|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | | 75 | Date: | $date.get(‘MMMM YYYY’) | |
| Archetype Modeling Language (AML)  #set ($thisversion = “1.0”) #set ($thisdoc = “Archetype Modeling Language (AML)”)  #set($documentNo = “health/2014-10-01”)  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) 5

2 $dp2 6

2.1 $bookmark.create($obj2.ID, $dp3) 6

2.2 $dp4 6

2.2.1 $bookmark.create($obj3.ID, $dp5) 6

2.2.2 $dp6 6

**Preface**

**OMG**

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

**OMG Specifications**

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

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

Specifications within the Catalog are organized by the following categories:

**OMG Modeling Specifications**

• UML  
• MOF  
• XMI  
• CWM  
• Profile specifications

**OMG Middleware Specifications**

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

**Platform Specific Model and Interface Specifications**

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

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

**Typographical Conventions**

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

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

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

Courier - 10 pt. Bold: Programming language elements.

Helvetica/Arial - 10 pt : Exceptions

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

# Scope

## Archetype Modeling Language (AML) Background

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

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

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

Development of the AML specification was guided by:

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

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

## AML Intended Users

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

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

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

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

## AML Profiles

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

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

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

# Conformance

## Conformance Points

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

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

## AML Reference Model Profile

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

## AML Terminology Binding Profile

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

## AML Constraint Model Profile

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

# Normative References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Terms and Definitions

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

Archetype

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

Archetype Definition Language (ADL)

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

Archetype Instance

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

Archetype Model (AM)

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

Archetype Object Model (AOM)

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

Archetype Query Language (AQL)

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

Clinical Data Repository (CDR)

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

Clinical Document Architecture (CDA)

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

Clinical Information Model (CIM)

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

Clinical Information Modeling Initiative (CIMI)

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

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

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

Clinical Model Governance

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

Clinical Model Repository

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

Clinical Model Verification

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

Clinical Modeling Language

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

Clinical Requirement

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

Code System

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

Common Terminology Services 2 (CTS2)

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

Concept

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

Concept Domain

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

Concept Domain Binding

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

Conceptual Information Model

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

Conformance

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

Constraint Model

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

Detailed Clinical Model

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

Fully Defined Concept

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

Information Model

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

Node

A Node is a named part of an information model.

Ontology

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

Reference Model

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

Reference Terminology

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

Semantic Binding

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

Terminology

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

Terminology Binding

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

Value Binding

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

Value Set

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

# Symbols

## Graphical Symbols

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

## Abbreviations

ADL Archetype Definition Language

AM Archetype Model

AML Archetype Modeling Language

AOM Archetype Object Model

AQL Archetype Query Language

CDA Clinical Document Architecture

CDL Clinical Document Language

CDR Clinical Data Repository

CIM Clinical Information Model

CIMI Clinical Information Modeling Initiative

CMP Constraint Model Profile

CRM Clinical Reference Model

CTS2 Common Terminology Services 2

EHR Electronic Health Record

HL7 Health Level Seven

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

LOINC Logical Observation Identifiers Names and Codes

MDA Model Driven Architecture

OCL Object Constraint Language

OMG Object Management Group

OpenEHR Open Electronic Health Record

PIM Platform Independent Model

PSM Platform Specific Model

RM Reference Model

RMP Reference Model Profile

SNOMED CT Systematized Nomenclature of Medicine – Clinical Terms

TBP Terminology Binding Profile

UML Unified Modeling Language

URI Uniform Resource Identifier

URM Underlying Reference Model

# Additional Information

## Changes to Adopted OMG Specifications

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

## Acknowledgements

The following companies submitted this specification:

1. Mayo Clinic
2. Visumpoint, LLC

The following companies supported this specification:

1. Escape Velocity, LLC

The following individuals aided the development of this specification:

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

# The AOM and the AML Metamodel

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

##

## 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($level = 0)

#foreach ($pkg in $packageScope)

#packageList($pkg, 1)

#end

##

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

#macro (writeText $txt)

#set($txt1 = $text.html($txt))$txt1#end

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

#macro (writeBookmark1 $obj1 $dp1 $withNum)

#if($withNum == “true”)

