|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | 75 | Date: | $date.get(‘MMMM YYYY’) | |
| Archetype Modeling Language (AML)  #set ($thisversion = “1.0”) #set ($thisdoc = “Archetype Modeling Language (AML)”)  #set($documentNo = “health/2015-04-01”)  Version: $thisversion  **OMG Document Number: $documentNo**  **Standard document URL: http://www.omg.org/spec/AML/1.0**  Original File: N/A |

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

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

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

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

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

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

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

RESTRICTED RIGHTS LEGEND

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

TRADEMARKS

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

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

**OMG’s Issue Reporting Procedure**

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

Table of Contents

1 $bookmark.create($obj1.ID, $dp1) 28

2 $dp2 29

2.1 $bookmark.create($obj2.ID, $dp3) 29

2.2 $dp4 30

2.2.1 $bookmark.create($obj3.ID, $dp5) 30

2.2.2 $dp6 30

**Preface**

**OMG**

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.  
  
OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG™s specifications implement the Model Driven Architecture (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG™s specifications include: UML® (Unified Modeling Language); CORBA® (Common Object Request Broker Architecture); CWM (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.  
  
More information on the OMG is available at http://www.omg.org/.

**OMG Specifications**

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

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

Specifications within the Catalog are organized by the following categories:

**OMG Modeling Specifications**

• UML  
• MOF  
• XMI  
• CWM  
• Profile specifications

**OMG Middleware Specifications**

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

**Platform Specific Model and Interface Specifications**

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

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

**Typographical Conventions**

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

**Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.**

Courier - 10 pt. Bold: Programming language elements.

Helvetica/Arial - 10 pt.: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

# Resolution of Requirements – Mandatory Requirements

## AML Logical Profile

| **Section** | **Requirement** | **How Addressed** |
| --- | --- | --- |
| **6.5.1 AML Logical Profile** | Submissions shall specify an AML Logical Profile as defined below. The AML Profile shall be a set of UML stereotypes and properties which support the modeling of CIMI archetypes. The use of the AML Profile shall result in UML models that are free from dependency on any physical representation (such as XML Schema). In MDA terms, the AML Profile is a specification of the platform independent model (PIM). | The profile is implemented as a Platform Independent Model (PIM) that is described in clause 8. |
| **6.5.1.1 AML Sub-Profiles** | Submissions shall be comprised of a minimum of three sub-profiles: the Reference Model Profile, Constraint Model Profile, and the Terminology Binding Profile. Effectively, the AML Logical Profile is an aggregation of these sub profiles. | See sub clauses 8.2, 8.3, and 8.4 for the definitions of the three required profiles. The «import» relationship is used to define the profile package dependencies that eliminate redefinition of elements as described 8.1. |
| Elements defined in one sub profile should be reused in the other sub profiles; they should not be redefined |
| **6.5.1.2 AML Model Lifecycle Meta Data** | Submissions shall provide support for AML Lifecycle metadata on the model to address version management, profile naming, references, namespace, etcetera. At a minimum, submissions shall support the items below. Submitters shall designate which metadata items are mandatory.  ·       Identifier -- unique id of this model  ·       namespace -- reverse domain name of organization that published this model  ·       modelVersion – version of the archetype (i.e. model)  ·       profileVersion – version of the AML profile  ·       profileTool – tool and version of tool used to produce the model  ·       referenceModel -- name of reference model on which this model is based  ·       referenceModelVersion -- version of the reference model on which this model is based  ·       referenceModelPublisher -- name of organization publishing the reference model on which this model is based  ·       referenceModelPackageClosure -- name of the reference model package whose association closure defines the group for this model. It defines the package or packages that are considered the namespace used that may be reused and constrained, i.e. the name of an RM package whose class provides the set of classes that may be subsetted.  ·       referenceModelClass -- name of the class from the reference model which is the root class (Primogenitor) of this model  ·       lifecycleState --  with values like Initial, Draft, In Review, Approved, Published, Superseded, Obsolete  ·       isGenerated -- indicates whether this model was generated rather than being authored. This is used to determine whether or not the model can be overwritten by regeneration. | ·       Identifier -- Name of the Archetype (Mandatory)  ·       namespace -- ReferenceModel.rmNamespace (optional)  ·       modelVersion – AuthoredResource .release\_version (mandatory)  ·       profileVersion – Archetype.amlVersion (mandatory)  ·       profileTool – (Not supported - models can be managed by multiple tools, so this makes no sense)  ·       referenceModel -- the name of the ReferenceModel package imported by ArchetypeLibrary (Mandatory)  ·       referenceModelVersion -- ReferenceModel.rmVersion (optional)  ·       referenceModelPublisher -- ReferenceModel.rmPublisher (optional)  ·       referenceModelPackageClosure -- ArchetypeLibrary.rm\_package (mandatory)  ·       referenceModelClass -- the name of the "primogenitor" (root) class of the Constrains generalization for the root archetype (mandatory) ·       lifecycleState --  Lifecycle\_state enumeration, as recorded in AuthoredResource.lifecycle\_state (mandatory)  ·       isGenerated -- Archetype.is\_generated (mandatory -- default is False) |
| **6.5.1.3 AML Model Descriptive Meta Data** | Submissions shall include the concept of an ALM Model as part of the Profile. The profile shall allow for metadata tagging of ALM Models based upon the metadata items enumerated below, taken from ISO 13606.2. Deviations from this metadata set shall be substantiated in the submission.  ·       Description: Text  ·       originalAuthor: Hash <String, String> - example:  o   "name": "John Doe"  o   "organisation": "Beverly Hillbillies"  o   "email": "john.doe@gmail.com"  o   "date": "12/04/2011"  ·       contributors: List <String>  ·       purpose: Text  ·       use: Text  ·       misuse: Text  ·       keywords: List <Text>  ·       resources: List<Uri>otherDetails: Hash <String, String>  ·       copyright: Text | The reference to ALM model is a typo in the RFP, should have been AML.  The AuthoredResource «Stereotype» (See sub-clause 8.4.9) carries all of the required AML Model Descriptive Metadata. There are no deviations -- all of the attributes below are supported. |
| **6.5.1.4 Original Language Metadata Support** | Submissions shall provide the ability for AML model to be tagged to indicate the original natural language of expression and language translations. Example items such as the following subset are included for consideration. Submissions shall substantiate the metadata chosen.  ·       originalLanguage  ·       translations: including, for each language:  o   language  o   author  o   otherDetails (e.g. name, value pair)  o   accreditation | The ResourceTranslation «Stereotype» (See sub-clause 8.4.14) carries all of the translation details. The «Usage» relationship named "original\_language" identifies the original language and "translation" the translation. |
| **6.5.1.5 Archetype Object Model (AOM) 1.5** | Submissions shall provide functional coverage for the full set of capabilities or equivalent, as defined in AOM 1.5 [AOM]. | As noted elsewhere, this specification addresses the requirements found in AOM version 2, rather than version 1.5. Section 9 - AML-UML Transformation Reference formally defines the functional coverage between AOM 2.0 and AML. |
| **6.5.1.6 Language Translations** | Submissions shall allow each model element to have one or more translations recorded for each of its values (e.g. between international and realm-specific reference terms or between semantically equivalent terms in different coding systems). | The ResourceTranslation «Stereotype» (See sub-clause 8.4.14) supports the ability to assign language specific terms and descriptions as well as the ability to associate AML artifacts with codes that are defined in outside code systems. The AML specification does *not* support the ability to assert "semantic equivalence" between terms in different coding systems, as that is the function of a terminology server, not a modeling language. |

