**Date:** May 2016

Semantic Information Modeling for Federation (*SIMF*)

*Version 0.8*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

OMG Document Numbers

Manifest  ad/2016-05-01  
Submission document ad/2016-05-02 (this document)  
SIMF Conceptual model XMI ad/2016-05-03  
SIMF MOF Meta model XMI ad/2016-05-04  
SIMF Profile XMI ad/2016-05-05  
SIMF Profile mapping XMI ad/2016-05-06  
SIMF OWL Mapping XMI ad/2016-06-07  
SIMF Kernel Formalization ad/2016-05-08  
Non-normative artifacts (ZIP) ad/2016-05-09

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Copyright © 2016, Object Management Group, Inc.

Copyright © 2016, Data Access Technologies, Inc. (Model Driven Solutions Division)

Copyright © 2016, PNA-Group, Ltd.

Copyright © 2016, No Magic Inc.

Copyright © 2016, 88 Solutions, Inc.

Copyright © 2016, Thematix Partners 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.

The IPR mode for this submission is **Non-Assert**.

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 ATION 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 ATION, 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

[0 Submission-related material xiv](#_Toc451350101)

[0.1 Submission Introduction xiv](#_Toc451350102)

[0.2 Submission Team xiv](#_Toc451350103)

[0.2.1 Submitters xiv](#_Toc451350104)

[0.2.2 Contributors & Supporters xiv](#_Toc451350105)

[0.3 Proof of concept xiv](#_Toc451350106)

[0.4 Resolution of Requirements xiv](#_Toc451350107)

[0.4.1 Mandatory requirements xiv](#_Toc451350108)

[0.4.2 Non-mandatory features xviii](#_Toc451350109)

[0.5 Resolution of Discussion Issues xviii](#_Toc451350110)

[1 Scope 20](#_Toc451350111)

[1.1 Business Need 20](#_Toc451350112)

[1.2 Approach 22](#_Toc451350113)

[1.3 Unified Meta Model & Notation 24](#_Toc451350114)

[2 Conformance 24](#_Toc451350115)

[3 Normative References 25](#_Toc451350116)

[4 Terms and Definitions 25](#_Toc451350117)

[5 Introduction to SIMF Concepts 26](#_Toc451350118)

[*5.1* *SIMF Concept of concept (Non normative)* 26](#_Toc451350119)

[*5.2* *Pragmatic world view* 27](#_Toc451350120)

[*5.3* *Models* 28](#_Toc451350121)

[*5.4* *Concepts* 28](#_Toc451350122)

[5.4.1 Dictionary Concepts 30](#_Toc451350123)

[*5.5* *Facts* 31](#_Toc451350124)

[*5.6* *Context* 31](#_Toc451350125)

[*5.7* *Entities* 32](#_Toc451350126)

[*5.8* *Values and Anything* 32](#_Toc451350127)

[*5.9* *Situations* 32](#_Toc451350128)

[5.10 Relationships 32](#_Toc451350129)

[*5.11* *Types* 33](#_Toc451350130)

[5.11.1 Types and Instances 33](#_Toc451350131)

[5.11.2 Multiple inheritance and multiple classification 33](#_Toc451350132)

[5.11.3 Situation Types & Properties 33](#_Toc451350133)

[5.11.4 Facets 34](#_Toc451350134)

[5.11.5 Roles 34](#_Toc451350135)

[5.11.6 Phases 34](#_Toc451350136)

[5.12 Rules 35](#_Toc451350137)

[5.12.1 Constraints 35](#_Toc451350138)

[5.12.2 Mapping rules 35](#_Toc451350139)

[5.13 Expressions 35](#_Toc451350140)

[5.14 Characteristics, Quantity Kinds & Unit Types 35](#_Toc451350141)

[5.15 Terms and Naming 36](#_Toc451350142)

[5.16 SIMF Lexical Scope & Physical Representations 36](#_Toc451350143)

[5.17 Patterns 36](#_Toc451350144)

[5.18 Situation and pattern instances 36](#_Toc451350145)

[5.19 Record Structures and Structured Types 37](#_Toc451350146)

[5.20 Views and Viewpoints 37](#_Toc451350147)

[6 SIMF Conceptual Model & Abstract Syntax [Normative] 38](#_Toc451350148)

[6.1 SIMF Conceptual Model 39](#_Toc451350149)

[6.1.1 Diagram: SIMF Packages 39](#_Toc451350150)

[6.2 SIMF Conceptual Model::Expressions 40](#_Toc451350151)

[6.2.1 Diagram: Expressions 40](#_Toc451350152)

[6.2.2 Class Computed Fact 40](#_Toc451350153)

[6.2.3 Class Constant Reference 40](#_Toc451350154)

[6.2.4 Association Constant Value 41](#_Toc451350155)

[6.2.5 Class Equality 41](#_Toc451350156)

[6.2.6 Class Evaluation 41](#_Toc451350157)

[6.2.7 Association Expression Context 41](#_Toc451350158)

[6.2.8 Class Expression Context 41](#_Toc451350159)

[6.2.9 Class Expression Node 42](#_Toc451350160)

[6.2.10 Class Function Call 42](#_Toc451350161)

[6.2.11 Association Function Implementation 42](#_Toc451350162)

[6.2.12 Class Function Type 43](#_Toc451350163)

[6.2.13 Association Function Type 43](#_Toc451350164)

[6.2.14 Class Object Operation Type 43](#_Toc451350165)

[6.2.15 Association OO Target 43](#_Toc451350166)

[6.2.16 Association Return type 44](#_Toc451350167)

[6.2.17 Class Traversal 44](#_Toc451350168)

[6.2.18 Association traverse through 44](#_Toc451350169)

[6.3 SIMF Conceptual Model::Identifiers 46](#_Toc451350170)

[6.3.1 Diagram: Identifiers 46](#_Toc451350171)

[6.3.2 Association Identification 46](#_Toc451350172)

[6.3.3 Class Identifier 46](#_Toc451350173)

[6.3.4 Class IRI Identifier 47](#_Toc451350174)

[6.3.5 Association Namespace 47](#_Toc451350175)

[6.3.6 Class Term 47](#_Toc451350176)

[6.3.7 Class Text Identifier 47](#_Toc451350177)

[6.4 SIMF Conceptual Model::Kernel 48](#_Toc451350178)

[6.4.1 Diagram: SIMF Kernel Base 48](#_Toc451350179)

[6.4.2 Diagram: SIMF Kernel Rules 49](#_Toc451350180)

[6.4.3 Diagram: SIMF Kernel Values 50](#_Toc451350181)

[6.5 SIMF Conceptual Model::Lexical Scope 51](#_Toc451350182)

[6.5.1 Diagram: Lexical Scope 51](#_Toc451350183)

[6.5.2 Class Conceptual Package 51](#_Toc451350184)

[6.5.3 Association Definition 51](#_Toc451350185)

[6.5.4 Class Definition 52](#_Toc451350186)

[6.5.5 Association Documentation 52](#_Toc451350187)

[6.5.6 Class Include 52](#_Toc451350188)

[6.5.7 Class Lexical Reference 52](#_Toc451350189)

[6.5.8 Class Lexical Scope 53](#_Toc451350190)

[6.5.9 Class Logical Package 53](#_Toc451350191)

[6.5.10 Class Mapping Package 53](#_Toc451350192)

[6.5.11 Class Model 54](#_Toc451350193)

[6.5.12 Class Package 54](#_Toc451350194)

[6.5.13 Class Physical Package 54](#_Toc451350195)

[6.5.14 Association Prefix 54](#_Toc451350196)

[6.5.15 Class Prefix 54](#_Toc451350197)

[6.5.16 Association Scope 55](#_Toc451350198)

[6.5.17 Association Scope Reference 55](#_Toc451350199)

[6.5.18 Association Statement 55](#_Toc451350200)

[6.6 SIMF Conceptual Model::Mapping 56](#_Toc451350201)

[6.6.1 Diagram: Facades 56](#_Toc451350202)

[6.6.2 Diagram: Mapping Rules 57](#_Toc451350203)

[6.6.3 Class Computed Facade 57](#_Toc451350204)

[6.6.4 Class Facade 58](#_Toc451350205)

[6.6.5 Association from rue 58](#_Toc451350206)

[6.6.6 Class Map Rule 58](#_Toc451350207)

[6.6.7 Class Map Rule End 58](#_Toc451350208)

[6.6.8 Association Map Rule Type Assertion 59](#_Toc451350209)

[6.6.9 Association Mapped Property 59](#_Toc451350210)

[6.6.10 Association Mapped Relationship 59](#_Toc451350211)

[6.6.11 Class Mapping 60](#_Toc451350212)

[6.6.12 Class Property End 60](#_Toc451350213)

[6.6.13 Association Representation 60](#_Toc451350214)

[6.6.14 Class Representation Rule 60](#_Toc451350215)

[6.6.15 Association Represented Concept 61](#_Toc451350216)

[6.6.16 Association to rule 61](#_Toc451350217)

[6.6.17 Class Type End 61](#_Toc451350218)

[6.7 SIMF Conceptual Model::Patterns 63](#_Toc451350219)

[6.7.1 Diagram: Patterns 63](#_Toc451350220)

[6.7.2 Association Map rules 63](#_Toc451350221)

[6.7.3 Class Pattern 63](#_Toc451350222)

[6.7.4 Association Pattern Properties 64](#_Toc451350223)

[6.7.5 Class Pattern property 64](#_Toc451350224)

[6.7.6 Class Pattern Property Subset 65](#_Toc451350225)

[6.7.7 Class Pattern Relationship 65](#_Toc451350226)

[6.7.8 Association Pattern Relationships 66](#_Toc451350227)

[6.8 SIMF Conceptual Model::Records 68](#_Toc451350228)

[6.8.1 Diagram: Records 68](#_Toc451350229)

[6.8.2 Class Record 68](#_Toc451350230)

[6.8.3 Class Record Type 68](#_Toc451350231)

[6.9 SIMF Conceptual Model::Relationships 70](#_Toc451350232)

[6.9.1 Diagram: Annotations 70](#_Toc451350233)

[6.9.2 Diagram: Relationships 71](#_Toc451350234)

[6.9.3 Class Annotation Property 71](#_Toc451350235)

[6.9.4 Class Annotation Relationship Type 72](#_Toc451350236)

[6.9.5 Class Relationship 72](#_Toc451350237)

[6.9.6 Class Relationship Type 72](#_Toc451350238)

[6.10 SIMF Conceptual Model::Rules 73](#_Toc451350239)

[6.10.1 Diagram: Rules 73](#_Toc451350240)

[6.10.2 Class Conditional Constraint 73](#_Toc451350241)

[6.10.3 Class Covering Constraint 74](#_Toc451350242)

[6.10.4 Class Disjoint 74](#_Toc451350243)

[6.10.5 Class Enumerated 74](#_Toc451350244)

[6.10.6 Class Equivalent 74](#_Toc451350245)

[6.10.7 Association Generalizations 74](#_Toc451350246)

[6.10.8 Class Multiplicity Constraint 75](#_Toc451350247)

[6.10.9 Association Multiplicity Perspective 75](#_Toc451350248)

[6.10.10 Association Multiplicity Target 76](#_Toc451350249)

[6.10.11 Class Pattern Constraint 76](#_Toc451350250)

[6.10.12 Class Property Constraint 76](#_Toc451350251)

[6.10.13 Class Property Generalization Constraint 76](#_Toc451350252)

[6.10.14 Association Property Generalizations 77](#_Toc451350253)

[6.10.15 Association Property Specializations 77](#_Toc451350254)

[6.10.16 Class Property Transitivity Constraint 77](#_Toc451350255)

[6.10.17 Association Property Type 78](#_Toc451350256)

[6.10.18 Class Property Type Constraint 78](#_Toc451350257)

[6.10.19 Class Rule 78](#_Toc451350258)

[6.10.20 Association Rule constrains 79](#_Toc451350259)

[6.10.21 Association Specializations 79](#_Toc451350260)

[6.10.22 Class Type Constraint 79](#_Toc451350261)

[6.10.23 Class Type Generalization Constraint 79](#_Toc451350262)

[6.10.24 Class Type or Property 80](#_Toc451350263)

[6.10.25 Association Unique Set 80](#_Toc451350264)

[6.10.26 Class Uniqueness Constraint 80](#_Toc451350265)

[6.11 SIMF Conceptual Model::Structures 82](#_Toc451350266)

[6.11.1 Diagram: Structures 82](#_Toc451350267)

[6.11.2 Class Binding 82](#_Toc451350268)

[6.11.3 Association Bound in situation 83](#_Toc451350269)

[6.11.4 Association Bound individual 83](#_Toc451350270)

[6.11.5 Association Bound property 83](#_Toc451350271)

[6.11.6 Association Properties 83](#_Toc451350272)

[6.11.7 Class Property 84](#_Toc451350273)

[6.11.8 Class Situation 84](#_Toc451350274)

[6.11.9 Class Situation Type 85](#_Toc451350275)

[6.11.10 Class Structure 85](#_Toc451350276)

[6.11.11 Class Structured Type 85](#_Toc451350277)

[6.12 SIMF Conceptual Model::Top level 87](#_Toc451350278)

[6.12.1 Diagram: Context 87](#_Toc451350279)

[6.12.2 Diagram: TopLevel 87](#_Toc451350280)

[6.12.3 Class Anything 87](#_Toc451350281)

[6.12.4 Association Assertion 88](#_Toc451350282)

[6.12.5 Class Context 88](#_Toc451350283)

[6.12.6 Class Entity 88](#_Toc451350284)

[6.12.7 Association In Context 89](#_Toc451350285)

[6.12.8 Class Proposition 89](#_Toc451350286)

[6.12.9 Association Term Preference 89](#_Toc451350287)

[6.13 SIMF Conceptual Model::Types 91](#_Toc451350288)

[6.13.1 Diagram: Type-instance 91](#_Toc451350289)

[6.13.2 Diagram: Types 92](#_Toc451350290)

[6.13.3 Class Facet 92](#_Toc451350291)

[6.13.4 Class Intersection Type 93](#_Toc451350292)

[6.13.5 Class Phase 93](#_Toc451350293)

[6.13.6 Class Role 93](#_Toc451350294)

[6.13.7 Class Type 93](#_Toc451350295)

[6.13.8 Association Type Instance Relation 94](#_Toc451350296)

[6.13.9 Direct Supertypes 94](#_Toc451350297)

[6.13.10 Class Union Type 94](#_Toc451350298)

[6.14 SIMF Conceptual Model::Values 95](#_Toc451350299)

[6.14.1 Diagram: Values 96](#_Toc451350300)

[6.14.2 Class Base Unit Type 96](#_Toc451350301)

[6.14.3 Class Boolean 97](#_Toc451350302)

[6.14.4 Class Number 97](#_Toc451350303)

[6.14.5 Class Primitive Value 97](#_Toc451350304)

[6.14.6 Class Quantity kind 97](#_Toc451350305)

[6.14.7 Class Quantity Value 98](#_Toc451350306)

[6.14.8 Class System of Units 98](#_Toc451350307)

[6.14.9 Class Text 98](#_Toc451350308)

[6.14.10 Class Unit Type 98](#_Toc451350309)

[6.14.11 Class Value 99](#_Toc451350310)

[6.14.12 Class Value Type 100](#_Toc451350311)

[7 Foundational Assumptions (Normative) 100](#_Toc451350312)

[7.1 Multiple representations of overlapping concepts 100](#_Toc451350313)

[7.2 Models may include “ground facts” 100](#_Toc451350314)

[7.3 Conceptual Models 100](#_Toc451350315)

[7.4 Identity and identifiers 100](#_Toc451350316)

[7.5 Facts & propositions 101](#_Toc451350317)

[7.6 Representations of a concept 101](#_Toc451350318)

[7.6.1 Represents Relation 101](#_Toc451350319)

[7.7 Representation identifiers 101](#_Toc451350320)

[7.7.1 Example physical identifiers 101](#_Toc451350321)

[7.8 Sources 102](#_Toc451350322)

[7.9 Ownership 102](#_Toc451350323)

[7.10 Lifetime and context of facts 102](#_Toc451350324)

[8 Mapping to OWL 2 (normative) 103](#_Toc451350325)

[8.1 Class 103](#_Toc451350326)

[8.2 Class Generalization 103](#_Toc451350327)

[8.3 Class with Datatype Property 103](#_Toc451350328)

[8.4 Class with Self-Referential Object Property 104](#_Toc451350329)

[8.5 Class with Object Property 105](#_Toc451350330)

[8.6 Property Holder with Datatype Property 105](#_Toc451350331)

[8.7 Property Holder with Self-Referential Object Property 106](#_Toc451350332)

[8.8 Property Holder with Object Property 106](#_Toc451350333)

[8.9 Class with Object Property without Range 106](#_Toc451350334)

[8.10 Class with Subproperty 107](#_Toc451350335)

[8.11 Class with Universal Quantification Constraint on Property I 108](#_Toc451350336)

[8.12 Class with Universal Quantification Constraint on Property II 108](#_Toc451350337)

[8.13 Class with Existential Quantification Constraint on Property 109](#_Toc451350338)

[8.14 Property Holder with Self-Referential Subproperty 110](#_Toc451350339)

[8.15 Property Holder with Subproperty 110](#_Toc451350340)

[8.16 Class with Subproperty without a Range 111](#_Toc451350341)

[8.17 Class with Necessary and Sufficient Property 112](#_Toc451350342)

[8.18 Class With Property Having Unspecified Multiplicity 113](#_Toc451350343)

[9 Annex A: UML Conceptual Modeling Profile (normative) 114](#_Toc451350344)

[9.1 UML Conceptual Model Primer 114](#_Toc451350345)

[9.1.2 Specific kinds of classes 118](#_Toc451350346)

[9.1.3 Assertions about concepts 120](#_Toc451350347)

[9.1.4 Constraining properties and associations 121](#_Toc451350348)

[9.1.5 Tightening a property’s type 122](#_Toc451350349)

[9.1.6 Inferring a type from its properties 123](#_Toc451350350)

[9.2 Conceptual Modeling UML Profile 125](#_Toc451350351)

[9.2.1 UML Subset 125](#_Toc451350352)

[9.2.2 Stereotype: Classifies [Generalization, Realization] 126](#_Toc451350353)

[9.2.3 Stereotype: Conceptual Model [Package] 126](#_Toc451350354)

[9.2.4 Stereotype: Disjoint with [Dependency] 127](#_Toc451350355)

[9.2.5 Stereotype: Enumerates [Dependency] 127](#_Toc451350356)

[9.2.6 Stereotype: Equivalent with [Dependency] 128](#_Toc451350357)

[9.2.7 Stereotype: Intersection [Classifier] 128](#_Toc451350358)

[9.2.8 Stereotype: Is in context [Dependency] 129](#_Toc451350359)

[9.2.9 Stereotype: Phase [Class] 129](#_Toc451350360)

[9.2.10 Stereotype: Quantity Kind [Classifier] 129](#_Toc451350361)

[9.2.11 Stereotype: Represents [Dependency, Generalization] 130](#_Toc451350362)

[9.2.12 Stereotype: Role [Class] 131](#_Toc451350363)

[9.2.13 Stereotype: Source Reference [Element] 131](#_Toc451350364)

[9.2.14 Stereotype: Sufficient [Property] 131](#_Toc451350365)

[9.2.15 Stereotype: Union [Classifier] 132](#_Toc451350366)

[9.2.16 Stereotype: Unit [Classifier] and Base Unit [Classifier] 132](#_Toc451350367)

[9.2.17 Stereotype: Value 133](#_Toc451350368)

[9.2.18 Stereotype: Has value [Dependency] 134](#_Toc451350369)

[9.3 UML Profile – SIMF Rules & Model Mapping Primer 135](#_Toc451350370)

[9.3.1 Rule Model 135](#_Toc451350371)

[9.3.2 Representations 135](#_Toc451350372)

[9.3.3 Mapping Rules 136](#_Toc451350373)

[9.3.4 Representation traversals and patterns 137](#_Toc451350374)

[9.3.5 Representation Subsets 138](#_Toc451350375)

[9.3.6 <<Pattern Element>> computations and constraints 140](#_Toc451350376)

[9.3.7 <<Pattern Element>> strength 140](#_Toc451350377)

[9.3.8 <<Pattern Element>> quantifier 142](#_Toc451350378)

[9.3.9 <<Pattern Element>> explicit 142](#_Toc451350379)

[9.3.10 Facades and Representation Computations 143](#_Toc451350380)

[9.3.11 Property Chains 145](#_Toc451350381)

[9.4 SIMF Rules Profile Specification 147](#_Toc451350382)

[9.4.1 Class: ExistsRule 147](#_Toc451350383)

[9.4.2 Stereotype: Facade [Classifier] 148](#_Toc451350384)

[9.4.3 Stereotype: Filter [Property, Connector, ConnectorEnd] 149](#_Toc451350385)

[9.4.4 Stereotype: Map [Connector] 149](#_Toc451350386)

[9.4.5 Stereotype: Rule Specification [Package] 150](#_Toc451350387)

[9.4.6 Stereotype: Match [Connector, Property] 151](#_Toc451350388)

[9.4.7 Class: Rule 151](#_Toc451350389)

[9.4.8 Class: Rule Computation 152](#_Toc451350390)

[9.4.9 Stereotype: Representation Rule [Structured Classifier] 152](#_Toc451350391)

[9.4.10 Stereotype: Represents [Realization, Dependency] 153](#_Toc451350392)

[9.4.11 Stereotype: Subset of [Connector End] 154](#_Toc451350393)

[10 Profile mapping to SIMF Model (Normative) 156](#_Toc451350394)

[10.1 SIMFProfileToModelMapping::High level representation 156](#_Toc451350395)

[10.1.1 Diagram: Anything 156](#_Toc451350396)

[10.1.2 Diagram: Classes 156](#_Toc451350397)

[10.1.3 Diagram: Lexical Structure 157](#_Toc451350398)

[10.1.4 Diagram: Patterns 158](#_Toc451350399)

[10.1.5 Diagram: Relationships 159](#_Toc451350400)

[10.1.6 Diagram: Rules 160](#_Toc451350401)

[10.1.7 Diagram: Types 161](#_Toc451350402)

[10.1.8 Diagram: Values 162](#_Toc451350403)

[10.2 SIMFProfileToModelMapping::Mapping rules 163](#_Toc451350404)

[10.3 Class Annotation value mapping 163](#_Toc451350405)

[10.3.1 Attributes 163](#_Toc451350406)

[10.4 Class Association mapping 164](#_Toc451350407)

[10.4.1 Attributes 164](#_Toc451350408)

[10.5 Class Class mapping 165](#_Toc451350409)

[10.5.1 Attributes 165](#_Toc451350410)

[10.6 Class Class property mapping 166](#_Toc451350411)

[10.6.1 Attributes 166](#_Toc451350412)

[10.7 Class Containment mapping 167](#_Toc451350413)

[10.7.1 Attributes 167](#_Toc451350414)

[10.8 Class Enumeration mapping 168](#_Toc451350415)

[10.8.1 Attributes 168](#_Toc451350416)

[10.9 Class Equivalent property chain mapping 169](#_Toc451350417)

[10.9.1 Attributes 169](#_Toc451350418)

[10.10 Class Equivalent property mapping 170](#_Toc451350419)

[10.10.1 Attributes 170](#_Toc451350420)

[10.11 Class Equivalent with mapping 171](#_Toc451350421)

[10.11.1 Attributes 171](#_Toc451350422)

[10.12 Class Generalization mapping 171](#_Toc451350423)

[10.12.1 Attributes 172](#_Toc451350424)

[10.13 Class Generalization set covering mapping 173](#_Toc451350425)

[10.13.1 Attributes 173](#_Toc451350426)

[10.14 Class Generalization set disjoint mapping 174](#_Toc451350427)

[10.14.1 Attributes 174](#_Toc451350428)

[10.15 Class Is in context mapping 175](#_Toc451350429)

[10.15.1 Attributes 175](#_Toc451350430)

[10.16 Class Mapping rule mapping 176](#_Toc451350431)

[10.16.1 Attributes 176](#_Toc451350432)

[10.17 Class Named element Mapping 177](#_Toc451350433)

[10.17.1 Attributes 177](#_Toc451350434)

[10.18 Class Pattern property mapping 178](#_Toc451350435)

[10.18.1 Attributes 178](#_Toc451350436)

[10.19 Class Property hierarchy mapping 179](#_Toc451350437)

[10.19.1 Attributes 179](#_Toc451350438)

[10.20 Class Synonym mapping 180](#_Toc451350439)

[10.20.1 Attributes 180](#_Toc451350440)

[11 Concept Index 181](#_Toc451350441)

Preface

**OMG**

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

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

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

**OMG Specifications**

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

*http://www.omg.org/spec*

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

1. CORBA/IIOP
2. Data Distribution Services
3. Specialized CORBA

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

1. UML, MOF, CWM, XMI
2. UML Profile

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

1. CORBAServices
2. CORBAFacilities

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

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

OMG Headquarters  
109 Highland Ave,

Needham, MA 02494 USA  
  
Tel: +1-781-444-0404  
Fax: +1-781-444-0320  
Email: *pubs@omg.org*

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

**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.

**Issues**

The reader is encouraged to report any technical or editing issues/problems with this specification to <http://www.omg.org/report_issue.htm>.

# Submission-related material

## Submission Introduction

The SIMF submission team is pleased to present a revised submission to the “Semantic Information Modeling for Federation” Request for Proposal ad/2011-12-10

The IPR mode for this submission is **Non-Assert**.

Clause 0 of this document contains information specific to the OMG submission process and is not part of the proposed specification. The proposed specification starts with Clause 1. All clauses are normative unless otherwise specified.

## Submission Team

### Submitters

The following companies submitted this specification:

1. Data Access Technologies, Inc. (Model Driven Solutions Division)
2. Cory Casanave
3. Nomagic, Inc.
4. Jim Logan
5. PNA-Group, Ltd.
6. Sjir Nijssen
7. 88 Solutions
8. Manfred Koethe
9. Thematix Partners LLC
10. Elisa Kendall

### Contributors & Supporters

Contributors

* Tibco Software Inc.
  + Paul Brown

## Proof of concept

Nomagic has a released product implementing most of the SIMF profile and OWL mapping. Prototype efforts for mapping are expected but have not yet fully validated the model and mappings.

## Resolution of Requirements

### Mandatory requirements

|  |  |
| --- | --- |
| 6.5.1.1 Proposals shall define the SIMF Conceptual Model as a model of the concepts required to model information and achieve federation using SIMF. This model shall be a conceptual domain model of SIMF itself, expressed in the SIMF Notation (see requirement 6.5.2.2). | The conceptual models are specified in clause 8 using UML/SIMF notation expressed using the SIMF conceptual modeling profile. |
| 6.5.1.2 The SIMF Conceptual Model shall define the concepts necessary for creating conceptual domain models (CDMs), sufficiently general to express the semantics being represented by the information modeling constructs in the languages identified in requirement 6.5.3.1, including the following capabilities: | Due to the similarity of needs, the SIMF model is one language that may be used for different levels of abstraction. The level of abstraction and purpose of a specific model is specified. |
| 1. General capabilities for modeling all relevant aspects (i.e., all rules, laws, etc.) of concepts, including (but not necessarily limited to) the definition of: individual things, relationships, classification of individual things (including multiple classification), sub-classification and inheritance (including multiple inheritance), roles (that describe how individual things are involved in various processes, compositions and relationships), composition and constraints. | Capabilities relevant to information federation are included. Modeling of “all rules and laws” is considered out of our information federation scope. Concepts applicable to rules and laws may be modeled like any other concept.  All the following are included:   * Individuals * Relationships * Classification (of anything) * Multiple classification * Sub-classification (Generalization), including multiple inheritance. * Roles * Constraints   Specific models of compositions are intended to be included in SIMF models. |
| 1. Definition of one or more names by which users refer to a concept, as well as one or more *separate* reference identifiers that would normally be hidden from users. (This is required to maintain the stability of concept references across multiple languages, communities and viewpoints.) | Concepts may have any number of names and identifiers. Names and identifiers may be scoped by a context. |
| 1. Definition of the *context* of concepts, allowing for the grouping of concepts such that no single dominant decomposition is required (that is, in addition to just a hierarchical grouping, allow for a multi-dimensional separation of concerns [Ossher1999] delineated by multiple contexts). | Context is a first-class SIMF concept that relates a set of things to applicate assertions that hold for them. Something may be in any number of context across multiple contextual dimensions. |
| 1. Definition of *patterns* of reusable, parameterized conceptual structures and the use of such patterns within a context. | Patterns are a first-class concept. |
| 1. Definition of *units* that describe what can be measured about various conceptual *quantities* and asserting that some conceptual quantity is measured in specific units. | Units are represented using unity types bound to quantity kinds. |
| 1. Ability for *federated definition* of concepts; that is, allowance for the definition of a concept in a CDM such that it can be modified and/or extended across multiple contexts and models. | SIMF uses an open world assumption that may be closed in a specific context. As such definitions may be federated and extended. |
| 6.5.1.3 The SIMF Conceptual Model shall define the concepts necessary for creating logical information models (LIMs), capable of representing information context, information structures, integrity rules, derivation rules, views and viewpoints as may be found in the languages referenced in requirement 6.5.3.1, but not be bound to any particular data representation or schema language, including the following capabilities: | Due to the similarity of needs, the SIMF model is one language that may be used for different levels of abstraction. The level of abstraction and purpose of a specific model is specified. |
| 1. Usage of one or more terms and/or concepts defined in a CDM, as identified by MBRs between a LIM and the CDM, to define the semantics of information elements in one or more LIMs. | A CDM concept may be represented by any number of LIM concepts. |
| 1. Identification of concepts from a CDM (“what can be known about a subject domain”) as being required or optional in a LIM (“what may or must be included in a particular information structure”), with appropriate cardinalities. | A CDM specifies the semantics of the domain, not the data. Data cardinalities may be different from real-world cardinalities. What we can know, must know and do know may be independent. This is accomplished by using the “represents” relation and mappings. |
| 1. Ability for different LIMs related to the same CDM to represent different (and possibly incompatible) subsets of information about conceptually the same things (as semantic precision does not imply universal agreement). | See response to (b). |
| 1. Ability for a LIM to *close* the definition of a concept that has a federated definition in the related CDM, fixing it relative to a specific context in the CDM relevant to the LIM. (Once a definition is closed, it can then be assumed that no further statements will be made about that concept within the context relevant to a particular LIM thus allowing for the application of defaults and constraints impacting that concept.) | SIMF rules operate on a closed set of models based on their context whereas the models may be extended or refined in other context. Context is the foundation for closing the world. |
| 1. Ability to define *viewpoints* that specify *views* on a CDM or LIM that act as effective contexts for a particular purpose relevant to one or more other LIMs, including formation of views from composite concepts. | Each LIM is effectively a viewpoint that is mapped to the underlying CDM using MBRs. See 5.20. |
| 6.5.1.4 The SIMF Conceptual Model shall define the concepts necessary for creating model bridging relations (MBRs), sufficient to enable independently conceived models at all levels (CDM, LIM, PDS) to be federated, such that the similarities and differences between elements defined in each can be expressed, including the following capabilities: | Due to the similarity of needs, the SIMF model is one language that may be used for different levels of abstraction. The level of abstraction and purpose of a specific model is specified. |
| 1. Ability to relate identical and similar information concepts that have been independently conceived and represented in information models using the same or different information modeling languages or physical schema. | Identical and similar concepts are mapped using representation and mapping rules. The mapped models may or may not be independently conceived. |
| 1. Ability to handle differences in name, structure, representation, property sets and underlying semantic theories. | Mapping rules provide for differences in naming and structure. Mapping rules may be defined between compatible semantic theories. The interpretation of compatibility of theories is asserted in the mapping rules defined in SIMF models. |
| 1. Ability to relate the same information across views that share the same underlying concepts and to specify one view of a model from another (*projection*). | See the response (b). |
| 1. Ability to state the *purpose* for an information structure in one model relative to the related structure in another model. (Examples of purposes include creating, reading, updating and deleting recorded information and providing a snapshot in time, measurement, expected value or required value of a property of or association between information records.) | The purpose of a mapping may be specified in the textual documentation of a mapping; no other support is deemed as necessary. |
| 6.5.1.5 Proposals shall define a Kernel as a subset of the SIMF Conceptual Model with the minimum set of foundational concepts necessary in order to precisely define all other concepts within the SIMF Conceptual Model. Proposals shall provide a formal logic interpretation of the semantics of the SIMF Kernel, expressed in a formal logic such as Common Logic as defined in ISO standard 24707. | The kernel is defined as a subset of the SIMF model expressed in the diagrams under the package “Kernel”.  The kernel is defined as a mapping to the fUML subset of UML, which has a logical formulation. As the kernel is also specified in UML, no specific mapping is required.  Like MOF, fUML does not comprehend SIMF subsets and redefines restrictions, as such these restrictions are not enforced by the fUML kernel. |
| 6.5.2.1 Proposals shall define a SIMF Metamodel as a MOF or SMOF model of the abstract syntax of a modeling notation sufficient for completely defining any conceptual data model (CDM), logical information models (LIM) or model bridging relation (MBR). | A MOF meta model of SIMF is included. It is directly derived from the SIMF conceptual model by removing SIMF extensions not valid in MOF. MOF does not comprehend SIMF subsets and redefines restrictions, as such these restrictions are removed in the MOF model and must be enforced by other means. They remain restrictions on the model structure. |
| 6.5.2.2 Proposals shall define at least one graphical concrete and at least one textual concrete syntax for the SIMF Metamodel. The graphical notations shall be specified using the OMG diagram definition standard based on the abstract syntax. | The SIMF graphical notation included leverages UML and the SIMF profile for UML.  While various “fact modeling” textual notations have been evaluated in creating SIMF, no text notation is included at this time.  It is anticipated that other notations will be defined against the SIMF model in later efforts. |
| 6.5.2.3 To the greatest extent practical, the SIMF Metamodel and notations shall be based on reuse or adaptation of existing modeling and logic languages. Proposals shall provide justification when this is not considered to be the best solution. | The SIMF graphical notation utilizes UML. In keeping with the philosophy of SIMF, the relationship to other models is expressed as mappings. |
| 6.5.2.4 The content of models expressed using the SIMF Metamodel shall be Web addressable resources, each having a unique Web identity in support of Linked Open Data. | As a MOF meta model SIMF models are web addressable. The OWL/RDF mapping of SIMF also produces web addressable artifacts. |
| 6.5.2.5 Proposals shall provide an MBR model bridging from the SIMF Conceptual Model to the SIMF Metamodel, specifying how CDMs, LIMs and MBRs based on concepts defined in the SIMF Conceptual Model may be represented using the SIMF Metamodel and so expressed in SIMF notations. Conversely, all statements made as part of any model represented using the SIMF Metamodel shall have a precise and well-defined semantic mapping to the SIMF Conceptual Model. | The SIMF meta model is a minor transformation from the SIMF conceptual model using the same semantics, terms, constructs and model identity. For this reason, no mapping was deemed as required. |
| Proposals shall define normative MBR models, in the SIMF Language, that bridge the SIMF Conceptual Model to metamodels for the following existing languages, in order support the federation of information defined in these languages.   1. Entity-relationship (ER) modeling, with a metamodel such as that proposed for IMM 2. SQL Data Definition Language (DDL), with a metamodel such as that proposed for IMM 3. XML schema definitions (XSDs), with a metamodel such as that proposed for IMM 4. Unified Modeling Language (UML) 5. Semantics of Business Vocabularies and Rules (SBVR) 6. OWL web ontology language, with the metamodel as given in ODM 7. RDF Schema (RDF/S), with the metamodel as given in ODM | Mappings are specified for UML and OWL. Additional mappings may be included as user demand indicates. Experience indicates that mappings should be independent of the foundation specification such that they can be developed and maintained independently. This helps to avoid monolithic specifications. |
| 6.5.3.2 Proposals shall provide a minimum of four non-normative examples drawn from different domains, demonstrating the overall applicability of the proposed SIMF Language to the definition, extension, validation, federation and integration of information models and their physical schema representations. | Extensive examples are provided in OMG submissions based on SIMF. These include “threat/risk” (OMG document SYSA/2016-0-02 ) and draft versions of FIBO. Other multiple other small examples are included in this document.  Numerous examples are provided in this specification. |

### Non-mandatory features

|  |  |
| --- | --- |
| 6.6.1 Proposals may provide a direct mapping from the SIMF Metamodel to RDF, RDF/S and/or OWL, as an exchange format beyond that provided by XMI based on the SIMF Metamodel abstract syntax. | A mapping to OWL-2 is included. |
| 6.6.2 UML Profile for SIMF  6.6.2.1 Proposals may define a profile of UML that represents all or part of SIMF using UML stereotypes, tagged values and OCL constraints.  6.6.2.2 If a UML Profile is included, an MBR shall be defined between the profile and the SIMF Metamodel.  6.6.2.3 If a UML Profile is included, proposals shall describe the fidelity of the profile and any information loss between the profile and corresponding models expressed in SIMF notation. | A UML profile for SIMF (at all levels) is included and defines the graphical syntax for SIMF. Other notations may be added in the future. |

## Resolution of Discussion Issues

|  |
| --- |
| References to and naming of individuals. |

All SIMF entities may have multiple names and identifiers. This includes individuals as well as types and metadata. Each term is a first-class entity that may be defined in a context independent of the original definition. Context may also be used to cope human languages.