# $bookmark.create($obj1.ID, $dp1)

#else

# $bookmark.create($obj1.ID, $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#end

##

## printAttr

## MACRO printAttr

#**macro**(printAttr $att)

**• $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

##

## 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($disp = “<$headType> $disp”)

#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

**##**

**##**

**## 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))

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

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

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

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

#findNestedElement($element)

#end

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

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

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

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

#findNestedElement($element)

#end

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

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

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

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

#findNestedElement($element)

#end

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

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

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

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

#findNestedElement($element)

#end

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

#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

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

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

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

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

#findNestedElement($element)

#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())

#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(‘pkgname.replace(/[0-9\.]\s\*/gi, “”)’, ‘pkgname’, $parentPackage.name))

#if($js.eval(‘(pkgn.charAt(0) == “NEVER”)&&(pkgn.indexOf(“.”) != -1)’, ‘pkgn’, $parentPackage.name))

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

#else

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

#end

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

#writeText($parentPackage.documentation)

#end

#printDiagrams($diagramList $plevel)

#end

#set ($subPackages = $parentPackage.nestedPackage)

#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)

##

## Iterate across each diagram in the package printing the package followed

## by the various documentation

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

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

#if($diag.documentation != “”)

#writeText($diag.documentation)

#end

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

#set($diagramElements = $report.getDiagramElements($diag))

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

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

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

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

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

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

##

## Split all of the elements into interfaces, classes, datatypes,

## primitivetypes and enumerations, removing all of the elements from

## the surrounding package as they are printed

##foreach($element in $diagramElements)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

##end

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

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

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

##end

##end

##

## Process data types

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

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

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

#if(!$printedDataTypes.contains($dtp))

#set($tmp = $printedDataTypes.add($dtp))

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

#else

#set($tmp = $knownDataTypes.add($dtp))

#end

#end

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

#writeHeading($knownDataTypes, “Known other Data Types”, “false”, 7, “”, “false”)

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

#foreach($e in $knownDataTypes)$bookmark.open($e.ID, $e.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Process interfaces

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

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

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

#if(!$printedInterfaces.contains($interface))

#set($tmp = $printedInterfaces.add($interface))

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

#else

#set($tmp = $knownInterface.add($interface))

#end

#end

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

#writeHeading($knownInterface, “Known other interfaces” , “false”, 7, “”, “false”)

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

#foreach($i in $knownInterface)$bookmark.open($i.ID, $i.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Process classes

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

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

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

#if(!$printedClasses.contains($class))

#set($tmp = $printedClasses.add($class))

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

#else

#set($tmp = $knownClass.add($class))

#end

#end

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

#writeHeading($knownClass, “Known other classes” , “false”, 7, “”, “false”)

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

#foreach($c in $knownClass)$bookmark.open($c.ID,$c.name)#if($size != $velocityCount), #end#end

#end

#end

##

## Process enumerations

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

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

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

#if(!($printedEnums.contains($enum)))

#set ($tmp = $printedEnums.add($enum))

#createEnumerationContent($enum, $diagIndent)

#else

#set($tmp = $knownEnum.add($enum))

#end

#end

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

#writeHeading($knownEnum, “Known other enumerations” , “false”, 7, “”, “false”)

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

#foreach($e in $knownEnum)$bookmark.open($e.ID, $e.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Process stereotypes

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

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

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

#if(!$printedStereoTypes.contains($stp))

#set($tmp = $printedStereoTypes.add($stp))

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

#else

#set($tmp = $knownStereoTypes.add($stp))

#end

#end

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

#writeHeading($knownStereoTypes, “Known other Stereotypes” , “false”, 7, “”, “false”)

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

#foreach($e in $knownStereoTypes)$bookmark.open($e.ID,$e.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Process Primitive Types

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

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

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

#if(!$printedPrimitiveTypes.contains($ptp))

#set($tmp = $printedPrimitiveTypes.add($ptp))

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

#else

#set($tmp = $knownPrimitiveTypes.add($ptp))

#end

#end

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

#writeHeading($knownPrimitiveTypes, “Known other primitive types” , “false”, 7, “”, “false”)

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

#foreach($e in $knownPrimitiveTypes)

