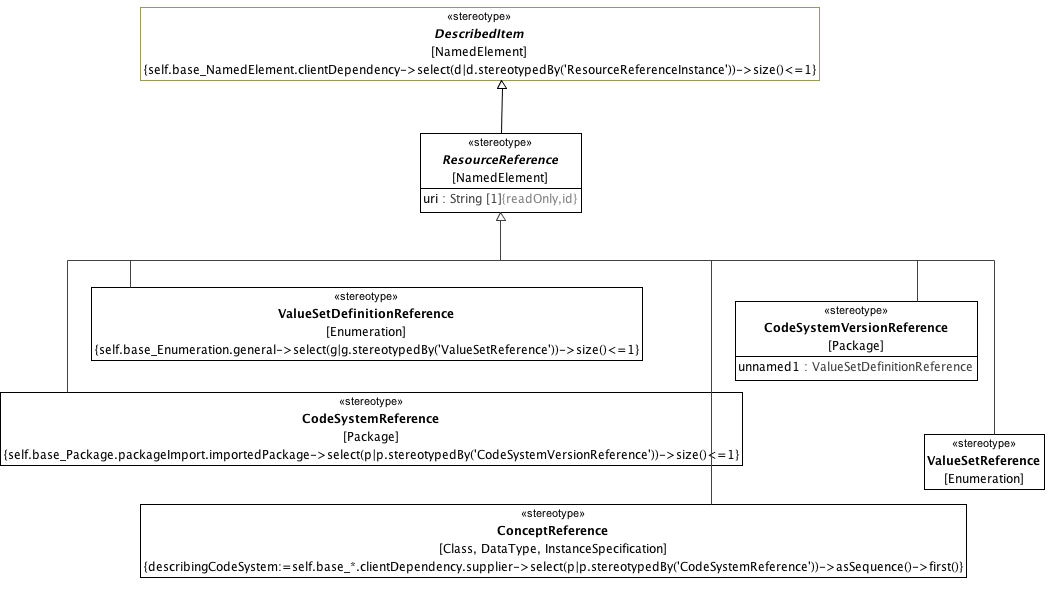
### Resource References

This clause describes the profiles corresponding to the MetaModel constructs:

* ResourceReference
* CodeSystemAndVersionReference
* ValueSetAndDefinitionReference

The first clause below defines how each of these elements are represented. The second clause defines a set of extensions to the Abstraction relationship providing links between model elements and their target ResourceReferences.

#### Resource References

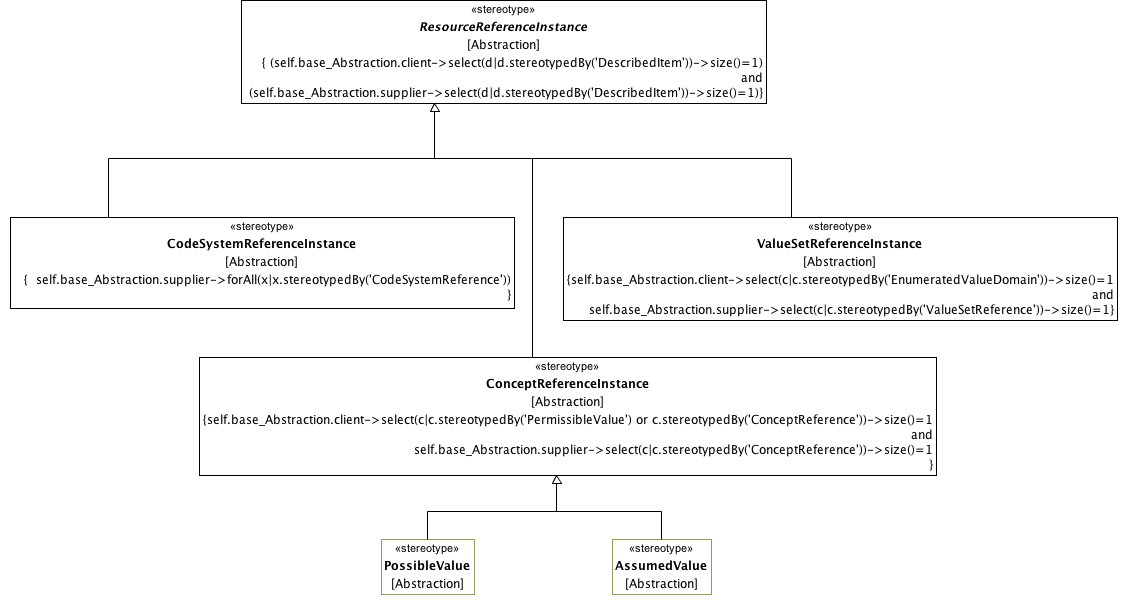


**Resource References**

A URI naming the resource. This should conform to the semantics of the CTS2 **PersistentURI**.

#### Resource Reference Relationships

This section describes the set of associations that allow the various types of ResourceReference to be linked to their referencing elements.

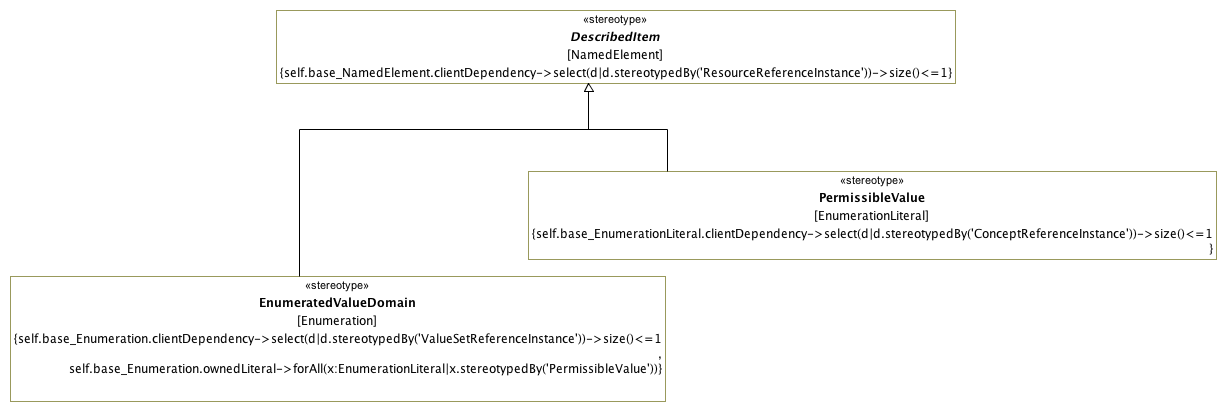


**Resource Reference Relationships**

A URI that names the resource. This should conform to the semantics of the CTS2 **PersistentURI**.

### Enumerated Value Domains

This clause defines the required AML extensions to Enumeration and EnumerationLiteral that represent the EnumeratedValueDomain and PermissibleValue elements respectively.



**Enumerated Value Domains**

### Profile Elements

#### <Class> DV\_Date

**Description**

Represents an absolute point in time, as measured on the Gregorian calendar, and specified only to the day. [cite: SUPPORT\_IM, p24]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedDate](#_2a37a0f4cd69c5ee1a32f787b23de875)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::date**

ISO8601 string for date, in format **YYYYMMDD** or **YYYY-MM-DD**, or a partial invariant.

#### <Class> DV\_DateTime

**Description**

Represents an absolute point in time, specified to the second. [cite: SUPPORT\_IM, p27]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::dateTime**

A valid ISO8601 string for date/time, in format

**YYYYMMDDThhmmss[,sss][Z | ±hh[mm]]** or in extended format **YYYY-MM-DDThh:mm:ss[,sss][Z | ±hh[mm]]** or a partial variant.

Note that this class includes 2 deviations from ISO 8601:2004:

• for partial date/times, any part of the date/time up to the month may be missing, not just seconds and minutes as in the standard;

• the time 24:00:00 is not allowed, since it would mean the date was really on the next day.

#### <Class> DV\_Duration

**Description**

Represents a period of time corresponding to a difference between two time-points. See: ISO8601\_DURATION [cite: SUPPORT\_IM, p30]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::duration**

Value is a valid ISO 8601 duration, i.e. takes the form:

• **P[nnY][nnM][nnW][nnD][T[nnH][nnM][nnS]]** Where each nn represents a number of years, months, etc. nnW represents a number of 7- day weeks.

Note: allowing the **W** designator in the same expression as other designators is an exception to the published standard, but necessary in clinical information (typically for representing pregnancy duration).

#### <Class> DV\_Time

**Description**

Represents an absolute point in time from an origin usually interpreted as meaning the start of the current day, specified to the second. [cite: SUPPORT\_IM, p25]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedTime](#_6395ddb1d0c31197b0c966c1281ec5aa)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::time**

Value representation is ISO8601 string for time, i.e. in form: **hhmmss[,sss][Z|±hh[mm]]** or the extended form: **hh:mm:ss[,sss][Z|±hh[mm]]**, or a partial invariant.

A small deviation to the ISO 8601:2004 standard in this class is that the time 24:00:00 is not allowed, for consistency with **DateTime**.

#### <Class> TerminologyCode

**Description**

Represents a scoped identifier that is unique within the context of a given namespace. While a terminology code is treated as data type from the AML perspective, there multiple ways of representing the same "value" including:

* terminologyId + code -- a combination of the URI of a scoping namespace (terminologyId) and identifier (code)
* uri -- a URI that embodies both the namespace and the code
* terminologyVersion -- the URI of a terminology (code system, ontology) that contains a description of what the code represents.
* term -- a language specific string that provides a hint at the intended meaning of the code

Note that "terminologyId" and "terminologyVersion" represent very different constructs. "terminologyId" represents a namespace, not an ontology / code system. "terminologyVersion" references an collection of code descriptions.

**TerminologyCode** represents the contents of either an **EnumerationLiteral** or a **PermissibleValue**. It corresponds to the AOM **Terminology\_Code**, and its function is to gather sufficient information that, in combination with the KnownNamespace Enumeration extension, it will be possible to populate a variety of target coded types, including the ISO 21090 CD type, the openEHR DV\_CODED\_TEXT, the CIMI CODED\_TEXT, etc.

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + terminologyId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The *name* or *uri* of the *namespace* that scopes *code.*

Either the *name* of the **Enumeration** that owns the **EnumerationLiteral** represented by the *code* attribute or the *uri* tag of the **Enumeration** if it is extended by the **ScopedIdentifier** stereotype and the tag is present.

**• <property> ::= + code : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The representation of an **EnumerationLiteral.** This will usually be the name of the **EnumerationLiteral**, but it should be possible for implementations to associate other numeric or string values with literals through either the *slot* or *id* of an **IdentifiedItem** extension that will become the code value instead.

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The URI of a **ConceptReference**. This should be populated in the case where the selected value is a **PermissibleValue** and either (a) the owning **Enumeration** is stereotyped by **ScopedIdentifier** and it has a *uriPattern* attribute or (b) the **PermissibleValue** includes a *meaning* tag of type **ConceptReference**, in which case *uri* equals the *uri* of the **ConceptReference**.

If two TerminologyCodes have the same *uri*, they are considered to be identical. If a uri is absent in one or both TerminologyCodes, they are considered to be identical only if the *terminologyId* and *code* are the same.

**• <property> ::= + terminologyVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The uri of a CodeSystem or CodeSystemVersion that carried a description of the particular code at some point in time. This attribute is strictly informative and should is not part of the identity of the terminology code.

If the source **EnumerationLiteral** is stereotyped by **PermissibleValue**, the meaning tag is present and the **ConceptReference** in the meaning tag has a **CodeSystemAndVersion**, this is set to:

* The *uri* of the **CodeSystemVersionReference** if it is present
* The *uri* of the **CodeSystemReference** otherwise.

**• <property> ::= + term : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A string designating the intended target of the **TerminologyCode** in a given language and context. This is strictly informative and is not part of the part of the identity of a **TerminologyCode** as it may change as data moves through different languages or contexts.

#### <Stereotype> ArchetypeType

**Description**

An implementation specific enumeration of possible archetype types. In ADL, this would extend an enumeration that includes literals for *archetype*, *template, template\_overlay* and *operational\_template* and their flattened equivalents.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

#### <Stereotype> AssumedValue

**Description**

An instance of a link between a **ConceptReferenceConstraint** and an assumedValue.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1)

#### <Stereotype> CodeSystemReference

**Description**

A ResourceReference whose *uri* identifies to a code system (aka. "concept system, "terminology" or "ontology"). Note, while the UML **Package** is the base for the **CodeSystemReference**, its intent is to represent a the *uri* and optional name of a code system and a single, optional *version* and should contain nothing else. The optional *version* of type **CodeSystemVersionReference** is represented as a *packageImport*.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **version**

There must be at most 1 imported <<CodeSystemVersionReference>> Package.

[Binary]

self.base\_Package.packageImport.importedPackage->select(p|p.stereotypedBy('CodeSystemVersionReference'))->size()<=1

#### <Stereotype> CodeSystemReferenceInstance

**Description**

**CodeSystemReferenceInstance** represents the optional link between a **ConceptReference** and a describing code system and optional version.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Constraints**

* **isCodeSystemReference**

Must be a valid instance of a CodeSystemReference.

[OCL]

self.base\_Abstraction.supplier->forAll(x|x.stereotypedBy('CodeSystemReference'))

#### <Stereotype> CodeSystemVersionReference

**Description**

A URI referencing a specific version of a code system. Note, while UML **Package** is used as the base for the **CodeSystemVersionReference** stereotype, a **Package** extended by **CodeSystemVersionReference** may not have any members. It is simply the name of a code system version and not the actual entity itself.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Attributes**

**• <property> ::= + unnamed1 : TerminologyProfile::ValueSetDefinitionReference**

#### <Stereotype> ConceptReference

**Description**

A URI uniquely identifing a "concept" (aka. class, entity, individual or, in some contexts "term"), accompanied by additional information conveying the intended meaning, code and source of the information used to determine the intent of the URI.

**ConceptReference** can be used to bind a **Class** or **DataType** constraint to its terminology binding (aka "meaning" in ISO 11179-3). **ConceptReference** may also be associated with a describing code system and optional code system version by a usage relationship whose client is **ConceptReference** and whose supplier is **CodeSystemReference**. The purpose of this association is to identify the particular description that was used when selecting the concept reference.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **describingCodeSystem**

Definition of describingCodeSystem

describingCodeSystem:=self.base\_\*.clientDependency.supplier->select(p|p.stereotypedBy('CodeSystemReference'))->asSequence()->first()

[English]

describingCodeSystem:=self.base\_\*.clientDependency.supplier->select(p|p.stereotypedBy('CodeSystemReference'))->asSequence()->first()

#### <Stereotype> ConceptReferenceInstance

**Description**

A **ConceptReferenceInstance** associates a referencing element with a corresponding **ConceptReference**. It is used to link a **PermissibleValue** with a value meaning, an **ObjectConstraint** with what the constraint is "about" (aka. "meaning in ISO 11179) and to identify the possible and assumed values for a **ConceptReferenceConstraint**.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Direct Subclasses (Specialization)**

[AssumedValue](#_7c76bb5bbde4643a957c898bfc8af67a),

[PossibleValue](#_ef85d33c740061c0c756c4e535c34ccc)

**Constraints**

* **isConceptReference**

There must be one <<PermissibleValue>> or <<ConceptReference>> client and one <<ConceptReference>> supplier.

[OCL]

self.base\_Abstraction.client->select(c|c.stereotypedBy('PermissibleValue') or c.stereotypedBy('ConceptReference'))->size()=1 and self.base\_Abstraction.supplier->select(c|c.stereotypedBy('ConceptReference'))->size()=1

#### <Stereotype> DataBinding

**Description**

The **DataBinding** stereotype identifies a "bridge" class providing a mapping between an internal AML data type identified by the **AMLDataType** stereotype and a corresponding Reference Model **DataType** or **Class** identified by the **MappedDataType** stereotype.

The Class or DataType extended by a DataBinding functions as the base point for AML constraints.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Constraints**

* **oneAMLDataType1**

This Class must specialized an <<AMLDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **oneAMLDataType2**

This DataType must specialize an <<AMLDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **atLeastOneMappedDataType1**

This Class must be stereotyped by <<MappedDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->exists(x|x.stereotypedBy('MappedDataType'))

* **atLeastOneMappedDataType2**

This DataType must be specialized from <<MappedDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->exists(x|x.stereotypedBy('MappedDataType'))

* **amlTypeMatch**

[English]

A data binding should specialize exactly two classifiers, one that is stereotyped by MappedDataType and a second that must be a descendant of UML Boolean, UML Integer, UML String, UML Real, Date, Time, DateTime, Duration or TerminologyCode. Furthermore, the amlType of the classifier stereotyped by MappedDataType must reference the corresponding classifier of the other generalization.