# Scope

## Business Need

Our ability to share, manage, analyze, communicate and act upon information is at the foundation of the modern enterprise and open, collaborative government. Information sharing is essential for an integrated approach to enterprise supply chains, fighting terrorism, business and government intelligence, inter-organizational collaboration and integrating enterprise applications. Yet, this essential capability has remained difficult and expensive to achieve in information systems which are frequently isolated, stove piped, and difficult to integrate. The inability of our systems to share information hampers the ability of our organizations to collaborate and for our processes, services, and information resources to work together. Much of our information technology budgets are consumed by attempts to overcome this “semantic friction” in our systems and organizations are currently spending more on application integration than on building new applications [Gartner2011]. The overall human and financial cost to society from our failure to share and reuse information is many times the cost of the systems’ operation and maintenance.

In general, information sharing can be understood at a number of different levels.

* *Infrastructure* is the technology used to maintain data and move it from one place to another.
* *Format* is the way data are structured, its syntax.
* *Semantics* deals with how data is interpreted as meaningful information. For an information system, this interpretation is reflected in how the data is processed in order to carry out the business purpose of the system.

We are effective at dealing with data infrastructures today, and we are somewhat effective at handling multiple data formats, albeit via manual and point-to-point integrations. However, we are not very good at understanding how the semantics of data in independent data sources are related. Too often, how each system interprets shared data is implicit in the specific design and operation of the systems. Differences in structure, terminology, viewpoint, and notations make system-specific data structures hard to integrate, negatively impacting the capability to federate these systems or the information they contain.

Full semantic integration requires information systems to all properly and consistently interpret the data exchanged among the systems. This, in turn, requires that there be an explicit understanding of what the desired semantic interpretation *is* at a business level. A semantic *model* can be used to express this understanding in a way that can be validated by the business stakeholders of the systems being integrated. And, given a computational underpinning for such a model, it can then also be used for supporting analyses and deductions necessary to carry out the necessary integration.

Unfortunately, for most existing information systems, the desired semantics have not been effectively modeled. The following are some scenarios in which semantic integration is, nevertheless, critical. Diverse and disparate efforts are currently being made to address these scenarios, examples of which are included with the scenario descriptions below. But, as of today, there is no consistent way to address modeling for semantic integration in general across all these areas.

* *Data integration between business systems.* Many large businesses have a critical need to better integrate systems in support of complex products. Not only may their business area have suffered financial distress, but there may be a need for new government reporting or new analytics and integration due to acquisitions. Such organizations typically have multiple layers of existing data bases, middleware specifications and XML schemas for use in web services, event brokers, etc. Most, if not all, of the existing systems and technologies still need to be supported. There may be dozens or even hundreds of enterprise systems involved and hundreds or thousands of small applications and spreadsheets.

*Example.* A common approach chosen for integrating major business systems is to create a “canonical model” of the domain and then map data into and out of that model using data mapping tools. Unfortunately, while there are various proprietary tools to support such an effort, there is no widely available standard-based tooling for the job. For instance, while UML can be and is used for the modeling part of the job, a general modeling notation such as UML is not for the conceptual level of modeling required, and there is currently no standard profile to adapt it to the task nor for mapping data into and out of a canonical model in general. (The Model Driven Message Interoperability specification provides some support for the latter, but only limited to message format transformation for the financial services domain.)

* *Data federation across multi-disciplinary teams.* Developing complex systems often involves many parties who are widely distributed in location and time. Such development therefore requires efficient and effective information exchange during the complete development and operations lifecycle of the system. This can only be achieved by realizing semantic integration between all involved parties.

*Example.* The European Cooperation for Space Standardization (ECSS)[[1]](#footnote-1) addresses this issue by introducing the concept of *a global conceptual model.* This model is used in the implementation of “space system data repositories” as federations of physical databases. These databases are geographically dispersed and change over time but are logically integrated in an interoperable architecture, so that data can be exchanged effectively and reliably. Such data repositories need to be stable over a long period of time, so modeling must be at the semantic level *independent* of technology and tools. This modeling allows for upgrading the implementation technology without changing the model and data itself. The primary aim of this is to substantially reduce the system development and operation costs while achieving greater precision and federation.

* *Information federation across an industry.* Major industries, such as finance and telecommunications, need to deal with the representation of information relative to multiple contexts, taking into account different business processes, specific modeling goals and needs, visualization and implementation requirements or the existence of overlapping modeling domains. These differing contexts and conditions may require emphasizing different aspects and characteristics of essentially the same information. The representation of a concept in one view may be different from the representation of the same concept in another view as the context-specific details that are relevant differ from view to view. Information can be described using different yet compatible paradigms (e.g., domain-specific languages vs. UML and profiles) yet the meaning and semantics of the information should stay the same regardless of the format and viewpoint. This, again, highlights the need to focus on a common core model of shared semantic concepts.

*Examples.* Some examples of efforts to deal with industry-level information federation are the Shared Information and Data (SID) Model, developed by the TM Forum [TMForum], the Common Information Model (CIM) developed by the Distributed Management Task Force [DMTF] and the Reference Information Model (RIM)developed by Health Level Seven [HL7].

* *Information sharing and federation of threat and risk information,* Threats and risks are increasingly multi-dimensional in nature – spanning physical space and cyber space. Threat actors understand and exploit our stove piped approach to sharing and analyzing information which leads to ineffective collaboration and mediation. Only by federating information across multiple domains such as cyber, physical, critical infrastructure, criminal, intelligence and defense, irrespective of technical and political boundaries, can we effectively counter multi-dimensional intentional threats, natural events and system failures.

*Examples.* Attacks on our critical infrastructure have and will combine cyber attacks with physical attacks. This has been seen in exploits of our electric power grid where physical weaknesses are combined with Cyber to harm our physical infrastructure. By combining Cyber, criminal and terrorist information we will be better able to deal with these critical threats.

* *Data federation across government organizations.* Information sharing has been recognized by governments as a key enabler for purposes as diverse as fighting terrorism to financial transactions. There has been some progress in standardizing exchange schemas, which is a big step ahead of no standards at all, but the need exists to ensure that there is no ambiguity in the semantics of the exchanged data in order to safely enable the reuse of that data. In addition, any such standard must accept that there are and will be other such standards and that these also need to be federated.

*Example.* The U.S. Information Sharing Environment (ISE) “provides analysts, operators, and investigators with integrated and synthesized terrorism, weapons of mass destruction, and homeland security information needed to enhance national security and help keep our people safe” [ISE]. ISE depends on fixed schemas for information sharing, i.e., the National Information Exchange Model (NIEM) and the Universal Core (UCORE). These schemas provide XML Schema definitions that are claimed to be sufficiently common and universally understood by relevant stakeholders regardless of the IT systems being used within their intended domains. Even within NIEM, though, hundreds of overlapping schemas have been defined.

* *Model federation across different modeling metamodels.* The OMG itself has multiple standards related to modeling. These standards were originally created independently, resulting in difficulties when users try to use them together to share information embodied in models using the different standards. A conceptual model abstracting from the existing OMG modeling standards, would facilitate their comparison, acknowledging the commonality (or lack thereof) between the different concepts and definitions and bridging those concepts.

*Example.* OMG specifications related to just process modeling include BPMN, UML Activities, BPDM, and SPEM. A case in point in the difficulty this has caused relates to the *UML Profile for DODAF and MoDAF* (UPDM), a wide ranging profile supporting US Department of Defense (DOD) and UK Ministry of Defence (MOD) architecture frameworks. The UPDM community wishes, for example, to be able to use BPMN process models in the context of their UML Profile. A stopgap tactic has been to define an additional *UML Profile for BPMN*, which allows BPMN-looking diagrams to be drawn in UML, but it is clear this is not a strategic approach. A better approach would be to create a “process modeling” conceptual domain model that would then permit model bridging relations between BPMN, UML, BPDM and SPEM models, allowing sharing across users’ process models

* *Schema Evolution*. As information systems evolve to support changing enterprise needs, the datasets they use need to evolve as well. While some changes are additive and readily accommodated, others involve factoring and evolving concepts. At their core, such changes require the evolution of the dataset schema underlying the system and the migration of the data from the old to new schemas. Such changes also impact the logic that interacts with the dataset and every external interface and related data structure. While there is some tooling available for schema migration, there is little available to aid in the evolution of the logic and external interfaces. The absence of semantic understanding of the relationship between the schema and external interface data structures makes tooling to aid in the evolution problematic.

*Example.* It is common for an enterprise to represent the concept of *customer* as a composite of information about the person and the role that person plays with respect to the enterprise. Evolving needs, including regulatory requirements, require many enterprises to now factor this concept so that they can represent that the same person may play other roles as well, such as employee. Such semantic understanding is required to enforce constraints such as a prohibition against the same individual playing both the customer and employee role in a transaction. The absence of semantics-based tooling makes such changes labor intensive and error prone.

Current standards for information and data modeling may be effective at defining a particular data model for a particular application using a particular technology to solve a particular problem. But, as highlighted by the above examples, the methodology for using these standards at a higher level of abstraction – namely for cross-domain and cross-organizational semantic modeling – is not as well or as widely understood. As a consequence, the models available within a given organizational context are often not well suited to application across multiple dimensions or technologies, and so poorly support the needs for sharing and federation.

## Approach

As described in Section 6.2.1, the scope of SIMF encompasses conceptual domain modeling, logical information modeling and the modeling of bridging relations between models at all levels. Figure 1 summarizes the general organization of information models into conceptual, logical and physical layers and indicates the scope of SIMF within it. The following more precisely defines the terminology used to describe this scope.

* *Conceptual Domain Model (CDM).* A CDM is not a traditional data model, as such, but, rather, a model of the terms and concepts of an area of concern or *domain[[2]](#footnote-2),* which may be a broad industry area such as telecommunications, finance or even metamodeling or may be more focused on a specific organization or application area. From a formal point of view, a CDM is a model of a real or possible world. It primarily addresses the semantics, concepts and terminology of a domain, capturing the meaning that usually is not available or not specific in a data model, while abstracting out data representation, viewpoint and application specific considerations. The objective of a CDM is to capture the semantics of one or more domains as a well-defined set of (potentially federated) concepts, predicates (to express properties about the concepts and to relate them) and integrity rules (constraining instances). For a given domain, many CDMs may co-exist, e.g. CDMs that have been developed by different entities and express differing conceptual frameworks.

For the purpose of this RFP, conceptual domain modeling is limited to modeling the information concerns of a domain as a conceptual domain model. Modeling processes and services is considered out of scope. However, this does not imply that all modeling of the dynamics of concepts is necessarily out of scope. (Future RFPs may address further process-oriented conceptual modeling requirements.)

* *Logical Information Model (LIM).* A LIM acts as an intermediary between CDMs and physical data schema (see below). The objective of a LIM is to provide a purpose-specific but implementation technology-independent view of information in terms of logical data structures. There can be multiple different ways to represent the same information from different viewpoints and for specific purposes. Each viewpoint may have its own structure, local vocabulary and subset of all possible information in a domain. These purpose specific commitments are made in the LIM. Elements of a LIM are related to the CDM concepts, predicates and integrity rules they represent (using Model Bridging Relations, see below) and may extend or embed other logical elements. A LIM model addresses a specific viewpoint and purpose and, as such, selects those types, properties and relations of interest and structures them for that purpose. I LIM may impose more structure that CDM in that for a particular purpose users want to look at information in a particular way – with specific organization and hierarchies. Many “enterprise data models” fall into the LIM category.
* *Physical Data Schema (PDS).* A PDS describes how to implement a LIM in a database or exchange format of choice. That is, it defines the application- and technology-specific representations of data. There can be many PDS representations of the same LIM. PDSs grounded in LIMs (using Model Bridging Relations, see below) provide the basis for federation of data defined in those schemas. Such fixed schemas become a particular *projection* of information for a particular purpose, but not the only way to access the same information. Most PDS models are imported from existing schema for mapping or generated from a LIM based on production rules using MDA.
* *Model Bridging Relation (MBR).* An MBR defines a connection between different sets of elements in the same or different models. This connection may be between models across the conceptual, logical and physical modeling layers, or within models of a given layer. MBR connections allow for different representations and organizations of the same things in different ways. MBRs may bridge different models created as part of a single wider effort or they may address connections between independently conceived models. Linking the semantics of information in its different conceptual, logical and physical representations, MBRs are the foundation of federation.

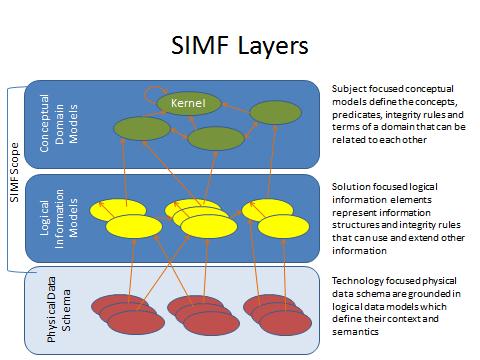


Figure Information Modeling Layers and SIMF Modeling Scope

As indicated in Figure 1, the scope of SIMF includes CDMs, LIMs and MBRs (where the lines in the diagram represent MBRs). While PDSs are out of scope for SIMF, model bridging relations to PDSs are part of SIMF.

## Unified Meta Model & Notation

While SIMF supports modeling at multiple levels, there is a single meta model for SIMF. A model defines the abstraction layer it is intended to model. Likewise, the same syntax is used at all levels. While there is a unified meta model, some model elements are more appropriate for one layer or another – which is defined in section 2.

# Conformance

The Conformance clause identifies which clauses of the specification are mandatory (or conditionally mandatory) and which are optional in order for an implementation to claim conformance to the specification.

Note: For conditionally mandatory clauses, the conditions must, of course, be specified.

There are five distinct types of conformance. These are listed below. Unless otherwise stated these types of conformance are independent.

1. *Abstract syntax conformance.* A tool demonstrating abstract syntax conformance provides a user interface and/or API that enables instances of concrete SIMF metaclasses to be created, read, updated, and deleted. The tool must also provide a way to validate the well-formedness of models that corresponds to the constraints defined in the SIMF metamodel. Abstract syntax may be further refined as either:
   1. Conceptual model conformance – corresponding to all elements not included in the packages “Rules” & “Mapping rules”.
   2. Rule syntax conformance – corresponding all conceptual model packages.
2. *UML Profile conformance.* A tool demonstrating UML Profile conformance provides a user interface and/or API that enables instances of SIMF UML notation to be created, read, updated, and deleted. Note that a conforming tool may provide the ability to create, read, update and delete additional diagrams and notational elements that are not defined in SIMF. UML Profile conformance may be further refined as either:
   1. Conceptual Modeling Profile Conformance – All elements defined for the conceptual modeling profile, clause 10.2
   2. SIMF Rules profile conformance - All elements defined for the SIMF Rules profile, clause 10.4.

3. *Model interchange conformance.* A tool demonstrating model interchange conformance can import and export conformant XMI for all valid SIMF models, including models with profiles defined and/or applied. Model interchange conformance implies abstract syntax conformance. A conforming SIMF tool shall be able to load and save XMI as a SIMF MOF meta model.

5. *Semantic conformance.* A tool demonstrating semantic conformance provides a demonstrable way to interpret SIMF semantics, e.g., data transformers, code generation, model execution, or semantic model analysis.

Where the SIMF specification provides options for a conforming tool, these are explicitly stated in the specification. In a number of other cases, certain aspects of the semantics are listed as "undefined" or “intentionally not specified” or “not specified”, allowing for domain- or application-specific customizations. Only customizations that do not contradict the provisions of this specification will be deemed to conform to it. However, models whose meaning is based on such customizations can only be interchanged without loss with tools that support the same or compatible customizations.

This specification comprises this document together with XMI serialization contained in machine-consumable files as listed on the cover page. If there are any conflicts between this document and the machine-consumable files, the machine-consumable files take precedence.

# Normative References

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

List of normative references. (specific reference to be included)

|  |  |
| --- | --- |
| [UML] | OMG Unified Modeling Language (UML) Superstructure v2.5  <http://www.omg.org/spec/UML/2.5/> |
| [MOF] | OMG Specification ptc/2013-08-20, Meta Object Facility (MOF) Core, v2.5 |
| [ODM] | ODM Ontology Denition Metamodel, 2 September 2014. <http://www.omg.org/spec/ODM/1.1/> |
| [OWL-2] | W3C/TR REC-owl2-syntax:2009 OWL 2 Web Ontology Language: Structural Specication and Functional-Style Syntax. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/ |
| [NIEM] | <http://reference.niem.gov/> |
| [WGS-84] | <http://earth-info.nga.mil/GandG/wgs84/> |
| [JCGM 200:2008] | <http://www.iso.org/sites/JCGM/VIM/JCGM_200e_FILES/MAIN_JCGM_200e/01_e.html> |
| [NIST-SI] | <http://physics.nist.gov/cuu/pdf/sp811.pdf> |
| [NIST-800] | <http://csrc.nist.gov/publications/PubsSPs.html> |
| [BFO] | <http://ifomis.uni-saarland.de/bfo/> |
| [MathWorld] | From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/FibonacciNumber.html> |

To provide for informal definitions of common concepts various non-normative sources have been used. This included Wikipedia.com accessed under their creative commons license.

# Terms and Definitions

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

Terms defined in other sections

* See section 1.2 for definitions of:
  + *Conceptual Domain Model (CDM).*
  + *Logical Information Model (LIM).*
  + *Physical Data Schema (PDS).*
  + *Model Bridging Relation (MBR).*
* All terms defined in the model, clause 6, are defined in SIMF.

Other Terms

* **Instance**: An “instance” designates something categorized by a type (including meta types). For example; “Fido is an instance of Dog” means that Fido <has type> Dog and that Dog <categorizes> Fido. Note: Instance does not imply (or prevent) any implementation or other restrictions such as “Factories” or “Single Classification” as do some programming languages.
* **Fac**t: Facts are something that someone or something asserts to be true. The class of things that can be asserted are called “propositions” as they can be true or false. Once asserted these propositions are facts. Of course the relevance, trust or belief in facts is open to interpretation.
* **Concept**: Everything we describe in a SIMF model is considered a *concept*. A concept is anything conceived. For something to be in a model there must be a conception of it. Concepts are inclusive of types, categories, values and individuals.

# Introduction to SIMF Concepts

The following is a high-level, non-normative, description of some of the fundamental SIMF concepts.

The fundamental concepts will be described in a way that most practitioners can relate it to their familiar experiences. In this chapter we will gradually build a semantic-conceptual architecture (an architecture that is completely independent of any particular technology and in which there is a clear distinction between the world of the things and the world of the names (representations)).

The prime aim of this chapter is to demonstrate the value that a SIMF semantic-conceptual model and transformations can offer to the ever increasing need of federation of (information) systems in business and government practice.

## *SIMF Concept of concept (Non normative)*

The SIMF meta model can be used to model and integrate information from multiple source at different levels of abstraction. A key element of SIMF is conceptual models as described above – models a real or possible world. To provide for mapping models of data must also be expressed.

A conceptual model (CDM) includes these real world concepts. Data models include concepts of data about these things. Thus the term “concept” in this document can refer to anything that can be referenced in a SIMF model, which is anything. Note that in the SIMF model the “top level”, the class that all others inherit from is called “anything”. There is no class “Concept” in SIMF. The term concept simply means “something we conceive of and model”. To model something, we must have a conception of it, so to be more precise – we express our conception (concept) of something in a model.

The conceptual domain model (CDM) is a model where we express our conception of a real or possible world

There are other times when the topic is the model its self or the components of the model. In this respect a SIMF model is just data. When referencing the SIMF representation of a concept the term “SIMF Element” will be used. A SIMF element is the data representation of a concept in a model repository.

## *Pragmatic world view*

Any conceptual system has strengths and weaknesses and is colored by the influences, people and domains which influenced the conceptualizations. From a pragmatic point of view there is a good understanding of what a “Tree” is (the word sense of tree that refers to a plant most babies could identify). However, when you look very close and at the “edges” such categorizations can be fuzzy. On a similar note when you get to the edges of quantum mechanics and near-light speed travel, our common views of the world can break down, or become unknown or controversial. Those who study language carefully may say that every meaning of every word is contextual and that dictionary and thesaurus don’t completely work.

While all of the above may be true, taken to the extreme it would make communication impossible and synthesizing or transforming information impractical. On the other hand, if you take a pragmatic world view there is a lot that can be done to communicate, even across those with different backgrounds, and most of us just don’t deal with quantum mechanics and “warp speed”. SIMF and the conceptual models built on it use a pragmatic world view and are designed to deal with structured (or at least semi-structured) data. However, such pragmatics are modularized into component models such that those who need to deal with extreme detail in some area may do so without breaking the entire system.

If the “precision dial” were turned all the way up to account for all these edge cases the model would get complex and unusable. Most people understand things based on what the “mostly are”, not the edge cases.

The other extreme is to make definitions and their representation very narrow and specific, perhaps to a system or particular discipline. This other extreme may also have such “fuzzy” definitions that they are no more than hints. This extreme also has its problems and would make wide-scale communications and synthesis impossible. In fact, such overly specific data representations and fuzzy schema are essentially what we are federating with SIMF – the design of specific systems.

So between overly complex and overly specific or fuzzy is the useful middle ground we seek. That middle ground is to be precise and true within this pragmatic world view.

SIMF does not restrict a model to being very loose or very precise, but in general we expect:

* Terms mean what they do in a dictionary definition, refined to a word sense
* Where a concept is more specific it will be scoped by appropriate context (e.g. an expert’s quantification [Keslick, John A. (2004). ["Tree Biology Dictionary"](http://www.treedictionary.com/). Retrieved 2012-07-30.] of what defines a tree may be more specific than common concepts)
* Concepts will have the most general definition and relationships reasonable within the context in which they are defined (note that this sometimes introduces general abstract concepts). Note also that this is different from common practices in defining schema where the same essential meaning is repeated everywhere it is used.
* We are treating the world in terms of Newtonian mechanics (e.g. space/time is not relative). Where applicable, we are concerned with the surface of the earth and the near vicinity (e.g. in understanding location and gravity). However, such assumptions should reference an earth context and off-earth models are possible.
* Generic concepts are better under-defined (having fewer axioms than may be possible) than over-specified. This allows more specific concepts or “theories of the world” to specialize and reuse the generic ones.
* The concepts defined and mapped are being used in some structured way, not unstructured natural language (this does not preclude the structuring of natural language by automated or “human in the loop” means, but this is out of our scope). We do expect that natural language processes may interpret text to define SIMF axioms.

SIMF does provide semantics to support the evaluation of context, scope and trust so domains “on the edges” may still use SIMF, but should be careful about what SIMF features or SIMF models may say about common concepts.

In terms of the spectrum of formal logics and formal languages, SIMF is in the middle. It is specifically designed and intended for the role of semantic mediation or real (and probably messy) data. This is different from most of the “ontology languages” that are designed for general inference and “decidability” of 100% valid and consistent formal models. The priority for SIMF conceptual models is that they capture the real domain semantics, even if those semantics may be inconvenient for some inference algorithms. As such, the SIMF logic does not fit into some of the common categories; it is not a “description logic”, nor is it necessarily a “frame logic” or “first order”. In particular, SIMF relationships are “first class”, that is statements can say things about other statements like “Jim doesn’t believe joe likes sue”. Each of these relationships is its own thing, not part of any “frame” or “object”. SIMF includes a specific concept of context to scope statements. Also, you can also create new kinds of kinds – “metatypes” in modeling land. For these reasons a SIMF conceptual model will probably not be “first order”, but will use higher order logic.

Those wishing to use specific inferencing capabilities, such as those provided by OWL-DL, may use techniques to simulate higher order logic of first-order logics or ignore higher order statements. The SIMF mapping rules are intended to be able to operate against SIMF models without the use of higher order logic inferencing.

These capabilities are not arbitrary – they represent real concepts people need to express about the real world and map real data. By expressing those concepts directly we have a valid “pivot point” for federating data. It should be noted that the kernel of the SIMF language is formalized -described in formal logic, but SIMF does not require such formality in conceptual or information models.

## *Models*

SIMF models describe some subject area or domain for the purpose of defining and sharing *information* within and about that domain and with other related domains. SIMF defines a modeling language for this purpose; users of SIMF use that modeling language to describe applicable concepts; in some cases, a model captures generic, cross domain concepts while in the remaining cases a model may be more specific, that is specific to a discipline, domain, organization or system (we will consider all of these a “domain”). Ultimately a model focused on a domain, any domain, shares many concepts with other domains – the sharing of such concepts is the basis for communications among people, organizations and systems with different backgrounds and purposes. For this reason, domain specific models typically use or specialize more generic cross-domain concepts. To support this requirement SIMF also provides for models that describe general concepts and then provides capabilities to map those general concepts to more specific domain concepts.

**Information**

Models are themselves *information* **about something**. A concept defined in a model is not “the thing”, but a **description of that thing from some perspective**. For this reason, the subject of the model and the perspective are important for understanding the content of the model and how it may relate to other models. A model is fundamentally *information*, that information may be represented in different ways: textually, as tables, graphically, etc. SIMF defines some ways to represent information but other representations may be used as is appropriate for a purpose. The same information is typically rendered in multiple different formats for different purposes and stakeholders.

**Bridging technologies**

A model needs some way to store information. SIMF defines an abstract syntax (nevertheless illustrated with sufficient concrete examples) for storing SIMF information and multiple physical representations, in keeping with the SIMF philosophy of *bridging* technologies. Regardless of the technology, a SIMF model can be thought of as a repository of information. \

**Model representation and exchange using MOF**

Since SIMF is, its self, information about information it is also called a “meta model”. As a foundation for expressing and sharing concrete SIMF models, SIMF uses [MOF], an OMG technology for model management. MOF includes the specification of XML, the format that is used for all MOF models and therefor SIMF.

## *Concepts*

Everything we describe in a SIMF model is considered a *concept*. The SIMF meta model can be used to model and integrate information from multiple source at different levels of abstraction. A key element of SIMF is conceptual models as described above – models a real or possible world. To provide for mapping models of data must also be expressed.

A conceptual model (CDM) includes these real world concepts. Data models include concepts of data about these things. Thus the term “concept” in this document can refer to anything that can be referenced in a SIMF model, which is anything. Note that in the SIMF model the “top level”, the class that all others inherit from is called “anything”. There is no class “Concept” in SIMF. The term concept simply means “something we conceive of and model”. To model something, we must have a conception of it, so to be more precise – we express our conception (concept) of something in a model using the SIMF syntax and vocabulary.

The conceptual domain model (CDM) is a model where we express our conception of a real or possible world

There are other times when we must consider is the model “data” its self or the components of the model data. In this respect a SIMF model is just data. When referencing the SIMF representation of a concept the term “SIMF Element” will be used. A SIMF element is the data representation of a concept in a model repository.

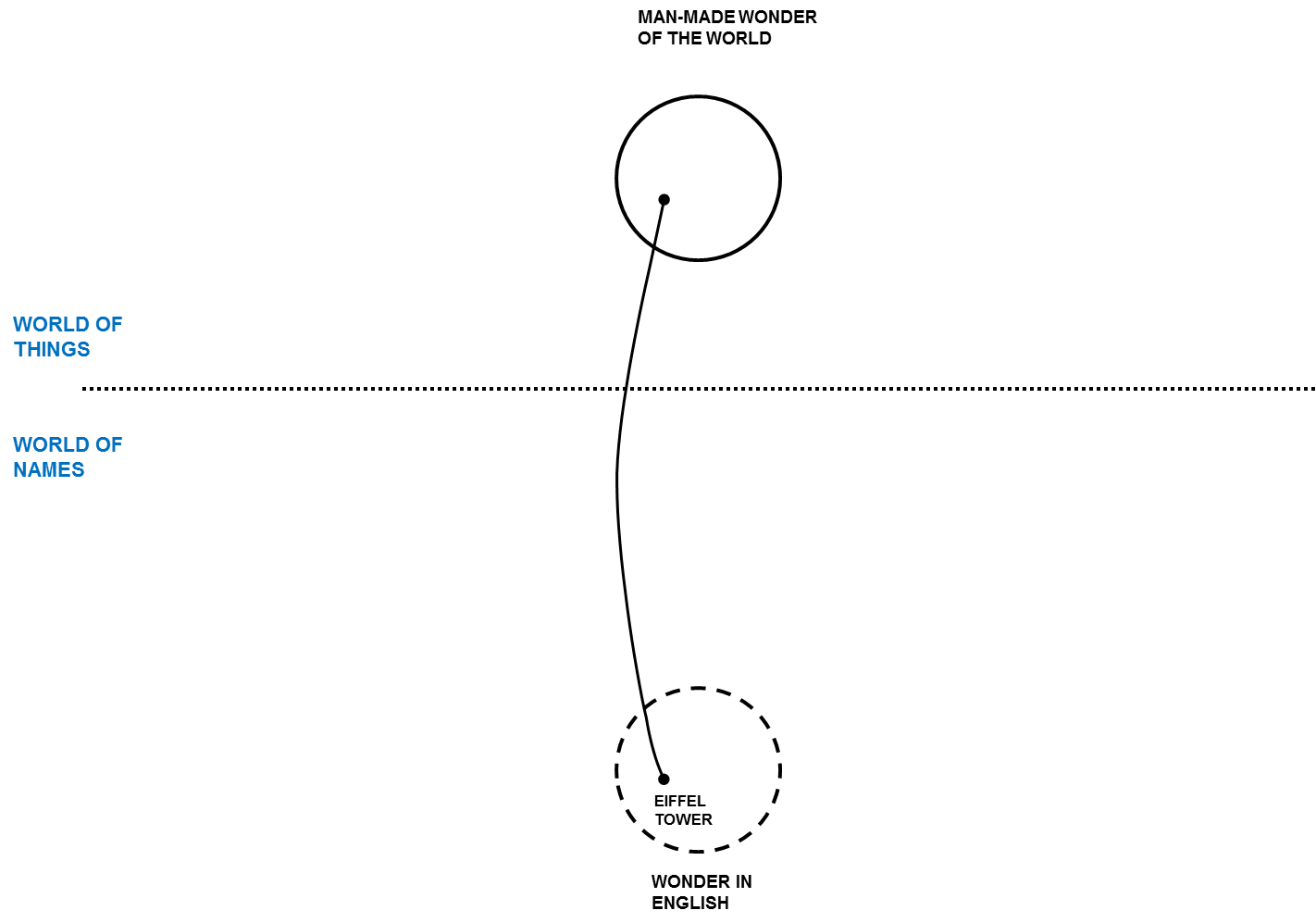
The term concept is used because a model describes how we conceive or understand something from a perspective. For example, we may all understand there is something we call the “Eiffel tower”. In the figure below this information is represented in a very familiar format, an extended Venn diagram. We see a clear distinction as said above between the world of the things and the world of names and information about the things, an essential distinction in SIMF.

