An OMG® Archetype Modeling Language™ Publication

Archetype Modeling Language™

Version 1.0

OMG Document Number: formal/2018-02-01

Release Date: March 2018

Standard Document URL: http://www.omg.org/spec/AML/1.0

Normative Machine Consumable File(s):

http://www.omg.org/spec/AML/20150501/ConstraintProfile.xmi
http://www.omg.org/spec/AML/20150501/ReferenceModelProfile.xmi
http://www.omg.org/spec/AML/20150501/TerminologyProfile.xmi

Informative Machine Consumable File(s):

http://www.omg.org/spec/AML/20150501/AMLGlobals.qvto
http://www.omg.org/spec/AML/20150501/AMLplatformBinding.qvto
http://www.omg.org/spec/AML/20150501/adl2uml.qvto
http://www.omg.org/spec/AML/20150501/uml2adl.qvto

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

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://issues.omg.org/issues/create-new-issue).

Table of Contents

Preface

OMG

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

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

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

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

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

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA

Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

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

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification via the report form at:

http://issues.omg.org/issues/create-new-issue

This page intentionally left blank.

1 Scope

1.1 Overview

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.

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.

1.2 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.

1.3 AML Profiles

The AML is specified by three UML profiles collectively 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.

2 Conformance

2.1 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

2.2 AML Reference Model Profile

Sub clause 8.2 of this specification defines the AML Reference Model Profile.

2.3 AML Terminology Binding Profile

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

2.4 AML Constraint Model Profile

Sub clause 8.4 of this specification defines the AML Constraint Model Profile. The Constraint Model Profile imports both the Reference Model Profile and Terminology Binding Profile.

3 Normative References

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

[ADL]	openEHR *Archetype Definition Language: ADL 2*, Revision 2.0.5, http://www.openehr.org/releases/trunk/architecture/am/adl2.pdf
[AOM]	*openEHR Archetype Object Model* (AOM), Revision 2.1.14, 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
[AQL]	Archetype Query Language Description https://openehr.atlassian.net/wiki/idsplay/spec/Archetype+Query+Language+Description
[ARCH]	*openEHR Archetypes: Constraint-based Domain Models for Future-proof Information Systems*, http://www.openehr.org/publications/archetypes/archetypes_beale_oopsla_2002.pdf
[CEM]	*Standards for detailed clinical models as the basis for medical data exchange and decision support. Int J Med Inf*, 69(2-3), 157-74.
[CIMI]	CIMI Reference Model Requirements, http://informatics.mayo.edu/CIMI/index.php/CIMI_Reference_Model_Requirements
[CKM]	openEHR Clinical Knowledge Manager http://openehr.org/ckm/
[CTS2]	OMG *Common Terminology Service 2 (CTS2)*, 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
[ISO13606-2]	*Health informatics — Electronic health record communication Part 2:Archetype interchange specification,* 2008-12-01
[KAI]	openEHR Knowledge Artefact Identification, Revision 0.7.5, http://www.openehr.org/releases/trunk/architecture/am/knowledge_id_system.pdf
[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/
[NIEM]	OMG *UML Profile for NIEM Version 1.0*, 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/

4 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 'ADL2.'
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.

5 Symbols

5.1 Graphical Symbols

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

5.2 Abbreviations

ADL	Archetype Definition Languge
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
CEM	Clinical Element Models
CIM	Clinical Information Model
CIMI	Clinical Information Modeling Initiative
CKM	Clinical Knowledge Manager
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
ISO13606-2	Archetype interchange specification
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

6 Additional Information

6.1 Acknowledgements

The following companies submitted this specification:

- Mayo Clinic
- Visumpoint, LLC

The following companies supported this specification:

- Escape Velocity, LLC

The following individuals aided the development of this specification.

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

7 ADL, AOM, and AML (Informative)

7.1 Overview

This clause describes the relationship between the Archetype Definition Language (ADL) [ADL], the Archetype Object Model (AOM) [AOM], and the Archetype Modeling Language (AML) specification.

7.2 Business Purpose

The Archetype formalism, comprising the Archetype Definition Language (ADL) and its sibling specification Archetype Object Model (AOM) were devised by the openEHR Foundation as part of an approach to account for the need to accommodate 'domain semantics' and 'domain models,' which are numerous and highly variable, while preserving existing 'information models' where the latter are understood as the definition of data / instances, in the orthodox object-oriented and relational manner. The same need was recognized by the CEN and ISO committees in health with the result that AOM became an ISO standard (13606-2) [ISO13606-2] in 2008. The same need was identified since 2011 by the Clinical Information Modeling Initiative (CIMI) [CIMI], which chose the latest version of ADL/AOM as its modeling formalism. Independently of this lineage of development, Intermountain Healthcare developed over many years a system of domain content modeling known as Clinical Element Models (CEMs) [CEM] which in its technical form and tooling approach is very close to the Archetype approach, so much so that inter-conversion from CEMs to Archetypes are available today, and Archetype → CEM convertibility is imminent.

In the following, the term 'Archetype' can be assumed to also stand for Intermountain CEMs.

To make the distinction between domain and information models concrete, information models in openEHR, CIMI, and more generally in e-health typically define things like 'clinical data types,' such as Quantity (with units, accuracy, etc.), Coded text, Ordinal (an Integer/symbol conjunction), and fairly generic clinical structures, such as 'clinical statement' (often denoted by the type Entry), clinical document, report, and so on. Such a class model may contain 50-100 classes, including 20+ classes for the clinical data types. This enables the construction of instance structures corresponding to the various parts and sections of e.g., a clinical encounter note or a hospital discharge summary. However, neither a class model of this size, nor the capabilities of standard UML 2.5 can naturally accommodate the explosion of diversity of possible values of instances which can make up a clinical document created in any particular situation (e.g., a specific kind of patient visiting a specialist), for example the tens of thousands of clinical observations (e.g., 'systolic blood pressure,' 'visual acuity,' etc. many of them consisting of multiple data points in specific structures), or the O(100k) laboratory analyte result types. Further, the size of terminology needed to annotate data items, both 'names' and 'values' in a name/value understanding of the data is in the O(100k) concepts range, as exemplified by the SNOMED CT and ICD11 terminologies.

The above situation applies across most information-rich industries, with varying but generally very large numbers; health is used here just as a convenient example.

Although technically these numerous possible values could just be understood as the specific values that 'happen to occur' in a situation of data creation, it is widely understood within IT in general that domain data value 'complexes' (co-occurring structures of data) correspond to meaningful patterns that constitute a relatively small fraction of the astronomical number of *possible* combinations of values within structures. Thus, while some tens to hundreds of thousands of 'clinical statement' patterns would adequately cover nearly all of general medical data recording (i.e., leaving the terminal real world values such as actual blood pressure open, within their respective sanity ranges), the information models in typical use would permit possible instance structures in the O(1E10) and higher ranges. In other words, most *possible* data constructions are garbage.

It is also widely recognized that mechanisms are needed to enable some sort of domain level 'modeling' or 'templating,' to enable the common patterns to be defined, and thus to allow the creation of software or other mechanisms (e.g., pre-

built UI forms) to limit the possible instance structures to those that actually make sense. The general need was identified in Martin Fowler's 1991 publication 'Analysis Patterns,' in which 'patterns' are illustrated in 'above the line' parts of UML diagrams, but has been known for some decades. It is generally understood that this kind of modeling cannot simply be an extension of the existing software or database schemata; if it is, it implies endless maintenance and updating of deployed software, and worse, frequent database migration. In systems operating 24x365, and routinely creating Terabytes of data per year per hospital, this is not an acceptable approach.

Consequently, most large system software products in the health and other domains have some kind of configuration or template building tool(s) that enable modeling of typical domain content patterns (often conceived of as forms).

The problem to date has been that no such capability is available independent of particular software products (specific vendors), concrete forms (UI forms, XSDs, etc.) or domains (e.g., process and control systems engineering have domain specific languages) – i.e., even tools that may be technically powerful enough are buried inside specific products, and are usually targeted to the database schemas of the product.

An important economic factor is that the creation of good quality domain models is time-consuming and expensive, relying as it does on domain experts – typically experienced clinicians, engineers, etc. – rather than IT staff. If models are created inside a specific product (e.g., a particular hospital information system), and that product is replaced, there is often little appetite or availability of the staff to recreate the work done to create the models/templates created in the first product. Multiplied across products, sites, and whole industry verticals, the lack of standard ways of representing models of domain content has become a significant blockage to the production of high quality information systems. Instead, as each solution is replaced, its domain models usually die with it.

The need for an efficient, formal, and product- and format-independent domain modeling capability is therefore clear. The sheer numbers of content patterns / models in health have led to the creation of an approach, centered around the Archetype formalism, used in conjunction with available terminologies (i.e., SNOMED CT, LOINC, ICDx, and many others).

The archetype formalism primarily addresses the expression of models of possible *data instance structures*, rather than higher level concepts such as workflows, clinical guidelines (which are decision graphs), and so on, although its general approach can be applied to any of these, i.e., the use of a model of 'what can be said' and a formalism or mechanism for *constraining* possibilities to the meaningful subset.

The openEHR ADL/AOM formalism is designed to be independent of any specific information model (known as a 'reference model'), product, technical format, or industry vertical. It is designed so instances of the formalism, known as Archetypes, can be computationally processed into desired output forms corresponding to specific technology environments. This is routinely performed in openEHR and Intermountain tooling environments.

It also supports two distinct types of domain content models, relating to a universal need, which is to be able to represent both use-independent definitions of 'data points,' and use-case dependent definition of 'data sets.' Consider the case of recording patient vital signs. Assume that a content model can be defined for 'blood pressure,' 'heart rate,' and 'blood oxygen.' These definitions need to be independent of specific uses such as patient home measurement, GP encounter, and hospital bedside measurement, since in all these cases, the blood pressure etc. are recorded in exactly the same way. However in each case, these vital signs data points are recorded *within* a larger data set of items that correspond to the health system event occurring, such as a GP patient health checkup. Thus there are two related needs: to be able to model domain data items and structures, and secondly, to be able to model larger structures in which they may occur. The alternative would be to create a domain model for every data set and within many of these models, to repeatedly create the same sub-model of recurring content, such as blood pressure. The former approach results in two layers of domain models: reusable data point models (Archetypes), and use-case specific data-set models (Templates, in ADL parlance).

7.3 Technical Aims of ADL / AOM

The ADL/AOM specifications published by openEHR [ADL], and later adopted in various forms by ISO and CIMI, take the following technical approach to domain content modeling:

- Domain content models are separated into two layers – re-usable Archetypes and use-case specific data-set models, known as Templates.
- A single formalism is used for all models: ADL syntax and its parse-tree equivalent AOM; a Template is understood as a specific kind of Archetype, constructed of elements chosen from specific Archetypes.
- The formalism is designed on the basis of constraints on a reference model (i.e., any standard UML information model), such that instances of the domain models (i.e., actual Archetypes or Templates) are guaranteed to be legal technical instances of the underlying reference model.
- The ADL and AOM expressions of the formalism structurally follow the graph nature of instance networks resulting from class model instantiation, that is to say, ADL is a block-structured syntax, and the AOM defines equivalent in-memory graph structures that relate to corresponding structures from the underlying Reference Model.
- The formalism is independent of natural language, and can accommodate domain models in any language, as well as translation into other languages.
- The formalism accommodates 'bindings' to any terminology, enabling the relationship between semantic entities (terminology concepts and ontology entities) to be formally expressed.
- Specialization and Composition between models are supported, in similar ways to inheritance and association in UML.
- Every individual element in an Archetype or Template is identified by a path that can be used to create statements in a query language for data retrieval.
- Various structured, multi-lingual meta-data are supported, including language, translation details, purpose, use, misuse, keywords, IP-related meta-data, and annotations.

The specifications of ADL and AOM can be referred to for details, but one key feature of the formalism is worth pointing out here: it relies systematically on a simple conjunction of reference model class names with codes, representing domain entities. The following fragment of ADL illustrates this. The names **CLUSTER**, **ELEMENT** and **DV_QUANTITY** are type-names from the openEHR Reference Model, while the codes [id3], [id22], etc. stand for domain semantic definitions such as 'Blood pressure measurement' and so on, as shown in the comments (the actual code definitions are in the lower part of the archetype definition, not shown here). This simple device allows, for example, two **ELEMENT** objects to be marked as representing two types of blood pressure. In its general form, it can be understood as a way of marking standard building block instances as being different parts of a domain instance structure, such as a medical result or complex document.

```
CLUSTER[id3] matches {      -- Blood pressure measurement
 items matches {
  ELEMENT[id22] matches { -- systolic blood pressure
    value matches {
          DV_QUANTITY[id35]
    }
  }
  ELEMENT[id23] matches { -- diastolic blood pressure
```

```
        value matches {
            DV_QUANTITY [id37]
        }
    }
}
```

This ‘concept-marking’ of nodes is applied universally throughout an Archetype, and where nodes have siblings, the codes are defined in an Archetype-local terminology.

An additional specification defines the structure and semantics of Archetype identifiers, versioning and lifecycle [KAI].

Functionally, archetypes and templates are used at design time to define domain content models, and at runtime for two purposes:

- Creating initial instance structures (from Templates); these must be by definition correct domain content structures, assuming the Archetypes are correct and complete.
- Validating previously created data retrieved from a data source or message channel, including data not originally created using Archetypes.

In openEHR, a third key function, querying, is performed using queries in the Archetype Query Language (AQL) [AQL], written solely based on Archetype paths and Reference Model relations, but independent of physical data storage schema.

These uses of Archetypes and Templates provide a basis for lifting data processing to a domain semantic level, from what would otherwise be a syntactic level; it enables higher level functionality such as decision support and business intelligence to reliably refer to domain semantic entities rather than trying to match in *ad hoc* ways.

To provide a practical idea of use to date, there are nearly 500 openEHR archetypes, with an average of 15 data points per archetype published – a total of around 7,500 substantive clinical data point definitions - on the openEHR.org Clinical Knowledge Manager (CKM) repository [CKM]. Additionally some thousands of archetypes have been created in national repositories in certain countries, and also within vendor products. Many thousands of Templates and AQL queries are constructed from this base of archetypes, and are operating in deployed openEHR systems around the world.

The Intermountain internal CEM repository [CEM] has around 6,500 CEMs, each with one substantive data point (the granularity is finer). The CEMs bind directly to Intermountain’s own controlled terminology, and are used to build Template equivalents, known as CE-Types.

7.4 Technical Aims of AML

AML’s purpose is to provide the capabilities of the Archetype Object Model (AOM) in a native UML environment. Due to the way UML works, the technical aims can be understood in somewhat different, although equivalent terms.

The starting point is the same, i.e., a generic but otherwise orthodox UML information model, acting as the Reference Model.

The key difference between the native AOM approach and AML is that the latter converts the explicit conjunction of a class name with a domain code into a new class name that corresponds to the meaning of the code. For example, a Reference Model class Element could be subtyped in AML to the class SystolicBloodPressureElement.

The AML profiles and stereotypes enable the equivalent definition of local terminology as described above to be done, along with the definition of classes representing the nodes. An AML Archetype will therefore be isomorphic with the equivalent ADL/AOM archetype i.e., same node structure in the definition part. Other parts of the profile support the definition of the same meta-data items as defined by the AOM, enabling an AML Archetype to be treated as a 'model' in its own right.

This page intentionally left blank.

8 Profiles

8.1 Overview

There is a need for information interoperability between health entities. Information needs to be shared between organizations and across international boundaries. The inability to share this information in a repeatable manner greatly affects the quality of care provided. The Clinical Information Modeling Initiative (CIMI) has the potential to be a disruptive innovation in eHealth. By providing the AML specifications for the representation of health information content, semantically interoperable information may be created and shared in health records, messages, and documents. The CIMI initiative affords the opportunity to enable the storage of lifelong health information; simplify data exchange, aggregation, querying and analysis; and support knowledge-based activities such as decision support. This will be achieved through the development of non-proprietary, common, and fully defined information models of clinical content and known transformations.

The clinical reference model (an instance of a reference model), clinical archetype models, and associated terminology will serve as the domain vocabulary for clinical information. The syntax and semantics defined by these clinical archetype models shall be maintained by users, in a common language that can be consistently understood and shared. The AML Profile enables the creation, definition, and use of this common language in UML.

The purpose of the AML Profile is to enable a UML ecosystem that supports and underpins CIMI activities through the use of adopted standards. The AML Profile provides a clear, consistent means of designing clinical models using UML, where tool vendors may add additional value/usability; clinical modeling concepts are separated from specific solutions (ex. XML, JSON, DB schema, etc.); and the creation of open source solutions is enabled.

The AML Profile:

- Specifies a collection of complementary UML profiles that work together to support the creation of CIMI content models.
- Supports the specification of CIMI content models in UML, such that they can be translated into AOM 2.0.
- Is capable of being used in other domain areas, with other reference models.
- Is capable of being used in developing specific implementations of CIMI content models using platform specific solutions (e.g., Clinical Document Architecture (CDA), openEHR, etc.).

The AML Profile provides consistency by ensuring that a UML representation of a CIMI model produced by one developer can be accurately interpreted by developers, modelers, and transformations. It offers completeness by ensuring that a developer can produce a UML representation of any CIMI reference and constraint model. Finally, the AML Profile offers practicality by ensuring that a developer/modeler can develop a CIMI compliant clinical model by employing the profile in current UML modeling tools.

Within the healthcare community the pattern of creating a common model that is reused by others to create specialized models through constraining the original model is often referred to as reference / constraint modeling. The reference model consists of syntax-neutral and technology- independent building blocks that can be used for data modeling. Major benefits of this approach include improved reuse of existing data artifacts and improved enterprise interoperability.

The AML profile provides a family of UML sub-profiles that enables the representation of semantic-based information models and addresses the problem of the lack of semantic interoperability within and between applications and databases in healthcare computing environments, including across enterprises and national borders. The AML Profile is the aggregation of three sub-profiles:

1. The Reference Model Profile (RMP)
2. The Constraint Model Profile (CMP)

3. The Terminology Binding Profile (TBP).

Traditionally, a single model containing all the required information concepts is designed for a specific application, transport, and database without regard to interoperability. Standards for the exchange of that health data between applications and databases have been focused on static message definitions that have not enabled a sufficient degree of interoperability or flexibility. They have not enabled 'single-source' modeling, whereby a single definition (e.g., a microbiology lab result) can be re-used for multiple purposes, such as a message definition, a document definition, a screen display form, a screen data capture form, or a report.

A more flexible and interoperable way of standardizing business semantics has long been required.

This specification provides a means for developing a common set of semantic building blocks that represent the general types of healthcare data in use today. The solution should provide an approach for the creation of new healthcare information models and the semantic binding of these information models to published terminologies to achieve semantic interoperability of data.

8.2 Dependencies

Figure 8.1 shows the AML sub-profile dependencies. The TerminologyProfile provides a generic set of extensions that allow UML model elements to be identified, designated, and associated with ontological concepts that identify the intended meaning of the model elements, enumerations, and associated value sets.

The ReferenceModelProfile identifies the set of elements in a UML Reference Model that can be further constrained via the ConstraintProfile. It also provides a mechanism to associate primitive type constaints in the constraint profile with the corresponding elements in the UML reference model.

The ConstraintProfile provides mechanisms for constraining the names, cardinality, types, and possible values of elements in the UML Reference Model.

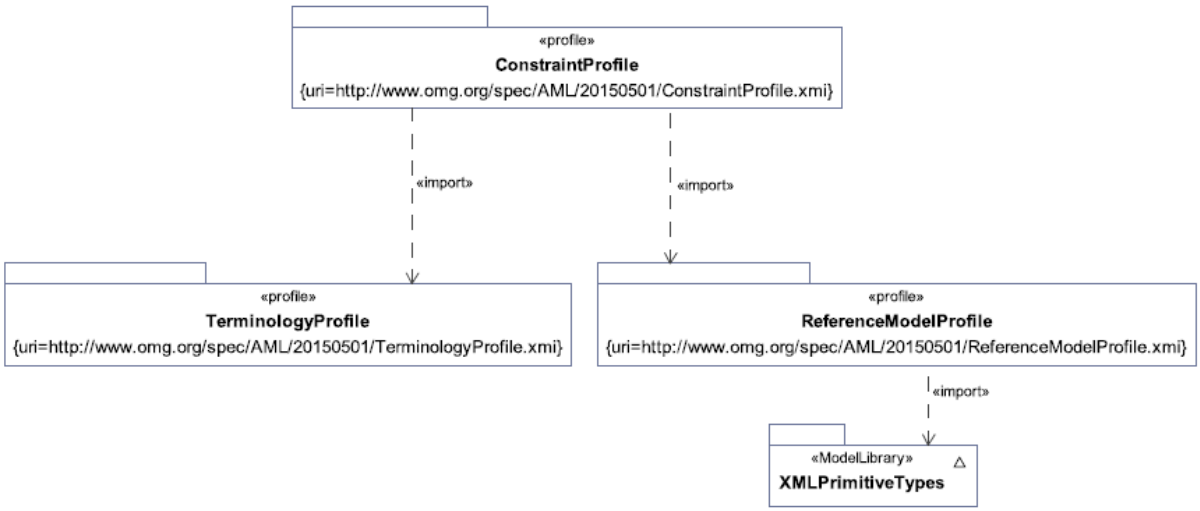

Figure 8.1 - AML Dependencies

8.3 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.

