ide

|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | 75 | Date: | May 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 7

1.1. Archetype Modeling Language (AML) Background 7

1.2. AML Intended Users 8

1.3. AML Profiles 8

2. Conformance 9

2.1. Conformance Points 9

2.2. AML Reference Model Profile 9

2.3. AML Terminology Binding Profile 9

2.4. AML Constraint Model Profile 9

3. Normative References 9

4. Terms and Definitions 10

5. Symbols 14

5.1. Graphical Symbols 14

5.2. Abbreviations 14

6. Additional Information 15

6.1. Changes to Adopted OMG Specifications 15

6.2. Acknowledgements 15

7. The AOM and the AML Metamodel 15

8. Specification 15

8.1. Introduction 15

8.2. Scope 16

8.3. Profiles 16

8.3.1. Dependencies 16

8.3.2. ReferenceModelProfile [Profile] 16

8.3.2.1. Infrastructure [Stereotype] 16

8.3.2.2. MappedDataType [Stereotype] 16

8.3.2.3. ReferenceModel [Stereotype] 17

8.3.2.4. Runtime [Stereotype] 18

8.3.3. Terminology Binding Profile 18

8.3.3.1. ArchetypeType [Enumeration] 19

8.3.3.2. about [Stereotype] 20

8.3.3.3. ArchetypeTerm [Stereotype] 20

8.3.3.4. AssumedValue [Stereotype] 21

8.3.3.5. CodeSystemReference [Stereotype] 22

8.3.3.6. CodeSystemVersionReference [Stereotype] 22

8.3.3.7. ConceptReference [Stereotype] 22

8.3.3.8. DescribedIdentifier [Stereotype] 23

8.3.3.9. Entry [Stereotype] 24

8.3.3.10. EnumeratedValueDomain [Stereotype] 24

8.3.3.11. IdentifiedItem [Stereotype] 26

8.3.3.12. IdEntry [Stereotype] 26

8.3.3.13. PermissibleValue [Stereotype] 27

8.3.3.14. PossibleValue [Stereotype] 28

8.3.3.15. ResourceReference [Stereotype] 28

8.3.3.16. ScopedIdentifier [Stereotype] 29

8.3.3.17. TermResourceTranslation [Stereotype] 29

8.3.3.18. ValueSetDefinitionReference [Stereotype] 30

8.3.3.19. ValueSetReference [Stereotype] 30

8.3.4. ConstraintProfile [Profile] 31

8.3.4.1. ArchetypeType [Enumeration] 32

8.3.4.2. Lifecycle\_state [Enumeration] 33

8.3.4.3. VERSION\_STATUS [Enumeration] 34

8.3.4.4. Archetype [Stereotype] 34

8.3.4.5. ArchetypeDefinition [Stereotype] 43

8.3.4.6. ArchetypeLibrary [Stereotype] 43

8.3.4.7. ArchetypeRoot [Stereotype] 44

8.3.4.8. ArchetypeSlot [Stereotype] 47

8.3.4.9. AuthoredResource [Stereotype] 49

8.3.4.10. ComplexObjectConstraint [Stereotype] 52

8.3.4.11. Constrains [Stereotype] 63

8.3.4.12. ObjectConstraint [Stereotype] 63

8.3.4.13. ResourceAnnotationNodeItem [Stereotype] 70

8.3.4.14. ResourceTranslation [Stereotype] 71

9. AML-UML Transformation Reference (Informative) 72

9.1. Introduction 72

9.1.1. AML Provisioning Context 73

9.1.2. QVT Packaging 74

9.1.3. Transformation Reuse and Composition 75

9.1.4. Transformation Notation 77

9.1.5. Platform Binding 78

9.1.6. Global Properties 79

9.2. Archetype Library 80

9.3. Archetype 80

9.4. Terminology Definition 81

9.5. Terminology Binding 83

9.6. Local Value-Sets 84

9.7. Archetype Definition 85

9.8. Object References 86

9.9. Primitive Constraints 88

9.10. Temporal Constraints 89

9.11. Code Constraints 90

9.12. Assertions 91

**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

This section describes the purpose behind the AML Metamodel and how it relates to the AOM. The actual AML Metamodel can be found in Appendix A

# Specification

## Introduction

## Scope

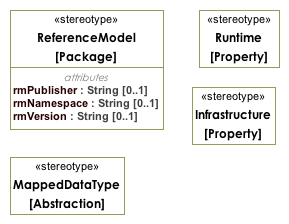
## Profiles

### Dependencies

### ReferenceModelProfile [Profile]

The Reference Model Profile (RMP) enables the specification of reference models, upon which archetypes can be based. The RMP enables the identification of a root package that identifies the set of reference model elements (REM) that can be constrained by a collection of archetype model elements (AME) in an Archetype Library.

The RMP provides the ability to associate the "primitive data types" identified in an instance of a Constraint Model Profile with the corresponding (mapped) data types in the reference model. It also allows properties in the target reference model to be identified as "unconstrainable", because they carry "runtime" provenance and workflow information or because they carry "infrastructure" items such as archetype identifiers, etc.



**Reference Model Profile**

#### Infrastructure [Stereotype]

**Description**

The Infrastructure stereotype indicates that 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 Profile](#_89a4cd915790ea992ed5844704e8ecde)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Property

#### MappedDataType [Stereotype]

**Description**

The MappedDataType stereotype associates a Reference Model Classifier with an AML Primitive Type. AML Primitive types are defined by the UML Type Library and/or the XML Primitive Type Library. The client of the Abstraction is a Reference Model Classifier. The supplier of the Abstraction is an AML Primitive type. The mapping of the Abstraction defines the transformations between the RM Classifier and its AML Primitive Type counterpart. Note that AML constraints are defined with respect to AML Primitive Types even when the Type being constrained is expressed as a Reference Model Type.

**Diagrams**

[Reference Model Profile](#_89a4cd915790ea992ed5844704e8ecde)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Abstraction

**Constraints**

* **isAMLDataType**

The supplier AML Primitive Type must be an AML Primitive Type defined in the UML Primitive Type or XML Primitive Type libraries.

[OCL]

self.base\_Abstraction.supplier->exists(s|s.oclIsKindOf(PrimitiveType) and ((s.namespace.name='XMLPrimitiveTypes') or (s.namespace.name='PrimitiveTypes')))

#### ReferenceModel [Stereotype]

**Description**

The ReferenceModel stereotype identifies the base Package as a *Reference Model* -- a UML model that defines the complex data types and structural patterns that can be constrained by a collection of *Archetypes*.

ReferenceModel also allows the specification of a publisher, namespace and version of a Reference Model in a form compatible with a modeling language such as ADL.

**Diagrams**

[Reference Model Profile](#_89a4cd915790ea992ed5844704e8ecde)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Package

**Attributes**

**•** rmPublisher : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** rmNamespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** rmVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

#### Runtime [Stereotype]

**Description**

The Runtime profile that the base Property represents a dynamic or "runtime" element such as a time stamp, that cannot be constrained using AML.

**Diagrams**

[Reference Model Profile](#_89a4cd915790ea992ed5844704e8ecde)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Property

### Terminology Binding Profile

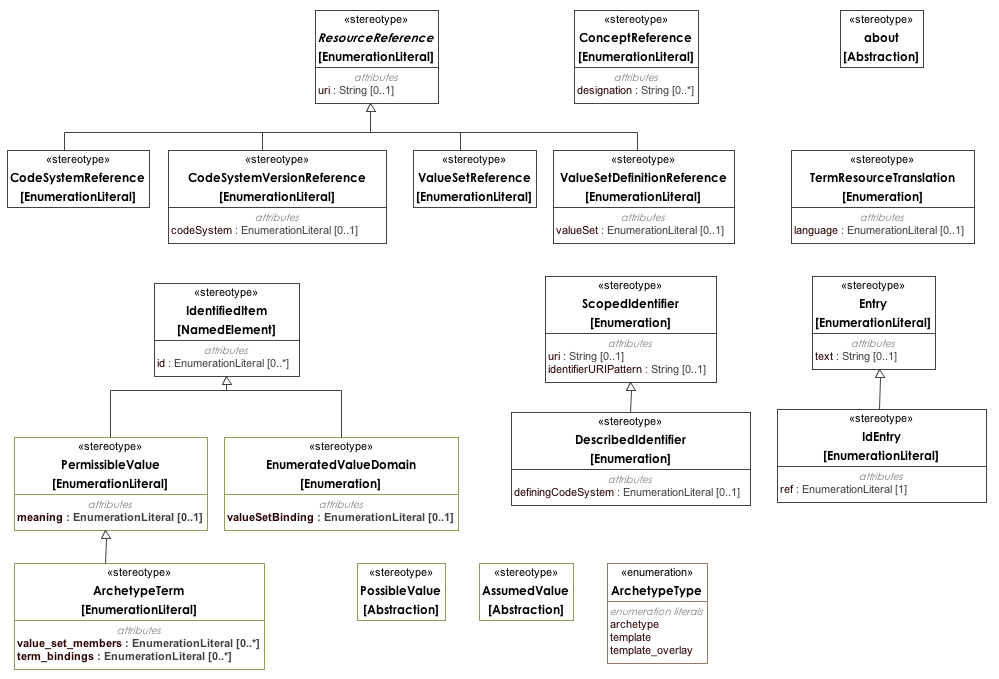
The Terminology Binding Profile supports the binding of information models terminology, with optional support for binding to CTS2. Profile bindings include:

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

The Terminology Binding Profile includes the UML equivalent of the ADL 2.0 terminology section, including:

* Identifiers -- The IdentifiedItem stereotype allows "id", "at" and "ac" identifiers to be assigned to Class constraints, Enumeration Literals and Enumerations respectively.
* Term definitions -- The ResourceTranslation, Entry and IdEntry stereotypes allow language specific text/ description tuples to be assigned directly to model elements (Entry) or indirectly to identified elements
* Term bindings
  + Model elements may be associated with a concept reference in an external terminology using the *about* association, which includes term bindings for ADL 2.0 "id" codes
  + Enumerations may be associated with a value set and optional definition that identifies the list of possible "meanings" that can be associated with the owned enumeration literals, which includes term bindings for ADL "ac" codes
  + Enumeration literals may be associated concept references in an external terminology that define the intended meaning of the enumeration literal in the context of the containing enumeration which includes term bindings for ADL "at" codes

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.



**Terminology Binding Profile**

#### ArchetypeType [Enumeration]

**Description**

An Enumeration which represents the type of an Archetype. The ArchetypeType is the UML representation for Archetype structural variants described by AOM 2.0 in terms of the AOM Archetype attributes is\_template and is\_overlay.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Enumeration Literals**

* **archetype**

The Archetype is structured as a source Archetype.

Source archetypes can be specialised, in which case their definition structure is a partial overlay on the flat parent, or ‘top-level’, in which case the definition structure is complete. C\_ARCHETYPE\_ROOT instances may only occur representing direct references to other archetypes - ‘external references’.

An "archetype" corresponds to an AOM ARCHETYPE in which both is\_template and is\_overlay are false.

* **template**

The Archetype is structured as a Template.

A source template is an archetype containing C\_ARCHETYPE\_ROOT objects representing slot fillers - each referring to an external archetype or template, or potentially an overlay archetype.

Corresponds to an AOM Archetype with is\_template=True.

* **template\_overlay**

The archetype is structured as a template overlay.

These are purely local components of templates, and include only the *definition* and *terminology*. The definition structure is always a specialised overlay on something else, and may not contain any slot fillers or external references, i.e. no C\_ARCHETYPE\_ROOT objects. No identifier, *adl\_version*, *languages* or *description* are required, as they are considered to be propagated from the owning root template.

Accordingly, template overlays act like a simplified specialised archetype. Template overlays

can be thought of as being similar to ‘anonymous’ or ‘inner’ classes in some object-oriented programming languages.

A template\_overlay corresponds to an AOM ARCHETYPE with both is\_template and is\_overlay set to true.

#### about [Stereotype]

**Description**

The about stereotype extends the Abstraction association and associates an AML model element with its intended ontological meaning. The about stereotype models the ISO 11179-3 "meaning" association between a Data Element and a Data Element Concept and a Value Domain and a Conceptual Domain.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Abstraction

**Constraints**

* **isConceptReference**

The supplier (target) of the about stereotype must be stereotyped by ConceptReference

[OCL]

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

#### ArchetypeTerm [Stereotype]

**Description**

The ArchetypeTerm stereotype corresponding to the AOM "ac" code and may define a value set by populating the tag value\_set\_members with the sibling ArchetypeTerms representing AOM "at" codes.

An ArchetypeTerm corresponding to an AOM "at" code may be associated with a term\_binding. A term\_binding is modeled by populating the tag term\_bindings with ConceptReference EnumerationLiterals corresponding to each technology binding associated with that "at" code.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Attributes**

**•** value\_set\_members : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..\*]

When the <<ARCHETYPE\_TERM>> corresponds to an AOM ac-code, value\_set\_members represents value-set relationships. This <<ARCHETYPE\_TERM>> represents an ac-code, the value\_set\_members represent at-codes. All of the at-codes must be in the same Enumeration as the ac-code.

This tag encapsulates the AOM concept of value\_sets within the archetype terminology. The specification of value\_set\_members should only occur in the ARCHETYPE's terminology\_original\_language.

**•** term\_bindings : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..\*]

This tag is used to specify term bindings for the <<ARCHETYPE\_TERM>>. A term binding may be used to specify the "meaning" of the <<ARCHETYPE\_TERM>>, either in the ISO 11179 sense or as a reference to a terminology definition provided by an external service. In either case, the term binding is a <<ConceptReference>> within an Enumeration representing the AOM concept of term\_binding.

**Constraints**

* **VTLC- language consistency**

VTLC- language consistency. Languages consistent: all term codes and constraint codes exist in all languages.

[OCL]

self.base\_EnumerationLiteral.namespace.oclAsType(Enumeration).general->notEmpty() and self.base\_EnumerationLiteral.namespace.oclAsType(Enumeration).general->forAll(language|language.namespace.oclAsType(Package).ownedType ->select(t|(t<>language)and t.stereotypedBy('ResourceTranslation')).oclAsType(Enumeration).\_generalizationOfGeneral.specific.oclAsType(Enumeration) ->forAll(sibling|sibling.ownedLiteral->exists(ol|ol.name=self.base\_EnumerationLiteral.name)) )

#### AssumedValue [Stereotype]

**Description**

AssumedValue associates a ConceptReferenceConstraint with the ConceptReference that is the "assumed value" for the constrained object. An "assumed value" is the default value for an element if it is absent in a data record. Note that "assumed value" is not the same as "default value", which, in this context, is the value that is provided as the default when a new data record is being created.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Abstraction

#### CodeSystemReference [Stereotype]

**Description**

A reference to a code system (aka. "terminology", "classification scheme", "ontology").

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

#### CodeSystemVersionReference [Stereotype]

**Description**

"A reference to a specific version of a code system and, if known, the code system which it is a version of." [cts2]

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Attributes**

**•** codeSystem : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

A reference to the code system that this reference is a version if. The codeSystem tag is used in situations where the code system itself has a well-known URI but there referenced version does not. In this case the version URI can be omitted and the reference used in its place.

#### ConceptReference [Stereotype]

**Description**

ConceptReference is the scoped identifier of a concept. The name of the base EnumerationLiteral is the code (aka. "id", "label" and in the case of CTS2, "name") of the target concept. The scoping namespace is supplied by the owning ScopedIdentifier.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Attributes**

**•** designation : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

A contextually appropriate name or signifier for the referenced concept.

**Constraints**

* **mustBeScopedIdentifier**

The owning enumeration must be stereotyped with ScopedIdentifier

[OCL]

self.base\_EnumerationLiteral.namespace.stereotypedBy('ScopedIdentifier')

* **mustBeCodeSystemOrVersion**

If present, definingCodeSystem must be stereotyped by CodeSystemReference or CodeSystemVersionReference.

[English]

not(self.definingCodeSystem.oclIsUndefined()) implies( self.definingCodeSystem.stereotypedBy('CodeSystemReference') or self.definingCodeSystem.stereotypedBy('CodeSystemVersionReference') )

* **definingCodeSystem**

The EnumerationLiteral referenced by definingCodeSystem must be stereotyped by CodeSystemReference or CodeSystemVersionReference

[OCL]

not(self.definingCodeSystem.oclIsUndefined()) implies( self.definingCodeSystem.stereotypedBy('CodeSystemReference') or self.definingCodeSystem.stereotypedBy('CodeSystemVersionReference') )

#### DescribedIdentifier [Stereotype]

**Description**

The DescribedIdentifier stereotype is a ScopedIdentifier whose ownedLiterals are concept references. DescribedIdentifier includes an optional codeSystem tag that can reference the code system or code system version that describes the owned concept references. The codeSystem of an owned concept reference can be overridden on the concept reference level.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ScopedIdentifier](#_59faf6918f4c546323d6df67392c366b)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Enumeration

**Attributes**

**•** definingCodeSystem : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

The default definingCodeSystem for the owned ConceptReferences.