A person preferring a textual representation could use the following formal representation:

(1a) “There is something in the world that we are thinking of when the term “Eiffel tower” is used”.

Or in short but sufficient format:

(1b) There is something in the world that is referred to as Eiffel tower.



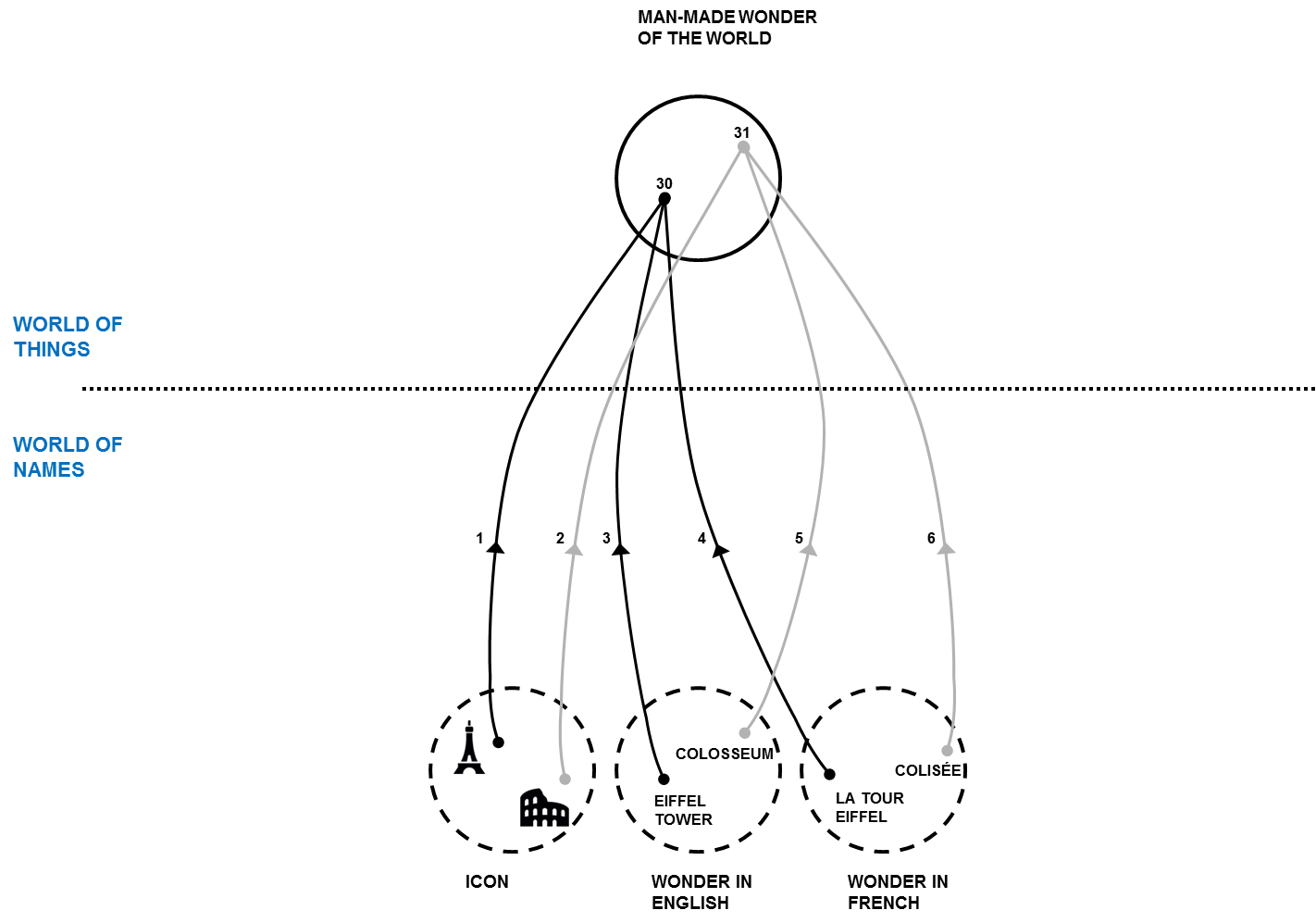
Figure

But how we think about it (the thing designated by the term “Eiffel tower”) or describe it may be different. Differences aside, there is something in the world that we are thinking of when the term “Eiffel tower” is used so we want to connect these different descriptions of the same thing – it is the “real thing” that connects us even though there is no way to have that real thing in our minds or models. We connect the dots through our concepts grounded in our mutual understanding of the world.

Concepts in SIMF may be about anything: “Real things” like the Eiffel Tower, Imaginary things like Unicorns, types of things, actual physical things, characteristics of things, relationships between things, rules, even names.

Once you define a concept in a model you can start adding to it, saying more about it – of course each thing you add is a new concept. A SIMF model becomes a connected graph of these concepts and concepts about concepts. Some of those concepts will also come from other models.

In Figure 3 quite a few additional pieces of information are added to say more about the Eiffel tower. We also bring in a second example, in this case the well-known Colosseum. Please note that the English term Colosseum identifies the same thing as the French term Colisée. Figure 3 can be used to illustrate the essential concepts of context and perspective in a way that practitioners easily understand.



Figure

### Dictionary Concepts

In many modeling languages you need to know a lot about something to define it in a model. In SIMF, you can invent new concepts any time you like as a dictionary defined concept and may or may not provide further formalization. In defining such a new concept you may give it a name and it is best to give it a description. Other concepts are defined more contextually and may not require a name. For example, the concept that the Eiffel Tower was erected in 1889AD or that it is 324 meters tall (including the TV antenna) do not need extra names; they are just facts about the Eiffel tower expressed as relationships concerning the Eiffel tower and some other thing or value. Facts are concepts describing one or more other concepts. In the figures above, each line is a fact. However, facts do have “identity” so their provenance can be traced, crucial in trusting any “fact”.

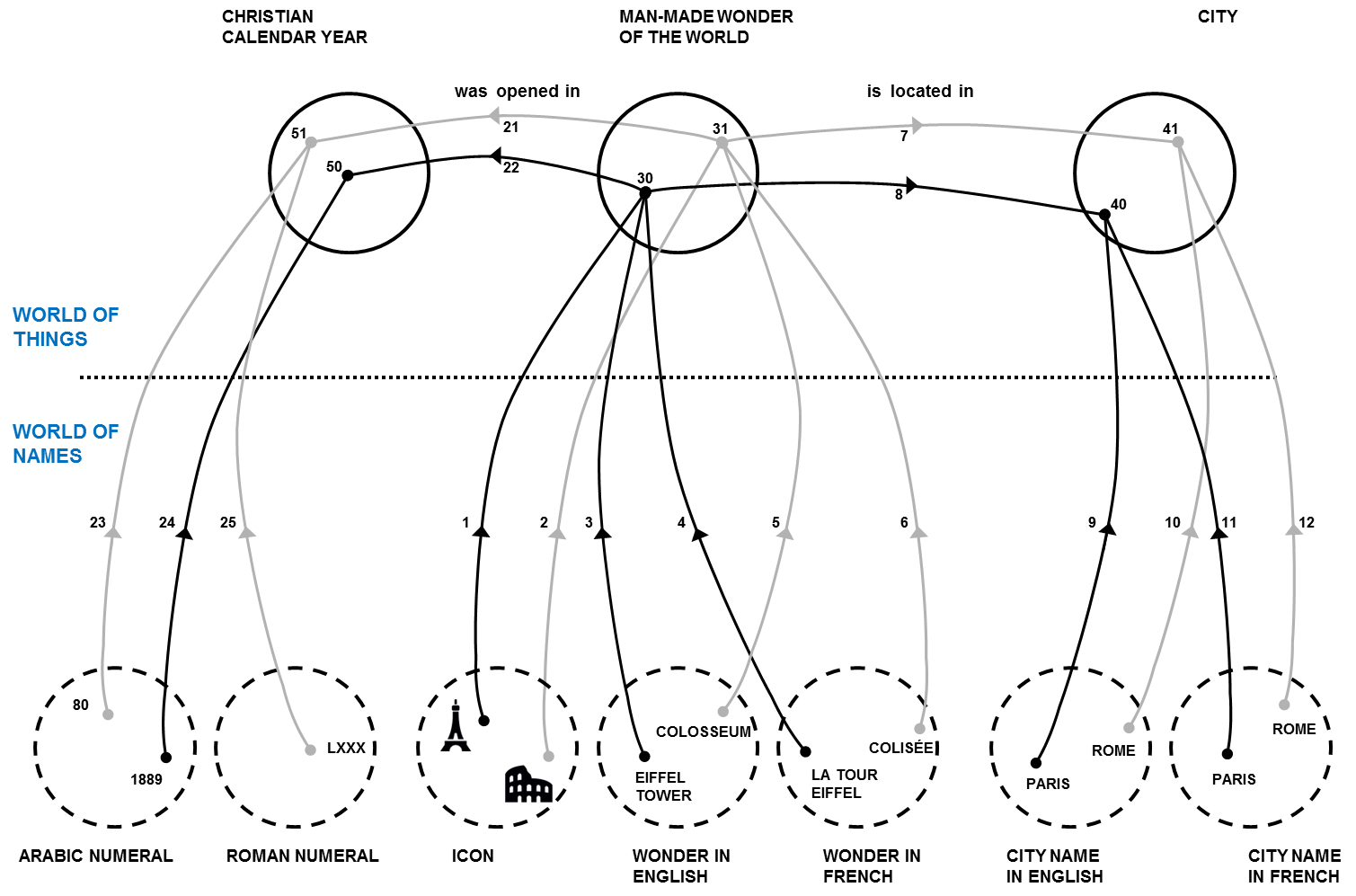
## *Facts*

Facts are something that someone or something asserts to be true. The class of things that can be asserted are called “propositions” as they can be true or false. Once asserted these propositions are facts. Of course the relevance, trust or belief in facts is open to interpretation. SIMF provides information to evaluate the source and context of facts, trust can then be evaluated but this evaluation is outside of the scope of this specification. SIMF does not deal with “Truth” from a philosophical perspective.

Typical kinds of facts are relationships, constrains and rules. Relationships are “ground facts” about the actual things – constraints and rules are more general assertions about groups or classifications of things.

## *Context*

In this section we will use Figure 4 as the diagrammatic representation to illustrate contexts and perspectives.



Figure

As descriptions of something are from a certain perspective, not all concepts are the same. You may trust some and not others, some may be true of a certain timeframe, and some may only be valid in specific conditions. A central problem in information federation is that the context of information from various information repositories is different and often implicit. Different authorities may or may not agree on facts, they may or may not use the same or conflicting terms, they may describe different aspects of the same thing. Each description, each concept, needs to be understood within the context in which it is defined and used. SIMF defines “context” as a “first class” concept – that means that dealing with context is part of using SIMF.

In SIMF, a context connects a set of facts with a set of things those facts apply to. The context *contextualizes* a set of concepts that it applies to. Facts and rules are defined in one or more context and apply to everything those contexts contextualize.

There is a fact that the Eiffel Tower is the tallest building in Paris. Who said that? Can you believe it? If I got that fact from Wikipedia, does that make it believable? The source of information is one of the contextual dimensions. What if a taller building were built in 2020? The fact that “the Eiffel Tower is the tallest building in Paris” would be true in the timeframe from 1889-2020. Timeframe can also contextualize a fact. From this example we can see that the applicability and interpretation of a fact is contextual and that multiple contexts may be applicable to any one fact. Your perspective defines what contexts are meaningful and trusted in a situation. SIMF defines context and provides annotating models with context and provenance, the interpretation of that information to form “trust” and “belief” is outside of this specification. Tools that build on SIMF may provide was to evaluate or quantify trust and belief based on the information SIMF provides.

## *Entities*

Individual things that exist in our world, or some conceived world, are entities. Examples of entities include Barack Obama, The Eiffel Tower, Hurricane Andrew, an agreement between Barack Obama and the country France, the mortgage for my house, etc. Entities include physical things (with a “special/temporal extent”) that do exist, may exist in the future or have existed in the past. Entities also include specific things we conceive, such as agreements between specific individuals. Entities have a “lifetime”, and they may change over that lifetime however that lifetime may or may not be known and we may have facts about an entity before, during and after its lifetime. An important aspect of entities is that they have some kind of identity that remains with the entity over its lifetime – even as aspects of that entity may change.

Entities also include types and classification. Thus the concept of frogs is also considered an entity. A specific frog would be an “individual entity”.

Entities may be differentiated from “Values”, below.

## *Values and Anything*

Values, such as the number 5 or the text “Frog” are “values”, they are information we use to describe things. Information and values form a tree that is distinct with “Entities”. Entities are anything that you can identify other than values. Above entities and values is the superclass “Anything”. All elements in SIMF derive from “Anything”

Numbers are the most common kind of such values – you can’t “create” the concept of “5”, it “just is” based on our accepted theory of numbers. Values are typically used as the object of properties (also called attributes) such as something’s size, mass or some other quantity. Numbers, strings and certain fixed enumerated values are considered value types.

## *Situations*

Various configurations, states and properties of things exist over some timeframe, as understood from some perspective. Such configurations of things are *situations*. A situation is a description of any configuration of things, including the logical, temporal and physical relationships between them. Situations can be large and comprehensive – like our solar system or as granular as your weight at a particular time. Facts about anything can be contextualized by a situation and that situation may, in turn, be contextualized – perhaps by a timeframe. Situations may be real or imagined, in the past, present or future.

Situations form the basis for relationships and facts in SIMF as any fact or relationship could be bound by a time dimension when it was true. As all situations are bound by some timeframe where they are true (regardless of whether that timeframe is known or not), situations are the link between “things” and time. When we look as a “snapshot” of the world, or of data, we are looking at a particular situation.

## Relationships

Relationships (a kind of situation) are concepts that logically connect other things and represent the primary semantic propositions of SIMF. For example, the height of the Eiffel Tower is a “Height” relation between the Eiffel Tower and the quantity “324 meters”. That my computer is on my desk may be a “supports” relation between my desk (as the support) and my computer (as the thing supported). While the set of relations describing something may change over time, each relation is considered atomic– the “parts” bound to the relation (the things they relate) don’t change, any change in the things related would be a new relation. Said another way, the related entities define the identity of the relationship.

Relations can involve any number of “properties” that define how other concepts are connected to the relation – in SIMF relationships may have any number of properties. Properties are the “ends” of the relationships. While most relationships have 2 ends, any number of ends may be represented.

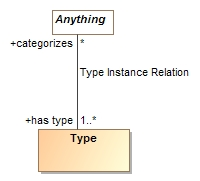
Relationships are considered “atomic propositions” in SIMF and are either true or false. Relationships that are time sensitive should include time as one or their properties.

It should also be noted that in SIMF relations are “first class”, that is relations may have relationships to other things, including other relationships. Relationships quantifying the timeframe where other relationships are true are a very common form of relationships applying to relationships. Note that this provides for SIMF models that may represent “4D” concepts, where time is the 4th dimension.

## *Types*

Types categorize and describe sets of things. Everything in SIMF has at least one type and may have any number of other types as well. The set of types that categorize a concept may change over time and be specific to a context. Rules describe and may constrain the extent of a type. As in most modeling languages, types may specialize other types and form a hierarchy. A type may specialize any number of other types, thus supporting “multiple inheritance”.

### Types and Instances

The essential intent of types is that they categorize a set of things. A type provides the context for the rules and intent of what is categorized. The set of things categorized by a type is the “Type Instance Relation” where a type <categorizes> a set of things and.the things thus categorized <has type> of one or more types. The “Type Instance Relation” may be explicitly asserted in some context or may be inferred based on rules. One way or the other, the “Type Instance Relation” is the complete set of instances of a type.

In a conceptual model where we are modeling the “real world” a type is assumed complete unless otherwise constrained. For example; the type “Dog” categorizes all dogs, not just those that may be known to you or some computer system. Of course you may have subsets of all Dogs in a particular context that is known to you or a computer system – such as the set of Dogs in your house.

In some programming languages the term “Instance” may imply other things, such that a “Class” is a “Factory” for a set of instances or that something may be an instance of only one type. The term “Instance” as used in SIMF designates something categorized by a type and does not imply (or prevent) any implementation or other restrictions. For example; “Fido is an instance of Dog” only means that Fido <has type> Dog and that Dog <categorizes> Fido.

### Multiple inheritance and multiple classification

A type may specialize any number of other types, thus supporting “multiple inheritance”. In addition, a concept may have any number of types and these types may change over the lifetime of the concept (see Facets 5.11.3). Note that this is different from the typical “OO Language” where an instance may have only one type and that type is fixed. Multiple inheritance and multiple classification better capture true domain semantics.

Since the assumption is that a concept may have any set of types, it is necessary to sometimes declare that some types can’t be combined in the same concept – for this purpose as specific “Disjoint” constraint is used. For example, something can’t be a Frog and a Planet. The assumption is that any 2 types can be mixed unless they are specified as Disjoint. In the profile Disjoint may be specified with the <<disjoint with>> dependency or a UML generalization set.

### Situation Types & Properties

Situations have their own specialization of the type concept, one that defines– Properties. A property represents the state or behavior of something involved in a particular kind of situation – frequently expressed as a verb phrase. For example, in an “Employment” relationship for employer “has employee” Fred and the employee “has employer” Acme Industries. Each situation type defines the properties that can be involved with that kind of situation.

Properties have a set of potential rules that can be defined for them in various context. These include the types able to bind to the property, how many things can be bound, etc. We can also define a hierarchy of such properties.

Relationships define a set of properties for the related things. In a conceptual model properties use verb phrases to describe the related thing from the perspective of its opposite ends but the entire relationship defines a single fact with its own identity. The relationship serves as the connective tissue between these different ways to read the same fact. Note that this is different from some ontology languages like OWL, where independent “predicates” describe the same fact as an “inverse”. Since SIMF ascribes identity to facts it is important that these verb phrases are understood to be different “readings” of the same fact.

Properties are bound to individuals in a situation. Properties also serve the purpose of “roles”, “variables” and “arguments” in programming languages.

Examples:

* Barack Obama is bound to the “President” role in the Organization “United States of America” during the period January 20th, 2009 (Noon EST) to January 20th, 2016 (Noon EST).
* The “Eiffel Tower” is bound to the “height of” role and “324 meters” is bound to the “has height” property in the relationship “Height”.

**Note on naming properties**: In a conceptual model it is best practice to use verb phrases for the “ends” of a relationship. In physical models it is more common to use noun phrases.

### Facets

Some types are an essential part of what something is and its identity. For example a Dog “Fido” is born and dies a dog, you can’t take away Fido’s dogness (at least not today, we shall see how genetic engineering goes). On the other hand Fido may be a “Good dog” or a “Wild animal” or a “Pet” at various times. These “extra” classifications of Fido are not essential to Fido being Fido. In [BFO] this distinction is made in terms of “rigid” Vs. “non rigid” sortals (types). The non-rigid classifications are called “Facets” in SIMF.

Something can’t just be a Facet, it has to have some rigid non-faceted type as well, this rigid type is the foundation for the identity of individuals. Facets don’t create a new identity, they classify an existing one (sometimes in a specific context).

It is frequently desirable to specify what rigid types a Facet can classify. This is done by marking the generalization from the Facet to the rigid type “as facet” in the meta model or a <<classifies>> stereotype in the UML profile. This says that the facet must be a facet of this type and also implies that the subtype must be a facet. It is also possible to have a facet of a facet, where the rigid identity comes from a supertype of the more general facet.

Note that there are various ways to map facets to implementation technologies – in most cases each facet will be a different “technology object” that represents the different facet of the same underlying thing in the world.

There are specific kinds of Facets, “Roles” and “Phases” as defined below.

### Roles

A role is a facet that defines how something behaves or participates in some situation. Typical examples are “Teacher”, “Student”, “Pet”. In a conceptual model it is valuable to separate “what something is” from “what it is for” or “what it is doing”. Roles allow this important separation of concerns.

### Phases

A phase (or state) is a characteristic of something that exists for limited time(s). Phases are time-bound facets. Something takes on or loses a phase as a result of some event (including the passage of time). E,g, Teenager, living, closed invoice.

## Rules

Rules define some general proposition that hold within (must be considered true) within a context (including situations, types and relationships). There are various kinds of rules, some of which are defined in SIMF and some which are defined in SIMF models. Rules are used to define the constraints of entities, relationships and other concepts. Rules are also the basis for defining the mapping between different models.

A rule is something that can be asserted to “hold within” some context (including types and situations). That rule then applies to everything that is contextualized by that context. Context are the essential binding between rules (or any assertion of a proposition) and the things that rule applies to. There are types of rules, so rules also form a hierarchy.

### Constraints

Constraints are rules that define some condition that must be true, else the model is not valid. A constraint may be enforced for all time (at once = false) or at a particular instant in time elements (at once = true). SIMF does not infer any elements to attempt to satisfy a constraint, such inferences are done with rules or by external systems.

Examples of constraints include:

* Equivalence
* Disjointness
* Types of properties
* Multiplicity of properties
* Generalization (subclassing of properties and types)

SIMF can also express general constraints as patterns, such that all fish live in water.

### Mapping rules

An essential feature of SIMF is mapping between different representations of the same concept. Mapping rules are based on patterns – mapping a pattern in one model to a pattern in another model. These patterns are constrained by the SIMF type system and “representation rules” such that for each thing to be mapped, SIMF also expressed what (data focused) types can represent SOMF domain concepts.

The combination of representation rules and mapping patterns provide the basis for information federation.

## Expressions

SIMF provides for general expressions which are then used in other parts of SIMF, for example in rules. The SIMF expression meta model is minimal, the bulk of the functionality to be provided by function libraries. The built-in capabilities are:

* Constant reference
* Traversing relationships
* Testing equality
* Calling functions

The philosophy of SIMF expressions is to put almost all expression semantics into functions. A function library is anticipated as a future SIMF model.

## Characteristics, Quantity Kinds & Unit Types

While some relationships are between entities, others relate a characteristic of an entity to some value.

To provide for a semantic definition, in a conceptual model characteristics always have a domain-relevant type (not a data type, like “Int”). In the case of quantities (a characteristic that can be identified with a number and a Unit), there is always a unit type corresponding to some “quantity kind” such as temperature or mass. Units are, of course, critical in understanding what a quantity really means. Characteristics are defined with relationships where one of the properties is a value. The “object” of such a characteristic relationship is always a value type.

Based on the definitions in JCGM 200:2008 : A “Quantity Property Type” (e.g. Height) has a range of a “Quantity Kind” (e.g. Length) which is represented by a “Measurement Unit” (e.g. Meters) . An instance of a Measurement Unit is a “Quantity Value” (e.g. 324 Meters).

SIMF defines the modeling capability for defining quantities, quantity kinds and unit types. Models expressed in SIMF define libraries of quantity kinds and units, but these models are not part of the SIMF specification. Later specifications may provide Unit Type libraries.

## Terms and Naming

A SIMF concept can be named by any number of terms. A “Term” binds a string label to a concept in some context. A concept may have multiple terms in multiple context different domain taxonomies and/or natural languages can identify the same concept. Concepts can also be nominated as “preferred”. While terms are considered primarily for human consumption, concepts can also have various identifiers that are more meaningful in implementation technologies. The “URI” is the most common form of identifier. Concepts can also have multiple identifiers.

## SIMF Lexical Scope & Physical Representations

A SIMF model is an information resource and as such has structure and semantics as defined in the SIMF abstract syntax. Most of the SIMF model describes the modeled domain or subject. Other “lexical” (or syntax) aspects describe the model itself or elements of that model – its content, source and organization.

Only a small subset of that abstract syntax is relevant to the structure of the information, this structure is defined in the “Lexical Scope” portion of the model. The lexical scope structure is then mapped to various technology formats such as MOF-XMI or RDF. MOF-XMI is the normative representation.

Like most modeling languages, a SIMF model is “block structured”, that is certain elements contain other elements. The model structure is one form of context that may or may not be relevant to its interpretation, SIMF attaches no implicit semantics to model structure. Most users will not be concerned with the details of the lexical structure, but an understanding of it is important for implementers, as follows…

A key element is the abstract “*Lexical Scope*” which can define and/or import any context in any model. Referencing a context makes it visible within the lexical scope, importing a context asserts that external scope within the lexical scope. A lexical scope may also define Terms, which provide labels for concepts.

A package is a user defined lexical scope for organization purposes. There can be packages for various purposes which are defined as subtypes of Package. As a lexical scope and context, a package MAY be used for assertions.

A *Model* is a package with no parent, it is an independent data resource.. A model is identified as modeling a particular level of abstraction (Conceptual, Logical, Physical or Binding) and subject area.

## Patterns

Patterns describe a class of similar situations with some constant and some variable parts. Patterns utilize “pattern properties” as the variables and provide various rules for what those variables do or may represent. Pattern variables are combined with relationships within the pattern – things that are always true for all instances of the pattern. Relationships in patterns can reference both pattern variables as well as actual things. Patterns are used for rules, including mapping rules. Patterns may also be used for general constraints.

## Situation and pattern instances

Particular situations as well as patterns “bind” things to properties in relationships. Bindings bind an individual to a property in a situation. Each “end” of an instance of a relationship type is a binding.

Bindings are possible due to the SIMF concept identity. Every concept in SIMF has identity or a value: Every individual, relation, binding and type. For this reason, SIMF relationships may be made between anything. It may be noted that this is not “first order”, as in “first order logic” (FOL). Supporting a rich conceptual model and mappings requires concepts that are not first order. Portions of SIMF models may be mapped to first order languages using well known patterns to take advantage of their inference capabilities.

## Record Structures and Structured Types

Data schema (PDSs) and information models (LIMs) typically have specific data structures that hold information about something as a “snapshot” in time. In these kinds of structures that properties are “owned by” the structure, sometimes these properties are called “slots”. To provide for an accurate representation of “target” data structures SIMF allows the definition of “Records”. Unlike entities in conceptual model, records may directly contain properties. These records may then be matted to their conceptual counterparts.

To support records, SIMF defined Record Types and Records (an instance of a record type). Record types are intended for LIMs and PDSs but CDMs are not prevented from using records.

There is a general pattern that can be noted in SIMF, that of “something with properties”. SIMF defines an abstract “Structured Type” as the owner of properties. Anything that can have properties inherits, directly or indirectly, from “Structured Type”. This includes Situation Types, Relationship Types, Record Types, Patterns and Operation Types.

Bindings are used to bind a particular thing to a property within a situation.

## Views and Viewpoints

The separation of general concepts from viewpoints of that general model and then specific views of a model is well established. In evaluating existing schema and languages it has become apparent that the industry is reasonably good at defining viewpoints – that the existing languages are already viewpoints that can then express multiple views. What the industry has not been good at is connecting the viewpoints to the underlying concepts. The SIMF CDM is a representation of the underlying concepts. Our assumption is then that the views and viewpoints will be expressed in the LIM and PDS models that SIMF is mapped to. For this reason, SIMF is not attempting to define viewpoints. Existing technologies such as MOF and the OMG/MDA stack are already able to define such viewpoints. SIMF can map these viewpoints to common concepts – what they are viewpoints of.

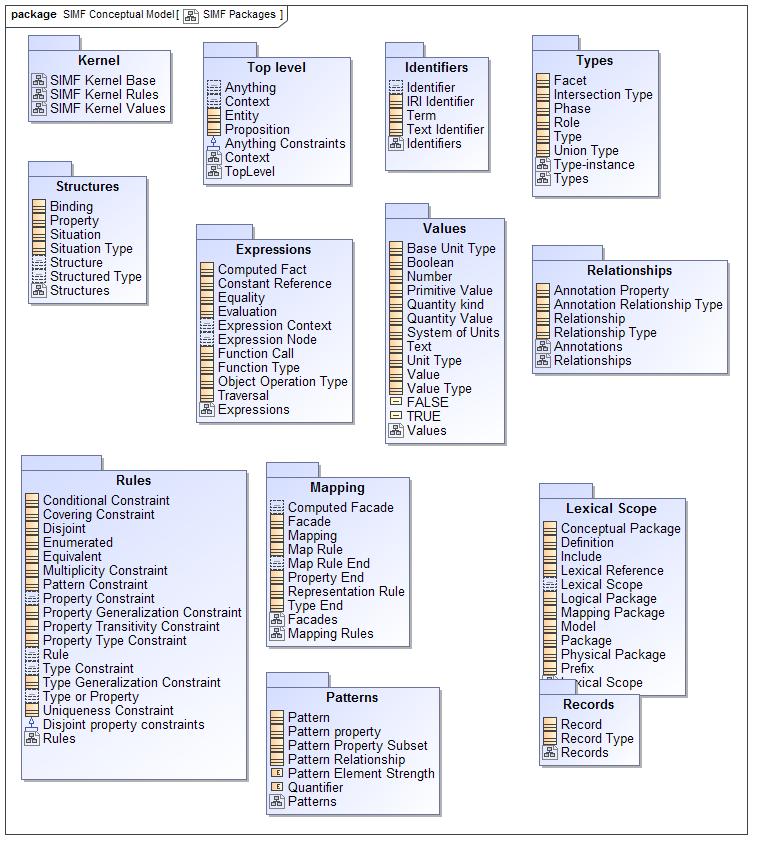
# SIMF Conceptual Model & Abstract Syntax [Normative]

This section presents the normative specification for the SIMF conceptual model and metamodel. It begins with an overview of the metamodel structure followed by a description of each sub-package.

The SIMF MOF metamodel (abstract syntax) is derived directly from the SIMF conceptual model by removing the SIMF profile specific elements that would not be interpreted correctly by MOF – in particular the expression of restriction rules based on subsets and redefines. In addition, data types are mapped to the MOF counterpart. For this reason, there is only one model documentation where as there are two model artifacts – one for the conceptual model and one for the MOF metamodel.

## SIMF Conceptual Model

### Diagram: SIMF Packages

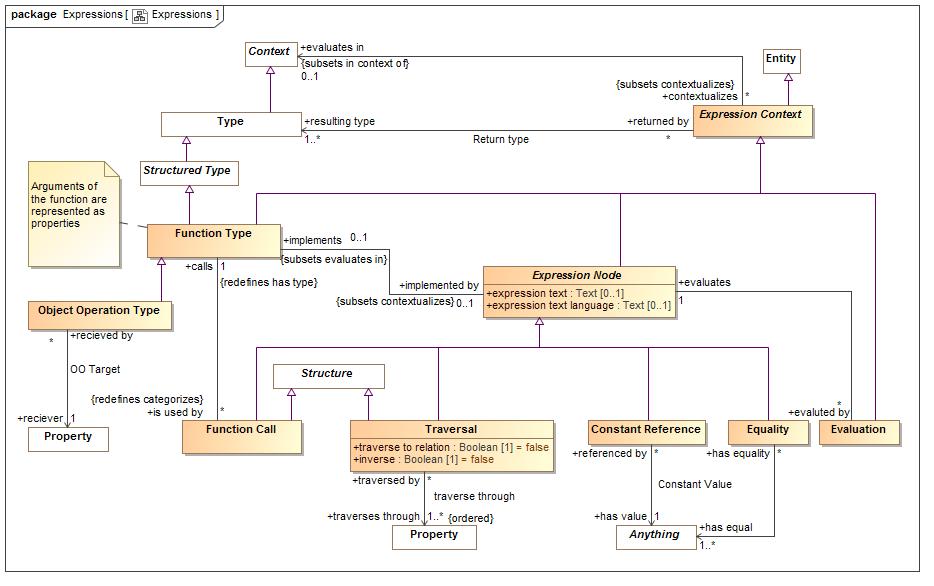


1. SIMF Packages

## SIMF Conceptual Model::Expressions

Expressions define computations across SIMF models.

### Diagram: Expressions



1. Expressions

Expressions define computations

### Class Computed Fact

The actual evaluation of an expression which may cause the creation of more assertions (i.e. representing the result of a calculation).

**package** SIMF Conceptual Model::Expressions

### Class Constant Reference

A calculation that returns a thing. The thing <has value>.

#### Direct Supertypes

[Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

**package** SIMF Conceptual Model::Expressions

#### Associations

-58492342.jpg has value : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

A constant value referenced in an expression.

### Association Constant Value

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-58492342.jpg has value : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

A constant value referenced in an expression.

-58492342.jpg referenced by : [Constant Reference](#_f3f61859903284f1b00fc6feee0b33f8) [\*]

Referencing constant.

### Class Equality

Returns TRUE if all <has equal> things have the same value or represent the same thing or set of things regardless of how they are represented.  
Equality will return TRUE or FALSE.

#### Direct Supertypes

[Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

**package** SIMF Conceptual Model::Expressions

#### Associations

-58492342.jpg has equal : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1..\*]

Set of things that must have the same value or represent the same thing or set of things for Equality to return true.

### Class Evaluation

The evaluation of an expression. All references to an evaluation shall return the result of evaluating the <evaluates> expression node. All expression nodes referenced within an evaluation shall return the result of evaluating that expression node.  
An evaluation may be used in place of anything that requires the <resulting type> of the evaluation.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc)

**package** SIMF Conceptual Model::Expressions

#### Associations

-1808253052.jpg evaluates : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [1]

The expression node "head" an evaluation evaluates.