#if($js.eval(‘(primtypename.indexOf(“UML Standard Profile”) == -1)’, ‘primtypename’, $e.qualifiedName))$bookmark.open($e.ID,$e.name)

#else$e.name#end#if($size != $velocityCount), #end

#end

#end

#end

#end

##

## End of package diagrams print section

## Iterate over all of the package contents printing the various content types

##

## Data Types

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

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

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

#if(!$printedDataTypes.contains($dtp))

#set($tmp = $printedDataTypes.add($dtp))

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

#else

#set($tmp = $knownDataType.add($dtp))

#end

#end

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

#writeHeading($knownDataType, “Known other Data Types” , “false”, 7, “”, “false”)

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

#foreach($i in $knownDataType)$bookmark.open($i.ID,$i.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Interfaces

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

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

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

#if(!$printedInterfaces.contains($interface))

#set($tmp = $printedInterfaces.add($interface))

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

#else

#set($tmp = $knownInterface.add($interface))

#end

#end

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

#writeHeading($knownInterface, “Known other interfaces” , “false”, 7, “”, “false”)

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

#foreach($i in $knownInterface)$bookmark.open($i.ID, $i.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Classes

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

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

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

#if(!$printedClasses.contains($class))

#set($tmp = $printedClasses.add($class))

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

#else

#set($tmp = $knownClass.add($class))

#end

#end

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

#writeHeading($knownClass, “Known other classes” , “false”, 7, “”, “false”)

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

#foreach($c in $knownClass)$bookmark.open($c.ID, $c.name)#if($size != $velocityCount), #end

#end

#end

#end

##

## Enumerations

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

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

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

#if(!$printedEnums.contains($enum))

#set($tmp = $printedEnums.add($enum))

#createEnumerationContent($enum, $diagIndent)

#else

#set($tmp = $knownEnum.add($enum))

#end

#end

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

#writeHeading($knownEnum, “Known other enumerations” , “false”, 7, “”, “false”)

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

#foreach($e in $knownEnum)$bookmark.open($e.ID, $e.name)

#if($size != $velocityCount), #end

#end

#end

#end

##

## Primitive Types

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

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

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

#if(!$printedPrimitiveTypes.contains($ptp))

#set($tmp = $printedPrimitiveTypes.add($ptp))

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

#else

#set($tmp = $knownPrimitiveType.add($ptp))

#end

#end

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

#writeHeading($knownPrimitiveType, “Known other primitive types” , “false”, 7, “”, “false”)

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

#foreach($i in $knownPrimitiveType)

#if($js.eval(‘(primtypename.indexOf(“UML Standard Profile”) == -1)’, ‘primtypename’, $i.qualifiedName))$bookmark.open($i.ID,$i.name)#else$i.name#end

#if($size != $velocityCount), #end

#end

#end

#end

## ERROR - there should be more ends above!

##

## Stereotypes

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

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

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

#if(!$printedStereoTypes.contains($stp))

#set($tmp = $printedStereoTypes.add($stp))

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

#else

#set($tmp = $knownStereoType.add($stp))

#end

#end

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

#writeHeading($knownStereoType, “Known other Stereotypes” , “false”, 7, “”, “false”)

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

#foreach($i in $knownStereoType)$bookmark.open($i.ID,$i.name)#if($size != $velocityCount), #end

#end

#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(‘ename.replace(/[0-9\.]\s\*/gi, “”)’, ‘ename’, $diag1))#end

#if($size != $velocityCount), #end

#end

#end

#set($enumLiterals = $enum.ownedLiteral)

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

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

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

#writeListItem($enumLit.name)

#if($enumLit.documentation != “”)

#writeText($enumLit.documentation)

#end

#end

#end

#end

##

##

## MACRO createCommonContent

#macro(createCommonContent $umlType $typeName $ind)

#set($indent = $ind + 1)

#writeHeading($umlType, $umlType.name, “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)

$bookmark.open($targetDiag.get(0).ID, $js.eval(‘ename.replace(/[0-9\.]\s\*/gi, “”)’, ‘ename’, $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 Known 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.name)

#else

$bclass.name

#end

#if($size !=$velocityCount), #end

#end

#end

#set($specClassifier = $umlType.specificClassifier)

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

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

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

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

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

# Appendix A: AML MetaModel