#### <Stereotype> DescribedItem

**Description**

The **DescribedItem** stereotype allows multilingual designations/descriptions and scoped identifiers to be assigned a **NamedElement**.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e),

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb),

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77),

[IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Subclasses (Specialization)**

[EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a),

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2),

[PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692),

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **validAbout**

There must be a maximum of one <<ResourceReferenceInstance>> Abstraction

[OCL]

self.base\_NamedElement.clientDependency->select(d|d.stereotypedBy('ResourceReferenceInstance'))->size()<=1

#### <Stereotype> DesignatableItem

**Description**

The **DesignatableItem** stereotype allows a set of language / sign / description tuples to be assigned to a **NamedElement**. This represents a flattened version of the **ItemDescription** in the AML Meta Model.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff),

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

**• <property> ::= + language : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

A reference to a spoken or written language. *language* must be an **EnumerationLiteral** in an **Enumeration** stereotyped by the **Language** stereotype. Only one *sign*/description tuple is allowed per language.

**• <property> ::= + sign : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A string designating or "signifying" the name of the *NamedElement* in the corresponding *language. sign* is a required field and cannot contain an empty String ("").

**• <property> ::= + description : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A definition or description of the **NamedElement** in the corresponding *language*. *description* is optional and its absence is indicated by the empty ("") String.

**Constraints**

* **language sign description tuples are designations**

The ordered sequences language, sign, description must be same length and language must be in a <<Language>>.

[OCL]

self.language->size()=self.sign->size() and self.language->size()=self.description->size() and self.language.enumeration->forAll(l|l.stereotypedBy('Language'))

#### <Stereotype> EnumeratedValueDomain

**Description**

An **EnumeratedValueDomain** represents a discrete set of possible values (**PermissibleValues**) for a particular field or data element. Each **PermissibleValue** identifies a unique value and (optionally) its intended meaning. An **EnumeratedValueDomain** may be associated with a scoped identifier, which, in the ADL case would be an "ac" code. The set of **PermissibleValues** for the **EnumeratedValueDomain** would represent an ADL value\_set, which would bind an "ac" code to a set of "at" codes.

An **EnumeratedValueDomain** may also reference a **ValueSetReference** using the **ValueSetReferenceInstance** relationship which would resolve to a set of **PermissibleValues**.

**Diagrams**

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Constraints**

* **meaningIsConceptReference**

There is a maximum of one <<ValueSetReferenceInstance>> Abstractions.

[OCL]

self.base\_Enumeration.clientDependency->select(d|d.stereotypedBy('ValueSetReferenceInstance'))->size()<=1

* **permissibleValues**

All instances must be type permissibleValue

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('PermissibleValue'))

#### <Stereotype> IdentifiedItem

**Description**

The **IdentifiedItem** stereotype allows the assignment of one or more optional identifiers to be assigned to a **NamedElement**.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

**• <property> ::= + id : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

An identifier unique within the context of the containing **Enumeration**.

**Constraints**

* **scopedIdentifierLiteral**

Every *id* property is an instance of a ScopedIdentifier.

[OCL]

self.id->forAll(x:EnumerationLiteral | x.enumeration.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.

[OCL]

self.id->forAll(l1 | self.id->forAll(l2 | l1.enumeration = l2.enumeration implies l1 = l2))

#### <Stereotype> Infrastructure

**Description**

A stereotype indicating a base Property represents an aspect of an Archetype implementation such as a specific Archetype identifier or other element. Properties with the Infrastructure stereotype cannot be constrained in AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

#### <Stereotype> KnownNamespace

**Description**

**KnownNamespace** extends the **Enumeration** listing the set of namespace identifiers recognized by a particular implementation or archetype library. The name of a member literal represents a local namespace identifier. Each local identifier can, in turn, be realized by a second namespace. As an example, an AML model might have 3 known namespaces:

1. "SCT"
2. "SNOMED-CT"
3. "LOINC"

The first two namespaces would reference the same realization target, a **ScopedIdentifier** for the SNOMED CT identifiers referenced by the model, while the third would reference another **ScopedIdentifier** **Enumeration** containing the LOINC identifiers referenced by the model.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Constraints**

* **hasScopedIdentifiers**

Set if knownNamespace has an identifier uniquely referencing its scoping namespace.

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('ScopedIdentifier'))

* **namespaceInstances**

All owned EnumerationLiterals must be <<NamespaceInstance>>

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('NamespaceInstance'))

#### <Stereotype> Language

**Description**

The set of languages referenced by an AML instance. Typically these will be drawn from ISO 639-1 or one of its derivatives.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

#### <Stereotype> MappedDataType

**Description**

Extends a Class or DataType in the Reference Model and indicates the base Class or DataType represents the classifier referenced by the *amlType* tag and is to be treated as an atomic "primitive" type that cannot be further decomposed.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + amlType : UML Standard Profile::UML2 Metamodel::Classifier**

The assumed or built-in AML Data Type corresponding to the extended Class or DataType.

**Constraints**

* **isAMLDataType**

The amlType must reference a classifier (Class or DataType) that has a AMLDataType stereotype.

[OCL]

self.amlType.stereotypedBy('AMLDataType')

* **amltypematch**

#### <Stereotype> NamespaceInstance

**Description**

NamespaceInstance is an instance of Namespace.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Constraints**

* **mustBeScopedIdentifier**

There must be a single <<KnownNamespace>> client and a single <<ScopedIdentifier>> supplier.

[OCL]

self.base\_Realization.client->size()=1 and self.base\_Realization.supplier->size()=1 and self.base\_Realization.client->forAll(x|x.stereotypedBy('KnownNamespace')) and self.base\_Realization.supplier->forAll(x|x.stereotypedBy('ScopedIdentifier'))

#### <Stereotype> PermissibleValue

**Description**

A unique within the context of the owning **EnumeratedValueDomain**. A **PermissibleValue** can be associated with an identifier, which, in ADL would be an "at" code. A permissible value can be associated with one or more language specific designations and optional descriptions, which represent the ADL term\_definitions section (as applied to "at" codes). A **PermissibleValue** may also reference a **ConceptReference** via the **ConceptReferenceInstance** association,where the **ConceptReference** identifies the intended meaning for the value.

**Diagrams**

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Constraints**

* **valueMeaningIsConceptReference**

PermissibleValue.about must be a concept reference

[OCL]

self.base\_EnumerationLiteral.clientDependency->select(d|d.stereotypedBy('ConceptReferenceInstance'))->size()<=1

#### <Stereotype> PossibleValue

**Description**

An instance of a link between a **ConceptReferenceConstraint** and a set of possible values.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1)

#### <Stereotype> ReferenceModel

**Description**

This stereotype identifies a package as a Reference Model -- a package which contains the collection of UML Classes that can be constrained by the Archetypes in an Archetype Library. The Reference Model stereotype also allows the specification of the publisher, namespace and version of a Reference Model in a form compatible with a modeling language such as ADL.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + rmPublisher : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The name of the Reference Model publisher. Corresponds to *rm\_publisher* in AOM 2.0

**• <property> ::= + rmNamespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The owning domain name of the archetype. Corresponds to the *namespace* attribute in AOM2.0.

**• <property> ::= + rmVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

Designates the version id of the reference model on which the archetype is based. Corresponds to *rm\_release* in AOM 2.0

#### <Stereotype> ResourceReference

**Description**

A ResourceReference couples a NamedElement with a uri referencing an external class, category, individual or "concept". It should be noted the uri in ResourceReference has the semantics associated with the PersistentURI in the CTS2 specification -- it is not intended to reference a resource directly, but to "name" a resource that has a description in one or more terminologies, (code systems, classification systems, ontologies).

The uri serves as the identity of a ResourceReference. It may be accompanied by additional language specific designations and descriptions as well as by a ScopedIdentifer that identifies the target as a namespace/name tuple.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[DesignatableItem](#_80448b03d480bba05b1e156796878f77)

**Direct Subclasses (Specialization)**

[CodeSystemReference](#_7de70b4fbd9e6a164f7f00cde47dfd5a),

[CodeSystemVersionReference](#_9a8de95c38ebe2d6ce506cbc9bef7b7a),

[ConceptReference](#_57ae94153b82f28889d42ad4aa8fe1e0),

[ValueSetDefinitionReference](#_a4fedb7858ead8d2272640d51b53719a),

[ValueSetReference](#_1a1ca20b54028ee5e2eb20af35411f6e)

**Attributes**

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

URI of the resource.

**Constraints**

* **uniqueId**

Every identifier must come from a different namespace

[OCL]

self.id->forAll(i1 | self.id->forAll(i2 | i1.enumeration = i2.enumeration implies i1 = i2))

* **language sign description represent tuple**

The ordered sequences language, sign, description must be the same length and language must be part of a <<Language>> Enumeration.

[OCL]

self.language->size()=self.sign->size() and self.language->size()=self.description->size() and self.language.enumeration->forAll(l|l.stereotypedBy('Language'))

#### <Stereotype> ResourceReferenceInstance

**Description**

ResourceReferenceInstance is an abstraction that associates a NamedElement that is stereotyped by DescribedItem with an instance of its reference.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77),

[IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Subclasses (Specialization)**

[CodeSystemReferenceInstance](#_e055a6cce06d0838055b62dbfbf235f2),

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1),

[ValueSetReferenceInstance](#_f3184cb0f8e704f5122c5e97fb9f130c)

**Constraints**

* **isResourceReference**

Must be an instance of a resource reference class

[OCL]

(self.base\_Abstraction.client->select(d|d.stereotypedBy('DescribedItem'))->size()=1) and (self.base\_Abstraction.supplier->select(d|d.stereotypedBy('DescribedItem'))->size()=1)

#### <Stereotype> Runtime

**Description**

Indicates that the base Property represents a dynamic or "runtime" element that cannot be constrained using AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

#### <Stereotype> ScopedIdentifier

**Description**

A **ScopedIdentifier** is an enumeration that may include a uri referencing the scope of the identifier and a uri pattern describing how uri's are constructed from the member EnumerationLiterals.

As an example, the SNOMED CT identifier namespace might have a *uri* of "http://snomed.info/sct", which identifies the sole namespace and an *identifierURIPattern* of "http://snomed.info/id/$1" which indicates an EnumerationLiteral named 74400008 would be represented as "http://snomed.info/id/74400008".

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[ArchetypeType](#_7dc1530ae1ef855ecc3eb9bd5b555a14),

[KnownNamespace](#_b9f78b93edc24bb3301ba69a57e4afc3),

[Language](#_446e0591a4f825b22cd9e573c1239a72)

**Attributes**

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A URI referencing the namespace associated with the ScopedIdentifier. Examples: <http://snomed.info/sct> http://loinc.org

**• <property> ::= + identifierURIPattern : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A URI substitution pattern, where "$1" indicates where the name of an owned EnumerationLiteral would be substituted to create a URI. Example: http://loinc.org/id/$1.

#### <Stereotype> ValueSetDefinitionReference

**Description**

A *uri* that names the value set that the extended **Enumeration** represents. The set of **EnumerationLiterals** in an **Enumeration** stereotyped by **ValueSetDefinitionReference** represent the set or a subset of the **ConceptReferences** returned by the interpretation of the referenced definition.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **specializesValueSet**

A ValueSetDefinitionReference is a specialization of a ValueSetReference.

[OCL]

self.base\_Enumeration.general->select(g|g.stereotypedBy('ValueSetReference'))->size()<=1

#### <Stereotype> ValueSetReference

**Description**

A *uri* that names the value set that the extended Enumeration represents. The set of EnumerationLiterals in an Enumeration stereotyped by ValueSetReference represent the set or a subset of the ConceptReferences returned by the "current" (in the CTS2 sense) definition of the ValueSet.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **definition**

This Enumeration must have exactly one <<ValueSetDefinitionReference>> Generalization.

[OCL]

self.base\_Enumeration.general->select(g|g.stereotypedBy('ValueSetDefinitionReference'))->size()<=1

#### <Stereotype> ValueSetReferenceInstance

**Description**

A link between a **ConceptReferenceConstraint** and a value set and optional definition, where the value set resolves to a set of **ConceptReferences**.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Constraints**

* **isValueSetReference**

Must have one <<EnumeratedValueDomain>> client and one <<ValueSetReference>> supplier.

[OCL]

self.base\_Abstraction.client->select(c|c.stereotypedBy('EnumeratedValueDomain'))->size()=1 and self.base\_Abstraction.supplier->select(c|c.stereotypedBy('ValueSetReference'))->size()=1

## Constraint Profile

The constraint profile defines the modeling elements that may be applied to a reference model and archetype. These elements “constrain” the target model narrowing the semantics and syntax. These elements are drawn from ADL 2.0 and ISO 13606.

### Archetypes

This clause defines the profiles for Archetypes, Archetype Libraries and their identification, provenance and workflow. It corresponds to the *archetype*, *language*, and *description* sections of ADL 2.0.

#### Archetype Libraries

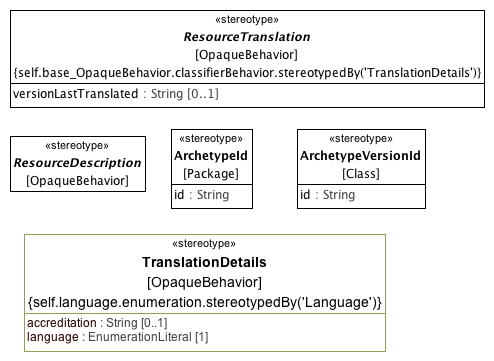
This clause defines the stereotypes that represent **Archetype**, **ArchetypeLibrary** and **ArchetypeVersion** as well as defining the link to the **AuthoredResource** metadata.



**Archetypes**

#### Archetype Metadata

This clause identifies the stereotypes used to connect an **AuthoredResource** to metadata describing translations, ownership, copyright, state, etc.



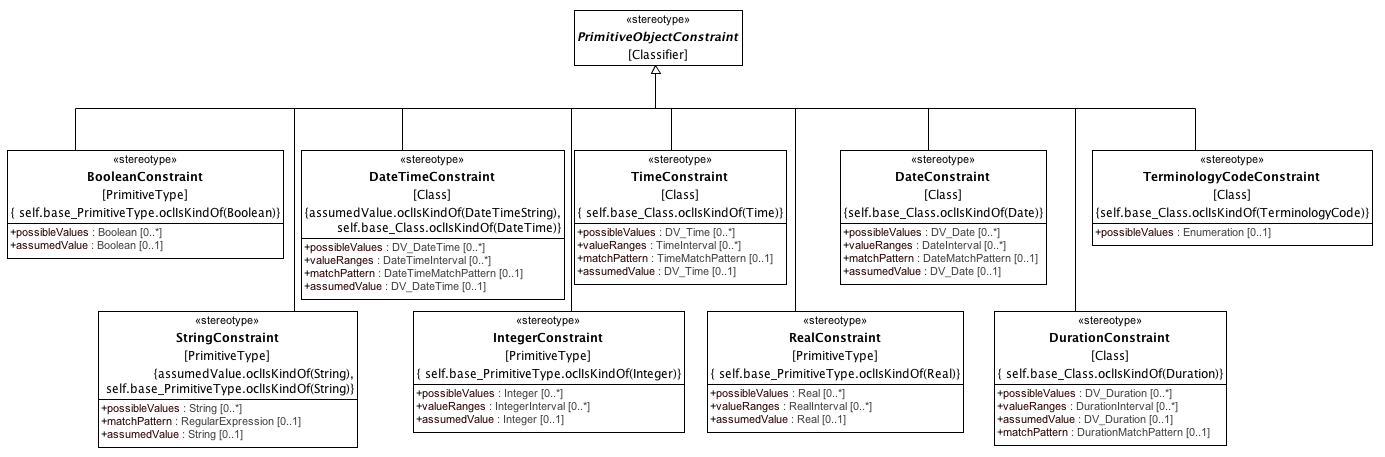
**Archetype Metadata**

### Data Type Constraints

This clause defines the stereotypes that are used to constrain the "primitive" or assumed types that are built in to the AML Profile.

#### Primitive Type Constraints

This clause identifies the basic "primitive" type constraints



**Primitive Type Constraints**

#### Date and Time Match Types

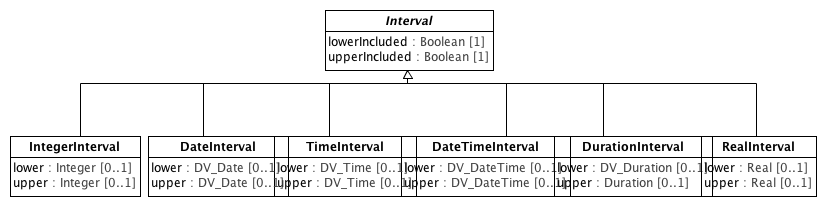
This clause describes the various matching types derived from ISO 8601 *Data elements and interchange formats – Information interchange – Representation of dates and times*



**Date and Time Match Types**

#### Intervals

This section contains the definition of the intervals used in the primitive type constraints.