### Association Expression Context

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-58492342.jpg evaluates in : [Context](#_66d62b068053cee3464e1e03e6035eed) [0..1]

Context of evaluation and namespace resolution for an expression.

-58492342.jpg contextualizes : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]

### Class Expression Context

An abstract element defining the static or dynamic evaluation context and resulting type of an expression.  
An expression context that is referenced by another expression context inherits the referencing context by default.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Expressions

#### Associations

-58492342.jpg evaluates in : [Context](#_66d62b068053cee3464e1e03e6035eed) [0..1] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)

Context of evaluation and namespace resolution for an expression.

-58492342.jpg resulting type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]

Type of the result of a function

### Class Expression Node

The computation of a value which is then bound to the context from which it is called. Each expression node has a type of the most general type it can return.  
  
An expression node may reference other elements. Where the other elements are also expression nodes they will be considered part of the referencing expression and evaluated in the context of that expression.  
  
The set of related expression nodes forms a "tree" for evaluation.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc)

**package** SIMF Conceptual Model::Expressions

#### Attributes

-1626698621.jpg expression text : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2) [0..1]

Textual expression of the expression which is further refined by subtypes of expression.

-1626698621.jpg expression text language : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2) [0..1]

expression language used for the expression text

#### Associations

-1808253052.jpg implements : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [0..1] *Subsets*: evaluates in:[Context](#_66d62b068053cee3464e1e03e6035eed)

Function implemented by an expression

-1808253052.jpg evaluted by : [Evaluation](#_764178c56513beb91e5b5964ec31da8e) [\*]

Evaluations of an expression node.

### Class Function Call

An element of an expression that performs some operation based on a function type and produces a result. I.e. plus(a,1).  
Arguments are bound to the function call via bindings.

#### Direct Supertypes

[Expression Node](#_f9bba899ada544a47c36bb071e9024f5), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Expressions

#### Associations

-1808253052.jpg calls : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [1] *Redefines*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

Function called

### Association Function Implementation

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-1808253052.jpg implemented by : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [0..1] *Redefines*: has type: [Type](#_dfe1514224ca21cedba7b2b29802db50)

Expression which defines the implementation of a function.

-1808253052.jpg implements : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [0..1] *Redefines*: has type: [Type](#_dfe1514224ca21cedba7b2b29802db50)

Function implemented by an expression

### Class Function Type

A declaration of a function which performs a calculation on arguments (properties) to produce a result (function result). I.e. the definition of plus(a:Number, b:Number).  
  
Functions are intended to be side-effect free and context free (they only depend on their arguments and don't change anything) but assertions to specify that certain functions are pure may be required,  
Note: FUNCTION ARGUMENTS ARE PROPERTIES of the function.

#### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc), [Structured Type](#_50241f5936e61055293ca95f860768d8)

**package** SIMF Conceptual Model::Expressions

#### Associations

-1808253052.jpg is used by : [Function Call](#_db3e44e523a232e5b77a133d74842e81) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

Function calls using a function declaration.

-1808253052.jpg implemented by : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [0..1] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

Expression which defines the implementation of a function.

### Association Function Type

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-1808253052.jpg calls : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [1] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

Function called

-1808253052.jpg is used by : [Function Call](#_db3e44e523a232e5b77a133d74842e81) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

Function calls using a function declaration.

### Class Object Operation Type

An operation bound to a spcific "reciever" in the OO sense.

#### Direct Supertypes

[Function Type](#_cff99d2f22ee84a9e95ea582786a897b)

**package** SIMF Conceptual Model::Expressions

#### Associations

-58492342.jpg reciever : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]

The property that is the receiver of an object operation. The receiver must be a property of the function type.

### Association OO Target

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-58492342.jpg reciever : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]

The property that is the receiver of an object operation. The receiver must be a property of the function type.

-58492342.jpg recieved by : [Object Operation Type](#_e6c2e5d52e1652a6c3d27d411345c754) [\*]

### Association Return type

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-58492342.jpg resulting type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]

Type of the result of a function

-58492342.jpg returned by : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]

Ty

### Class Traversal

Traversal from the current <evaluates in> context to another across a relation or other structure.  
  
A traversal is a structure such that the structure's bindings may hold other properties of a traversal constant as independent variables where <traverses through> is the dependent variable. The traversal shall be considered to have the type of the relation it is traversing. Traversing binary relations does not require any bindings.

#### Direct Supertypes

[Expression Node](#_f9bba899ada544a47c36bb071e9024f5), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Expressions

#### Attributes

-1626698621.jpg traverse to relation : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1] = false

Where traverse to relation is false, the traversal will return the bound element(s) of the <traverses through> property from the current context via any intermediate relationships.  
  
Where traverse to relation is true, the traversal shall return the structure/situation/relationship owning the property binding.  
  
By default, traverse to relation is false.

-1626698621.jpg inverse : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [1] = false

Indicates that the traversal is defined based on properties that reference the current context. This results in traversing "backwards" across a property to an inverse property or the relation.  
Where the situation referencing the property has more than one other property, <traverse to relation> must be TRUE.

#### Associations

-58492342.jpg traverses through : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

Property or properties through which a traversal traverses as the dependent variable(s).

### Association traverse through

Property of current context which will be traversed.

**package** SIMF Conceptual Model::Expressions

#### Association Ends

-58492342.jpg traverses through : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

Property or properties through which a traversal traverses as the dependent variable(s).

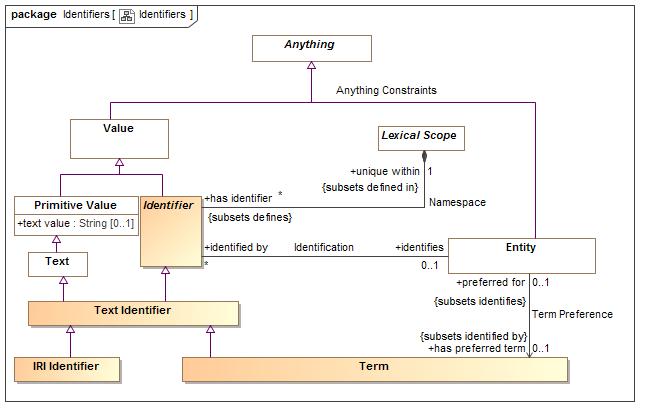
-58492342.jpg traversed by : [Traversal](#_0492440b12b90a76377a15324efa2182) [\*]

Traversals through a property.

## SIMF Conceptual Model::Identifiers

Terms and identifiers provide for signs for (ways to identify) anything.

### Diagram: Identifiers



1. Identifiers

### Association Identification

**package** SIMF Conceptual Model::Identifiers

#### Association Ends

-1808253052.jpg identifies : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]

The entity an identifier identifies.

-1808253052.jpg identified by : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*]

An identifier for an <Entity>.

### Class Identifier

An <Identifier> is a value used to uniquely identify an <Entity>. The identified entity is referenced by what the identifier <identifies>.  
An identifier identifies a particular "meaning" as represented by what it <identifies>.   
Identifiers are defined and <unique> within> a lexical scope as its namespace.  
Multiple identifiers may use the same word or text value (or other forms of values) in differing <unique within> namespaces such that the same word may have different meanings in different context.  
An entity may have any number of identifiers.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Identifiers

#### Associations

-1808253052.jpg identifies : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]

The entity an identifier identifies.

195961665.jpg unique within : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

The namespace in which an identifier is defined and has a unique value.

### Class IRI Identifier

A IRI/URI Identifier for a an entity,   
as defined in [RFC3987].

#### Direct Supertypes

[Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

**package** SIMF Conceptual Model::Identifiers

### Association Namespace

**package** SIMF Conceptual Model::Identifiers

#### Association Ends

195961665.jpg unique within : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

The namespace in which an identifier is defined and has a unique value.

195961665.jpg has identifier : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

An <Identifier> defined within a scope.

### Class Term

A word, phrase or name used by stakeholders to identify entities.

#### Direct Supertypes

[Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

**package** SIMF Conceptual Model::Identifiers

### Class Text Identifier

An <Identifier> that is represented using text. e.g. a "word", "phrase" or "name".

#### Direct Supertypes

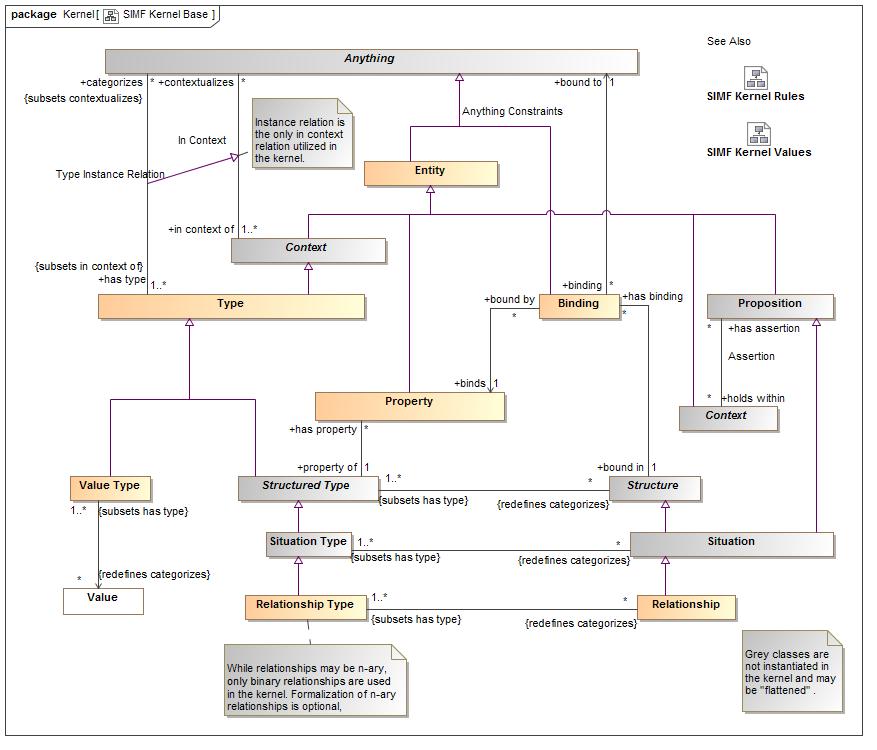
[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26), [Text](#_0e6e6fe0a29fb43221940aa4118b04a2)

**package** SIMF Conceptual Model::Identifiers

## SIMF Conceptual Model::Kernel

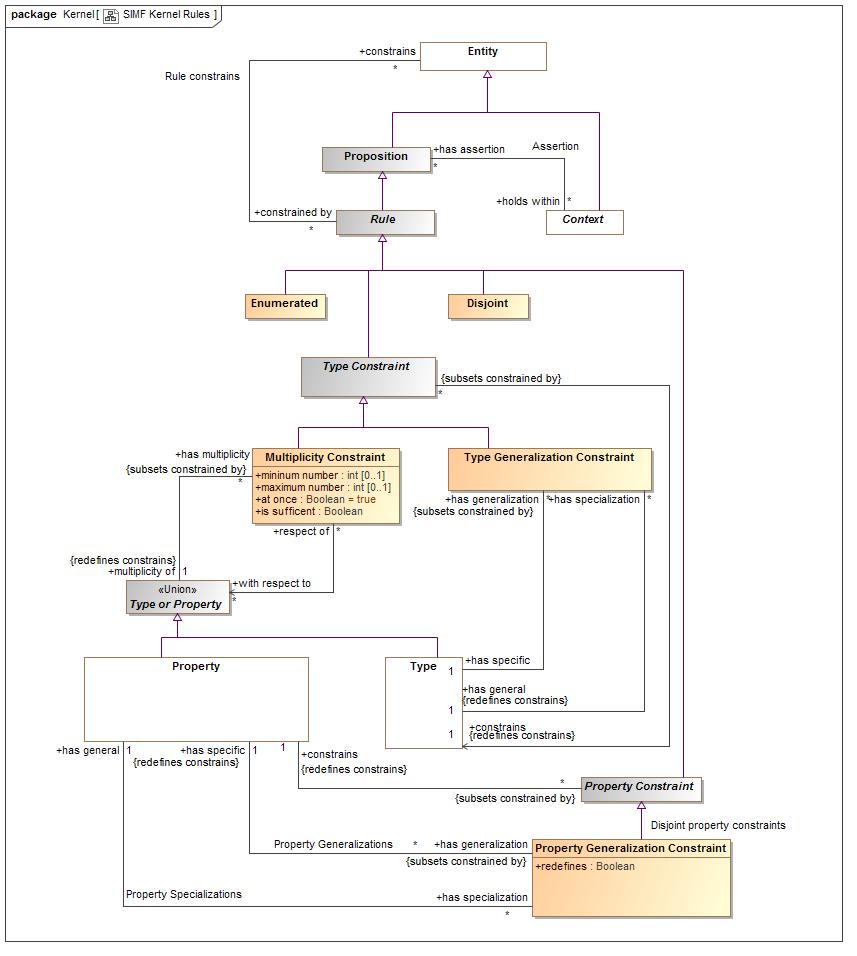
The kernel defines concrete classes that are used to define the SIMF language

### Diagram: SIMF Kernel Base



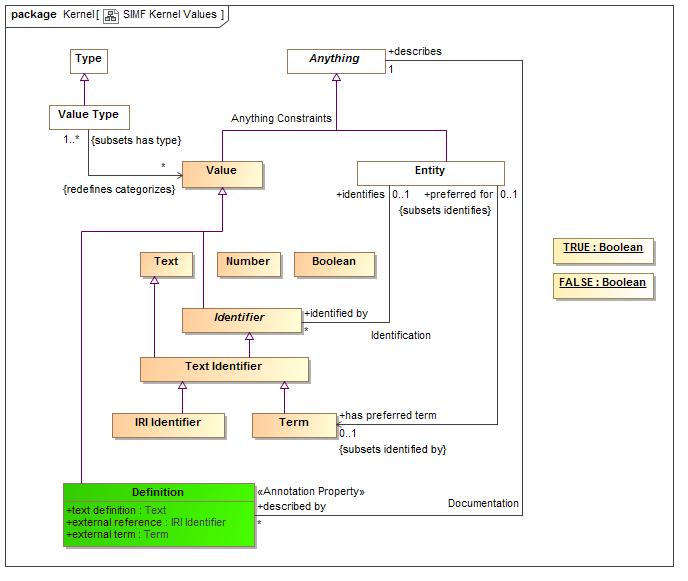
1. SIMF Kernel Base

### Diagram: SIMF Kernel Rules



1. SIMF Kernel Rules

### Diagram: SIMF Kernel Values

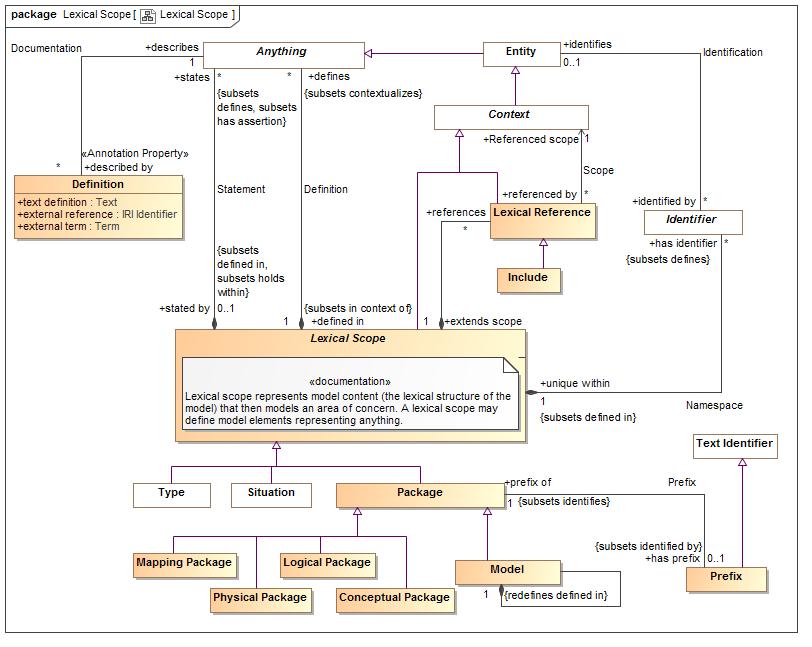


1. SIMF Kernel Values

## SIMF Conceptual Model::Lexical Scope

Lexical scope defines the structure of models

### Diagram: Lexical Scope



1. Lexical Scope

An indicator that the referenced context is asserted by (included by reference) the referencing context.

### Class Conceptual Package

A model if a real or possible world as conceived by the model authors.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

### Association Definition

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

195961665.jpg defines : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

A model element defined within a lexical scope.  
Definition within a scope does not assert everything within a scope but the lexical scope may be independently asserted, thus asserting what it defines.

195961665.jpg defined in : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

Lexical scope defining model elements.

### Class Definition

An informal or natural language definition of a something and potentially a reference to external definitions.  
A <Definition> May be in the context of a natural language to scope the language it is expressed in.

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Lexical Scope

#### Attributes

-1626698621.jpg text definition : [Text](#_0e6e6fe0a29fb43221940aa4118b04a2)

Text describing a something in natural language. The language may be indicated by a context of the definition.

-1626698621.jpg external reference : [IRI Identifier](#_f904ff1da5bfc3387d892b7e0fe9ecb1)

A reference to an external information resource that further defines the something.

-1626698621.jpg external term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8)

Specific term in an external resource that further defined something.

#### Associations

-1808253052.jpg describes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

Some thing described by a definition.

### Association Documentation

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

-1808253052.jpg describes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

Some thing described by a definition.

-1808253052.jpg described by : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [\*]

An informal description of something.

### Class Include

An "Include" is an external scope that is visible and asserted by the owning lexical scope.

#### Direct Supertypes

[Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0)

**package** SIMF Conceptual Model::Lexical Scope

### Class Lexical Reference

A Lexical Reference is an external scope that is visible to but not necessarily asserted by the owning lexical scope.

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

-58492342.jpg Referenced scope : [Context](#_66d62b068053cee3464e1e03e6035eed) [1]

A referenced context, potentially in another model, that provides visibility to the elements in that context.

195961665.jpg extends scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1]

A lexical scope that is extended by a lexical reference.

### Class Lexical Scope

Lexical scope represents model content (the lexical structure of the model) that then models an area of concern. A lexical scope may define model elements representing anything.

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

195961665.jpg defines : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

A model element defined within a lexical scope.  
Definition within a scope does not assert everything within a scope but the lexical scope may be independently asserted, thus asserting what it defines.

195961665.jpg references : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]

A reference providing visibility of a lexical scope to an internal or external context.

195961665.jpg has identifier : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*] *Subsets*: defines:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

An <Identifier> defined within a scope.

195961665.jpg states : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: defines:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) has assertion:[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

<states> combines <defines> with <has assertion> to both define and assert an element within a lexical scope. <states> provides a more "structural" organization of concepts that are both defined and asserted in the same structure.  
  
<states> is a convenience for the common case where assertion and lexical containment are combined.

### Class Logical Package

A model of information about a real or possible world independent of technical representation.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

### Class Mapping Package

A model defining relationships between other models.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

### Class Model

A root package. A model has no owner and may be directly referenced as an independent information resource. A model is defined in it's self.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

195961665.jpg : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: defines:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) has assertion:[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

195961665.jpg : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: defines:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) has assertion:[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

### Class Package

A group of model elements that provides a definitional scope for those elements.

#### Direct Supertypes

[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

-1808253052.jpg has prefix : [Prefix](#_f02d6606b404e367c8d0a72afd7f68e5) [0..1] *Subsets*: identified by:[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)

An abbreviation that can be used to identify a package.

### Class Physical Package

A physical, technology specific, data schema representing information about a real or possible world.

#### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

**package** SIMF Conceptual Model::Lexical Scope

### Association Prefix

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

-1808253052.jpg has prefix : [Prefix](#_f02d6606b404e367c8d0a72afd7f68e5) [0..1] *Subsets*: identified by:[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)

An abbreviation that can be used to identify a package.

-1808253052.jpg prefix of : [Package](#_0506f167988dfda7ae188b66aefe4f05) [1] *Subsets*: identified by:[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)

An abbreviation for...

### Class Prefix

A technical abbreviation for a package.

#### Direct Supertypes

[Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

**package** SIMF Conceptual Model::Lexical Scope

#### Associations

-1808253052.jpg prefix of : [Package](#_0506f167988dfda7ae188b66aefe4f05) [1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

An abbreviation for...

### Association Scope

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

-58492342.jpg Referenced scope : [Context](#_66d62b068053cee3464e1e03e6035eed) [1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

A referenced context, potentially in another model, that provides visibility to the elements in that context.

-58492342.jpg referenced by : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

References to a context.

### Association Scope Reference

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

195961665.jpg references : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

A reference providing visibility of a lexical scope to an internal or external context.

195961665.jpg extends scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

A lexical scope that is extended by a lexical reference.

### Association Statement

**package** SIMF Conceptual Model::Lexical Scope

#### Association Ends

195961665.jpg states : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

<states> combines <defines> with <has assertion> to both define and assert an element within a lexical scope. <states> provides a more "structural" organization of concepts that are both defined and asserted in the same structure.  
  
<states> is a convenience for the common case where assertion and lexical containment are combined.

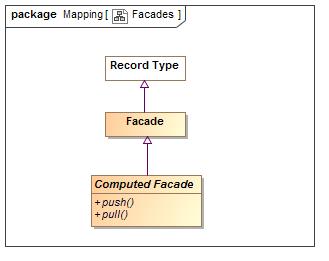
195961665.jpg stated by : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [0..1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

<stated by> is a lexical scope that both defines and asserts a model element.

## SIMF Conceptual Model::Mapping

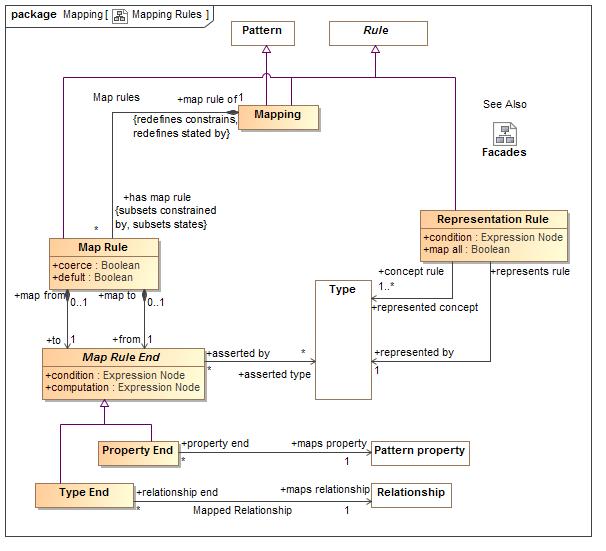
Mapping rules define how data represents concepts or haw different data representations are related.

### Diagram: Facades



1. Facades

### Diagram: Mapping Rules



1. Mapping Rules

### Class Computed Facade

A facade that is computed by calling external methods.

#### Direct Supertypes

[Facade](#_90ac762f4f29c31e6c33236231df6a9a)

**package** SIMF Conceptual Model::Mapping

#### Operations

-2016296986.jpg public push ()

An operation called to evoke the behavior associated with a new facade element being created or modified.

-2016296986.jpg public pull ()

An operation called to evoke the behavior associated with a facade representing existing elements.

### Class Facade

An intermediary data type used to hold common mappings. Facades may be computed and/or have mapping rules.

#### Direct Supertypes

[Record Type](#_d2ebf1b96697234b6aef9b3bfac15784)

**package** SIMF Conceptual Model::Mapping

### Association from rue

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-575849932.jpg from : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

One end of a mapping, to be used for more concrete end.

-575849932.jpg map to : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [0..1] *Subsets*: identifies:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

Mapping rule owning a "from" end.

### Class Map Rule

A rule that the 2 ends represent the same things or information about a thing.  
Redundant mappings are ignored and identity is preserved across all mappings.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Mapping

#### Attributes

-1626698621.jpg coerce : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

Where <coerce> has a value of TRUE a map rule will be evaluated even if the <from> is not type compatible with the <to> type.  
Where <coerce> is FALSE or unstated a map rule will be evaluated only if the <from> is type compatible with the <to> type.  
Type compatible shall be defined as one of: Being the same type, <from> being a subtype of <to> (as defined by a type generalization rule), <from> being a representation of <to> (as defined by a representation rule).  
Representation rules applied to a supertype apply to a subtype.

-1626698621.jpg defult : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

True if the map should be enforced only if no other maps or prior values apply.

#### Associations

-575849932.jpg from : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]

One end of a mapping, to be used for more concrete end.

-575849932.jpg to : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]

One end of a mapping, to be used for more abstract end.

195961665.jpg map rule of : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d) stated by:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

Mapping containing a map rule.

### Class Map Rule End

One end of a mapping from one thing to another that may be qualified with a condition.  
The set of elements to be mapped is the union of the sets of all mapped types and mapped variables that conform to the condition.  
Mapping rules are constrained to apply to only conforming types or types that represent the mapped ends (as specified by a representation rule).   
Representation rules applied to a supertype apply to a subtype unless a more specific representation rule is specified for the corresponding types.

**package** SIMF Conceptual Model::Mapping

#### Attributes

-1626698621.jpg condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

Condition that must be TRUE for the mapping rule to apply.

-1626698621.jpg computation : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

Computation computes a value for the mapping end based on the expression applied to the mapped property or relationship.   
Where computation is used inverse mapping is not specified - any inverse mapping is implementation specific.

#### Associations

-58492342.jpg asserted type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]

Type that will be asserted for the end that is more specific than the type of a property or relationship. e.g. a unit type.

### Association Map Rule Type Assertion

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-58492342.jpg asserted type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]

Type that will be asserted for the end that is more specific than the type of a property or relationship. e.g. a unit type.

-58492342.jpg asserted by : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [\*]

Map rule and that asserts a type

### Association Mapped Property

Type that is the source or target of the mapping

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-58492342.jpg maps property : [Pattern property](#_4d83e476040c7444758dda440d3096fc) [1]

Property that defines a set of elements to map to the other side of the mapping rule based on a pattern property. The set of elements shall be those bound to the property on evaluation of the mapping.

-58492342.jpg property end : [Property End](#_7decbae485fa9be88d63fca6f68547ec) [\*]

Map rule end for a property

### Association Mapped Relationship

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-58492342.jpg maps relationship : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [1]

Defines a set of relationships to map to the other side of the mapping rule based on a relationship contained in the pattern.

-58492342.jpg relationship end : [Type End](#_2d6f605b4614a2c3324d5d7f1fe4743a) [\*]

Relationship mapping end for a type

### Class Mapping

A mapping is a rule that defines how different representations of the same things correspond.  
Patterns define a set of related elements to be mapped.   
Types may be defined to be a representation (data about) a concept.  
Map rules define how elements in each of the patterns are mapped, bidirectionally.  
A maps utilizing more specific types subsumes maps for more general types.  
Within a Mapping a Property Generalization Rule specifies that one pattern property is the subset of another.

#### Direct Supertypes

[Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f), [Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Mapping

#### Associations

195961665.jpg has map rule : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956) states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

Map rule that is asserted by a mapping.

### Class Property End

An end of a map rule where the end maps a property of the mapping pattern, the elements bound to that property for each match of the pattern will be mapped.

#### Direct Supertypes

[Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

**package** SIMF Conceptual Model::Mapping

#### Associations

-58492342.jpg maps property : [Pattern property](#_4d83e476040c7444758dda440d3096fc) [1]

Property that defines a set of elements to map to the other side of the mapping rule based on a pattern property. The set of elements shall be those bound to the property on evaluation of the mapping.

### Association Representation

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-58492342.jpg represented by : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

The representation of a concept in a more specific form

-58492342.jpg represents rule : [Representation Rule](#_e2101d86d43ebb1c6717af7a9f48ebc1)

### Class Representation Rule

A representation rule states that the <represented concept> has a representation defined by the <represented by> type.  
Representation rules are used to filter Map Rules such that only represented concepts may be mapped.  
A representation is usually complimented with one or more mapping rules.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Mapping

#### Attributes

-1626698621.jpg condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

Condition that must be TRUE for the mapping rule to apply.

-1626698621.jpg map all : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

Implies a direct mapping between instances of the types in both directions.  
<map all> is equivalent to a mapping with a rule mapping properties of each type but is lower precedence than other mappings - if types have a more specific map it will apply first.

#### Associations

-58492342.jpg represented concept : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]

A more general or abstract concept that is being represented.

-58492342.jpg represented by : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

The representation of a concept in a more specific form

### Association Represented Concept

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-58492342.jpg represented concept : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]

A more general or abstract concept that is being represented.

-58492342.jpg concept rule : [Representation Rule](#_e2101d86d43ebb1c6717af7a9f48ebc1)

### Association to rule

**package** SIMF Conceptual Model::Mapping

#### Association Ends

-575849932.jpg to : [Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]

One end of a mapping, to be used for more abstract end.

-575849932.jpg map from : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [0..1]

Mapping rule owning a "to" end.

### Class Type End

An end of a map rule where the end maps to a relationship, the instances of the relationship between the properties will be mapped.  
The relationship must be defined in one of the <mapped pattern>s of the mapping.

#### Direct Supertypes

[Map Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

**package** SIMF Conceptual Model::Mapping

#### Associations

-58492342.jpg maps relationship : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [1]

Defines a set of relationships to map to the other side of the mapping rule based on a relationship contained in the pattern.

## SIMF Conceptual Model::Patterns

Templates are patterns that may then be expressed as instances of the pattern.

### Diagram: Patterns



1. Patterns

### Association Map rules

**package** SIMF Conceptual Model::Patterns

#### Association Ends

195961665.jpg has map rule : [Map Rule](#_63d69e49de8214503f0947e7f9dbc652) [\*]

Map rule that is asserted by a mapping.

195961665.jpg map rule of : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [1]

Mapping containing a map rule.

### Class Pattern

A pattern represents a set of assertions true about individuals or sets of individuals qualified by pattern properties.  
   
The structure of the pattern is defined by the bindings and asserted (sub) situations (including relationships) that are asserted by the pattern.  
  
In many cases the relationships and rules defined for a pattern will reference pattern properties. These relationships will hold for instances of the pattern where things are bound to the pattern properties.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259), [Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Patterns

#### Associations

195961665.jpg ownes property : [Pattern property](#_4d83e476040c7444758dda440d3096fc) [\*] *Subsets*: states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) *Redefines*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

195961665.jpg ownes relationship : [Pattern Relationship](#_ef18e024624b062995d1bb5acd223c15) [\*] *Subsets*: has assertion:[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

195961665.jpg : [Pattern Property Subset](#_f81964c3ea1d96486073fdffcbb6fe29) *Subsets*: states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

### Association Pattern Properties

**package** SIMF Conceptual Model::Patterns

#### Association Ends

195961665.jpg ownes property : [Pattern property](#_4d83e476040c7444758dda440d3096fc) [\*] *Subsets*: states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

195961665.jpg has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1] *Subsets*: states:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

### Class Pattern property

A pattern property is a part of a pattern that provides a contextual property within that pattern for rules and relationships to be bound to.  
A pattern property is a placeholder for all or a subset of the instances of the properties type as defined by the quantifier.  
Properties of a relationship may be bound to a pattern property where the type of the pattern property is compatible with the type of the relationship's property type.

#### Direct Supertypes

[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

**package** SIMF Conceptual Model::Patterns

#### Attributes

-1626698621.jpg quantifier : [Quantifier](#_b82d09af5a9584abc6560d3bfb03e524) [0..1]

A property may define a quantification within a pattern. The quantifier defines the set of things that will populate the pattern property for all instances of the pattern.  
  
e.g. for all people p: People is the context and P is the quantified property. In SIMF the quantified property would typically be named <quantifier> <type>. So the above quantified property would be named "all people". The quantified property will be asserted to have the quantified type.  
  
Quantifier should not be stated at the same time as strength.

-1626698621.jpg is composite part : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [0..1]

True if the property represents an internal part of the pattern. False if the property is external to the pattern and must be provided by connection to another pattern.

-1626698621.jpg is boundary part : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) [0..1]

True if the property is on the boundary of the pattern and connectible (may have relationships) external to the pattern. e.g. "Port"

-1626698621.jpg has strength : [Pattern Element Strength](#_8e8a996acf04b8f7f4fbad8fd901f2c2) [0..1]

<strength> defines the behavior of an element with respect to a pattern - how it impacts the selection, evaluation or assertion of the pattern.

-1626698621.jpg explicit : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

If true, Element must be explicitly asserted as the indicted type, not derived or inferred from a supertype or super property.

#### Associations

-58492342.jpg condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [\*]

Condition that must be true for all things bound to a pattern property.

195961665.jpg has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1] *Redefines*: property of:[Structured Type](#_50241f5936e61055293ca95f860768d8) stated by:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

-58492342.jpg computation : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [0..1]

Computation computes a value for the pattern element based on the expression applied to the related property or relationship.   
Where computation is used inverse mapping is not specified - any inverse mapping is implementation specific.

### Class Pattern Property Subset

In a pattern or mapping rule, defines a property that represents a subset of another property. The subset may be constrained by a more specific type, expressions or required cardinalities.

#### Direct Supertypes

[Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab)

**package** SIMF Conceptual Model::Patterns

#### Attributes

-1626698621.jpg defaut : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

True if the subset should be populated only if no other subsets have been populated.

#### Associations

195961665.jpg : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) *Redefines*: stated by:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

### Class Pattern Relationship

A pattern relationship is part of the definition of a pattern, it extends a basic relationship in that it adds properties to determine the effect the relationship has on pattern instances.