Figure 8.2 - Reference Model Profile

8.3.1 Infrastructure [Stereotype]

Description

An «Infrastructure» Property models an Archetype implementation aspect such as a specific Archetype identifier. Properties with an applied Infrastructure Stereotype cannot be constrained in AML.

Diagrams

Figure 8.2 Reference Model Profile

Meta-classes

UML::Property

8.3.2 MappedDataType [Stereotype]

Description

A «MappedDataType» Abstraction specifies the AML Primitive Type abstraction for a Reference Model Classifier. 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 Reference Model Classifier and its AML Primitive Type counterpart. Note that AML primitive Archetype Constraints are defined with respect to AML Primitive Types even when the Type being constrained is a Reference Model Type.

Diagrams

Figure 8.2 Reference Model Profile

Meta-classes

UML::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')))
```

8.3.3 ReferenceModel [Stereotype]

Description

A «ReferenceModel» Package defines the complex data types and structural patterns that can be constrained by a collection of Archetypes. A «ReferenceModel» Stereotype includes tag definitions for the publisher, namespace, and version of a Reference Model in a form compatible with a modeling language such as [ADL].

Diagrams

Figure 8.2 Reference Model Profile

Meta-classes

UML::Package

Attributes

- rmPublisher : UML::PrimitiveTypes::String [1]
  The value of this tag is the name of the Reference Model publisher. This tag definition maps to the [AOM] ARCHETYPE_HRID/rm_publisher attribute.
- RmNamespace : UML::PrimitiveTypes::String [0..1]
  The value of this tag is the reverse domain name of the namespace (for example, uk.gov.nhs). This tag definition maps to the [AOM] ARCHETYPE_HRID/namespace attribute.
- RmVersion : UML::PrimitiveTypes::String [0..1]
  The value of this tag is the version id of the reference model on which the archetype is based. The tag maps to the [AOM] ARCHETYPE/rm_release attribute.

Constraints

- **[AOM] ARCHETYPE_HRID:Invariant:Rm_publisher_validity**

The [AOM] ARCHETYPE_HRID/rm_publisher must have a value. This [AOM] Invariant maps to the AML Constraint that the «ReferenceModel» rmPublisher is required.

[English]
```
The requirement that there is a specified rmPublisher is enforced by the UML Semantic for
the required tag rmPublisher.
```

8.3.4 Runtime [Stereotype]

Description

A «Runtime» Property models a dynamic or "runtime" element such as a time stamp. A Property with an applied Runtime Stereotype cannot be constrained using AML.

Diagrams

Figure 8.2 Reference Model Profile

Meta-classes

UML::Property

8.4 TerminologyProfile [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.

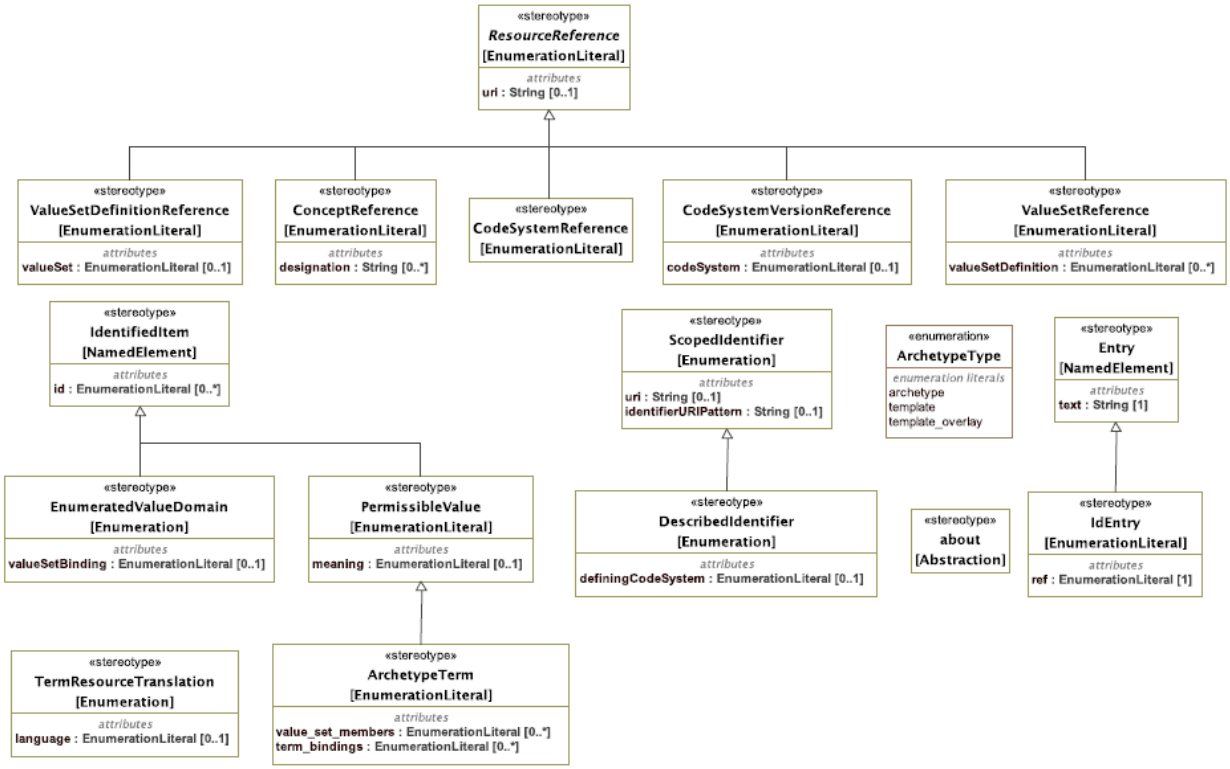

Figure 8.3 - Terminology Binding Profile

8.4.1 ArchetypeType [Enumeration]

Description

The «ArchetypeType» Enumeration specifies the structural type of an Archetype. The ArchetypeType is mapped to the structural variants described by [AOM] ARCHETYPE/is_template and ARCHETYPE/is_overlay attributes.

Diagrams

Figure 8.3 Terminology Binding Profile

Literals

- **archtype**

This literal specifies that the Archetype is structured as a source Archetype.

Source archetypes can be specialized, 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. «ArchetypeRoot» instances may only be used to represent external references to other Archetypes.

An “archetype” maps to an [AOM] ARCHETYPE in which both ARCHETYPE/is_template and ARCHETYPE/is_overlay are false.

- **template**

This literal specifies that the Archetype is structured as a Template.

A source template is an Archetype containing «ArchetypeRoot» elements representing slot fillers - each referring to an external Archetype or template, or potentially an overlay archetype.

An Archetype template corresponds to an [AOM] ARCHETYPE/is_template attribute having value=True.

- **template_overlay**

This literal specifies that 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 specialized overlay on something else, and may not contain any slot fillers or external references, i.e., no «ArchetypeRoot» 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 specialized archetype. Template overlays can be thought of as being similar to ‘anonymous’ or ‘inner’ classes in some object-oriented programming languages.

A template_overlay is mapped to an [AOM] ARCHETYPE with both ARCHETYPE/is_template and ARCHETYPE.is_overlay set to true.

8.4.2 about [Stereotype]

Description

The «about» Abstraction specifies an ontological meaning for an AML model element. The about stereotype is used to model the ISO 11179-3 “meaning” association between a Data Element and a Data Element Concept or between a Value Domain and a Conceptual Domain.

Diagrams

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::Abstraction

Constraints

- **isConceptReference**

The supplier (target) of an «about» Abstraction must be a «ConceptReference».

[OCL]
```
self.base_Abstraction.supplier->
select(c|c.stereotypedBy('ConceptReference'))->size()=1
```

8.4.3 ArchetypeTerm [Stereotype]

Description

An «ArchetypeTerm» EnumerationLiteral is used to model a definition identifier. A definition identifier is used to isolate an Archetype model element from its definition in a specific natural language, technology binding, or value set

grouping. An «IdentifiedItem» element within an Archetype specifies an «ArchetypeTerm» as the language-neutral identification of a term. Language-specific terminology names and definitions are specified by «IdEntry» EnumerationLiterals, each of which has an id whose value is an «ArchetypeTerm» EnumerationLiteral. The natural language associated with an «IdEntry» EnumerationLiteral is in the containing «ResourceTranslation» Enumeration, which defines all the meta-data associated with a given natural language.

An «ArchetypeTerm» EnumerationLiteral may be used to represent the [AOM] concepts of "id," "ac," or "at" codes, all of which have natural language translations. An "at" code represents value term codes within a terminology and may be used as possible values on terminological constraints. An "ac" code represents a value set, which is a set of "at" codes. The "id" codes are used to provide identification for other element nodes within an Archetype.

The term_binding tag within an «ArchetypeTerm» is used to specify technology bindings for "at" codes. Each "at" code may be bound to many «ResourceReference» EnumerationLiterals, each of which may specify a URI as its technology-specific identifier.

The value_set_members tag within an «ArchetypeTerm» is used to specify value set membership for an "ac" code. The members of the value set are sibling «ArchetypeTerm»s, each of which represent an "at" code.

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::PermissibleValue

8.4.12 Permissible Value [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- value_set_members : UML::EnumerationLiteral [0..*]
  When the «ArchetypeTerm» corresponds to an [AOM] "ac" code, the value_set_members tag is used to identify the sibling «ArchetypeTerm» "at" codes logically contained in the "ac" value set. All of the "at" codes must be in the same Enumeration as the "ac" code.

  This tag encapsulates the [AOM] concept of VALUE_SET.

- term_bindings : UML::EnumerationLiteral [0..*]
  When the «ArchetypeTerm» corresponds to an [AOM] "at" code, the term_bindings tag is used to identify the list of «ResourceReference» EnumerationLiterals to be associated with the "at" code. A term binding may be used to specify the "meaning" of the «ArchetypeTerm», either in the ISO 11179 sense or as a reference to a terminology definition provided by an external service. In either case, each term_binding member is a «ConceptReference» within an Enumeration representing the [AOM] concept of term_binding.

Constraints

- **[AOM] ARCHETYPE_TERM:Invariant:code_valid_code**

The name of the base_EnumerationLiteral must be defined.

[OCL]
```
not(self.base_EnumerationLiteral.name.oclIsUndefined())
and(self.base_EnumerationLiteral.name<>'')
```

- **[AOM] ARCHETYPE_TERMINOLOGY:Invariant:term_bindings_validity**

A terminology binding is specified by the values of the term_bindings tag. Each terminology binding must be to a Concept Reference within the same Archetype.

[OCL]
```
self.term_bindings ->
forAll(binding| binding.stereotypedBy('ConceptReference') and
(binding.namespace.namespace.namespace=self.base_EnumerationLiteral.namespace.
namespace.namespace) )
```

- **[AOM] VALUE_SET:Invariant:Id_valid**

An [AOM] VALUE_SET maps to an AML «ArchetypeTerm» which has values in the value_set_members tag. The [AOM] VALUE_SET/id must be a valid value set code.

[English]
```
In AML, the [AOM] Constraint is definitional; the ArchetypeTerm name is an
Identifier Definition id as well as the Value Set definition.
```

- **[AOM] VALUE_SET:Invariant:Members_valid**

An [AOM] VALUE_SET maps to an AML «ArchetypeTerm» which has values in the value_set_members tag. Each member of value_set_members must be a sibling «ArchetypeTerm» within the same Enumeration.

[OCL]
```
self.value_set_members→
forAll(m|m.namespace=self.base_EnumerationLiteral.namespace)
```

- **[AOM] VETDF- external term validity**

Each external term used within the archetype definition must exist in the relevant terminology.

[OCL]
```
self.term_bindings->forAll(b|b.namespace.namespace.name='term_bindings')
```

- **[AOM] VTCBK- terminology constraint binding key valid**

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

[OCL]
```
self.term_bindings->forAll(b|b.namespace.namespace.name='term_bindings')
```

- **[AOM] VTLC- language consistency**

Every code must exist in all languages. In AML, every Archetype Term must be referenced by a definitional IdEntry in each of the language-specific Resource Translations.

[OCL]
```
self.base_EnumerationLiteral.namespace.oclAsType(Enumeration).
namespace.oclAsType(Package).ownedType →
select(t|t.stereotypedBy('ResourceTranslation')).oclAsType(Enumeration)
```

```
forAll(sibling|sibling.ownedLiteral→
exists(ol|ol.appliedStereotype('IdEntry').oclAsType(IdEntry).ref=
self.base_EnumerationLiteral))
```

- **[AOM] VTSD- specialisation level of codes**

Term or constraint code defined in archetype terminology must be of the same or a less specialized level than the specialization level of the Archetype.

[English]
```
Term or constraint code defined in archetype terminology must be of the same or a less
specialized level than the specialization level of the Archetype.
```

- **[AOM] VTTBK- terminology term binding key valid**

Every term binding must be to a defined archetype term (‘at-code’).

[OCL]
```
self.term_bindings->forAll(b|b.namespace.namespace.name='term_bindings')
```

- **[AOM] 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]
```
This [AOM] Validity Rule is definitional in AML. The definition of the identifying code
of a value set is an Archetype Term which has a non-empty value for the value_set_members
tag.
```

- **[AOM] VTVSMD- value-set members defined**

The member codes of a value set must be defined in the term definitions of the current archetype.

Note that in AML, this is equivalent to requiring that value_set_members are in the same Enumeration as this ArchetypeTerm.

[OCL]
```
self.value_set_members→
forAll(member|member.namespace=self.base_EnumerationLiteral.namespace)
```

- **[AOM] VTVSUQ- value-set members unique**

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

The member codes of a value set must be from the same Enumeration as this ArchetypeTerm.

Note that the value_set_members tag definition is declared to be unique.

[OCL]
```
self.value_set_members→
forAll(member|member.namespace=self.base_EnumerationLiteral.namespace)
```

8.4.4 CodeSystemReference [Stereotype]

Description

«CodeSystemReference» is used to model a reference to a code system (aka. “Terminology,” “Classification scheme,” or “ontology”).

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ResourceReference

8.4.13 ResourceReference [Stereotype]

Meta-classes

UML::EnumerationLiteral

8.4.5 CodeSystemVersionReference [Stereotype]

Description

«CodeSystemVersionReference» is used to model a reference to a specific version of a code system and, if known, the code system which it is a version of. [CTS2]

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ResourceReference

8.4.13 ResourceReference [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- codeSystem : UML::EnumerationLiteral [0..1]
  The codeSystem tag is a reference to the code system. The codeSystem tag is used in situations where the code system itself has a well-known URI but the referenced version does not. In this case the version URI can be omitted and the reference used in its place.

8.4.6 ConceptReference [Stereotype]

Description

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

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ResourceReference

8.4.13 ResourceReference [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- designation : UML::PrimitiveTypes::String [0..*]
  The designation tag models 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')
```

8.4.7 DescribedIdentifier [Stereotype]

Description

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

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ScopedIdentifier

8.4.14 ScopedIdentifier [Stereotype]

Meta-classes

UML::Enumeration

Attributes

- definingCodeSystem : UML::EnumerationLiteral [0..1]
  The definingCodeSystem tag specifies the default Code System for the owned ConceptReferences.

Constraints

- **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') )
```

- **membersMustBeConceptReference**

All of the member EnumerationLiterals must be stereotyped by ConceptReference.

[OCL]
```
self.base_Enumeration.ownedLiteral→
forAll(ol|ol.stereotypedBy('ConceptReference'))
```

8.4.8 Entry [Stereotype]

Description

An «Entry» EnumerationLiteral models a language specific name (text) and optional description. The description of an Entry is provided in the ownedComment.body of the EnumeraitonLiteral. The name (text) of the Entry is defined in the required text tag.

Diagrams

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::NamedElement

Direct Subclasses (Specialization)

TerminologyProfile::IdEntry
8.4.11 IdEntry [Stereotype]

Attributes

- text : UML::PrimitiveTypes::String [1]
  The text tag models the language specific name of a terminology definition.

Note that the same name may be used more than once in a language-specific terminology definition (for example, “cold” may be a medical condition or a temperature). Since UML does not allow multiple EnumerationLiterals to have the same name, the terminology definition name is placed in this tag rather than as the name of the EnumerationLiteral.

Constraints

- **[AOM] ARCHETYPE_TERM:Invariant:text_valid**

The text tag must have a value.

[English]
```
The [AOM] Invariant is enforced by the UML semantic for the required text tag.
```

8.4.9 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 via the valueSetBinding tag. Implementations may use the tag value to validate the PermissibleValue meaning links, populate the permissible values in the EnumeratedValueDomain, or provide selection lists for existing mappings.

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::IdentifiedItem
8.4.10 IdentifiedItem [Stereotype]

Meta-classes

UML::Enumeration

Attributes

- valueSetBinding : UML::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

- **bindingIsValueSetOrDefinition**

The valueSetBinding tag value, if present, must reference an EnumerationLiteral that is stereotyped by ValueSetReference or ValueSetDefinitionReference.

[OCL]
```
not(self.valueSetBinding.oclIsUndefined()) implies
( self.valueSetBinding.stereotypedBy('ValueSetReference') or
self.valueSetBinding.stereotypedBy('ValueSetDefinitionReference') )
```

- **permissibleValues**

All ownedLiterals must be stereotyped by PermissibleValue.

[OCL]
```
self.base_Enumeration.ownedLiteral->forAll(x:EnumerationLiteral|
x.stereotypedBy('PermissibleValue')
```

8.4.10 IdentifiedItem [Stereotype]

Description

An «IdentifiedItem» is a NamedElement which may reference definition identifiers. A definition identifier («ArchetypeTerm») enables natural language terminology definitions, technology bindings, and value sets to be specified independently from the Archetype definition.

Diagrams

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::NamedElement

Direct Subclasses (Specialization)

TerminologyProfile::EnumeratedValueDomain

8.4.9 EnumeratedValueDomain [Stereotype]

ConstraintProfile::ObjectConstraint

8.5.12 ObjectConstraint [Stereotype]

TerminologyProfile::PermissibleValue

8.4.12 PermissibleValue [Stereotype]

Attributes

- id : UML::EnumerationLiteral [0..*]
  The id tag references one or more «ArchetypeTerm» EnumerationLiterals as the definition identifiers for the Archetype element. The referenced Archetype Terms enable multiple language terminology definitions, technology bindings, and value set composition.

Constraints

- **uniqueScopes**

Every id must belong to a unique Enumeration. An identified Item cannot have two or more identifiers drawn from the same Enumeration.

[OCL]
```
self.id->forAll(l1 | self.id→forAll(l2 |
l1.oclAsType(EnumerationLiteral).namespace =
l2.oclAsType(EnumerationLiteral).namespace implies l1 = l2))
```

8.4.11 IdEntry [Stereotype]

Description

An «IdEntry» EnumerationLiteral models a language-specific term / description representation for the definition identifier («ArchetypeTerm») specified in the ref tag. An «IdEntry» is contained by a «ResourceTranslation», which provides meta-data about the language translation used in the context of a particular Archetype.

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::Entry

8.4.8 Entry [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- ref : UML::EnumerationLiteral [1]
  The ref tag references an «ArchetypeTerm», which is the language-independent definition identifier for an Archetype. The referenced «ArchetypeTerm» serves as a binding from the Archetype constraint model to multiple terminology definition languages.

Constraints

- **[AOM] ARCHETYPE_TERM:Invariant:description_valid**

Every term definition must have a description.

[OCL]
```
self.base_EnumerationLiteral.ownedComment._'body'→
exists(b|not(b.oclIsUndefined()))
```

8.4.12 PermissibleValue [Stereotype]

Description

A «PermissibleValue» EnumerationLiteral models 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

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::IdentifiedItem

8.4.10 IdentifiedItem [Stereotype]

Meta-classes

UML::EnumerationLiteral

Direct Subclasses (Specialization)

TerminologyProfile::ArchetypeTerm

8.4.3 ArchetypeTerm [Stereotype]

Attributes

- meaning : UML::EnumerationLiteral [0..1]
  The ConceptReference that provides the meaning for the permissible value

Constraints

- **mustBeConceptReference**

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

[OCL]
```
not(self.meaning.oclIsUndefined()) implies
self.meaning.stereotypedBy('ConceptReference')
```

8.4.13 ResourceReference [Stereotype]

Description

ResourceReference couples a local identifier with an optional URI which references the target resource. ResourceReference models the [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

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::EnumerationLiteral

Direct Subclasses (Specialization)

TerminologyProfile::CodeSystemReference

8.4.4 CodeSystemReference [Stereotype]

TerminologyProfile::CodeSystemVersionReference

8.4.5 CodeSystemVersionReference [Stereotype]

TerminologyProfile::ConceptReference

8.4.6 ConceptReference [Stereotype]

TerminologyProfile::ValueSetDefinitionReference

8.4.16 ValueSetDefinitionReference [Stereotype]

TerminologyProfile::ValueSetReference

8.4.17 ValueSetReference [Stereotype]

Attributes

- uri : UML::PrimitiveTypes::String [0..1]
  The uri tag specifies the URI of the referenced resource. This tag definition is similar to the [CTS2] specification which defines the concept as "A globally unique URI that identifies the intended meaning of the identifier."

8.4.14 ScopedIdentifier [Stereotype]

Description

The ScopedIdentifier stereotype models both the ISO 11179-3 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 URIs 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

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::Enumeration

Direct Subclasses (Specialization)

TerminologyProfile::DescribedIdentifier

8.4.7 DescribedIdentifier [Stereotype]

Attributes

- uri : UML::PrimitiveTypes::String [0..1]
  The uri tag specifies 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::PrimitiveTypes::String [0..1]
  The identifierURIPattern tag models 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.

8.4.15 TermResourceTranslation [Stereotype]

Description

TermResourceTranslation is a collection of designations and descriptions/definitions in a target language. TermResourceTranslation represents a refactoring of the ISO 11179-3 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

Figure 8.3 Terminology Binding Profile

Meta-classes

UML::Enumeration

Attributes

- language : UML::EnumerationLiteral [0..1]
  The language tag identifies the target language.

Constraints

- **translationEntries**

All of the ownedLiterals must be stereotyped by Entry.

[OCL]