**Constraints**

* **membersMustBeConceptReference**

All of the member EnumerationLiterals must be stereotyped by ConceptReference.

[English]

self.base\_Enumeration.ownedLiteral->forAll(ol|ol.stereotypedBy('ConceptReference'))

* **definingCodeSystem**

definingCodeSystem, if present, must reference an enumeration literal stereotyped by CodeSystemReference or CodeSystemVersionReference

[OCL]

not(self.definingCodeSystem.oclIsUndefined()) implies( self.definingCodeSystem.stereotypedBy('CodeSystemReference') or self.definingCodeSystem.stereotypedBy('CodeSystemVersionReference') )

#### Entry [Stereotype]

**Description**

A language specific name (text) and optional description of a NamedElement.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Direct Subclasses (Specialization)**

[TerminologyProfile::IdEntry](#_639b489bb158a6e00785ad9ecb38b643)

**Attributes**

**•** text : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

If present, the actual text of the translation. If absent, the text of the translation is the name of the base EnumerationLiteral. The text tag exists to address the situation where two or more elements in the model have the same name in the same language. In this situation, both of the Entry elements cannot have the same name (text), so, instead they are given arbitrary unique names and the text tag carries the actual translation.

#### EnumeratedValueDomain [Stereotype]

**Description**

The EnumeratedValueDomain stereotype 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 reference a value set or value set definition. Implementations may use this reference to validate the PermissibleValue meaning links, populate the permissible values in the EnumeratedValueDomain or provide selection lists for mapping existing mappings.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Enumeration

**Attributes**

**•** valueSetBinding : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

The identifier of the value set or value set definition whose resolution defines the set of possible value meanings for this set of permissible values.

**Constraints**

* **permissibleValues**

All ownedLiterals must be stereotyped by PermissibleValue

[OCL]

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

* **VTVSID- value-set id defined**

VTVSID- value-set id defined. The identifying code of a value set must be defined in the term definitions of the terminology of the current archetype.

[English]

self.base\_Enumeration.ownedLiteral->isEmpty() implies self.base\_Enumeration.clientDependency ->select(d|d.oclIsKindOf(Usage)).supplier ->select(s|s.stereotypedBy('ARCHETYPE\_TERM')).name ->exists(n|n=self.base\_Enumeration.name)

* **VTVSUQ- value-set members unique**

VTVSUQ- value-set members unique. The member codes of a value set must be unique within the value set.

[English]

self.base\_Enumeration.ownedLiteral->isEmpty() implies( self.base\_Enumeration.clientDependency.select->(d|d.oclIsKindOf(Usage)).supplier->select(s|s.stereotypedBy('ARCHETYPE\_TERM'))->asSet()->size() =self.base\_Enumeration.clientDependency.select->(d|d.oclIsKindOf(Usage)).supplier->select(s|s.stereotypedBy('ARCHETYPE\_TERM'))->asSequence()->size() )

* **bindingIsValueSetOrDefinition**

valueSetBinding, if present, must reference an enumeration that is stereotyped by "ValueSetReference" or "ValueSetDefinitionReference"

[OCL]

not(self.valueSetBinding.oclIsUndefined()) implies( self.valueSetBinding.stereotypedBy('ValueSetReference') or self.valueSetBinding.stereotypedBy('ValueSetDefinitionReference') )

#### IdentifiedItem [Stereotype]

**Description**

The IdentifiedItem stereotype allows one or more instances of a ScopedIdentifier to be assigned to a NamedElement.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::NamedElement

**Direct Subclasses (Specialization)**

[TerminologyProfile::EnumeratedValueDomain](#_c7f411daaf64f83e013bec437cb8f30a),

[ConstraintProfile::ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2),

[TerminologyProfile::PermissibleValue](#_5bb7ce8128b60ee5eb2ca275444e9692)

**Attributes**

**•** id : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..\*]

An identifier for the NamedElement.

**Constraints**

* **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.oclAsType(EnumerationLiteral).namespace = l2.oclAsType(EnumerationLiteral).namespace implies l1 = l2))

#### IdEntry [Stereotype]

**Description**

A term / description translation for a NamedElement with an IdentifiedItem stereotype.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::Entry](#_03adbea6748e4e63f69642986c8df42d)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Attributes**

**•** ref : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [1]

A reference to the id tag of a named element that stereotyped by IdentifiedItem.

#### PermissibleValue [Stereotype]

**Description**

A possible value in a data record. A permissible value may be a context specific code (e.g. 0, 1, "M", "A", etc.) a concept identifier (e.g. "74400008", "16285-9"), a URI or, in the case of ADL, an "AT" code. Note that the meaning of the permissible value is assigned by its meaning.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Direct Subclasses (Specialization)**

[TerminologyProfile::ArchetypeTerm](#_2516a2c296345f1369282ef5b8ff5bf1)

**Attributes**

**•** meaning : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

The ConceptReference that provides the meaning for the permissible value

**Constraints**

* **VATDF- value code validity**

VATDF- value code validity. Each value code (at-code) used in a term constraint in the archetype definition must be defined in the term\_definitions part of the terminology of the flattened form of the current archetype.

[English]

(self.base\_EnumerationLiteral.getNearestPackage().name='value\_sets') implies self.base\_EnumerationLiteral.getNearestPackage().nestingPackage.nestedPackage ->select(p|p.name='term\_definitions').ownedType ->select(t|t.oclIsKindOf(Enumeration)).oclAsType(Enumeration) ->forAll(e|e.ownedLiteral->exists(l|l.name=self.base\_EnumerationLiteral.name))

* **VTVSMD- value-set members defined**

VTVSMD- value-set members defined. The member codes of a value set must be defined in the term definitions of the terminology of the flattened form of the current archetype.

[OCL]

(self.base\_EnumerationLiteral.getNearestPackage().name='value\_sets') implies self.base\_EnumerationLiteral.getNearestPackage().nestingPackage.nestedPackage ->select(p|p.name='term\_definitions').ownedType ->select(t|t.oclIsKindOf(Enumeration)).oclAsType(Enumeration) ->forAll(e|e.ownedLiteral->exists(l|l.name=self.base\_EnumerationLiteral.name))

* **mustBeConceptReference**

meaning, if present, must reference an EnumerationLiteral that is stereotyped by ConceptReference

[OCL]

not(self.meaning.oclIsUndefined()) implies self.meaning.stereotypedBy('ConceptReference')

#### PossibleValue [Stereotype]

**Description**

A link to a ConceptReference that supplies a possible (allowed) value for a ConceptReferenceConstraint.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Abstraction

#### ResourceReference [Stereotype]

**Description**

ResourceReference couples a local identifier with an optional URI which references the target resource. ResourceReference models the CTS2 [cts2] NameAndMeaningReference data type, where the domain is determined by the specializing stereotype and the name by the name of the base EnumerationLiteral. The CTS2 href attribute is not part of ResourceReference as it is an aspect of a service instance, not a model.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Direct Subclasses (Specialization)**

[TerminologyProfile::CodeSystemReference](#_7de70b4fbd9e6a164f7f00cde47dfd5a),

[TerminologyProfile::CodeSystemVersionReference](#_9a8de95c38ebe2d6ce506cbc9bef7b7a),

[TerminologyProfile::ConceptReference](#_57ae94153b82f28889d42ad4aa8fe1e0),

[TerminologyProfile::ValueSetDefinitionReference](#_a4fedb7858ead8d2272640d51b53719a),

[TerminologyProfile::ValueSetReference](#_1a1ca20b54028ee5e2eb20af35411f6e)

**Attributes**

**•** uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

The URI of the referenced resource. "A globally unique URI that identifies the intended meaning of the identifier." [cts2]

**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))

#### ScopedIdentifier [Stereotype]

**Description**

The ScopedIdentifier stereotype models both the ISO 11179-3[iso11179] namespace, "... a set of designations and/or scoped identifiers for a particular business need" and Scoped\_Identifier, "the identifier of an identified item within a specified namespace". ScopedIdentifier extends ENumeration, where Enumeration plays the role of the scoping namespace and the owned EnumerationLiterals the contained identifiers. A ScopedIdentifier may include an optional URI that identifies the scoping namespace and, if necessary, a uri pattern that defines how uri's for the contained identifiers are constructed.

As an example, the SNOMED CT identifier namespace would have a uri of "http://snomed.info/id/", indicating that an EnumerationLiteral named 74400008 would be represented as "http://snomed.info/id/74400008".

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Enumeration

**Direct Subclasses (Specialization)**

[TerminologyProfile::DescribedIdentifier](#_7935aa90cdd719e5718a690fd5dd0137)

**Attributes**

**•** uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

The URI of the namespace. As an example, one might have a ScopedIdentifier named "owl" with a URI of "http://www.w3.org/2002/07/owl#".

**•** identifierURIPattern : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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. If no URI substitution pattern is supplied, URI's are assumed to be constructed by concatenating the name of an enumeration literal onto the value of the *uri* attribute.

#### TermResourceTranslation [Stereotype]

**Description**

ResourceTranslation is a collection of designations and descriptions/definitions in a target language. ResourceTranslation represents a refactoring of the ISO 11179-3[iso11179] Designatable\_Item, where Designation sign is represented as the name of the extended EnumerationLiteral and the Definition text as the associated comment(s). The language attribute is represented by the language tag.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Enumeration

**Attributes**

**•** language : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

The identifier of the target language.

**Constraints**

* **translationEntries**

All of the ownedLiterals must be stereotyped by Entry.

[OCL]

self.base\_Enumeration.ownedLiteral->forAll(ol|ol.stereotypedBy('Entry'))

* **uniqueEntries**

The ref tags of the ownedLiterals must all be unique. No two translation entries may reference the same identifier.

[OCL]

self.base\_Enumeration.ownedLiteral->size() = self.base\_Enumeration.ownedLiteral.appliedStereotype('IdEntry').oclAsType(IdEntry).ref->asSet()->size()

#### ValueSetDefinitionReference [Stereotype]

**Description**

"A reference to a set of rules for constructing a value set along with the corresponding value set if known" [cts2]

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Attributes**

**•** valueSet : UML Standard Profile::UML2 Metamodel::EnumerationLiteral [0..1]

The reference to the value set that is defined by this value set definition. This tag reduces the requirement to assign unique uri's to each value set definition. If the definition itself has a uri, the valueSet tag link can provide sufficient information to get a known definition without having to generate a new uri.

#### ValueSetReference [Stereotype]

**Description**

"A reference to a named set of entity references." [cts2] ValueSetReference references a set of ConceptReferences. The members of the set can vary over time and context and depend on (a) the particular value set definition (aka. version) of the value set d and (b) the particular version of the code system(s) that are used to to resolve the rules in the value set definition.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Direct Superclasses (Generalization)**

[TerminologyProfile::ResourceReference](#_1b2eec63ad4ef6c72d57b9985e0346ff)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::EnumerationLiteral

**Constraints**

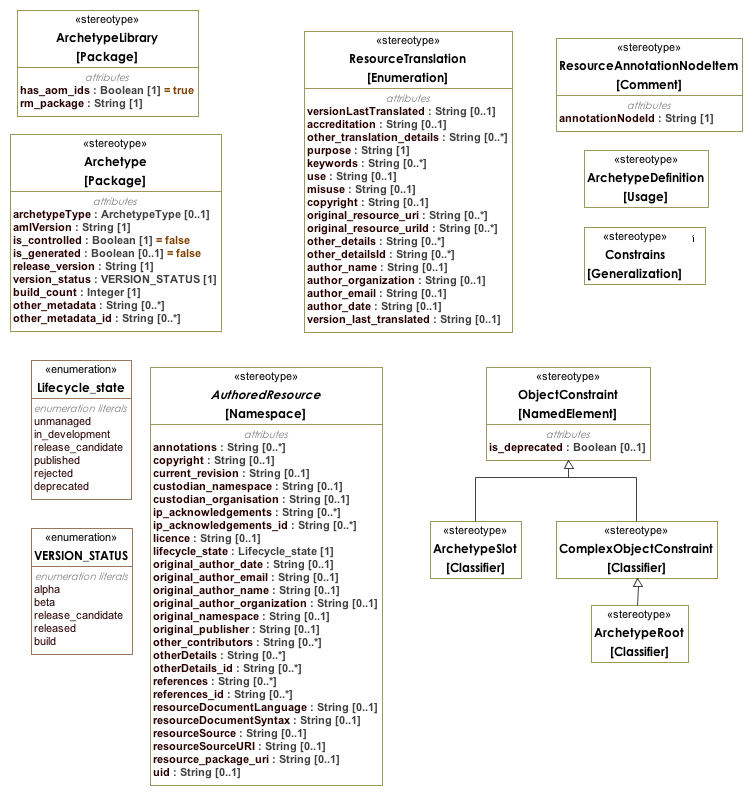
* **definition**

This Enumeration must have exactly one > Generalization.

[OCL]

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

### ConstraintProfile [Profile]



**Constraint Profile**

#### ArchetypeType [Enumeration]

**Description**

An Enumeration which represents the type of an Archetype. The ArchetypeType is the UML representation for Archetype structural variants described by AOM 2.0 in terms of the AOM Archetype attributes is\_template and is\_overlay.

**Diagrams**

[Terminology Binding Profile](#_b3fed7a67275752ffa266629e157b6c0)

**Enumeration Literals**

* **archetype**

The Archetype is structured as a source Archetype.

Source archetypes can be specialised, in which case their definition structure is a partial overlay on the flat parent, or ‘top-level’, in which case the definition structure is complete. C\_ARCHETYPE\_ROOT instances may only occur representing direct references to other archetypes - ‘external references’.

An "archetype" corresponds to an AOM ARCHETYPE in which both is\_template and is\_overlay are false.

* **template**

The Archetype is structured as a Template.

A source template is an archetype containing C\_ARCHETYPE\_ROOT objects representing slot fillers - each referring to an external archetype or template, or potentially an overlay archetype.

Corresponds to an AOM Archetype with is\_template=True.

* **template\_overlay**

The archetype is structured as a template overlay.

These are purely local components of templates, and include only the *definition* and *terminology*. The definition structure is always a specialised overlay on something else, and may not contain any slot fillers or external references, i.e. no C\_ARCHETYPE\_ROOT objects. No identifier, *adl\_version*, *languages* or *description* are required, as they are considered to be propagated from the owning root template.

Accordingly, template overlays act like a simplified specialised archetype. Template overlays

can be thought of as being similar to ‘anonymous’ or ‘inner’ classes in some object-oriented programming languages.

A template\_overlay corresponds to an AOM ARCHETYPE with both is\_template and is\_overlay set to true.

#### Lifecycle\_state [Enumeration]

**Description**

The lifecycle\_state is an important property of an archetype, which is used to record its state in a defined lifecycle. The lifecycle state machine and versioning rules are explained fully in the openEHR Knowledge Artefact Identification specification[openEHRkai]. Here we simply note that the value of the property is a coded termcorresponding to one of the macro-state names on Figure 4 , i.e. ‘unmanaged’, ‘in\_development’, and so on.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Enumeration Literals**

* **deprecated**

An artifact that is obsolete, suspended or withdrawn from active use

* **in\_development**
* **published**
* **rejected**

An artifact that is withdrawn from development prior to being published.

* **release\_candidate**
* **unmanaged**

An artefact with no recognised owner organization.

#### VERSION\_STATUS [Enumeration]

**Description**

Status of this version, as one of a number of possible values: uncontrolled, prerelease, release, build.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Enumeration Literals**

* **alpha**

Value representing a version which is ‘unstable’, i.e. contains an unknown size of change with respect to its base version. Rendered with the build number as a string in the form “N.M.P-alpha.B” e.g. “2.0.1-alpha.154”.

* **beta**

Value representing a version which is ‘beta’, i.e. contains an unknown but reducing size of change with respect to its base version. Rendered with the build number as a string in the form “N.M.P-beta.B” e.g. “2.0.1-beta.154”.

* **build**

Value representing a version which is a build of the current base release. Rendered with the build number as a string in the form “N.M.P+B” e.g. “2.0.1+33”.

* **released**

Value representing a version which is ‘released’, i.e. is the definitive base version. Rendered with the build number as a string in the form “N.M.P” e.g. “2.0.1”.

* **release\_candidate**

Value representing a version which is ‘release candidate’, i.e. contains only patch-level changes on the base version. Rendered as a string as “N.M.P-rc.B” e.g. “2.0.1-rc.27”.

#### Archetype [Stereotype]

**Description**

The Archetype stereotype extends a Package that contains a

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**.

The name of the **Archetype** package is the physical\_id of the Archetype.