#### Direct Supertypes

[Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166)

**package** SIMF Conceptual Model::Patterns

#### Attributes

-1626698621.jpg has strength : [Pattern Element Strength](#_8e8a996acf04b8f7f4fbad8fd901f2c2)

<strength> defines the behavior of an element with respect to a pattern - how it impacts the selection, evaluation or assertion of the pattern.

-1626698621.jpg explicit : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

If true, Element must be explicitly asserted as the indicted type, not derived or inferred from a supertype or super property.

#### Associations

195961665.jpg has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1] *Redefines*: holds within:[Context](#_66d62b068053cee3464e1e03e6035eed) stated by:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

### Association Pattern Relationships

**package** SIMF Conceptual Model::Patterns

#### Association Ends

195961665.jpg ownes relationship : [Pattern Relationship](#_ef18e024624b062995d1bb5acd223c15) [\*] *Redefines*: holds within: [Context](#_66d62b068053cee3464e1e03e6035eed) stated by: [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

195961665.jpg has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1] *Redefines*: holds within: [Context](#_66d62b068053cee3464e1e03e6035eed) stated by: [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

#### Enumeration Pattern Element Strength

Pattern Element Strength values define the behavior of an element with respect to a pattern - how it impacts the selection, evaluation or assertion of the pattern.

package SIMF Conceptual Model::Patterns

public enum Pattern Element Strength

{Match, Assert, Default, Exists}

##### Literals

-1758359570.jpg Match

Match is used in query and mapping patterns, all elements of the classified type that match the pattern are selected as instances of the pattern.  
Match may be considered a qualified "All". Match does not assert the existence of something, it determines the existence of a pattern match such that other assertions may be made.  
Relationships between properties with <quantifier>=Match must hold between the matched properties for the pattern to match.

-1758359570.jpg Assert

The element will be asserted as required for a valid pattern. Assert is the default.

-1758359570.jpg Default

The element will be asserted only if no other values are asserted within the pattern or as pre-existing assertions.

-1758359570.jpg Exists

Existing element that will be used to compute other values but does not otherwise impact the pattern.

#### Enumeration Quantifier

The set of quantifiers for pattern variables

package SIMF Conceptual Model::Patterns

public enum Quantifier

{None, There Exists, Exactly One, Most, All}

##### Literals

-1758359570.jpg None

A quantifier where no instance of the type may fill the role. E.g. "there may not exist".

-1758359570.jpg There Exists

The existential quantifier - at least one. A logical "supertype" of "One of" and "Most"

-1758359570.jpg Exactly One

The existential quantifier limited to exactly one of a potentially larger set

-1758359570.jpg Most

A stratified existential quantifier with a default for a "typical" value - example: <Most> people have 2 arms.  
For logics that do not support "most", most may be interpreted as "There exists".

-1758359570.jpg All

The universal quantifier - the quantified property is a stand-in for all elements of the existent of the quantified type

## SIMF Conceptual Model::Records

Records represent structured data, usually used for logical or physical data models, not conceptual models.

### Diagram: Records



1. Records

### Class Record

A record of the condition of an entity at a point in time - this includes facts, speech acts and DBMS records.   
Records are typically used in data representations, not conceptual models.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259)

**package** SIMF Conceptual Model::Records

#### Associations

-1808253052.jpg : [Record Type](#_d2ebf1b96697234b6aef9b3bfac15784) [1] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

### Class Record Type

Type of the record of the condition of an entity at a point in time - this includes facts, speech acts and DBMS records.   
A record type may involve variant and invariant types as variables. Those that are enumerated in a "uniqueness constraint" are invariant (independent variables) uniquely identify the situation which is the subject of the fact type where as the other variables may change over time (dependent variables).  
Record types may be grounded in atomic relations by using invariant conditions.  
Record types represent typical "data structures".

#### Direct Supertypes

[Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Records

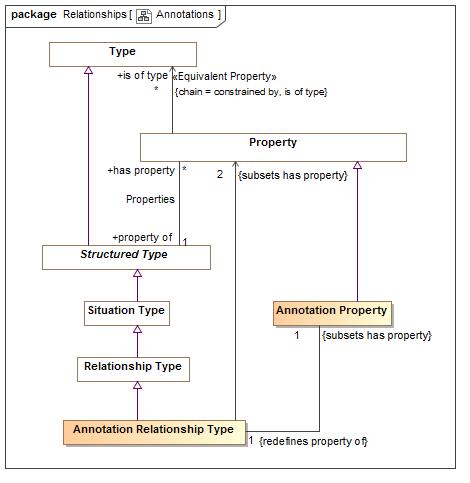
#### Associations

-1808253052.jpg : [Record](#_8b38efa9c56da3bc8ecb501e56419e41) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

## SIMF Conceptual Model::Relationships

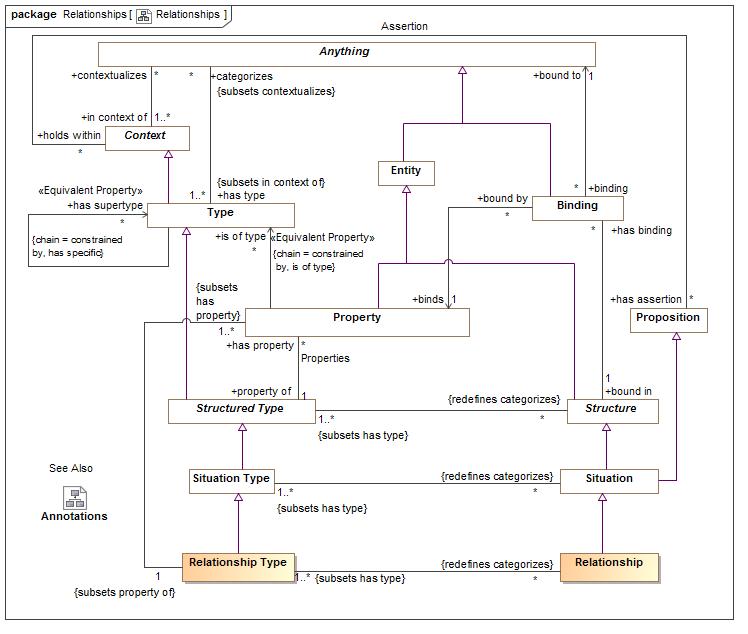
Relations are primitive facts about anything, relating at least 2 individuals through the roles they play.

### Diagram: Annotations



1. Annotations

### Diagram: Relationships



1. Relationships

Relations are atomic situations that bind 2 or more properties as a fact.

### Class Annotation Property

An annotation property is a specialization of property where the referenced elements represent metadata about the related proposition, structure or information (or model element) rather than a fact or condition of the domain being represented.  
For an annotation property, <is of type> describes instances of the structured type for which the property is defined.  
Typical uses of annotations include provenance of information, when a record was created, etc.

#### Direct Supertypes

[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

**package** SIMF Conceptual Model::Relationships

#### Associations

-1808253052.jpg : [Annotation Relationship Type](#_7a2b4b15c2377efe429d3d9e5b30b859) [1] *Redefines*: property of:[Structured Type](#_50241f5936e61055293ca95f860768d8)

### Class Annotation Relationship Type

An annotation relationship type is a specialization of relationship type where the relationship represents metadata about the related element, structure or information (or model element) rather than a fact or condition of the domain being represented.  
  
Typical uses of annotations include provenance of information, when a record was created, etc.

#### Direct Supertypes

[Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a)

**package** SIMF Conceptual Model::Relationships

#### Associations

-1808253052.jpg : [Annotation Property](#_630a40beaf4bbbac635cfcacaaf353d1) [1] *Subsets*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-58492342.jpg : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [2] *Subsets*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

### Class Relationship

Any proposition stated within a context. Each proposition <has Binding>s to the variables of the relationship type that uniquely identify the relationship instance and do not change in the life of the relationship.

#### Direct Supertypes

[Situation](#_bed9725f6bf8a4fff6fd1087850b7259)

**package** SIMF Conceptual Model::Relationships

#### Associations

-1808253052.jpg : [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

### Class Relationship Type

Any type of proposition or fact that may be asserted within a context that is true or false within that context. Each relationship type has a number of properties bindiungs of which uniquely identify a relationship and do not change.   
  
A relationship may be true of false within its context but is atomic in its truth value.  
Relationships may participate in (be bound to) other relationships and as such bindings involving relationships may change over time.  
  
Note that the properties of a relationship (its identity) are differentiated from other relationships that reference it, such other relationships have their own identity, truth value and life cycle.  
  
The properties of a relationship represent the verb phrases connecting the relationship with the related types.

#### Direct Supertypes

[Situation Type](#_c568596a9f8653e157460ae519e96906)

**package** SIMF Conceptual Model::Relationships

#### Associations

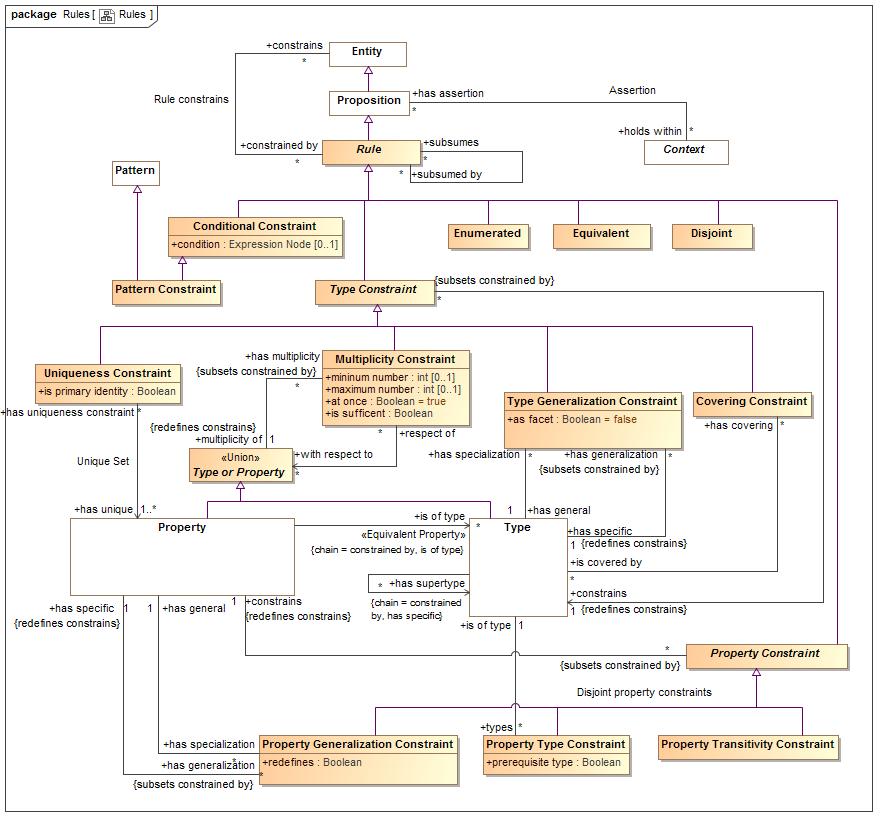
-1808253052.jpg : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-1808253052.jpg : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*] *Subsets*: has property:[Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

## SIMF Conceptual Model::Rules

Rule define constraints or behaviors that must be applied in specified context.

### Diagram: Rules



1. Rules

### Class Conditional Constraint

A rule with a general expression as a condition

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [0..1]

Condition that must be TRUE for the rule to "fire". All other values are FALSE.

### Class Covering Constraint

A constraint that the extent (<categorizes> things) of the <constrains> type is equivalent to the union of the extents of the <is covered by> types.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Associations

-1808253052.jpg is covered by : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]

A type covered by a covering constraint.  
  
The <constrains> type must be a direct supertype of all <is covered by> types.

### Class Disjoint

Rule that different elements defined in models do not and may not denote any of the same set of real or possible world entities.  
When applied to a context (including types) all elements contextualized are included in the set of disjoint individuals.  
Each context the rule <constrains> is disjoint.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Class Enumerated

The contextualized elements of the <constrains> context is a closed (enumerated) set, it can not be extended. A.K.A. "Closed World Assumption". Elements may not be asserted by any context other than the one specified.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Class Equivalent

Rule that different elements defined in models denote the same set of real or possible world entities. Each context the rule <constrains> is equivalent - has the same extent.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

### Association Generalizations

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg has general : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

The general type in the Generalization rule.

-1808253052.jpg has specialization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]

Specialization rules for a type.

### Class Multiplicity Constraint

A Multiplicity constraint constrains the number of bindings <multiplicity of> properties may have in a particular instance of a type.   
  
The number of instances binding to the property for the set of instances bound to <with respect to> shall be limited by the minimum and maximum number of the multiplicity.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg mininum number : [int](#_0d30278207cac92be6fa561506a22f92) [0..1]

Minimum number in a set as constrained by a multiplicity.

-1626698621.jpg maximum number : [int](#_0d30278207cac92be6fa561506a22f92) [0..1]

Maximum number in a set as constrained by a multiplicity.

-1626698621.jpg at once : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = true

When at once is true, the constraint applies for each snapshot in time but not across snapshots (e.g. a car can have at most one driver at a time). When at once is false the constraint applies across all time (e.g. a person has exactly one birth mother across all time).

-1626698621.jpg is sufficent : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

One of the set of sufficient conditions that will infer the type designated in <constrains>.

#### Associations

-1808253052.jpg multiplicity of : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

The type or property that is the subject of a multiplicity constraint.

-58492342.jpg with respect to : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [\*]

One or more types or properties that define the <from> side of a multiplicity.   
  
Where with respect to is undefined and <multiplicity of> is a property, all properties that are <property of> the same structured type as <multiplicity of> shall be considered the set of <with respect to> properties. I.e. all the "other ends" of a relationship.  
  
<with respect to> provides for complex multiplicities across n-ary situations and relationships.

### Association Multiplicity Perspective

**package** SIMF Conceptual Model::Rules

#### Association Ends

-58492342.jpg with respect to : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [\*]

One or more types or properties that define the <from> side of a multiplicity.   
  
Where with respect to is undefined and <multiplicity of> is a property, all properties that are <property of> the same structured type as <multiplicity of> shall be considered the set of <with respect to> properties. I.e. all the "other ends" of a relationship.  
  
<with respect to> provides for complex multiplicities across n-ary situations and relationships.

-58492342.jpg respect of : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]

Multiplicity constraints using a property or type as a <with respect to> reference.

### Association Multiplicity Target

Type with references to the "to type", normally a role or a situation type - but can be any type.

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg multiplicity of : [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c) [1]

The type or property that is the subject of a multiplicity constraint.

-1808253052.jpg has multiplicity : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]

Multiplicity constraint of a type or property.

### Class Pattern Constraint

A pattern rule asserts that the pattern is true for the pattern variables identified.

#### Direct Supertypes

[Conditional Constraint](#_3d425949001fb1cb0502a6157c8cf51e), [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f)

**package** SIMF Conceptual Model::Rules

### Class Property Constraint

Constraints that constrain properties.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Associations

-1808253052.jpg constrains : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

### Class Property Generalization Constraint

A property may be asserted to be the specialization of another. The specialized property represents a set that is the same as or a subset of the more general property.  
  
Where <redefines> is true the more specific property subsumes the more general property in the definition context. In this case the more general and more specific sets are equivalent.   
  
A more specific property shall be consistent with the property it specializes, but it may add specific constraints or other details that are particular to instances of the specializing context that do not contradict constraints in the general context.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg redefines : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

Defines the generalization as a redefinition, subsuming the more general property.  
  
Where <redefines> is true the more specific property subsumes the more general property in the definition context. In this case the more general and more specific sets are equivalent. A property may be redefined multiple times, as long as it is unambiguous which definition applies for a particular instance.  
  
Where <redefines> is false or not defined the more specific property represents a subset of the more general property.

#### Associations

-1808253052.jpg has general : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]

The more general property as part of a property generalization

-1808253052.jpg has specific : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

The more specific property as part of a property generalization

### Association Property Generalizations

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg has specific : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

The more specific property as part of a property generalization

-1808253052.jpg has generalization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

Specialization rules for a property.

### Association Property Specializations

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg has general : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

The more general property as part of a property generalization

-1808253052.jpg has specialization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

Generalization rules of a property

### Class Property Transitivity Constraint

A transitive property interlinks two individuals A and C whenever it interlinks A with B and B with C for some individual B.   
For example "larger than" is transitive in that if Joe is larger than Sue and Sue is Larger then Sam, then Joe is larger than Sam.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

### Association Property Type

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

A required type of a thing bound to a property.  
Note that the type may be inferred based on the value of <prerequisite type>.

-1808253052.jpg types : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*] *Redefines*: constrains: [Entity](#_eb8398b5a178c638b98597120ec51c4d)

Properties typed by a type

### Class Property Type Constraint

A property type rule defines the type(s) of a property.  
All elements bound to a property must have the type <is of type>. <is of type> may be pre-existing or inferred based on the value of <prerequisite type>.  
Note that Property Type Constraint is a rule independent of the definition of a property to allow for the type of a property to be refined in a more restrictive context.

#### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg prerequisite type : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

If true, <is of type> is a prerequisite - the bound thing must be of the given type for the property to be bound. A non prerequisite type will cause a binding to infer <is of type>, provided all prerequisite types have been satisfied.

#### Associations

-1808253052.jpg is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

A required type of a thing bound to a property.  
Note that the type may be inferred based on the value of <prerequisite type>.

### Class Rule

A rule is a proposition that constrains one or more entities by limiting possible conditions or producing some effect.  
Note that rules may or may not be defined in the same context that they hold within or constraint. This support the "open world assumption" that a rule may be asserted outside of the scope of the rule.

#### Direct Supertypes

[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

**package** SIMF Conceptual Model::Rules

#### Associations

-1808253052.jpg constrains : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]

The entity or entities constrained by a rule.   
Where a rule constrains a context, all things contextualized by the context shall be subject to the rule.  
Where there are no <constrains> for a rule, the rule applies globally - to the universal context.

-1808253052.jpg subsumes : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]

When rule subsumes another the subsumed rule will not apply (fire) if the <subsumed by> rules applies (fires).  
Where rules are also patterns, a rule may specialize another which will subsume the specialized rule as well as include the generalized rule parts as parts of the specialized rule.

-1808253052.jpg subsumed by : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]

When rule is <subsumed by> another the subsumed rule will not apply (fire) if the <subsumed by> rules applies (fires).

### Association Rule constrains

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg constrains : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]

The entity or entities constrained by a rule.   
Where a rule constrains a context, all things contextualized by the context shall be subject to the rule.  
Where there are no <constrains> for a rule, the rule applies globally - to the universal context.

-1808253052.jpg constrained by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]

Rules applying to an entity.

### Association Specializations

**package** SIMF Conceptual Model::Rules

#### Association Ends

-1808253052.jpg has specific : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

The specific type in a generalization rule.

-1808253052.jpg has generalization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]

Generalization rules for a type

### Class Type Constraint

A constraint of a type, including Relationships types.

#### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

**package** SIMF Conceptual Model::Rules

#### Associations

-58492342.jpg constrains : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

### Class Type Generalization Constraint

A Type Generalization Constraint is a taxonomic relationship between a more general <has general> type and a more specific <has specific> type. Each instance of the specific type is also an instance of the general type. The specific type inherits the properties and rules of the more general type.   
The extent (<categorizes> property) of the specific type is the same as or a subset of the extent of the more general type.  
Note that "multiple inheritance" is supported.

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg as facet : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = false

If <as facet> is true the specialized type is "non rigid" with respect to the general (rigid) type - that is the <has specific> type may change over the lifetime of instances of the <has general> type. The <has specific> type will be inferred to be a Facet. e.g. "Registered voter".   
  
If <as facet> is false the specialized type is "rigid" with respect to the general type - that is the <has specific> type may not change over the lifetime of instances of the base type. e.g. a "Tree" is always a "Plant".  
  
[BFO] Non rigid sortal

#### Associations

-1808253052.jpg has specific : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1] *Redefines*: constrains:[Entity](#_eb8398b5a178c638b98597120ec51c4d)

The specific type in a generalization rule.

-1808253052.jpg has general : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

The general type in the Generalization rule.

### Class Type or Property

The union of types and variables such that they can both be generalized and used in constraints.

**package** SIMF Conceptual Model::Rules

#### Associations

-1808253052.jpg has multiplicity : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

Multiplicity constraint of a type or property.

### Association Unique Set

**package** SIMF Conceptual Model::Rules

#### Association Ends

-58492342.jpg has unique : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

The set of involved properties within a type that uniquely identify an individual.

-58492342.jpg has uniqueness constraint : [Uniqueness Constraint](#_982e84b7afc784b4d0aa763204953a3d) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

Uniqueness constraints for a property.

### Class Uniqueness Constraint

A constraint that, within the type the rule applies to, the set of instances bound to the set of types in the "has unique" relation must be unique and serves to define the "identity" of each individual. A Uniqueness constraint that is not constrained by a situation indicates that the property may have only one member.  
Note: Uniqueness may be used to define a "key".

#### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

**package** SIMF Conceptual Model::Rules

#### Attributes

-1626698621.jpg is primary identity : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)

A uniqueness constraint that can be interpreted as a "primary key", the identity of an entity.

#### Associations

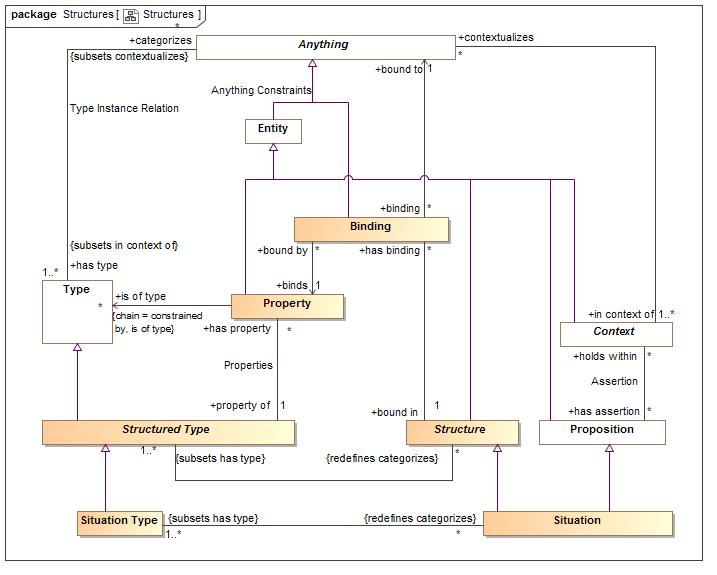
-58492342.jpg has unique : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

The set of involved properties within a type that uniquely identify an individual.

## SIMF Conceptual Model::Structures

Structures provide for things with properties (or variables) that are part of their logical or physical composition.  
  
Structures include situations where situations are arrangements of individuals, assertions and the relations and assertions between them over a timespan. Any "condition" that exists is a situation - including relations.

### Diagram: Structures



1. Structures

A set of role bindings about a specific subject

### Class Binding

A binding binds a particular thing to a property in a structure (including relationships and records). A.K.A. the value of a property.  
  
For relations (which are structures), this is also known as an "association end".  
  
Where a binding is to an expression evaluation, the binding shall evaluate to the evaluation of the expression.  
  
The bound to thing must conform with the <is of type> type of the property. If the bound individual conforms to the "requires type" of the property, the <is of type> of the thing will be asserted.  
  
The type of the <bound in> structure must (directly or indirectly) have the type the "binds" properties <property of> type.

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Structures

#### Associations

-58492342.jpg binds : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]

The property a binding binds a thing to.

-58492342.jpg bound to : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

The thing(s) bound to a property in a specific situation.

-1808253052.jpg bound in : [Structure](#_8c517cf1950741c0f89edebf828214cc) [1]

The structure asserting the binding.

### Association Bound in situation

**package** SIMF Conceptual Model::Structures

#### Association Ends

-1808253052.jpg has binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]

Bindings asserted in a structure.

-1808253052.jpg bound in : [Structure](#_8c517cf1950741c0f89edebf828214cc) [1]

The structure asserting the binding.

### Association Bound individual

**package** SIMF Conceptual Model::Structures

#### Association Ends

-58492342.jpg bound to : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]

The thing(s) bound to a property in a specific situation.

-58492342.jpg binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]

Bindings a thing is bound to.

### Association Bound property

**package** SIMF Conceptual Model::Structures

#### Association Ends

-58492342.jpg bound by : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]

Bindings referencing a property.

-58492342.jpg binds : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]

The property a binding binds a thing to.

### Association Properties

**package** SIMF Conceptual Model::Structures

#### Association Ends

-1808253052.jpg has property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [\*]

A property of a structured type such that there may be bindings of a thing to instances of the structured type with reference to the property which defines the semantics of the bound thing withing the context of the structure.

-1808253052.jpg property of : [Structured Type](#_50241f5936e61055293ca95f860768d8) [1]

Structured type for which a property is relevant.

### Class Property

A property defines way in which instances of a type play a roles in a structured type (including relationships). Sometimes called a variable, argument or role.  
  
In a conceptual model the <property> typically represents the "verb phrase" defining how instances of the involved type play a role an a situation.  
  
In a data structure the property is a "slot" of a record.  
  
So that constraints of a type flow to relationships involving that type: All propositions that hold within a type referenced by <is of type> hold within the structured type referenced by <property of>. I.e. the structured type is in the context of the types of its properties.  
  
In a function, a property is a function argument.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d), [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c)

**package** SIMF Conceptual Model::Structures

#### Associations

-1808253052.jpg property of : [Structured Type](#_50241f5936e61055293ca95f860768d8) [1]

Structured type for which a property is relevant.

-1808253052.jpg : [Property Constraint](#_91be190e8014514a597300b286148d3e) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

-1808253052.jpg has specialization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*]

Generalization rules of a property

-1808253052.jpg has generalization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

Specialization rules for a property.

-58492342.jpg is of type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]

A type of instances bound to a property. Also known as the "range" of a property.  
If asserted the property rule shall be owned and asserted by the properties <property of> type.

-1808253052.jpg : [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a) [1] *Subsets*: property of:[Structured Type](#_50241f5936e61055293ca95f860768d8)

### Class Situation

An identifiable arrangement of individuals, assertions and the relations and assertions between them over a timespan. Any condition, relationship or occurrence is a situation.  
  
Things are involved in a situation through bindings to situation types.  
  
Syn. "State of affairs"

#### Direct Supertypes

[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151), [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1), [Structure](#_8c517cf1950741c0f89edebf828214cc)

**package** SIMF Conceptual Model::Structures

#### Associations

-1808253052.jpg : [Situation Type](#_c568596a9f8653e157460ae519e96906) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

### Class Situation Type

A type of identifiable arrangement of individuals, assertions and the relations between them over a timespan. Elements of a situation may change over time.  
  
The roles things play in a situation are identified as properties of the situation type.  
  
Syn. Type of a state of affairs.  
  
[BFO] Endurant

#### Direct Supertypes

[Structured Type](#_50241f5936e61055293ca95f860768d8)

**package** SIMF Conceptual Model::Structures

#### Associations

-1808253052.jpg : [Situation](#_bed9725f6bf8a4fff6fd1087850b7259) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

### Class Structure

A structure of elements where the structure is represented by a set of bindings that bind individuals to the structure with reference to properties.  
  
A structure may by conceptual, logical or physical - only requiring that something may have some form of properties.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Structures

#### Associations

-1808253052.jpg : [Structured Type](#_50241f5936e61055293ca95f860768d8) [1..\*] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

-1808253052.jpg has binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]

Bindings asserted in a structure.

### Class Structured Type

A type that has properties such that instances (structures), may bind things to structures based on properties.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Structures

#### Associations

-1808253052.jpg : [Structure](#_8c517cf1950741c0f89edebf828214cc) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

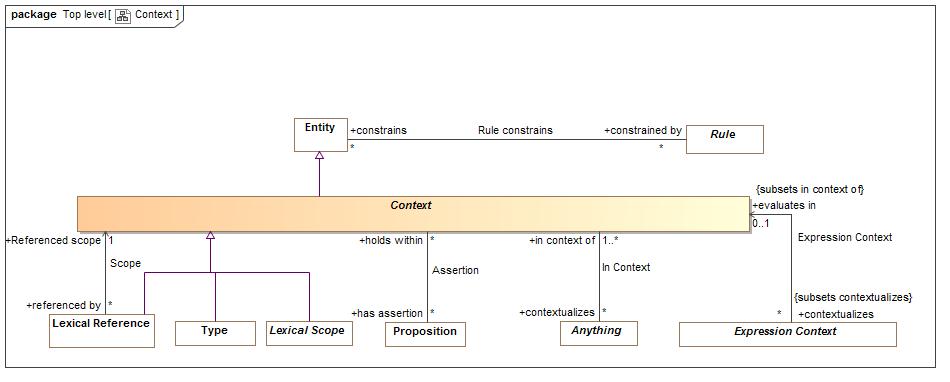
-1808253052.jpg has property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [\*]

A property of a structured type such that there may be bindings of a thing to instances of the structured type with reference to the property which defines the semantics of the bound thing withing the context of the structure.

## SIMF Conceptual Model::Top level

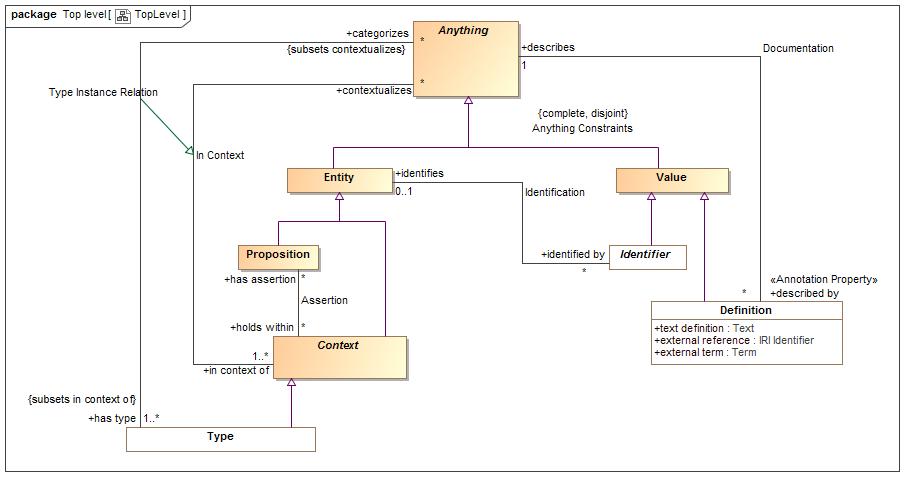
The top level objects provide the foundation for all objects in the model

### Diagram: Context



1. Context

### Diagram: TopLevel



1. TopLevel

### Class Anything

Any thing or value that does or may exist in any possible world. Anything is the supertype of all types and may therefore participate in unbounded relations.  
Instances of anything are refered to as a "thing" in this model.

**package** SIMF Conceptual Model::Top level

#### Associations

-1808253052.jpg described by : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [\*]

An informal description of something.

-1808253052.jpg has type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)

A type that holds for something.  
Things may have multiple types and these types may change over time.

-1808253052.jpg in context of : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*]

A <Context> that contextualizes <Anything> making what it <contextualizes> the thing and subject to the propositions referenced by <has assertion> of the context.  
A thing may be <in context of> one or more context.

195961665.jpg defined in : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)

Lexical scope defining model elements.

195961665.jpg stated by : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [0..1] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) holds within:[Context](#_66d62b068053cee3464e1e03e6035eed)

<stated by> is a lexical scope that both defines and asserts a model element.

### Association Assertion

The relationship between a context and the propositions asserted within that context.

**package** SIMF Conceptual Model::Top level

#### Association Ends

-1808253052.jpg has assertion : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) holds within:[Context](#_66d62b068053cee3464e1e03e6035eed)

Proposition that is asserted (must be true) for anything contextualized by a context.

-1808253052.jpg holds within : [Context](#_66d62b068053cee3464e1e03e6035eed) [\*] *Subsets*: defined in:[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) holds within:[Context](#_66d62b068053cee3464e1e03e6035eed)

Context in which a proposition is asserted (required to be true). Anything contextualized by the context is subject to the proposition.

### Class Context

A <Context> is a grouping of <Anything> where the set of grouped things is defined by <contextualizes>.   
A <Context> also <has assertion>s that hold for all things the context <contextualizes>.  
Subtypes of <Context>, such as <Type> ascribe more semantics to the context as well as the things it <contextualizes>.  
  
A context provides a binding between a set of propositions and the things those propositions apply to.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Top level

#### Associations

-1808253052.jpg contextualizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]

The set of things contextualized by a <Context>, or "in" the <Context> and therefor subject to the <has assertion> propositions of the <Context>.

-1808253052.jpg has assertion : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*]

Proposition that is asserted (must be true) for anything contextualized by a context.

### Class Entity

An entity is any identifiable thing other than values, this includes individuals, types, axioms, situations, speech acts, information structures, etc.  
Entities have some kind of identity and may have identifiers.

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Top level

#### Associations

-1808253052.jpg identified by : [Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*]

An identifier for an <Entity>.

-58492342.jpg has preferred term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8) [0..1] *Subsets*: identified by:[Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)

Default term to use for an entity.  
Where multiple terms are preferred in differing context any method for selecting the most preferred term is implementation specific and not specified by this standard.

-1808253052.jpg constrained by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]

Rules applying to an entity.

### Association In Context

The association between a context and the set of concepts contextualized by that context.

**package** SIMF Conceptual Model::Top level

#### Association Ends

-1808253052.jpg contextualizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]

The set of things contextualized by a <Context>, or "in" the <Context> and therefor subject to the <has assertion> propositions of the <Context>.

-1808253052.jpg in context of : [Context](#_66d62b068053cee3464e1e03e6035eed) [1..\*]

A <Context> that contextualizes <Anything> making what it <contextualizes> the thing and subject to the propositions referenced by <has assertion> of the context.  
A thing may be <in context of> one or more context.