## Reference Profile Requirements

| **Section** | **Requirement** | **How Satisfied** |
| --- | --- | --- |
| **6.5.2.1 Model element references to reference model** | Submissions shall support the ability to model the relationships between archetype model elements (AME) and the reference model elements (RME) from which they are defined. | This is done through the Constrains «Stereotype» (see Sub-Clause 8.4.11) |
| **6.5.2.2 Model element references to progenitor model elements** | Submissions shall support the ability of each archetype model element (AME) to be associated with progenitor model elements.  Each AME should have the ability to be defined by reference to another AME. This enables an AME to be defined as a subset of an AME from a parent archetype model. It also allows archetype model instances to be associated on an element by element basis with its progenitor / primogenitor archetype model. | The UML generalization relationship as used in the AML Profile as the Constrains «Stereotype» is a specialization of Generalization and used to constrain model elements. |
| **6.5.2.3 Reference Model Primogenitor** | Submissions shall support the ability of each archetype model element to be directly or indirectly (through subsetting) extended from a single meta element, which contains all the tags/properties required by all archetype model elements.   Submissions shall substantiate properties chosen. | Genralization/Specialization in UML allows all attributes from progenitor model elements through to primogenetor ones to be extended. |
| **6.5.2.4 Primitive Types** | Submissions shall provide a set of primitive data types. Example items such as the following are included for consideration. Submissions shall substantiate the set of primitive types chosen.   Integer, Real, Boolean, Character, String, Date, Time, DateTime and Duration.  The set of primitive types selected shall be sufficient to support the definition of the CIMI Reference Model(http://informatics.mayo.edu/CIMI/index.php/Main\_Page). | Integer, Real, Boolean, String, Date, Time, DateTime and Duration are all derived from the corresponding XML Schema (XSD) data types. TerminologyCode can be derived from the ConceptReference «Stereotype» (See sub-clause 8.3.7)  A primitive data type for "Character" was not included, as it was determined during the revision of the openEHR ADL/AOM specifications to version 2 that "Character" was no longer required, since not included in the AOM2. The CIMI reference model included a character type, but it was never constrained. |

## Constraint Model Profile Requirements

| **Section** | **Requirement** | **How Satisfied** |
| --- | --- | --- |
| **6.5.3.1 Identification** | Submissions shall provide the ability to immutably and uniquely name model elements. | The identity of the Archetype itself is described in Section 8.4.4. The IdentifiedItem «Stereotype» allows unique identifiers (e.g. "id", "at" and "ac" codes) to be assigned to model elements. The UML name of the model element itself provides a second level of unique naming. |
| **6.5.3.2 Reference Model (RM) conformance** | Submissions shall provide the ability to define archetype models (AMs) as progeniture constrained subset models of a reference model. All elements in an AM are directly associated with the RM and their instances are a subset of the instances of the RM.  An archetype model conforms to and is subsumed by the reference model if and only if it does not violate any inheritance, containment, association relationship, multiplicity of any property, type, constraint, etc in the reference model. An archetype model SHALL NOT EXTEND the reference model (ie. Add new properties, classes, up cast types, or relax existing constraints). The archetype model may only narrow the reference model or another archetype model. | The combination of «Stereotype» Constrains, which prohibits the introduction of new ownedAttributes, combined with existing UML rules on generalization and specialization, meet all of these requirements. |
| **6.5.3.3 Model Specialization (Narrowing)** | Submissions shall provide the ability to reference and define specializations of previously defined archetype models (AMs).  It should be possible to create archetype models (progeniture) that are specializations of other archetype models, where the specialized (progeniture) model is more tightly constrained than its parent (progenitor) model. | The UML package «import» and generalization/specialization relationship, as applied in the AML Profile, support the reference and specialization of previously defined archetype models |
| **6.5.3.4 Differential representation of specialized models** | Submissions shall provide the ability for differential representation of archetype models (AMs). | The submitters assert that modeling the information required to support differential representation is supported by the AML, though the display of such differential representations should be considered a feature of tooling developed in accordance with this specification. |
|  | A specialized AM is represented differentially with respect to its parent AM, by only representing the differences between the specialized AM and its parent. This differential representation uses the same approach as a subtype class in UML which contains only the differences with respect to the ancestor class. This ensures maintainability. | As noted in the requirement, AML takes full advantage of the generalization relationship, which most UML tools show as the default representation. |
| **6.5.3.5 Model Flattening** | Submissions shall provide a flattening specification.   The profile supports the flattening of lineages of specialized AMs represented in differential form into the effective ‘flat form’ at any point in the specialization hierarchy. The flattened form should include all constraints that have been applied in the given AM and in any ancestor (progenitor) AM. The application of the flattening specification must always produce the same results. | The submitters assert that modeling the information required to support flattening is supported by the AML, and tools conformant with it should provide a mode supporting the display of all inherited specializations and attributes. |
| **6.5.3.6 Model Flattening Pedigree** | Submissions shall provide the ability to capture and maintain tractable lineage from flattened to progenitor AMs.   The profile shall support the ability to capture and maintain the lineage of specialization from each element of a specialized AM to the corresponding element of each ancestor (progenitor) AM. | In AML there is essentially no difference between a flattened and non-flattened models, because viewing a class's attributes provides the flattened perspective, while viewing the class's ownedAttributes provides the non-flattened perspective. Lineage can be determined by following the generalization relationship links and noting ownedAttributes for each class. |
| **6.5.3.7 Model Direct Associations** | Submissions shall provide the ability for an AM to be re-used by another AM by direct reference. | This is supported by the Package «import» mechanism in UML and therefore available in the AML Profile. |
| **6.5.3.8 Model Constrained Associations (‘Slots’)** | Submissions shall provide the ability for an AM to be composed from a set of other pre-existing AMs.   An AM can specify a ‘slot’ or entry point at which other AMs can be included. The valid set of AMs that can be included in a given slot can be specified in terms of constraints on the includable AMs based on their content/concept. | «Stereotype» ArchetypeRootProxy provides this capability. |
| **6.5.3.9 Model Constraints – Reference Model class subtype** | Submissions shall allow model elements within an AM to be constrained to a given Reference Model class subtype. | This capability is supported in the AML Profile through its use of the UML generalization in combination with the application of the subsets and redefines attribute rules in the UML. |
| **6.5.3.10 Attribute Value Constraints** | Submissions shall provide the ability in an AM to constrain RM attributes to a given type and to specify a valid value set.  Attributes of a primitive type (typically Integer, Real, Boolean, Character, String, Date, Time, DateTime and Duration) within the reference model can be constrained to specific value subsets in an AM, e.g. using ranges, specific values or by other means such as patterns.  For example, an attribute defined as a string value shall be able to be constrained to limit acceptable values, such as a two-character strings where the first is Alphabetic and the second is numeric from 1-5. | The primitive types all provide the ability to restrict permissible values to possible values, value ranges where appropriate, and type-specific match patterns where appropriate. |
| **6.5.3.11 Default and Assumed values** | Submissions shall provide the ability to provide default and assumed values for AM elements of both primitive and complex types. | The AssumedValue «Stereotype», which extends the UML Abstraction relationship provides assumed values for AML model elements. The native UML default is used to specify default values. |
| **6.5.3.12 Occurrences** | Submissions shall provide the ability to specify the allowable occurrence of instance data.  An AM can specify for any object node the allowed number of occurrences of data instances that conform to that node within a container node. Sibling objects defined within the same container nodes can each have occurrences defined for them. | The built in UML subset and redefinition mechanisms provide the ability to restrict the number of occurrences of instances and to validate conformance with the container node. |
| **6.5.3.13 Cardinality** | Submissions shall provide the ability to define cardinality that is subsumed by progenitor elements.  Attributes (i.e. properties) from the RM can have their cardinality constrained within an AM. | Use of the UML subsets mechanism in the AML Profile provides this capability. |