```
self.base_Enumeration.ownedLiteral->forAll(ol|ol.stereotypedBy('Entry'))
```

8.4.16 ValueSetDefinitionReference [Stereotype]

Description

A «ValueSetDefinitionReference» EnumerationLiteral represents the [CTS2] concept of a reference to a set of rules for constructing a value set along with the corresponding value set if known.

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ResourceReference

8.4.13 ResourceReference [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- valueSet : UML::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 URIs 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.

8.4.17 ValueSetReference [Stereotype]

Description

ValueSetReference models the [CTS2] concept of "A reference to a named set of entity references." ValueSetReference references a set of ValueSetDefinitionReferences. 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 and (b) the particular version of the code system(s) that are used to resolve the rules in the value set definition.

Diagrams

Figure 8.3 Terminology Binding Profile

Direct Superclasses (Generalization)

TerminologyProfile::ResourceReference

8.4.13 ResourceReference [Stereotype]

Meta-classes

UML::EnumerationLiteral

Attributes

- valueSetDefinition : UML::EnumerationLiteral [0..*]
  The value of this tag is a set of references to value set definition references. Each element of this set must be a «ValueSetDefinitionReference».

Constraints

- **definition**

Each member of valueSetDefinition must be a «ValueSetDefinitionReference» EnumerationLiteral.

[OCL]

```
self.valueSetDefinition→
forAll(d|d.stereotypedBy('ValueSetDefinitionReference'))
```

8.5 ConstraintProfile [Profile]

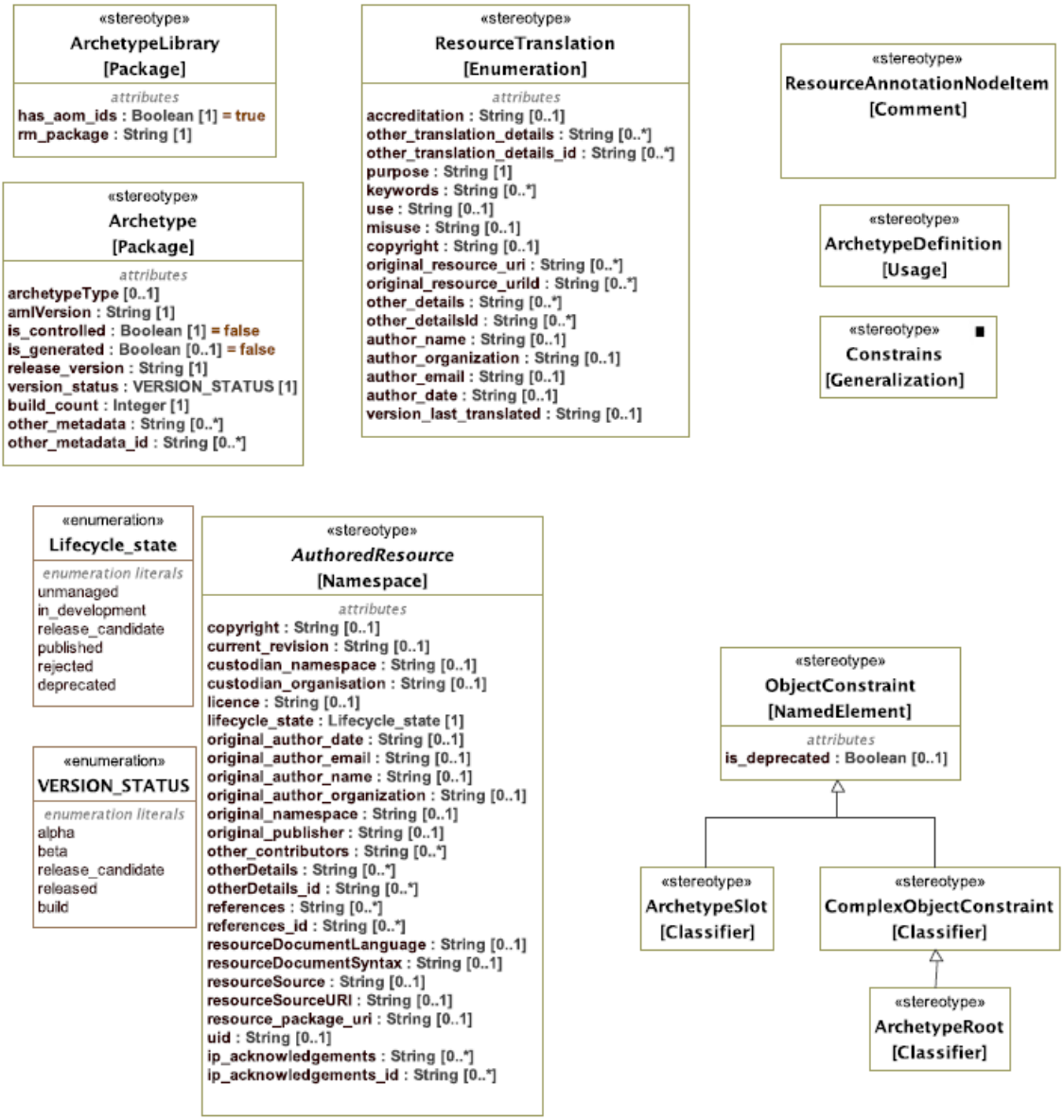

Figure 8.4 - Constraint Profile

There is a need for information interoperability between health entities. Information needs to be shared between organizations and across international boundaries. The inability to share this information in a repeatable manner greatly affects the quality of care provided. The Clinical Information Modeling Initiative (CIMI)[CIMI] has the potential to be a disruptive innovation in eHealth. The AML specification for the representation of health information content, semantically interoperable information may be created and shared in health records, messages, and documents. The CIMI initiative affords the opportunity to enable the storage of lifelong health information; simplify data exchange, aggregation, querying and analysis; and support knowledge-based activities such as decision support. This will be

achieved through the development of non-proprietary, common and fully defined information models of clinical content and known transformations.

The clinical reference model (an instance of a reference model), clinical archetype models and associated terminology will serve as the domain vocabulary for clinical information. The syntax and semantics defined by these clinical archetype models shall be maintained by users, in a common language that consistently can be understood and shared. The AML Profile enables the creation, definition, and use of this common language in UML.

The purpose of the AML Profile is to enable a UML ecosystem that supports and underpins CIMI activities through the use of adopted standards. The AML Profile provides a clear, consistent means of designing clinical models using UML, where tool vendors may add additional value/usability; clinical modeling concepts are separated from specific solutions (ex. XML, JSON, DB schema, etc.); and the creation of open source solutions is enabled.

The AMP Profile:

- Specifies a collection of complementary UML profiles that work together to support the creation of CIMI content models.
- Supports the specification of CIMI content models in UML, such that they can be translated into AOM 2.0.
- Is capable of being used in other domain areas, with other reference models.
- Is capable of being used in developing specific implementations of CIMI content models using platform specific solutions (e.g., Clinical Document Architecture (CDA), openEHR, etc.).

The AML Profile provides consistency by ensuring that a UML representation of a CIMI model produced by one developer can be accurately interpreted by developers, modelers, and transformations. It offers completeness by ensuring that a developer can produce a UML representation of any CIMI reference and constraint model. Finally, the AML Profile offers practicality by ensuring that a developer/modeler can develop a CIMI compliant clinical model by employing the profile in current UML modeling tools.

Within the healthcare community the pattern of creating a common model that is reused by others to create specialized models through constraining the original model is often referred to as reference / constraint modeling. The reference model consists of syntax-neutral and technology- independent building blocks that can be used for data modeling. Major benefits of this approach include improved reuse of existing data artifacts and improved enterprise interoperability.

The AML profile provides a family of UML sub-profiles that enables the representation of semantic-based information models and addresses the problem of the lack of semantic interoperability within and between applications and databases in healthcare computing environments, including across enterprises and national borders.

The AML Profile is the aggregation of three sub-profiles:

1. The Reference Model Profile (RMP)
2. The Constraint Model Profile (CMP)
3. The Terminology Binding Profile (TBP)

Traditionally, a single model containing all the required information concepts is designed for a specific application, transport and database without regard to interoperability. Standards for the exchange of that health data between applications and databases have been focused on static message definitions that have not enabled a sufficient degree of interoperability or flexibility. They have not enabled ‘single-source’ modeling, whereby a single definition (e.g., a microbiology lab result) can be re-used for multiple purposes, such as a message definition, a document definition, a screen display form, a screen data capture form, or a report.

A more flexible and interoperable way of standardizing business semantics has long been required.

This specification provides a means for developing a common set of semantic building blocks that represent the general types of healthcare data in use today. The solution should provide an approach for the creation of new healthcare

information models and the semantic binding of these information models to published terminologies to achieve semantic interoperability of data. It can be employed wherever health data is being defined, stored, used, shared, or exchanged.

8.5.1 ArchetypeType [Enumeration]

Description

The «ArchetypeType» Enumeration specifies the structural type of an Archetype. The ArchetypeType is mapped to the structural variants described by [AOM] ARCHETYPE/is_template and ARCHETYPE/is_overlay attributes.

Diagrams

Figure 8.3 Terminology Binding Profile

Literals

- **archetype**

This literal specifies that the Archetype is structured as a source Archetype.

Source archetypes can be specialized, 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. «ArchetypeRoot» instances may only be used to represent external references to other Archetypes.

An "archetype" maps to an [AOM] ARCHETYPE in which both ARCHETYPE/is_template and ARCHETYPE/is_overlay are false.

- **template**

This literal specifies that the Archetype is structured as a Template.

A source template is an Archetype containing «ArchetypeRoot» elements representing slot fillers - each referring to an external Archetype or template, or potentially an overlay archetype.

An Archetype template corresponds to an [AOM] ARCHETYPE/is_template attribute having value=True.

- **template_overlay**

This literal specifies that 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 specialized overlay on something else, and may not contain any slot fillers or external references, i.e., no «ArchetypeRoot» 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 specialized archetype. Template overlays can be thought of as being similar to 'anonymous' or 'inner' classes in some object-oriented programming languages.

A template_overlay is mapped to an [AOM] ARCHETYPE with both ARCHETYPE/is_template and ARCHETYPE.is_overlay set to true.

8.5.2 Lifecycle_state [Enumeration]

Description

The Lifecycle_state Enumeration is used to specify the state of an Archetype within its defined lifecycle. The lifecycle state machine and versioning rules are explained fully in the openEHR Knowledge Artefact Identification specification [KIS].

Diagrams

Figure 8.4 Constraint Profile

Literals

- **deprecated**
  A code indicating that the artifact that is obsolete, suspended, or withdrawn from active use.
- **in_development**
  A code indicating that an artifact is actively being created or modified.
- **published**
  A code indicating that an artifact is available and is in active use.
- **rejected**
  A code indicating that an artifact is withdrawn from development prior to being published.
- **release_candidate**
  A code indicating that an artifact is being considered for publication.
- **unmanaged**
  A code indicating that an artifact has no recognized owner organization.

8.5.3 VERSION_STATUS [Enumeration]

Description

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

Diagrams

Figure 8.4 Constraint Profile

Literals

- **alpha**
  A code indicating a version which is ‘unstable’ i.e., that the version 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**
  A code indicating a version which is ‘beta’ i.e., that the version 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**
  A code indicating a version is 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**
  A code indicating 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**
  A code indicating 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").

8.5.4 Archetype [Stereotype]

Description

An «Archetype» is a topic- or theme-based model of domain content, expressed in terms of constraints on a reference information model. Since each «Archetype» constitutes an encapsulation of a set of data points pertaining to a topic, it is of a manageable, limited size, and has a clear boundary. An «Archetype» includes descriptive meta-data, language information, annotations, revision history, the Archetype definition, rules, and terminology.

The Archetype definition for an Archetype is modeled using exactly one «ArchetypeDefinition» Usage. The client of the Usage is the «Archetype» while the supplier of the Usage is the top level «ComplexObjectConstraint» Classifier constituting the definition of the «Archetype». The overall structure of an Archetype definition consists of the top level Classifier, which owns attributes that have compositions of Classifiers, which owns attributes, etc. All of the Classifiers in the Archetype definition are owned by the «Archetype» Package.

«Archetype» Rules are represented in AML as Constraints owned by the «Archetype» and constraining the Archetype definition.

Archetype terminology is modeled in a nested Package with the name "ontology." Within the "ontology" Package, there are nested packages representing the terminology concepts in [AOM]:

- Terminology definitions - The Terminology definitions package contains a set of «ResourceTranslation»s, one for each natural language translation of the definition identifiers referenced in the Archetype definition structure. Additionally, there is an Identifier Definition Enumeration that serves to bind elements from the Archetype definition to the various natural language translations, the technology bindings, and value set compositions. The name of the Terminology Package is the terminology identifier, a name such as ISO_639-1.
- term_bindings - The term_bindings package contains technology bindings for codes defined in the Identifier Definition. Each technology binding is modeled as an Enumeration.
- terminology_extracts - The terminology_extracts package contains extracts from external terminologies such as SNOMED CT. Each extract is modeled as an Enumeration consisting of codes and preferred term rubrics, enabling small value sets to be captured locally in the model.

In [AOM], the name of an ARCHETYPE is specified in the attribute ARCHETYPE_HRID/physical_id, which is expressed using the following syntax:

[rm_publisher]-[rm_closure]-[rm_class].[concept_id].v[release_version]-[version_status].[build_count]

In AML, the physical_id is implicit, being derived from the following model elements:

- rm_publisher - This is the value of the «ReferenceModel»::rmPublisher tag, where «ReferenceModel» is imported by the containing «ArchetypeLibrary».
- rm_closure - This is the value of the «ArchetypeLibrary»::rm_package tag.
- rm_class - This is derived from the root (defining) class of this «Archetype». The value is the name of the «ReferenceModel» Class specialized by the Archetype definition of this «Archetype».
- concept_id - The value is the name of the «Archetype» Package.
- release_version - The value is the full numeric version of this «Archetype», as specified in the release_version tag of this «Archetype».

- version_status - The value is the status of the version, as specified in the version_status tag of this «Archetype».
- build_count - The value is the build_count, as specified in the build_count tag of this «Archetype».

The URI of the «Archetype» Package represents the [AOM] attribute ARCHETYPE/namespace.

«Archetype» specialization may be modeled using a PackageImport from some parent «Archetype».

An «Archetype» Package contains (directly or indirectly) a set of constraints on «ReferenceModel» Classifiers. The constrained Classifiers must be owned by the «ReferenceModel» imported from the nesting Package «ArchetypeLibrary».

Diagrams

Figure 8.4 Constraint Profile

Direct Superclasses (Generalization)

ConstraintProfile::AuthoredResource

8.5.9 AuthoredResource [Stereotype]

Meta-classes

UML::Package

Attributes

- archetypeType : [0..1]
  The archetypeType tag models the kind of the archetype. The archetypeType value may be one of archetype, template, or template_overlay.
- amlVersion : UML::PrimitiveTypes::String [1]
  The version of the AML specification used to define this version of the Archetype.
- is_controlled : UML::PrimitiveTypes::Boolean [1]
  The is_controlled tag indicates whether the archetype is change-controlled or not. If is_controlled is true, the Archetype should have a revision history section included, otherwise an Archetype may omit the revision history. This indicator enables Archetypes to be privately edited in an early development phase without generating large revision histories of little or no value.

  If this indicator is false, the Archetype is an ad hoc, uncontrolled artifact, not formally associated with any organization. This case is typical for an experimental archetype. If true, then the archetype must include a namespace which denotes the original authoring organization.
- is_generated : UML::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::PrimitiveTypes::String [1]
  The release_version tag specifies the full numeric version of this Archetype. In AOM, this identifier will consist of 3 parts: the major, the minor, and the patch version.
- version_status : ConstraintProfile::VERSION_STATUS [1]
  The version_status tag specifies the status of the version.
- build_count : UML::PrimitiveTypes::Integer [1]
  The build_count tag specifies the number of builds since the last increment of any version part.
- other_metadata : UML::PrimitiveTypes::String [0..*]
  The other_metadata tag contains additional information about an Archetype.

- other_metadata_id : UML::PrimitiveTypes::String [0..*]
  The other_metadata_id tag contains the name associated with additional information about an Archetype. other_metadata_id Strings are matched to other_metadata Strings by order.

Constraints

- **mustBeOwned**

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

[OCL]
```
self.base_Package.owningPackage.stereotypedBy('ArchetypeLibrary')
```

- **ownsObjectConstraints**

Types (other than Associations) 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'))
```

- **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))
```

- **[AOM] ARCHETYPE:Invariant:Concept_valid**

The [AOM] invariant requires the concept_code to exist in the terminology definition. In AML, the concept_code is the language-specific text for the top-level Archetype definition Classifier. Thus in AML, this [AOM] invariant maps to the requirement that the Archetype definition Classifier must have an «Archetype» id tag whose value is an «ArchetypeTerm» within an Identifier Definition. Note that this Constraint ends up the same as [AOM] ARCHETYPE_TERMINOLOGY:Invariant:concept_code_validity.

[OCL]
```
self.base_Package.clientDependency  →
select(d|d.stereotypedBy('ArchetypeDefinition')).supplier  →
exists(s|s.appliedStereotypeInstance.oclAsType(ComplexObjectConstraint).id →
notEmpty())
```

- **[AOM] ARCHETYPE:Invariant:Definition_exists**

Every Archetype Package must have exactly one clientDependency that is an ArchetypeDefinition and whose supplier is a ComplexObjectConstraint within the same Archetype Package.

[OCL]
```
self.base_Package.clientDependency    →
exists(d|d.stereotypedBy('ArchetypeDefinition') )
```

- **[AOM] ARCHETYPE:Invariant:Original_language_valid**

The Archetype must have a valid original language. In AML, the original language is identified as a Usage from the Archetype to a Resource Translation.

[OCL]
```
self.base_Package.clientDependency → select(d|d.oclIsKindOf(Usage) and
(d.name='original_language')).supplier →
select(e|e.stereotypedBy('ResourceTranslation')) ->size()=1
```

[AOM] ARCHETYPE:Invariant:Rules_valid

- **[AOM] ARCHETYPE:Invariant:Rules_valid**

The [AOM] ARCHETYPE/rules, if present, must not be an empty list. [AOM] ARCHETYPE/rules maps to AML «Archetype» ownedRule Constraints. An [AOM] ARCHETYPE/rules that has no value is mapped to an empty list of AML «Archetype» ownedRule Constraints.

[English]
```
This [AOM] Invariant is definitional in AML. In AML, if rules is empty, it implies rules
are undefined.
```

- **[AOM] ARCHETYPE:Invariant:Specialisation_validity**

The [AOM] ARCHETYPE/is_specialised value is true if this archetype is a specialization of another. The [AOM] ARCHETYPE/specialisation_depth is larger than 0 if this archetype has a parent. In AML, both is_specialised and specialisation_depth are derived, if required, during mapping.

[English]
```
During AML transformations, is_specialised is derived and will be consistent with
specialisation_depth, which is also derived.
```

[AOM] ARCHETYPE:Invariant:Terminology_exists

- **[AOM] ARCHETYPE:Invariant:Terminology_exists**

A terminology definition must exist for an Archetype. This [AOM] Invariant maps to the AML Constraint that an Archetype must have one or more Resource Transformations.

[OCL]
```
self.base_Package.nestedPackage.nestedPackage.ownedType→
exists(o|o.stereotypedBy('ResourceTranslation'))
```

- **[AOM] ARCHETYPE_HRID:Invariant:Base_version_validity**

The tag release_version MUST have a value.

[English]
```
The release_version tag is required, so this [AOM] invariant is enforced by [UML]
semantics.
```

- **[AOM] ARCHETYPE_HRID:Invariant:Concept_id_validity**

The [AOM] ARCHETYPE_HRIN/concept_id must have a value. [AOM] ARCHETYPE_HRIN/concept_id maps to the name of the «Archetype» Package.

[OCL]
```
not(self.base_Package.name.oclIsUndefined()) and (self.base_Package.name<>'')
```

- **[AOM] ARCHETYPE_TERMINOLOGY:Invariant:concept_code_validity**

The [AOM] ARCHETYPE/concept_code must be represented in the terminology definition. The [AOM] ARCHETYPE/concept_code is mapped to the AML language-specific text for the top-level Archetype definition Classifier. Thus in AML, this [AOM] invariant maps to the requirement that the Archetype definition Classifier must have an «IdentifiedItem» id tag whose value is a «ArchetypeTerm» within an Identifier Definition.

[OCL]
```
self.base_Package.clientDependency →
select(d|d.stereotypedBy('ArchetypeDefinition')).supplier→
exists(s|s.appliedStereotypeInstance.oclAsType(ComplexObjectConstraint).id→
notEmpty())
```

- **[AOM] ARCHETYPE_TERMINOLOGY:Invariant:original_language_validity**

There must exist an original language for the [AOM] ARCHETYPE_TERMINOLOGY. This [AOM] invariant maps to the requirement that an AML Archetype Package must have exactly 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 →
select(e|e.stereotypedBy('ResourceTranslation')) → size()=1
```

- **[AOM] ARCHETYPE_TERMINOLOGY:Invariant:Parent_archetype_valid**

The [AOM] ARCHETYPE/terminology is required. The [AOM] ARCHETYPE/terminology maps to a nested Package of the «Archetype» Package named “ontology.”