The physical\_id of an Archetype is derived from the [rm\_publisher], [rm\_closure], [rm\_class], [concept\_id], [release\_version], [version\_status], and [build\_count] as follows:

[rm\_publisher]-[rm\_closure]-[rm\_class].[concept\_id].v[release\_version]-[version\_status].[build\_count]

where :

* rm\_publisher. This is derived from the <<ReferenceModel>> rmPublisher tag, where <<ReferenceModel>> is imported by the containing <<ArchetypeLibrary>>.
* rm\_closure. This is derived from the <<ArchedtypeLibrary>> rm\_package tag. The <<ReferenceModel>> Package defines the reachability graph where the *rm\_class* class is found (there can be more than one possibility in many reference models).
* rm\_class. Name of the root class of this archetype. Derived from the reference model class extended by the definition (<<ArchetypeDefinition>>) of this <<Archetype>>.
* concept\_id. The short concept name of the archetype as used in the multi-axial *archetype\_hrid*. Derived from the name of the underlying Package.
* release\_version. The full numeric version of this archetype consisting

of 3 parts, e.g. 1.8.2. The *archetype\_hrid* feature includes only the major

version.

* version\_status. The status of the version.
* build\_content. The build count since last increment of any version part.

The URI of the **Archetype** Package represents the Archetype namespace.

An **Archetype** may import a parent **Archetype**.

An **Archetype** has exactly one <<ArchetypeDefinition>> Usage to identify the top level <<ComplexObjectConstraint>> Classifier which defines this <<Archetype>>.

An **Archetype** owns a terminology. In UML, terminology is modeled as a nested Package with the name "ontology". Within the "ontology" Package, there are nested packages representing the containment of terminology concepts in AOM:

term\_definitions

term\_bindings

value\_sets

A value\_sets is represented as an <<EnumeratedValueDomain>> Enumeration. The members of the value\_sets are represented as <<PermissibleValue>> EnumerationLiterals. The name of the Enumeration (an ac\* name) must exist as an <<ARCHETYPE\_TERM>> EnumerationLiteral in each of the term\_definitions languages. The name of each EnumerationLiteral must similarly exist as an <<ARCHETYPE\_TERM>> EnumerationLiteral in each of the term\_definitions languages.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Direct Superclasses (Generalization)**

[ConstraintProfile::AuthoredResource](#_13ad6987e15b787d385f0b30ff25d6c9)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Package

**Attributes**

**•** archetypeType : TerminologyProfile::ArchetypeType [0..1]

The implementation specific type or classification of the archetype. One of archetype, template or template\_overlay.

**•** amlVersion : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [1]

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

**•** is\_controlled : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean [1]

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

If false, the Archetype is an *ad hoc*, uncontrolled artifact, not formally associated with any organisation, typical for experimental archetypes, and pre-ADL 1.5 archetypes not yet upgraded to have a namespace. If true, then the archetype must include a namespace which denotes the original authoring organization.

**•** is\_generated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean [0..1]

A flag indicating whether the archetype was generated (is\_generated = true) or authored (is\_generated = false). This marker is used to support the migration to differential archetype representation introduced in ADL 1.5, to enable proper representation of specialized archetypes.

**•** release\_version : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [1]

The full numeric version of this archetype. In ADL, this identifier will consist of 3 parts, the major, the minor and the patch version.

**•** version\_status : ConstraintProfile::VERSION\_STATUS [1]

The status of the version.

**•** build\_count : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Integer [1]

IThe build count since last increment of any version part.

**•** other\_metadata : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** other\_metadata\_id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**Constraints**

* **mustBeOwned**

Every Archetype Package must be owned by a Package with an ArchetypeLibrary stereotype.

[OCL]

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

* **ownsObjectConstraints**

All of the NamedElements owned by an Archetype Package must be ObjectConstraints.

[OCL]

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

* **currentVersion**

Every Archetype Package must have exactly one clientDependency which is an ArchetypeDefinition and whose supplier is a ComplexObjectConstraint.

***// Note: Original documentation said "within the same <<Archetype>> package" but the constraint doesn't state this.***

***// Note: What does this have to do with "currentVersion", which is the constraint name***

[OCL]

self.base\_Package.clientDependency->select(x|x.stereotypedBy('ArchetypeDefinition')and x.supplier->exists(s|s.stereotypedBy('ComplexObjectConstraint')))->size()=1

* **specializesArchetype**

If an Archetype specializes another Archetype the ArchetypeDefinition supplier of both Archetypes must be the same. In other words, both Archetypes must constrain the same "root class" in the reference model.

[OCL]

self.base\_Package.packageImport.importedPackage->select(p|p.stereotypedBy('Archetype')).clientDependency->select(t|t.stereotypedBy('ArchetypeDefinition')).supplier.oclAsType(Classifier).general ->forAll(x|self.base\_Package.clientDependency->select(t|t.stereotypedBy('ArchetypeDefinition')).supplier.oclAsType(Classifier).general->includes(x))

* **VATDA- value set assumed value code validity**

VATDA- value set assumed value code validity Each value code (at-code) used as an assumed\_value for a value set in a term constraint in the archetype definition must exist in the value set definition in the terminology for the identified value set.

[English]

An AOM term constraint is represented in UML as a Property whose type corresponds to the value set Enumeration. The assumed\_value is represented in UML as the default value for the Property. Based on UML constraints, the default value for the Property must exist as an EnumerationLiteral (member) owned by the value set Enumeration.

* **VETDF- external term validity**

VETDF- external term validity Each external term used within the archetype definition must exist in the relevant terminology (subject to tool accesibility; codes for inaccessible terminologies should be flagged with a warning indicating that no verification was possible)

[English]

Each external term used within the archetype definition must exist in the relevant terminology (subject to tool accesibility; codes for inaccessible terminologies should be flagged with a warning indicating that no verification was possible)

* **VOKU- object key unique**

VOKU- object key unique Within any keyed list in an archetype, including the desription, terminology, and annotations sections, each item must have a unique key with respect to its siblings.

[English]

This AOM validation rule is enforced by the UML constraint that names within a namespace must be unique.

* **VARDT- archetype definition typename validity**

VARDT- archetype definition typename validity The typename mentioned in the outer block of the archetype definition section must match the type mentioned in the first segment of the archetype id.

[English]

In AML, the typename in the archetype id is derived from the name of the constrained RM Type in the outer block of the archetype definition section.

* **VRANP- annotation path valid**

VRANP- annotation path valid Each path mentioned in an annotation within the annotations section must either be a valid archetype path, or a ‘reference model’ path, i.e. a path that is valid for the root class of the archetype.

[English]

Each path mentioned in an annotation within the annotations section must either be a valid archetype path, or a ‘reference model’ path, i.e. a path that is valid for the root class of the archetype.

* **VRRLP- rule path valid**

VRRLP- rule path valid Each path mentioned in a rule in the rules section must be found within the archetype, or be an RM-valid extension of a path found within the archetype.

[English]

Each path mentioned in a rule in the rules section must be found within the archetype, or be an RM-valid extension of a path found within the archetype.

* **VARID- archetype identifier validity**

VARID- archetype identifier validity Apply to ARCHETYPE objects for which is\_overlay = False: The archetype must have an identifier that conforms to the openEHR specification for archetype identifiers.

[OCL]

self.release\_version.match('[0-9]\*(\\.[0-9]\*(\\.[0-9]\*)?)?')

* **VDEOL- original language specified**

VDEOL- original language specified apply to ARCHETYPE objects for which is\_overlay = False: An original\_language section containing the meta-data of the original authoring language must exist

[OCL]

self.base\_Package.clientDependency ->select(d|d.oclIsKindOf(Usage) and (d.name='original\_language')).supplier ->select(s|s.stereotypedBy('ResourceTranslation')) ->notEmpty()

* **VARD- description specified**

VARD- description specified apply to ARCHETYPE objects for which is\_overlay = False: A description section containing the main meta-data of the archetype must exist.

[OCL]

(self.archetypeType.repr()<>'template\_overlay') implies self.base\_Package.clientDependency.supplier->select(d|d.stereotypedBy('ResourceTranslation'))->notEmpty()

* **VASID- archetype specialisation parent identifier validity**

VASID- archetype specialisation parent identifier validity apply to specialised archetypes. The archetype identifier stated in the specialise clause must be the identifier of the immediate specialisation parent archetype.

[English]

The AOM attribute parent\_archetype\_id is represented in AML-UML as a packageImport from the <<Archetype>> Package to the supplier <<Archetype>> Package. During provisioning of AOM artifacts, the parent\_archetype\_id property of AUTHORED\_RESOURCE is provisioned as a property whose type is derived from the importedPackage (importedPackage::Pacomputed property the UML importPackage apply to specialised archetypes.   The archetype identifier stated in the specialise clause must be the identifier of the immediate specialisation parent archetype.

* **VALC- archetype language conformance**

VALC- archetype language conformance apply to specialised archetypes. The languages defined in a specialised archetype must be the same as or a subset of those defined in the flat parent.

[OCL]

self.base\_Package.packageImport.importedPackage ->forAll(superArch|self.base\_Package.clientDependency.supplier->select(s|s.stereotypedBy('ResourceTranslation')).name ->forAll(language|superArch.clientDependency.supplier->select(s|s.stereotypedBy('ResourceTranslation')).name->exists(n|n=language)) )

* **VACSD- archetype concept specialisation depth**

VACSD- archetype concept specialisation depth apply to specialised archetypes. The specialisation depth of the concept code must be one greater than the specialisation depth of the parent archetype.

[English]

This AOM Validation Rule is definitional for AML-UML, the specialisation\_depth is implicitly derived from the number of parent-archetype specializations above the current archetype.

* **VATCD- archetype code specialisation level validity**

VATCD- archetype code specialisation level validity apply to specialised archetypes. Each archetype term (‘at’ code) and constraint code (‘ac’ code) used in the archetype definition part must have a specialisation level no greater than the specialisation level of the archetype.

[English]

apply to specialised archetypes.   Each archetype term (‘at’ code) and constraint code (‘ac’ code) used in the archetype definition part must have a specialisation level no greater than the specialisation level of the archetype.

* **VTSD- specialisation level of codes**

VTSD- specialisation level of codes Term or constraint code defined in archetype terminology must be of the same specialisation level as the archetype (differential archetypes), or the same or a less specialised level (flat archetypes).

[English]

Term or constraint code defined in archetype terminology must be of the same specialisation level as the archetype (differential archetypes), or the same or a less specialised level (flat archetypes).

* **VTTBK- terminology term binding key valid**

VTTBK- terminology term binding key valid Every term binding must be to either a defined archetype term (‘at-code’) or to a path that is valid in the flat archetype

[English]

Every term binding must be to either a defined archetype term (‘at-code’) or to a path that is valid in the flat archetype

* **VTCBK- terminology constraint binding key valid**

VTCBK- terminology constraint binding key valid Every constraint binding must be to a defined archetype constraint code (‘ac-code’).

[English]

This AOM validation rule is obsolete. Not applicable to AML-UML.

* **Original\_language\_valid**

original\_language /= void and language /= Void and then code\_set (Code\_set\_id\_languages).has\_code (original\_language)

[OCL]

self.base\_Package.clientDependency->select(d|d.oclIsKindOf(Usage) and (d.name='original\_language')).supplier ->exists(e|e.stereotypedBy('ResourceTranslation'))

* **Concept\_valid**

terminology.has\_term\_code (concept\_code)

[OCL]

self.base\_Package.clientDependency ->exists(d|d.stereotypedBy('ArchetypeDefinition') and d.supplier->exists(s|s.appliedStereotypeInstance.oclAsType(ComplexObjectConstraint).id->notEmpty()))

* **Definition\_exists**

definition /= Void

[OCL]

self.base\_Package.clientDependency ->exists(d|d.stereotypedBy('ArchetypeDefinition') )

* **Terminology\_exists**

terminology /= Void

[OCL]

self.base\_Package.nestedPackage.nestedPackage.ownedType->exists(o|o.stereotypedBy('ResourceTranslation'))

* **Specialisation\_validity**

is\_specialised **implies** specialisation\_depth > 0

[English]

In UML, is\_specialised is derived and will be consistent with specialisation\_depth, which is also derived.

* **Rules\_valid**

rules /= Void **implies not** rules.is\_empty

[English]

In UML, if rules is empty, it implies there are no rules.

* **Rm\_publisher\_validity**

rm\_publisher /= Void ***and then not***

rm\_publisher.is\_empty

[English]

In AML, Rm\_publisher is derived from the name of the Package and will always be present if the name itself is valid (see VARID rule).

* **Rm\_package\_validity**

rm\_package /= Void ***and then not*** rm\_package.is\_empty

[English]

rm\_package is derived from the package name, so will always be present if the name is valid (see also rule VARID).

* **Rm\_class\_name\_validity**

rm\_class /= Void ***and then not*** rm\_class.is\_empty

[English]

rm\_class is derived from the package name and will always exist if the name is correct (see VARID rule).

* **Concept\_id\_validity**

concept\_id /= Void ***and then not*** concept\_id.is\_empty

[English]

concept\_id is derived from the Package name and will exist if the name conforms to the VARID rule.

* **Base\_version\_validity**

release\_version /= Void ***and then not***

release\_version.is\_empty

[English]

release\_version is derived from the Package name and will not be empty if the name conforms to rule VARID.

* **Is\_specialised**

***Is\_specialised***

[English]

In AML, Is\_specialised is derived.

* **Component\_terminologies\_existence**

component\_terminologies /= Void

[English]

component\_terminologies is a implicit, derived property in AML. The value will not be empty when the Archetype is an operational template.

* **original\_language\_validity**

Every ArchetypePackage has at least one Usage association named "terminology\_original\_language" whose supplier is a ResourceTranslation.

[OCL]

self.base\_Package.clientDependency->select(d|d.oclIsKindOf(Usage) and (d.name='terminology\_original\_language')).supplier ->exists(e|e.stereotypedBy('ResourceTranslation'))

* **concept\_code\_validity**

***// This appears to be almost the same constraint as the "currentVersion" constraint below. It certainly has nothing to do with concept\_code\_validity???***

[OCL]

self.base\_Package.clientDependency ->select(d|d.stereotypedBy('ArchetypeDefinition')).supplier->exists(s|s.appliedStereotypeInstance.oclAsType(ComplexObjectConstraint).id->notEmpty())

* **term\_definitions\_validity**

The AOM term\_definitions property on an ARCHETYPE\_TERMINOLOGY is represented in AML as a set of ResourceTranslation Enumerations, which must be indirectly contained by the Archetype Package. Exactly one of the ResourceTranslation Enumerations must be referenced by an original\_language Usage from the Archetype Package.

[OCL]

self.base\_Package.clientDependency->select(d|d.oclIsKindOf(Usage) and (d.name='original\_language')).supplier ->exists(e|e.stereotypedBy('ResourceTranslation'))

* **term\_bindings\_validity**

The individual term definition entries may be referenced by ArchetypeTerm EnumerationLiterals.

The presence of a term\_bindings element within any ArchetypeTerm establishes the existence of term bindings for the entire Archetype Package.

[English]

 The individual term definitions may be referenced by <<ARCHETYPE\_TERM>> EnumerationLiterals.   The presence of a term\_bindings within any <<ARCHETYPE\_TERM>> establishes the existence of term bindings for the <<Archetype>>.

* **Parent\_archetype\_valid**

parent\_archetype /= Void **and then**

parent\_archetype.terminology = Current

The AOM concept of ARCHETYPE\_TERMINOLOGY is represented in AML as a nestedPackage of the <<Archetype>> Package. The nested Package is named "ontology".

[OCL]

self.base\_Package.nestedPackage->exists(p|p.name='ontology')

* **code\_valid**

code /= void **and then not** code.is\_empty

[OCL]

not(self.base\_EnumerationLiteral.name.oclIsUndefined()) and(self.base\_EnumerationLiteral.name<>'')

* **text\_valid**

text\_valid. text /= void

[English]

(self.base\_EnumerationLiteral.namespace.name<>'IdentifierDefinition') implies not(self.text.oclIsUndefined())

* **description\_valid:**

description /= void Every term definition must have a description

[English]

(self.base\_EnumerationLiteral.namespace.name<>'IdentifierDefinition') implies not(self.description.oclIsUndefined())

* **Id\_valid**

id /= void **and then** is\_valid\_value\_set\_code (id)

An AOM VALUE\_SET is modeled in AML as an <<ARCHETYPE\_TERM>> which has values in the value\_set\_members tag. The elements within value\_set\_members must each be a sibling <<ARCHETYPE\_TERM>> within the same Enumeration.

[English]

An AOM VALUE\_SET is modeled in AML as an <<ARCHETYPE\_TERM>> which has values in the value\_set\_members tag.  The elements within value\_set\_members must each be a sibling <<ARCHETYPE\_TERM>> within the same Enumeration.

* **Members\_valid**

members /= void and then not members.is\_empty and then members.

for\_all (a\_member: String | is\_valid\_value\_code (a\_member))

An AOM VALUE\_SET is modeled in AML as an <<ARCHETYPE\_TERM>> which has values in the value\_set\_members tag. The elements within value\_set\_members must each be a sibling <<ARCHETYPE\_TERM>> within the same Enumeration.

[English]

  An AOM VALUE\_SET is modeled in AML as an <<ARCHETYPE\_TERM>> which has values in the value\_set\_members tag.  The elements within value\_set\_members must each be a sibling <<ARCHETYPE\_TERM>> within the same Enumeration.