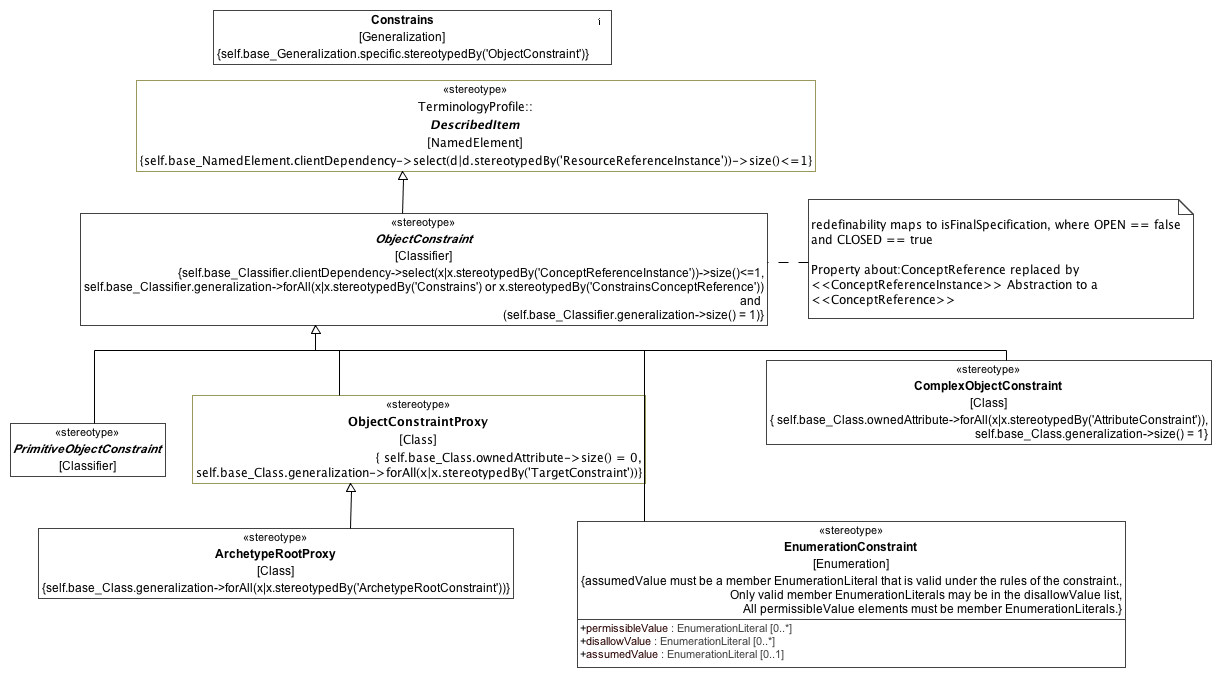


**Intervals**

### Object and Property Constraints

#### Object Constraints

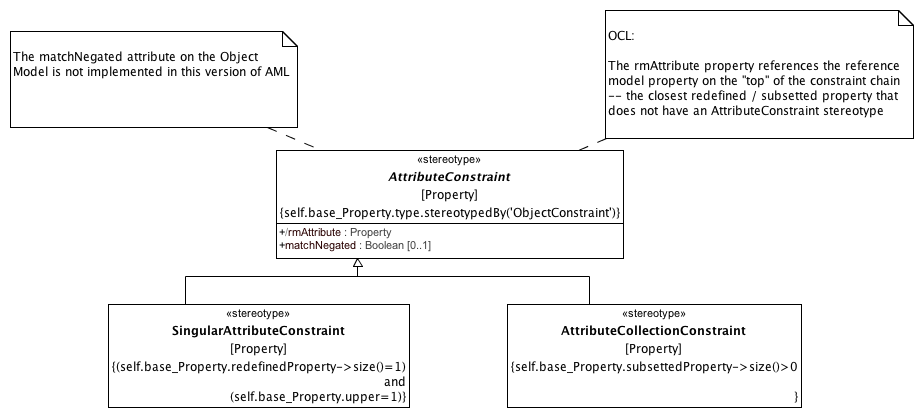
This clause describes the various forms of constraint that can be applied to the UML **Class.**



**Object Constraints**

#### Attribute Constraints

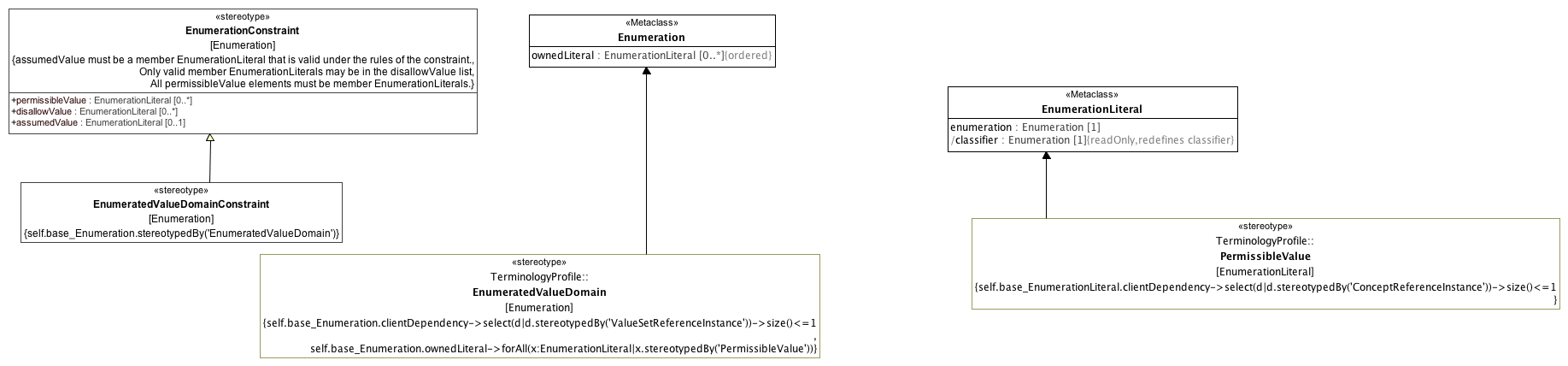
This clause defines the constraints that can be applied to a UML **Property**.



**Attribute Constraints**

#### Enumeration Constraints

This clause defines the constraints that can be applied to UML **Enumeration** and its derivatives.



**Enumeration Constraints**

#### Constraint Proxies

This clause defines mechanisms for reusing and "importing" constraints.



**Constraint Proxies**

### Terminology Constraints

This clause defines the profiles used for constraining Enumerations, EnumeratedValueDomains and TerminologyCodes.



**TerminologyConstraints**

### Profile Elements

#### <Class> DateInterval

**Description**

A set of contiguous Dates.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : ReferenceModelProfile::DV\_Date**

The earliest Date that may be included in the interval.

**• <property> ::= + upper : ReferenceModelProfile::DV\_Date**

The latest Date that may be included in the interval.

#### <Class> DateTimeInterval

**Description**

A set of contiguous DateTimes.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : ReferenceModelProfile::DV\_DateTime**

The earliest DateTime that may be included in the interval.

**• <property> ::= + upper : ReferenceModelProfile::DV\_DateTime**

The latest DateTime that may be included in the interval.

#### <Class> DurationInterval

**Description**

A set of contiguous Duration magnitudes.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : ReferenceModelProfile::DV\_Duration**

The shortest Duration that may be included in the interval.

**• <property> ::= + upper : UML Standard Profile::UML2 Metamodel::Duration**

The longest Duration that may be included in the interval.

#### <Class> DV\_Date

**Description**

Represents an absolute point in time, as measured on the Gregorian calendar, and specified only to the day. [cite: SUPPORT\_IM, p24]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedDate](#_2a37a0f4cd69c5ee1a32f787b23de875)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::date**

ISO8601 string for date, in format **YYYYMMDD** or **YYYY-MM-DD**, or a partial invariant.

#### <Class> DV\_DateTime

**Description**

Represents an absolute point in time, specified to the second. [cite: SUPPORT\_IM, p27]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::dateTime**

A valid ISO8601 string for date/time, in format

**YYYYMMDDThhmmss[,sss][Z | ±hh[mm]]** or in extended format **YYYY-MM-DDThh:mm:ss[,sss][Z | ±hh[mm]]** or a partial variant.

Note that this class includes 2 deviations from ISO 8601:2004:

• for partial date/times, any part of the date/time up to the month may be missing, not just seconds and minutes as in the standard;

• the time 24:00:00 is not allowed, since it would mean the date was really on the next day.

#### <Class> DV\_Duration

**Description**

Represents a period of time corresponding to a difference between two time-points. See: ISO8601\_DURATION [cite: SUPPORT\_IM, p30]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::duration**

Value is a valid ISO 8601 duration, i.e. takes the form:

• **P[nnY][nnM][nnW][nnD][T[nnH][nnM][nnS]]** Where each nn represents a number of years, months, etc. nnW represents a number of 7- day weeks.

Note: allowing the **W** designator in the same expression as other designators is an exception to the published standard, but necessary in clinical information (typically for representing pregnancy duration).

#### <Class> DV\_Time

**Description**

Represents an absolute point in time from an origin usually interpreted as meaning the start of the current day, specified to the second. [cite: SUPPORT\_IM, p25]

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d),

[Sample Data Binding](#_8548f4699761fde908c0fa2fe95f29ba)

**Direct Subclasses (Specialization)**

[MappedTime](#_6395ddb1d0c31197b0c966c1281ec5aa)

**Attributes**

**• <property> ::= + value : XMLPrimitiveTypes::time**

Value representation is ISO8601 string for time, i.e. in form: **hhmmss[,sss][Z|±hh[mm]]** or the extended form: **hh:mm:ss[,sss][Z|±hh[mm]]**, or a partial invariant.

A small deviation to the ISO 8601:2004 standard in this class is that the time 24:00:00 is not allowed, for consistency with **DateTime**.

#### <Class> IntegerInterval

**Description**

A set of contiguous integers.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Integer**

The smallest integer that may be included in the interval.

**• <property> ::= + upper : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Integer**

The largest integer that may be included in the interval.

#### <Class> Interval

**Description**

A set of contiguous values bounded by upper and lower limits. Inclusion of the upper and lower limits is governed by the upperIncluded and lowerIncluded properties.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Subclasses (Specialization)**

[DateInterval](#_eb65cb2938a6220d8f4a10f0d8aba136),

[DateTimeInterval](#_956e6c028c830c0453b74cbd2204109e),

[DurationInterval](#_3d4fd0ce80d2a2e88d2f42b3cb7dbec5),

[IntegerInterval](#_86618450de28d822bd6b57b67a32ab2b),

[RealInterval](#_d4f7314ff920dd15ee0e834cfbd4e6f2),

[TimeInterval](#_2db4f3574d756c0312a2a6559efd3ad9)

**Attributes**

**• <property> ::= + lowerIncluded : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

A Boolean value indicating whether the value asserted to be the lower limit is included in the set it bounds.

**• <property> ::= + upperIncluded : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

A Boolean value indicating whether the value asserted to be the upper limit is included in the set it bounds.

#### <Class> RealInterval

**Description**

A set of contiguous Real numbers

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Real**

The smallest Real that may be included in the interval

**• <property> ::= + upper : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Real**

The largest Real that may be included in the interval

#### <Class> TerminologyCode

**Description**

Represents a scoped identifier that is unique within the context of a given namespace. While a terminology code is treated as data type from the AML perspective, there multiple ways of representing the same "value" including:

* terminologyId + code -- a combination of the URI of a scoping namespace (terminologyId) and identifier (code)
* uri -- a URI that embodies both the namespace and the code
* terminologyVersion -- the URI of a terminology (code system, ontology) that contains a description of what the code represents.
* term -- a language specific string that provides a hint at the intended meaning of the code

Note that "terminologyId" and "terminologyVersion" represent very different constructs. "terminologyId" represents a namespace, not an ontology / code system. "terminologyVersion" references an collection of code descriptions.

**TerminologyCode** represents the contents of either an **EnumerationLiteral** or a **PermissibleValue**. It corresponds to the AOM **Terminology\_Code**, and its function is to gather sufficient information that, in combination with the KnownNamespace Enumeration extension, it will be possible to populate a variety of target coded types, including the ISO 21090 CD type, the openEHR DV\_CODED\_TEXT, the CIMI CODED\_TEXT, etc.

**Diagrams**

[AML Data Types](#_c6849d5d118a9832b3df7e75a9d1a83d)

**Attributes**

**• <property> ::= + terminologyId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The *name* or *uri* of the *namespace* that scopes *code.*

Either the *name* of the **Enumeration** that owns the **EnumerationLiteral** represented by the *code* attribute or the *uri* tag of the **Enumeration** if it is extended by the **ScopedIdentifier** stereotype and the tag is present.

**• <property> ::= + code : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The representation of an **EnumerationLiteral.** This will usually be the name of the **EnumerationLiteral**, but it should be possible for implementations to associate other numeric or string values with literals through either the *slot* or *id* of an **IdentifiedItem** extension that will become the code value instead.

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The URI of a **ConceptReference**. This should be populated in the case where the selected value is a **PermissibleValue** and either (a) the owning **Enumeration** is stereotyped by **ScopedIdentifier** and it has a *uriPattern* attribute or (b) the **PermissibleValue** includes a *meaning* tag of type **ConceptReference**, in which case *uri* equals the *uri* of the **ConceptReference**.

If two TerminologyCodes have the same *uri*, they are considered to be identical. If a uri is absent in one or both TerminologyCodes, they are considered to be identical only if the *terminologyId* and *code* are the same.

**• <property> ::= + terminologyVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The uri of a CodeSystem or CodeSystemVersion that carried a description of the particular code at some point in time. This attribute is strictly informative and should is not part of the identity of the terminology code.

If the source **EnumerationLiteral** is stereotyped by **PermissibleValue**, the meaning tag is present and the **ConceptReference** in the meaning tag has a **CodeSystemAndVersion**, this is set to:

* The *uri* of the **CodeSystemVersionReference** if it is present
* The *uri* of the **CodeSystemReference** otherwise.

**• <property> ::= + term : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A string designating the intended target of the **TerminologyCode** in a given language and context. This is strictly informative and is not part of the part of the identity of a **TerminologyCode** as it may change as data moves through different languages or contexts.

#### <Class> TimeInterval

**Description**

A set of contiguous Times.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b)

**Direct Superclasses (Generalization)**

[Interval](#_3ae971b2839139d9692e47ec472148b6)

**Attributes**

**• <property> ::= + lower : ReferenceModelProfile::DV\_Time**

The earliest Time that may be included in the interval.

**• <property> ::= + upper : ReferenceModelProfile::DV\_Time**

The latest Time that may be included in the interval.

#### <Stereotype> 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.

**Diagrams**

[Intervals](#_8566828d89ca01e8919a51d19aa6cd8b),

#### <Stereotype> Archetype

**Description**

An **Archetype** is a package that contains a versioned set of constraints on a **Class** that is a member of the **ReferenceModel** that is owned by the containing **ArchetypeLibrary**.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

**Attributes**

**• <property> ::= + archetypeName : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The human readable name of the Archetype. Example: "clinical data group", "laboratory test", "serum sodium", "cimi composition"

**• <property> ::= + originalLanguage : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The original spoken or written language in which the archetype was authored (Example: EN, DE, ES). A language type drawn for the language numeration.

**• <property> ::= + archetypeType : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The implementation specific type or classification of the archetype. In the ADL context, this would include archetype, template\_overlay, etc.

**• <property> ::= + rmPackagePath : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The qualifiedName of the package containing the root class constrained by this archetype.

**• <property> ::= + rmClassName : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

Name of the root Class constrained of this archetype.

**Constraints**

* **mustBeOwned**

The containing Package must be an <<ArchetypeLibrary>>.

[OCL]

self.base\_Package.owningPackage.stereotypedBy('ArchetypeLibrary')

* **ownsVersions**

All members of a package must be <<ArchetypeVersion>>.

[OCL]

self.base\_Package.ownedMember->select(x|x.oclIsKindOf(Classifier) and not(x.oclIsKindOf(Association)))->forAll(x|x.stereotypedBy('ArchetypeVersion'))

* **originalLanguage**

originalLanguage must be contained by a <<Language>> Enumeration.

[OCL]

self.originalLanguage.enumeration.stereotypedBy('Language')

* **archetypeType**

Each archetypeType must be an <<ArchetypeType>>.

[OCL]

self.archetypeType.enumeration->forAll(x|x.stereotypedBy('ArchetypeType'))

* **constrainsRMElement**

The Class or one of its generalization ancestors is a member of the ArchetypeLibrary Reference Model.