### Class Proposition

A proposition is statement, or condition with a truth value that can be determined or asserted.  
All "facts", statements, speech acts, relationships and rules are propositions.  
Propositions may be asserted to be true within a context which they <holds within>.

#### Direct Supertypes

[Entity](#_eb8398b5a178c638b98597120ec51c4d)

**package** SIMF Conceptual Model::Top level

#### Associations

-1808253052.jpg holds within : [Context](#_66d62b068053cee3464e1e03e6035eed) [\*]

Context in which a proposition is asserted (required to be true). Anything contextualized by the context is subject to the proposition.

### Association Term Preference

**package** SIMF Conceptual Model::Top level

#### Association Ends

-58492342.jpg has preferred term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8) [0..1]

Default term to use for an entity.  
Where multiple terms are preferred in differing context any method for selecting the most preferred term is implementation specific and not specified by this standard.

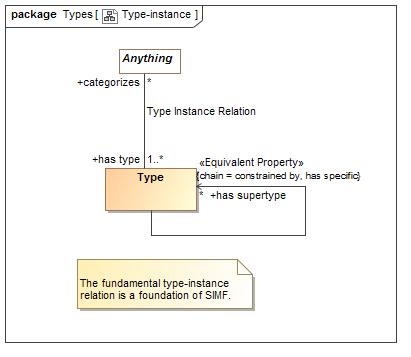
-58492342.jpg preferred for : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]

The entity a term is preferred for.

## SIMF Conceptual Model::Types

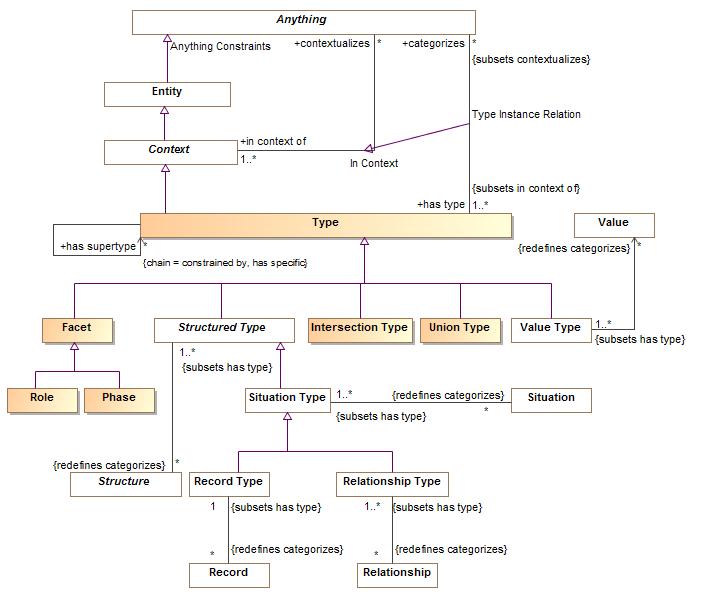
Type provide for ways to categorize anything based on what it is, the roles it plays or the phases it may be in.  
Something may be categorized by any number of types (multiple classification assumption).

### Diagram: Type-instance



1. Type-instance

### Diagram: Types



1. Types

### Class Facet

A facet is a "mix in" type that defines some aspect of something but does not represent the "fundamental" type or identity of that thing, but some potentially transient role, phase or other way to classify it. Something must have at least one type that is not a facet.  
Facets do not define independent identity of the referent but technology implementations may create independent objects to represent a facet.  
An instance of a facet must also have a type that is not a facet to provide the identity of the instance.  
  
The type(s) a facet may categorize may be constrained by setting the "as facet" property of Generalization. E.g. Policeman is a role of a person.  
  
BFO: Non-rigid sortal

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

### Class Intersection Type

An intersection is a type that has an extent which is the complete intersection of the extents of all supertypes. Intersection is a stronger statement than a subtype as a subtype may not be a complete intersection.  
  
[MathWorld] The intersection of two sets A and B is the set of elements common to A and B. This is written A intersection B, and is pronounced "A intersection B" or "A cap B."

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

### Class Phase

A phase (or state) is a characteristic of something that exists for limited time(s). Something takes on or looses a phase as a result of some event. E,g, Teenager, living, closed invoice.

#### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013)

**package** SIMF Conceptual Model::Types

### Class Role

A role is a type facet type that defines a specific purpose or behavior of a class of things. E.g. teacher, policeman, employer.

#### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013)

**package** SIMF Conceptual Model::Types

### Class Type

A <Type> is a categorization of <Anything> based on specific criteria. The specific criteria may or may not be formalized in a model.  
A <Type> <categorizes> a set of <Anything> which comprises the "extent" of the type.  
A <Type> is a <Context> where the things it <categorizes> are <in the context> of the <Type>.

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed), [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151), [Type or Property](#_0ae2de066cc60f0d0ea185e32c2ee88c)

**package** SIMF Conceptual Model::Types

#### Associations

-1808253052.jpg categorizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*] *Subsets*: contextualizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

The set of things described by a type, the "extent" of the type.  
The thing a type <categorizes> is subject to the <has assertion> propositions of the type.

-1808253052.jpg has generalization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*] *Subsets*: constrained by:[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

Generalization rules for a type

-1808253052.jpg has specialization : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]

Specialization rules for a type.

-1808253052.jpg types : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*]

Properties typed by a type

-58492342.jpg has supertype : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*]

Supertypes(s) of a type as defined by generalization rules.  
  
If asserted the generalization rule shall be owned and asserted by the subtype.

-1808253052.jpg has covering : [Covering Constraint](#_507049575ebfa9f535e8f25db14a0760) [\*]

Covering constraints of a type.

### Association Type Instance Relation

The relation between a type and the concepts that type categorizes, the instances

### Direct Supertypes

[In Context](#_52c887644007b8e51a1f6e976113707a)

**package** SIMF Conceptual Model::Types

#### Association Ends

-1808253052.jpg categorizes : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]

The set of things described by a type, the "extent" of the type.  
The thing a type <categorizes> is subject to the <has assertion> propositions of the type.

-1808253052.jpg has type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1..\*]

A type that holds for something.  
Things may have multiple types and these types may change over time.

### Class Union Type

A Union is a type that has an extent which is the complete union of the extents of all types that specialize the Union.   
  
[MathWorld] Given two sets A and B, the union is the set that contains elements or objects that belong to either A or to B or to both. We write A È B

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Types

## SIMF Conceptual Model::Values

The values package defines the concepts of values and quantities expressed in units.

Values may be differentiated from entities in that values have no independent lifetime or "identity" other than the value its self. E.g. the number 5 "just is" and can't be changed. Properties and relations referencing values can, of course, change but the values are constant.

The failure to properly express units in data models often results in errors, inefficiencies and risk. Translation and federations between models, schema and data sources that is not cognizant of the units used would be even more error prone and risky. For example, what does “Speed limit 50” mean? For these reasons the SIMF language provides specific support for specifying quantity kinds and unit types in conceptual, logical and physical models. The SIMF mapping rules may then perform the appropriate unit conversions.

The foundation of information specification in SIMF at all levels is the type system. Types specified for all properties and relations involving values must match the types of the related values. The concepts of units and values as defined in "VIM" [JCGM 200-2008] is used as the basis for defining the types used in SIMF to guarantee type safety of quantities across different representations. Since many existing models and schema do not include well defined units some effort may be required to find and then specify the implicit units based on documentation, SME interviews or inspection of data or source code. It is recommended that the units used by external models and schema be determined prior to attempting federation and integration of information based on those models or schema.

**VIM [JCGM 200-2008] concepts of quantities and units**

VIM defines

* quantity: property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference [ed. to a unit]
* kind of quantity (kind): aspect common to mutually comparable quantities
* measurement unit (unit): real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number

**SIMF concepts of quantities and units**

SIMF uses the VIM concepts to define "quantity values" and types to capture the quantity kind and unit. Types are defined for each Unit. The goals for this type based approach are:

* That it is clearly grounded in semantics as defined in VIM
* That a type may be used to specify the range of a property or relation involving unit based values.
* That a quantity value (e.g. 5 grams) be representable as a simple number with a type.
* That there is a clear type hierarchy starting with a representationally independent type in a conceptual model (e.g. mass) that can be further specialized to a specific unit in a logical model (e.g. grams) and further specialized to be represented by a physical data type (e.g. “double”).
* That external models and schema may have unit specifications asserted without changing the schema.
* That a quantity of an entity be able to be referenced without a specific quantity value being known (e.g. John’s weight).
* That systems of units such as [ISO-80000] or [OMG QUDV] (A part of SysML) be able to be directly referenced as the definition of a unit.

SIMF defines three types to realize the above goals: Quantity Kind, Unit Type, Base Unit Type. SIMF also defines Quantity Values, which are instances of unit types.

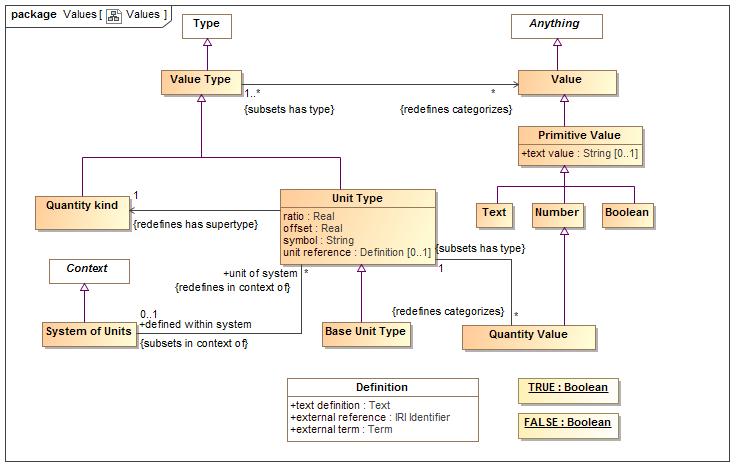
In VIM a quantity has a magnitude that is expressed as a number and a reference. The SIMF quantity value is the numeric value of such a quantity where the reference is specified by the “unit reference” property of the quantity value’s type. The quantity value’s type is a “Unit Type”. The Unit type has attributes for converting a unit to a base unit, a symbol and a unit reference. Based on VIM the unit reference may be “a measurement unit, a measurement procedure, a reference material, or a combination of such” and is specified with a description that contains reference information. In summary, the reference of a SIMF quantity value is determined indirectly through its unit type. A quantity value has exactly one unit type and exactly one Quantity Kind. A quantity value expressed in any unit of the same quantity kind may be converted to any other unit of the same quantity kind.

This type-based sapproach allows specification of a property at the conceptual (quantity kind) logical (unit type) or physical (unit type with a numeric type) levels. Such specifications use the same type-based approach used for other aspects of the models. Given this information a SIMF implementation may correctly and reliably convert between compatible types regardless of representation. Please see the specification of the value types, attributes and relationships for more detail.

**Example:**

* A specification for a road segment has a property “Speed limit”.
* The type of this property in a reference conceptual model is “Speed:Quantity Kind”.
* A unit “Kilometer per Hour:Unit Type” is defined as a subtype of “Speed:Quantity Kind” with a “unit reference” of “[ISO-80000.4] Kilometer per Hour”. Note that quantity kinds and unit types would normally be defined in reference models that correspond to a “system of units”.
* Miles per hour is also defined as a subtype of Speed.
* A physical schema defines “Speed-KPH: Integer”.
* A SIMF mapping rule maps “Speed limit” to “Speed-KPH” and asserts a type of “Kilometer per Hour” on the “Speed-KPH” end.
* A data file defines a road “Route One” with a speed limit of 100:KPH-Int.
* When converted to a U.S. application this speed limit of route one can be viewed as 62:MPH-Int.

### Diagram: Values



1. Values

### Class Base Unit Type

One unit type of a quantity kind may be marked as the base unit within a system of units. The base unit provides the basis for conversions between units of the same quantity kind. The base unit always has a ratio of one and an offset of zero.  
  
Type of a [JCGM 200:2008] measurement unit that is adopted by convention for a base quantity   
  
There ma be at most one base unit for a quantity kind within a system of units.

#### Direct Supertypes

[Unit Type](#_9a97d5f73bf658c81147f5fab194bf88)

**package** SIMF Conceptual Model::Values

### Class Boolean

A value that may be true or false.

#### Direct Supertypes

[Primitive Value](#_035e014ea6fe30370c344201d32a790f)

**package** SIMF Conceptual Model::Values

### Class Number

The concept of a number that may be expressed in any way.

#### Direct Supertypes

[Primitive Value](#_035e014ea6fe30370c344201d32a790f)

**package** SIMF Conceptual Model::Values

### Class Primitive Value

#### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

**package** SIMF Conceptual Model::Values

#### Attributes

-1626698621.jpg text value : [String](#_e8a6ce315d976318da3ab784a645ea44) [0..1]

Textual form of the value, if any. All primitive types, including numbers, should have a text value - the textual representation of the value. Numbers assume base 10 for a textual value.  
Mapping engines are responsible for mapping all primitive value types.

### Class Quantity kind

[JCGM 200:2008] A Quantity Kind is an aspect common to mutually comparable quantities represented by one or more units. Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. e.g. temperature.  
  
Quantity kinds are a supertype of unit types which are then a type of all quantity values, Quantity values are mutually comparable with all other quantity values categorized by the same quantity kind.

#### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

**package** SIMF Conceptual Model::Values

### Class Quantity Value

A quantity value is a numeric magnitude with a unit type that may be used as the value of a quantity property as defined by [JCGM 200:2008]. The reference of the quantity is defined by the "unit reference" property of the Unit Type.  
  
Each quantity value has exactly one Quantity Kind as a type.  
  
In a physical model a quantity value must have a type that specifies its unit (e.g. "Gram") and may have a data type specifying its numeric representation (e.g. "Double").  
  
[JCGM 200:2008] A quantity is a property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed as a number and a reference.   
  
Note: A quantity as defined here is a scalar. However, a vector or a tensor, the components of which are quantities, is also considered to be a quantity.  
  
e.g. 5cm is an instance of the unit type "Centimeter"

#### Direct Supertypes

[Number](#_f5a86db7bd9636f0fa472c3859bc9c3c)

**package** SIMF Conceptual Model::Values

#### Associations

-1808253052.jpg : [Unit Type](#_9a97d5f73bf658c81147f5fab194bf88) [1] *Subsets*: has type:[Type](#_dfe1514224ca21cedba7b2b29802db50)

### Class System of Units

[JCGM 200:2008] A set of base units and derived units, together with their multiples and submultiples, defined in accordance with given rules, for a given system of quantities

#### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

**package** SIMF Conceptual Model::Values

#### Associations

-1808253052.jpg unit of system : [Unit Type](#_9a97d5f73bf658c81147f5fab194bf88) [\*] *Redefines*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)

### Class Text

Text is a <Value> represented using symbols which have a meaning to stakeholders but otherwise have no formal semantic implication. Properties involving values may have a semantic implication.

#### Direct Supertypes

[Primitive Value](#_035e014ea6fe30370c344201d32a790f)

**package** SIMF Conceptual Model::Values

### Class Unit Type

A Unit type is a type of a quantity value referencing a specific unit. A Unit Type a required type of a property representing a quantity.   
  
Each quantity value has a reference as defined by the "unit reference" property of the quantity value's type.  
  
[JCGM 200:2008] A Unit is a real scalar quantity, defined and adopted by convention, with which any other quantity of the same quantity kind can be compared to express the ratio of the two quantities as a number. e.g. Degrees Centigrade, Miles.  
  
Each unit type represents refinement of a quantity kind using generalization and is thus substitutable for that quantity kind. Typically quantity kinds are used in conceptual models and unit types in physical or logical models.  
  
Unit types may only subtype quantity kinds.  
  
Note that unit types are not units, but the type of quantity values expressed in a common unit as defined in [JCGM 200:2008].

#### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

**package** SIMF Conceptual Model::Values

#### Attributes

-1626698621.jpg ratio : [Real](#_aef4bcae5ebc35dd9653214547b3e3cc)

the multiplier by which to multiple the referenced unit to convert to the base unit within a system of units.

-1626698621.jpg offset : [Real](#_aef4bcae5ebc35dd9653214547b3e3cc)

the difference between zero in the referenced unit and zero in the base unit after the ratio is applied within a system of units.

-1626698621.jpg symbol : [String](#_e8a6ce315d976318da3ab784a645ea44)

the accepted symbol for the unit referenced by the unit type

-1626698621.jpg unit reference : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [0..1]

The unit reference is the reference to a unit shared by all quantities values that are instances of a unit type.  
  
[JCGM 200:2008] A reference can be a measurement unit, a measurement procedure, a reference material, or a combination of such. For magnitude of a quantity.  
  
Typical references include ISO 8000 and OMG QUDV.

#### Associations

-1808253052.jpg : [Quantity Value](#_746074a5bfbd6e26906da5d4bd0d2a7f) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-58492342.jpg : [Quantity kind](#_ca1c56b440439615024c837658185d15) [1] *Redefines*: has supertype:[Type](#_dfe1514224ca21cedba7b2b29802db50)

-1808253052.jpg defined within system : [System of Units](#_7e22047cc4643bdb106af5bc777cd98a) [0..1] *Subsets*: in context of:[Context](#_66d62b068053cee3464e1e03e6035eed)

The system of units in which a unit is defined and is the basis for ratio and offset.  
  
By default the system of units is "si": http://www.iso.org/iso/iso\_catalogue/catalogue\_ics/catalogue\_detail\_ics.htm?csnumber=30669

### Class Value

A Value is an atomic piece of information without a specific lifetime or identity independent of the value. Values include numbers, strings and other atomic "primitive" data.  
  
In UML values may be defined by the name of an instance specification with a value type.

#### Direct Supertypes

[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

**package** SIMF Conceptual Model::Values

### Class Value Type

A type categorizing values where a value is an atomic piece of information without a specific lifetime or identity independent of that value. Values include numbers, strings and other atomic "primitive" data.

#### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

**package** SIMF Conceptual Model::Values

#### Associations

-58492342.jpg : [Value](#_a739673c8d53da123e392b7e5059ceec) [\*] *Redefines*: categorizes:[Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

# Foundational Assumptions (Normative)

## Multiple representations of overlapping concepts

A base assumption of SIMF is that there are and will be multiple information resources that represent information about the same and overlapping things in different ways. These different representations may use different definitional languages (e.g. XML Schema, SQL, RDF, UML) and well as different domain terminology and human languages (e.g. Risk, Danger, Achtung, Risque). Our task is to federate (combine) information from multiple sources as well as express information from one source in another form.

## Models may include “ground facts”

Any structured set of information about anything is considered a “model”. This usage of the term model is consistent with the ontological usage but is an expansion of the term as typically used in UML and similar modeling languages. “Ground facts” are facts about individuals (real things) in the real world or some imagined world. E.g. “George weighs 155 pounds” is a ground fact.

Abstractions such as types, constraints and rules are not considered ground facts. SIMF models may include ground facts as well as more abstract concepts.

## Conceptual Models

For a SIMF implementation there will exist a conceptual model that contains all the concepts to be interpreted and various mappings to physical or logical data representations. It should also be noted that representation may or may not be fully physical, some representations may, in turn, have various syntaxes. For example, there are multiple syntaxes for RDF. We are treating all these layers of representation the same with the assumption there is an isomorphic transform between elements in the same representation family (e.g. the RDF logical model and RDF in Turtle or JSON-LD Syntax). The conceptual model will be a “hub” or “pivot point” to integrate multiple representation models. We do not assume any one hub model but expect that some conceptual models will be broadly applicable. For physical representations to be mapped, there must be some CDM hub in common.

Conceptual models are intended to be models of the world – real world things (or possible worlds we can conceive). Facts about real world things are represented by data models in a data representation form but these facts also have a concept counterpart.

There are many possible implementation choices for concept models. Implementations may use the conceptual model only as a “virtual” rule set and have no physical representation of information about concept instances. Implementations may instantiate a physical representation corresponding directly to the conceptual model.

There is no assumption of any single or distinguished conceptual model. concept model is a role for a model, some conceptual models may be derived from existing representations – particularly model based representations such as UML or OWL. Of course, some models will make better conceptual hubs than others.

## Identity and identifiers

There are different forms of identity and identifiers. The distinguishing characteristic of entities is that they have identity (they can be distinguished from other things in some way). Entities may or may not have identifiers for those identities. There is the identity of the “real world” thing that some representation is stating facts about and there is the identity of the representation (the data or assertion - facts or statements). An identifier for a thing is called its “sign”. The sign of the real-world thing and the sign of data about that thing are not the same. Most representation models have representation identifiers but sometimes confuse these as signs for (identifiers for) the real world thing. Further, anything may have multiple identifiers – however many technologies only have one. “Keys” tend to be signs for real things, but this is not consistent.

## Facts & propositions

The term “fact” as used here is based on “fact modeling” but does not necessarily mean that something is “true”, a fact is some proposition that is asserted in some context - it may or may not be true and that truth may be contextual. Other terms could be “assertion” or “statement”. Facts are based on Propositions – anything that can be true or false. Once a proposition is asserted it is considered a fact in that context.

## Representations of a concept

Where there exists some concept of something in a model there is a representational data element representing facts about this concept. The concept is also a representation in some respects, but this representation is playing the role of representing the real-world thing, not a data structure.

### Represents Relation

“Represents” rules provide the basis for the relation between a representation and a concept. Both ends or the represents relation require an identity for the entity being represented. Thus “represents” depends on the ability to identify both the representation and concept. See 8.14.7.

The represents relation asserts that the representation element represents information about the concept. The information is not assumed to be complete or consistent. For various types of concepts this implies:

* **Individuals:** That the representation individual is a sign for facts (data) about the concept individual. Individuals are identifiable instances where the extent of the sign represents a single “thing” in the world. Types, context, relations and values are not individuals.
* **Types**: That the representation type describes a concept type. The extent of the representation type may be a subset of the extent of the concept type.
* **Context**: That the representation context has data about all or a subset of the domain context.
* **Relations**: That the relation semantics holds true for the bound elements of both the representation and concept

## Representation identifiers

An information resource (i.e. a repository or DBMS) contains a set of facts, such a resource may change over time so the set of facts may also change. At a particular time an information resource has a fixed set of facts. Within an information resource a particular fact has an identity and any number of identifiers for that identity. Identifiers are used to match identity within and between information resources.

Anything in an information resource (some repository of information) has an identifier. The form and structure of that identifier is technology dependent and not determined in this specification but it must be representable as a text string. Given an identifier any entity or fact must be traceable to its origin. An information model may provide additional provenance information as part of a transaction and/or resource source reference but the semantics of that provenance is not assured unless the source is also a SIMF model.

Identical representation identifiers, within the same resource, are assumed to identify the same thing. Identical identifiers within different resources may be interpreted as identifying the same concept. Such identifiers are frequently called UUIDs or GUIDS and may be expressed as an IRI.

### Example physical identifiers

id="opensource:ttp-62ff7599-e1db-414f-b742-908def2ec219"

external reference=" http://www.omg.org/spec/UML/2.5"

timestamp="2014-11-06T17:46:28.913306+00:00"

## Sources

All concepts come from some source outside of the conceptual system (even if this is a rule or inference operating in the conceptual system). Concepts have a *source*. All sources produce concept representations via some rule (inferences are considered rules). A representation of a concept (represents relation) is one such rule. Thus all concepts may be traced to their source as it existed at a particular time and was interpreted or computed based on a rule at that time.

Since facts come from an identifiable source, that source and any internal structure may be used as a hint in producing the fact identifier. For resources with internal structure, this would typically be the Resource followed by a set of Packages followed by some name, ID or path of the representation fact.

The same source may result in multiple concepts. A representation concept is unique to a source but may be declared as equivalent to other representations concept.

Source information utilizes [MOF].

## Ownership

Representation information shall be “owned” directly or indirectly by a model, this ownership shall correspond with the structure of the source where such structure is important in interpreting the concept. A SIMF statement is owned, the ownership of the statement has no semantic interpretation with relation to what the concept means unless there are statements about the owning context. Ownership of the concept may have a semantic interpretation as to the authority, scope or timeframe of the concept.

Ownership is one dimension of “context”, a concept may be contextualized by any number of context. Context may impact the interpretation of or trust in a concept. The context set of an owner in included in the context sets of concepts it owns.

## Lifetime and context of facts

Logically concepts and facts are never “deleted”, they may become irrelevant or dated as would be determined by statements about the concept. Implementations may optimize a knowledge base by deleting facts that have been determined to be irrelevant for all purposes of the knowledge base (i.e. they would never impact the truth of any computation or query). Deletion of facts is not, in general, recommended (deletion is so 20th century).

Example: the fact “Thomas Woodrow Wilson is president of the United States” was true “March 4, 1913 through March 4, 1921”. We accept it will always be true that this fact was true in this timeframe, so there is no sense in “deleting” the fact. The timeframe fact “March 4, 1913 through March 4, 1921” is a fact about the fact: “Thomas Woodrow Wilson is president of the United States”. Another fact about this fact is that its source resource was <http://www.whitehouse.gov> on January 14th, 2015 at 9:05PM-EST. This “source” fact is classified as an “Annotation” which represents metadata.

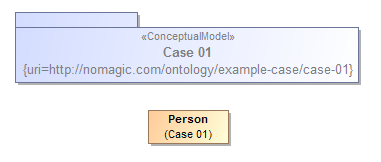
Note that the above implies that there may be more than one Ffact that says exactly the same thing, but they may come from different sources or be stated in different context or timeframes. Or, different sources or context may assert conflicting facts about the same thing. Trust is a function of the interpretation of a set of facts, including their source, timeframe and context. Interpretation is time dependent (i.e. transactional), with respect to a given set of context operating on a given knowledge base at a given time. (Note that a knowledge base may include all or part of the contents of other repositories and that repositories may be physical or virtual). A knowledge base could be as small as one or more files on your tablet or as large as Google or Wikipedia.

# Mapping to OWL 2 (normative)

Examples are given below that show the transformation of UML modeled in SIMF to an exported OWL 2 ontology. The OWL ontologies are presented in OWL Functional Syntax.

The first diagram below, for a simple UML class, shows the ontology is transformed as the package containing the UML class. Subsequent diagrams do not show the package in the diagram for the sake of brevity.

## Class



Ontology(<http://nomagic.com/ontology/example-case/case-01>

Declaration(

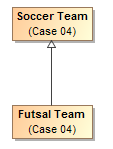
Class(:Person)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

)

## Class Generalization



Ontology(<http://nomagic.com/ontology/example-case/case-04>

Declaration(

Class(:FutsalTeam)

)

Declaration(

Class(:SoccerTeam)

)

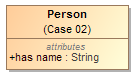
AnnotationAssertion(rdfs:label :FutsalTeam "Futsal Team"@en)

SubClassOf(:FutsalTeam :SoccerTeam)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

)

## Class with Datatype Property



Ontology(<http://nomagic.com/ontology/example-case/case-02>

Import(<http://www.omg.org/spec/PrimitiveTypes/20100901>)

Declaration(

Class(:Person)

)

Declaration(

DataProperty(:hasName)

)

Declaration(

AnnotationProperty(<http://purl.org/dc/terms/description>)

)

Declaration(

Datatype(xsd:string)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

SubClassOf(

:Person

ObjectIntersectionOf(

DataMaxCardinality(1 :hasName xsd:string)

DataMinCardinality(1 :hasName xsd:string)

)

)

AnnotationAssertion(rdfs:label :hasName "has name"@en)

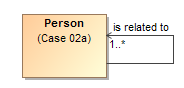
DataPropertyDomain(:hasName :Person)

DataPropertyRange(:hasName xsd:string)

AnnotationAssertion(<http://purl.org/dc/terms/description> <http://www.omg.org/spec/PrimitiveTypes/20100901#String> "An instance of String defines a piece of text. The semantics of the string itself depends on its purpose, it can be a comment, computational language expression, OCL expression, etc. It is used for String attributes and String expressions in the metamodel."@en)

)

## Class with Self-Referential Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-02a>

Declaration(

Class(:Person)

)

Declaration(

ObjectProperty(:isRelatedTo)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

SubClassOf(

:Person

ObjectIntersectionOf(

ObjectMinCardinality(1 :isRelatedTo :Person)

)

)

AnnotationAssertion(rdfs:label :isRelatedTo "is related to"@en)

ObjectPropertyDomain(:isRelatedTo :Person)

ObjectPropertyRange(:isRelatedTo :Person)

)

## Class with Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03>

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(

ObjectProperty(:consistsOf)

)

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

SubClassOf(

:SoccerTeam

ObjectIntersectionOf(

ObjectMaxCardinality(11 :consistsOf :SoccerPlayer) ObjectMinCardinality(5 :consistsOf :SoccerPlayer)

)

)

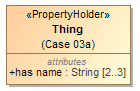
AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

## Property Holder with Datatype Property



Ontology(<http://nomagic.com/ontology/example-case/case-03a>

Import(<http://www.omg.org/spec/PrimitiveTypes/20100901>)

Declaration(

DataProperty(:hasName)

)

Declaration(

AnnotationProperty(<http://purl.org/dc/terms/description>)

)

Declaration(

Datatype(xsd:string)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

DataMaxCardinality(3 :hasName xsd:string)

DataMinCardinality(2 :hasName xsd:string)

)

)

AnnotationAssertion(rdfs:label :hasName "has name"@en)

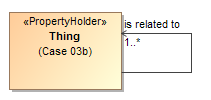
DataPropertyRange(:hasName xsd:string)

AnnotationAssertion(<http://purl.org/dc/terms/description>

<http://www.omg.org/spec/PrimitiveTypes/20100901#String> "An instance of String defines a piece of text. The semantics of the string itself depends on its purpose, it can be a comment, computational language expression, OCL expression, etc. It is used for String attributes and String expressions in the metamodel."@en)

)

## Property Holder with Self-Referential Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03b>

Declaration(

ObjectProperty(:isRelatedTo)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isRelatedTo)

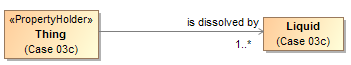
)

)

AnnotationAssertion(rdfs:label :isRelatedTo "is related to"@en)

)

## Property Holder with Object Property



Ontology(<http://nomagic.com/ontology/example-case/case-03c>

Declaration(

Class(:Liquid)

)

Declaration(

ObjectProperty(:isDissolvedBy)

)

AnnotationAssertion(rdfs:label :Liquid "Liquid"@en)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isDissolvedBy :Liquid)

)

)

AnnotationAssertion(rdfs:label :isDissolvedBy "is dissolved by"@en)

ObjectPropertyRange(:isDissolvedBy :Liquid)

)

## Class with Object Property without Range



Ontology(<http://nomagic.com/ontology/example-case/case-03d>

Declaration(

Class(:Receptacle)

)

Declaration(

ObjectProperty(:holds)

)

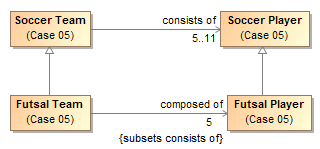
AnnotationAssertion(rdfs:label :Receptacle "Receptacle"@en)

AnnotationAssertion(rdfs:label :holds "holds"@en)

ObjectPropertyDomain(:holds :Receptacle)

)

## Class with Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-05>

Declaration(

Class(:FutsalPlayer)

)

Declaration(

Class(:FutsalTeam)

)

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(

ObjectProperty(:composedOf)

)

Declaration(

ObjectProperty(:consistsOf)

)

AnnotationAssertion(rdfs:label :FutsalPlayer "Futsal Player"@en)

SubClassOf(:FutsalPlayer :SoccerPlayer)

AnnotationAssertion(rdfs:label :FutsalTeam "Futsal Team"@en)

SubClassOf(:FutsalTeam :SoccerTeam)

SubClassOf(

:FutsalTeam

ObjectIntersectionOf(

ObjectMaxCardinality(5 :composedOf :FutsalPlayer) ObjectMinCardinality(5 :composedOf :FutsalPlayer)

)

)

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

SubClassOf(

:SoccerTeam

ObjectIntersectionOf(

ObjectMaxCardinality(11 :consistsOf :SoccerPlayer) ObjectMinCardinality(5 :consistsOf :SoccerPlayer)

)

)

AnnotationAssertion(rdfs:label :composedOf "composed of"@en)

SubObjectPropertyOf(:composedOf :consistsOf)

ObjectPropertyDomain(:composedOf :FutsalTeam)

ObjectPropertyRange(:composedOf :FutsalPlayer)

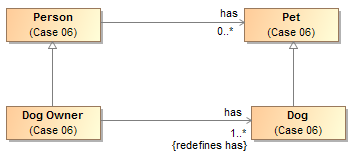
AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

## Class with Universal Quantification Constraint on Property I



Ontology(<http://nomagic.com/ontology/example-case/case-06>

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogOwner)

)

Declaration(

Class(:Person)

)

Declaration(

Class(:Pet)

)

Declaration(

ObjectProperty(:has)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog :Pet)

AnnotationAssertion(rdfs:label :DogOwner "Dog Owner"@en)

SubClassOf(:DogOwner :Person)

SubClassOf(

:DogOwner

ObjectIntersectionOf(

ObjectMinCardinality(1 :has :Dog)

ObjectAllValuesFrom(:has :Dog)

)

)

AnnotationAssertion(rdfs:label :Person "Person"@en)

AnnotationAssertion(rdfs:label :Pet "Pet"@en)

AnnotationAssertion(rdfs:label :has "has"@en)

ObjectPropertyDomain(:has :Person)

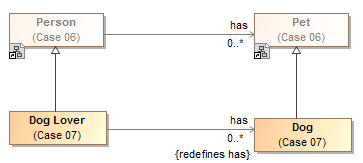
ObjectPropertyRange(:has :Pet)

)

## Class with Universal Quantification Constraint on Property II

This example differs from the previous example primarily in that the superclasses “Person” and “Pet” are from a different package than their subclasses “Dog Lover” and “Dog,” respectively. This is reflected in the OWL ontology by the import of this namespace.