#### ArchetypeDefinition [Stereotype]

**Description**

The ArchetypeDefinition stereotype associates a ComplexObjectConstraint supplier with an Archetype client, indicating that the ComplexObjectConstraint is the root definition of the client Archetype. ArchetypeDefinition corresponds to the definition association between ARCHETYPE and C\_COMPLEX\_OBJECT in the AOM 2.0 model [aom].

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Usage

#### ArchetypeLibrary [Stereotype]

**Description**

The ArchetypeLibrary stereotype indicates that the base Package contains a collection of archetypes that constrain classes contained in an imported ReferenceModel.

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.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Package

**Attributes**

**•** has\_aom\_ids : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean [1]

If true, then all Archetypes in this ArchetypeLibrary use the IdentifiedItem id to identify the EnumerationLiterals which provide AOM term definitions. The term definitions themselves must be stereotyped with the IdEntry (vs. Entry) stereotype. When has\_aom\_ids is true, all id values (i.e. the "id", "ac" and "at" codes) must be preserved within the model itself, as their values may be referenced outside of the scope of the current model.

If false, IdentifiedItem id's are not required, as references are to the UML artifacts themselves. The simpler Entry stereotype may be used for resource translations which allows translations to reference the translated object directly. Note that id's may still be used in this situation, but they will not be required.

**•** rm\_package : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [1]

Name of the package in whose reachability graph the "root" or "primogenitor" class is found (there can be more than one possibility in many reference models).

**Constraints**

* **oneReferenceModel**

The must be exactly one packageImport of a Package stereotyped as a ReferenceModel.

[OCL]

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

* **onlyArchetypes**

All packaged elements must be Archetypes.

[OCL]

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

#### ArchetypeRoot [Stereotype]

**Description**

ArchetypeRoot is a specialization of ComplexObjectConstraint.

A specialisation of <<ComplexObjectConstraint>> whose node\_id attribute is an archetype identifier rather than the normal internal node code (i.e. at-code).

Used in two situations. The first is to represent an ‘external reference’ to an archetype from within another archetype or template. This supports re-use. The second use is within a template, where it is used as a slot-filler.

For a new external reference, the node\_id is set in the normal way, i.e. with a new code for the specialisation level of the archetype.

For a slot-filler or a redefined external reference, the node\_id is set to a specialised version of the node\_id of the node being specialised, allowing matching to occur during flattening.

In all uses within source archetypes and templates, the children attribute is Void.

In an operational template, the node\_id is converted to the archetype\_ref, and the structure contains the result of flattening any template overlay structure and the underlying flat archetype.

<<C\_ARCHETYPE\_ROOT>> stands for the root node of an archetype; it enables another archetype to be referenced from the present one. Used in both archetypes and templates.

<<C\_ARCHETYPE\_ROOT>> may only occur representing direct references to other archetypes - ‘external references’.

A source template is an archetype containing <<C\_ARCHETYPE\_ROOT>> objects representing slot fillers - each referring to an external archetype or template, or potentially an overlay archetype.

The root node of an operational template, along with every archetype/template root node within, is represented using a <<C\_ARCHETYPE\_ROOT>> object.

A <<C\_ARCHETYPE\_ROOT>> node can be redefined by:

A <<C\_ARCHETYPE\_ROOT>>, where the archetype\_ref of the redefining node is a specialisation of that mentioned in the parent node.

One or more <<C\_ARCHETYPE\_ROOT>> nodes taken together, can be considered to (re-)define a ‘filled’ version of an <<ARCHETYPE\_SLOT>>.

AML Represents the reference to a <<ComplexObjectConstraint>> in another <<Archetype>> as a <<constrains>> Generalization, where the specific is this <<C\_ARCHETYPE\_ROOT>> and the general is the external <<ComplexObjectConstraint>>.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Direct Superclasses (Generalization)**

[ConstraintProfile::ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Classifier

**Constraints**

* **VARXNC- external reference node identifier validity**

If the reference object is a redefinition of either a slot node, or another external reference node, the node\_id of the object must conform to (i.e. be the same or a child of) the node\_id of the corresponding parent node.

Note that the AML-UML transformation to AOM may coerce the node\_id of provisioned C\_ARCHETYPE\_ROOT to a value in conformance with the AOM Validation Rule.

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ArchetypeRoot')or g.stereotypedBy('ArchetypeSlot')) implies self.id.oclAsType(EnumerationLiteral) ->forAll(e|e.name.startsWith(self.base\_Classifier.general.appliedStereotypeInstance.oclAsType(ObjectConstraint).id.oclAsType(EnumerationLiteral).name->asSequence()->first()))

* **VARXTV- external reference type validity**

The reference model type of the reference object archetype identifier must be identical, or conform to the type of the slot, if there is one, in the parent archetype, or else to the reference model type of the attribute in the flat parent under which the reference object appears in the child archetype.

[English]

The AOM Validation Rule is enforced by UML semantics for inheritance, since the <<ArchetypeRoot>> inherits from the <<ArchetypeSlot>> in the archetype parent archetype

* **VARXS- external reference conforms to slot**

The archetype reference must conform to the archetype slot constraint of the flat parent and be of a reference model type from the same reference model as the current archetype.

[English]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ArchetypeSlot')) implies self.base\_Classifier.general.ownedRule->select(r|r.constrainedElement=r.namespace).specification->select(e|e.oclIsKindOf(Expression)).oclAsType(Expression) ->forAll(ex| ((ex.symbol='includes') and ex.operand->exists(o|o.satisfiesAssertion(self.base\_Classifier))) or ((ex.symbol='excludes') and not(ex.operand->exists(o|o.satisfiesAssertion(self.base\_Classifier)))) ) [Need to define/implement satisfiesAssertion to specify above as OCL] rules apply to C\_ARCHETYPE\_ROOT objects:   the archetype reference must conform to the archetype slot constraint of the flat parent and be of a reference model type from the same reference model as the current archetype.

* **VARXR- external reference refers to resolvable artefact**

The archetype reference must refer to an artefact that can be found in the current repository.

[English]

In AML-UML, the archetype reference is modeled using inheritance, which enforces that the referenced artefact is accessible.

* **VARXID- external reference slot filling id validity**

An external reference node defined as a filler for a slot in the parent archetype must have a node id that is a specialization of that of the slot.

Note that the AML-UML transformation to AOM may coerce the node\_id of provisioned C\_ARCHETYPE\_ROOT to a value in conformance with the AOM Validation Rule.

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ArchetypeRoot')or g.stereotypedBy('ArchetypeSlot')) implies self.id.oclAsType(EnumerationLiteral) ->forAll(e|e.name.startsWith( self.base\_Classifier.general.appliedStereotypeInstance.oclAsType(ObjectConstraint).id.oclAsType(EnumerationLiteral).name->asSequence()->first() ) )

* **Archetype\_ref\_validity**

The base Classifier must directly specialize an ObjectConstraint that acts a supplier (i.e. defines) an Archetype. An ArchetypeRoot must directly constrain a "root" or "defining" Archetype.

**// NOTE: I'm not sure this is what the OCL actually says.**

[OCL]

self.base\_Classifier.general->exists(g|g.namespace.stereotypedBy('Archetype'))

* **VARXAV- external reference node archetype reference validity**

If the reference object is a redefinition of another external reference node, the archetype\_ref of the object must match a real archetype that has as an ancestor the archetype matched by the archetype reference mentioned in the corresponding parent node.

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ArchetypeRoot')) implies self.base\_Classifier.general.general->forAll(g|g.namespace=self.base\_Classifier.namespace.oclAsType(Package).packageImport.importedPackage->select(p|p.stereotypedBy('Archetype'))->asSequence()->first())

#### ArchetypeSlot [Stereotype]

**Description**

The ArchetypeSlot stereotype provides a set of constraints on the possible Archetypes that determines which other archetypes can appear at that point in the current archetype. It can be thought of like a keyhole, into which few or many keys might fit, depending on how specific its shape is. Logically it has the same semantics as a ComplexObjectConstraint, except that the constraints are expressed in another Archetype, not the current one. ArtchetypeSlot allows an archetype to have a composition relationship with any number of archetypes matching some constraint pattern. Depending on what archetypes are available within the system, the archetypes matched may vary. Assertions are used in ArchetypeSlots, in order to express the ‘included’ and ‘excluded’ archetypes for the slot. In this case, each assertion is an expression that refers to parts of other archetypes, such as its identifier (e.g. ‘include archetypes with short\_concept\_name matching xxxx’). Assertions are modelled here as a generic expression tree of unary prefix and binary infix operators. In UML, the assertions are modeled using a Constraint whose specification is a UML Expression. The Expression symbol will be 'include' or 'exclude' to reflect the kind of assertion to be made on the slot. An > can be redefined by: - one or more > nodes taken together, considered to define a ‘filled’ version of the slot; - an >. If the UML attribute 'leaf' is set, the slot is closed to further filling either in further specialisations or at runtime. If leaf is not set, the slot remains open. The redefinition of an > is modeled as a Generalization in UML, in which the general is an > and the specific is either a > or an >.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Direct Superclasses (Generalization)**

[ConstraintProfile::ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Classifier

**Constraints**

* **VDSSP- specialised archetype slot definition parent validity**

VDSSP- specialised archetype slot definition parent validity rules apply to ARCHETYPE\_SLOTs defined as the specialisation of a slot in the parent archetype: The flat parent of the specialisation of an archetype slot must be not be closed (is\_closed = False).

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ARCHETYPE\_SLOT')) implies self.base\_Classifier.general->forAll(g|not(g.isLeaf))

* **VDSEV- archetype slot ‘exclude’ constraint validity**

VDSEV- archetype slot ‘exclude’ constraint validity rules for ARCHETYPE\_SLOTs are as follows: The ‘exclude’ constraint in an archetype slot must conform to the slot constraint validity rules.

[OCL]

self.base\_Classifier.ownedRule.specification->select(r|r.oclAsType(Expression).symbol='includes')->notEmpty() implies self.base\_Classifier.ownedRule.specification->select(r|r.oclAsType(Expression).symbol='excludes')->isEmpty()

* **VDSSM- specialised archetype slot definition match validity**

VDSSM- specialised archetype slot definition match validity rules apply to ARCHETYPE\_SLOTs defined as the specialisation of a slot in the parent archetype: The set of archetypes matched from a library of archetypes by a specialised archetype slot definition must be a proper subset of the set matched from the same library by the parent slot definition.

[English]

rules apply to ARCHETYPE\_SLOTs defined as the specialisation of a slot in the parent archetype:  The set of archetypes matched from a library of archetypes by a specialised archetype slot definition must be a proper subset of the set matched from the same library by the parent slot definition.

* **VDSSID- slot redefinition child node id**

VDSSID- slot redefinition child node id rules apply to ARCHETYPE\_SLOTs defined as the specialisation of a slot in the parent archetype: a slot node in a specialised archetype that redefines a slot node in the flat parent must have an identical node id.

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ArchetypeRoot')) implies (self.base\_Classifier.appliedStereotypeInstance.oclAsType(ArchetypeRoot).id ->forAll(e|e =self.base\_Classifier.general.appliedStereotypeInstance.oclAsType(ArchetypeRoot).id->asSequence()->first()) )

* **VDSSC- specialised archetype slot definition closed validity**

VDSSC- specialised archetype slot definition closed validity rules apply to ARCHETYPE\_SLOTs defined as the specialisation of a slot in the parent archetype: In the specialisation of an archetype slot, either the slot can be specified to be closed (is\_closed = True) or the slot can be narrowed, but not both.

[OCL]

self.base\_Classifier.general->forAll(g|g.stereotypedBy('ARCHETYPE\_ROOT')) implies (self.base\_Classifier.ownedRule->isEmpty()=self.base\_Classifier.isLeaf)

* **VDSIV- archetype slot ‘include’ constraint validity**

VDSIV- archetype slot ‘include’ constraint validity rules for ARCHETYPE\_SLOTs are as follows: The ‘include’ constraint in an archetype slot must conform to the slot constraint validity rules.

[OCL]

(self.base\_Classifier.ownedRule->select(r|r.specification.oclAsType(Expression).symbol='excludes')->notEmpty()) implies (self.base\_Classifier.ownedRule->select(r|r.specification.oclAsType(Expression).symbol='includes')->isEmpty())

* **VDFAI- archetype identifier validity in definition**

VDFAI- archetype identifier validity in definition rules for ARCHETYPE\_SLOTs are as follows: Any archetype identifier mentioned in an archetype slot in the definition section must conform to the published openEHR specification for archetype identifiers.

[English]

In UML, a direct reference is made to the <<Archetype>> (as opposed to a string representing the archetype). The archetype identifier is defined by the name of the <<Archetype>>, which has a constraint on its syntax. The archetype identifier is provisioned from the <<Archetype>> as required. As a consequence, the AOM Validation Rule is enforced by the typing of Value Specifications within an Expression plus the constraints on an <<Archetype>> name.

* **includes\_valid**

includes /= Void **implies not** includes.is\_empty

[OCL]

self.base\_Classifier.ownedRule.oclAsType(UML::Constraint) ->select(r|r.oclAsType(UML::Constraint).constrainedElement->exists(c|c=self.base\_Classifier)).specification ->select(s|s.oclIsKindOf(UML::Expression)).oclAsType(UML::Expression) ->forAll(e| ((e.symbol='and')and e.operand->select(op|op.oclIsKindOf(UML::Expression)).oclAsType(UML::Expression)->select(o|o.symbol='includes')->forAll(i|i.operand->notEmpty())) or ((e.symbol='includes')and e.operand->notEmpty()) )

* **excludes\_valid**

excludes /= Void implies not excludes.is\_empty

[OCL]

self.base\_Classifier.ownedRule.oclAsType(UML::Constraint) ->select(r|r.constrainedElement->exists(c|c=self.base\_Classifier)).specification ->select(s|s.oclIsKindOf(UML::Expression)).oclAsType(UML::Expression) ->forAll(e| ( (e.symbol='and') and e.oclAsType(UML::Expression).operand->select(o| o.oclIsKindOf(UML::Expression) and (o.oclAsType(UML::Expression).symbol='excludes') ).oclAsType(UML::Expression)->forAll(i|i.oclAsType(UML::Expression).operand->notEmpty()) ) or ((e.symbol='excludes')and e.oclAsType(UML::Expression).operand->notEmpty()) )

#### AuthoredResource [Stereotype]

**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., via a Usage from <<AuthoredResource>> to a <<ResourceTranslation>> Enumeration.

If the name of the Usage is "original\_language", then the Usage specifies the original language for the Archetype. There must be exactly one "original\_language" Usage for each Archetype.

If the name of the Usage is "terminology\_original\_language", then the Usage specifies the original language for the Archetype Terminology. There must be exactly one "terminology\_original\_language" Usage for each Archetype.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Namespace

**Direct Subclasses (Specialization)**

[ConstraintProfile::Archetype](#_fa40d7338aecd18f94732d3b02e2bd79)

**Attributes**

**•** annotations : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** copyright : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** current\_revision : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** custodian\_namespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

A formal namespace corresponding to the current custodian of the artefact. It enables the user of the artefact to determine who is the definitive maintainer and publisher.

**•** custodian\_organisation : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

A human-readable organisation identifier corresponding to the current custodian of the artefact. It enables the user of the artefact to determine who is the definitive maintainer and publisher.

**•** ip\_acknowledgements : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** ip\_acknowledgements\_id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** licence : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** lifecycle\_state : ConstraintProfile::Lifecycle\_state [1]

The *description.lifecycle\_state* is an important property of an archetype, which is used to record its state in a defined lifecycle. The lifecycle state machine and versioning rules are explained fully in the openEHR Knowledge Artefact Identification specification. Here we simply note that the value of the property is a coded term corresponding to one of the macro-state names on the diagram, i.e. ‘unmanaged’, ‘initial’, ‘in\_development’, and so on.

**•** original\_author\_date : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Date of original authoring of the resource.

Corresponds to AOM RESOURCE\_DESCRIPTION original\_author where id="date".

**•** original\_author\_email : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Name of original author's email.

Corresponds to AOM RESOURCE\_DESCRIPTION original\_author where id="email".

**•** original\_author\_name : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Name of original author for resource.

Corresponds to AOM RESOURCE\_DESCRIPTION original\_author where id="name".

**•** original\_author\_organization : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Name of original author's organization.

Corresponds to AOM RESOURCE\_DESCRIPTION original\_author where id="organization".

**•** original\_namespace : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** original\_publisher : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** other\_contributors : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** otherDetails : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** otherDetails\_id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** references : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

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

**•** references\_id : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

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

**•** resourceDocumentLanguage : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** resourceDocumentSyntax : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** resourceSource : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** resourceSourceURI : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

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

**•** resource\_package\_uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** uid : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**Constraints**

* **Adl\_version\_validity**

adl\_version /= Void **and then** valid\_version\_id

(adl\_version)

[English]

adl\_version /= Void and then valid\_version\_id (adl\_version)

* **Rm\_release\_validity**

rm\_release /= Void **and then** valid\_version\_id (rm\_release)

[English]

rm\_release /= Void and then valid\_version\_id (rm\_release)

* **Build\_uid\_validity**

build\_uid /= Void

[English]

For AML, the build\_uid is derived and is implicitly updated based on model changes.