[OCL]

self.base\_Package.nestingPackage.packageImport.importedPackage.nestedPackage.packagedElement->select(x|x.oclIsKindOf(Classifier)).oclAsType(Classifier) ->exists(x|self.base\_Package.packagedElement->select(p|p.stereotypedBy('ArchetypeVersion')).oclAsType(Classifier).general->includes(x))

* **currentVersion**

The package must contain one <<ArchetypeCurrentVersion>>.

[OCL]

self.base\_Package.packagedElement->select(x|x.stereotypedBy('ArchetypeCurrentVersion'))->size()=1

* **specializesArchetype**

If an Archetype specializes another Archetype, they both constrain the same class.

[OCL]

self.base\_Package.packageMerge.mergedPackage.ownedType->select(t|t.stereotypedBy('ArchetypeVersion')).oclAsType(Classifier).general ->forAll(x|self.base\_Package.ownedType->select(t|t.stereotypedBy('ArchetypeVersion')).oclAsType(Classifier).general->includes(x))

* **archetypeIdType**

The Package must be stereotyped by ArchetypeId.

[OCL]

self.base\_Package.stereotypedBy('ArchetypeId')

#### <Stereotype> ArchetypeCurrentVersion

**Description**

The archetype version that is considered to be the "latest" or "current" version. The current version is the referent of specializations and archetype references that omit the complete version identifier.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903) ,

**Constraints**

* **isArchetypeVersion**

The Class must be stereotyped by ArchetypeVersion.

[OCL]

self.base\_Class.stereotypedBy('ArchetypeVersion')

#### <Stereotype> ArchetypeId

**Description**

An artifact uniquely identifying 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.

**Diagrams**

[Archetype Metadata](#_c8719d73828cfab6778459fc65cfee21)

**Attributes**

**• <property> ::= + id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

ID for the Archetype

#### <Stereotype> ArchetypeLibrary

**Description**

An **ArchetypeLibrary** is a **Package** that contains a collection of archetypes that constrain classes within the imported **ReferenceModel**. An **ArchetypeLibrary** must import exactly one **Package** that is stereotyped as a **ReferenceModel**.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

**Constraints**

* **oneReferenceModel**

The must be one <<ReferenceModelImport>> PackageImport.

[OCL]

self.base\_Package.packageImport->select(stereotypedBy('ReferenceModelImport'))->size() = 1

* **onlyArchetypes**

All packaged elements must be <<Archetype>>.

[OCL]

self.base\_Package.packagedElement->forAll(p|p.stereotypedBy('Archetype'))

#### <Stereotype> ArchetypeRootConstraint

**Description**

Connects an <<ArchetypeVersion>> to an <<ArchetypeRootProxy>> to be constrained.

**Diagrams**

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Superclasses (Generalization)**

[TargetConstraint](#_6de94cd3c6736f017766fe61020a5a13)

**Constraints**

* **specificIsRootProxy**

Specific must be an <<ArchetypeRootProxy>>

[OCL]

self.base\_Generalization.specific.stereotypedBy('ArchetypeRootProxy')

* **generalIsArchetypeVersion**

The general must be an <<ArchetypeVersion>>

[OCL]

self.base\_Generalization.general.stereotypedBy('ArchetypeVersion')

#### <Stereotype> ArchetypeRootProxy

**Description**

A reference to a target Archetype that constrains the same subclass or descendant thereof as that referenced by the constrains attribute of the NamedObjectConstraint itself.

**ArchetypeRootProxy** implements the use\_archetype construct in ADL. Note that an ArchetypeRootProxy may or may not reference a specific version of the target Archetype. If the targetVersion attribute is absent, the ArchetypeRootProxy is understood to reference the ArchetypeVersion that is considered to be "current" at whatever time that the proxy is referenced.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Superclasses (Generalization)**

[ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e)

**Constraints**

* **redefinesConstrains**

Any generalization must be <<ArchetypeRootConstraint>>

[OCL]

self.base\_Class.generalization->forAll(x|x.stereotypedBy('ArchetypeRootConstraint'))

#### <Stereotype> ArchetypeType

**Description**

An implementation specific classification of Archetype types. In the ADL context, the ADL 2.0.5 specification describes the following types:

* archetype
* template
* template\_overlay
* operational\_template

This would be implemented with a specialization of the ArchetypeType enumeration with the four literals above. ArchetypeType does not affect the semantics of an AML implementation.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

#### <Stereotype> ArchetypeType

**Description**

An implementation specific enumeration of possible archetype types. In ADL, this would extend an enumeration that includes literals for *archetype*, *template, template\_overlay* and *operational\_template* and their flattened equivalents.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

#### <Stereotype> ArchetypeVersion

**Description**

A set of constraints that can be applied as a predicate against instances of **Class** referenced by the constrains attribute of the containing Archetype.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

**Direct Superclasses (Generalization)**

[AuthoredResource](#_13ad6987e15b787d385f0b30ff25d6c9),

[ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62)

**Attributes**

**• <property> ::= + amlVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The version of the AML specification used to define this version of the archetype.

**Constraints**

* **archetypeRoot**

This Class must have a generalization.

[OCL]

self.base\_Class.generalization->size()=1

* **ownedByArchetype**

The namespace owner must be <<Archetype>>

[OCL]

self.base\_Class.namespace.stereotypedBy('Archetype')

* **archetypeVersionIdType**

This Class must also be stereotyped by <<ArchetypeVersionId>>.

[OCL]

self.base\_Class.stereotypedBy('ArchetypeVersionId')

#### <Stereotype> ArchetypeVersionId

**Description**

An **Archetype** identifier. **ArchetypeId** must uniquely identify an **Archetype** within the context of the containing **ArchetypeLibrary**. **ArchetypId** may be extended to support specific workflows and community needs, but all implementations must support a string representation.

**Diagrams**

[Archetype Metadata](#_c8719d73828cfab6778459fc65cfee21)

**Attributes**

**• <property> ::= + id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The archetype identifier.

#### <Stereotype> AssumedValue

**Description**

An instance of a link between a **ConceptReferenceConstraint** and an assumedValue.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1)

#### <Stereotype> AttributeCollectionConstraint

**Description**

A constraint on an attribute with an upper multiplicity of greater than one. An AttributeCollectionConstraint may assert an ordering on the member AttributeCollectionMembers and may assert the minimum and maximum number of instances that may occur for the referenced attribute.

**Diagrams**

[Attribute Constraints](#_88dce20413dd8833c4e90da9fe432855)

**Direct Superclasses (Generalization)**

[AttributeConstraint](#_1bf8a3231ae21af2dec84426b5618c38)

**Constraints**

* **mustHaveSubsettedProperty**

Must have at least one subsettedProperty.

[OCL]

self.base\_Property.subsettedProperty->size()>0

#### <Stereotype> AttributeConstraint

**Description**

In any information model, attributes are either single-valued or multiply-valued, i.e. of a generic container. Both have existence, while multiply-valued attributes also have cardinality

**Diagrams**

[Attribute Constraints](#_88dce20413dd8833c4e90da9fe432855)

**Direct Subclasses (Specialization)**

[AttributeCollectionConstraint](#_4bc615eb2707782fc8254702b7e0b435),

[SingularAttributeConstraint](#_2d1a6d8b2806092b50ec3fd4cd2db35b)

**Attributes**

**• <property> ::= + rmAttribute : UML Standard Profile::UML2 Metamodel::Property**

**• <property> ::= + matchNegated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

**Constraints**

* **objectConstraintTarget**

The Property type must be an <<ObjectConstraint>>

[OCL]

self.base\_Property.type.stereotypedBy('ObjectConstraint')

#### <Stereotype> 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**.

**AuthoredResource** can be associated with **ResourceTranslations**, **Descriptions**, etc through **OpaqueBehaviors** stereotyped by **ResourceTranslation** and **ResourceDescription** respectively.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

**Direct Subclasses (Specialization)**

[ArchetypeVersion](#_61831f1c446a753e3069251f603bfa37)

**Attributes**

**• <property> ::= + originalLanguage : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The original language the model was authored in.

**• <property> ::= + isControlled : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

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

**• <property> ::= + isGenerated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

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 specialized archetypes.

**• <property> ::= + resourceSource : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

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

**• <property> ::= + resourceDocumentLanguage : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

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

**• <property> ::= + resourceDocumentSyntax : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

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

**• <property> ::= + resourceSourceURI : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

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.

**Constraints**

* **languageEnumeration**

originalLanguage must be in a <<Language>> Enumeration

[OCL]

self.originalLanguage.enumeration.stereotypedBy('Language')

#### <Stereotype> BooleanConstraint

**Description**

A constraint on a property of Boolean type.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

A set of allowed Boolean values (e.g., True, False).

**• <property> ::= + assumedValue : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean**

A Boolean value to be assumed to apply if no value is provided.

**Constraints**

* **constrainsBoolean**

The constraint is applied to a Boolean.

[English]

self.base\_PrimitiveType.oclIsKindOf(Boolean)

#### <Stereotype> CodeSystemReference

**Description**

A ResourceReference whose *uri* identifies to a code system (aka. "concept system, "terminology" or "ontology"). Note, while the UML **Package** is the base for the **CodeSystemReference**, its intent is to represent a the *uri* and optional name of a code system and a single, optional *version* and should contain nothing else. The optional *version* of type **CodeSystemVersionReference** is represented as a *packageImport*.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **version**

There must be at most 1 imported <<CodeSystemVersionReference>> Package.

[Binary]

self.base\_Package.packageImport.importedPackage->select(p|p.stereotypedBy('CodeSystemVersionReference'))->size()<=1

#### <Stereotype> CodeSystemReferenceInstance

**Description**

**CodeSystemReferenceInstance** represents the optional link between a **ConceptReference** and a describing code system and optional version.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Constraints**

* **isCodeSystemReference**

Must be a valid instance of a CodeSystemReference.

[OCL]

self.base\_Abstraction.supplier->forAll(x|x.stereotypedBy('CodeSystemReference'))

#### <Stereotype> CodeSystemVersionReference

**Description**

A URI referencing a specific version of a code system. Note, while UML **Package** is used as the base for the **CodeSystemVersionReference** stereotype, a **Package** extended by **CodeSystemVersionReference** may not have any members. It is simply the name of a code system version and not the actual entity itself.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Attributes**

**• <property> ::= + unnamed1 : TerminologyProfile::ValueSetDefinitionReference**

#### <Stereotype> ComplexObjectConstraint

**Description**

A collection of constraints on an instance of an instance of a UML **Class.** A **ComplexObjectConstraint** may constrain the existence, cardinality or possible values on one or more of the constrained class attributes.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903) ,

[Object Constraints](#_f139650e10793c0005c93b604992495c)

**Direct Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Subclasses (Specialization)**

[ArchetypeVersion](#_61831f1c446a753e3069251f603bfa37)

**Constraints**

* **singleParent**

Every constraint must specialize exactly one Class

[OCL]

self.base\_Class.generalization->size() = 1

* **allAttributeConstraints**

All owned attributes must be <<AttributeConstraint>>

[OCL]

self.base\_Class.ownedAttribute->forAll(x|x.stereotypedBy('AttributeConstraint'))

#### <Stereotype> ConceptReference

**Description**

A URI uniquely identifing a "concept" (aka. class, entity, individual or, in some contexts "term"), accompanied by additional information conveying the intended meaning, code and source of the information used to determine the intent of the URI.

**ConceptReference** can be used to bind a **Class** or **DataType** constraint to its terminology binding (aka "meaning" in ISO 11179-3). **ConceptReference** may also be associated with a describing code system and optional code system version by a usage relationship whose client is **ConceptReference** and whose supplier is **CodeSystemReference**. The purpose of this association is to identify the particular description that was used when selecting the concept reference.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **describingCodeSystem**

Definition of describingCodeSystem

describingCodeSystem:=self.base\_\*.clientDependency.supplier->select(p|p.stereotypedBy('CodeSystemReference'))->asSequence()->first()

[English]

describingCodeSystem:=self.base\_\*.clientDependency.supplier->select(p|p.stereotypedBy('CodeSystemReference'))->asSequence()->first()

#### <Stereotype> ConceptReferenceInstance

**Description**

A **ConceptReferenceInstance** associates a referencing element with a corresponding **ConceptReference**. It is used to link a **PermissibleValue** with a value meaning, an **ObjectConstraint** with what the constraint is "about" (aka. "meaning in ISO 11179) and to identify the possible and assumed values for a **ConceptReferenceConstraint**.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Direct Subclasses (Specialization)**

[AssumedValue](#_7c76bb5bbde4643a957c898bfc8af67a),

[PossibleValue](#_ef85d33c740061c0c756c4e535c34ccc)

**Constraints**

* **isConceptReference**

There must be one <<PermissibleValue>> or <<ConceptReference>> client and one <<ConceptReference>> supplier.

[OCL]

self.base\_Abstraction.client->select(c|c.stereotypedBy('PermissibleValue') or c.stereotypedBy('ConceptReference'))->size()=1 and self.base\_Abstraction.supplier->select(c|c.stereotypedBy('ConceptReference'))->size()=1

#### <Stereotype> Constrains

**Description**

Constraining relationship between a generalization and an **ObjectConstraint**.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Subclasses (Specialization)**

[TargetConstraint](#_6de94cd3c6736f017766fe61020a5a13)

**Constraints**

* **specificObjectConstraint**

Specific must be an <<ObjectConstraint>>

[OCL]

self.base\_Generalization.specific.stereotypedBy('ObjectConstraint')

#### <Stereotype> DataBinding

**Description**

The **DataBinding** stereotype identifies a "bridge" class providing a mapping between an internal AML data type identified by the **AMLDataType** stereotype and a corresponding Reference Model **DataType** or **Class** identified by the **MappedDataType** stereotype.

The Class or DataType extended by a DataBinding functions as the base point for AML constraints.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Constraints**

* **oneAMLDataType1**

This Class must specialized an <<AMLDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **oneAMLDataType2**

This DataType must specialize an <<AMLDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->select(x|x.stereotypedBy('AMLDataType'))->size() = 1

* **atLeastOneMappedDataType1**

This Class must be stereotyped by <<MappedDataType>>

[OCL]

not(self.base\_Class.oclIsUndefined()) implies self.base\_Class.general->exists(x|x.stereotypedBy('MappedDataType'))

* **atLeastOneMappedDataType2**

This DataType must be specialized from <<MappedDataType>>

[OCL]

not(self.base\_DataType.oclIsUndefined()) implies self.base\_DataType.general->exists(x|x.stereotypedBy('MappedDataType'))

* **amlTypeMatch**

[English]

A data binding should specialize exactly two classifiers, one that is stereotyped by MappedDataType and a second that must be a descendant of UML Boolean, UML Integer, UML String, UML Real, Date, Time, DateTime, Duration or TerminologyCode. Furthermore, the amlType of the classifier stereotyped by MappedDataType must reference the corresponding classifier of the other generalization.

#### <Stereotype> DateConstraint

**Description**

A constraint on a property of Date type

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : ReferenceModelProfile::DV\_Date**

A set of allowed Date values (e.g., ‘2000-01-01’)

**• <property> ::= + valueRanges : ConstraintProfile::DateInterval**

A set of Date value ranges, any value of which is considered valid (e.g., ‘|>= 2000-01-01|’, ’2000-01-01..2005-06-30’)

**• <property> ::= + matchPattern : ConstraintProfile::DateMatchPattern**

A string pattern implying a set of valid Date values (e.g., ‘2000-??-xx’, ‘yyyy-xx-xx’)

**• <property> ::= + assumedValue : ReferenceModelProfile::DV\_Date**

A Date value to be assumed to apply if no value is provided

**Constraints**

* **constrainsDate**

Constraint must be applied to a DateType

[English]

self.base\_Class.oclIsKindOf(Date)

#### <Stereotype> DateTimeConstraint

**Description**

A constraint on a property of DateTime type.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : ReferenceModelProfile::DV\_DateTime**

A set of allowed DateTime values (e.g., ‘2000-01-01T09:30:00’).

**• <property> ::= + valueRanges : ConstraintProfile::DateTimeInterval**

A set of DateTime value ranges, any value of which is considered valid (e.g., ‘|>=2000-01-01T 09:30:00|’, ’ 2000-01-01T09:30:00.. 2000-01-01T11:30:00’).

**• <property> ::= + matchPattern : ConstraintProfile::DateTimeMatchPattern**

A string pattern implying a set of valid DateTime values (e.g., ‘yyyy-mm-ddT13:??:xx’, ‘yyyy-mm-ddThh:xx:xx’).

**• <property> ::= + assumedValue : ReferenceModelProfile::DV\_DateTime**

A DateTime value to be assumed to apply if no value is provided.