|  |  |  |
| --- | --- | --- |
| **Section** | **Requirement** | **How Satisfied** |
| **6.5.4.1 Semantic Binding of Classes** | Submissions shall provide the ability to bind an AM class to either a concept code from SNOMED CT (or other terminology), or to a Concept Domain, that represents the semantic meaning of the AM class. | The "about" tag of «Stereotype» ResourceReference provides this feature in the case of binding an AM class.   The concept domain is realized by the ValueSetReference and the meaning by a CodeSystemReference. |
| **6.5.4.2 Semantic Binding of Associations and Relationships** | Submissions shall provide the ability to bind an association between AM classes to a concept code from SNOMED CT (or other terminology), or to a Concept Domain, that represents the semantic meaning of the AM association | The about «Stereotype» in the Terminology Profile allows the association of any UML client with a ConceptReference supplier. The submitters argue that the CTS2 notion of "Concept Domain" -- an indirection that allows multiple value sets to be associated with the same data element -- is addressed through the ability to create purpose specific archetypes, each of which references a particular value set in the context of the archetype itself. |
| **6.5.4.3 Semantic Binding of Properties** | Submissions shall provide the ability to bind AM properties to a concept code from SNOMED CT (or other terminology), or to a Concept Domain, that represents the semantic meaning of the AM properties. | (same as response to 6.5.4.2) |
| **6.5.4.4 Value Binding of Classes and Properties** | Submissions shall provide the ability to bind AM classes and attributes to code systems, value sets and/or concept domains. This binding constrains the allowed values of the instances. | The AML Profile allows enumerations and derivatives, as well as the terminology code primitive type, to be bound to value sets. For code systems in this context, value sets are capable of representing all codes in the code system.  See: 6.4.5.2 above for discussions of Concept Domains. |
|  |
| Value bindings shall bind an AM class or attribute to a value set defined in an external terminology, to a code system, or to a concept domain. |
| **6.5.4.5 Usage Context Definition** | Submissions shall provide the ability to define usage context in the UML model, including at least an identifier and name. | The usage context of a given AML model can be described in the metadata elements for the Archetype and AuthoredResource «Stereotype»s |
|  |
| As an example, China or US. |
| **6.5.4.6 Concept Domain Definition** | Submissions shall provide the ability to define concept domains in the UML model, including at least an identifier and name. | See: 6.4.5.2 for discussion of Concept Domains. |
| **6.5.4.7 Code System Version Definition** | Submissions shall provide the ability to define metadata about a code system version in the UML model, including at least identifier, name, version, and source URL. | «Stereotype» CodeSystemReference and «Stereotype» CodeSystemVersionReference satisfy this requirement. |
| **6.5.4.8 Value Set Version Definition** | Submissions shall provide the ability to define metadata about a value set version in the UML model, including at least identifier, name, version, and source URL. | «Stereotype» ValueSetReference and «Stereotype» ValueSetDefinitionReference satisfy this requirement. |
| **6.5.4.9 Resolved Value Set Members** | Submissions shall provide the ability to define members within a value set version. | The specification allows the definition of UML enumeration, which could be perceived as local value sets. The definition of external value sets is viewed as outside the scope of the specification and is the role of an external terminology service such as CTS2. |
|  |
| Each member is defined by at least a code, concept name, and code system reference. |
| **6.5.4.10 Concept Domain Binding** | Submissions shall provide the ability to bind a concept domain with a resolved value set in a specified usage context. | This is related to the concept domain binding as discussed above. This requirement was actually determined to be unnecessary, as not required by ADL or CIMI. |
|  |
| For example, the use of “state codes” supported for a string value intended to represent geographic regions within a country. |
| **6.5.4.11 Concept Model Terminology Definition** | Submissions shall allow each concept model structure within an AM to be defined as a single terminology expression created by combining the values of each component data element within the structure.    In some cases, a single concept (e.g. diagnosis) can be represented either using a model structure (e.g. diagnosis name: ‘arthritis’, body site: ‘knee’, laterality: ‘left’), or as a pre or post-coordinated terminology expression (e.g. ‘arthritis of the left knee’). By associating a concept model structure with a terminology expression that defines how instances of the structural components could be combined together inside the terminology, instance data can be represented either way (i.e. either in structure or in terminology) in an isomorphic manner. | The submitters assert that the requirement describes a complex problem to which AML plays a key role, but the solution requires a model and grammar that crosses the terminology space, hence essentially out of scope for the AML, as driven by the requirements of ADL and CIMI. |
| **6.5.4.12 Terminology Constraint Bindings** | Submissions shall provide the ability to define constraints on an AM, using concepts and relationships defined in an external terminology. | The submitters assert that this requirement is actually out of scope for the AML specification, since the AML provides a representation of the semantics of the ADL in the UML, and the required capability is not a feature of the ADL (or CIMI). |