#### ComplexObjectConstraint [Stereotype]

**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.

A **ComplexObjectConstraint** corrsponds to a C\_COMPLEX\_OBJECT in the AOM 2.0 model. It defines constraints on a Reference Model Classifier. The constrained Reference Model Classifier is modeled as the general Classifier for the **ComplexObjectConstraint** (which corresponds to the *rm\_type\_name* property of C\_OBJECT in the AOM Model).

The owned properties of a **ComplexObjectConstraint** correspond to the attributes of a C\_COMPLEX\_OBJECT (which are C\_ATTRIBUTEs in the AOM model). Each owned Property (i.e., C\_ATTRIBUTE) constrains a Reference Model attribute (whose name corresponds to the C\_ATTRIBUTE *rm\_attr\_name* in AOM). The constrained Reference Model attribute is modeled in UML using either a *subsetsProperty* or *redefinesProperty* . The constraining Property is subject to the rules for subsetting/redefinition within UML, which in general means it may have a multiplicity range within the constrained Property's multiplicity range, and the type of the Property must be a valid subset of the constrained Property's type. A Reference Model Property may have multiple subseting/redefining Properties. Together, the Properties define an ordering for contained instances, and must overall conform to multiplicity constraints of the Reference Model Property.

The types of constraining Properties will generally be archetype-specific subtypes of corresponding Reference Model types referenced by the Reference Model Properties. In the case of primitive types, the archetype model will generally reference the same type referenced by the Reference Model (e.g., string, integer, etc.).

In addition to the multiplicity and type constraints, each constraining Property may express additional constraints in the form of UML Constraints. The UML Constraint will be owned by the Classifier owning the Property, but will specifically be defined to constrain the Property. The specification of the Constraint is a ValueSpecification, typically in the form of a UML Expression which evaluated to the required boolean result of a Constraint Specification. A UML Expression consists of a symbol (i.e., an operator) and a set of operands (which are also ValueSpecifications). The various expressions used in AOM may be represented in the form of these expression trees.

The AOM concept of default*\_value* for a Classifier is modeled as the defaultValue for the Property defining the containment of the Classifier.

The AOM concept of *assumed\_value* for a Classifier is modeled as a constraint containing an expression whose symbol is 'assumed\_value' and whose operand is the ValueSpecification for the Classifier (e.g., a LiteralSpecification or an InstanceValue whose instance is an EnumerationLiteral).

The AOM *node\_id* is modeled as a tag value on the inherited **ObjectConstraint** Stereotype and constitutes a reference to a term defined in the Archetype Terminology model.

The AOM concept of is\_frozen is modeled as leaf

The AOM concept of *default\_value* is currently not supported.

The overall structure of an Archetype can be described as a containment structure consisting of AOM C\_COMPLEX\_OBJECT==>C\_ATTRIBUTE==>C\_COMPLEX\_OBJECT... Normally the aggregation kind of each Property (C\_ATTRIBUTE ) is composite. Within AOM, there is a notion of a "Proxy", which is basically a reference to a C\_COMPLEX\_OBJECT (instead of a composite). In UML, this is modeled at the Property level using an aggregation of "none" (instead of composite). Thus, in UML the AOM C\_COMPLEX\_OBJECT\_PROXY is not explicitly part of the AML model.

A <<ComplexObjectConstraint>> may represent the concept of an AOM C\_ARCHETYPE\_ROOT. When an attribute has a type which is a <<ComplexObjectConstraint>> in another <<Archetype>> Package, there is an implicit C\_ARCHETYPE\_ROOT with appropriate archetype\_id.

A Property owned by a <<ComplexObjectConstraint>> may express ownership of the AOM concept of ARCHETYPE\_SLOT as a UML Constraint on that Property wherein the specification is a UML Expression with an operand of "and", "includes", or "excludes".

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Direct Superclasses (Generalization)**

[ConstraintProfile::ObjectConstraint](#_ad75af95f635bdf35f69d9db9b17aae2)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Classifier

**Direct Subclasses (Specialization)**

[ConstraintProfile::ArchetypeRoot](#_880ae5af371f2c1be5c3669ca302ce11)

**Constraints**

* **singleParent**

Every constraint must specialize exactly one Class

[OCL]

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

* **allAttributeConstraints**

All owned attributes are attribute constraints. Each must subset or redefine a reference model property.

[OCL]

self.namespace.stereotypedBy('ComplexObjectConstraint') implies ((self.subsettedProperty->notEmpty()) or( self.redefinedProperty->notEmpty()))

* **VARCN- archetype concept validity**

VARCN- archetype concept validity The node\_id of the root object of the archetype must be of the form id1{.1}\*, where the number of ‘.1’ components equals the specalisation depth, and must be defined in the terminology. self.base\_Classifier.supplierDependency->exists(d|d.stereotypedBy('ArchetypeDefinition')) implies self.node\_id.repr().match('id1(\\.1)\*')

[OCL]

self.base\_Classifier.supplierDependency->exists(d|d.stereotypedBy('ArchetypeDefinition')) implies self.id.oclAsType(EnumerationLiteral)->forAll(e|e.name.match('id1(\\.1)\*'))

* **VCARM- attribute name reference model validity**

VCARM- attribute name reference model validity an attribute name introducing an attribute constraint block must be defined in the underlying information model as an attribute (stored or computed) of the type which introduces the enclosing object block.

[English]

This AOM Validation Rule is enforced by UML type constraints for subsetted and/or redefined Properties.

* **VCAEX- archetype attribute reference model existence conformance**

VCAEX- archetype attribute reference model existence conformance the existence of an attribute, if set, must conform, i.e. be the same or narrower, to the existence of the corresponding attribute in the underlying information model.

[English]

The AOM validation rules is enforces by UML validation rules for subsetted or redefined properties.

* **VCAM- archetype attribute reference model multiplicity conformance**

VCAM- archetype attribute reference model multiplicity conformance the multiplicity, i.e. whether an attribute is multiply- or single-valued, of an attribute must conform to that of the corresponding attribute in the underlying information model.

[English]

This validation rule is enforced by UML Constraints on subsetted and/or redefined Properties.

* **VDIFV- archetype attribute differential path validity:**

VDIFV- archetype attribute differential path validity: an archetype may only have a differential path if it is specialised

[English]

an archetype may only have a differential path if it is specialised

* **VDIFP- specialised archetype attribute differential path validity**

VDIFP- specialised archetype attribute differential path validity applies to redefinition in a specialised archetype: if an attribute constraint has a differential path, the path must exist in the flat parent, and also be valid with respect to the reference model, i.e. in the sense that it corresponds to a legal potential construction of objects.

[English]

applies to redefinition in a specialised archetype:   if an attribute constraint has a differential path, the path must exist in the flat parent, and also be valid with respect to the reference model, i.e. in the sense that it corresponds to a legal potential construction of objects.

* **VSANCE- specialised archetype attribute node existence conformance**

VSANCE- specialised archetype attribute node existence conformance applies to redefinition in a specialised archetype: the existence of a redefined attribute node in a specialised archetype, if stated, must conform to the existence of the corresponding node in the flat parent archetype, by having an identical range, or a range wholly contained by the latter.

[English]

The AOM Validation Rule is enforced by UML type and cardinality constraints associated with subsetted and/or redefined Properties.

* **VSAM- specialised archetype attribute multiplicity conformance**

VSAM- specialised archetype attribute multiplicity conformance applies to redefinition in a specialised archetype the multiplicity, i.e. whether an attribute is multiply- or single-valued, of a redefined attribute must conform to that of the corresponding attribute in the parent archetype

[English]

The AOM validation rule is enforced by UML multiplicity constraints applicable to subsetted and/or redefined Properties.

* **VACSO- single-valued attribute child object occurrences validity**

VACSO- single-valued attribute child object occurrences validity apply to single-valued attributes, i.e when C\_ATTRIBUTE.is\_multiple is False: the occurrences of a child object of a single-valued attribute cannot have an upper limit greater than 1. The AOM validation rule is realized by UML redefinition and/or property subsetting constraints. We verify in this UML Constraint that Properties within a > are indeed redefined or subsetted.

[OCL]

self.namespace.stereotypedBy('ComplexObjectConstraint') implies( self.subsettedProperty->notEmpty() or self.redefinedProperty->notEmpty() )

* **VACMCU- cardinality/occurrences upper bound validity**

VACMCU- cardinality/occurrences upper bound validity apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True: where a cardinality with a finite upper bound is stated on an attribute, for all immediate child objects for which an occurrences constraint is stated, the occurrences must either have an open upper bound (i.e. n..\*) which is interpreted as the maximum value allowed within the cardinality, or else a finite upper bound which is = the cardinality upper bound.

[English]

The AOM cardinality attribute of C\_ATTRIBUTE is derived in UML from the set of sibling Properties subsetting/redefining a common Reference Model Property. The derivation is implemented as part of the AML-UML to AOM QVT transformation. The derived cardinality is computed from the sum of the lower and sum of the upper multiplicity ranges for the related Properties. The collection type characteristics are also derived from the related Properties. In summary, the AOM validation rule is enforced during provisioning from AML-UML to AOM.

* **VACMCO- cardinality/occurrences orphans**

VACMCO- cardinality/occurrences orphans apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True: it must be possible for at least one instance of one optional child object (i.e. an object for which the occurrences lower bound is 0) and one instance of every mandatory child object (i.e. object constraints for which the occurrences lower bound is >= 1) to be included within the cardinality range.

[English]

apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True:   it must be possible for at least one instance of one optional child object (i.e. an object for which the occurrences lower bound is 0) and one instance of every mandatory child object (i.e. object constraints for which the occurrences lower bound is >= 1) to be included within the cardinality range.

* **VCACA- archetype attribute reference model cardinality conformance**

VCACA- archetype attribute reference model cardinality conformance apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True: the cardinality of an attribute must conform, i.e. be the same or narrower, to the cardinality of the corresponding attribute in the underlying information model.

[English]

This cardinality rule is enforced by UML constraints on subsetted and/or redefined properties.

* **WACMCL- cardinality/occurrences lower bound validity**

WACMCL- cardinality/occurrences lower bound validity apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True: where a cardinality with a finite upper bound is stated on an attribute, for all immediate child objects for which an occurrences constraint is stated, the sum of occurrences lower bounds should be lower than the cardinality upper limit.

[English]

apply to container attributes, i.e when C\_ATTRIBUTE.is\_multiple is True:    where a cardinality with a finite upper bound is stated on an attribute, for all immediate child objects for which an occurrences constraint is stated, the sum of occurrences lower bounds should be lower than the cardinality upper limit.

* **VSANCC- specialised archetype attribute node cardinality conformance**

VSANCC- specialised archetype attribute node cardinality conformance applies to cardinality redefinition in a specialised archetype: the cardinality of a redefined (multiply-valued) attribute node in a specialised archetype, if stated, must conform to the cardinality of the corresponding node in the flat parent archetype by either being identical, or being wholly contained by the latter.

[English]

The AOM validation rule is enforced by UML constraints related to cardinality of subsetted and/or redefined Properties.

* **VCATU- attribute uniqueness**

VCATU- attribute uniqueness sibling attributes occurring within an object node must be uniquely named with respect to each other, in the same way as for class definitions in an object reference model.

[English]

The AOM Validation Rules is enforced by UML Namespace constraints for names.

* **VUNT- use\_node reference model type validity**

VUNT- use\_node reference model type validity rules applies to internal references: the reference model type mentioned in an C\_COMPLEX\_OBJECT\_PROXY node must be the same as or a supertype (according to the reference model) of the reference model type of the node referred to.

[English]

The AOM validation rule is enforced by UML type constraints related to subsetted and/or redefined Properties.

* **VUNP- use\_node path validity**

VUNP- use\_node path validity rules applies to internal references: the path mentioned in a use\_node statement must refer to an object node defined elsewhere in the same archetype or any of its specialisation parent archetypes, that is not itself an internal reference node, and which carries a node identifier if one is needed at the reference point. Note that the AML-UML representation for a use\_node is a > owned property which has aggregation=none.

[English]

(self.namespace.stereotypedBy('ComplexObjectConstraint') and self.aggregation=AggregationKind::none) implies( (self.type.getNearestPackage()=self.getNearestPackage()) or self.getNearestPackage().packageImport.importedPackage->includes(self.type.getNearestPackage()) )

* **VSUNT- use\_node specialisation parent validity**

VSUNT- use\_node specialisation parent validity rule applies to the redefinition of an internal reference in a specialised archetype: a C\_COMPLEX\_OBJECT\_PROXY node may be redefined in a specialised archetype by another C\_COMPLEX\_OBJECT\_PROXY (e.g. in order to redefine occurrences), or by a C\_COMPLEX\_OBJECT structure that legally redefines the target C\_COMPLEX\_OBJECT node referred to by the reference.

[English]

A proxy is modeled using a Property aggregation of none. The aggregation of a redefined/subsetted Property, and consequently whether or not the node is a Proxy, may be different than the archetype Property. Therefore, it is permitted to use a proxy or regular node in a specialized archetype.

* **VOBAV- object node assumed value validity**

VOBAV- object node assumed value validity rules for C\_PRIMITIVE\_OBJECTs are as follows: the value of an assumed value must fall within the value space defined by the constraint to which it is attached.

[English]

rules for C\_PRIMITIVE\_OBJECTs are as follows:   the value of an assumed value must fall within the value space defined by the constraint to which it is attached.

* **VACDF- constraint code validity**

VACDF- constraint code validity Each value set code (ac-code) used in a term constraint in the archetype definition must be defined in the term\_definitions part of the terminology of the current archetype.

[OCL]

(self.\_'context'.stereotypedBy('ComplexObjectConstraint') and self.specification.oclIsKindOf(Expression) and (self.specification.oclAsType(Expression).symbol='=') ) implies self.specification.oclAsType(Expression).operand ->forAll(o|o.oclIsKindOf(InstanceValue) and o.oclAsType(InstanceValue).instance.stereotypedBy('ARCHETYPE\_TERM') and o.oclAsType(InstanceValue).instance.getNearestPackage().nestingPackage.nestingPackage=self.getNearestPackage() )

* **VSONPI- specialised archetype prohibited object node AOM node id validity**

VSONPI- specialised archetype prohibited object node AOM node id validity This rule governs C\_OBJECTs in specialised archetypes: A redefined object node in a specialised archetype with occurrences matching {0} must have exactly the same node id as the node in the flat parent being redefined.

[OCL]

self.base\_Classifier.attribute->select(a|a.upper=0).type->forAll(t| t.oclAsType(Classifier).general ->forAll(g|g.appliedStereotypeInstance.oclAsType(ObjectConstraint).id ->forAll(e|e=t.appliedStereotypeInstance.oclAsType(ObjectConstraint).id->asSequence()->first()) ) )

* **Groups\_valid**

groups /= Void **implies** is\_multiple

The AOM concept of C\_OBJECT\_GROUP and its members are modeled in AML as two levels of Property subsetting.

The first subsets is a Property which subsets another Property in the <<ComplexObjectConstraint>> (as opposed to subsetting a Property from the Reference Model).

This first level of subsetting corresponds to the AOM C\_OBJECT\_GROUP, with the AOM parent corresponding to the target of the subsettedProperty.

The second level of subsetting is a set of Properties which subsets the first level.

Each of these second level Properties are typed by a Classifier which correspond to the children of the C\_ATTRIBUTE.

The AOM concept of is\_multiple within C\_ATTRIBUTE is modeled in AML as a Property which has a subsettedProperty. It follows that a Property having a subsettedProperty in the same <<ComplexObjectConstraint>> requires that the subsettedProperty itself has a subsettedProperty.

[English]

( self.namespace.stereotypedBy('ComplexObjectConstraint') and self.subsettedProperty->exists(p|p.namespace=self.namespace) ) implies self.subsettedProperty.subsettedProperty->notEmpty()

* **Rm\_type\_name\_valid**

rm\_type\_name /= Void **and then not**

rm\_type\_name.is\_empty

In AML, the rm\_type\_name is implicitly derived from the name of the Reference Model Classifier which is the supertype of an Archetype Classifier.

[English]

self.base\_Classifier.general->forAll(g|not(g.name.oclIsUndefined()) and (g.name<>''))

* **Prohibited\_validity**

not (any\_allowed and is\_prohibited)

The AOM concept of any\_allowed is modeled in AML as a property whose type is in the Reference Model.

The AOM concept of is\_prohibited is modeled in AML as a property whose upper bound is 0.

[OCL]

self.namespace.stereotypedBy('ComplexObjectConstraint') implies not( (self.upper=0) and not(self.namespace.oclAsType(Classifier).general->exists(g|g.stereotypedBy('ComplexObjectConstraint'))) )

* **Any\_allowed\_validity**

Any\_allowed\_validity. any\_allowed xor not attributes.is\_empty In AML, any\_allowed is modeled as a Property whose type is a Reference Model. The AOM concept of attributes of a C\_COMPLEX\_OBJECT corresponds to the attributes of a >. This AOM Invariant effectively states that a > must own properties.