**Constraints**

* **assumedDateTime**

AssumeDateTime is applied to a DateTimeString.

[English]

assumedValue.oclIsKindOf(DateTimeString)

* **constrainsDateTime**

Base class is a DateTime.

[English]

self.base\_Class.oclIsKindOf(DateTime)

#### <Stereotype> DescribedItem

**Description**

The **DescribedItem** stereotype allows multilingual designations/descriptions and scoped identifiers to be assigned a **NamedElement**.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e),

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb),

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77),

[IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Subclasses (Specialization)**

[EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a),

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2),

[PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692),

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **validAbout**

There must be a maximum of one <<ResourceReferenceInstance>> Abstraction

[OCL]

self.base\_NamedElement.clientDependency->select(d|d.stereotypedBy('ResourceReferenceInstance'))->size()<=1

#### <Stereotype> DesignatableItem

**Description**

The **DesignatableItem** stereotype allows a set of language / sign / description tuples to be assigned to a **NamedElement**. This represents a flattened version of the **ItemDescription** in the AML Meta Model.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff),

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

**• <property> ::= + language : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

A reference to a spoken or written language. *language* must be an **EnumerationLiteral** in an **Enumeration** stereotyped by the **Language** stereotype. Only one *sign*/description tuple is allowed per language.

**• <property> ::= + sign : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A string designating or "signifying" the name of the *NamedElement* in the corresponding *language. sign* is a required field and cannot contain an empty String ("").

**• <property> ::= + description : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A definition or description of the **NamedElement** in the corresponding *language*. *description* is optional and its absence is indicated by the empty ("") String.

**Constraints**

* **language sign description tuples are designations**

The ordered sequences language, sign, description must be same length and language must be in a <<Language>>.

[OCL]

self.language->size()=self.sign->size() and self.language->size()=self.description->size() and self.language.enumeration->forAll(l|l.stereotypedBy('Language'))

#### <Stereotype> DurationConstraint

**Description**

A constraint on a property of Duration type

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : ReferenceModelProfile::DV\_Duration**

A set of allowed Duration values (e.g., ‘P5d’).

**• <property> ::= + valueRanges : ConstraintProfile::DurationInterval**

A set of Duration value ranges, any value of which is considered valid (e.g., ‘|>= P5d|’, ’P5d..P8d’).

**• <property> ::= + assumedValue : ReferenceModelProfile::DV\_Duration**

A Duration value to be assumed to apply if no value is provided.

**• <property> ::= + matchPattern : ConstraintProfile::DurationMatchPattern**

A string pattern that implies a set of valid Duration values (e.g., ‘Pd’, ‘PThm’).

**Constraints**

* **constrainsDuration**

The constrain is applied to a Duration.

[English]

self.base\_Class.oclIsKindOf(Duration)

#### <Stereotype> EnumeratedValueDomain

**Description**

An **EnumeratedValueDomain** represents a discrete set of possible values (**PermissibleValues**) for a particular field or data element. Each **PermissibleValue** identifies a unique value and (optionally) its intended meaning. An **EnumeratedValueDomain** may be associated with a scoped identifier, which, in the ADL case would be an "ac" code. The set of **PermissibleValues** for the **EnumeratedValueDomain** would represent an ADL value\_set, which would bind an "ac" code to a set of "at" codes.

An **EnumeratedValueDomain** may also reference a **ValueSetReference** using the **ValueSetReferenceInstance** relationship which would resolve to a set of **PermissibleValues**.

**Diagrams**

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Constraints**

* **meaningIsConceptReference**

There is a maximum of one <<ValueSetReferenceInstance>> Abstractions.

[OCL]

self.base\_Enumeration.clientDependency->select(d|d.stereotypedBy('ValueSetReferenceInstance'))->size()<=1

* **permissibleValues**

All instances must be type permissibleValue

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('PermissibleValue'))

#### <Stereotype> EnumeratedValueDomainConstraint

**Description**

A constraint on a **EnumeratedValueDomain** that represents a discrete set of possible values for a particular field or data element.

**Diagrams**

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[TerminologyConstraints](#_759880e6f359cf189858504f68956934)

**Direct Superclasses (Generalization)**

[EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9)

**Constraints**

* **constrainsEVD**

This Enumeration is an <<EnumeratedValueDomain>>

[OCL]

self.base\_Enumeration.stereotypedBy('EnumeratedValueDomain')

#### <Stereotype> EnumerationConstraint

**Description**

A constraint on a possible values in a UML **Enumeration**. **EnumerationConstraint** allows a modeler to:

1. List which **EnumerationLiterals** owned by the **Enumeration** are permitted.
2. List which **EnumerationLiterals** owned by the **Enumeration** are not permitted.
3. Add an **EnumerationLiteral** that is the assumed value for this instance.
4. Add additional **EnumerationLiterals**, extending the parent **Enumeration**.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[TerminologyConstraints](#_759880e6f359cf189858504f68956934)

**Direct Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Subclasses (Specialization)**

[EnumeratedValueDomainConstraint](#_7b4688dbd3826f33c726c87847ae4a72)

**Attributes**

**• <property> ::= + permissibleValue : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The **Enumeration** *members* allowed in this constrained instance. If not empty, only the **EnumerationLiterals** in this list are valid values. Only **EnumerationLiterals** are valid for the general **Enumeration** or **EnumerationConstraint** can be listed in *permissibleValue*.

**• <property> ::= + disallowValue : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The **Enumeration** *members* not allowed in this constrained instance. An **EnumerationLiteral** in the disallowValue takes precedence over a *member* in the *permissibleValue* -- if it appears in both places it is not allowed.

**• <property> ::= + assumedValue : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

The value to be assumed if it is absent in a data instance. *assumedValue* must be a valid value for the constrained enumeration -- if there is a non-empty list of *permissibleValues*, it must appear in the list and it may not appear in the *disallowValue* list.

**Constraints**

* **validPermissibleValue**

[English]

All permissibleValue elements must be member EnumerationLiterals.

* **validDisallowValue**

[English]

Only valid member EnumerationLiterals may be in the disallowValue list

* **validDefaultValue**

[English]

assumedValue must be a member EnumerationLiteral that is valid under the rules of the constraint.

#### <Stereotype> IdentifiedItem

**Description**

The **IdentifiedItem** stereotype allows the assignment of one or more optional identifiers to be assigned to a **NamedElement**.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Attributes**

**• <property> ::= + id : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

An identifier unique within the context of the containing **Enumeration**.

**Constraints**

* **scopedIdentifierLiteral**

Every *id* property is an instance of a ScopedIdentifier.

[OCL]

self.id->forAll(x:EnumerationLiteral | x.enumeration.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.

[OCL]

self.id->forAll(l1 | self.id->forAll(l2 | l1.enumeration = l2.enumeration implies l1 = l2))

#### <Stereotype> Infrastructure

**Description**

A stereotype indicating a base Property represents an aspect of an Archetype implementation such as a specific Archetype identifier or other element. Properties with the Infrastructure stereotype cannot be constrained in AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

#### <Stereotype> IntegerConstraint

**Description**

A constraint on a property of Integer type.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Integer**

A set of allowed Integer values (e.g., 2, 5).

**• <property> ::= + valueRanges : ConstraintProfile::IntegerInterval**

A set of Integer value ranges, any value of which is considered valid.

**• <property> ::= + assumedValue : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Integer**

An Integer value to be assumed to apply if no value is provided.

**Constraints**

* **assumedInteger**

assumedValue is an integer.

[English]

assumedValue.oclIsKindOf(Integer)

* **constrainsInteger**

The constraint is applied to an Integer.

[English]

self.base\_PrimitiveType.oclIsKindOf(Integer)

#### <Stereotype> KnownNamespace

**Description**

**KnownNamespace** extends the **Enumeration** listing the set of namespace identifiers recognized by a particular implementation or archetype library. The name of a member literal represents a local namespace identifier. Each local identifier can, in turn, be realized by a second namespace. As an example, an AML model might have 3 known namespaces:

1. "SCT"
2. "SNOMED-CT"
3. "LOINC"

The first two namespaces would reference the same realization target, a **ScopedIdentifier** for the SNOMED CT identifiers referenced by the model, while the third would reference another **ScopedIdentifier** **Enumeration** containing the LOINC identifiers referenced by the model.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Constraints**

* **hasScopedIdentifiers**

Set if knownNamespace has an identifier uniquely referencing its scoping namespace.

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('ScopedIdentifier'))

* **namespaceInstances**

All owned EnumerationLiterals must be <<NamespaceInstance>>

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|x.stereotypedBy('NamespaceInstance'))

#### <Stereotype> Language

**Description**

The set of languages referenced by an AML instance. Typically these will be drawn from ISO 639-1 or one of its derivatives.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Superclasses (Generalization)**

[ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

#### <Stereotype> MappedDataType

**Description**

Extends a Class or DataType in the Reference Model and indicates the base Class or DataType represents the classifier referenced by the *amlType* tag and is to be treated as an atomic "primitive" type that cannot be further decomposed.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + amlType : UML Standard Profile::UML2 Metamodel::Classifier**

The assumed or built-in AML Data Type corresponding to the extended Class or DataType.

**Constraints**

* **isAMLDataType**

The amlType must reference a classifier (Class or DataType) that has a AMLDataType stereotype.

[OCL]

self.amlType.stereotypedBy('AMLDataType')

* **amltypematch**

#### <Stereotype> NamespaceInstance

**Description**

NamespaceInstance is an instance of Namespace.

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Constraints**

* **mustBeScopedIdentifier**

There must be a single <<KnownNamespace>> client and a single <<ScopedIdentifier>> supplier.

[OCL]

self.base\_Realization.client->size()=1 and self.base\_Realization.supplier->size()=1 and self.base\_Realization.client->forAll(x|x.stereotypedBy('KnownNamespace')) and self.base\_Realization.supplier->forAll(x|x.stereotypedBy('ScopedIdentifier'))

#### <Stereotype> ObjectConstraint

**Description**

An **ObjectConstraint** is a specialization of a **Classifier** where there are no new *ownedAttributes* that do not *subset* or *redefine* an inherited *attribute*.

The **Classifier** stereotyped by an **ObjectConstraint** must participate in exactly one **Generalization** relationship which must be stereotyped by the **Constrains** stereotype.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Direct Subclasses (Specialization)**

[ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62),

[EnumerationConstraint](#_1bc74c3698f61990aff3aec96088f0a9),

[ObjectConstraintProxy](#_c8ae60f7f44b70cf5dce7db03aa6ac1e),

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Constraints**

* **redefinesGeneralization**

This Classifier must have exactly one generalization, and that is a <<Constrains>> or <<ConstrainsConceptReference>> generalization.

[OCL]

self.base\_Classifier.generalization->forAll(x|x.stereotypedBy('Constrains') or x.stereotypedBy('ConstrainsConceptReference')) and (self.base\_Classifier.generalization->size() = 1)

* **about**

At most one "about" reference via <<ConceptReferenceInstance>> Abstraction

[OCL]

self.base\_Classifier.clientDependency->select(x|x.stereotypedBy('ConceptReferenceInstance'))->size()<=1

#### <Stereotype> ObjectConstraintProxy

**Description**

An *ObjectConstraintProxy* asserts a constraint defined elsewhere in the archetype is to apply to the **Classifier** instance referenced by the proxy *constrains* attribute.

**Diagrams**

[Object Constraints](#_f139650e10793c0005c93b604992495c),

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Subclasses (Specialization)**

[ArchetypeRootProxy](#_f5f73ce565f73d8b4808997e54e8e698)

**Constraints**

* **redefinesConstrains**

All generals are <<TargetConstraint>>

[OCL]

self.base\_Class.generalization->forAll(x|x.stereotypedBy('TargetConstraint'))

* **noNewAttributes**

No attributes allowed on this Class

[OCL]

self.base\_Class.ownedAttribute->size() = 0

#### <Stereotype> PermissibleValue

**Description**

A unique within the context of the owning **EnumeratedValueDomain**. A **PermissibleValue** can be associated with an identifier, which, in ADL would be an "at" code. A permissible value can be associated with one or more language specific designations and optional descriptions, which represent the ADL term\_definitions section (as applied to "at" codes). A **PermissibleValue** may also reference a **ConceptReference** via the **ConceptReferenceInstance** association,where the **ConceptReference** identifies the intended meaning for the value.

**Diagrams**

[Enumeration Constraints](#_4049ef2e39f1ca7b7abb65f10409d85f),

[Enumerated Value Domains](#_bbceab59bf026d3b0616400766b2619e)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1)

**Constraints**

* **valueMeaningIsConceptReference**

PermissibleValue.about must be a concept reference

[OCL]

self.base\_EnumerationLiteral.clientDependency->select(d|d.stereotypedBy('ConceptReferenceInstance'))->size()<=1

#### <Stereotype> PossibleValue

**Description**

An instance of a link between a **ConceptReferenceConstraint** and a set of possible values.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1)

#### <Stereotype> PrimitiveObjectConstraint

**Description**

A constraint on a UML or AML primitive type

The abstract primitive object constraint has no properties, but those of its specializations follow common patterns. (It also seems to have no relationships, e.g., composition to complex constraint.)

These patterns include properties defining allowed values via explicit enumeration, range definition, and matching patterns. These allowed value properties are to be treated as permissive rather than restrictive, with the result that if more than one of these properties is defined, the set of valid values for an instance is the union of values so defined. Inclusion and exclusion of interval end points in the valueRanges properties is governed by the interval data types.

Another pattern comprises the ‘assumed value’ properties. In an archetype containing optional data elements, assumed values can be used to specify values that can reliably be inferred by receivers. For example, an archetype for ‘blood pressure measurement’ might include an optional data element describing the patient position, with choices ‘lying’, ‘sitting’ and ‘standing’. Since this element is optional, conformant data could be created that does not contain it. The ‘assumed value’ property allows a value to be explicitly stated so that all users/systems know what value to assume when optional items are not included in the data, in cases where such an assumption is appropriate. In the ‘blood pressure measurement’ example, one might define such a value as ‘sitting’—the most common value—with the understanding that such an assumption makes it important for instances in which the value is different, or not known, to say so, notwithstanding that the value is technically optional.

Assumed values are definable on any primitive type. If the archetype does not define an assumed value, no reliable assumption can be made by the receiver of an archetype instance concerning the value of the property.

Date and Time constraints follow the string syntax of the ISO 8601 Representation of dates and times.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35),

[Object Constraints](#_f139650e10793c0005c93b604992495c)

**Direct Superclasses (Generalization)**

[ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Direct Subclasses (Specialization)**

[BooleanConstraint](#_40ee863e6fd02692437dae1d81ba12de),

[DateConstraint](#_ff8930d68c378c02c221704764a5c9d4),

[DateTimeConstraint](#_7dde1322feeec9c32a95df44c39d8e48),

[DurationConstraint](#_384c080719f5bd1b45eae1293215b466),

[IntegerConstraint](#_2219fb1dcaf5f26a0ed07de77d69cd5e),

[RealConstraint](#_b921bc493035fb4e067213114372e254),

[StringConstraint](#_982033c222702fafb1d4d3ed7b399317),

[TerminologyCodeConstraint](#_ef76317db67a290898f39af3c51eee9c),

[TimeConstraint](#_d8c772ca77efc45bee8711f1de17afc0)

#### <Stereotype> RealConstraint

**Description**

A constraint on a property of Real type.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Real**

A set of allowed Real values (e.g., 37.2, 100.265).

**• <property> ::= + valueRanges : ConstraintProfile::RealInterval**

A set of Real value ranges, any value of which is considered valid.

**• <property> ::= + assumedValue : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Real**

Real value to be assumed to apply if no value is provided.

**Constraints**

* **constrainsReal**

Constraint is applied to a real.

[English]

self.base\_PrimitiveType.oclIsKindOf(Real)

#### <Stereotype> ReferenceModel

**Description**

A stereotype that classifies a package as a reference model, identifying it as a valid target for archetype constraints

A reference model defines the classes that the archetypes in an archetype library can constrain, and the ReferenceModel profile ensures that the reference model can provide the metadata that an archetype library may require.

An archetype library is associated with exactly one ReferenceModel, and it may use the properties that the model defines in the construction of archetype identifier strings. See the openEHR Archetype Definition Language exposition of archetype identification for one example of how this may work.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35),

#### <Stereotype> ReferenceModel

**Description**

This stereotype identifies a package as a Reference Model -- a package which contains the collection of UML Classes that can be constrained by the Archetypes in an Archetype Library. The Reference Model stereotype also allows the specification of the publisher, namespace and version of a Reference Model in a form compatible with a modeling language such as ADL.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

**Attributes**

**• <property> ::= + rmPublisher : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The name of the Reference Model publisher. Corresponds to *rm\_publisher* in AOM 2.0

**• <property> ::= + rmNamespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The owning domain name of the archetype. Corresponds to the *namespace* attribute in AOM2.0.