[OCL]
```
self.base_Package.nestedPackage->exists(p|p.name='ontology')
```

- **[AOM] ARCHETYPE_TERMINOLOGY:Invariant:term_definitions_validity**

The [AOM] ARCHETYPE_TERMINOLOGY/term_definitions maps to a set of AML «ResourceTranslation»s, one per language. There must be one or more «ResourceTranslation»s for an Archetype.

[OCL]
```
self.base_Package.clientDependency->select(d|d.oclIsKindOf(Usage) and
(d.name='original_language')).supplier →
exists(e|e.stereotypedBy('ResourceTranslation'))
```

- **[AOM] OPERATIONAL_TEMPLATE:Invariant:Component_terminologies_existence**

An [AOM] OPERATIONAL_TEMPLATE maps to an AML Archetype with value of tag archetypeType=ArchetypeType::template. Within the implicit context of an OPERATIONAL_TEMPLATE, there is a logical compendium of flattened terminologies from externally referenced archetypes.

[English]
```
The compendium of terminologies from externally archetypes is derivable from the AML
model. Although not explicitly defined in AML, it is conceptually derivable from an
Archetype. The derived value would contain a list of the flattened terminologies from the
externally referenced archetypes.
```

- **[AOM] OPERATIONAL_TEMPLATE:Invariant:Is_specialised**

An [AOM] OPERATIONAL_TEMPLATE maps to an AML Archetype with value of tag archetypeType =ArchetypeType::template.

[English]
```
This [AOM] invariant is definitional in AML.
```

- **[AOM] VACSD- archetype concept specialisation depth**

The specialization depth of the concept code must be one greater than the specialization 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.
```

- **[AOM] VALC- archetype language conformance**

The languages defined in a specialized 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))    )
```

- **[AOM] VARD- description specified**

For an ArchetypeType of 'archetype' or 'template,' 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()
```

- **[AOM] VARDT- archetype definition typename validity**

The [AOM] C_OBJECT/rm_type_name mentioned in the outer block of the archetype definition section must match the type mentioned in the [AOM] ARCHETYPE_HRID/rm_class. The [AOM] C_OBJECT/rm_type_name maps to the name of the AML Reference Model Class specialized by the archetype definition. In other words, the [AOM] rm_type_name and rm_class map to a single element in AML.

[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, so the [AOM] validity rule
is always satisfied.
```

- **[AOM] VARIABLE_DECLARATION:Invariant:Name_valid**

An [AOM] VARIABLE_DECLARATION/name maps to a UML InstanceSpecification/name owned by an «Archetype» Package. The name must be defined and not empty.

[OCL]
```
self.namespace.stereotypedBy('Namespace')implies(
not( self.name.oclIsUndefined()) and(self.name<>'') )
```

- **[AOM] VARID- archetype identifier validity**

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]*)?)?')
```

- **[AOM] VASID- archetype specialisation parent identifier validity**

The [AOM] attribute AUTHORED_RESOURCE/parent_archetype_id must be the identifier of the immediate specialization parent archetype.

The [AOM] attribute AUTHORED_RESOURCE/parent_archetype_id maps to an AML packageImport from the «Archetype» Package to the parent «Archetype» Package. There may be at most one imported parent «Archetype» Package.

[OCL]
```
self.base_Package.packageImport.importedPackage→
select(p|p.stereotypedBy('Archetype'))->size()<=1
```

- **[AOM] VATCD- archetype code specialization level validity**

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

[English]
```
The specialization level used for archetype terms and constraint codes are derived and
enforced during mapping to [AOM] to conform with the specialization level validity
constraint.
```

- **[AOM] VDEOL- original language specified**

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()
```

- **[AOM] VOKU- object key unique**

Within any keyed list in an archetype, including the description, 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.
```

8.5.5 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] model.

Diagrams

Figure 8.4 Constraint Profile

Meta-classes

UML::Usage

Constraints

- **[AOM] ARCHETYPE_HRID:Invariant:Rm_class_name_validity**

The [AOM] ARCHETYPE_HRID/Rm_class maps to the name of the Reference Model Classifier which is the general of the top level Archetype definition Classifier. The [AOM] invariant maps to the AML requirement that the top level Archetype definition Classifier must exist.

[OCL]

```
self.base_Usage.supplier->exists(s|s.stereotypedBy('ObjectConstraint') and
s.oclAsType(Classifier).general->notEmpty())
```

8.5.6 ArchetypeLibrary [Stereotype]

Description

An «ArchetypeLibrary» is a container of «Archetype»s. An archetype library imports exactly one «ReferenceModel» Package. The imported «ReferenceModel» provides metadata used to derive parts of Archetype identifiers.

All Archetypes within a library constrain Classifiers and Properties within the same Reference Model. The Reference Model also specifies how to bind Reference Model Classifiers to AML Primitive Types (namely the UML Primitive Types plus the XML Primitive Types).

Diagrams

Figure 8.4 Constraint Profile

Meta-classes

UML::Package

Attributes

- has_aom_ids : UML::PrimitiveTypes::Boolean [1]

  If true, then all Archetypes in this Archetype Library use the id tag in «IdentifiedItem» to specify an «ArchetypeTerm» for binding model elements to multiple natural languages, technology bindings, and value set compositions. The has_aom_ids must be true to enable round-trip engineering between [AOM] models and AML. In this mode, the names of «ArchetypeTerm» EnumerationLiterals will be based on the [AOM] naming conventions (they will be of the form “id*,” “ac*,” or “at*”).

  If false, a somewhat simplified modeling paradigm is available, and the id tag in «IdentifiedItem» is not used. Transformation to [AOM] is not supported in this mode. It is assumed that there is a single Natural Language, and that language has no metadata. The «Entry» Stereotype may be applied to a model element in order to specify a terminology name independent of the NamedElement name. The description for the term may be specified in an ownedComment body. In this case, code value constraints are modeled as «PermissibleValue» EnumerationLiterals. A (single) technology binding would be specified as the value (having type «ResourceReference») of the meaning tag on the «PermissibleValue». A set of «PermissibleValue»s would be owned by an «EnumeratedValueDomain» Enumeration. A value set may be modeled using the valueSetBinding tag on the «EnumeratedValueDomain» Enumeration.

- rm_package : UML::PrimitiveTypes::String [1]

  Name of the Reference Model Package in whose closure the rm_class is found (there can be more than one possibility in a reference model). This tag corresponds to the [AOM] model attribute ARCHETYPE_HRID/rm_closure. Since this does not necessarily correspond to a physical package in the UML Reference Model, the type of this tag is a String.

Constraints

- **oneReferenceModel**

There 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'))
```

- **[AOM] ARCHETYPE_HRID:Invariant:Rm_closure_validity**

The [AOM] ARCHETYPE_HRID:Rm_closure maps to the AML «ArchetypeLibrary» rm_package tag. The rm_package tag must have a value, which is enforced by the multiplicity of the tag being [1].

[English]
```
The [AOM] Invariant is enforced by the UML semantic for a required Property.
```

8.5.7 ArchetypeRoot [Stereotype]

Description

«ArchetypeRoot» is a specialization of «ComplexObjectConstraint » in which an external «Archetype» is being reused. The reuse reference to the external «Archetype» is modeled as a «Constrains» Generalization in which the specific Classifier is this «ArchetypeRoot» and the general is the top-level definition Classifier for the external «Archetype».

An «ArchetypeRoot» does not own any attributes.

Diagrams

Figure 8.4 Constraint Profile

Direct Superclasses (Generalization)

ConstraintProfile::ComplexObjectConstraint

8.5.10 ComplexObjectConstraint [Stereotype]

Meta-classes

UML::Classifier

Constraints

- **[AOM] C_ARCHETYPE_ROOT:Invariant:Archetype_ref_validity**

The «Archetype» Classifier must specialize an «ObjectConstraint» Classifier that is the root of an Archetype definition.

[OCL]
```
self.base_Classifier.general.supplierDependency.client→
exists(g|g.stereotypedBy('Archetype'))
```

- **[AOM] VARXAV-external reference node archetype reference validity**

If this Archetype root is a redefinition of another external node, then that external node must have as an ancestor the Archetype definition which is the parent of this Archetype.

[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())
```

- **[AOM] 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 mapping to [AOM] may coerce the value of C_ARCHETYPE_ROOT/node_id to a value in conformance with this [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.appliedStereotype('ObjectConstraint').oclAsType(
ObjectConstraint).id.oclAsType(EnumerationLiteral).name→asSequence()→
first()))
```

- **[AOM] 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.appliedStereotype
Instance.oclAsType(ObjectConstraint).id.oclAsType(EnumerationLiteral).name->
asSequence()->first()))
```

- **[AOM] 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.
```

- **[AOM] VARXS – external reference conforms to slot**

The archetype referenced must be in the same library as this archetype.

[OCL]
```
self.base_Classifier.general.namespace.namespace→
forAll(al|al=self.base_Classifier.namespace.namespace)
```

- **[AOM] VARXTV – external reference type validity**

This Archetype Root must conform to the type it references, which may be a Reference Model Classifier or an Archetype Classifier.

[English]
```
The [AOM] Validation Rule is enforced by UML semantics for inheritance, since the
«ArchetypeRoot» inherits from the «ArchetypeSlot» in the archetype parent archetype.
```

8.5.8 ArchetypeSlot [Stereotype]

Description

An «ArchetypeSlot» 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.

ArchetypeSlot 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 modeled in AML as a generic expression tree of unary prefix and binary infix operators.

In AML, the assertions are modeled using a Constraint whose specification is a UML Expression. The UML Expression construct enables representation of the generic [AOM] expression tree consisting of unary prefix and binary infix operators.

The top level Expression symbol for an Assertion will be 'include' or 'exclude' to reflect the kind of assertion to be made on the slot.

An ArchetypeSlot can be redefined by:

- one or more ArchetypeRoot nodes taken together, considered to define a 'filled' version of the slot.
- an ArchetypeSlot. If the UML attribute 'leaf' is set, the slot is closed to further filling either in further specializations or at runtime. If leaf is not set, the slot remains open.

The redefinition of an ArchetypeSlot is modeled as a Generalization in UML, in which the general is an ArchetypeSlot and the specific is either an ArchetypeRoot or another ArchetypeSlot.

Diagrams

Figure 8.4 Constraint Profile

Direct Superclasses (Generalization)

ConstraintProfile::ObjectConstraint

8.5.12 ObjectConstraint [Stereotype]

Meta-classes

UML::Classifier

Constraints

- **[AOM] ARCHETYPE_SLOT:Invariant:excludes_valid**

The [AOM] ARCHETYPE_SLOT/excludes is defined implies ARCHETYPE_SLOT/excludes is not empty.

The [AOM] ARCHETYPE_SLOT/excludes maps to AML as a list of ‘exclude’ Expressions.

[English]
```
The [AOM] Invariant is definitional for AML. An undefined [AOM] ARCHETYPE_SLOT/excludes
maps to an empty list of exclude Expressions.
```

- **[AOM] ARCHETYPE_SLOT:Invariant:includes_valid**

The [AOM] ARCHETYPE_SLOT/includes is defined implies ARCHETYPE_SLOT/includes is not empty.

The [AOM] ARCHETYPE_SLOT/includes maps to AML as a list of 'include' Expressions.

[English]
```
The [AOM] Invariant is definitional for AML. An undefined [AOM] ARCHETYPE_SLOT/includes
maps to an empty list of include Expressions.
```

- **[AOM] VDSEV-archetype slot 'exclude' constraint validity**

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='includes')→notEmpty() implies
select(r|r.oclAsType(Expression).symbol='excludes')->isEmpty()
```

- **[AOM] VDSIV – archetype slot 'include' constraint validity**

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())
```

- **[AOM] VDSSC – specialised archetype slot definition closed validity**

In the specialization 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)
```

- **[AOM] VDSSID- slot redefinition child node id**

A slot node in a specialized 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.appliedStereotype('ArchetypeRoot').
oclAsType(ArchetypeRoot).id →forAll(e|e =self.base_Classifier.general.
appliedStereotype('ArchetypeRoot').ocl AsType(ArchetypeRoot).id->
asSequence()->first()) )
```

- **[AOM] VDSSM-specialised archetype slot definition match validity**

The set of archetypes matched from a library of archetypes by a specialized archetype slot definition must be a proper subset of the set matched from the same library by the parent slot definition.

[English]
```
The set of archetypes matched from a library of archetypes by a specialized archetype
slot definition  must be a proper subset of the set matched from the same library by the
parent slot definition.
```

- **[AOM] VDSSP-specialised archetype slot definition parent validity**

The flat parent of the specialization of an archetype slot must 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))
```

8.5.9 AuthoredResource [Stereotype]

Description

An «AuthoredResource» provides descriptive details about a resource as well as metadata about governance and usage. An «AuthoredResource» represents a combination of the [AOM] AUTHORED_RESOURCE and RESOURCE_DESCRIPTION model elements.

Diagrams

Figure 8.4 Constraint Profile

Meta-classes

UML::Namespace

Direct Subclasses (Specialization)

ConstraintProfile::Archetype

8.5.4 Archetype [Stereotype]

Attributes

- copyright : UML::PrimitiveTypes::String [0..1]
  The copyright property records the copyright applying to the artefact, and is normally in the standard form '(c) name' or '(c) year name.' The special character © may also be used (UTF-8 0xC2A9).
- current_revision : UML::PrimitiveTypes::String [0..1]
  The current_revision tag specifies the most recent revision in revision_history if *is_controlled* is true else "(uncontrolled)."
- custodian_namespace : UML::PrimitiveTypes::String [0..1]
  The custodian_namespace tag specifies the formal namespace corresponding to the current custodian of the artefact. It enables the user of the artefact to determine the definitive maintainer and publisher.
- custodian_organisation : UML::PrimitiveTypes::String [0..1]
  The custodian_organisation tag specifies a human-readable organization identifier corresponding to the current custodian of the artefact. It enables the user of the artefact to determine the definitive maintainer and publisher.
- licence : UML::PrimitiveTypes::String [0..1]
  The licence (US: 'license') under which the artefact can be used. The recommended format is 'licence name <reliable URL to licence statement>'.
- lifecycle_state : ConstraintProfile::Lifecycle_state [1]
  The lifecycle_state property is used to record its state in a defined lifecycle. The lifecycle state machine and versioning roles are explained fully in the openEHR Knowledge Artefact Identification [KIS] specification.
- original_author_date : UML::PrimitiveTypes::String [0..1]
  Date of original authoring of the resource.
  Maps to [AOM] RESOURCE_DESCRIPTION/original_author where id="date."
- original_author_email : UML::PrimitiveTypes::String [0..1]
  The original_author_email tag specifies the original author's email address.
  Corresponds to [AOM] RESOURCE_DESCRIPTION/original_author where id="email."
- original_author_name : UML::PrimitiveTypes::String [0..1]
  The original_author_name tag specifies the name of original author.
  Maps to [AOM] RESOURCE_DESCRIPTION/original_author where id="name."
- original_author_organization : UML::PrimitiveTypes::String [0..1]
  The original_author_organization tag specifies the name of original author's organization.
  Maps to [AOM] RESOURCE_DESCRIPTION/original_author where id= "organization."
- original_namespace : UML::PrimitiveTypes::String [0..1]
  The original publishing organization namespace, i.e., the original publishing environment where the artefact was first imported or created. The original_namespace property is normally the same value as archetype_id.namespace, unless the artefact has been forked into its current custodian, in which case archetype_id.namespace and custodian_namespace will be the same.
- original_publisher : UML::PrimitiveTypes::String [0..1]
  The original publishing organization, i.e., the original publishing environment where the artefact was first imported or created.
- other_contributors : UML::PrimitiveTypes::String [0..*]
  The other_contributors property is a simple list of strings, one for each contributor. The recommended format of the string is one of:
  - 'first names last name, organization'
  - 'first names last name, organization <contributor email address>'
  - 'first names last name, organization <organization email address>'
  - otherDetails : UML::PrimitiveTypes::String [0..*]

The otherDetails tag contains additional information about an Authored Resource.

- otherDetails_id : UML::PrimitiveTypes::String [0..*]
  The otherDetails_id tag contains the name associated with additional information about an Archetype. otherDetails_id Strings are matched to otherDetails Strings by order.
- references : UML::PrimitiveTypes::String [0..*]
  The references tag contains external citations and references.
- references_id : UML::PrimitiveTypes::String [0..*]
  The references_id tag contains the name associated with external citations and references. references_id Strings are matched to references Strings by order.
- resourceDocumentLanguage : UML::PrimitiveTypes::String [0..1]
  The language (e.g., [AOM], CEM, ...) of the source of the constraints, if any. This tag has no mapping to [AOM].
- resourceDocumentSyntax : UML::PrimitiveTypes::String [0..1]
  The syntax of the resource document ([ADL], [XML], [XMI], ...). This tag has no mapping to [AOM].
- resourceSource : UML::PrimitiveTypes::String [0..1]
  A URI that references the source document (if any) from which the original resource was derived. This tag has no mapping to [AOM].
- resourceSourceURI : UML::PrimitiveTypes::String [0..1]
  The resourceSourceURI tag specifies 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.
  This tag has no mapping to [AOM].
- resource_package_uri : UML::PrimitiveTypes::String [0..1]
  The optional resource_package_uri property enables the recording of a reference to a package of archetypes or other resources, to which this archetype is considered to belong. It may be in the form of 'text <URL>'. This tag has no mapping to [AOM].
- uid : UML::PrimitiveTypes::String [0..1]
  The uid attribute defines a machine identifier equivalent to the human readable archetype_id.semantic_id, i.e., ARCHETYPE_HRID up to its major version, and changes whenever the latter does. It is defined as optional but to be practically useful would need to be mandatory for all archetypes within a custodian organization where this identifier was in use. It could in principle be synthesized at any time for a custodian that decided to implement it. This tag has no mapping to [AOM].
- ip_acknowledgements : UML::PrimitiveTypes::String [0..*]
  The ip_acknowledgements tag contains acknowledgements about intellectual property.
- ip_acknowledgements_id : UML::PrimitiveTypes::String [0..*]
  The ip_acknowledgements_id tag contains the name associated with acknowledgements about intellectual property.
  ip_acknowledgements_id Strings are matched to ip_acknowledgements Strings by order.

8.5.10 ComplexObjectConstraint [Stereotype]

Description

A «ComplexObjectConstraint» is a Classifier constraining (directly or indirectly) a Reference Model Classifier. A ComplexObjectConstraint may constrain the existence, cardinality and/or possible values of any or all of the constrained Reference Model Classifier attributes.

A ComplexObjectConstraint maps to an [AOM] C_COMPLEX_OBJECTl. 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 C_OBJECT/rm_type_name property 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 subsetting/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 evaluates 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 typed by 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 in AML as isLeaf.

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 (but is instead mapped to the aggregation of the Property).

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

Figure 8.4 Constraint Profile

Direct Superclasses (Generalization)

ConstraintProfile::ObjectConstraint

8.5.12 ObjectConstraint [Stereotype]

Meta-classes

UML::Classifier

Direct Subclasses (Specialization)

ConstraintProfile::ArchetypeRoot

8.5.7 ArchetypeRoot [Stereotype]

Constraints

- **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()))
```

- **singleParent**

Every constraint must specialize exactly one Class.

[OCL]
```
self.base_Classifier.generalization->size() = 1
```

- **[AOM] ASSERTION:Invariant:Expression_valid**

An [AOM] ASSERTION/expression must exist and must be of type Boolean. It maps to a UML Constraint/specification.

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

- **[AOM] ASSERTION:Invariant:Tag_valid**

An [AOM] tag maps to the name of a UML Constraint and must either be defined or be a non-empty String.

[OCL]
```
(self.constrainedElement.oclIsKindOf(Property) and
self.constrainedElement.oclAsType(Property).namespace.stereotypedBy('ComplexObject
Constraint')) implies(self.name.oclIsUndefined() or (self.name<>'') )
```

- **[AOM] C_ATTRIBUTE:Invariant:Alternatives_valid**

If this is a Property whose upper bound is 1, then the occurrences of all children have an upper bound of 1.

[OCL]
```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint'))
.subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper=1) ->
forAll(refProperty| self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->union(sibling.redefinedProperty)->
includes(refProperty))->forAll(p|p.upper=1) )
```

- **[AOM] C_ATTRIBUTE:Invariant:Cardinality_valid**

The [AOM] C_ATTRIBUTE/is_multiple is false if children are alternative child objects (e.g., a choice); otherwise, a value of true means children are a collection of independent child objects. The [AOM] C_ATTRIBUTE/is_multiple =false maps to AML Properties that have an aggregation of 'shared,' where as a value of true maps to AML Properties that have an aggregation of 'composite.' The sibling AML Properties mapped to the [AOM] children must all have the same aggregation kind.