[English]

The invariant is definitional, it defines the result of a derived attribute.

* **Attributes\_valid**

attributes /= Void

[English]

UML always returns a collection for attributes, it is never oclIsUndefined().

* **Tuples\_valid**

attribute\_tuples /= Void **implies** attribute\_tuples.for\_all (cat:

C\_ATTRIBUTE\_TUPLE|cat.members.for\_all (ca: C\_ATTRIBUTE)|attributes.has(ca) ))

In AML, *the* *tuple* *is* *nested* *inside* *its* *Complex* *Object,* *inherits* *that* *Complex* *Object,* *and* *subsets/redefines* *properties in that base Complex Object.*

[OCL]

self.base\_Classifier.oclIsKindOf(Class) implies self.base\_Classifier.oclAsType(Class).nestedClassifier ->select(n|n.general->exists(g|g=self.base\_Classifier)).attribute ->forAll(a|a.subsettedProperty->forAll(s|s.namespace=self.base\_Classifier)and a.redefinedProperty->forAll(s|s.namespace=self.base\_Classifier))

* **Representation\_validity**

constraint /= Void The AOM concept of C\_PRIMITIVE\_OBJECT is modeled in AML as a Property whose type resolves to a recognized AML Primitive Type. An AML Primitive Type is a UML PrimitiveType which resides in either the UML Type Library or the XML Type Library. Resolving a type may include navigating the > Abstraction from the type of the Property residing in a ReferenceModel to one of the AML Primitive Types. The Property must be constrained by a UML Constraint.

[OCL]

( self.namespace.stereotypedBy('ComplexObjectConstraint') and ( (self.type.namespace.name='PrimitiveTypes') or(self.type.namespace.name='XMLPrimitiveTypes') or self.type.clientDependency->select(d|d.stereotypedBy('MappedDataType')).supplier->exists(t|(t.namespace.name='PrimitiveTypes')or(t.namespace.name='XMLPrimitiveTypes')) ) ) implies self.namespace.ownedRule->notEmpty()

* **Assumed\_value\_valid**

has\_assumed\_value implies assumed\_value.conforms\_to\_type (rm\_type\_name) and valid\_value (assumed\_value) The AOM assumed\_value concept is represented in AML as the defaultValue of a Property. The defaultValue is a ValueSpecification which should be typed in accordance with the type of the Property and should have a value in conformance with the constraints expressed on that Property's Type (The constraint implementation is looking for type equivalence, but does not check value conformance).

[OCL]

( self.namespace.stereotypedBy('ComplexObjectConstraint') and not(self.defaultValue.oclIsUndefined()) ) implies( (self.defaultValue.type=self.type) or (self.type.clientDependency->select(d|d.stereotypedBy('MappedDataType')).supplier->asSequence()->first()=self.defaultValue.type) )

* **Binary\_consistency**

true\_valid ***or*** false\_valid

In AML, a constraint on a Boolean typed Property is expressed as an "or" of LiteralBooleans.

[OCL]

( self.constrainedElement.oclIsKindOf(Property) and self.constrainedElement.oclAsType(Property).namespace.stereotypedBy('ComplexObjectConstraint') and (self.constrainedElement.oclAsType(Property).type.name='Boolean') and self.specification.oclIsKindOf(Expression) and ( self.specification.oclAsType(Expression).symbol='or') ) implies( self.specification.oclAsType(Expression).operand->forAll(o|o.oclIsKindOf(LiteralBoolean)) )

* **Prototype\_value\_consistency**

value ***and*** true\_valid ***or else not*** .value ***and***

false\_valid

[English]

value and true\_valid or else not .value and false\_valid

* **Pattern\_validity**

pattern /= Void **implies**

valid\_iso8601\_date\_constraint\_pattern(pattern)

[English]

pattern /= Void implies valid\_iso8601\_date\_constraint\_pattern(pattern)

* **Terminology\_id\_validity**

terminology\_id /= Void

[English]

terminology\_id /= Void

* **Code\_list\_validity**

code\_list /= Void

[English]

code\_list /= Void

* **Tag\_valid**

tag /= Void **implies not** tag.is\_empty

The AOM tag is represented in AML as the name of a Constraint. In UML, it may be undefined, an empty String, or a non-empty String.

[OCL]

( self.constrainedElement.oclIsKindOf(Property) and self.constrainedElement.oclAsType(Property).namespace.stereotypedBy('ComplexObjectConstraint') ) implies( self.name.oclIsUndefined() or (self.name<>'') )

* **Expression\_valid**

expression /= Void and then expression.

type.is\_equal(“BOOLEAN”)

[English]

This AOM invariant is enforced by the UML constraint that a Constraint specification must be of type Boolean.

* **Name\_valid**

name /= Void and then not name.is\_empty

An AOM VARIABLE\_DECLARATION is represented in AML as an InstanceSpecification owned by an <<Archetype>> Package.

[OCL]

self.namespace.stereotypedBy('ComplexObjectConstraint') implies( not( self.name.oclIsUndefined()) and(self.name<>'') )

* **Type\_valid**

type /= Void **and then not** type.is\_empty

[OCL]

self.namespace.stereotypedBy('ComplexObjectConstraint') implies self.classifier->notEmpty()

* **EXPR\_VARIABLE.Expression\_valid**

expression /= Void

[English]

The AOM concept of EXPR\_VARIABLE expression is represented in AML as the specification of an InstanceSpecification owned by an <<ComplexObjectConstraint>>. In AML, if the specification is defined, then it is considered an EXPR\_VARIABLE, otherwise it is considered a BUILTIN\_VARIABLE.

* **Context\_valid**

Context\_valid. context /= Void implies not context.is\_empty The AOM concept of QUERY\_VARIABLE is represented in AML as an InstanceSpecification whose specification is an OpaqueExpression. The AOM QUERY\_VARIABLE context is represented in AML as the name of the OpaqueExpression. The AOM QUERY\_VARIABLE query\_id is represented in AML as the name of the InstanceSpecification.

[OCL]

( self.namespace.stereotypedBy('ComplexObjectConstraint') and not(self.specification.oclIsUndefined()) and self.specification.oclIsKindOf(OpaqueExpression) ) implies( self.specification.name.oclIsUndefined() or(self.specification.name<>'') )

* **Query\_id\_valid**

query\_id /= Void **and then not** query\_id.is\_empty

[OCL]

( self.namespace.stereotypedBy('ComplexObjectConstraint') and not(self.specification.oclIsUndefined()) and self.specification.oclIsKindOf(OpaqueExpression) ) implies( not(self.name.oclIsUndefined()) and(self.specification.name<>'') )

* **EXPR\_ITEM.Type\_valid**

type /= Void **and then not** type.is\_empty

The AOM EXPR\_ITEM is represented in AML as a ValueSpecification. A ValueSpecification is a TypedElement wian optional type attribute.

[OCL]

self.getNearestPackage().stereotypedBy('ComplexObjectConstraint') implies not(self.type.oclIsUndefined())

* **Value\_valid**

value /= Void

[English]

value /= Void

* **Item\_valid**

item /= Void

[English]

item /= Void

* **Constraint\_validity**

item.is\_pattern -- **and** item.pattern matches

ARCHETYPE\_ID.pattern\_template

[English]

item.is\_pattern -- and item.pattern matches ARCHETYPE\_ID.pattern\_template

* **Path\_valid**

path /= Void

[English]

path /= Void

* **Declaration\_valid**

declaration /= Void

The AOM concept of EXPR\_VARIABLE\_REF is represented in AML as an InstanceValue whose instance represents the AOM VARIABLE\_DECLARATION.

[English]

The AOM concept of EXPR\_VARIABLE\_REF is represented in AML as an InstanceValue whose instance represents the AOM VARIABLE\_DECLARATION. The UML cardinality constraint of 1..1 on instance enforces this AOM Invariant.

* **operand\_valid**

operand /= Void

The AOM concept of EXPR\_UN\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM unary operators.

[English]

The AOM concept of EXPR\_UNARY\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM unary operators. Thus, the operand (i.e., symbol) is never void.

* **left\_operand\_valid**

operand /= Void

The AOM concept of EXPR\_BINARY\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM binary operators and which has exactly two operands.

[English]

The AOM concept of EXPR\_BINARY\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM binary operators and which has exactly two operands. Thus, the left operand (i.e., symbol) is never void.

* **right\_operand\_valid**

operand /= Void

The AOM concept of EXPR\_BINARY\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM binary operators and which has exactly two operands.

[English]

The AOM concept of EXPR\_BINARY\_OPERATOR is modeled in AML as an Expression whose symbol is one of the defined AOM binary operators and which has exactly two operands. Thus, the right operand (i.e., symbol) is never void.

* **Validity**

valid\_operator(value)

The AOM concept of OPERATOR\_KIND is represented in AML as an Expression symbol.

[English]

The AOM concept of OPERATOR\_KIND is represented in AML as an Expression symbol.

#### Constrains [Stereotype]

**Description**

The Constrains stereotype extends the Generalization association, and associates a general Reference Model Classifier or AML ObjectConstraint with a specific AML ObjectConstraint, where specific ObjectConstraint constrains the names, possible values, cardinalities or other attributes of the general Classifier. The Generalization stereotype represents both the rm\_type\_name attribute of C\_OBJECT and the parent attribute of ARCHETYPE\_CONSTRAINT in the AOM 2.0 model [aom].

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Generalization

**Constraints**

* **specificObjectConstraint**

The specific property must reference an ObjectConstraint.

[OCL]

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

* **constrainsRMElement**

The general property must reference a Classifier within the Package extended by the ReferenceModel stereotype that is imported by the containing ArchetypeLibrary or it must reference another ObjectConstraint.

[English]

( self.base\_Generalization.specific.namespace.oclAsType(Package).nestingPackage.packageImport.importedPackage ->select(p|p.stereotypedBy('ReferenceModel'))->exists(referenceModel|self.base\_Generalization.general.getNearestReferenceModel()=referenceModel) ) or self.base\_Generalization.general.stereotypedBy('ObjectConstraint')

#### ObjectConstraint [Stereotype]

**Description**

An **ObjectConstraint** is a specialization of a NamedElement. If **ObjectConstraint** is a Classifier, then there are no new *ownedAttributes* that do not *subset* or *redefine* an iinherited *attribute*. An **ObjectConstraint** may also be modeled as a UML Constraint on a Property, in which case it is specifying constraints on the value space for the PrimitiveType which is the type of the Property.

AOM defines a node\_id. The node\_id provides a code to which a human-readable terminology definition can be attached, as well as potentially a terminology binding. A characteristics of a node\_id is further defined as either a node-id, an at-code, or an ac-code. In all cases, the node\_id has a terminology definition in each of many natural languages. The terminology definition associates the node\_id with a term and a description. The ac-code and at-code identifiers are used to reference a local or external value set definition. The external value set reference is defined as a terminology binding. The local value set reference is defined as a value-set. A terminology binding definition associates an at-code with a terminology code (e.g., a URI or id) for one or more terminologies (each identified via ac-code). A value-set definition associates an ac-code with one or more at-codes.

In AML, the AOM concept of node\_id is modeled using the inherited **id** tag. The value of **id** is an EnumerationLiteral which is an <<ARCHETYPE\_TERM>> within a language-specific terminology definition. The language-specific Enumeration must be the same as the terminology original language.

In the case of primitive types, the **id** used as the node-id is not required and implicitly has a value of id9999.

In the case of single-valued attributes, the **id** used as the node-id is not required and is implicitly the name of the type.

AML provides an alternative abstraction for <<ArchetypeLibrary>>s which avoids explicit use of the **id** tag. Extent of round-trip engineering is limited for such models: the original AOM node\_id would not be preserved, and there may be some duplication of term definitions. Instead of using the <<IdentifiedIterm>> **id** tag, the language/sign/description tags from <<DesignatableItem>> are used to represent language-specific terms and definitions.

Terminology bindings are represented by a <<ValueSetDefinitionReference>>, which owns <<ConceptReference>> EnumeraitonLiterals. By convention, there is an unnamed and unstereotyped EnumerationLiteral which represents the <<ValueSetDefinitionReference>> as a whole. Reference to these EnumerationLiterals will be from an InstanceValue within a UML Expression.

Local Value Sets are represented by an <<EnumeratedValueDomain>>, which owns <<PermissibleValue>> EnumerationLiterals. By convention, there is an unnamed and unstereotyped EnumerationLiteral which represents the <<EnumeratedValueDomain>> as a whole. Reference to these EnumerationLiterals will be from an InstanceValue within a UML Expression.

For both terminology bindings and local value sets, language-specific term names and definitions may be specified in the tags for the respective Enumeration and EnumerationLiterals. Provisioning to AOM will transform these into terminology definitions.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Direct Superclasses (Generalization)**

[TerminologyProfile::IdentifiedItem](#_4b28f60cd7e8328f1d31dbcfa39d2ff3)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::NamedElement

**Direct Subclasses (Specialization)**

[ConstraintProfile::ArchetypeSlot](#_635388e7087eae0698b1e64ddc5126ce),

[ConstraintProfile::ComplexObjectConstraint](#_bd9b14c4d7198d36c5a9dec9c2836b62)

**Attributes**

**•** is\_deprecated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::Boolean [0..1]

True if this node and by implication all subnodes are deprecated for use.

**Constraints**

* **redefinesGeneralization**

This Classifier must have exactly one generalization, and that is a > Generalization.

[OCL]

self.base\_NamedElement.oclIsKindOf(Classifier) implies( self.base\_NamedElement.oclAsType(Classifier).generalization->forAll(x|x.stereotypedBy('Constrains')) and (self.base\_NamedElement.oclAsType(Classifier).generalization->size() = 1) )

* **about**

At most one "about" reference via > Abstraction

[OCL]

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

* **VCORM- object constraint type name existence**

VCORM- object constraint type name existence a type name introducing an object constraint block must be defined in the underlying information model

[English]

All reference model types are explicitly inherited and/or referenced as types from an archetype structure. Thus the UML model will always use types defined in the underlying Reference Model information model.

* **VCORMT- object constraint type validity**

VCORMT- object constraint type validity a type name introducing an object constraint block must be the same as or conform to the type stated in the underlying information model of its owning attribute

[English]

The AOM validation rule is enforced by UML type conformance constraints associated with subsetted and/or redefined Properties.

* **VCOCD- object constraint definition validity**

VCOCD- object constraint definition validity an object constraint block consists of one of the following (depending on subtype): an ‘any’ constraint; a reference; an inline definition of sub-constraints, or nothing, in the case where occurrences is set to {0}.

[English]

an object constraint block consists of one of the following (depending on subtype): an ‘any’ constraint; a reference; an inline definition of sub-constraints, or nothing, in the case where occurrences is set to {0}.

* **VCOID- object node identifier validity**

VCOID- object node identifier validity every object node must have a node identifier.

[English]

An object node may have a node\_id specified as a tag. If not specified, than an implicit node\_id is based on the object node name. In case of constraints on primitive types, the node\_id may have an implied value starting with 'id9999'. As a consequence of the UML defaults described, there will be a node identifier associated with every object node.

* **VCOSU- object node identifier validity**

VCOSU- object node identifier validity every object node must be unique within the archetype.

[English]

every object node must be unique within the archetype.

* **VSONT- specialised archetype object node meta-type conformance**

VSONT- specialised archetype object node meta-type conformance rules govern C\_OBJECTs in specialised archetypes. the metatype of a redefined object node (i.e. the AOM node type such as C\_COMPLEX\_OBJECT etc) in a specialised archetype must be the same as that of the corresponding node in the flat parent, with the following exceptions: a C\_COMPLEX\_OBJECT with no child attributes may be redefined by a node of any AOM type; a C\_COMPLEX\_OBJECT\_PROXY, may be redefined by a C\_COMPLEX\_OBJECT; a ARCHTEYPE\_SLOT may be redefined by C\_ARCHETYPE\_ROOT (i.e. ‘slot-filling’). See also validity rules VDSSID and VARXID.

[English]

rules govern C\_OBJECTs in specialised archetypes.   the metatype of a redefined object node (i.e. the AOM node type such as C\_COMPLEX\_OBJECT etc) in a specialised archetype must be the same as that of the corresponding node in the flat parent, with the following exceptions: a C\_COMPLEX\_OBJECT with no child attributes may be redefined by a node of any AOM type; a C\_COMPLEX\_OBJECT\_PROXY, may be redefined by a C\_COMPLEX\_OBJECT; a ARCHTEYPE\_SLOT may be redefined by C\_ARCHETYPE\_ROOT (i.e. ‘slot-filling’). See also validity rules VDSSID and VARXID.

* **VSONCT- specialised archetype object node reference type conformance**

VSONCT- specialised archetype object node reference type conformance rules govern C\_OBJECTs in specialised archetypes. the reference model type of a redefined object node in a specialised archetype must conform to the reference model type in the corresponding node in the flat parent archetype by either being identical, or conforming via an inheritance relationship in the relevant reference model.