**• <property> ::= + rmVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

Designates the version id of the reference model on which the archetype is based. Corresponds to *rm\_release* in AOM 2.0

#### <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**.

**Diagrams**

[Archetypes](#_1c8c4aa3906888e5c2c8895f5ed19903)

**Constraints**

* **libraryOnly**

importing namespace must be an <<ArchetypeLibrary>>

[OCL]

self.base\_PackageImport.importingNamespace.stereotypedBy('ArchetypeLibrary')

* **libraryReferenceModel**

importedPackage must be <<ReferenceModel>> and importNamespace must be <<ArchetypeLibrary>>

[OCL]

self.base\_PackageImport.importedPackage.stereotypedBy('ReferenceModel') and self.base\_PackageImport.importingNamespace.stereotypedBy('ArchetypeLibrary')

#### <Stereotype> ResourceDescription

**Description**

A detailed description of the source, provenance, copyright, etc of the ADL Resource.

**Diagrams**

[Archetype Metadata](#_c8719d73828cfab6778459fc65cfee21)

#### <Stereotype> ResourceReference

**Description**

A ResourceReference couples a NamedElement with a uri referencing an external class, category, individual or "concept". It should be noted the uri in ResourceReference has the semantics associated with the PersistentURI in the CTS2 specification -- it is not intended to reference a resource directly, but to "name" a resource that has a description in one or more terminologies, (code systems, classification systems, ontologies).

The uri serves as the identity of a ResourceReference. It may be accompanied by additional language specific designations and descriptions as well as by a ScopedIdentifer that identifies the target as a namespace/name tuple.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[DescribedItem](#_d45578f848d02aad83980903e5bde7d1),

[DesignatableItem](#_80448b03d480bba05b1e156796878f77)

**Direct Subclasses (Specialization)**

[CodeSystemReference](#_7de70b4fbd9e6a164f7f00cde47dfd5a),

[CodeSystemVersionReference](#_9a8de95c38ebe2d6ce506cbc9bef7b7a),

[ConceptReference](#_57ae94153b82f28889d42ad4aa8fe1e0),

[ValueSetDefinitionReference](#_a4fedb7858ead8d2272640d51b53719a),

[ValueSetReference](#_1a1ca20b54028ee5e2eb20af35411f6e)

**Attributes**

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

URI of the resource.

**Constraints**

* **uniqueId**

Every identifier must come from a different namespace

[OCL]

self.id->forAll(i1 | self.id->forAll(i2 | i1.enumeration = i2.enumeration implies i1 = i2))

* **language sign description represent tuple**

The ordered sequences language, sign, description must be the same length and language must be part of a <<Language>> Enumeration.

[OCL]

self.language->size()=self.sign->size() and self.language->size()=self.description->size() and self.language.enumeration->forAll(l|l.stereotypedBy('Language'))

#### <Stereotype> ResourceReferenceInstance

**Description**

ResourceReferenceInstance is an abstraction that associates a NamedElement that is stereotyped by DescribedItem with an instance of its reference.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[DesignatableItem](#_80448b03d480bba05b1e156796878f77),

[IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Direct Subclasses (Specialization)**

[CodeSystemReferenceInstance](#_e055a6cce06d0838055b62dbfbf235f2),

[ConceptReferenceInstance](#_6a2f733d1d3ea9bc232f96caadb113e1),

[ValueSetReferenceInstance](#_f3184cb0f8e704f5122c5e97fb9f130c)

**Constraints**

* **isResourceReference**

Must be an instance of a resource reference class

[OCL]

(self.base\_Abstraction.client->select(d|d.stereotypedBy('DescribedItem'))->size()=1) and (self.base\_Abstraction.supplier->select(d|d.stereotypedBy('DescribedItem'))->size()=1)

#### <Stereotype> ResourceTranslation

**Description**

A collection of translations for a resource.

**Diagrams**

[Archetype Metadata](#_c8719d73828cfab6778459fc65cfee21)

**Attributes**

**• <property> ::= + versionLastTranslated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

The ArchetypeVersion identifier of the version that was last translated.

**Constraints**

* **hasOneTranslationDetails**

classifierBehavior must be <<TranslationDetails>>

[OCL]

self.base\_OpaqueBehavior.classifierBehavior.stereotypedBy('TranslationDetails')

#### <Stereotype> Runtime

**Description**

Indicates that the base Property represents a dynamic or "runtime" element that cannot be constrained using AML.

**Diagrams**

[Reference Model Stereotypes](#_549d0bc2f9c4ac0312fc619b71bf4d08)

#### <Stereotype> ScopedIdentifier

**Description**

A **ScopedIdentifier** is an enumeration that may include a uri referencing the scope of the identifier and a uri pattern describing how uri's are constructed from the member EnumerationLiterals.

As an example, the SNOMED CT identifier namespace might have a *uri* of "http://snomed.info/sct", which identifies the sole namespace and an *identifierURIPattern* of "http://snomed.info/id/$1" which indicates an EnumerationLiteral named 74400008 would be represented as "http://snomed.info/id/74400008".

**Diagrams**

[IdentificationAndDesignation](#_0628ce7e5381e7543ba417bb320e03fb)

**Direct Subclasses (Specialization)**

[ArchetypeType](#_7dc1530ae1ef855ecc3eb9bd5b555a14),

[KnownNamespace](#_b9f78b93edc24bb3301ba69a57e4afc3),

[Language](#_446e0591a4f825b22cd9e573c1239a72)

**Attributes**

**• <property> ::= + uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A URI referencing the namespace associated with the ScopedIdentifier. Examples: <http://snomed.info/sct> http://loinc.org

**• <property> ::= + identifierURIPattern : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A URI substitution pattern, where "$1" indicates where the name of an owned EnumerationLiteral would be substituted to create a URI. Example: http://loinc.org/id/$1.

#### <Stereotype> SingularAttributeConstraint

**Description**

An AttributeConstraint that constrains the possible values for a singular property -- a property with an upper multiplicity of 1.

**Diagrams**

[Attribute Constraints](#_88dce20413dd8833c4e90da9fe432855)

**Direct Superclasses (Generalization)**

[AttributeConstraint](#_1bf8a3231ae21af2dec84426b5618c38)

**Constraints**

* **mustHaveRedefinedProperty**

Property must have exactly one redefined Property and an upper bound of 1.

[OCL]

(self.base\_Property.redefinedProperty->size()=1) and (self.base\_Property.upper=1)

#### <Stereotype> StringConstraint

**Description**

A constraint on a property of String type. Most constrainable text values are best constrained by language-independent terminology constraints, but this tactic is supported.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A set of allowed String values (e.g., ‘Robert’)

**• <property> ::= + matchPattern : ConstraintProfile::RegularExpression**

A Perl regular expression defining allowed string constructions.

**• <property> ::= + assumedValue : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

A String value to be assumed to apply if no value is provided.

**Constraints**

* **assumedString**

The assumed value must be a String

[English]

assumedValue.oclIsKindOf(String)

* **constrainsString**

The constraint is applied to a String.

[English]

self.base\_PrimitiveType.oclIsKindOf(String)

#### <Stereotype> TargetConstraint

**Description**

Connects an <<ObjectConstraint>> to an <<ObjectConstraintProxy>>

**Diagrams**

[Constraint Proxies](#_17547bff44c4353bd3a454a0c3c7e577)

**Direct Superclasses (Generalization)**

[Constrains](#_f91b532413834ad1de94d0b0af526f5b)

**Direct Subclasses (Specialization)**

[ArchetypeRootConstraint](#_6eafe370f24f01390e7ab79d6568ea94)

**Constraints**

* **specificIsProxy**

Specific must be an <<ObjectConstraintProxy>>

[OCL]

self.base\_Generalization.specific.stereotypedBy('ObjectConstraintProxy')

* **generalIsObjectConstraint**

General must be <<ObjectConstraint>>

[OCL]

self.base\_Generalization.general.stereotypedBy('ObjectConstraint')

#### <Stereotype> TerminologyCodeConstraint

**Description**

A **TerminologyCodeConstraint** is a constraint on the possible values of the AML **TerminologyCode** type. It constrains the possible values of a terminology code by referencing an **Enumeration** whose member **EnumerationLiterals** represent the possible values for the constrained **TerminologyCode.** *possibleValues* can reference a simple **Enumeration**, a **ScopedIdentifier** or an **EnumeratedValueDomain**. The referenced **Enumeration** or **EnumeratedValueDomain** may be extended by one or more **EnumerationConstraints** or **EnumeratedValueDomainConstraints** respectively.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : UML Standard Profile::UML2 Metamodel::Enumeration**

An **Enumeration** whose (possibly constrained) instances represent the possible values for the constrained **TerminologyCode**. If absent, the TerminologyCode values are not constrained.

**Constraints**

* **mustBeEnumeratedValueDomain**

valueSet, if defined, must be an <<EnumeratedValueDomain>>

[OCL]

not(self.valueSet.oclIsUndefined()) implies self.valueSet.stereotypedBy('EnumeratedValueDomain')

* **constrainsConceptReference**

This Class must be a <<ConceptReference>>

[OCL]

self.base\_Class.oclIsKindOf(TerminologyCode)

#### <Stereotype> TimeConstraint

**Description**

A constraint on a property of Time type.

**Diagrams**

[Primitive Type Constraints](#_ae5d1776f07899a7af4725351a338b35)

**Direct Superclasses (Generalization)**

[PrimitiveObjectConstraint](#_c72b6d9c8a46b96f02fdfefe3b8b0568)

**Attributes**

**• <property> ::= + possibleValues : ReferenceModelProfile::DV\_Time**

A set of allowed Time values (e.g., ‘09:30:00’).

**• <property> ::= + valueRanges : ConstraintProfile::TimeInterval**

A set of Time value ranges, any value of which is considered valid (e.g., ‘|>= 09:30:00|’, ’09:30:00..11:30:00’).

**• <property> ::= + matchPattern : ConstraintProfile::TimeMatchPattern**

A string pattern implying a set of valid Time values (e.g., ‘13:??:xx’, ‘hh:xx:xx’).

**• <property> ::= + assumedValue : ReferenceModelProfile::DV\_Time**

A Time value to be assumed to apply if no value is provided.

**Constraints**

* **constrainsTime**

The constraint is applied to a Time type.

[English]

self.base\_Class.oclIsKindOf(Time)

#### <Stereotype> TranslationDetails

**Description**

A basic stereotype for adding translation details to a resource translation

**Diagrams**

[Archetype Metadata](#_c8719d73828cfab6778459fc65cfee21)

**Attributes**

**• <property> ::= + accreditation : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String**

**• <property> ::= + language : UML Standard Profile::UML2 Metamodel::EnumerationLiteral**

**Constraints**

* **languageEnumeration**

language must be contained by a <<Language>> Enumeration

[OCL]

self.language.enumeration.stereotypedBy('Language')

#### <Stereotype> ValueSetDefinitionReference

**Description**

A *uri* that names the value set that the extended **Enumeration** represents. The set of **EnumerationLiterals** in an **Enumeration** stereotyped by **ValueSetDefinitionReference** represent the set or a subset of the **ConceptReferences** returned by the interpretation of the referenced definition.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **specializesValueSet**

A ValueSetDefinitionReference is a specialization of a ValueSetReference.

[OCL]

self.base\_Enumeration.general->select(g|g.stereotypedBy('ValueSetReference'))->size()<=1

#### <Stereotype> ValueSetReference

**Description**

A *uri* that names the value set that the extended Enumeration represents. The set of EnumerationLiterals in an Enumeration stereotyped by ValueSetReference represent the set or a subset of the ConceptReferences returned by the "current" (in the CTS2 sense) definition of the ValueSet.

**Diagrams**

[Resource References](#_aa2597b87275ed4e1724a94a9c31d90b)

**Direct Superclasses (Generalization)**

[ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Constraints**

* **definition**

This Enumeration must have exactly one <<ValueSetDefinitionReference>> Generalization.

[OCL]

self.base\_Enumeration.general->select(g|g.stereotypedBy('ValueSetDefinitionReference'))->size()<=1

#### <Stereotype> ValueSetReferenceInstance

**Description**

A link between a **ConceptReferenceConstraint** and a value set and optional definition, where the value set resolves to a set of **ConceptReferences**.

**Diagrams**

[Resource Reference Relationships](#_6d065336864e77a240c6e7a3eb36e8cd)

**Direct Superclasses (Generalization)**

[ResourceReferenceInstance](#_9d682f32f4917feea358e696d1fd146d)

**Constraints**

* **isValueSetReference**

Must have one <<EnumeratedValueDomain>> client and one <<ValueSetReference>> supplier.

[OCL]

self.base\_Abstraction.client->select(c|c.stereotypedBy('EnumeratedValueDomain'))->size()=1 and self.base\_Abstraction.supplier->select(c|c.stereotypedBy('ValueSetReference'))->size()=1

# Appendix A: AML Meta Model

The AML Object Model package describes how the Archetype Definition Language (ADL) and Archetype Object Model (AOM) requirements relate to the corresponding entities the UML 2.5 Specification, ISO 11179-3 and the OMG Common Terminology Services (CTS2) 1.1 specification. It models the required features as UML classes and describes their relationship to the classes and properties in the UML specification itself.

## AArchetype Meta Model

This section shows the relationship between archetype libraries, archetypes and archetype versions. It also describes the ADL/AOM specific metadata that accompanies a given version of an archetype.

### Archetype Libraries

This section describes Archetype Libraries, Archetypes and Archetype versions and their relationships.



**Archetype Libraries**

This diagram shows the relationships between the archetype library, its member archetypes and their versions and to the corresponding UML classes. It also shows the links between archetypes and their associated metadata.

### Archetypes and the UML Reference Model

This section shows the details of the relationships between archetype libraries, archetypes, archetype versions and the corresponding UML 2.5 classes. Note that the UML classes are shaded grey.



**ArchetypeRM**

An Archetype references (or constrains) a single UML Class.

The constrained Class must be a member of the UML Package that is constrained by the Archetype Library.

### ADL Archetype Metadata

This section describes the AOM 2.0 specific metadata that accompanies the AuthoredResource archetype description. For the purposes of the AML specification, this section is intended to:

1. Provide a concrete example of the sort of provenance and workflow related metadata that will accompany archetypes
2. Provide a model that can be used to extend the metadata in the AML profile to accomodate ADL/AOM specific metadata requirements.



**Metadata Object Model**

This diagram show the ADL 2.0 metadata applied as an specialize to the AuthoredResource class. Note that different implementations of AML may have different AuthoredResource specializations.

## AReference Model Meta Model

This section identifies the subset of the UML 2.5 Specification that is used in the definition of the AML Object Model. Classes with direct UML analogs will be shown in grey. We have removed generalizations, associations and properties that are addressed in the AML specifciation and, in several cases, have flattened inherited attributes and associations into a single class.

### 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.



**Intervals**

This diagram shows data types for intervals that can be used to construct AML primitive type constraints. The duration intervals are realized by substitution of their respective types through the parameterized Interval class.



**PrimitiveDataTypes**

This diagram shows the set of primitive data types that can be constrained using AML primitive type constraints along with the various match patterns that are used in to construct constraints.

Date and Time constraints follow the string syntax of the *ISO 8601 Representation of dates and times*. Constraint patterns are formed by using the ‘?’ (optional) or ‘x’ (prohibited) characters.

The syntax of legal patterns is shown by the following regular expressions:

**date\_pattern**: yyyy-(mm|??|XX)-(dd|??|XX)

**time\_pattern**: hh:(mm|??|XX):(ss|??|XX)

**time\_in\_date\_pattern**: T(hh|??|XX):(mm|??|XX):(ss|??|XX)

**date\_time\_pattern**: date\_pattern time\_in\_date\_pattern

Date and time intervals also require operators to define boundaries. The greater than, less than, and equal s signs are used to indicate values bounded by a value. Two stops (periods) are used to define an interval between two values.

**Single value pattern**: |> 09:30:00| -- any time after 9:30 am

**Interval pattern**: |2004-05-20..2004-06-02| -- a date range

Durations use the ISO 8601 prefix ‘P’ (period) followed by a set of unit/magnitude pairs indicating the number of years, months, weeks, days, and (with a ‘T’ time delimiter), minutes, etc. intended.