# Scope

## Archetype Modeling Language (AML) Background

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

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

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

Development of the AML specification was guided by:

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

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

## AML Intended Users

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

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

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

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

## AML Profiles

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

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

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

# Conformance

## Conformance Points

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

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

## AML Reference Model Profile

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

## AML Terminology Binding Profile

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

## AML Constraint Model Profile

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

# Normative References

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

[ADL] openEHR *Archetype Definition Language: 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>

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

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

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

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

[KIS] openEHR Knowledge Artefact Identification, Revision 0.7.5,   
[http://www.openehr.org/releases/trunk/architecture/am/knowledge\_id\_system.pdf](http://www.omg.org/spec/CTS2/1.1/)

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

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

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

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

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

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

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

# Terms and Definitions

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

Archetype

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

Archetype Definition Language (ADL)

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

Archetype Instance

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

Archetype Model (AM)

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

Archetype Object Model (AOM)

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

Archetype Query Language (AQL)

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

Clinical Data Repository (CDR)

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

Clinical Document Architecture (CDA)

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

Clinical Information Model (CIM)

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

Clinical Information Modeling Initiative (CIMI)

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

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

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

Clinical Model Governance

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

Clinical Model Repository

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

Clinical Model Verification

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

Clinical Modeling Language

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

Clinical Requirement

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

Code System

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

Common Terminology Services 2 (CTS2)

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

Concept

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

Concept Domain

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

Concept Domain Binding

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

Conceptual Information Model

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

Conformance

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

Constraint Model

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

Detailed Clinical Model

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

Fully Defined Concept

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

Information Model

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

Node

A Node is a named part of an information model.

Ontology

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

Reference Model

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

Reference Terminology

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

Semantic Binding

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

Terminology

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

Terminology Binding

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

Value Binding

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

Value Set

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

# Symbols

## Graphical Symbols

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

## Abbreviations

ADL Archetype Definition Language

AM Archetype Model

AML Archetype Modeling Language

AOM Archetype Object Model

AQL Archetype Query Language

CDA Clinical Document Architecture

CDL Clinical Document Language

CDR Clinical Data Repository

CIM Clinical Information Model

CIMI Clinical Information Modeling Initiative

CMP Constraint Model Profile

CRM Clinical Reference Model

CTS2 Common Terminology Services 2

EHR Electronic Health Record

HL7 Health Level Seven

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

LOINC Logical Observation Identifiers Names and Codes

MDA Model Driven Architecture

OCL Object Constraint Language

OMG Object Management Group

OpenEHR Open Electronic Health Record

PIM Platform Independent Model

PSM Platform Specific Model

RM Reference Model

RMP Reference Model Profile

SNOMED CT Systematized Nomenclature of Medicine – Clinical Terms

TBP Terminology Binding Profile

UML Unified Modeling Language

URI Uniform Resource Identifier

URM Underlying Reference Model

# Additional Information

## Changes to Adopted OMG Specifications

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

## Acknowledgements

The following companies submitted this specification:

1. Mayo Clinic
2. Visumpoint, LLC

The following companies supported this specification:

1. Escape Velocity, LLC

The following individuals aided the development of this specification:

|  |  |
| --- | --- |
| 1. Thomas Beale | Ocean Informatics |
| 1. Dave Carlson | Intermountain Healthcare (Consultant) |
| 1. Angelique Cortez | Accenture |
| 1. Bob Daniel | Escape Velocity LLC |
| 1. Tom Digre | Visumpoint LLC |
| 1. Stanley M. Huff, MD | Intermountain Healthcare |
| 1. Patrick Langford | Intermountain Healthcare (Consultant) |
| 1. Robert Lario | Visumpoint LLC |
| 1. Jay Lyle | Ockham Information Services LLC |
| 1. Virginia Riehl | Independent Consultant |
| 1. Deepak Kumar Sharma, M.S. | Mayo Clinic |
| 1. Harold Solbrig | Mayo Clinic |
| 1. Mason Tran 2. Bill Fredricks | Visumpoint LLC  Visumpoint LLC |
| 1. Michael van der Zel | University Medical Center Groningen  Results4Care |

# ADL, AOM and AML

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

## 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).

## Technical Aims of ADL / AOM

The ADL/AOM specifications published by openEHR [REF - <http://www.openehr.org/programs/specification/releases/currentbaseline#ADL2> ],, 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;
* Specialisation 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 [Knowledge Artefact Identification - <http://www.openehr.org/releases/trunk/architecture/am/knowledge_id_system.pdf> ].

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 - <https://openehr.atlassian.net/wiki/display/spec/Archetype+Query+Language+Description> ], 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 [REF – <http://www.openEHR.org/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 [http://www.clinicalelement.com/ ] 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.

## 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.

##

## Entry point for processing

#import('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')

#import("query", "com.nomagic.reportwizard.tools.QueryTool")

#import('text', 'com.nomagic.reportwizard.tools.TextTool')

#set($printedEnums = $array.createArray())

#set($printedInterfaces = $array.createArray())

#set($printedClasses = $array.createArray())

#set($printedDataTypes = $array.createArray())

#set($printedStereoTypes = $array.createArray())

#set($printedPrimitiveTypes = $array.createArray())

#set($elemList = $array.createArray())

## hard coded work around for lack of meta-data ☹

## NOTE: set $tmp since the add method of a list outputs true

#set($profileNames = $array.createArray())

#set($tmp = $profileNames.add(“Reference Model Profile”))

#set($tmp = $profileNames.add(“Terminology Profile”) )

#set($tmp = $profileNames.add(“Constraint Profile”))

#set($ignoreList = $array.createArray())

#set($tmp = $ignoreList.add(“Sample Data Binding”))

#set($tmp = $ignoreList.add(“MappedDate”))

#set($tmp = $ignoreList.add(“MappedTime”))

#set($level = 0)

#foreach ($pkg in $packageScope)

#packageList($pkg, 1)

#end

##foreach($elem in $elemList)

##$elem [ref-$elemList.indexOf($elem)]

##end

##

## MACRO writeText – output the HTML representation of $txt

#macro (writeText $txt)