[OCL]
```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint'))
.subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper>0) ->

forAll(refProperty| self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->
union(sibling.redefinedProperty)->includes(refProperty)) ->
forAll(p|p.aggregation=self.aggregation) )
```

- **[AOM] C_ATTRIBUTE:Invariant:Children_occurrences_lower_sum_validity**

If the upper multiplicity of this Property is bounded, then the sum of the multiplicity lower bounds of children is less than or equal to the upper multiplicity of the constrained Property.

[OCL]
```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint'))
.subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper>0) ->
forAll(refProperty| self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->
union(sibling.redefinedProperty)->includes(refProperty)) .lower->
sum()<=refProperty.upper )
```

- **[AOM] C_ATTRIBUTE:Invariant:Children_orphans_validity**

If this is a Property with a bounded upper limit, then the minimum number of possible children must be less than or equal to the upper value of the constrained Property.

[OCL]
```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint'))
.subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper>0) ->
forAll(refProperty| self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->union(sibling.redefinedProperty)->
includes(refProperty)) ->select(p|p.lower>0) ->size()<refProperty.upper )
```

- **[AOM] C_ATTRIBUTE:Invariant:Children_validity**

An [AOM] C_ATTRIBUTE/children may be undefined, meaning the attribute type may be any legal type of the Reference Model attribute, or there is a non-empty list of archetype-specific children (types derived from the Reference Model type). An [AOM] C_ATTRIBUTE with undefined children maps to an AML Property whose type is the same as the Reference Model attribute's type. An [AOM] C_ATTRIBUTE with children maps to a list of Properties which subset or redefine the Reference Model attribute, but whose types are in the Archetype.

This [AOM] Invariant maps to an AML Constraint that the type of a Property may be (directly) a Reference Model Type only if it has no siblings which subset or redefine the same Reference Model attribute.

[OCL]
```
(self.namespace.stereotypedBy('ComplexObjectConstraint') and
not(self.type.getNearestReferenceModel().oclIsUndefined()) ) implies
self.namespace.oclAsType(Class).ownedAttribute->select(a|(a<>self) and
not(a.subsetsProperty->includes(self.subsetsProperty->first())) and
not(a.redefinesProperty->
includes(self.redefinesProperty->first())))->isEmpty()
```

- **[AOM] C_BOOLEAN:Invariant:Binary_consistency**

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)) )
```

- **[AOM] C_BOOLEAN:Invariant:Prototype_value_consistency**

[AOM] P_C_BOOLEAN/constraint maps to the operands of an 'or' Expression, each operand being a LiteralBoolean, and each operand having a unique value.

[English]
```
Each operand of an 'or' Expression must be a unique value.
```

- **[AOM] C_COMPLEX_OBJECT:Invariant:Any_allowed_validity**

An [AOM] C_COMPLEX_OBJECT/attributes which is empty maps to a UML Property whose type is a Reference Model Classifier (i.e., no additional Constraints on value).

[English]
```
The invariant is definitional; it defines representation for a Property which has no
value constraints (beyond the Reference Model type).
```

- **[AOM] C_COMPLEX_OBJECT:Invariant:Prohibited_validity**

A UML Property with upper multiplicity of 0 is prohibited from having any value.

[English]
```
This Constraint is enforced by UML semantic for a Multiplicity upper bound of 0.
```

- **[AOM] C_COMPLEX_OBJECT:InvariantAttributes_valid**

The [AOM] C_COMPLEX_OBJECT/attributes attribute must be defined. This is mapped to AML ownedAttributes, which is always defined (but may be empty).

[English]
```
The [AOM] invariant is always satisfied since its mapping to UML is ownedAttributes,
which is a collection and is never oclIsUndefined().
```

- **[AOM] C_COMPLEX_OBJECT:Tuples_valid**

The [AOM] C_COMPLEX_OBJECT/attribute_tuples is a list of C_ATTRIBUTE_TUPLE. Each member of C_ATTRIBUTE_TUPLE must be a member of C_COMPLEX_OBJECT/attributes.

The [AOM] C_ATTRIBUTE_TUPLE maps to a UML Classifier nested inside this Complex Object, and specializes this Complex Object. Each ownedAttribute of the nested Classifier must subset or redefine a Property owned by this 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))
```

- **[AOM] C_COMPLEX_OBJECT_PROXY:Invariant:Consistency**

An [AOM] C_COMPLEX_OBJECT_PROXY cannot be a proxy for any allowed Reference Model Classifier. This [AOM] Invariant is mapped to an AML Constraint that a Property of a Complex Object Constraint, whose aggregation is none, cannot be (directly) typed by a Reference Model Classifier.

[OCL]
```
((self.aggregation=AggregationKind::none) and
self.namespace.stereotypedBy('ComplexObjectConstraint')) implies
self.type.getNearestReferenceModel().oclIsUndefined()
```

- **[AOM] C_DATE:Invariant:Pattern_validity**

An [AOM] C_DATE/pattern_constraint (if present) maps to the first operand of a 'P_C_DATE' Expression, where that first operand is a LiteralString named 'pattern_constraint' and its value must conform to the ISO 8601 date constraint pattern.

[English]
```
An [AOM] C_DATE/pattern_constraint (if present) maps to the first operand of a 'P_C_DATE'
Expression,  where that first operand is a LiteralString named 'pattern_constraint' and
its value must conform to the ISO 8601 date constraint pattern.
```

- **[AOM] C_OBJECT:Invariant:Rm_type_name_valid**

An [AOM] rm_type_name maps to the name of the Reference Model Classifier which is the supertype of an Archetype Classifier. The Reference Model Classifier must have a non-empty name.

[OCL]
```
self.base_Classifier.general->forAll(g|not(g.name.oclIsUndefined()) and (g.name<>''))
```

- **[AOM] C_PRIMITIVE_OBJECT:Invariant:Assumed_value_valid**

The [AOM] C_PRIMITIVE_OBJECT/assumed_value maps to an AML Constraint on a Property whose specification includes a type-specific declaration of the assumed value. The assumed value is a ValueSpecification that 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.

[English]
```
The [AOM] C_PRIMITIVE_OBJECT/assumed_value maps to an AML Constraint on a Property whose
specification includes a type-specific declaration of the assumed value. The assumed
value is a ValueSpecification that 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.
```

- **[AOM] C_PRIMITIVE_OBJECT:Invariant:Representation_validity**

An [AOM] C_PRIMITIVE_OBJECT is mapped to an AML Property that is constrained by a UML Constraint and whose type resolves to a recognized AML Primitive Type (resides in either the UML Type Library or the XML Type Library). Resolving a type may include navigating the «MappedDataType» Abstraction from a Reference Model type to one of the AML Primitive Types.

[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()
```

- **[AOM] C_TERMINOLOGY_CODE:Invariant:Terminology_id_validity**

An [AOM] C_TERMINOLOGY_CODE/constraint maps to a UML Constraint. The Constraint is owned by a Complex Object Constraint Classifier, but constrains a Property. The Constraint has a specification of a "=" Expression. The Expression must have an operand that is an InstanceValue. The InstanceValue has an instance that must be an Archetype Term from the Archetype's Identifier Definition Enumeration.

[OCL]
```
self ->forAll(constraint|  constraint.namespace.
stereotypedBy('ComplexObjectConstraint') and
constraint.constrainedElement->forAll(ce|ce.oclIsKindOf(Property)) and
constraint.specification.oclIsKindOf(Expression) and
(constraint.specification.oclAsType(Expression).symbol='=') and
constraint.specification.oclAsType(Expression).operand->
forAll(o|o.oclIsKindOf(InstanceValue)) ) implies
self.specification.oclAsType(Expression).operand->
forAll(o|o.oclAsType(InstanceValue).instance.stereotypedBy('ArchetypeTerm'))
```

- **[AOM] EXPR_BINARY_OPERATOR:Invariant:left_operand_valid**

The [AOM] EXPR_BINARY_OPERATOR maps to a UML Expression whose symbol is one of the defined [AOM] binary operators and which has exactly two operands.

[OCL]
```
(self.getNearestPackage().stereotypedBy('Archetype')      and(
(self.symbol='=')          or (self.symbol='<>')      or
(self.symbol='<=')         or (self.symbol='<')       or
(self.symbol='>=')         or (self.symbol='>')       or
(self.symbol='matches')    or (self.symbol='and')     or
(self.symbol='or')         or (self.symbol='xor')     or
(self.symbol='implies')    or (self.symbol='for_all') or
(self.symbol='exists')     or (self.symbol='+')       or
(self.symbol='-')          or (self.symbol='*')       or
(self.symbol='/')          or (self.symbol='^') ) )
implies (self.operand->size()=2)
```

- **[AOM] EXPR_BINARY_OPERATOR:Invariant:right_operand_valid**

An [AOM] EXPR_BINARY_OPERATOR maps to a UML Expression whose symbol is one of the defined [AOM] binary operators and which has exactly two operands.

[OCL]
```
(self.getNearestPackage().stereotypedBy('Archetype')   and(
(self.symbol='=')          or (self.symbol='<>'       or
(self.symbol='<=')         or (self.symbol='<')       or
(self.symbol='>=')         or (self.symbol='>')       or
(self.symbol='matches')    or (self.symbol='and')     or
(self.symbol='or')         or (self.symbol='xor')     or
(self.symbol='implies')    or (self.symbol='for_all') or
(self.symbol='exists')     or (self.symbol='+')       or
```

```
(self.symbol='-')          or (self.symbol='*')       or
(self.symbol='/')          or (self.symbol='^') ) )
implies (self.operand->size()=2)
```

- **[AOM] EXPR_ITEM:Invariant:Type_valid**

The [AOM] EXPR_ITEM/type must be defined. It maps to a UML ValueSpecification/type.

[OCL]
```
self.getNearestPackage().stereotypedBy('Archetype')
implies not(self.type.oclIsUndefined())
```

- **[AOM] EXPR_UNARY_OPERATOR:Invariant:operand_valid**

An [AOM] EXPR_UNARY_OPERATOR maps to a UML Expression having a symbol which is one of the defined [AOM] unary operators and which has exactly one operand.

[OCL]
```
(self.getNearestPackage().stereotypedBy('Archetype')  and(
(self.symbol='=')          or (self.symbol='<>')      or
(self.symbol='<=')         or (self.symbol='<')       or
(self.symbol='>=')         or (self.symbol='>')       or
(self.symbol='matches')    or (self.symbol='and')     or
(self.symbol='or')         or (self.symbol='xor')     or
(self.symbol='implies')    or (self.symbol='for_all') or
(self.symbol='exists')     or (self.symbol='+')       or
(self.symbol='-')          or (self.symbol='*')       or
(self.symbol='/')          or (self.symbol='^') ) )
implies (self.operand->size()=1)
```

- **[AOM] OPERATOR_KIND:Invariant:Validity**

The [AOM] OPERATOR_KIND is mapped to a UML Expression symbol. The UML Expressions symbol must correspond to one of the operators defined in [AOM] OPERATOR_KIND.

[OCL]
```
(self.getNearestPackage().stereotypedBy('Archetype')  and(
(self.symbol='=')          or (self.symbol='<>')      or
(self.symbol='<=')         or (self.symbol='<')       or
(self.symbol='>=')         or (self.symbol='>')       or
(self.symbol='matches')    or (self.symbol='and')     or
(self.symbol='or')         or (self.symbol='xor')     or
(self.symbol='implies')    or (self.symbol='for_all') or
(self.symbol='exists')     or (self.symbol='+')       or
(self.symbol='-')          or (self.symbol='*')       or
(self.symbol='/')          or (self.symbol='^') ) )
implies( (self.operand->size()=2) or (self.operand->size()=1) )
```

- **[AOM] QUERY_VARIABLE:Invariant:Context_valid**

The [AOM] QUERY_VARIABLE maps to a UML InstanceSpecification whose specification is an OpaqueExpression. The [AOM] QUERY_VARIABLE/context maps to the name of the UML OpaqueExpression. The name of the OpaqueExpression must be either undefined or non-empty. The [AOM] QUERY_VARIABLE/query_id maps to the name of the UML 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<>'') )
```

- **[AOM] 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('ArchetypeTerm') and
o.oclAsType(InstanceValue).instance.getNearestPackage().
nestingPackage.nestingPackage=self.getNearestPackage()                )
```

- **[AOM] VACMCO-cardinality/occurrences orphans**

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.

[OCL]
```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint'))
.subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper>0) ->
forAll(refProperty| self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->union(sibling.redefinedProperty)->
includes(refProperty)).lower->sum()<=(refProperty.upper-1) )
```

- **[AOM] VACMCU-cardinality/occurrences upper bound validity**

When a cardinality with a finite upper bound is declared 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].
```

- **[AOM] VACSO-single-valued attribute child object occurrences validity**

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 «ComplexObjectConstraint» are indeed redefined or subsetted.

[OCL]
```
self.namespace.stereotypedBy('ComplexObjectConstraint') implies(
self.subsettedProperty->notEmpty() or self.redefinedProperty->notEmpty() )
```

- **[AOM] 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 specialization depth, and must be defined in the terminology.

[OCL]
```
self.base_Classifier.supplierDependency→
exists(d|d.stereotypedBy('ArchetypeDefinition')) implies
self.id.oclAsType(EnumerationLiteral)→
forAll(e|e.name.match('id1(\\.1)*'))
```

- **[AOM] 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 must exist in a value set within the terminology definition. In AML, the assumed_value is defined as part of the specification of a Constraint on the Property, where the assumed_value is an «ArchetypeTerm». The «ArchetypeTerm» may represent a value code (at-code) or a value set (ac-code). In any case, the term will be in the Identifier Definition.

Note that this [AOM] Validty rule maps to the same form of Constraint as rule VATDF.

[OCL]
```
self->select(constraint|constraint.namespace.
stereotypedBy('ComplexObjectConstraint') and
constraint.constrainedElement->forAll(ce|ce.oclIsKindOf(Property))) ->
select(c|c.specification ->forAll(e|e.oclIsKindOf(Expression) and
(e.oclAsType(Expression).symbol='=') and (e.oclAsType(Expression).operand->
forAll(o|o.oclIsKindOf(InstanceValue)))) ).constrainedElement implies
self.specification.oclAsType(Expression).operand.oclAsType(InstanceValue) ->
forAll(o|o.instance.stereotypedBy('ArchetypeTerm') and
(o.namespace.namespace.namespace.namespace=self.namespace.namespace))
```

- **[AOM] VATDF-value code validity**

Each value code (at-code) used in a term constraint within an Archetype must be an Archetype Term defined in the same Archetype. A term constraint in AML is modeled as a Constraint on a Property, where the Constraint specification is an "=" Expression and the operands are Archetype Terms.

[OCL]
```
self->select(constraint|constraint.namespace.
stereotypedBy('ComplexObjectConstraint') and constraint.constrainedElement->
forAll(ce|ce.oclIsKindOf(Property))) ->select(c|c.specification ->
forAll(e|e.oclIsKindOf(Expression) and (e.oclAsType(Expression).symbol='=') and
(e.oclAsType(Expression).operand->
forAll(o|o.oclIsKindOf(InstanceValue)))) ).constrainedElement implies
self.specification.oclAsType(Expression).operand.oclAsType(InstanceValue) ->
forAll(o|o.instance.stereotypedBy('ArchetypeTerm') and
(o.namespace.namespace.namespace.namespace=self.namespace.namespace))
```

- **[AOM] VCACA-archetype attribute reference model cardinality conformanc**e

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.
```

- **[AOM] VCAEX-archetype attribute reference model existence conformance**

The existence of an attribute 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 rule is enforced by UML validation rules for subsetted or redefined properties.

- **[AOM] 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.

- **[AOM] 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.

- **[AOM] 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 Rule is enforced by UML Namespace constraints for names.

- **[AOM] VDIFP-specialised archetype attribute differential path validity**

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]
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.

- **[AOM] VDIFV-archetype attribute differential path validity**

An archetype may only have a differential path if it is specialized.

[English]
An archetype may only have a differential path if it is specialized.

- **[AOM] VOBAV-object node assumed value validity**

The value of an assumed value must fall within the value space defined by the constraint to which it is attached.

[English]
The value of an assumed value must fall within the value space defined by the constraint to which it is attached.

- **[AOM] VSAM-specialised archetype attribute multiplicity conformance**

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.
```

- **[AOM] VSANCC-specialised archetype attribute node cardinality conformance**

The cardinality of a redefined (multiply-valued) attribute node in a specialized archetype, 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]
```
This [AOM] validation rule is enforced by UML constraints related to cardinality of
subsetted and/or redefined Properties.
```

- **[AOM] VSANCE-specialised archetype attribute node existence conformance**

The existence of a redefined attribute node in a specialized archetype 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]
```
This [AOM] Validation Rule is enforced by UML type and cardinality constraints associated
with subsetted and/or redefined Properties.
```

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

The [AOM] C_OBJECT/node_id must be valid with respect to any sibling C_OBJECT nodes.

An [AOM] C_OBJECT/node_id is mapped to a UML Constraint Object id tag value. The Constraint Object may be applied to a Classifier, or if the Classifier is not in the same Archetype, to the Property which is typed by the externalClassifier. This [AOM] Validation rule is mapped to an AML Constraint which requires that the id tag values for owned attributes of a Complex Object Constraint are unique.

[OCL]
```
self.base_Classifier.oclIsKindOf(Class) implies
self.base_Classifier.oclAsType(Class).ownedAttribute->
forAll(attribute|(attribute.stereotypedBy('ObjectConstraint') and
(self.base_Classifier.oclAsType(Class).ownedAttribute->
select(a|a<>attribute)->
forAll(sibling|(sibling.stereotypedBy('ObjectConstraint') and
not(sibling.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->
includes(attribute.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->asSequence()->first()))or
(sibling.type.stereotypedBy('ObjectConstraint') and
not(sibling.type.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->
includes(attribute.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->asSequence()->first()))))) or
( attribute.type.stereotypedBy('ObjectConstraint') and
( self.base_Classifier.oclAsType(Class).ownedAttribute →
select(a|a<>attribute)→
forAll(sibling|(sibling.stereotypedBy('ObjectConstraint') and
not(sibling.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).Id→
includes(attribute.type.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id→asSequence ()->first())) or
sibling.type.stereotypedBy('ObjectConstraint') and
not(sibling.type.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->
includes(attribute.type.appliedStereotype('ObjectConstraint').
```

```
oclAsType(ObjectConstraint).id->
asSequence()->first()))))))))
```

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

A redefined object node in a specialized 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.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id →
forAll(e|e=t.appliedStereotype('ObjectConstraint').
oclAsType(ObjectConstraint).id->asSequence()->first())))
```

- **[AOM] VSUNT-use_node specialisation parent validity**

An [AOM] C_COMPLEX_OBJECT_PROXY node may be redefined in a specialized 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.
```

- **[AOM] VUNP-use_node path validity**

The path mentioned in a use_node statement must refer to an object node defined elsewhere in the same archetype or any of its specialization 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 representation for a use_node is a «ComplexObjectConstraint» owned property which has aggregation=none.

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

- **[AOM] VUNT-use_node reference model type validity**

The Reference Model type mentioned in an [AOM] 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.
```

- **[AOM] WACMCL-cardinality/occurrences lower bound validity**

When 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.

[OCL]

```
self->select(s|s.namespace.stereotypedBy('ComplexObjectConstraint')) .
subsettedProperty->union(self.redefinedProperty) ->select(r|r.upper>0) ->
forAll(refProperty|self.namespace.oclAsType(Class).ownedAttribute->
select(sibling|sibling.subsettedProperty->union(sibling.redefinedProperty)->
includes(refProperty)).lower->sum()<=refProperty.upper )
```

8.5.11 Constrains [Stereotype]

Description

A «Constrains» Generalization specifies the Classifier to be constrained by the specific Object Constraint. The specific Classifier of the Generalization is an Object Constraint which may specify constraints on names, possible values, cardinalities, types, and other attributes of the general Classifier. The general of the Generalization may be either a Classifier in the Reference Model or another AML Object Constraint.

The «Constrains» Generalization maps to the [AOM] C_OBJECT/rm_type_name attribute when constraining a Reference Model Classifier, otherwise it maps to the [AOM] ARCHETYPE_CONSTRAINT/parent attribute.

Diagrams

Figure 8.4 Constraint Profile

Meta-classes

UML::Generalization

Constraints

- **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.

[OCL]

```
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')
```

- **specificObjectConstraint**

The specific property must reference an ObjectConstraint.

[OCL]

```
self.base_Generalization.specific.stereotypedBy('ObjectConstraint')
```

8.5.12 ObjectConstraint [Stereotype]

Description

An «ObjectConstraint» models a restriction on the possible values, types, cardinalities, and/or other aspects of the base NamedElement.

If an «ObjectConstraint» is applied to a UML Classifier, the Classifier may only have ownedAttributes that subset or redefine an inherited attribute. If an «ObjectConstraint» is applied to a UML Property, the Property may specify the value space, type, and/or cardinality subject to the UML Property subset/redefinition semantics.

«ObjectConstraint» is a specialization of «IdentifiedItem», thus allowing the base NamedElements to be assigned unique identifiers from external namespaces. In particular, when modeling [AOM], the value of the id will be an «ArchetypeTerm», thus enabling multiple natural language terminology definitions, technology bindings, and value set compositions for the underlying UML named elements.

Diagrams

Figure 8.4 Constraint Profile

Direct Superclasses (Generalization)

TerminologyProfile::IdentifiedItem

8.4.10 IdentifiedItem [Stereotype]

Meta-classes

UML::NamedElement

Direct Subclasses (Specialization)

ConstraintProfile::ArchetypeSlot

8.5.8 ArchetypeSlot [Stereotype]

ConstraintProfile::ComplexObjectConstraint

8.5.10 ComplexObjectConstraint [Stereotype]

Attributes

- is_deprecated : UML::PrimitiveTypes::Boolean [0..1]
  True if this node and by implication all sub-nodes are deprecated for use.

Constraints

- **redefinesGeneralization**

This Classifier must have exactly one generalization, and that is a «Constraint» 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))
```

- **[AOM] ARCHETYPE_CONSTRAIN:Invariant:path_exists**

The path of this node relative to root of the archetype is defined.