**Date duration template**: Pwd -- a duration containing weeks and/or days only, e.g. P4w

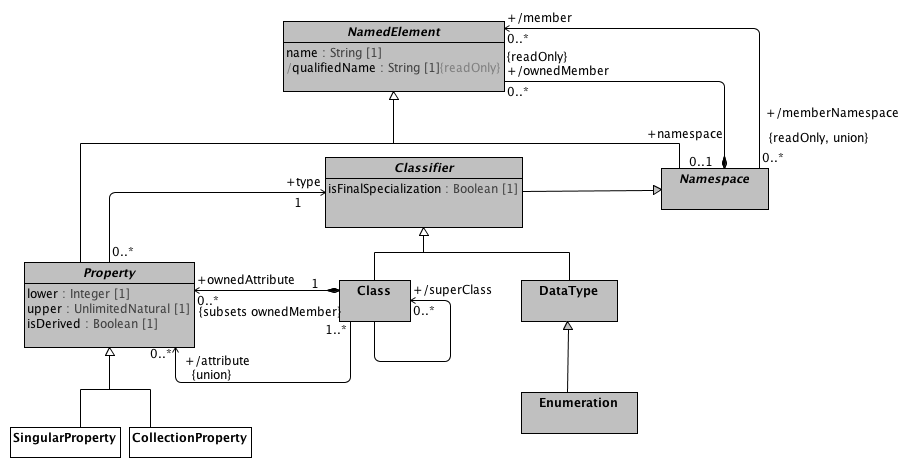
**Time duration template**: PThm -- a duration containing hours and/or minutes only, e.g. PT2h30m

**DateTime duration pattern**: P1dT8h -- 1 day 8 hrs

**Time range pattern**: |PT0m..PT1m30s| -- Reasonable time offset of first apgar sample

### Reference Metamodel

This section describes the aspects of the basic UML building blocks (Class, Property, DataType, Enumeration) that are used by the AML Object Model. Note that the representation, while faithful to UML 2.5, has been flattened and simplified.



**Reference Metamodel**

This diagram shows the aspects of the UML Classifier, Property and DataType elements that are used in the AML Object Model. Some of the aspects of the model such as the MultiplicityElement have been flattened for simplification. Note that the AML Object Model specializes UML Property by differentiating properties with an upper multiplicity value of 1 and an upper multiplicity value greater than one.

### Attribute Constraint References

This section describes the relationships between AML attribute constraints and the corresponding specializations of the UML Property



**Attribute Constraint References**

### Object Constraint References

This section describes the relationship between AML **NamedObjectConstraint, PrimitiveObjectConstraint** and the corresponding UML **Classifier** specializations. It should be noted that an AML constraint applies to *an instance of an instance of* a **Classifier**. As an example, a reference model might define an instance of a UML **Class** named "Person" with an attribute named "gender" with a type of an instance of a UML **Enumeration** called "Person Gender". A ComplexObjectConstraint named "Female Person" could be defined that restricts the instances of the Person class to those in a restriction Person Gender enumeration called "Female Gender"



**Object Constraint References**

### Template Metamodel

This section shows the subset of UML 2.5 Template Model that is applicable to the AML specification.

**Note:** The first version of the UML specification will not include the ability to author constraints on UML templates. This model is hsere strictly for future reference.



**Template Metamodel**

This diagram shows the aspects of UML Templates and UML template bindings that would be applicable to the AML constraint specification were it to address template and bindings. It is anticipated that the specification may be expanded at some future time to include these artifacts.

### Instance Metamodel

Instance specifications represent one of the bigger challenges for the AML specification. The UML Reference Model consists of a set of Classes, which instances of the UML 2.5 Class, DataTypes which are instances of the UML 2.5 DataType and/or PrimitiveTypes which are instances of the UML 2.5 PrimitiveType.

The archetype constraint language requires the ability to specify assumed values for **NamedObjectConstraints**, which will take the form of an instance of the Reference Model **Class** (or an instance of an instance of a UML class. The same pattern applies to **DataTypes** and (new) **PrimitiveTypes**.

The UML 2.5 modeling language supports **Class**, **DataType**, **PrimitiveType** and **Enumeration** an the metaclass level, and the models built using the modeling language support instances of **Class** (RMClass), **DataType** (RMDataType), **PrimitiveType** (RMPrimitiveType) and **Enumeration**. In addition, the language supports a small set of instances of instances of primitive types (**String**, **Real**, **Integer**, etc.) as well as instances of instances of the UML **Enumeration** Class (**EnumerationLiterals**).

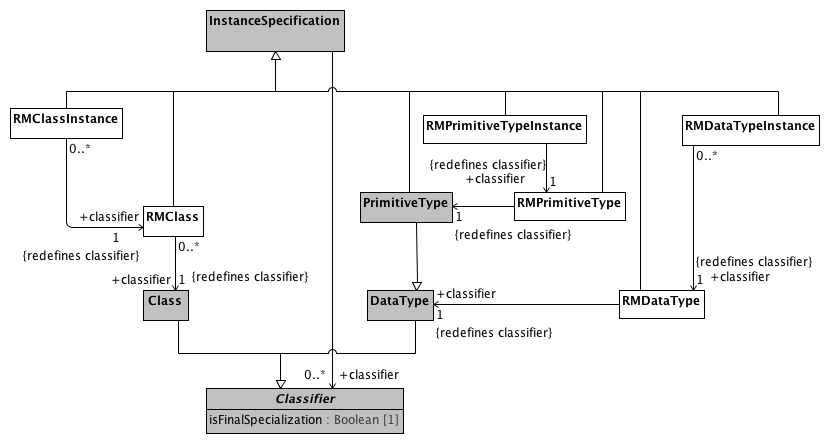
The modeling language, however, does not support a mechanism for specifying and exchanging instances of reference model classes, (new) primitive types or data types. This presents an issue when it becomes necessary to represent instances in constraints (permissible values) or assumed values.

The AML specification addresses the DataType and Primitive type issue by defining string representations for their values and constraints. It still faces the following issues:

1. How to represent instances of reference model classes as assumed values?
2. How to represent instances of the AML **ConceptReference** class, which consists of a URI, a namespace and name and other elements
3. How to represent the Designatable item, which consists of instances of a language/sign/description structure.

The AML profile has resolved these issues by:

1. Not allowing assumed values for non "primitive" types. This is a limitation on the ADL specification, but there currently aren't many, if any use cases.
2. ConceptReference is modelled as an extension to EnumerationLiteral, which allows additional instance properties.
3. Designations are represented as ordered collections of primitive types, where occurrence 0 of the language, sign, description represents the first element in the structure, occurrence 1 the second, etc.



**Instance Metamodel**

### Package Metamodel

This section describes the Namespace and Package aspects of the UML 2.5 model that are used in the AML Object Model Specification. Note that, while this model is faithful to the actual UML metamodel, it has been flattened for the purposes of simplification, showing only the classes and attributes that are relevant to the AML Object Model Specification.



**Package Metamodel**

This diagram shows the subset of the UML 2.5 Package specification that is used in the AML Object Model. In particular, this diagram is meant to call attention to the fact that every Class is a NamedElement and, at such, has a memberNamespace which, for our purposes is a UML Package.

AML requires that all archetypes within a given archetype library must directly or indirectly constrain an class that is a member of the Package referenced by the containing library.

### Enumeration Metamodel

ADL 2.0 provides the ability to constrain UML Enumeration types. Section 6.2.3.5 of the AOM specification describes how the AOM model represents constraints as instances of C\_INTEGER or C\_STRING to document the value assigned to an enumeration literal in a transformation to a programming language. The AML specification addresses the need to assign specific representations by noting that a given **Enumeration** type can have attributes, allowing the **EnumerationLiteral** instances to include both names and numeric or string values.

The diagram below shows the UML **Enumeration** and **EnumerationLiteral** data types and the AML extensions that blend them with the corresponding ISO 11179 Enumerated\_Value\_Domain and Permissible\_Value types.



**Enumeration Metamodel**

This diagram shows the relationship between the UML Enumeration and EnumerationLiteral data types and the EnumeratedValueDomain and PermissibleValue extensions.

## AConstraint Meta Model

The constraint model is the core of the archetype design. It specifies how constraints are defined, showing the object-attribute-object pattern characteristic of object constraints. Because objects are composed of properties, and properties consist of objects, the archetype definition consists of alternate layers of objects and attributes.

### Atomic Data Type Constraints

This section describes the atomic or "primitive type constraints" -- constraints that restrict the possible values of the built in AML data types and/or specify assumed values when the element is not included in a data instance.



**Atomic Data Type Constraints**

Primitive object constraints define patterns that archetypes use to constrain the values of primitive or atomic types. All complex constraints are composed of primitive constraints at the leaf level.

### Terminology Constraints

This section describes how instances of the **TerminologyCode** data type are constrained via a **TerminologyCodeConstraint**. The AML TerminologyCode acts as a bridge between the Common Terminology Services 2 (CTS2), the ISO 11179-3 designation, definition and identification sections and the

requirements imposed by various reference model targets, including ISO 21090 Coded Data (CD) types and its derivatives, FHIR Coding data type, openEHR DV\_CODED\_TEXT, CIMI CODED\_TEXT etc. The **TerminologyCode** target allows a given instance to include, at a bare minimum, an identifier of the scoping namespace (*terminologyId*) and the unique code within that namespace(*codeString*). While this information may or may not be sufficient to transform to an external target, it supports the minimum requirement that AML allow terminology codes to be restricted by simple, internal enumerations as represented by the "ac" ADL codes and the corresponding "at" code value sets.

Constraints are expressed as sets of **PermissibleValues** -- either directly via the *possibleValues* attribute or indirectly through the *valueSetConstraint* attribute which, possibly with the assistance of a terminology service, also results in a collection of **PermissibleValues**.

A PermissibleValue can represent:

* a simple **EnumerationLiteral**
* an **PermissibleValue** with one or more *identifiers* of type **Scoped\_Identifier**
* an **PermissibleValue** with an associated *valueMeaning* of type **ConceptReference**

Each of these situations has a different analog in terms of the target TerminologyCode.

**Simple EnumerationLiteral**

* The *name* of the containing **Enumeration** maps to *terminologyId*
* The *name* of the **EnumerationLiteral** maps to *codeString*

The uri attribute may be populated via the namespace map.

Note that the ADL specification may require that enumeration values may need to be represented as integers and strings. This may be accomplished by adding an inteter or string to the scoping **Enumeration** type and then adding a corresponding slot to each **EnumerationLiteral** instance. The mechanism and mapping for this approach are outside of the scope of the AML specification.

**PermissibleValue with an *identifier***

If the **PermissibleValue** has an associated *identifier*, the *identifier* attribute of the target *identifier* maps to codeString and, if present, the shorthand\_prefix of the identifier *scope* maps to *terminologyId*. If *shorthand\_prefix* is absent, then the *name* of the *scope* **Namespace** is used. If more than one *identifier* is present, an implementation may choose which identifier will be mapped.

The *uri* attribute may be populated via the namespace map.

**PermissibleValue with a *valueMeaning***

The *valueMeaning* attribute is of type **ConceptReference** and carries the equivalent of a "term binding" for an ADL "at" code. In this case, the target **ConceptReference** is mapped to the **TerminologyCode** as:

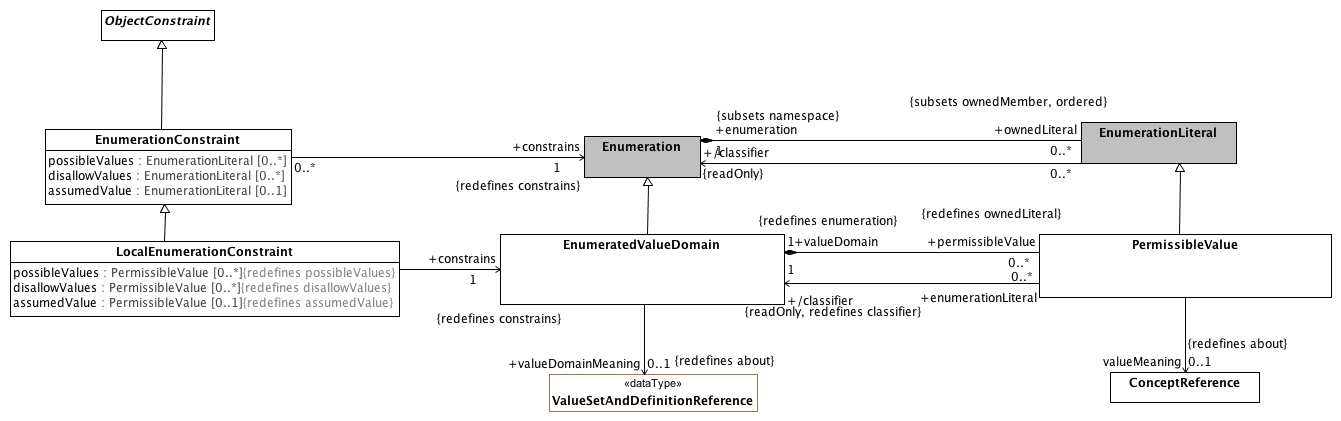
* The *conceptReference* *uri* maps to the *TerminologyCode* *uri*
* If the *conceptReference* has a *name* attribute (type **ScopedEntityName**), the *namespace* maps to the *terminologyId* attribute and *name* to *codeString*
* If the *conceptReference* does not have a *name* attribute, the identifier rules above should be employed or, lacking an identifier, the simple **EnumerationLiteral** rules.
* If the *conceptReference* has a *describingCodeSystemAndVersion* attribute thenit maps to the corresponding *terminologyVersion*. If the *conceptReference* does not have a *describingCodeSystemAndVersion* attribute, then the *terminologyVersion* attribute should be left empty.



**TerminologyConstraints**

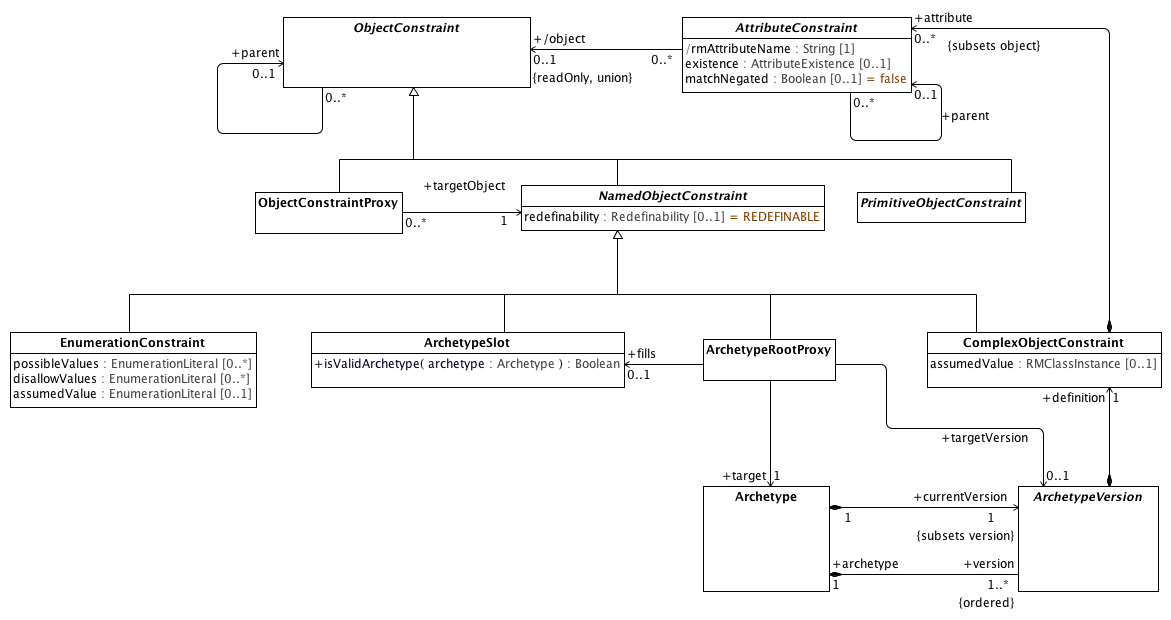
### Enumeration Constraints

This section describes the AML artifacts that constrain UML Enumerations and the ISO 11179-3 extension, EnumeratedValueDomain.