[English]

The AOM validation rule is enforced by UML inheritance from the specialized archetype, which itself inherits directly or indirectly from a Reference Model.

* **VSONIN- specialised archetype new object node identifier validity**

VSONIN- specialised archetype new object node identifier validity rules govern C\_OBJECTs in specialised archetypes. if an object node in a specialised archetype is a new node with respect to the flat parent, and it carries a node identifier, the identifier must be a ‘new’ node identifier, specalised at the level of the child archetype.

[English]

  rules govern C\_OBJECTs in specialised archetypes.   if an object node in a specialised archetype is a new node with respect to the flat parent, and it carries a node identifier, the identifier must be a ‘new’ node identifier, specalised at the level of the child archetype.

* **VSONIF- specialised archetype object node identifier validity in flat siblings**

VSONIF- specialised archetype object node identifier validity in flat siblings govern C\_OBJECTs in specialised archetypes. the identification (or not) of an object node in a specialised archetype must be valid with respect to any sibling object nodes in the flattened parent (see VACMI).

[English]

govern C\_OBJECTs in specialised archetypes.   the identification (or not) of an object node in a specialised archetype must be valid with respect to any sibling object nodes in the flattened parent (see VACMI).

* **VSONCO- specialised archetype redefine object node occurrences validity**

VSONCO- specialised archetype redefine object node occurrences validity rules govern C\_OBJECTs in specialised archetypes. the occurrences of a redefined object node in a specialised archetype, if stated, must conform to the occurrences in the corresponding node in the flat parent archetype by either being identical, or being wholly contained by the latter.

[English]

The AOM validation rule is enforced by UML multiplicity constraints for subsetted and/or redefined Properties.

* **VSONPT- specialised archetype prohibited object node AOM type validity**

VSONPT- specialised archetype prohibited object node AOM type validity rules govern C\_OBJECTs in specialised archetypes. the occurrences of a redefined object node in a specialised archetype, may only be prohibited (i.e. {0}) if the matching node in the parent is of the same AOM type.

[OCL]

( self.namespace->select(n|n.stereotypedBy('ComplexObjectConstraint') ).oclAsType(Classifier).general->forAll(g|g.stereotypedBy('ComplexObjectConstraint')) and (self.upper=0) ) implies( self.redefinedProperty.type->forAll(t|t=self.type) and self.subsettedProperty.type->forAll(t|t=self.type) )

* **VSONPO- specialised archetype object node prohibited occurrences validity**

VSONPO- specialised archetype object node prohibited occurrences validity rules govern C\_OBJECTs in specialised archetypes. the occurrences of a new (i.e. having no corresponding node in the parent flat) object node in a specialised archetype, if stated, may not be ‘prohibited’, i.e. {0}, since prohibition only makes sense for an existing node.

[OCL]

self.namespace->select(n|n.stereotypedBy('ComplexObjectConstraint') ).oclAsType(Classifier).general->select(g|g.stereotypedBy('ComplexObjectConstraint'))->isEmpty() implies (self.upper<>0)

* **VSSM- specialised archetype sibling order validity**

VSSM- specialised archetype sibling order validity rules govern C\_OBJECTs in specialised archetypes. the sibling order node id code used in a sibling marker in a specialised archetype must refer to a node found within the same container in the flat parent archetype.

[English]

rules govern C\_OBJECTs in specialised archetypes.   the sibling order node id code used in a sibling marker in a specialised archetype must refer to a node found within the same container in the flat parent archetype.

* **path\_exists**

path /= Void

[English]

In AML, the path to a node is implicit. Thus, the path is never undefined and the path exists.

* **Rm\_attribute\_name\_valid**

rm\_attribute\_name /= Void **and then not**

rm\_attribute\_name.is\_empty

[English]

In AML, an attribute owned by a <<ComplexObjectConstraint>> is implicitly the equivalen of an AOM C\_ATTRIBUTE, and either the subsetsProperty or redefinesProperty references an RM Property corresponding to rm\_attribute\_name. Thus, the implicit rm\_attribute\_name will never by empty, given other constraints which requie the Property to have a subsetsProperty or redefinesProperty and that Property must have a non-empty name.

* **Existence\_valid**

existence /= Void **implies** (existence.lower >= 0 **and** existence.

upper = 1)

[English]

In AML, existence is implicit and will have a lower, upper in the range of this invariant.

* **Children\_validity**

any\_allowed **xor** children /= Void

[English]

The AOM concept of any\_allowed type is modeled in AML using the type of a Property. If the type is a Class from the Reference Model (instead of being a <<ComplexObjectConstraint>>), then the implicit value of any\_allowed is true. The AOM concept of any\_allowed attribute is modeled in AML using a Property, where that Property is the only Property which subsets/redefines a reference model Property, and where the type of the Property is a Reference Model type (instead of being a <<ComplexObjectConstraint>>).

* **Cardinality\_valid**

cardinality /= Void **implies** is\_multiple

[English]

The AOM concept of is\_multiple is modeled in AML as a Property whose upper bound is not 0 or 1.

* **hildren\_occurrences\_lower\_sum\_validity**

(cardinality /= Void **and then not**

cardinality.interval.upper\_unbounded) **implies** occurrences\_lower\_sum = cardinality.

interval.upper

[English]

This AOM invariant is enforced by the UML constraints on the upper, lower values of a MultiplicityElement.

* **Children\_orphans\_validity**

(cardinality /= Void **and then not** cardinality.interval.

upper\_unbounded) **implies** minimum\_child\_count = cardinality.interval.

upper

[English]

This AOM Invariant corresponds to the UML semantic for a MultiplicityElement.

* **Differential\_path\_valid**

differential\_path /= Void **implies not**

differential\_path.is\_empty

[English]

In AML, the differential\_path is implicitly derived and will not be empty.

* **Alternatives\_valid**

not is\_multiple **implies** children.for\_all (co: C\_OBJECT |

co.occurrences.upper = 1)

[English]

In UML, the multiplicity of a Property must conform to the Property being subsetted/redefined. Since the occurrences of a type is modeled using the Property multiplicity, a reference model Property having upper=1 constrains all subsetted/redefined Properties to have an upper<=1.

* **Child\_occurrences\_validity**

cardinality /= Void **implies** cardinality.interval.intersects

(occurrences\_total\_range)

[English]

The AOM invariant is implemented by UML constraints regarding cardinality of subsetted/redefined Properties.

* **Cardinality Validity**

**not** interval.lower\_unbounded

[English]

UML types lower bound as Integer (instead of UnlimitedNatural). Thus, lower bound can not be unbounded.

* **Node\_id\_valid**

node\_id /= Void **and then not** node\_id.is\_empty

[English]

In AML, if the tag 'node\_id' has a value, then it must be an EnumerationLiteral. There are cases in which the tag has no value, such as for primitive types and other cases in which there is no term\_definition available..

* **Occurrences\_validity**

(occurrences /= Void **and** parent /= Void **and** parent.

is\_single) **implies** occurrences.upper = 1

[English]

This AOM invariant is enforced by the UML constraints covering cardinality of subsetted/redefined properties.

* **Sibling\_order\_validity**

sibling\_order /= Void **implies** specialisation\_depth > 0

and parent.is\_multiple

[English]

sibling\_order /= Void implies specialisation\_depth > 0 and parent.is\_multiple

* **sibling\_node\_id\_validity**

sibling\_node\_id /= Void

[English]

sibling\_node\_id /= Void

* **Default\_value\_valid**

has\_default\_value **implies** default\_value.conforms\_to\_type

(rm\_type\_name) **and** valid\_value (default\_value)

[English]

has\_default\_value implies default\_value.conforms\_to\_type (rm\_type\_name) and valid\_value (default\_value)

* **Consistency**

**not** any\_allowed

[English]

The AOM concept of C\_COMPLEX\_OBJECT\_PROXY is modeled in AML as a Property having an aggregation of none and a type of the target object within the same <<Archetype>> (i.e., an internal reference). The AOM concept of any\_allowed is modeled in AML as a Property whose type is in the Reference Model. It follows that the implicit AML  representation of an AOM C\_COMPLEX\_OBJECT\_PROXY will not  also represent the AOM any\_allowed concept.

* **Target\_path\_valid**

target\_path /= Void **and then not** target\_path.is\_empty

-- **and then** ultimate\_root.has\_path(target\_path)

[English]

The AOM concept of C\_COMPLEX\_OBJECT\_PROXY is modeled in AML as a Property having an aggregation of none and a type of the target object within the same <<Archetype>> (i.e., an internal reference). The AOM concept of target\_path is implicit in AML and is derived from the Property type.

#### ResourceAnnotationNodeItem [Stereotype]

**Description**

The annotations section of an archetype or template provides a place for node-level meta-data to be added to the archetype. This can be used during the design phase to track dependencies, design decisions, and specific resource references. Each annotation is keyed by IdentifiedItem id associated with the item being annotated.

The ResourceAnnotationNodeItem stereotype indicates that the base Comment is an annotation for the associated AML element. Annotations are language specific and are modeled as comments associated with AML ResourceTranslations, where the target language is identified by the referencing ResourceTranslation.

The UML annotatedElement attribute of Comment identifies the model element being annotated, which may be a Reference Model element or an ArchetypeTerm EnumerationLiteral within the Archetype IdentifierDefinition.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Comment

**Attributes**

**•** annotationNodeId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [1]

The identifier associated with the element being annotated. In ADL, this would be the "id", "ac" or "at" code associated with the element itself.

#### ResourceTranslation [Stereotype]

**Description**

The ResourceTranslation stereotype carries a translation of the identified elements in an archetype in a target language. ResourceTranslation also carries the language specific ADL2.0 [adl] metadata, including both the AOM2.0[aom] TRANSLATION\_DETAILS section, which is *about* the translation and the RESOURCE\_DESCRIPTION\_ITEM section, which carries language specific information about the Archetype itself.

ResourceTranslation is associated with an Archetype

In addition to providing translations and details for <<Archetype>> languages, the <<ResourceTranslation>> Enumeration (which is empty) serves as the super type for the language-specific term definitions and constraint definitions.

**Diagrams**

[Constraint Profile](#_791a7110fdfbf54fcba1948a5a8af829)

**Meta-classes**

UML Standard Profile::UML2 Metamodel::Enumeration

**Attributes**

**•** versionLastTranslated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

The ArchetypeVersion identifier of the version that was last translated.

**•** accreditation : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** other\_translation\_details : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** purpose : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [1]

**•** keywords : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** use : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** misuse : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** copyright : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**•** original\_resource\_uri : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** original\_resource\_uriId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** other\_details : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** other\_detailsId : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..\*]

**•** author\_name : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Name of translation author.

Corresponds to AOM author item where id='name'.

**•** author\_organization : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Name of author's organization.

Corresponds to AOM author item where id='organization'.

**•** author\_email : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Author's e-mail address.

Corresponds to AOM author item where id='email'.

**•** author\_date : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

Date author made translation.

Corresponds to AOM author item where id='date'.

**•** version\_last\_translated : UML Standard Profile::UML2 Metamodel::PrimitiveTypes::String [0..1]

**Constraints**

* **VOTM- terminology translations validity**

Translations must exist for term\_definitions and constraint\_definitions sections for all languages defined in the description / translations section.

[OCL]

self.base\_Enumeration.ownedLiteral->notEmpty()

# AML-UML Transformation Reference (Informative)

## Introduction

This clause provides component, structural and abstract orientation to the transformations between the UML Profile for AML and the AOM 2.0 Meta-model, as specified in [AOM]. The transformations are expressed in terms of OMG QVT [QVT]. The QVT and related metamodels and profiles are provided as machine-readable artifacts associated with this specification (see **Error! Reference source not found.**). This clause, and its associated QVT, are presented from a transformation engineering perspective and illustrate abstract model manipulation. Other clauses in this specification and/or informative artifacts associated with this specification provide illustrations of concrete target artifact syntax. The associated QVT are the normative expression for the mapping (in the sense defined in Clause **Error! Reference source not found.**). In case of apparent conflict between the informative orientation provided in this clause and the QVT, the QVT takes precedence.

### AML Provisioning Context

The transformations referenced in this clause are intended to constitute a provisioning process that enables representation of AOM 2.0 artifacts as AML-UML Models or in one of the native AOM-conformant formats, including XML. The overall provisioning process is illustrated in Figure 10‑1. The focus of this clause is to illustrate the transformation between AML-UML Models and AOM 2.0. The AOM 2.0 concrete artifacts addressed by these transformations are XML Documents conformant with the AOM 2.0 Archetype Schemas. The AOM architecture and tooling defines rendering of an AOM Model in multiple formats, including ADL and XML. A meta-model for Schemas is specified in Clause 10 (XML Schema InfosetModel) of the OMG MOF 2 XMI Mapping Specification [XMI]. A meta-model based on the AOM 2.0 Archetype Schemas is included in the machine-readable artifacts for this specification (see **Error! Reference source not found.**). AOM Artifacts provisioned by the transformations are represented (serialized) in their native XML form.

The Archetypes in a Library constrain a Reference Model. The AML-UML Profile does not specify any specific Reference Model. During transformation, the Archetypes are wired into UML representations of Reference Models. Examples of Reference Models include:

* *CIMI Reference Model.* The Reference Model whose target namespace is “*http://release.niem.gov/niem/structures/3.0/*”. A Reference Model used by the Clinical Information Modeling Initiative.
* *openEHR.* The NIEM NDR Schema whose target namespace is “*http://release.niem.gov/niem/proxy/xsd/3.0/*”. A Reference Model used by the openEHR community whose main focus is electronic patient records and systems.

The transformations use a set of shared, reusable libraries for:

* *PrimitiveTypes*. The UML Primitive Types library includes definitions for some of the Primitive Types supported by the AOM 2.0 meta-model: Boolean, String, Integer, and Real.
* *XML Primitive Types.* The UML XML Primitive Types library represents the data types defined in the XML Schema for Schemas. There is an isomorphic mapping between the types in the UML XML Primitive Type library and the explicitly defined SimpleTypeDefinitions in the Schema for Schemas. This type library is defined by the NIEM-UML Specification. The primary types referenced by AML-UML are the temporal types.

The AML-UML model which serves as source or target of a transformation is a «ArchetypeLibrary» Package.

* The AML Profiles are applied to the «ArchetypeLibrary» Package.
* The AML Profiles may import other Profiles and/or model libraries such as the XMLPrimitiveTypes.
* Some «ReferenceModel» is imported into the «ArchetypeLibrary». The Classifiers which are transitively owned by the «ReferenceModel» are constrained by Classifiers owned by the «Archetype»s within the «ArchetypeLibrary».

An AOM Model is an instance of an AOM 2.0 MOF Meta-model.

* The AOM Model is parsed-from/serialized-to an XML Document conformant with the AOM XML Schema.
* The AOM Architecture externalizes an Archetype Object Model in one of several forms. Based on AOM tools and specifications, an AOM XML Document may be translated to/from an ADL Specification.

There are two QVT «OperationalTransformation»s between an AML-UML Model and the AOM Model:

* adl2uml. Transforms a set of AOM XML Documents to an AML-UML «ArchetypeLibrary».
* uml2adl. Transforms an AML-UML «ArchetypeLibrary» to a set of AOM XML Documents.



Figure ‑ AML Provisioning Context

### QVT Packaging

The transformations referenced in this clause include:

* *adl2uml.* Transforms a library of AOM Archetype Documents to AML-UML.
* *uml2adl.* Transforms an AML-UML model to a library of AOM Archetype Documents.

Additionally, there are inherited common transformations:

* *AMLplatformBinding.* A set of platform-specific operations. For the purposes of this specification, these are defined as abstract operations.
* *AMLglobals.* A set of variables initialized at the beginning of the transformation, including references to Profiles and Stereotypes from AML-UML, and various constants referenced in the AOM 2.0 Specification.



Figure ‑ AML Transformations

### Transformation Reuse and Composition

Reuse and composition facilities are associated with QVT mapping operations. Disjunction enables selecting, among the set of disjunctive mappings, the first that satisfies the when clause and then invoking it. For the AML transformations, disjunction is used to identify a concrete MappingOperation to be selected from a given disjunctive MappingOperation. The disjunction hierarchy generally follows the AOM meta-model inheritance hierarchy and/or the UML meta-model inheritance hierarchy. Another reuse and composition facility associated with QVT mapping operations is inheritance. Inheritance enables reuse of the execution logic of an inherited mapping. Thus, disjunction is used to initially select a leaf mapping operation and inheritance is used to share common execution logic. For the AML transformations, inheritance is used to identify the hierarchy of execution logic required to populate target Elements from a source Element. The mapping inheritance generally follows the AOM meta-model inheritance hierarchy and/or the UML meta-model inheritance hierarchy. Figure 10‑3 illustrates the pattern of disjunction and inheritance used for the transformations.

* The notation «mapping» represents a QVT mapping operation.
* The notation «inherits» represents a QVT mapping inheritance.
* The notation «disjuncts» represents a QVT mapping disjunction.
* Only «mapping» operations with either inherits or disjuncts are included in the figure.
* The figure depicts «mapping» operations for the adl2uml transformation. The uml2adl transformation has a similar pattern of disjunction and inheritance.