#set($txt1 = $text.html($txt.replaceAll(“<<”, “«”).replaceAll(“>>”,”»”).replaceAll(“UML Standard Profile::UML2 Metamodel”,”UML”)))$txt1#end

## MACRO writeBookmark1 – write a numbered or unnumbered level 1 bookmark

#macro (writeBookmark1 $obj1 $dp1 $withNum)

#if($withNum == “true”)

# $bookmark.create($obj1.ID.substring($obj1.ID.indexOf(“ “)), $dp1)

#else

# $bookmark.create($obj1.ID.substring($obj1.ID.indexOf(“ “)), $dp1)

#end

#end

## MACRO writeHeader

#macro (writeHeader1 $dp2 $withNum)

#if($withNum == “true”)

# $dp2

#else

# $dp2

#end

#end

## MACRO writeBookmark2 --

#macro (writeBookmark2 $obj2 $dp3 $withNum)

#if($withNum == “true”)

## $bookmark.create($obj2.ID, $dp3)

#else

## $bookmark.create($obj2.ID, $dp3)

#end

#end

## MACRO writeHeader2 --

#macro (writeHeader2 $dp4 $withNum)

#if($withNum == “true”)

## $dp4

#else

## $dp4

#end

#end

#macro (writeBookmark3 $obj3 $dp5 $withNum)#if($withNum == “true”)

### $bookmark.create($obj3.ID, $dp5)

#else

### $bookmark.create($obj3.ID, $dp5)

#end#end#macro (writeHeader3 $dp6 $withNum)#if($withNum == “true”)

### $dp6

#else

### $dp6

#end#end#macro (writeBookmark4 $obj4 $dp7 $withNum)#if($withNum == “true”)

#### $bookmark.create($obj4.ID, $dp7)

#else

#### $bookmark.create($obj4.ID, $dp7)

#end#end#macro (writeHeader4 $dp8 $withNum)#if($withNum == “true”)

#### $dp8

#else

#### $dp8

#end#end#macro (writeBookmark5 $obj45 $dp75 $withNum)#if($withNum == “true”)

##### $bookmark.create($obj45.ID, $dp75)

#else

#### $bookmark.create($obj45.ID, $dp75)

#end#end#macro (writeHeader5 $dp55 $withNum)#if($withNum == “true”)

##### $dp55

#else

#### $dp55

#end#end#macro (writeBookmark $obj4 $dp7 $withNum)#if($withNum == “true”)

1. **$bookmark.create($obj4.ID, $dp7)**

#else

**$bookmark.create($obj4.ID, $dp7)**

#end#end#macro (writeHeader $dp8 $withNum)#if($withNum == “true”)

1. **$dp8**

#else

**$dp8**#end#end#macro (writeListItem $dp9)

* **$dp9**#end

##

## writeCode

#**macro** (writeCode $code)

$code.replaceAll(“<<”,”«”).replaceAll(“>>”,”»”)#end

##

## StripPrefix

#macro(stripPrefix $txt)$js.eval(‘e.replace(/[0-9\.]\s\*/, “”)’, ‘e’, $txt))#end

##

## MACRO printAttr

#**macro**(printAttr $att)

#set($vis = “~”)

#if($att.visibility == “public”)

#set($vis= “+”)

#elseif($att.visibility == “private”)

#set($vis = “-”)

#elseif($att.visibility == “protected”)

#set($vis = “#”)

#end

#set($mult = $att.multiplicity)

**•** $att.name : $att.type.qualifiedName.replaceAll(“UML Standard Profile::UML2 Metamodel”,"UML") #if($mult.length() > 0)[$mult]#end

#\*$att.visibility $att.name#if($att.type) : #if($js.eval(‘(typeQN.indexOf(“UML Standard Profile”) != -1)’, ‘typeQN’, $att.type.qualifiedName))$att.type.name#else$bookmark.open($att.type.ID, $att.type.name)#end #end#if($att.multiplicity != “”) [$att.multiplicity]#end#if($att.defaultValue) = $att.defaultValue.text#end\*#

#if($att.documentation != “”)

#writeText($att.documentation)

#end

#end

#\*

<property> ::= [<visibility>] [‘/’] <name> [‘:’ <prop-type>] [‘[‘ <multiplicity-range> ‘]’] [‘=’ <default>] [‘{‘ <prop-modifier > [‘,’ <prop-modifier >]\* ’}’]

\*#

##

## MACRO printOper

#**macro**(printOper $oper)

#set($paramLists = $oper.ownedParameter)

#set($size = 0)

#foreach($p in $paramLists)

#if($p.direction != “return”)

#set($size = $size +1)

#end

#end

#set($i = 1)

**• $oper.visibility $oper.name (#foreach($param in $paramLists)**

**#if($param.direction != “return”)$param.name**

**#if($param.type) : #if($js.eval(‘(typeQN.indexOf(“UML Standard Profile”) != -1)’, ‘typeQN’, $param.type.qualifiedName))$param.type.name#else $bookmark.open($param.type.ID, $param.type.name)#end#end#if($param.multiplicity != “”) [$param.multiplicity]#end#if($param.defaultValue) = $param.defaultValue.text#end#if($size != $i), #end#set($i = $i + 1)#end#end)#if($oper.type) : #if($js.eval(‘(typeQN.indexOf(“UML Standard Profile”) != -1)’, ‘typeQN’, $oper.type.qualifiedName))$oper.type.name#else $bookmark.open($oper.type.ID, $oper.type.name)#end#end#if($oper.hasTypeModifier() && $oper.typeModifier != “”)$oper.typeModifier#end**

#if($oper.documentation != “”)

#writeText($oper.documentation)

#end

#end

**##**

**## printAsso**

#**macro**(printAsso $attribute $association $object)

#foreach($member in $association.memberEnd)

#if($member.type != $object)

#set($memberEnd = $member)

#end

#end

**• $memberEnd.visibility#if($attribute.name != “”) $attribute.name#end#if($memberEnd.type) : #if($js.eval(‘(typeQN.indexOf(“UML Standard Profile”) != -1)’, ‘typeQN’,$memberEnd.type.qualifiedName))$memberEnd.type.name#else$bookmark.open($memberEnd.type.ID,$memberEnd.type.name)#end#end#if($memberEnd.multiplicity !=“”)[$memberEnd.multiplicity]#end#if($memberEnd.defaultValue) = $memberEnd.defaultValue.text#end**

#if($association.documentation)

#writeText($association.documentation)

#end

#if($memberEnd.documentation != “”)

#writeText($memberEnd.documentation)

#end

#end

**##**

**## writeHeading**

#macro (writeHeading $object $disp $isBookmark $headingLevel $headType $withNumbering)