The superclasses “Person” and “Pet”, defined in the package “Case 06”, are a different color and a lighter shade than the classes defined in the package “Case 07”. This is to distinguish them from the classes defined in this package. MagicDraw’s AutoStyler plugin can automatically set the properties for classes and other UML elements “defined elsewhere,” that is in a package not containing the defining diagram for the UML element (See section 2.2, Automatic Styling of Concept Models.).



Ontology(<http://nomagic.com/ontology/example-case/case-07>

Import(<http://nomagic.com/ontology/example-case/case-06>)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Person>)

)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Pet>)

)

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogLover)

)

Declaration(

ObjectProperty(<http://nomagic.com/ontology/example-case/case-06#has>)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog <http://nomagic.com/ontology/example-case/case-06#Pet>)

AnnotationAssertion(rdfs:label :DogLover "Dog Lover"@en)

SubClassOf(:DogLover <http://nomagic.com/ontology/example-case/case-06#Person>)

SubClassOf(

:DogLover ObjectIntersectionOf(

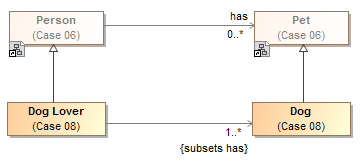
ObjectAllValuesFrom(<http://nomagic.com/ontology/example-case/case-06#has> :Dog)

)

)

)

## Class with Existential Quantification Constraint on Property



Ontology(<http://nomagic.com/ontology/example-case/case-08>

Import(<http://nomagic.com/ontology/example-case/case-06>)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Person>)

)

Declaration(

Class(<http://nomagic.com/ontology/example-case/case-06#Pet>)

)

Declaration(

Class(:Dog)

)

Declaration(

Class(:DogLover)

)

Declaration(

ObjectProperty(<http://nomagic.com/ontology/example-case/case-06#has>)

)

AnnotationAssertion(rdfs:label :Dog "Dog"@en)

SubClassOf(:Dog <http://nomagic.com/ontology/example-case/case-06#Pet>)

AnnotationAssertion(rdfs:label :DogLover "Dog Lover"@en)

SubClassOf(:DogLover <http://nomagic.com/ontology/example-case/case-06#Person>)

SubClassOf(

:DogLover

ObjectIntersectionOf(

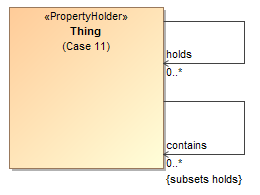
ObjectMinCardinality(1 <http://nomagic.com/ontology/example-case/case-06#has> :Dog) ObjectSomeValuesFrom(<http://nomagic.com/ontology/example-case/case-06#has> :Dog)

)

)

)

## Property Holder with Self-Referential Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-11>

Declaration(

ObjectProperty(:contains)

)

Declaration(

ObjectProperty(:holds)

)

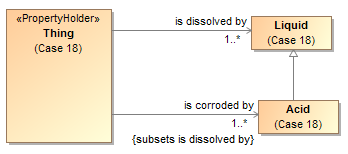
AnnotationAssertion(rdfs:label :contains "contains"@en)

SubObjectPropertyOf(:contains :holds)

AnnotationAssertion(rdfs:label :holds "holds"@en)

)

## Property Holder with Subproperty



Ontology(<http://nomagic.com/ontology/example-case/case-18>

Declaration(

Class(:Acid)

)

Declaration(

Class(:Liquid)

)

Declaration(

ObjectProperty(:isCorrodedBy)

)

Declaration(

ObjectProperty(:isDissolvedBy)

)

AnnotationAssertion(rdfs:label :Acid "Acid"@en)

SubClassOf(:Acid :Liquid)

AnnotationAssertion(rdfs:label :Liquid "Liquid"@en)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isCorrodedBy :Acid)

)

)

SubClassOf(

owl:Thing

ObjectIntersectionOf(

ObjectMinCardinality(1 :isDissolvedBy :Liquid)

)

)

AnnotationAssertion(rdfs:label :isCorrodedBy "is corroded by"@en)

SubObjectPropertyOf(:isCorrodedBy :isDissolvedBy)

ObjectPropertyRange(:isCorrodedBy :Acid)

AnnotationAssertion(rdfs:label :isDissolvedBy "is dissolved by"@en)

ObjectPropertyRange(:isDissolvedBy :Liquid)

)

## Class with Subproperty without a Range



Ontology(<http://nomagic.com/ontology/example-case/case-16>

Declaration(

Class(:Game)

)

Declaration(

Class(:SoccerMatch)

)

Declaration(

ObjectProperty(:isACompetitionBetween)

)

Declaration(

ObjectProperty(:isPlayedBetween)

)

AnnotationAssertion(rdfs:label :Game "Game"@en)

SubClassOf(

:Game

ObjectIntersectionOf(

ObjectMinCardinality(2 :isPlayedBetween)

)

)

AnnotationAssertion(rdfs:label :SoccerMatch "Soccer Match"@en)

SubClassOf(:SoccerMatch :Game)

SubClassOf(

:SoccerMatch

ObjectIntersectionOf(

ObjectMaxCardinality(2 :isACompetitionBetween) ObjectMinCardinality(2 :isACompetitionBetween)

)

)

AnnotationAssertion(rdfs:label :isACompetitionBetween "is a competition between"@en)

SubObjectPropertyOf(:isACompetitionBetween :isPlayedBetween)

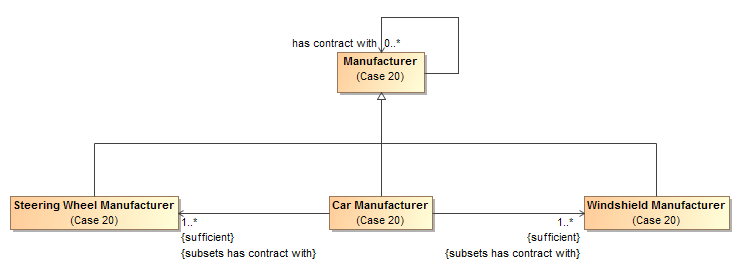
ObjectPropertyDomain(:isACompetitionBetween :SoccerMatch)

AnnotationAssertion(rdfs:label :isPlayedBetween "is played between"@en)

ObjectPropertyDomain(:isPlayedBetween :Game)

)

## Class with Necessary and Sufficient Property



Ontology(<http://nomagic.com/ontology/example-case/case-20>

Declaration(

Class(:CarManufacturer)

)

Declaration(

Class(:Manufacturer)

)

Declaration(

Class(:SteeringWheelManufacturer)

)

Declaration(

Class(:WindshieldManufacturer)

)

Declaration(

ObjectProperty(:hasContractWith)

)

AnnotationAssertion(rdfs:label :CarManufacturer "Car Manufacturer"@en)

EquivalentClasses(

:CarManufacturer

ObjectIntersectionOf(

ObjectMinCardinality(1 :hasContractWith :SteeringWheelManufacturer) ObjectSomeValuesFrom(:hasContractWith :SteeringWheelManufacturer)

)

)

EquivalentClasses(

:CarManufacturer

ObjectIntersectionOf(

ObjectMinCardinality(1 :hasContractWith :WindshieldManufacturer)

ObjectSomeValuesFrom(:hasContractWith :WindshieldManufacturer)

)

)

SubClassOf(:CarManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :Manufacturer "Manufacturer"@en)

AnnotationAssertion(rdfs:label :SteeringWheelManufacturer "Steering Wheel Manufacturer"@en)

SubClassOf(:SteeringWheelManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :WindshieldManufacturer "Windshield Manufacturer"@en)

SubClassOf(:WindshieldManufacturer :Manufacturer)

AnnotationAssertion(rdfs:label :hasContractWith "has contract with"@en)

ObjectPropertyDomain(:hasContractWith :Manufacturer)

ObjectPropertyRange(:hasContractWith :Manufacturer)

)

## Class With Property Having Unspecified Multiplicity

UML allows the cardinality of a property to be left unspecified. The concept modeling profile interprets unspecified cardinalities as being zero to many (“0..\*”).



Ontology(<http://nomagic.com/ontology/example-case/case-21>

Declaration(

Class(:SoccerPlayer)

)

Declaration(

Class(:SoccerTeam)

)

Declaration(ObjectProperty(:consistsOf))

AnnotationAssertion(rdfs:label :SoccerPlayer "Soccer Player"@en)

AnnotationAssertion(rdfs:label :SoccerTeam "Soccer Team"@en)

AnnotationAssertion(rdfs:label :consistsOf "consists of"@en)

ObjectPropertyDomain(:consistsOf :SoccerTeam)

ObjectPropertyRange(:consistsOf :SoccerPlayer)

)

# Annex A: UML Conceptual Modeling Profile (normative)

This section defines the UML profile for conceptual modeling and mapping. In order to improve UML’s suitability for modeling real-world concepts, this profile interprets standard with semantic features, as detailed below:

## UML Conceptual Model Primer

A conceptual model can be expressed in UML with the conceptual modeling profile. The profile defines the interpretation of UML concepts used, extends UML concepts with “stereotypes” and makes some UML semantics more specific to conceptual modeling. While there are some extensions, every effort is made to use “generic UML” class diagrams as they are well understood and supported. Readers are referred to the UML specification and the many books and courses on UML for an in-depth treatment of generic UML.

This section is intended as a short primer on how UML is used in this specification to represent conceptual models. The subset of UML used for conceptual modeling is primarily that known as “Class models”, the most commonly used part of UML. Our scope further narrows what we utilize to exclude behaviors and methods – elements used for object oriented design.

Concepts are, of course, the foundation of a conceptual model. Concepts are the elements of how we think about the world. A conceptual model is not a data model, when thinking about concepts we think about the real-world things, not data structures about those things. These real-world concepts become the pivot points around which we define and relate the many data structures that describe those things.

A conceptual model consists of a network of concepts with a simple essential structure. That structure is the definition of classes, relations between them and their characteristics. Classes represent the “things” in our world – including physical things like trees or people and “made up” things like agreements. Other concepts connect those things - the relationships between things-these are UML associations. Things also have characteristics such as weight or color, represented as properties. This basic network of classes, relationships and properties forms the foundation of the conceptual model and define the conceptual framework and vocabulary of a domain. Each of these concepts may be given names, which form the vocabulary of a domain of interest. Various assertions are then made about these concepts and their connections that further define the semantics of those concepts – multiplicities of relationships, specializations between concepts, essential properties of things, etc.

One of the fundamental ways we understand and organize concepts is their arrangement into hierarchies where general concepts are specialized to form more specific concepts within a specific context or with more specific characteristics. A conceptual model can arrange all the fundamental elements into hierarchies using generalization. Another kind of hierarchy is a structural data hierarchy – where data elements contain other data elements. As the conceptual model is not representing data, data hierarchies are not part of a conceptual model – they may be part of data models that are related to a conceptual model. To allow for the many viewpoints that can exist for any concept, a concept can be in many hierarchies at the same time.

The following section defines how basic UML is used to represent the foundational network of concepts using classes, associations and properties. Additional constraints are then attached to this basic framework to enhance semantic expression and the ability of automation to federate and analyze information about those concepts.

#### Classes

Classes classify a set of things according to some set of rules or understanding. Classification is the essential mechanism of conceptualization we use. Classes classify some set of things belonging to that class – this is called the class’s *extent*. Each element of the class is an *instance* of that class – that is it is something the class classifies. Classifications may be arranged in hierarchies.

In the UML conceptual model a class is diagramed as a box with a name at the top. In some cases a definition is also shown next to the box in a “note” form.

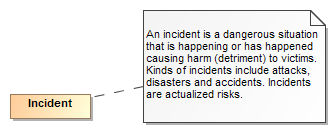


Figure Example of a Class

The above example shows the class “Incident” and its definition. It should be noted that a class is *a way* to classify something. It is natural to classify something multiple ways. For example we may classify a situation as also being a danger or, to someone else, an opportunity to do harm. This is different from many technology models (e.g. Java) that only allow something to be classified in one way and the classification is fixed. *The basic assumption of the conceptual model is that unless specified otherwise, something may be classified in any number of ways and those classifications may change over time*.

#### Instances

While not usually used in the definition of the conceptual model, instances can also be shown in UML and are utilized to illustrate examples. Since the model is conceptual, instances of classes are the “real thing” in the world – not data about them or other technology artifacts. However we sometimes want to show information about instances.

Instances are also shown as a box, but have a “:” separating the name of the instance from its classes.



Figure Instance Example

The above example shows a information about an instance named “Joe Smith” that is classified as a “Person” and a “Victim”.

#### Class Hierarchies

Since Aristotle classes have been arranged in hierarchies – from most general concepts to more specific ones. In UML this is shown as a Generalization – an arrow with a solid line from the more specific concept to the more general. The more specific class is known as the *Superclass* (or *Supertype*) and the more specific the *Subclass* (or *Subtype*). Generalization has some specific semantic rules:

* Everything that is true about the superclass must be true about all its subclasses
* The extent of the subclass is a subset of the extent of the superclass
* All properties and associations that apply to a class also apply to all its subtypes

In a conceptual model a class may have any number of superclasses or subclasses. Some technologies (Like XML Schema) limit the number of superclasses to one.

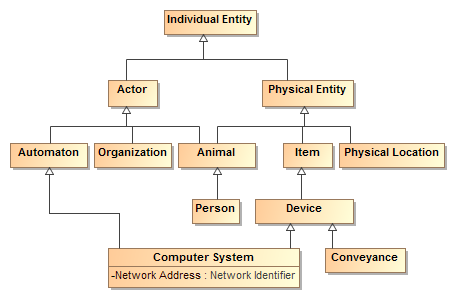


Figure Class Hierarchy Example

The above example shows a class hierarchy with multiple levels.

***Note that all properties and associations defined for all superclasses of a class apply to that class. For that reason a complete understanding of a class and its potential properties must include such superclasses.***

#### Properties

Properties represent qualities inherent in something, such as size, weight or a time. Each property has a “type” for the kind of value that represents that quality. Properties are shown in the box of the most general classifier that can have that quality.

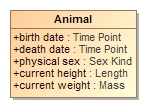


Figure Example of Properties

The above example shows that an animal has the qualities of birthdate, death date, physical sex, height and weight. Note that these is no assumtion that these qualities may be known, required or that different data sources may or may not agree on them – just that a person has these qualties. Instances of properties are facts about the entity they describe.

In conceptual models properties are only used for qualities, never to relate different entities.

#### Associations

Associations describe facts about how entities are related. Associations are shown as lines between the classes that have related instances. At each end of the line is the “association end” – the association end describes how the instances of the class on the far end relate to those of the near end. If there are limits to how many instances may be related, these are also shown. Since an association has two ends, the association may be read in either direction, but is the same “fact”. The ends of relations are typically verbs or verb phrases, but in some cases can also be role names. In either case the name denotes the intent of the class *at the other end of the line*.



Figure Association Example

The above example says that there are relations between actors and activities such that the *actor performs the activity* and the activity is performed by the actor. These are considered two ways to “read” the same fact. Like any fact, relations may be true for some period of time or in some specific situation.

As can be seen in the example the ends of associations are typically verb phrases which can then be read as <the actor> performs <the activity>. In other cases the ends are nouns in which case they represent a role being played. If a role were used above instead of “performed by” it could read: <activity> has performer <actor> (the *has* in this sentence being implied by english gramar).

This combination of classes and associations with ends forms the basis for nouns and verbs common to human language. The terms used for the nouns and verbs should be both consistent with their semantics and resonate with stakeholders – sometimes this is a bit of a challenge.

In some cases the ends of the relation are sufficent to define it, in other cases it makes more sense to give the association a name and its own definition. Associations and association ends, like classes, can be part of a hierarchy.

#### Property and association end hierarchies

Like class hierarchies, property and association end concepts (we will just call both properties from now on) can also be aranged in hierarchies of more or less specific concepts. In UML property hierarchies are represented using with either “Subsets” or “Redefines”. What a property subsets or redefines is shown next to its defintionin in the diagram (Note that by convention this is not shown on summary diagrams, only the primary definition of the property). If a property completely subsumes the other in a particular context it uses a “Redefines” – that is the redefining and redefined properties have the same set of values. If the more general concept can also be used in the context a “Subsets” is used.

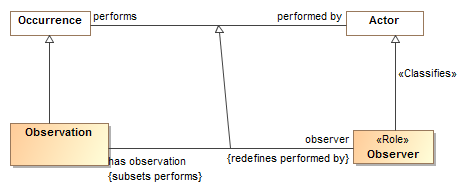


Figure Example of Association End Hierarchy

The above example shows that the “has observation-observer” concepts are specializations of the “performs-performed by” concepts. Observer redefines “performed by” – that is an observation always has an observer, never a “performed by” actor. Likewise “has observation” specializes “performs” but an actor can perform other activities as well. Note the generalization between the associations is impled, but is shown in this example for clarity.

Where a redefined or subset property has no name it is an indication that theproperty is constrained in some way – based on the type of the and or the cardinality (number that may be related), no new properties or associations are actualy defined for a constraint (more on this below).

#### Association Classes

In a conceptual model any “fact” (association or property) may have properties. Of particular importance is the “provenance” of the fact – where the fact came from and thus how much it can be trusted. Facts are also time-bound, true for some period. Every fact is assumed to have these properties. Where an association may have additional specific properties or may it’s self participate in other relationships an “association class” is used. As implied by its name, an association class has both the properties of an association and a class. More complex associations between things use association classes. An association class is diagrammed as an association line and a class box with a dotted line between the association line and its class. While these may seem somewhat visually distinct – they are the “same concept”.



Figure Association Class Example

The above example shows the “Stakeholder Desirability” relation. Between any situation and any stakeholder there can be some metrics as to how much that stakeholder desires or wants to avoid that situation. The Stakeholder Desirability association class represents these as properties of the association: net desirability, net harm, net benefit and net risk – which can all be poitive or negative reflecting a benefit or harm, respectivly.

### Specific kinds of classes

There are additional conceptual modeling specific stereotypes documented in the reference section that further define the semantics of a class. Some of these stareotypes are very important for understanding the conceptual model and are further explained here. These are roles, phases and quantity kinds.

#### Context specific types and <<Classifies>>

Some types may be considered the “fundamental” type of something that is essential to its being for its entire lifetime; this is the default assumption of most classes. Other types classify something in a specific context or for a period of time, SIMF calls these “classifications”. The types an instance is classified with is expected to change over time and may be only valid within a particular context or viewpoint. Classifications are defined with a <<Classifies>> generalization to another type, the type of thing that can be so classified. For example, “Policeman” can classify a “Person”.

Context specific types such as Roles and Phases are classifications and expected to be used in this more contextual and dynamic fashion; these types may be assigned to or removed from an instance over time or in a context.

For an instance to be classified with a classification, it must also have the type of what the classification <<classifies>>. To use the example above, a “Policeman” can’t classify a Toaster since the toaster is not a person. Please see the “Role” and “Phase” discussion for more usage scenarios of <<Classifies>>.

**Implementation note**: most programming languages do not allow for direct representation of multiple classifications, multiple inheritance or context. A common implementation pattern is to represent classifications, roles and phases as independent objects related to the object they classify. An example of this is the IUnkown pattern in .NET.

The following stereotypes define additional classification semantics.

#### Roles

Roles are classes that are expected to be dynamic and contextual, such as teacher, victim or president. A role is defined as a class with the <<Role>> stereotype. Implementation technologies should interpret roles as classifications that may be added to or removed from an instance over time and may be defined in a particular context. A role is usually required to be a role of some particular other class, for example a teacher is expected to be a role of a person (at least until a computer takes her job). The constraint of what a role must be a role of is defined using a <<Classifies>> stereotype of a generalization.

Many implementation languages don’t have the capacity to represent roles, so roles are defined is the single and unchangeable “type” of a class or DBMS table. The problem with this is that the same individual may not be connected across all their roles. Specifically representing roles allows the same individual to play multiple roles and for these roles to change – this better reflects the reality of the world and the way we think about it.

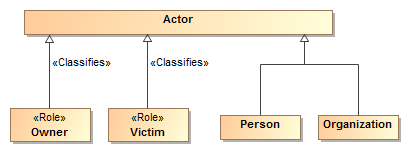


Figure Role Example

The above example shows that an actor can be a person or organization and that either could be classified as being able to play the Owner and/or a Victim role.

Roles help to decouple concepts in models and specifically allow an instance to “play” multiple roles at the same time or over time. Roles, when combined with quantification constraints, clearly define the semantics of roles. For example, we could say that a victim must be a victim of some incident and an owner must own something.

By convention, properties typed by roles may have the same name as the role, this can be read as “has <role>”, e.g. “has victim”, however full verb phrases may be more appropriate in some situations.

#### Phases

Phases are classes that are expected to classify an instance over a specific span of time, such as a teenager, “legal adult” or “Paid Invoice”. A teenager is a person between the ages of 13 and 19 (inclusive) – perhaps “legal adult” is of age 19 or older – we may also want to consider people living or dead, thus alive and dead would be phases. Phase may be considered a synonym for the “State” of something.

A phase is defined as a class with the <<Phase>> stereotype. Like roles, phases use the <<Classifies>> stereotype of a generalization to define what a phase must be a phase of.

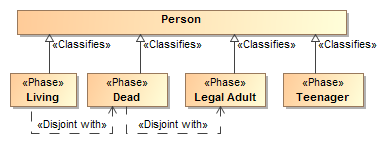


Figure Phases of a person

Also like roles, phases help to decouple concepts in models and specifically allow an instance to “be in” multiple phases (or multiple roles) at the same time or over time. If an instance cannot be in two phases at the same time or be in a role and a phase a “disjoint with” constraint can be used to state that restriction. For example, “Dead” is disjoint with “Legal Adult” and “Living”. Only a “Legal adult” can commit to a contract.

#### Quantity kinds and units

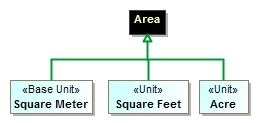
Fundamental to understanding and describing something is physical and other qualities such as temperature, length and color. Many data models fail to capture units of measure explicitly which can and has[[3]](#footnote-3) resulted in dramatic systems failures. A concept for somethings weight should properly be typed by a measure of weight, not an “int” or “real” – which are just ways to represent numbers without knowing what they mean. Of course there needs to be numbers, but in relation to their units.

Figure "Area" Example of quantity kinds and units

In that there are different units that can represent the same kind of measure, such as degrees Celsius and degrees Fahrenheit can represent the same temperature – an abstraction is used above like units. The abstraction for a measurable unit is called a <<Quantity Kind>>. Examples of quantity kinds include Length, mass, temperature, frequency, etc.

As any element of measurement data must be specific to a specific unit in a specific data exchange, the <<Unit>> stereotype is used to define a unit for a quantity kind. A <<Represents>> stereotype of generalization (Diagrammed as a green arrow) is used to say that the unit represents the quantity kind.

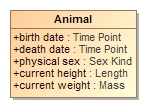
In the example above, the “Area” quantity kind (indicated by a black shaded class) can be represented by (the green lines) “Square Meter”, “Square Feet” or an “Acre”. One unit may be nominated as the “Base Unit” and will be used to express conversion factors between the units. As per SI specifications, the Square Meter is the base unit.

Figure - "Animal" example of using quantity kinds.

By convention quantity kinds are used in fully conceptual models whereas units are used in data models. The “Animal” example shows quantity kinds being used to define properties of animals.

### Assertions about concepts

Above we defined the network of essential concepts as classes, relationships and properties. Additional assertions may be made about those concepts using both UML foundational and extended profile capabilities. The following define the kinds of assertions that can be made. Note that the term “property” applies to both simple properties and the ends of associations.

#### Property Ownership

The concept modeling profile of UML interprets the owner of a property *definition* as the subject of that property (its domain) and the context in which that property must conform to certain constraints.

Constraints may be placed on a property. These constraints can include multiplicity, which includes a minimum cardinality and a maximum cardinality, a type for the property, existential quantification, and universal quantification. When an instance is a member of a class, all of that class’ constraints must be met.

Property ownership is not interpreted as “slots” in an object. Property values may or may not be independent of the instance that defined them, thus supporting an OWL/RDF, or “open world”, interpretation of properties and associations.

#### Cardinality

Cardinality defines how many instances of a property may exist for a particular subject instance. For example, how many ages can a person have? The obvious answer is that a person can have at most one age at any one point in time. Thus cardinalities represent the number of instances at any one time.

UML allows the cardinality of a property to be left unspecified. The concept modeling profile interprets unspecified cardinalities as being unconstrained - zero to many (“0..\*”), this is consistent with our general rule that anything unsaid is unconstrained.

#### Global Properties

Global properties are property declarations that can be used by any instance. Normally, a UML property cannot be defined outside of a classifier, so a global property declaration is represented as a UML property owned by a class that is stereotyped as a **«**PropertyHolder**».** The concept of a property holder was introduced in the NIEM-UML standard for a similar purpose. In the concept modeling profile, every property holder is equivalent to one topmost class (⊤) of which all other classes are subclasses. Thus, a property of a property holder is inherited by all subclasses. In addition, while the name of a property holder is irrelevant, consistently naming property holders “Thing”, “Concept”, or “Entity” in all concept models avoids any confusion with normal classes.

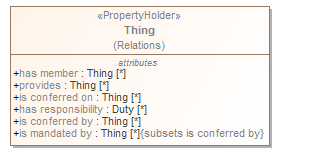


Figure Property Holder - "Thing"

### Constraining properties and associations

A cardinality of one or more defined for a property requires that an instance of the related element must exist for an instance of the domain (owning class) of that property or association end to be valid. For example, a living person must have exactly one living brain. This is known as an *existential quantification* (∃) or qualified constraint in first order logic. Existential quantification is defined using UML cardinality and *subsets*.

An existential quantification can be stated for a newly defined property or an existing one. For a newly defined property this is done by simply stating cardinality greater than one. For example, a phone must have at least one button with a “has buttons” association end property and a cardinality of “1..\*”. When a new property is being defined it is given a name. If an existing property is being constrained (without a new property being defined) it subsets or redefines the existing property and does not have a name. In the concept modeling interpretation of UML, any cardinality greater than zero creates an existential quantification constraint.

A property is not limited to a minimum and a maximum cardinality (known as multiplicity) for just one type. A property can have a multiplicity for a superclass, while at the same time having a more specific multiplicity for one or more subclasses of that superclass. This type of constraint is an assertion that, among other possible values, the number of values of one of these subclasses is between some minimum and maximum cardinality.

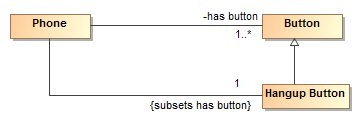


Figure Phone constraint: A phone must have a hangup button

For example, we may say a phone must have any number buttons with a “has buttons” property but exactly one of those buttons must be the “hang up button”. We would then define an unnamed property with the type “hang up button” that subsets the “has button” property with a cardinality of 1. If we wanted the hang up button to also define a new property, we would give that property a name.

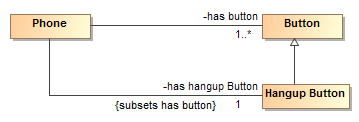


Figure Hangup button with new property

In the concept modeling interpretation of UML, subsetting or redefining a property without giving the new property a different name (or leaving off the new property name altogether) creates a constraint without defining a new property.

As {subsets} or {redefines} with an omitted name is not well defined in UML, in the concept modeling profile it is used to state that a subset of values must meet the stated cardinality and type constraints of the subsetting property. It does not create a new property, although it does create a context in which this constraint holds: the owning class and its subclasses.

The diagram below shows an existential quantification constraint on the global property “is conferred by” (from the property holder “Thing). The multiplicity is such that at least one of the instances of the property constraint must be one of the types in the union.

**Note that the property adding the constraint is unnamed. This is equivalent, in this case, to naming this property the same as the property being constrained (“is conferred by” from the property holder “Thing”).**

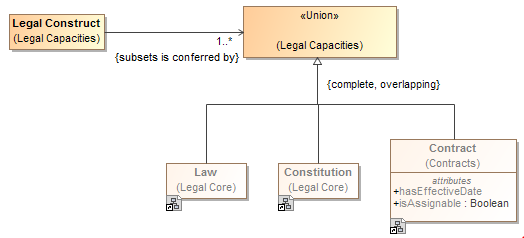
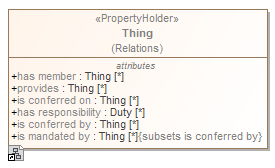


Figure Constraining a global property

### Tightening a property’s type

Sometimes it is necessary, in the context of some class, to constrain *all* the values of a property to a particular type. When defining a new property the type of that property assets that all values of that property must be of the given type. This is known as a *universal* quantification or *for-all* constraint (∀) in first order logic. This kind of constraint is an assertion that only values of the specified type are valid, and the number of values must be between some minimum and maximum cardinality.

Where all values of a property must be of a given types in a specialized property, UML *{redefines}* is used. In the concept modeling interpretation of UML, introducing a new property or redefining an existing property creates a universal quantification constraint in the context of the owning class. If the redefined property is given a name, a new property with the quantification is defined. If the redefined property does not have a name the existing property is constrained in the more specialized context (usually a subclass).

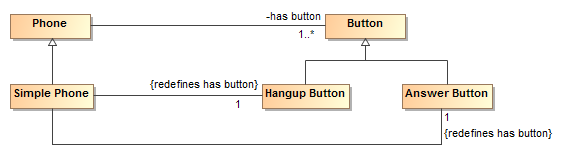


Figure Example of redefines

The example above shows a “simple phone” that has exactly two buttons and they must be an answer button and a hangup button. Since redefines is used, no other buttons are allowed.

The diagram below shows the introduction of a new property “consists of”, defining a universal quantification constraint on the property. The constraint states that, in the context of Soccer Team and any of its subclasses, all values of this property must be of the type “Soccer Player” and that there must be between 5 and 11 values of this property.



Figure Example of cardinality range

The diagram below shows a universal quantification constraint on the property “observer”. Where any occurrence can be performed by any actor, an observation must be performed by an entity in the role of observer.

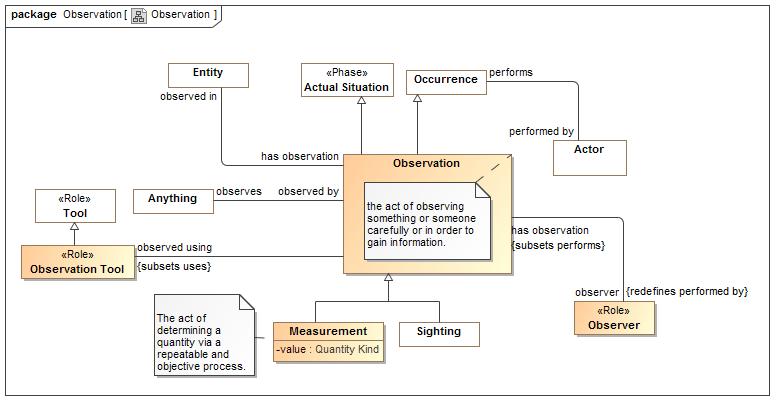


Figure Observation Example

### Inferring a type from its properties

A property's multiplicity or type is declared in the context of an owning class or a property holder. These declarations are always *necessary* conditions for an instance to be a member of the owning class, or, in the case of a property holder, for an instance to be valid at all.

Another kind of condition is known as *necessary and sufficient*. A class with at least one necessary and sufficient condition is known as a *defined* class, which means the differentiating characteristics of the class that make it distinguishable from its parent and sibling classes are defined.

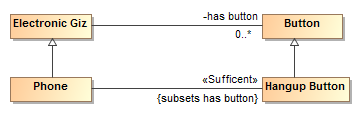


Figure Phone example for sufficient

The diagram above defines a phone as *any “electronic giz” that has a hangup button*. The existence of a hangup button is sufficient to know something is a phone.

In the concept modeling interpretation of UML, a property that has the <<Sufficient>> stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the <<sufficient>> property’s values, not only is a *necessary* condition for being an instance of the class met, a *sufficient* condition is also met to assume that the domain of that property is of that class. This necessary and sufficient condition allows an inferencing engine to classify that instance as a member of the class that owns the property. All <<sufficient>> constrains must be met for an instance’s type to be inferred.

## Conceptual Modeling UML Profile

The following documents the UML stereotypes defined for Conceptual models, as such not all of these stereotypes are necessarily used within the threat/risk model, but all are available for use. Each stereotype has a mapping to the SIMF Meta model and logical grounding.