Figure ‑ AML Transformation Disjunction and Inheritance

### Transformation Notation

Figure 10‑4 provides an example of how mappings are described for the transformations.

* Each figure depicts a related set of model concepts. Since the model mappings are largely isomorphic, a single figure is used to illustrate an AOM to AML transformation as well as an AML to AOM transformation.
* Each mapping figure has at least two models depicted, one being the AOM meta-model and the other being a representation of an AML-UML Model Instance. An AML-UML Model Instance is depicted as an actual AML-UML model fragment, when the UML graphical notation is appropriate. An AML-UML Model Instance may alternatively be depicted using UML Instance Specification notation, when there is no suitable UML graphical notation (as in the case of Value Specifications, Expressions, etc.). A Reference Model fragments is sometimes depicted as the third model.
* Each model is adorned with sample model notation used to depict concepts associated with that model.
* A QVT «mapping» is depicted as a Stereotyped Realization from the AOM meta-model to an instance of an AML-UML model. In cases where a Realization cannot be depicted, a Comment is shown annotating one or more model elements from the AOM meta-model and one or more instance model elements from the AML-UML model.
* Each QVT «mapping» is shown with the QVT mapping operation name. Details of the operation can be found in the associated QVT Files for this specification.
* Note that the figures in this clause are primarily intended as a high-level orientation to key «mapping»s of the QVTs. Neither the figures nor the accompanying narrative provide all detail associated with a mapping operation. For definitive information about fine-grained aspects of the mapping, please consult the associated QVT Files for this specification.



Figure ‑ AML Transformation Mapping Notation Overview

### Platform Binding

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

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

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

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

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

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

Determines whether the specified stereotype is applied to this element.

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

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

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

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

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

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

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

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

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

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

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

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

Context is a UML Multiplicity Element. The platform-specific operation sets the lower bound of the multiplicity interval.

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

Context is a UML Multiplicity Element. The platform-specific operation sets the upper bound of the multiplicity interval.

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

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

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

Context is any UML Element. The operation applies the specified stereotype to this element and returns an instance of the applied stereotype.

### Global Properties

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

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

## Archetype Library

The AML transformations are defined as a set of mappings between AOM Archetypes and AML-UML model elements. In general, there is a one-to-one correspondence between Elements in the AML-UML model and Elements in the AOM meta-model. At the highest compositional level defined within the AOM architecture, an archetype library is a container for a set of AUTHORED\_ARCHETYPEs. Figure 10‑5 illustrates the high-level packaging map between an AOM Archetype Library and an AML-UML model in the context of a UML Reference Model.

* A mapping is defined between a file system folder and an «ArchetypeLibrary» Package. Each AOM AUTHORED\_ARCHETYPE corresponds to a document within the file system folder and maps to an «Archetype» Package. Based on the rmPublisher and rmVersion, the «ArchetypeLibrary» is bound (via import) to some «ReferenceModel». The «ArchetypeLibrary» has the AML Profiles applied. The «Archetype»s within an «ArchetypeLibrary» must have the same rmPublisher and rmPackage. While the rmPublisher is specified in the «ReferenceModel», the logical notion of rmPackage is recorded in an «ArchetypeLibrary» tag.
* A mapping is defined between each AUTHORED\_ARCHETYPE document and an «Archetype» Package. 

Figure ‑ «ArchetypeLibrary» Mapping Overview

## Archetype

Figure 10‑6 illustrates mappings between AOM and AML related to an «Archetype» Package.

* An AML «Archetype» has tag definitions to capture information from an AOM AUTHORED\_ARCHETYPE. «Archetype» tags include attributes inherited by AUTHORED\_ARCHETYPE as well as those contained by some associated Classifiers.
* The AOM ARCHETYPE parent\_archetype\_id is represented as an import from one «Archetype» to another «Archetype» within the same «ArchetypeLibrary».
* The name of the «Archetype» corresponds to a concept\_id in the AOM meta-model. The AOM AUTHORED\_ARCHETYPE attributes rmPublisher and rmPackage are derivable from «ReferenceModel» and «ArchetypeLibrary», respectively. The rmClass attribute defined in AOM is derivable from the top level «ArchetypeDefinition» Usage. The remaining components of the physicalId attribute in AOM are captured as tags in the «Archetype».



Figure ‑ «Archetype» Mapping Overview

## Terminology Definition

Figure 10‑7 illustrates mappings related to Terminology Definitions.

* For an ARCHETYPE in AOM, there is exactly one terminology. The type of the terminology is ARCHETYPE\_TERMINOLOGY. That singleton ARCHETYPE\_TERMINOLOGY is represented in AML-UML as a package named “ontology”, nested within the «Archetype» Package.
* For an ARCHETYPE in AOM, natural languages have meta-data defined in TRANSLATION\_DETAILS and RESOURCE\_DESCRIPTION\_ITEM. The natural language is specified in a TERMINOLOGY\_CODE, which contains a combination of terminology\_id and a language code. In AML-UML, the terminology\_id is modeled as a Package containing a «ResourceTranslation» corresponding to each language code. In the example below, the terminology\_id is ISO\_639-1. The mapping from TERMINOLOGY\_CODE to Package is performed by the QVT «mapping» LanguagePackage.
* The language code of an AOM TERMINOLOGY\_CODE is mapped to a «ResourceTranslation» via the QVT «mapping» LanguageEnumeration. The mapping merges information from RESOURCE\_DESCRIPTION\_ITEM and TRANSLATION\_DETAILS. Thus, a «ResourceTranslation» contains tag definitions which encompass the language-specific AOM meta-information contained in both RESOURCE\_DESCRIPTION\_ITEM and TRANSLATION\_DETAILS.
* The terminology\_id Package (e.g., ISO\_639-1) contains an «EnumeratedValueDomain» Enumeration named IdentifierDefinition. The contents of the IdentifierDefinition are a set of node identifiers corresponding to the ids in an AOM Terminology Definition. These identifiers are used to associate Archetype Classifiers to multiple natural Languages, terminology bindings, and value sets. The EnumerationLiterals in this Enumeration are referenced as the “id” for Archetype Classifiers and other «IdentifiedItem»s, including the EnumerationLiterals within a «ResourceTranslation».
* The AOM AUTHORED\_RESOURCE attribute “original\_language” has a QVT «mapping» to a Usage named “original\_language” between an «Archetype» and the «ResourceTranslation » corresponding to that original\_language.
* Similarly, the AOM ARCHETYPE\_TERMINOLOGY attribute “original\_language” has a QVT «mapping» to a Usage named “terminology\_original\_language” between an «Archetype» and the «ResourceTranslation » corresponding to that original\_language.



Figure ‑ Terminology Definition Mapping Overview

## Terminology Binding

The ARCHETYPE\_TERMINOLOGY component of the AOM Model provides for multi-lingual terminology definitions, bindings of terminology to technology, and local value set constraints.

* The AOM ARCHETYPE\_TERMINOLOGY (of which there is one per «Archetype») has a term\_bindings «mapping» to a Package named “term\_bindings”. The term\_bindings Package owns all the «ValueSetDefinitionReference»s used to define terminology bindings.
* The AOM ARCHETYPE\_TERMINOLOGY attribute named “term\_definitions” is a set of tables keyed by language. Each entry in the term\_definitions set has a QVT CodeDefinitionSet «mapping» to a «ResourceDefinition» whose name is the language key.
* The columns of the table keyed by language are “id”, “text”, and a “description”. Each row of the table has an ARCHETYPE\_TERM «mapping» to an «IdentifiedItem» within the language’s «ResourceDefinition». The AOM “text” is mapped to the «IdentifiedItem» name and the “description” is mapped to the body of the ownedComment.
* The «IdentifiedItem» has an “id” tag whose value is the corresponding «IdentifiedItem» identifier within the IdentifierDefinition. AOM rules require that each language include text/description for all identifiers, so each language will have a term/definition for each «ARCHETYPE\_TERM» in the IdentifierDefinition.
* The AOM ARCHETYPE\_TERMINOLOGY attribute named “term\_bindings” is a set of tables keyed by a technology identifier. Each entry in the term\_bindings set has a QVT TermBindingSet «mapping» to a «ValueSetDefinitionReference» whose name is the technology identifier.
* The columns of the table keyed by terminology identifier are “id” and “uri”. Each row of the table has a TERM\_BINDING\_ITEM «mapping» to a «ConceptReference» within the terminology’s «ValueSetDefinitionReference». The AOM “id” is mapped to the «ConceptReference » name and the “uri” is mapped to the “uri” tag.
* The “id” tag from the AOM model corresponds to an «ARCHETYPE\_TERM» owned by the IdentifierDefinition. The «ARCHETYPE\_TERM» tag named “term\_bindings” references the term binding «ConceptReference».



Figure ‑ Terminology Binding Mapping Overview

## Local Value-Sets

The ARCHETYPE\_TERMINOLOGY component of the AOM Model provides for definition of local value set constraints in terms of «ARCHETYPE\_TERM» identifiers.

* The AOM ARCHETYPE\_TERMINOLOGY attribute named “value\_sets” is a set of “at” lists keyed by an “ac” identifier. Each entry in the value\_sets set is used to populate the “value\_set\_members” tag of an «ARCHETYPE\_TERM» within the IdentifierDefinition. The «ARCHETYPE\_TERM» whose name corresponds to the “ac” key of a value set is located. Each “at” identifier also has an «ARCHETYPE\_TERM» with a matching name. The value\_set\_members tag of the “ac” «ARCHETYPE\_TERM» is set to the list of “at” «ARCHETYPE\_TERM»s.



Figure ‑ Local Value-Sets

## Archetype Definition

An AOM ARCHETYPE has a distinguished C\_COMPLEX\_OBJECT which is the “definition” of an ARCHETYPE. The overall structure of an Archetype in AOM is basically a structure where objects contain attributes which contain objects, etc. Each Complex Object is a constraint of a Reference Model Classifier, and each attribute is a constraint on a Reference Model attribute.

* The AOM ARCHETYPE attribute named “definition” is a C\_COMPLEX\_OBJECT which is the root of a logical containment structure. The “definition” attribute itself has an ArchetypeDefinition «mapping» to an «ArchetypeDefinition » Usage from the «Archetype» Package to a «ComplexObjectConstraint» Classifier. Part of the AOM physical\_id attribute is the rm\_class, which is derived from the «Constrains» Classifier of the Classifier identified by the «ArchetypeDefinition » Usage.
* The «ComplexObjectConstraint» Classifier is mapped from an AOM C\_COMPLEX\_OBJECT via the C\_COMPLEX\_OBJECTAbstract «mapping». The «Constrains» Generalization is mapped from the rm\_type\_name of the AOM C\_OBJECT. The name of the «ComplexObjectConstraint» Classifier will be set to the term name associated with the node\_id in C\_OBJECT, if possible. The AOM C\_DEFINED\_OBJECT is\_frozen attribute is mapped to the UML Classifier isLeaf attribute. The AOM node\_id attribute of C\_OBJECT is mapped to the “id” tag of «ComplexObjectConstraint», which will have a value of the corresponding «ARCHETYPE\_TERM» in IdentifierDefinition.
* A Property is mapped from a C\_ATTRIBUTE via the P\_C\_ATTRIBUTE «mapping». The mapping will subset or redefine the Reference Model Property identified by rm\_attribute\_name of the AOM C\_ATTRIBUTE. The AOM C\_ATTRIBUTE will be mapped to multiple UML Properties, one for each of the defined children of \_C\_ATTRIBUTE. Cardinality of each such UML Property is determined by occurrences information in the child C\_OBJECT. A Property will nominally have composite Aggregation, unless it is a Proxy (in which case Aggregation is “none”). The Property is an Association End of a newly created Association. The type of the new Property is defined by the children, and is nominally a Classifier within the «Archetype». If the child itself does not have any constraining attributes, then the type of the Property is the same as that specified in the Reference Model.



Figure ‑ Archetype Definition Mapping Overview

## Object References

An AOM C\_OBJECT has specializations which provide for some variances in how objects may be referenced, reused, or constrained.

* The AOM ARCHETYPE\_SLOT is mapped to an «ArchetypeSlot » Classifier via the ARCHETYPE\_SLOT «mapping ». The «Constrains » Generalization may optionally be to a Classifier in a parent Archetype (however, the example below does not override a parent definition). The includes, excludes attributes of the AOM ARCHETYPE\_SLOT are mapped to a Constraint . The element constrained is the Property whose type is the «ArchetypeSlot » Classifier. In the example below, the Constrained element is the Property named “id97” within the Classifier named “Discharge delayed”.
* An AOM C\_COMPLEX\_OBJECT\_PROXY is essentially a reference to a Complex Object within the same Archetype. A C\_COMPLEX\_OBJECT\_PROXY effects the mapping of a Property. There is no Classifier created for a C\_COMPLEX\_OBJECT\_PROXY. Instead, the type of a Property is set to the target specified by the target\_path attribute of C\_COMPLEX\_OBJECT\_PROXY and the aggregation of the Property is set to “none”. In the example below, there is a «ComplexObjectConstraint » with id108 which was referenced via containment from id105. There is a Property on id120 which references id108 with no aggregation, corresponding to the AOM C\_COMPLEX\_OBJECT\_PROXY with a target\_path of id108. The original id of C\_COMPLEX\_OBJECT\_PROXY is retained in the «ObjectConstraint » placed on the Property with no aggregation.



Figure ‑ Object Reference Mapping

## Primitive Constraints

Constraints on an AOM C\_PRIMITIVE\_OBJECT are mapped to a UML Constraint on the Property whose type maps to a C\_PRIMITVE\_OBJECT.

* A C\_PRIMITVE\_OBJECT maps to a Constraint. A Property whose type maps to the C\_PRIMITIVE\_OBJECT is the constrainedElement of a Constraint. The Constraint is an ownedRule of the Classifier owning the Property.
* The Constraint has a specification which is an Expression. The symbol for these primitive expressions is normally “or” and the operands are either discrete literal values or Intervals.
* The assumed\_value of a C\_PRIMITIVE\_OBJECT maps to some Literal ValueSpecification which is the defaultValue for the Property.
* The AOM concept of an Interval constraint on a C\_ORDERED Primitive is mapped to a UML Interval, with Literal ValueSpecifications for the min and max of the Interval.
* A C\_STRING mapping has Expression operands and a defaultValue which are LiteralString.
* A C\_BOOLEAN mapping has Expression operands and a defaultValue which are LiteralBoolean.
* A C\_REAL mapping has Expression operands which are Intervals, where the min and max are LiteralReal.
* A C\_INTEGER mapping has Expression operands which are Intervals, where the min and max are LiteralInteger.



Figure ‑ Primitive Constraints

## Temporal Constraints

AOM Constraints on Temporal Primitives are specializations of constraints on ordered Primitives. As such, the AOM Temporal Primitive map to UML Constraints on a Property. The Constraint will have an “or” Expression with operands. The operands, in this case, will be TimeIntervals. The min/max will be TimeExpressions, with the exception of Duration, where the ValueSpecification is specified as min/max Duration.



Figure ‑ Temporal Constraints Mapping Overview

## Code Constraints

AOM Constraints on Codes are specializations of constraints on Primitives. As such, the AOM Code Constraint maps to UML Constraints on a Property. The Constraint will have an “or” Expression with operands. The operands and/or default Values, in this case, will be InstanceValues where the instance is an EnumerationLiteral.



Figure ‑ Code Constraint Mapping Overview

## Assertions

AOM Assertions may be placed on an Archetype as a whole or as the includes/excludes lists of an ARCHETYPE\_SLOT. The Assertions are mapped to UML Expression trees and become part of the Constraint specification for an «ArchetypeSlot » or «Archetype ».

* An AOM ASSERTION has one EXPR\_ITEM. The AOM EXPR\_ITEM is mapped via EXPR\_ITEM «mapping » to a UML Expression. The name of the Expression is from the AOM ASSERTION tag. The type of the Expression is derived from the AOM EXPR\_ITEM type.
* An AOM EXPR\_OPERATOR is mapped to a UML Expression via EXPR\_OPERATOR «mapping » (which inherits EXPR\_ITEM «mapping » ). The UML Expression symbol (i.e., operator) is derived from the kind of EXPR\_OPERATOR operator. An AOM EXPR\_OPERATOR is specialized into EXPR\_UNARY\_OPERATOR and EXP\_BINARY\_OPERATOR (with corresponding specializations of the QVT mappings).
* An AOM EXPR\_UNARY\_OPERATOR is mapped to an Expression with a single operand using the QVT «mapping » EXPR\_UNARY\_OPERATOR.
* An AOM EXPR\_BINARY\_OPERATOR is mapped to an Expression with a left operand and a right operand using the QVT «mapping » EXPR\_BINARY\_OPERATOR.
* An AOM EXPR\_LEAF is a specialization of EXPR\_ITEM. The type of the Expression is derived from the AOM EXPR\_ITEM type. The symbol (e.g., operator) is derived from EXPR\_LEAF.reference\_type.



Figure ‑ Assertions Mapping Overview