#if($headType != “”)

#set($disp = “$disp [$headType]”)

#end

#if($profileNames.contains($disp))

#set($disp = $disp.replace(“ “, “”) + “ [Profile]”)

#end

#if($headingLevel == 1)

#if($isBookmark == “true”)

#writeBookmark1($object $disp $withNumbering)

#else

#writeHeader1($disp $withNumbering)

#end

#elseif($headingLevel == 2)

#if($isBookmark == “true”)

#writeBookmark2($object $disp $withNumbering)

#else

#writeHeader2($disp $withNumbering)

#end

#elseif($headingLevel == 3)

#if($isBookmark == “true”)

#writeBookmark3($object $disp $withNumbering)

#else

#writeHeader3($disp $withNumbering)

#end

#elseif($headingLevel == 4)

#if($isBookmark == “true”)

#writeBookmark4($object $disp $withNumbering)

#else

#writeHeader4($disp $withNumbering)

#end

#elseif($headingLevel == 5)

#if($isBookmark == “true”)

#writeBookmark5($object $disp $withNumbering)

#else

#writeHeader5($disp $withNumbering)

#end

#elseif($headingLevel == 6)

#if($isBookmark == “true”)

#writeBookmark($object $disp “false”)

#else

#writeHeader($disp “false”)

#end

#else

#if($isBookmark == “true”)

#writeBookmark($object $disp $withNumbering)

#else

#writeHeader($disp $withNumbering)

#end

#end

#end

**##**

**## findNestedElement**

#macro(findNestedElement $object)

#set($innerElement = $report.getInnerElement($object))

#foreach($nested in $innerElement)

#if($nested.elementType != “package” && $nested.elementType != “model” && $nested.elementType != “profile” )

#if($nested.elementType == “interface”)

#set($tmp = $nestedInterface.add($nested))

#elseif($nested.elementType == “class”)

#set($tmp = $nestedClass.add($nested))

#elseif($nested.elementType == “enumeration”)

#set($tmp = $nestedEnum.add($nested))

#elseif($nested.elementType == “datatype”)

#set($tmp = $nestedDataTypes.add($nested))

#elseif($nested.elementType == “stereotype”)

#set($tmp = $nestedStereoTypes.add($nested))

#elseif($nested.elementType == “primitivetype”)

#if($js.eval(‘(primtypename.indexOf(“AML”) != -1)’, ‘primtypename’, $nested.name))

#set($tmp = $nestedPrimitiveTypes.add($nested))

#end

#end

#set($in = $report.getInnerElement($nested))

#if($in.size() > 0)

#findNestedElement($nested)

#end

#end

#end

#end

**##**

#macro(updateElemList $obj)

#if(!$elemList.contains($obj))

#set($tmp = $elemList.add($obj.qualifiedName))

#end

#end

**##**

**## packageList - entry point**

#macro (packageList, $parentPackage, $plevel)

#set($packageInterface = $array.createArray())

#set($packageClass = $array.createArray())

#set($packageEnum = $array.createArray())

#set($packageDataTypes = $array.createArray())

#set($packageStereoTypes = $array.createArray())

#set($packagePrimitiveTypes = $array.createArray())

#set($nestedInterface = $array.createArray())

#set($nestedClass = $array.createArray())

#set($nestedEnum = $array.createArray())

#set($nestedDataTypes = $array.createArray())

#set($nestedStereoTypes = $array.createArray())

#set($nestedPrimitiveTypes = $array.createArray())

#if(($parentPackage.elementType ==“package”)||($parentPackage.elementType == “profile”))

#foreach($element in $parentPackage.importedMember)

#if($js.eval(‘(n.indexOf(“UML Standard Profile”) == -1)’, ‘n’, $element.qualifiedName))

##TYPE: $element.elementType

#if($element.elementType == “interface”)

##INTERFACE: $element.name

#set($tmp = $packageInterface.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

#elseif($element.elementType == “class”)

##CLASS: $element.name

#set($tmp = $packageClass.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

##INSIDE: #if($packageClass.size() > 0) TRUE #end

#elseif($element.elementType == “enumeration”)

##ENUM: $element.name

#set($tmp = $packageEnum.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

#elseif($element.elementType == “datatype”)

##DATATYPE: $element.name

#set($tmp = $packageDataTypes.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

#elseif($element.elementType == “primitivetype”)

##PRIMITIVE: $element.name

#if($js.eval(‘(primtypename.indexOf(“AML”) != -1)’, ‘primtypename’, $element.name))

#set($tmp = $packagePrimitiveTypes.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

#end

##INSIDE: #if($packagePrimitiveTypes.size() > 0) TRUE #end

#elseif($element.elementType == “stereotype”)

##STEREOTYPE: $element.name

#set($tmp = $packageStereoTypes.add($element))

#set($inner = $report.getInnerElement($element))

#if($inner.size() > 0)

#findNestedElement($element)

#end

##INSIDE: #if($packageStereoTypes.size() > 0) TRUE #end

#end

#end

#end

#end

#set($tmp = $array.addCollection($packageInterface, $nestedInterface))

#set($tmp = $array.addCollection($packageClass, $nestedClass))

#set($tmp = $array.addCollection($packageEnum, $nestedEnum))

#set($tmp = $array.addCollection($packageDataTypes, $nestedDataTypes))

#set($tmp = $array.addCollection($packageStereoTypes, $nestedStereoTypes))

#set($tmp = $array.addCollection($packagePrimitiveTypes, $nestedPrimitiveTypes))

#set($diagramList = $array.createArray())

#if($elemList.contains($parentPackage.qualifiedName))

#writeHeading($displayTitle, $displayTitle, “false”, $plevel, “”, “true”) [ref-$elemList.indexOf($parentPackage)]

#else

#foreach($d in $sorter.humanSort($parentPackage.ownedDiagram))

#if(($d.diagramType == “Class Diagram”)|| ($d.diagramType == “Profile Diagram”) ||($d.diagramType == “Package Diagram”))

#set($tmp = $diagramList.add($d))

#end

#end

#if($parentPackage != $project.model)

#set($displayTitle = $js.eval(‘e.replace(/[0-9\.]+\s\*/, “”)’, ‘e’, $parentPackage.name))

#writeHeading($displayTitle, $displayTitle, “false”, $plevel, “”, “true”)

#if($parentPackage.documentation != “”)

#writeText($parentPackage.documentation)

#end

#printDiagrams($diagramList $plevel)

#end

#set ($subPackages = $parentPackage.nestedPackage)

##

## Data Types

#set ($diagIndent = $plevel)