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

- **[AOM] CARDINALITY:Invariant:Validity**

A lower interval cannot be unbounded.

[English]
```
UML types lower bound as Integer (instead of UnlimitedNatural). Thus, lower bound cannot
be unbounded.
```

- **[AOM] C_ATTRIBUTE:Invariant:Child_occurrences_validity**

The cardinality of children of property must be within the cardinality of the property.

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

- **[AOM] C_ATTRIBUTE:Invariant:Existence_valid**

In [AOM] C_ATTRIBUTE/existence must have a lower bound greater than or equal to 0 and upper bound of 1.

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

- **[AOM] C_ATTRIBUTE:Invariant:Rm_attribute_name_valid**

This Property must redefine or subset a Reference Model Property.

[OCL]
```
self.base_NamedElement.oclIsKindOf(Property) implies
self.base_NamedElement.oclAsType(Property)->forAll(p|p.redefinedProperty->
notEmpty() or p.subsettedProperty->notEmpty())
```

- **[AOM] C_ATTRIBUTE:Invariant:[Differential_path_valid**

The [AOM] C_ATTRIBUTE/differential_path, if it exists, references the parent object of the C_ATTRIBUTE. The [AOM] C_ATTRIBUTE/differential_path is derived during mapping from AML.

[English]
```
The [AOM] Invariant is enforced during mapping from AML to [AOM].
```

- **[AOM] C_COMPLEX_OBJECT_PROXY:Target_path_valid**

The [AOM] C_COMPLEX_OBJECT_PROXY maps to a UML Property having an aggregation of none and a type of the target object within the same «Archetype» (i.e., an internal reference).

An [AOM] target_path attribute is implicitly mapped to a UML Property type. The UML Property type must be defined.

[OCL]
```
(self.oclIsKindOf(Property) and
(self.oclAsType(Property).aggregation=AggregationKind::none)and
self.namespace.stereotypedBy('ComplexObjectConstraint')) implies
not(self.oclAsType(Property).type.oclIsUndefined())
```

- **[AOM] C_DEFINED_OBJECT:Invariant:Default_value_valid**

The default_value of a Property most conform to the type of the Property as well as any restrictions on the value range.

[English]
```
Validation of type constraints for defaultValue of a Property is enforced by UML Property
type and instance semantics. Value restrictions are expressed as Constraints on the
Property, but are not necessarily enforceable by UML semantics directly.
```

- **[AOM] C_OBJECT:Invariant:Node_id_valid**

If the tag 'id' has a value, then it must be an Archetype Term 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.

[OCL]
```
self.id ->forAll(i|i.stereotypedBy('ArchetypeTerm'))
```

- **[AOM] C_OBJECT:Invariant:Occurrences_validity**

If the upper bound of a Property is 1, then the upper bound of children properties is 1.

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

- **[AOM] C_OBJECT:Invariant:Sibling_order_validity**

If this Archetype is a specialization of another Archetype, then a sibling_order is implicitly derived from a merge of the parent with this Classifier.

[English]
```
The [AOM] sibling_order is mapped from a merge of the current attribute children with the
parent Classifier in the parent Archetype.
```

- **[AOM] SIBLING_ORDER:Invariant:sibling_node_id_validity**

The children of an attribute are ordered.

[English]
```
The order of children of an attribute is enforced by the UML semantic for Property
ordering.
```

- **[AOM] 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]
```
This [AOM] rule is definitional; subtypes of Object Constraint define content and
semantics.
```

- **[AOM] VCOID-object node identifier validity**

Every object node must have a node identifier.

[English]
```
An object constraint may have an id (which maps to [AOM] node_id) specified as a tag.
If not specified, then an implicit node_id is based on the object constraint name. In
case of constraints on primitive types, the derived 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 constraint node.
```

- **[AOM] 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.
```

- **[AOM] 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]
```
This [AOM] validation rule is enforced by UML type conformance constraints associated
with subsetted and/or redefined Properties.
```

- **[AOM] VCOSU- object node identifier validity**

Every object node identifier must be unique within the archetype.

[English]
```
Object node identifiers are the names of Archetype Term EnumerationLiterals within the
Definition Identifier and their uniqueness is enforced by UML name constraints.
```

- **[AOM] VSONCO-specialised archetype redefine object node occurrences validity**

The occurrences of a redefined object node in a specialized 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.
```

- **[AOM] VSONCT- specialised archetype object node reference type conformance**

The Reference Model type of a redefined object node in a specialized 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]
```
This [AOM] validation rule is enforced by UML inheritance from the specialized archetype,
which itself inherits directly or indirectly from a Reference Model.
```

- **[AOM] VSONIN-specialised archetype new object node identifier validity**

If an object constraint node in a specialized 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, specialized at the level of the child archetype.

[English]
```
If an object constraint node in a specialized 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, specialized at the level of the child archetype.
```

- **[AOM] VSONPO- specialised archetype object node prohibited occurrences validity**

The occurrences of a new (i.e., having no corresponding node in the parent Archetype) object constraint node in a specialized archetype, if stated, may not be ‘prohibited’, i.e., have an upper bound of {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)
```

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

The occurrences of a redefined object constraint node in a specialized archetype, may only be prohibited (i.e., have upper bound of {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) )
```

- **[AOM] VSONT-specialised archetype object node meta-type conformance**

The meta-type of a redefined object constraint node (i.e., the [AOM] node type such as C_COMPLEX_OBJECT, etc.) in a specialized 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.
- an ARCHTEYPE_SLOT may be redefined by C_ARCHETYPE_ROOT (i.e., ‘slot-filling’).

See also validity rules VDSSID and VARXID.

This [AOM] Validation rule maps to an AML Constraint:

- A «ComplexObjectConstraint» may specialize a «ComplexObjectConstraint»
- An «ArchetypeRoot» may specialize an «ArchetypeSlot»
- An «ArchetypeRoot» may specialize an «ArchetypeRoot»
- An «ArchetypeSlot» may specialize an «ArchetypeSlot»

[OCL]
```
((self.base_NamedElement.stereotypedBy('ComplexObjectConstraint') and
not self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.getNearestReferenceModel().oclIsUndefined())) implies
self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.stereotypedBy('ComplexObjectConstraint')) )  and
((self.base_NamedElement.stereotypedBy('ArchetypeRoot') and
not self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.getNearestReferenceModel().oclIsUndefined())) implies
self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.stereotypedBy('ArchetypeRoot') or g.stereotypedBy('ArchetypeSlot')) )  and
((self.base_NamedElement.stereotypedBy('ArchetypeSlot') and
not self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.getNearestReferenceModel().oclIsUndefined())) implies
self.base_NamedElement.oclAsType(Classifier).general->
forAll(g|g.stereotypedBy('ArchetypeSlot')) )
```

- **[AOM] VSSM-specialised archetype sibling order validity**

The [AOM] SIBLING_ORDER/sibling_node_id must refer to a node found within the same container. The [AOM] SIBLING_ORDER/sibling_node_id maps to the sequence of Properties in the ownedAttribute of a Class.

[English]
```
The order of sibling children are enforced by the UML semantics of List for the ordered
Attributes of a Class.
```

8.5.13 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 annotatedElement, i.e., some Model Element in the Archetype or Reference Model. Annotations are language specific and are modeled as comments owned by a ResourceTranslation, where the target language is identified by the owning 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

Figure 8.4 Constraint Profile

Meta-classes

UML::Comment

Constraints

- **[AOM] VRANP- annotation path valid**

Each resource annotation must annotate a reference model or archetype element.

[OCL]
```
not(self.base_Comment.annotatedElement->
includes(self.base_Comment.owningElement))
```

8.5.14 ResourceTranslation [Stereotype]

Description

A «ResourceTranslation» models language-specific terminology definitions. Each «ResourceTranslation» contains metadata about the language, metadata about the translation in the context of the Archetype, and an entry for each terminology definition associated with the identified elements within the Archetype definition.

A «ResourceTranslation» maps to [AOM] TRANSLATION_DETAILS and RESOURCE_DESCRIPTION_ITEM.

A Resource Translation is associated with an Archetype Package via a Usage relationship named “original_language” *and* a Usage named “terminology_original_language.” The “original_language” Usage maps to [AOM] AUTHORED_RESOURCE/original_language and the “terminology_original_language” Usage maps to [AOM] ARCHETYPE_TERMINOLOGY/original_language.

Diagrams

Figure 8.4 Constraint Profile

Meta-classes

UML::Enumeration

Attributes

- accreditation : UML::PrimitiveTypes::String [0..1]
  The accreditation tag is used to specify the credentials of the translator.

- other_translation_details : UML::PrimitiveTypes::String [0..*]
  The other_translation_details tag contains other details of a translation.
  This tag maps to the value portion of the [AOM] TRANSLATION_DETAILS.other_details property.
- other_translation_details_id : UML::PrimitiveTypes::String [0..*]
  The other_translation_details_id tag contains the name associated with other details of a translation.
  other_translation_details_id Strings are matched to other_translation_details Strings by order.
- purpose : UML::PrimitiveTypes::String [1]
  The purpose tag specifies the intended design concept of the artefact. This tag maps to the [AOM] attribute RESOURCE_DESCRIPTION_ITEM/purpose.
- keywords : UML::PrimitiveTypes::String [0..*]
  The keywords tag is a list of keywords for the artifact. This tag maps to the [AOM] attribute RESOURCE_DESCRIPTION_ITEM/keyword.
- use : UML::PrimitiveTypes::String [0..1]
  The use tag specifies uses of the archetype. This tag maps to the attribute [AOM] RESOURCE_DESCRIPTION_ITEM/*use*.
- misuse : UML::PrimitiveTypes::String [0..1]
  The misuse tag specifies common errors of use, or apparently reasonable but wrong assumptions about use.
  This tag maps to the attribute [AOM] RESOURCE_DESCRIPTION_ITEM/misuse.
- copyright : UML::PrimitiveTypes::String [0..1]
  The copyright property records the copyright applying to the artefact, and is normally in the standard form '(c) name' or '(c) year name'. The special character © may also be used (UTF-8 0xC2A9).
- original_resource_uri : UML::PrimitiveTypes::String [0..*]
  The original_resource_uri tag specifies references to original resources for this natural language. This tag maps to [AOM] RESOURCE_DESCRIPTION_ITEM/original_resource_uri.
- original_resource_uriId : UML::PrimitiveTypes::String [0..*]
  The original_resource_uriId tag contains the name associated with original resources for this natural language.
  original_resource_uriId Strings are matched to original_resource_uri Strings by order.
- other_details : UML::PrimitiveTypes::String [0..*]
  The other_details tag specifies additional information about the translation.
  This tag maps to attribute [AOM] RESOURCE_DESCRIPTION_ITEM/other_detail.
- other_detailsId : UML::PrimitiveTypes::String [0..*]
  The other_detailsId tag contains the name associated with additional information about the translation.
  other_detailsId Strings are matched to other_details Strings by order.
- author_name : UML::PrimitiveTypes::String [0..1]
  The author_name tag specifies the name of translation author.
  This tag maps to [AOM] TRANSLATION_DETAILS/author item where id='name'.
- author_organization : UML::PrimitiveTypes::String [0..1]
  The author_name tag specifies the name author's organization.
  This tag maps to [AOM] TRANSLATION_DETAILS/author item where id='organization'.
- author_email : UML::PrimitiveTypes::String [0..1]
  The author_ email tag specifies the name author's e-mail address.
  This tag maps to [AOM] TRANSLATION_DETAILS/author item where id='email'.
- author_date : UML::PrimitiveTypes::String [0..1]
  The author_ date tag specifies the date that the author made the translation.
  This tag maps to [AOM] TRANSLATION_DETAILS/author item where id='date'.

- version_last_translated : UML::PrimitiveTypes::String [0..1]
  The version_last_translated tag is used to specify when the last translation was carried out for this language. The value of the tag is expected to be a physical_id, as defined for the [AOM] attribute ARCHETYPE_HRID/physical_id. This tag enables maintainers to know when new translations are needed for some or all languages.

  This tag maps to the [AOM] attribute TRANSLATION_DETAILS/version_last_translated.

Constraints

- **translationEntries**

All of the ownedLiterals must be stereotyped by IdEntry.

[OCL]
```
self.base_Enumeration.ownedLiteral->forAll(ol|ol.stereotypedBy('IdEntry'))
```

- **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()
```

- **[AOM] 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()
```

This page intentionally left blank.

9 AML-UML Transformation Reference (Informative)

9.1 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. 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 case of apparent conflict between the informative orientation provided in this clause and the QVT, the QVT takes precedence.

9.1.1 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 9.1. 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. 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.* A Reference Model used by the Clinical Information Modeling Initiative.
- *openEHR*. 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 an «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 the Classifiers owned by the «Archetype»s within the «ArchetypeLibrary».

The 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.

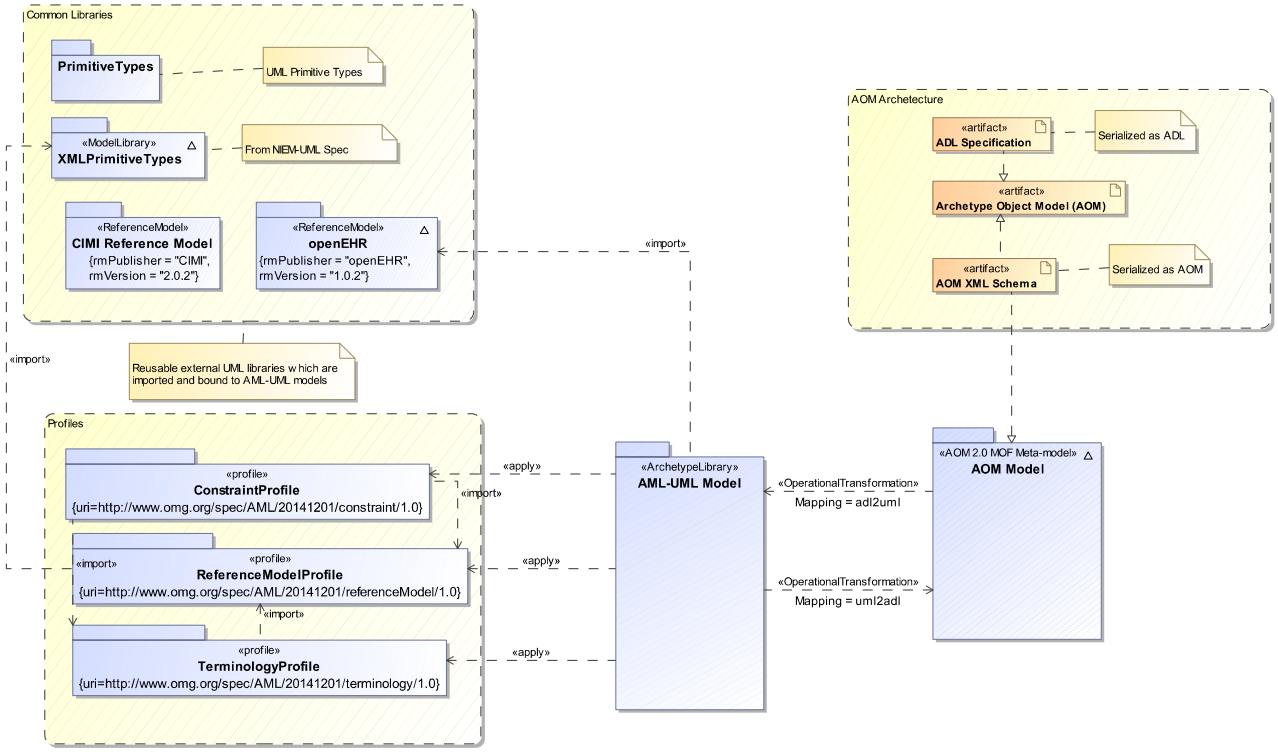

Figure 9.1 - AML Provisioning Context

9.1.2 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 plaform-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.

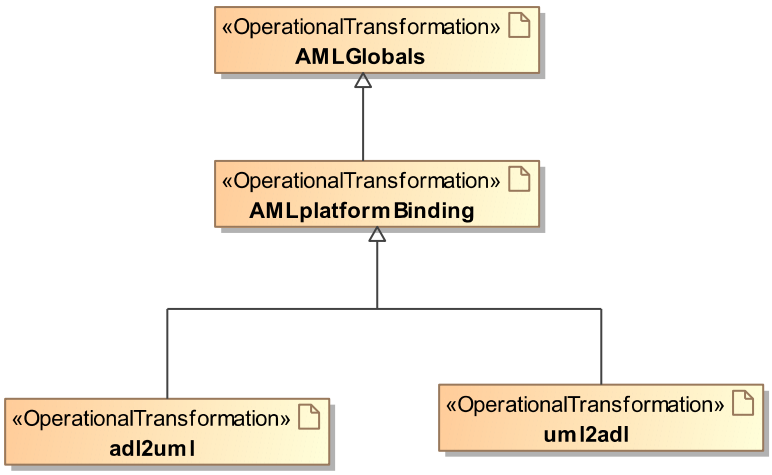

Figure 9.2 - AML Transformations

9.1.3 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 9.3 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 operation.
- 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.

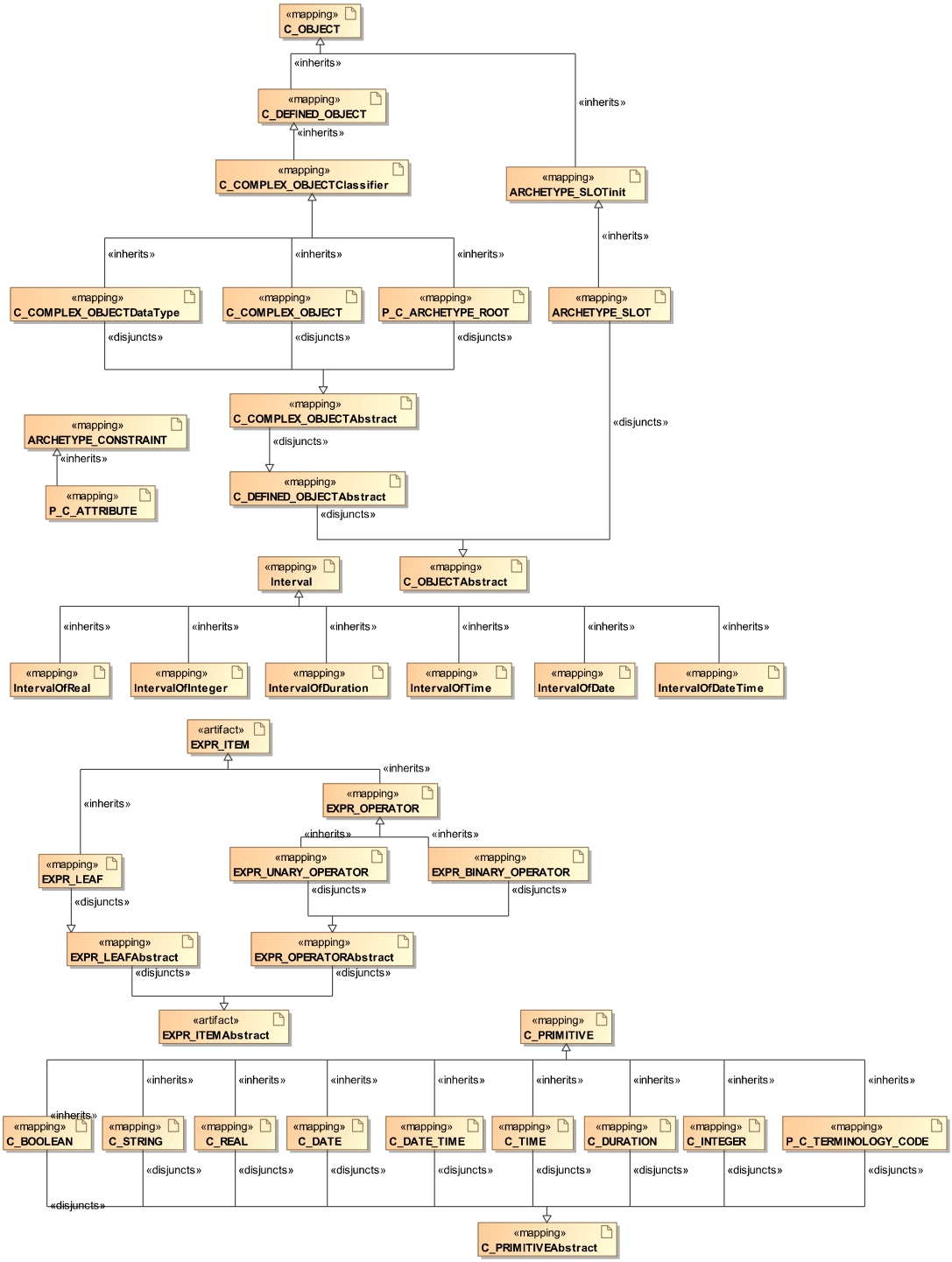

Figure 9.3 - AML Transformation Disjunction and Inheritance

9.1.4 Transformation Notation

Figure 9.4 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.