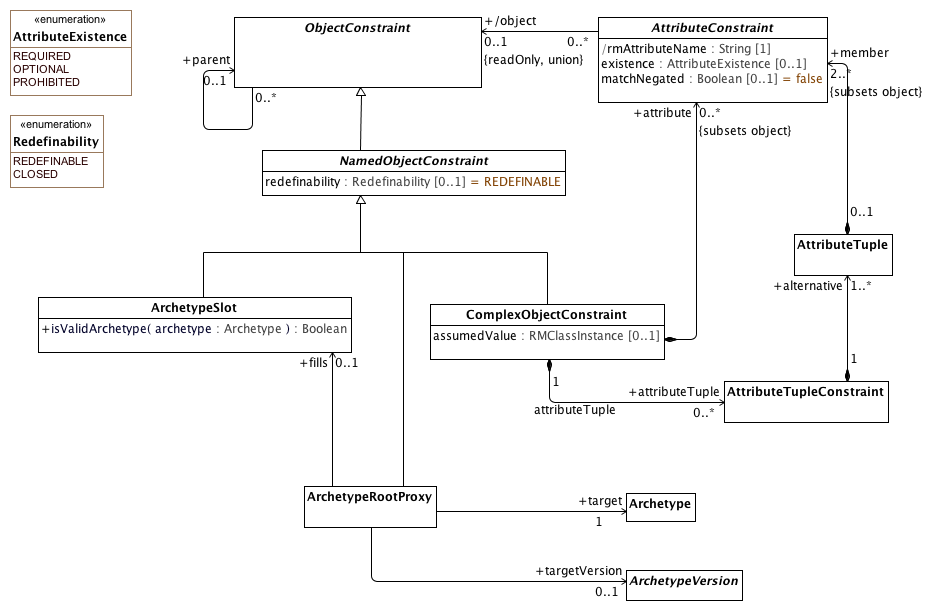
**EnumerationConstraints**

### Object Constraints



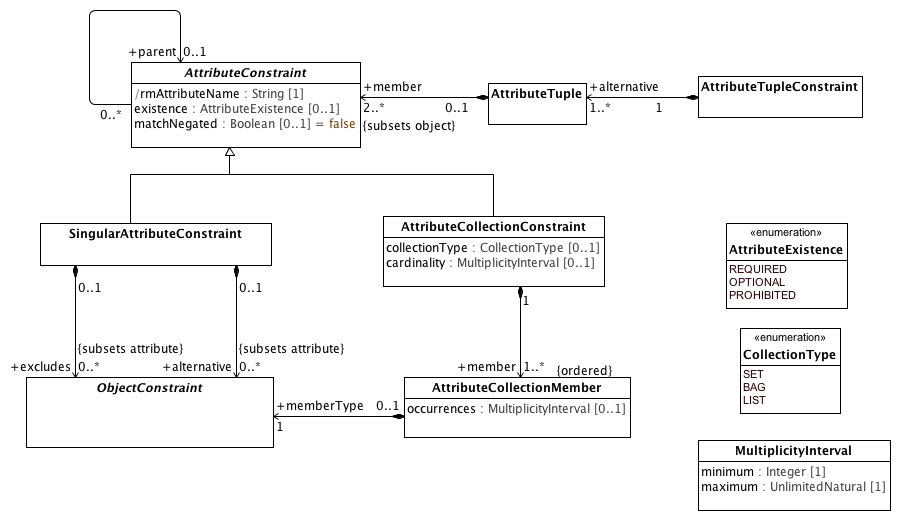
**Object Constraints**

#### Named Object Constraints



**Named Object Constraints**

### Attribute Constraints



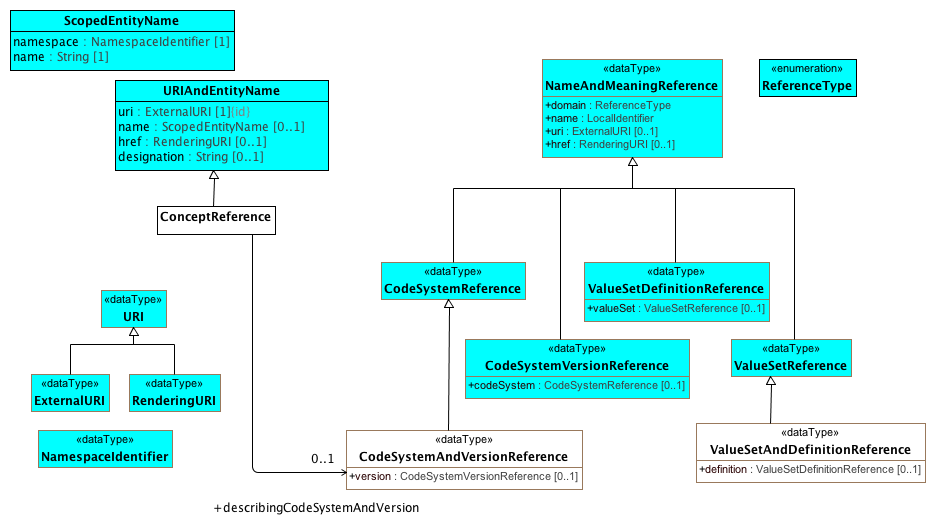
**Attribute Constraints**

## ATerminology Object Model

This section describes how a combination of the applicable elements of the Common Terminology Services (CTS2) and the ISO 11179-3 Metadata registries (MDR) - Part 3: Registry metamodel and basic attributes 3rd Edition can be combined to implement the ADL/AOM "id", "at" and "ac" identifier schemes and the term\_definitions, term\_bindings and value set sections of the Archetype Modeling Language.

We begin by describing the subsetting the core set of elements that are used from the CTS2 Core Model and some minor extensions that are needed for the AML Object Model. We then repeat this process with the ISO 11179-3 elements, integrating them with the CTS2 core elements. We finally identify the AML model components that have identities and corresponding entries in the terminology section.

### Common Terminology Services Components



**CTSCore Components**

This diagram contains the subset of the [CTS2 Core Model Elements](http://www.omg.org/cgi-bin/doc?formal/2013-12-04) that are used by the AML Object Model. The URI types are derived from Section 2.1.3, ScopedEntityName and URIAndEntityName from Section 2.2.3 Entity References and the NameAndMeaning references from Section

The AML specification adds the following restrictions to these types:

1. The href attribute is not used in URIAndENtityName or NameAndMeaningReference
2. NameAndMeaningReference.domain is determined by the type of the specialization and is omitted.
3. URI is mandatory for NameAndMeaning reference.

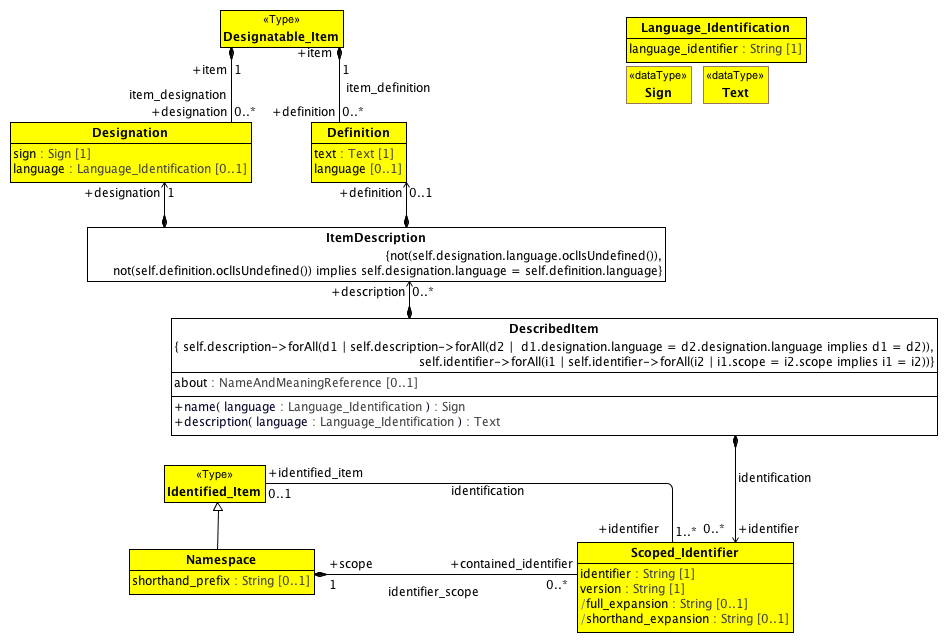
The ConceptReference class is specific to AML. The CTS2 model used the term "EntityReference", but the AML and 11179 communities are both used to using the notion of "Concept", although sometimes it is used to reference an identifier, sometimes a description and sometimes an abstract category in the viewer's mind. In AML, we are only interested in the first case, where ConceptReference is an identifier that can be described (or "included" in 11179 parliance) in one or more CodeSystems (Concept\_System in 11179 terms).

The ADL and AOM require that a Terminology\_Code (concept reference) include the identifier of the code system or version of a code system version that was used to determine the intended meaning of the concept at the point in time it was used. This requires one additional data type, "CodeSystemVersionReference", which in the AML context always includes the identifier of a code system but may or may not include a version.

### ISO 11179 Model Components

This section describes the subset of the *ISO 11179-3 Information technology - Metadata registries (MDR) - Part 3: Registry metamodel and basic attributes 3rd Edition* that are used in the construction of the AML Object Model. Model elements borrowed from ISO 11179 are identified with a yellow background to distinguish them from AML, CTS2 and UML Reference Model components. The first section describes the relationship between the designation, definition and identification sections and AML while the second addresses the meaning/representation links and their AML equivalent.

#### Designation, Definition and Identification



**Designation and Definition metamodel region**

This diagram contains the subset of the ISO 11179-3:2012(E) model elements in sections 7.2.2 Identification metamodel region and 7.3.2 Designation and Definition metamodel region that are used in the AML Object Model, accompanied by supporting elements drawn from several other sections.

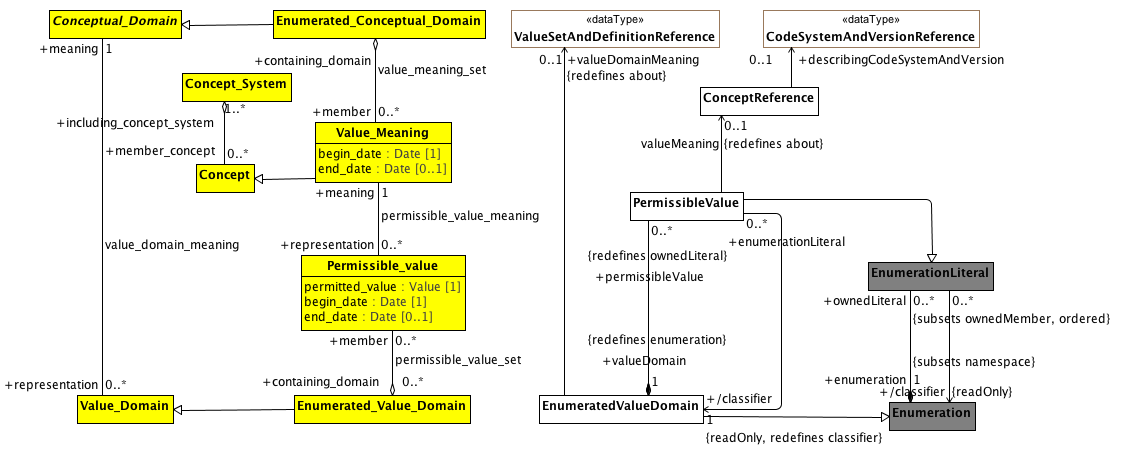
The AML object model is able to use the Designation, Definition and Scoped\_Identifier classes directly with the restrictions mentioned below. It is not able to use Designatable\_Item, which supports an unlimited number of definitions and/or designations while the AML specification includes the following restrictions:

1. There must be at most one ItemDescription per language
2. Every description must consist of exactly one Designation and zero or one Definitions
3. The language attribute is required
4. The language attributes of the Definition and Designation pairs must match.

It is also not able to take advantage of the Identified\_Item class because ISO 11179 requires that all Identified\_Items have at least one Scoped\_Identifier. While this is strictly true in the case of UML, the UML NamedEntity name serves this role and does not need to be exposed to the users unless they intend to export or import from an external system such as ADL. Because of this, we need to create a new relationship between DescribedItem and Scoped\_Identifier with a cardinality of 0..\*.

#### Describable Items in AML

##### Conceptual and Value Domains



**Conceptual and value domain metamodel region**

A Conceptual\_Domain sometimes contains a finite allowed inventory of notions that can be enumerated. Such a Conceptual\_Domain is referred to as an Enumerated\_Conceptual\_Domain.

##### Data Element and Data Element Concept



**Data Element and Data Element Concept**

This diagram shows the AML equivalent for the ISO 11179 Data\_Element and Data\_Element\_Concept element. The first thing to note is that, according to ISO 11179, a Data\_Element is "a unit of data that is *considered in context to be indivisible*". While this covers certain ObjectConstraints, the AML requirements need to assign meaning to both indivisible *and* divisible information artifacts, meaning that the data\_element\_meaning relationship is a subset of ObjectConstraint.about.

The second thing to note is that, from the perspective of ISO 11179, every Data\_Element must be associated with exactly one meaning. From the AML perspective, while every ObjectConstraint possibly *should* be associated with a corresponding (Data Element) concept that provides meaning, it will frequently be the case that such an association will not be available.

### AML Described Items

This section describes how the term\_definitions subsection of the ARCHETYPE\_TERMINOLOGY class is represented in the AML Object Model. The term\_definitions attribute is defined as "Hash <Hash <ARCHETYPE\_TERM, String>, String> [0..1]", where the outer hash key is a language code, e.g. "en", "de" while the inner hash codes are term codes, e.g. "id17", "at4".

ARCHETYPE\_TERM, in turn, is defined as a combination of a *code* with accompanying *text*, *description* and *other\_items*, an arbitrary collection of tag/value pairs.



**AML DescribedItems**

This document shows the AML artifacts that can have entries in the ADL terminology\_definitions section. The ADL equivalent of an ObjectConstraint is identified by an "id" code, an EnumeratedValueDomain as an "ac" code and a PermissibleValue as an "at" code.

Note that the ADL language imposes an additional constraint that there must be an entry for the archetype originalLanguage as well as an entry for each translation language and that there cannot be an entry for any additional languages. The AML specification leaves enforcement of this type of constraint to language specific validation and export tools.

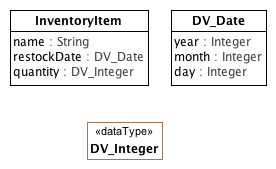
Also note that the ADL specification requires that every entry have an Scoped\_Identifier. Again, this requirement is not necessary if one is modeling strictly with UML and will need to be enforced in a language specific validation/export tool.

# B Reference Model Profile Examples

## BDataBinding Example

The following section is a non-normative example of how the AMLDataType, MappedDataType and DataBinding stereotypes are used to connect an AML Archetype to a Reference Model.

### BSample Reference Model

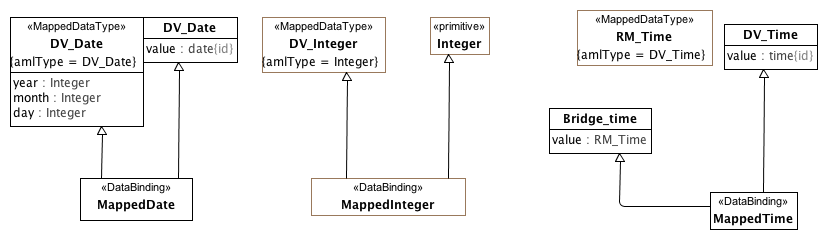


**Sample Reference Model**

The figure above shows a sample reference model. The class, InventoryItem has three attributes:

* *name*: type **String** (a UML **PrimitiveType**)
* *restockDate* : type **DV\_Date** (a **Class**)
* *quantity* : type **DV\_Integer** (a **DataType**)

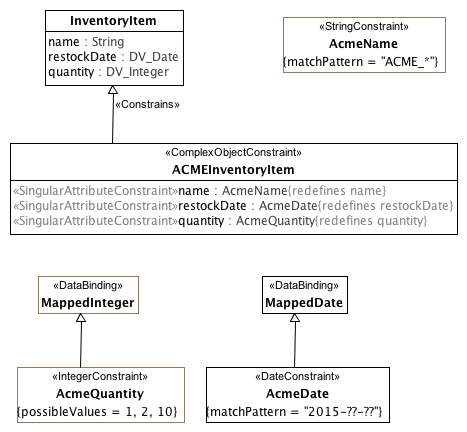
### BSample DataBinding



**Sample Data Binding**

This diagram shows the two data types defined in the Sample Reference Model data types with added MappedDataType stereotype. This identifies them as a mapped representation of an AML primitived type, which means AML constraints cannot be applied to their owned attributes (e.g. *year* cannot be constrained within the **DV\_Date** class because it is is mapped from the AML **Date** *value*.)

### BSample Constraint



**Sample Constraint**

The diagram above shows how attribute constraints are applied to the sample reference model mapping. The first constraint, **AcmeName** is a specialization of the UML PrimitiveType, **String**. As the *name* attribute in the **Inventory** class is of the same type, no data binding is needed -- the **ACMEInventoryItem** constraint can directly redefine it.

In the case of *restockDate* and *quantity*, the AML profile has no knowledge of the **DV\_Integer** and **DV\_Date** types. The **MappedInteger** and **MappedDate** classes allow AML constraints to be applied to the corresponding AML primitive types of **AMLInteger** and **Date** respectively, while simutaneously applying to and allowing redefinition of the **InventoryItem** *restockDate* and *quantity* attributes.

# Appendix A: AML MetaModel