#if($packageDataTypes.size() > 0)

#foreach($dtp in $sorter.humanSort($packageDataTypes))

#createCommonContent ($dtp, “DataType” , $diagIndent)

#end

#end

##

## Interfaces

#if($packageInterface.size() > 0)

#foreach($interface in $sorter.humanSort($packageInterface))

#createCommonContent($interface, “Interface” , $diagIndent)

#end

#end

##

## classes

#if($packageClass.size() > 0)

#foreach($class in $sorter.humanSort($packageClass))

#createCommonContent($class, “Class” , $diagIndent)

#end

#end

##

## enumerations

#if($packageEnum.size() > 0)

#foreach($enum in $sorter.humanSort($packageEnum))

#createEnumerationContent ($enum , $diagIndent)

#end

#end

##

## primitive types

#if($packagePrimitiveTypes.size() > 0)

#foreach($ptp in $sorter.humanSort($packagePrimitiveTypes))

#createCommonContent ($ptp, “Primitive Type”, $diagIndent)

#end

#end

##

## stereotypes

#if($packageStereoTypes.size() > 0)

#foreach($stp in $sorter.humanSort($packageStereoTypes))

#createCommonContent ($stp, “Stereotype”, $diagIndent)

#end

#end

#end

##

#foreach ($pkg in $sorter.humanSort($subPackages))

#if ($plevel == 1)

#packageList($pkg, 2)

#elseif($plevel == 2)

#packageList($pkg, 3)

#elseif($plevel == 3)

#packageList($pkg, 4)

#else

#packageList($pkg, 5)

#end

#end

#end

##

##

##MACRO printDiagrams

#macro (printDiagrams $pkgdiagrams $diagIndent)

#if($pkgdiagrams)

#if($pkgdiagrams.size() > 0)

#foreach($diag in $sorter.humanSort($pkgdiagrams))$image.setWidth($diag.image, -2)

**$bookmark.create($diag.ID,** $js.eval(‘e.replace(/[0-9\.]+\s\*/, “”)’, ‘e’, $**diag.name))**

#if($diag.documentation != “”)

#writeText($diag.documentation)

#end

#end

#end

#end

#end

##

##MACRO createEnumerationContent

#macro(createEnumerationContent $enum $ind)

#set($indent = $ind + 1)

#writeHeading($enum, $enum.name, “true”, $indent, “Enumeration”, “true”)

#if($enum.documentation != “”)

#writeHeading($enum, “Description”, “false”, 7, “”, “false”)

#writeText($enum.documentation)

#end

#set($allDiagrams = $project.getDiagrams())

#set($pas = $enum.presentationElement)

#set($size = $pas.size())#if($pas.size() > 0)

#set($diagramList = $array.createArray())

#foreach($pa in $pas)

#set($entry = $pa.diagramPresentationElement.name)

#if(!$diagramList.contains($entry))

#set($tmp = $diagramList.add($entry))

#end

#end

#end

#if($diagramList.size() > 0)

#writeHeading($diagramList, “Diagrams”, “false”, 7, “”, “false”)

#foreach($diag1 in $diagramList)

#set($targetDiag = $report.findElementByName($allDiagrams, $diag1))

#if ($targetDiag.size() > 0)$bookmark.open($targetDiag.get(0).ID, $js.eval(‘e.replace(/[0-9\.]+\s\*/, “”)’, ‘e’, $diag1))#end#if($size != $velocityCount), #end

#end

#end

#set($enumLiterals = $enum.ownedLiteral)

#if($enumLiterals.size() > 0)

#writeHeading($implementInterface “Literals”, “false”, 7, “”, “false”)

#foreach($enumLit in $sorter.humanSort($enumLiterals))

#writeListItem($enumLit.name)

#if($enumLit.documentation != “”)

#writeText($enumLit.documentation)

#end

#end

#end

#end

##

## MACRO qualName

#macro(getQualName $qname)

#writeText($qname.replaceAll(“UML Standard Profile::UML2 Metamodel”, “UML”))

#end

##

## MACRO createCommonContent

#macro(createCommonContent $umlType $typeName $ind)

#set($indent = $ind + 1)

#if(!$elemList.contains($umlType.qualifiedName))

##[include a reference to ref-$elemList.indexOf($umlType.qualifiedName)]

##else

#updateElemList($umlType)

#set($title = $umlType.name)

#if($profileNames.contains($umlType.name))

#set($title = $umlType.name + “ [Profile]”)

#end

#writeHeading($umlType, $title, “true”, $indent, $typeName, “true”)

#if($umlType.documentation != “”)

#writeHeading($umlType, “Description”, “false”, 7, “”,“false”)

#writeText($umlType.documentation)

#end

#set($allDiagrams = $project.getDiagrams())

#set($pas = $umlType.presentationElement)

#set($size = $pas.size())

#if($pas.size() > 0)

#set($diagramList = $array.createArray())

#foreach($pa in $pas)

#set($entry =$pa.diagramPresentationElement.name)

#if(!$diagramList.contains($entry))

#set($tmp = $diagramList.add($entry))

#end

#end

#end

#if($diagramList.size() > 0)

#writeHeading($diagramList “Diagrams”, “false”, 7, “”, “false”)

#foreach($diag1 in $sorter.humanSort($diagramList))

#set($targetDiag = $report.findElementByName($allDiagrams, $diag1))

#if ($targetDiag.size() > 0 && !$ignoreList.contains($diag1))$bookmark.open($targetDiag.get(0).ID, $js.eval(‘e.replace(/[0-9\.]+\s\*/, “”)’, ‘e’, $diag1))#end#if($size != $velocityCount), #end#end

#end

#if (($typeName != “DataType”)&&($typeName != “Interface”)&&($typeName != “Primitive Type”))

#set($implementInterface = $umlType.realizedInterface)

#set($size = $implementInterface.size())

#if($implementInterface.size() > 0)

#writeHeading($implementInterface “Implemented Interface”, “false”, 7, “”, “false”)

#foreach($interface in $sorter.humanSort($implementInterface))$bookmark.open($interface.ID,$interface.name)#if($size != $velocityCount), #end#end#end

#end

#set($baseClassifier = $umlType.baseClassifier)

#set($size = $baseClassifier.size())

#if($baseClassifier.size() > 0)

#writeHeading($baseClassifier, “Direct Superclasses (Generalization)”, “false”, 7, “”, “false”)

#foreach($bclass in $sorter.humanSort($baseClassifier))

#if($js.eval(‘(primtypename1.indexOf(“UML Standard Profile”) == -1)’, ‘primtypename1’, $bclass.qualifiedName))$bookmark.open($bclass.ID, $bclass.qualifiedName)#else$bclass.qualifiedName #end#if($size !=$velocityCount), #end#end