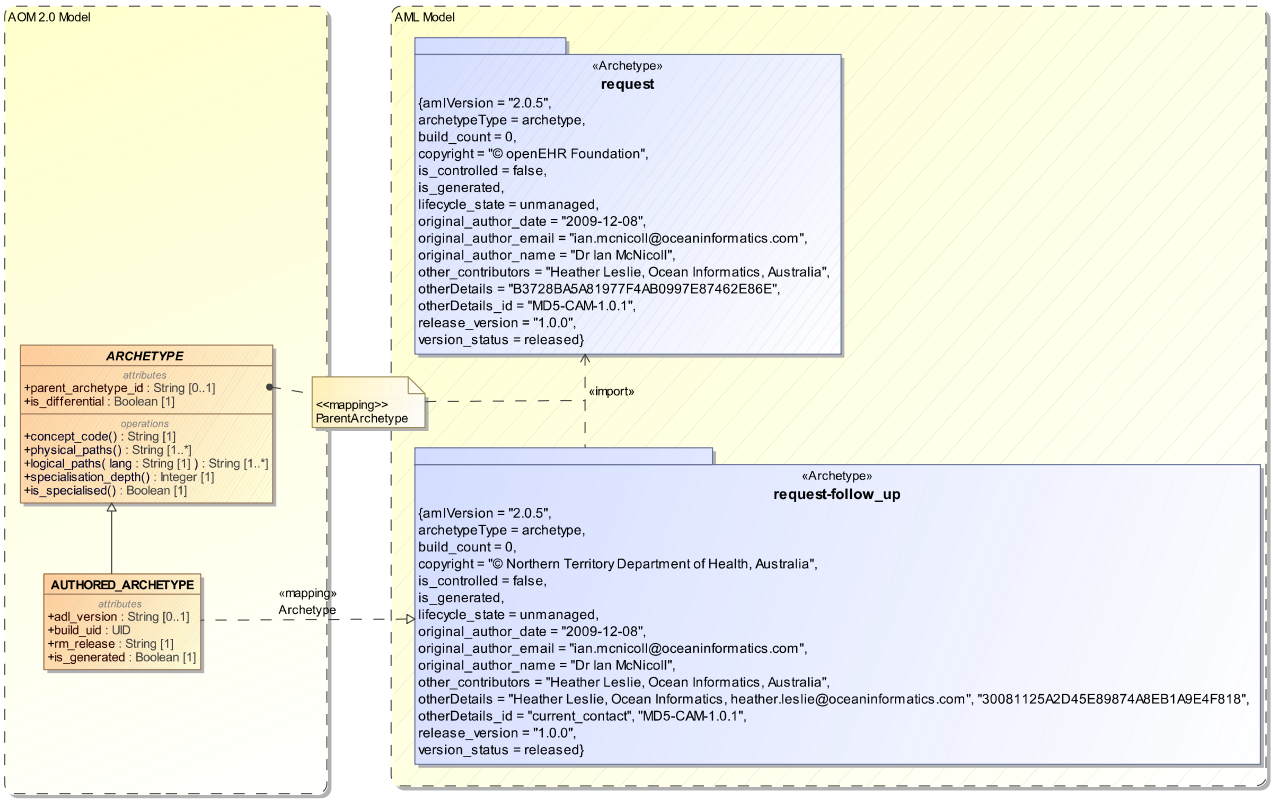

Figure 9.4 - AML Transformation Mapping Notation Overview

9.1.5 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.

9.1.6 Global Properties

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

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

9.2 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 9.5 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.

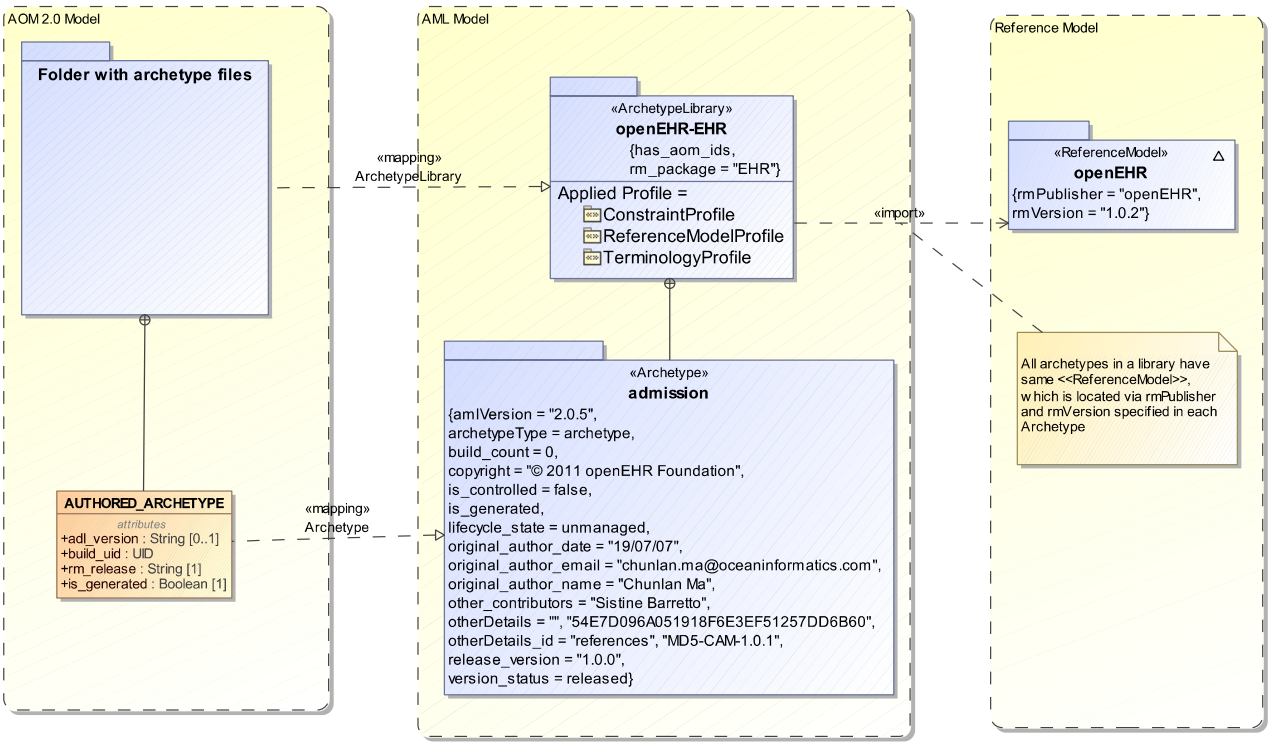

Figure 9.5 - «Archetype Library» Mapping Overview

9.3 Archetype

Figure 9.6 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».

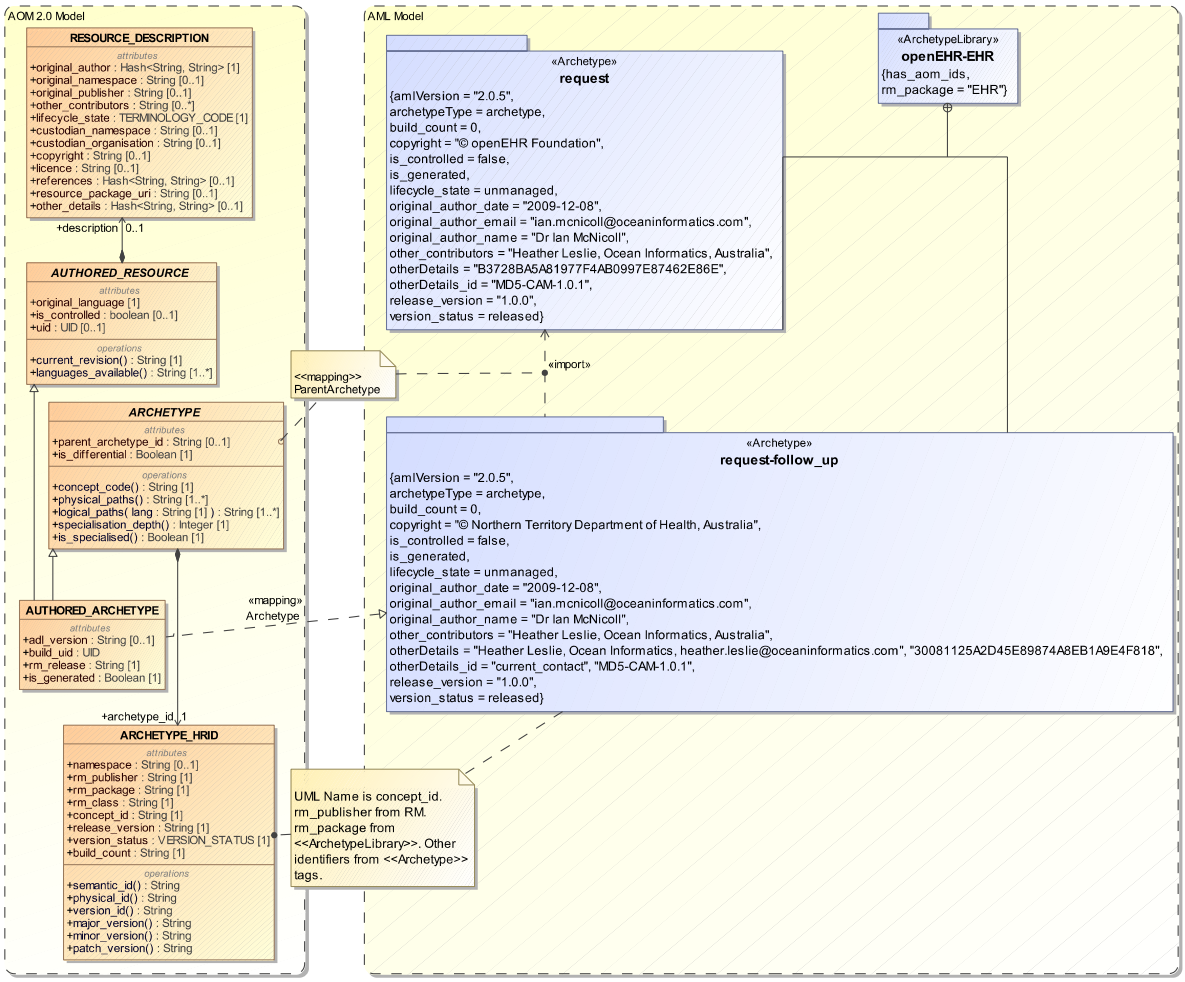

Figure 9.6 - «Archetype» Mapping Overview

9.4 Terminology Definition

Figure 9.7 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.

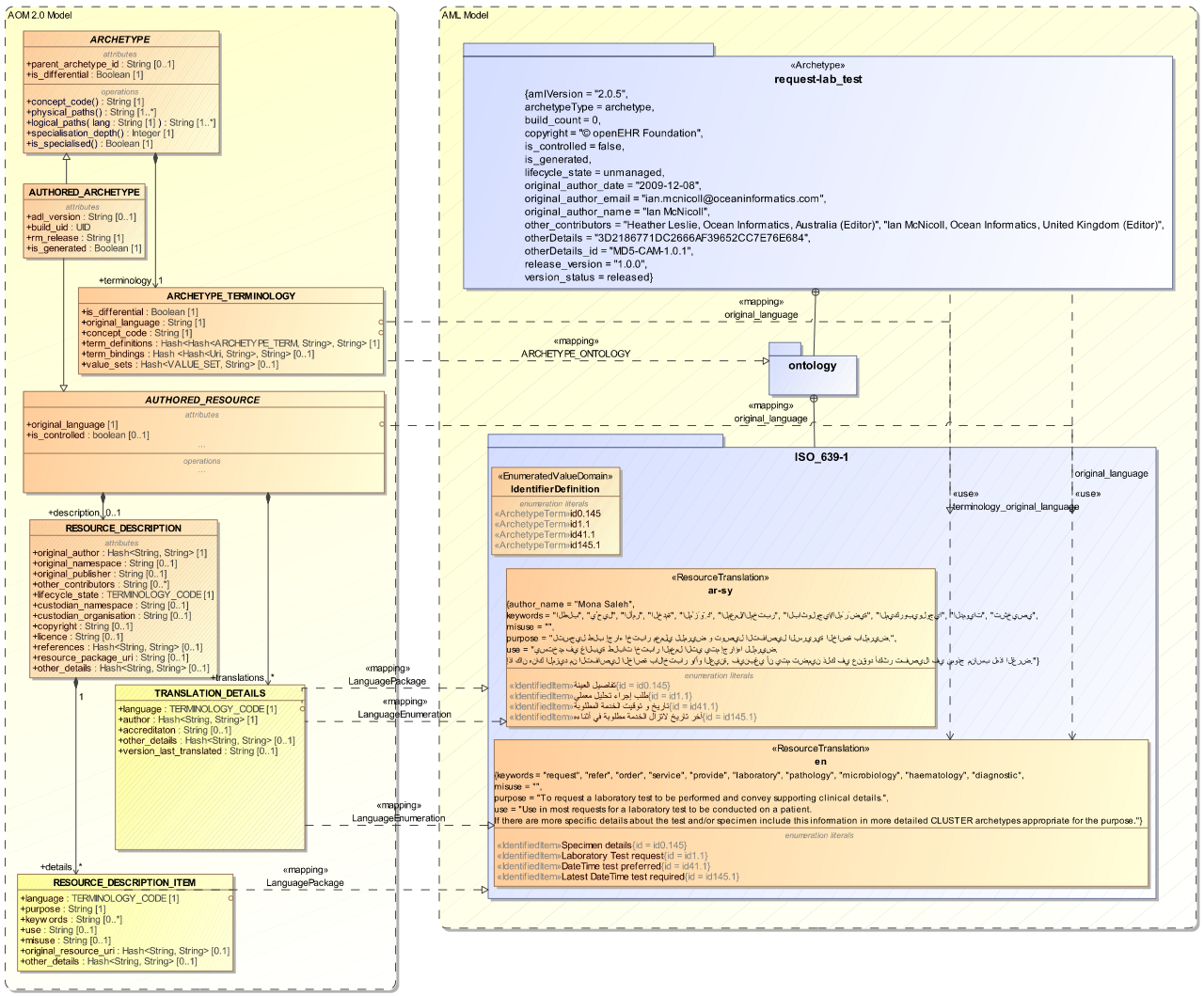

Figure 9.7 - Terminology Definition Mapping Overview

9.5 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».

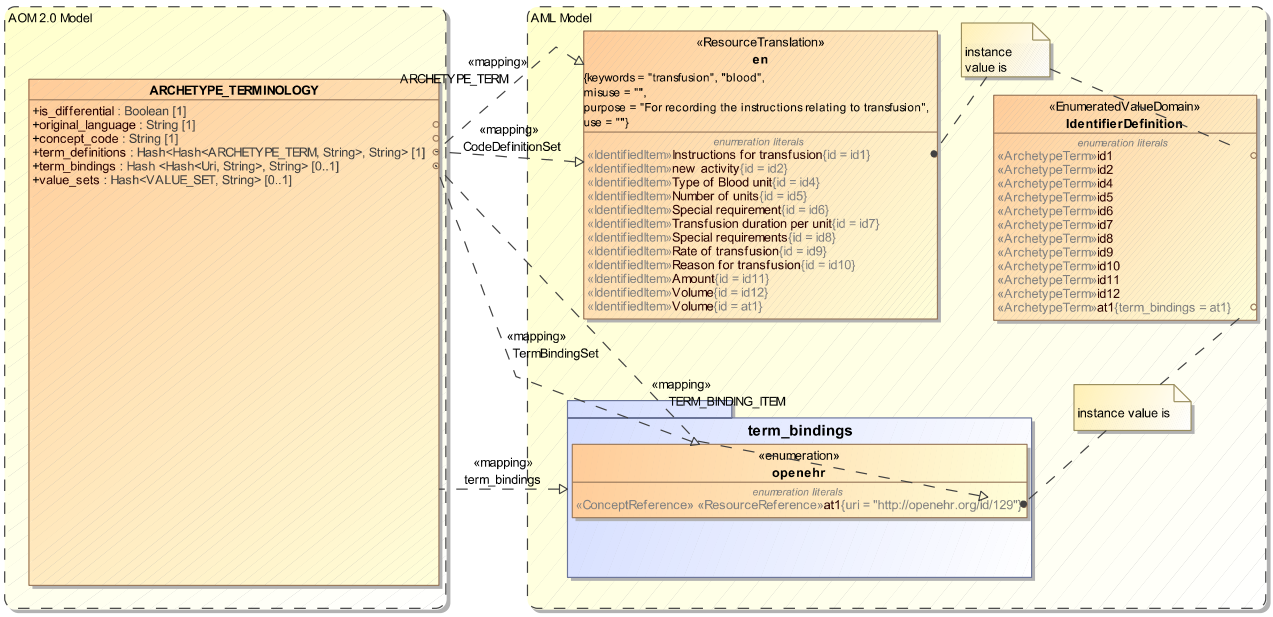

Figure 9.8 - Terminology Binding Mapping Overview

9.6 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.

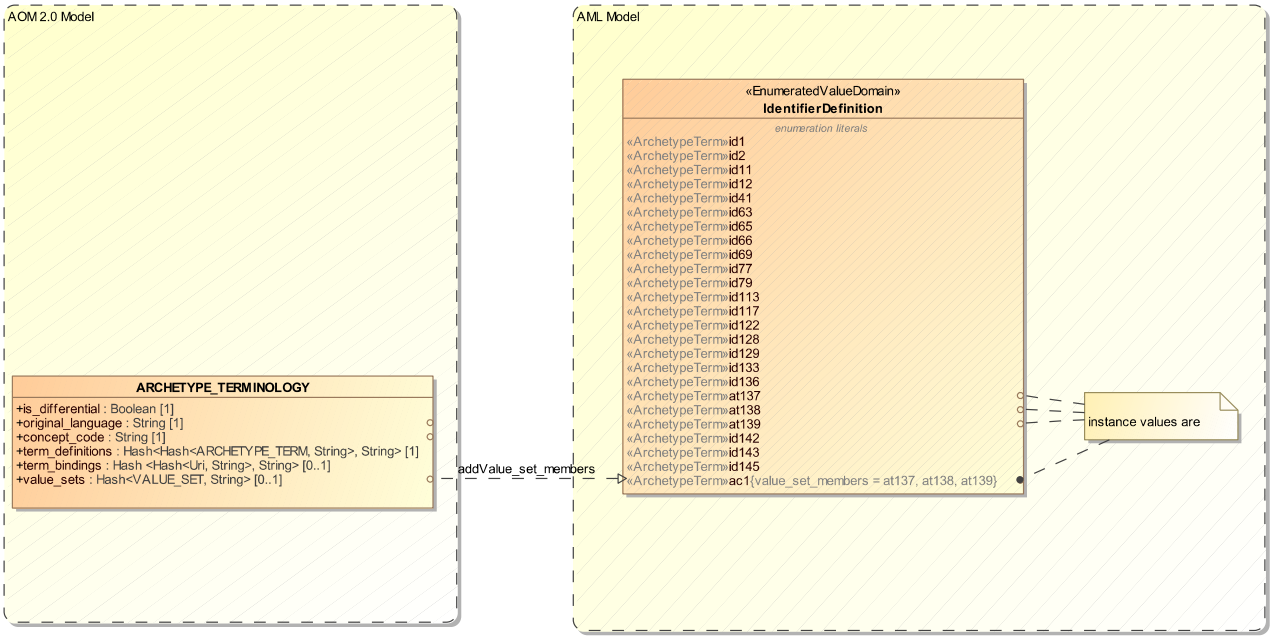

Figure 9.9 - Local Value-sets

9.7 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.

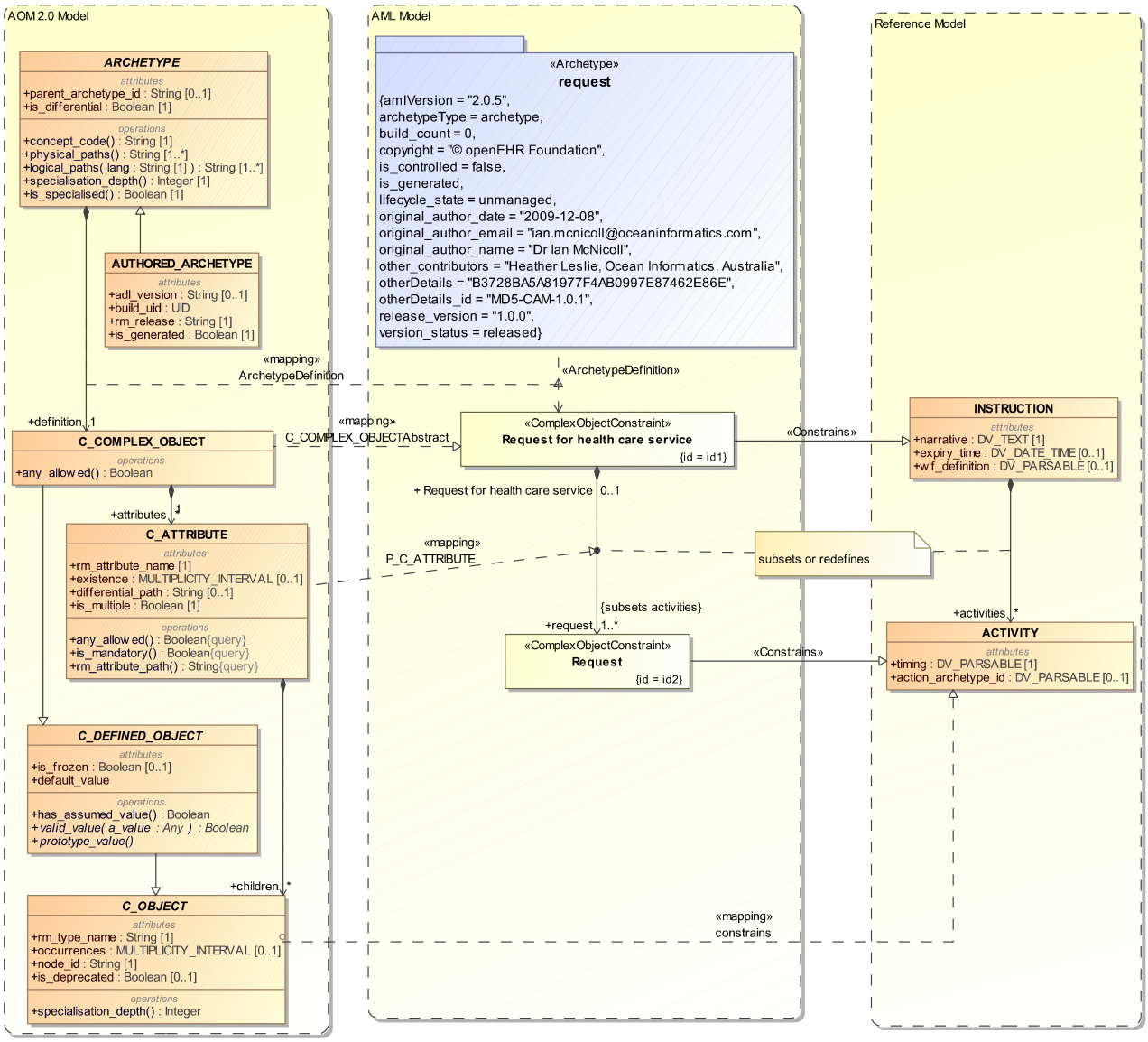

Figure 9.10 - Archetype Definition Mapping Overview

9.8 Object References

An AOM C_OBJECT has specializations that 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.

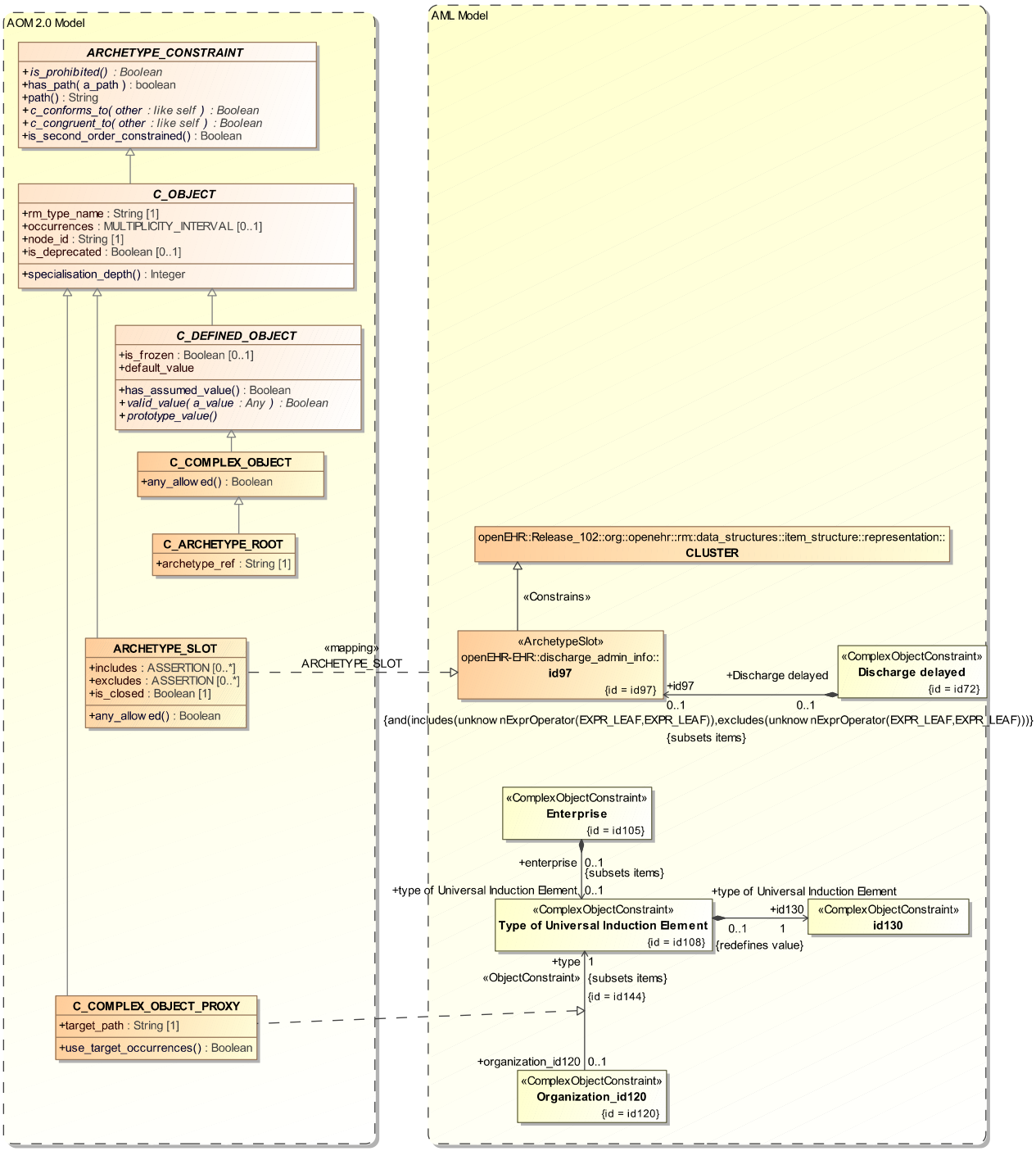

Figure 9.11 - Object Reference Mapping

9.9 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 that 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.

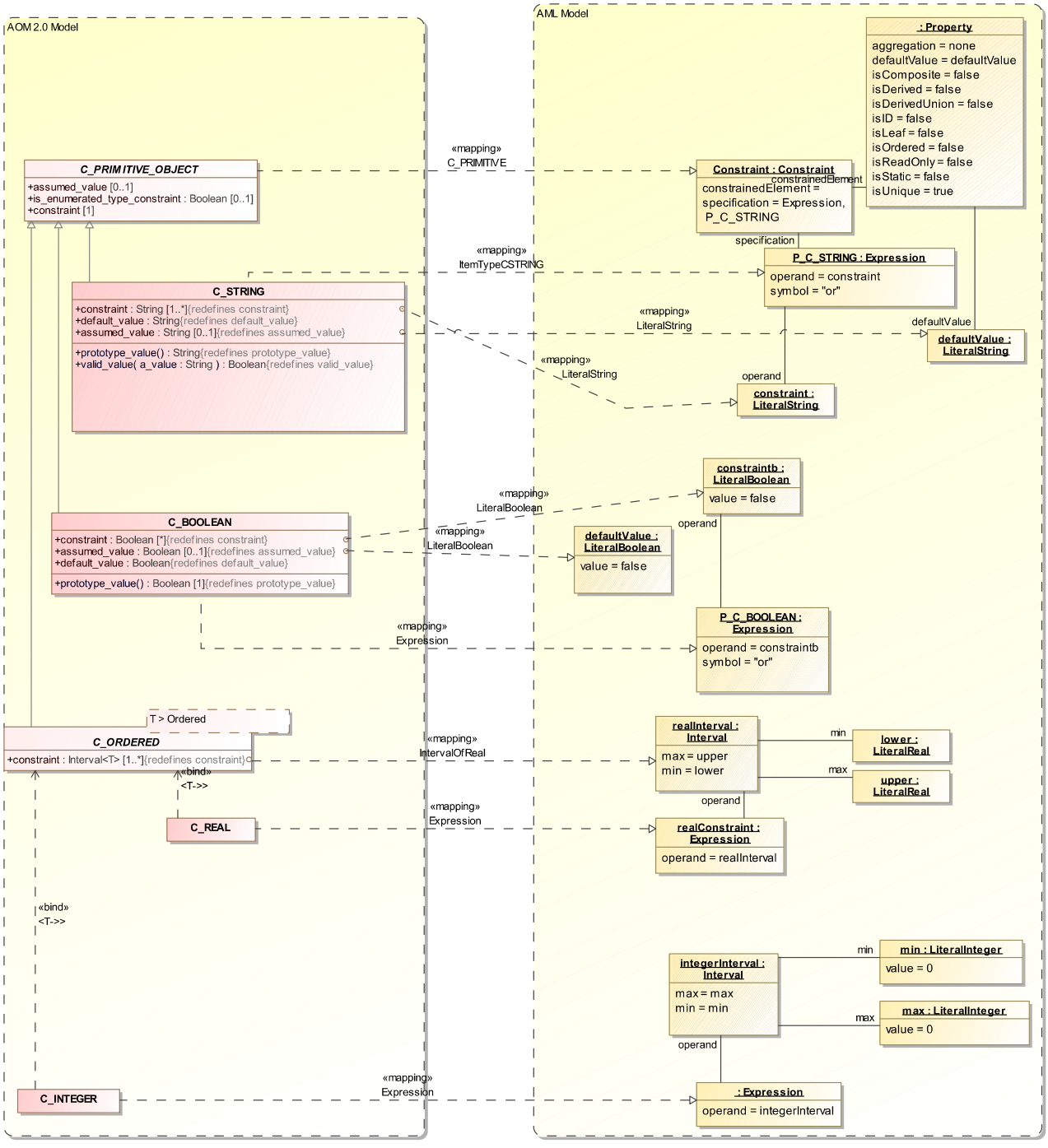

Figure 9.12 - Primitive Constraints

9.10 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.

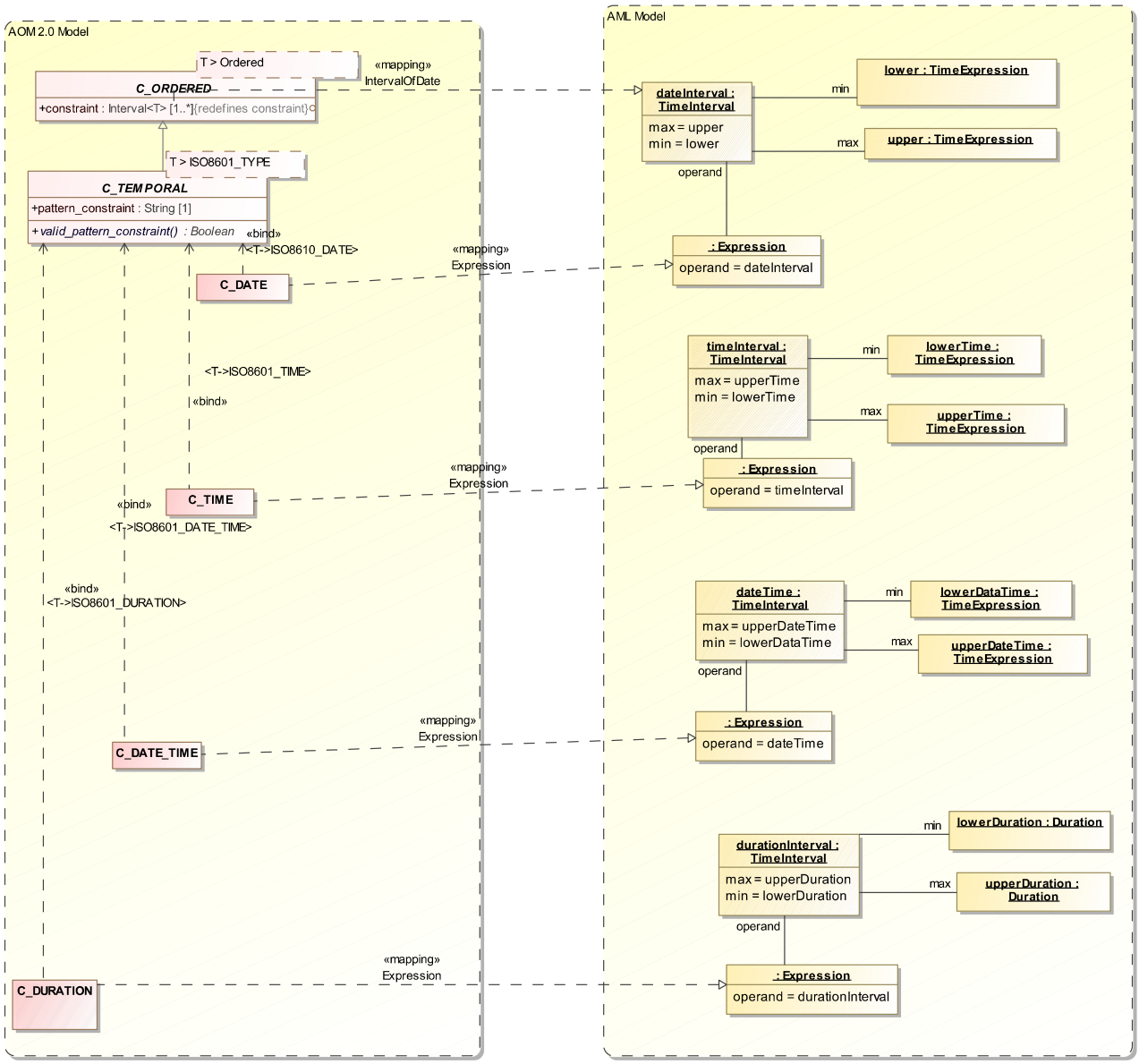

Figure 9.13 - Temporal Constraints Mapping Overview

9.11 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.

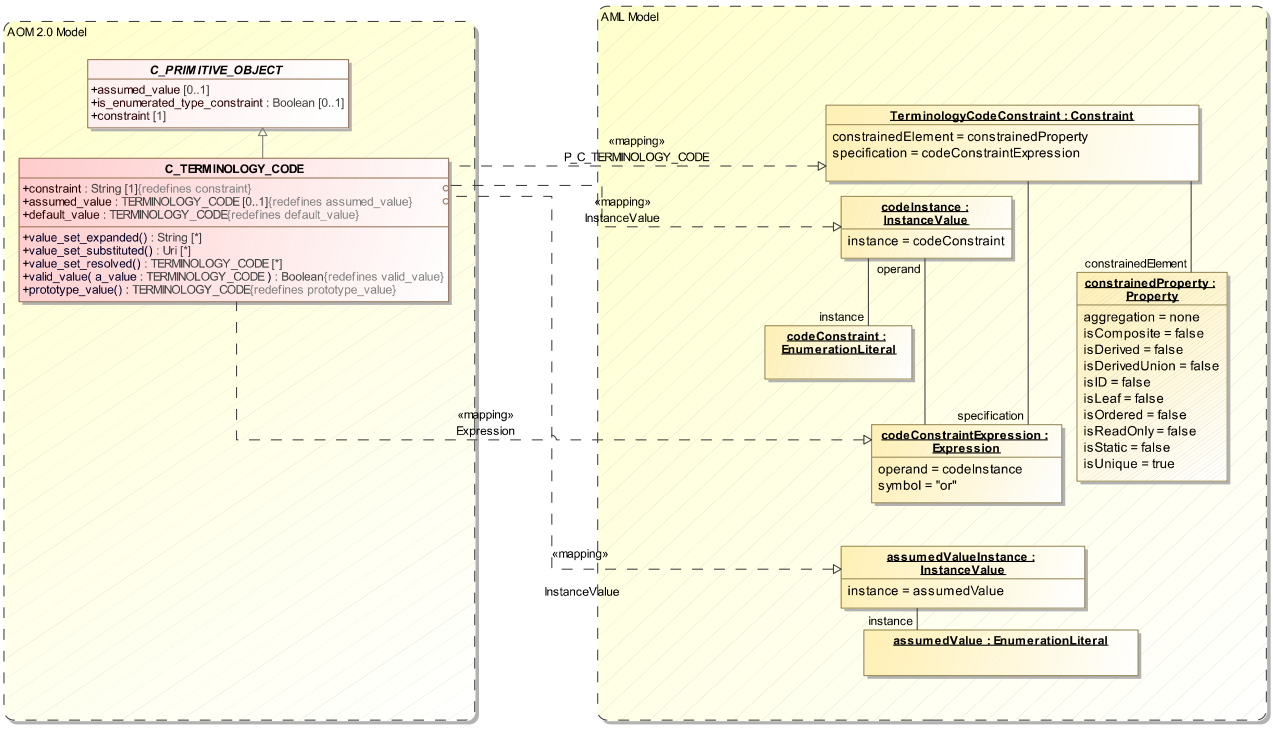

Figure 9.14 - Code Constraints Mapping Overview

9.12 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.

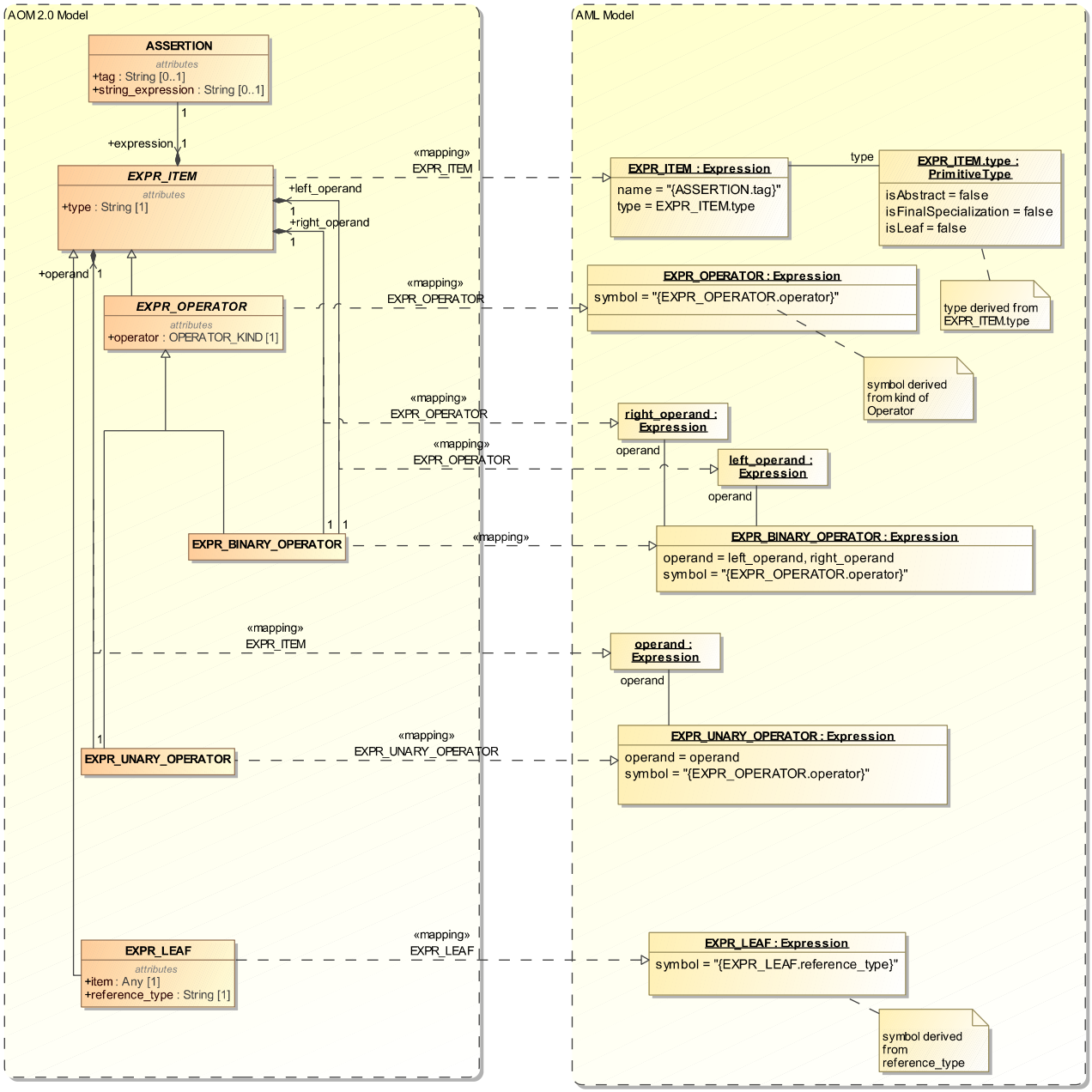

Figure 9.15 - Assertions Mapping Overview

This page intentionally left blank.