#end

#set($metaclasses = $umlType.metaclass)

#set($size = $metaclasses.size())

#if($metaclasses.size() > 0)

#writeHeading($metaclasses, “Meta-classes”, “false”, 7, “”, “false”)

#foreach($mclass in $sorter.humanSort($metaclasses))

$mclass.qualifiedName.replaceAll(“UML Standard Profile::UML2 Metamodel”, “UML”)

#end

#end

#set($specClassifier = $umlType.specificClassifier)

#set($size = $specClassifier.size())

#if($specClassifier.size() > 0)

#writeHeading($specClassifier, “Direct Subclasses (Specialization)”, “false”, 7, “”, “false”)

#foreach($sclass in $sorter.humanSort($specClassifier))

#if(!$ignoreList.contains($sclass.name))#if($js.eval(‘(primtypename2.indexOf(“UML Standard Profile”) == -1)’, ‘primtypename2’, $sclass.qualifiedName))$bookmark.open($sclass.ID, $sclass.qualifiedName)#else $sclass.qualifiedName #end#if($size !=$velocityCount), #end#end#end

#end

#set($allAtt= $array.createArray())

#foreach($a in $umlType.ownedAttribute)

#if(!$a.association)

#set($tmp = $allAtt.add($a))

#end

#end

#if($allAtt.size() > 0)

#writeHeading($allAtt, “Attributes”, “false”, 7, “”, “false”)

#foreach($att in $allAtt)

#if(!$att.association)

#printAttr($att)

#end

#end

#end

#set($allOper = $umlType.ownedOperation)

#if($allOper.size() > 0)

#writeHeading($allOper, “Operations”, “false”, 7, “”, “false”)

#foreach($oper in $allOper)

#printOper($oper)

#end

#end

#set($associationLists = $array.createArray())

#foreach($attribute in $umlType.ownedAttribute)

#if($attribute.association)

#if($js.eval(‘(assocnm.indexOf(“base\_”) == -1)’, ‘assocnm’, $attribute.name))

#set($tmp=$associationLists.add($attribute.association))

#end

#end

#end

#if($associationLists.size() > 0)

#writeHeading($associationLists, “Associations”, “false”, 7, “”, “false”)

#foreach($attribute in $umlType.ownedAttribute)

#if($attribute.association)

#if($js.eval(‘(assocnm.indexOf(“base\_”) == -1)’, ‘assocnm’, $attribute.name))

#set($association = $attribute.association)

#printAsso($attribute, $association, $umlType)

#end

#end

#end

#end

#set($rules = $array.createArray())

#foreach($rule in $umlType.ownedRule)

#set($tmp = $rules.add($rule))

#end

#if($rules.size() > 0)

#writeHeading($rules, “Constraints”, “false”, 7, “”, “false”)

#foreach ($rulei in $sorter.humanSort($rules))

#writeListItem($rulei.name)

#if ($rulei.specification)

#if(($rulei.documentation)&&($rulei.documentation !=“”))

#writeText($rulei.documentation)#end#if($rulei.specification.text != “”)

#if($rulei.specification.language)

$js.eval(‘lname.replace(“OCL2\.0”, “OCL”)’, ‘lname’, $rulei.specification.language.toString())

#end

#writeCode($rulei.specification.text)

#end

#end

#end

#end

#end

#end

# AML-UML Transformation Reference (Informative)

## Introduction

This clause provides component, structural and abstract orientation to the transformations between the UML Profile for AML and the AOM 2.0 Meta-model, as specified in [AOM]. The transformations are expressed in terms of OMG QVT [QVT]. The QVT and related metamodels and profiles are provided as machine-readable artifacts associated with this specification. 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.

### 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 11‑1. 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 a «ArchetypeLibrary» Package.

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

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

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

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

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



Figure 11‑1 AML Provisioning Context

### QVT Packaging

The transformations referenced in this clause include:

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

Additionally, there are inherited common transformations:

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



Figure 11‑2 AML Transformations

### Transformation Reuse and Composition

Reuse and composition facilities are associated with QVT mapping operations. Disjunction enables selecting, among the set of disjunctive mappings, the first that satisfies the when clause and then invoking it. For the AML transformations, disjunction is used to identify a concrete MappingOperation to be selected from a given disjunctive MappingOperation. The disjunction hierarchy generally follows the AOM meta-model inheritance hierarchy and/or the UML meta-model inheritance hierarchy. Another reuse and composition facility associated with QVT mapping operations is inheritance. Inheritance enables reuse of the execution logic of an inherited mapping. Thus, disjunction is used to initially select a leaf mapping operation and inheritance is used to share common execution logic. For the AML transformations, inheritance is used to identify the hierarchy of execution logic required to populate target Elements from a source Element. The mapping inheritance generally follows the AOM meta-model inheritance hierarchy and/or the UML meta-model inheritance hierarchy. Figure 11‑3 illustrates the pattern of disjunction and inheritance used for the transformations.

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



Figure 11‑3 AML Transformation Disjunction and Inheritance

### Transformation Notation

Figure 11‑4 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 11‑4 AML Transformation Mapping Notation Overview

### Platform Binding

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

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

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

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

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

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

Determines whether the specified stereotype is applied to this element.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### Global Properties

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

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

## Archetype Library

The AML transformations are defined as a set of mappings between AOM Archetypes and AML-UML model elements. In general, there is a one-to-one correspondence between Elements in the AML-UML model and Elements in the AOM meta-model. At the highest compositional level defined within the AOM architecture, an archetype library is a container for a set of AUTHORED\_ARCHETYPEs. Figure 11‑5 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 11‑5 «ArchetypeLibrary» Mapping Overview

## Archetype

Figure 11‑6 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 11‑6 «Archetype» Mapping Overview

## Terminology Definition

Figure 11‑7 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 11‑7 Terminology Definition Mapping Overview

## Terminology Binding

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

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



Figure 11‑8 Terminology Binding Mapping Overview

## Local Value-Sets

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

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



Figure 11‑9 Local Value-Sets

## Archetype Definition

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

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



Figure 11‑10 Archetype Definition Mapping Overview

## Object References

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

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



Figure 11‑11 Object Reference Mapping

## Primitive Constraints

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

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



Figure 11‑12 Primitive Constraints

## Temporal Constraints

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



Figure 11‑13 Temporal Constraints Mapping Overview

## Code Constraints

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



Figure 11‑14 Code Constraint Mapping Overview

## Assertions

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

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



Figure 11‑15 Assertions Mapping Overview