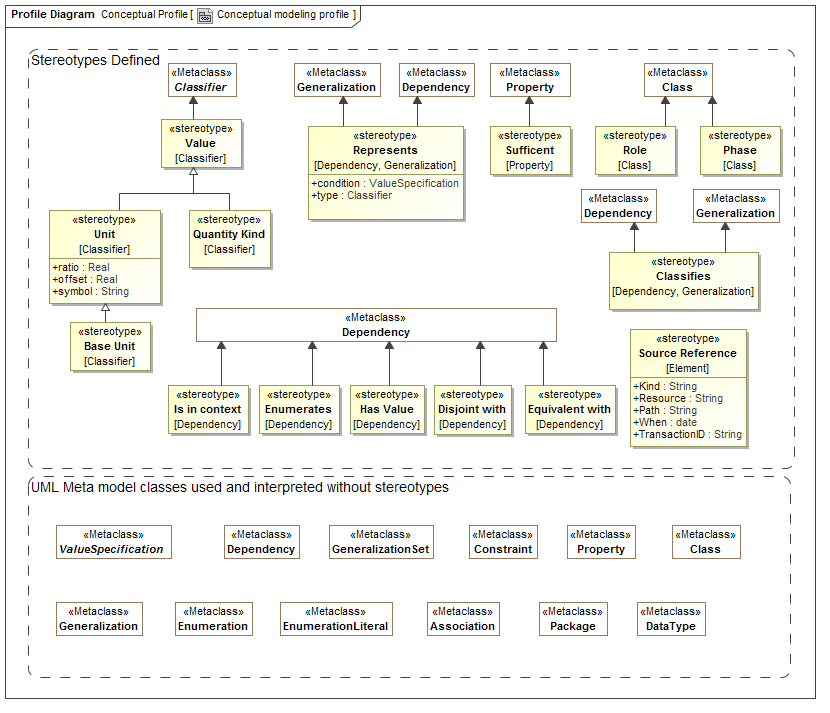


Figure Conceptual Profile

### UML Subset

The SIMF conceptual modeling profile uses a subset of UML, only the following concepts have an interpretation in a SIMF Conceptual Model:

* Association classes (Map to relations - however, all associations and properties are considered “first class”)
* Associations, associations ends {Map to Relation types}
* Cardinalities {Map to Constraint}
* Classes {Map to Entities}
* Data types and primitive types {Map to Values}
* Dependency {Map to in context}
* Enumeration {Map to class and closed set of instances}
* Generalization {Map to generalization}
* Generalization Set {Map to restrictions}
* Packages & Package URI {Map to Lexical and logical context}
* Properties for values {Map to Simple property relations}
* Property defaults {Map to a “Most” quantification}
* Realization {Map to Representation realizes concept}
* Structured classifiers {Map to templates}
* Subsets and redefines of association ends {Map to role generalization}
* Value Specifications {Map to expressions}

SIMF does not preclude other UML constructs; it just does not interpret them. Extensions of the SIMF profiles MAY interpret additional UML elements.

### Stereotype: Classifies [Generalization, Realization]

A classification defined by a <<Classifies>> generalization or realization is an arbitrary classification of an entity beyond any fundamental entity type. <<Classifies>> implies that the subtype is a classification. An instance must be typed by the classified Supertype for it to also be classified as the classifies subtype. A classification may be contextual, such as within a relation, situation and/or timeframe.

Examples

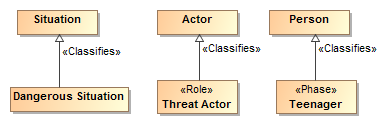


Figure Classifies Example

* A Dangerous Situation classifies a situation.
* Threat Actor is a role that classifies Actor
* A Teenager is a phase that classifies a person..

### Stereotype: Conceptual Model [Package]

A conceptual model represents signs, assertions and classifications for “real world” objects. A conceptual model models the world (or a conceptualization of it), not data or computation. Conceptual models are the foundation of this specification.

Properties

* URI:String (From UML) – the URI of the model

A unique logical identifier for the model.

Examples

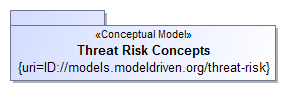


Figure Conceptual Model Package

The package “Threat Risk Concepts” is marked as a conceptual model. The URI for this model is: ID://models.modeldriven.org/threat-risk.

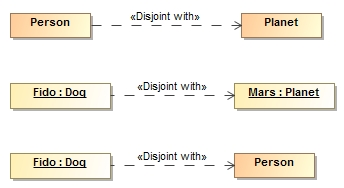
Note: As a convention we are using “ID” for “Identity” to distinguish such URIs from internet addresses but otherwise use the HTTP naming convention. The relationship between an ID and any physical resource is outside of this specification.

### Stereotype: Disjoint with [Dependency]

Disjoint specifies that the extent of real-world individuals for which the dependent elements are a sign, classification for or context of have no elements in common.

Note: We do NOT assume that anything has only one type or instance representation unless stated otherwise.

Examples



* No people are also planets
* Fido (A dog) is not Mars (APlanet). {the set of things that are Fido is disjoint with the set of things that are Mars}.
* Fido (A dog) is not a person (Even if Fido’s owner thinks he is). {the set of things that are Fido is disjoint with the set of things that are a Person}.

Disjoint may also be specified using generalization sets.

### Stereotype: Enumerates [Dependency]

Enumerates defines the set of acceptable values for a type, thus “closing the world” for instances of that type. The source of enumerates can either be an instance specification or a package containing instance specifications. The instance specifications must be of the type which is the target of the enumerates dependency. <<Enumerates>> is a more powerful representation of enumerations than a UML Enumeration data type in that <<Enumerates>> instance specifications are not restricted to primitive tag values, however UML enumerations may also be used where appropriate.

The enumerated values are only partially closed in that an additional model may enumerate additional values for a type.

Examples

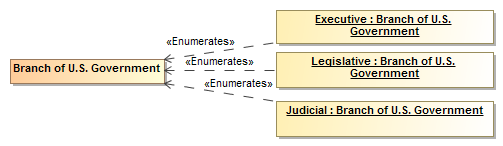


Figure Enumerates Example

### Stereotype: Equivalent with [Dependency]

<<Equivalent with>> is an assertion that two elements in a model or data resource represent the same thing or set of things. Unlike ontological languages it is not assumed that the two elements are consistent as statements from different context may or may not agree.

Examples

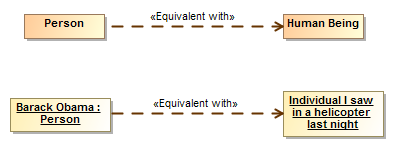


Figure Equivalent with example

* Person and Human are equivelent types, they have the same extent.
* Barack Obama is the person I saw in a helicopter last night.

See also

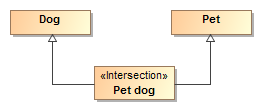
* ODM/OWL: SameAs, EquivalentClass, EquivalentProperty

### Stereotype: Intersection [Classifier]

An intersection is a class that has an extent which is the complete intersection of the extents of all supertypes. Intersection is a stronger statement than a subtype as a subtype may not be a complete intersection. If an instance has all supertypes the intersection type will be inferred.

For intersection, SIMF considers generalization and realization equivalent. This is due to ownership and legacy considerations in UML. Generalization is the preferred representation.

Examples



All dogs that are pets are pet dogs.

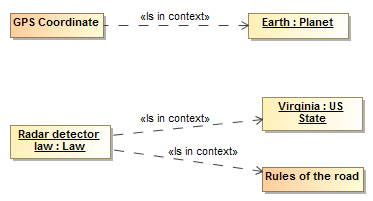
#### See also

* ODM/OWL: intersectionOf

### Stereotype: Is in context [Dependency]

<<Is in context>> is an assertion that the source of the relationship is in the context of the target of the relationship. All assertions and rules defined in the target context apply to the source and everything in the context of the source (in context is transitive). Packages, classes, situations and instances are typical context. Note that <<Is in context>> is the default interpretation of a dependency, if no stereotype is specified it will be interpreted as <<Is in context>>.

#### Examples



* GPS coordinates apply within the context of earth.
* The Radar detector law applies in the context of the state of Virginia and is in the context of rules of the road.

### Stereotype: Phase [Class]

Phase classifies some portion of an entities lifetime (Also known as a “State”).

Examples

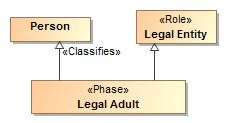


Figure Phase Example

Legal Adult is a phase (of a person) in the role of a legal entity. A Legal Adult represents a phase in the life of a person.

### Stereotype: Quantity Kind [Classifier]

<<Quantity Kind>> is an aspect common to mutually comparable quantities represented by one or more units. Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. E.g. temperature. [ JCGM 200:2008].

Quantity kinds are shown as black boxes.

Quantity kinds are the basis for units, typically Units <<Represents>> quantity kinds.

Direct Supertypes

* Value

Examples



Figure Quantity Kind Example

See also:

* JCGM 200:2008
* <<Unit>>

### Stereotype: Represents [Dependency, Generalization]

<<Represents>> is an assertion (rule) that the source type or feature provides a more concrete way to represent the target type or feature. Represents may be used within conceptual models or from a physical model to a conceptual model.

* A representation that is a dependency or realization makes no assumption that the types are substitutable, normally used between a logical/physical model and a conceptual model.
* A representation that is a generalization is substitutable for what it represents, normally used within a conceptual model.

Tag definitions

* Condition: Value specification – expression that must evaluate to true for the represents rule to be valid
* In context: Context – context that scopes the applicability of the representation (may also be specified as a dependency FROM the represents dependency rule).
* type: Classifier – Potentially more specific type that a property is mapped to

Examples

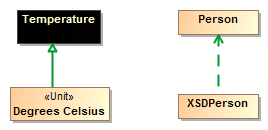


Figure Represents Example

* Degrees Celsius is a way to represent Temperature. Note that there could still be different ways to represent Degrees Celsius (e.g. As a real or integer)
* XSDPersonType (From a physical model) is a way to represent data about a Person (From a conceptual model)

### Stereotype: Role [Class]

A <<Role>> is a classification of an entity based on that entities behavior, participation in a situation or capabilities. A <<Role>> <<Classifies>> the types that may play that role.

Examples

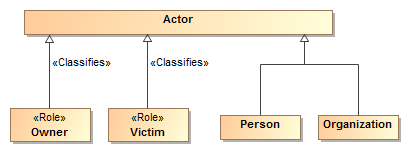


Figure Role Example

Owner and Victim are subtypes of an actor, which could be a person or organization.

See also:

* <<Classifies>>

### Stereotype: Source Reference [Element]

<<Source Reference>> provides traceability to the source of a piece of information in some information resource to facilitate provenance. Source Reference is a statement about the model data and has no semantic implication. Source reference may impact the trust in a statement but the evaluation of trust is outside of this specification. Source is typically populated by the SIMF tooling.

#### Tag definitions

* Kind:String Specifies the kind of resource being referenced. (E.G. “XML”). The values of “Kind” are implementation specific.
* Resource:URL Specifies the location of the resource
* Path:String specifies the location of the information in the source. The form of expression of the path is dependent on the Kind. Kinds and path languages are not defined in this specification.
* When: DateTime Specifies when the information was first asserted in the resource.
* TransactionID:String Transaction ID in which the information was created.

### Stereotype: Sufficient [Property]

The <<Sufficient>> stereotype of a property (or association end) indicates that the existence of the property implies the type of the properties subject. In logic this is known as a *necessary and sufficient* constraint. A class with at least one necessary and sufficient condition is known as a *defined* class, which means the differentiating characteristics of the class that make it distinguishable from its parent and sibling classes are defined.

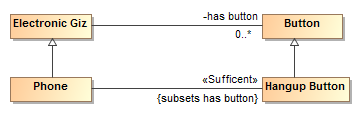


Figure Phone example for sufficient

The diagram above defines a phone as *any “electronic giz” that has a hangup button*. The existence of a hangup button is sufficient to know something is a phone.

In the concept modeling interpretation of UML, a property that has the <<Sufficient>> stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the <<sufficient>> property’s values, not only is a *necessary* condition for being an instance of the class met, a *sufficient* condition is also met to assume that the domain of that property is of that class. This necessary and sufficient condition allows an inferencing engine to classify that instance as a member of the class that owns the property.

**All <<sufficient>> constrains must be met for an instance’s type to be inferred.**

### Stereotype: Union [Classifier]

A <<Union>> is any context (Package or Classifier) that has an extent which is the complete union of the extents of all contexts that realize or specialize the Union.

For union, SIMF considers generalization and realization equivalent. This is due to ownership and legacy consideration in UML which do not allow a Supertype to be introduced to a class in another model.

Example



A Marine Mammal is either a Dolphin, Orka or Whale.

See Also

* OWL/ODM: UnionOf

### Stereotype: Unit [Classifier] and Base Unit [Classifier]

Units define real scalar quantities; defined and adopted by convention, with which any other quantity of the same quantity kind can be compared to express the ratio of the two quantities as a number. E.g. Degrees Centigrade, Miles.

Each unit represents a quantity kind using generalization and is thus substitutable for that quantity kind. Typically quantity kinds are used in conceptual models and units in data models.

One unit of a quantity kind may be marked as the base unit. The base unit provides the basis for conversions between units of the same quantity kind. The base unit always has a ratio of one and an offset of zero.

Direct Supertypes

* Value

Tag definitions

* Symbol – the accepted symbol for a unit.
* Ratio – the multiplier by which to multiple the unit to convert to the base unit
* Offset – the difference between zero in the unit and zero in the base unit after the ratio is applied

Examples

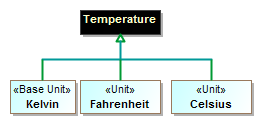


Figure Unit Example

The above defines Kelvin, Fahrenheit and Celsius as units of the quantity kind Temperature. Kelvin is defined as the base unit.

See Also

* Quantity Kind
* Value

### Stereotype: Value

A value is a type representing an atomic unit of information without independent identity. Values include numbers, strings and enumerations. In some cases values may have internal structure. Quantity kinds and units are also values. Values may stereotype any classifier. UML Data types, including primitives and enumerations, are implicitly values.

Examples



Figure Value Example

See Also

* Quantity Kind
* Unit
* Base Unit

### Stereotype: Has value [Dependency]

<<Has value>> identifies an instance specification that contains slots with acceptable values for properties of a type.

Where there is more than <<Has value>> for the same type, if any one <<Has value>> for the same property value is satisfied, the restriction is satisfied.

<<Has value>> is different from enumerates in that enumerates define complete instances that are classified by a type and closes the set of possible instances of a type. <<Has value>> defines property values that must be satisfied by any instance of a type, the set of instances is still open.

Examples

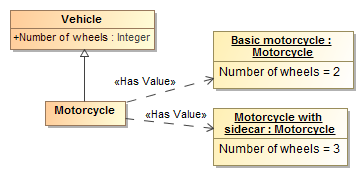


Figure Has Value Example

A motorcycle is a vehicle with 2 or 3 wheels.

See also:

* OWL has value
* Enumerates

## UML Profile – SIMF Rules & Model Mapping Primer

Rules provide a general framework for stating the consistency of and between SIMF elements. The primary use of consistency rules is for mappings between data models and conceptual models however a <<Rule>> may be used to assert consistency within a model, for example to represent generic assertions such as “all birds have feathers”.

Mapping rules define how a particular data model or schema <<*Represents>>* information about the concepts defined in conceptual models. This facilitates an “n-way” mapping of information represented using different data models. Since conceptual models are not data models they do not have any particular representation for “data instances” of that model. Instances of a conceptual model would be the real things in the real world. The real-world concepts are the “pivot points” between the data representations. Of course implementations may automate data models that correspond closely to the conceptual model, but that is outside of this specification.

Due to the various ways to represent information, mappings can become complex. The UML representation of mappings simplifies these mappings and much as possible. Note that details of the mapping relations are defined in the profile specification.

### Rule Model

<<Rule model>> is a stereotype of a package to indicate that the contents should be asserted and validated as rules.

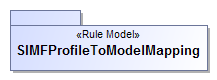


Figure Example Rule Model

The package SIMFProfileToModelMapping is a rule model.

### Representations

The foundation of mapping is the <<Represents>> dependency between classes. Represents says that a particular type found in a logical or physical model represents information about a concept in a conceptual model. By default, <<Represents>> does not implement the mapping, it defines what elements can be mapped and thus restricts mappings. For simple “one-one” mappings there is an optional tag for <<Represents>> to <<map-all>> known instances of one type to another.

Example

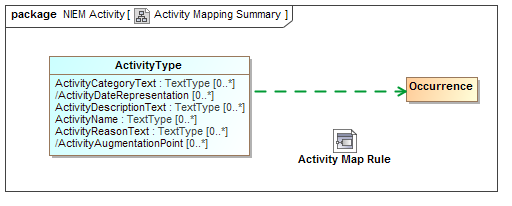


Figure Activity Mapping Summary Example

The above example shows that an “ActivityType” from NIEM-Core represents an Occurrence as defined in the threat/risk conceptual model. By convention we show the represents dependency as a green dashed arrow and do not explicitly show the represents stereotype. Representations provide the highest level of mapping. The diagram also shows that that there is a more detailed activity map rule which will map the properties and relationships between these types.

What this means is that *some* ActivityType instances represent *some* information about occurrences in the “real world”. This also implies that relationships to an occurrence can be validly mapped to relationships of an activity and that properties of an occurrence can validly be mapped to properties of an activity, <<Represents>> relations provide type-safety for mappings.

What this does not say is that ActivityType and Occurrence are equivalent and can necessarily be mapped 1..1. How they are mapped is detailed in mapping rules. However, if the <map-all> tag of <<Represents>> is set true then ActivityType and Occurrence will be mapped 1..1, bidirectionally (mapping of types and properties is considered independent, each property must also be mapped).

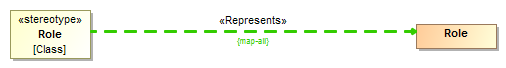


Figure {map all} Example

In the example, above, all UML classes stereotyped as <<Role>> will be mapped to the Role class in the SIMF model.

### Mapping Rules

The detail of mappings happens in classifiers stereotyped as <<Mapping Rule>>s. Mapping Rules define patterns of data types and patterns of concepts that have map correspondence rules. The map correspondence rules do the real work, mapping element by element.

Mapping representation rules are, externally, not that interesting. They are just a class (or component) stereotyped as <<Map Rule>>. However, note that Mapping Rules may specialize other rules – in which case they include the more general rule but may restrict the <<Match>> elements. Mapping rules may also <<Subsume>> other rules, in which case they take precedence over the other rule.



Figure Representation Rule External Example

The above defines a mapping rule for activities that is an assertion that the enclosed pattern must match and provides a context (in this case the enclosing package) where the map rules are also asserted. If we look inside the Activity Map Rule we see the structure and maps.

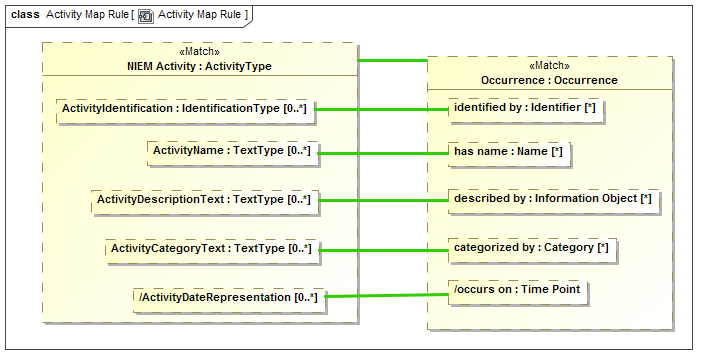


Figure Representation Rule Internal Structure

The above example is the internal “structure” of the Activity Map Rule. In this case the mapping is very 1..1 and simple. Inside of the rule we see “parts” that represent “ActivityType” named “NIEM Activity” and “Occurrence” named “Occurrence”. The green line between them is a “Map” rule, represented as a UML connector stereotyped as <<Map>>. This states that in this simple pattern NIEM Activities and Occurrences map 1..1. We could also have put filter constraints on that mapping, but in this case did not.

We also see the “<<Match>> on “Occurrence” and NIEM Activity. Match defines the “starting point” for the pattern with respect to the model containing the <<Match>> element. A mapping engine will find all instances of Occurrence (in any data format) and map those to NIEM Activity. It will also find all NIEM Activities and map them to Occurrences. All other parts of this mapping become relative to the “Match” elements.

Within both NIEM Activity and Occurrence we see other parts, parts of those types. The green lines create mapping assertions between those parts *within the context of this rule*. This within this rule “ActivityName: maps to “has name”.

A map correspondence is essentially “best efforts”, the types of the mapped elements must either match or have a mapping rule that allows them to be mapped. If, for example, an occurrence had an identifier that was an image and NIEM did not allow for image identifiers, that “fact” would not be mapped.

Mapping for primitive data types, such as strings and numbers, is provided by the mapping engine implementation. This allows, for example, an identifier that is represented as an integer to be mapped to a string.

The important point to remember is that mapping any fact requires that the types are compatible. That type compatibility is defined by <<Represents>> rules between the types. The requirement for type matching may be overridden by setting the <<coerce>> tag of the <<Map>> rule, but in most cases type safety of <<Map>> rules is desirable.

### Representation traversals and patterns

The above Activity Map Rule is simple and 1..1, when we get such a simple mapping we shout for joy – because our job is easy. However there is frequently complexity on “both sides” of the mapping – something in the data model may map to multiple things in the conceptual model or require a “Path” through multiple concepts. Likewise, there may be intermediate “technical artifacts” that have no real meaning in a conceptual model. This is why we say we are mapping patterns.

For our next example we will look at Incidents, which are a subclass of activities in NIEM and occurrences in the conceptual model. Since these are subclasses on “both sides”, we only need to describe the additional properties of an incident.

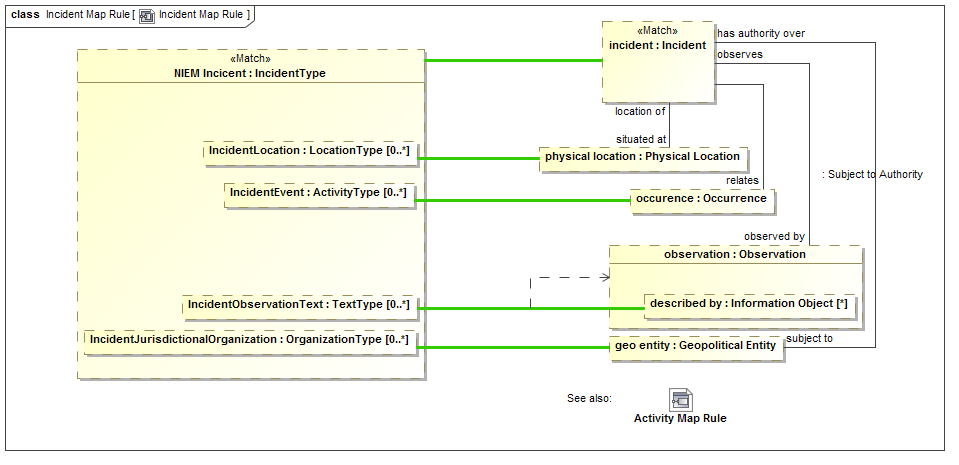


Figure More detailed map rule

The example above shows how a NIEM Incident (named NIEM Incident) maps to a conceptual incident (named “incident”). Incident Map is a subtype of Activity Map so the Activity map rules will all apply to incidents so we don’t need to repeat them here.

We will start with a <<Match>> of “Incident”. Note the line from “incident” to “physical location” labeled “situated at”. The mapping engine will start with an incident and fill in the set of “physical location”s iff the “situated at” relationship exists *for that instance* and *what it relates to is a Physical Location*. If that relationship does not exist, “physical location” will be null (empty). Note that physical location could also have multiple values since “situated at” does not have a restricted cardinality.

The values that “end up” in “physical location” will be mapped to “IncidentLocation” in NIEM. Likewise, any mapping in the other direction will hold – any populated “IncidentLocation” will populate “physical location” as well as the relationship to an incident. Once the rule is satisfied, the pattern will hold for all instances of NIEM IncidentType and Incidents.

Now consider the element “described by” within “observation”. This will be populated if the “has observation” relation exists from an incident and that instance has a “described by” property. IncidentObservationText is mapped to “described by” within such an observation. But, in this case, UML notation is a bit misleading, “described by” is a part of the Observation type, not this particular observation part. Since other objects in this rule may have a “described by” property it becomes non-deterministic which “described by” we are talking about. We want to say that we are mapping to the “described by” in the context of the “observation” part. The dependency from the green line to “observation” defines that the context of this map rule is only valid in the context of “observation”, thus making the map deterministic. As many context dependencies as are necessary may be specified for any map rule. All map rules are considered to be in the context of the enclosing Representation Rule. Some tools may report if a map is non deterministic.

### Representation Subsets

Conceptual models use sub classing, multiple inheritance, roles and phases to more accurately and intuitively represent the domain of interest. Many data technologies do not support these concepts and even if they did, would probably structure classes differently. In other cases, there may be restrictions on the “extant” of what maps to what that require calculations or other constraints. To provide for these cases we use <<Subsets>> in mapping patterns. A subset defines another part that holds a subset of the instances of the superset part, based on the type, relationship values and other constraints of the subset part.

To understand this feature we will first look at models for “Entity” and “Actor” in NIEM and the threat conceptual model, respectively.

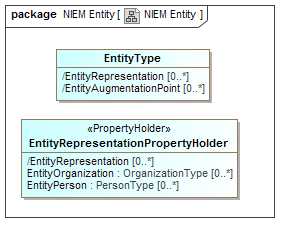


Figure NIEM Entity Example

In NIEM, an “EntityType” has a “substitution group” property with properties that can be “EntityOrganization” or “EntityPerson” to allow the entity to represent one or the other. The general rules for mapping NIEM state that substitution groups are considered subtypes of the primary type.

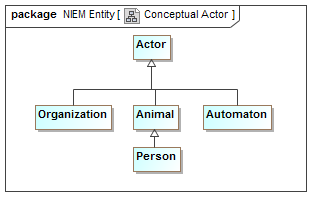


Figure Conceptual Actor Example

In the Threat conceptual model “Actor” is a Supertype of Organization and, indirectly person. It is also a Supertype of “Automaton”. An Automaton can’t be an actor in NIEM so it will not be mapped (However we could define a NIEM extension to allow this).

We want to map actors to NIEM entities, but see that they are very different “shapes”.

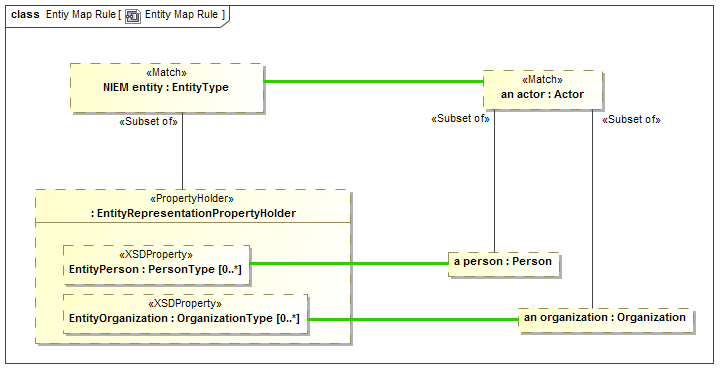


Figure Subset part example

In the above example we see the actor - EntityType mapping. Notice “a person” of type “Person”. “a person” is defined to be a <<Subset of>> actor – that is every actor that is of type “Person” will populate the “a person” part. If an actor is not a Person, “a person” will be null. “a person” is then mapped to “EntityPerson”, a property of “Entity” by way of the substitution group (sorry that this gets into some NIEM substitution group details, but you probably get the basic idea).

Likewise “an organization” will map to EntityOrganization iff “an actor” is an Organization. Note that if “an actor” is neither of these, it will not map to any NIEM entity property.

Note also that there could be other constraints on the subset parts, such as required relations or constraint expressions.

### <<Pattern Element>> computations and constraints

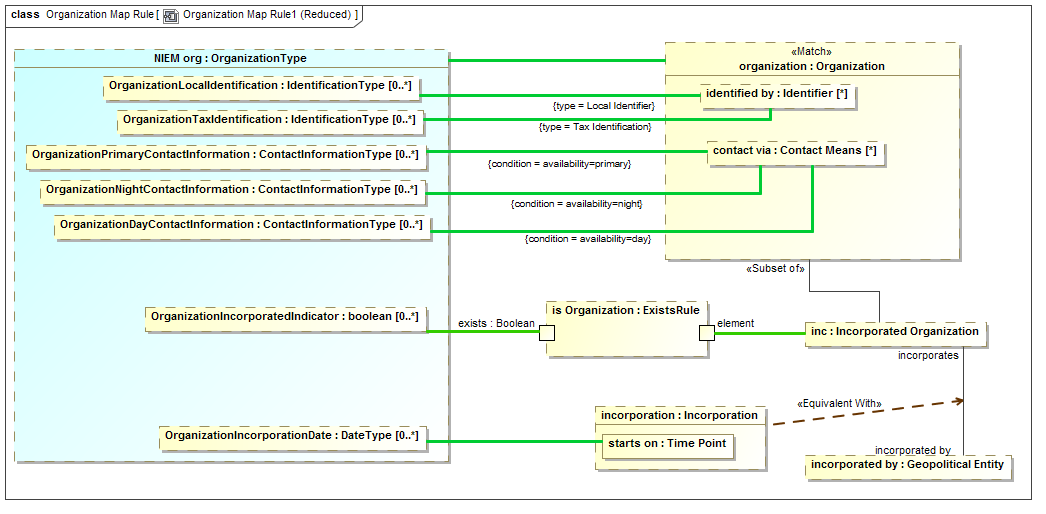


Figure Map constraints example

To continue the tour of the primary mapping capabilities we will look at a subset of the “Organization” mapping.

Note the “type=” on two maps to “identified by”. In the conceptual model there are subtypes of identifiers. In NIEM there are special properties for some of these identifiers. The “type=” constraint on a map says that the map will be constrained to the type (on the specified end) of the actual instance matched the specified type. So “OrganizationLocalIdentification” will only map to “identified by” if the type if the identifier includes “Local Identifier”. Likewise, “OrganizationTaxIdentification” will only map to “identified by” if the type includes “Tax Identifier” (remembering that a SIMF concept instance can have multiple types). Likewise the reverse is true; those properties will “assert” the type of the identifiers they reference.

On the maps to “contact via” we see “condition=”. Condition is a tag of <<Pattern Element>> that references a UML expression. The conditions referenced are properties of the association between an organization and “Contact Means”. The maps will be constrained to the “availability” property is set as indicated. Likewise if an organization is being created, that property will be set by the same condition.

Note that “inc” is a subset of an organization only if it plays the role of an “Incorporated Organization”. In NIEM there is a Boolean set if the organization is incorporated. The “ExistsRule” is a computation rule (that is its implementation is outside the specification). But in this case ExistsRule’s behavior is defined – the exists Boolean will be true when the mapped “element” has some value. This results in the NIEM “OrgainizationIncorporatedIndicator” corresponding to the organization being incorporated.

If the organization is incorporated it will have an incorporation relationship to its incorporating body (incorporated by). That incorporation relationship will contain its date of incorporation, which is mapped to the NIEM property. In UML association classes have to be put into a structure like this in two pieces, the “line” and the “box”. Since both the line and the box represent the same “fact”, they are asserted to be equivalent – this is only required when association class properties need to be accessed and is required because UML has no way to show connectors as association classes.

The end result is that the more “flat” representation of an Organization in NIEM is mapped to the conceptual structures.

### <<Pattern Element>> strength

A connector between parts will both follow a relationship and assert that relationship. There are times when the “strength” of a rule implied by a connector needs to be more explicitly specified. The <strength> tag of <<Pattern Element>> allows different options for what is asserted by a connector. These are: match, assert, default and exists. The default is “assert”. Definitions of each of these is in the reference section. In this section we will provide some examples.

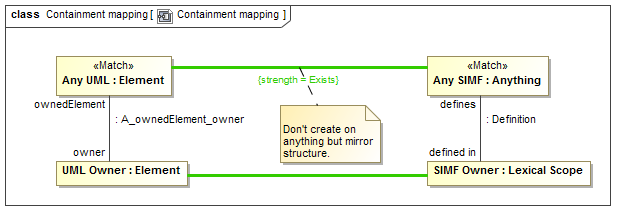


Figure strength = Exists example

In mapping UML to SIMF we do not necessarily want to map every element. But where there are mappings we would like to make the ownership hierarchies match. A match of “Any UML” has a <<Map>> to <<Any SIMF>> but the <strength>> is specified as “Exists”. This means that they should be mapped if the elements exist but the mapping should not be asserted by this rule (it may be asserted by other rules). Where they do exist, the UML owner should be mapped to the SIMF owner.



Figure strength = Default example

In the above fragment we several states/stated by connectors that have strength=Default. What is intended is that “states” (which defined model organization) will be asserted by this rule only if the value is not asserted by some other rule.

### <<Pattern Element>> quantifier

There are times when we need to define generic rules, not specific to a mapping or populate rule variables based on entire extent of a type, or some constrained subset of the extent. In logic this is done with “quantifiers”. A <<Pattern Element>> may have a <quantifier> tag that specifies the content of a pattern property based on this full extent.

The quantifier can have values of: None, There Exists, Exactly One, Most, All or Match. Note the close correspondence of quantifiers with those in predicate logics (with some extension). Definitions of each of these is in the reference section.

The quantified properties may then be used as the ends of relationships, making assertions about the set of quantified things.

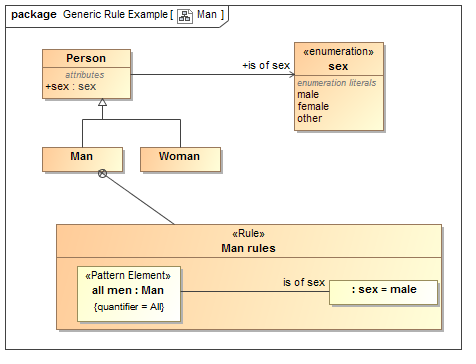


Figure Generic Rule Using Quantifiers

In the above example we assert the fact that “all men <is of sex> Male” in a <<Rule>> called “Man rules”. This rule is owned by (and therefor in the context of), the class “Man”.

The pattern element named “all men” has a type of “Man” and a quantifier=All. This means that the element will be “filled” with all men, statements made in this rule will apply to all men. We then have a connector typed by <is of sex> to “Male”. This asserts that all men are male.

Note that quantified elements can also have connectors to other quantified elements, so we could say things like “all men like Most tools” or “John likes all supermodels”.

### <<Pattern Element>> explicit

Most elements are mapped regardless of their source – explicitly asserted in a model or derived based on rules. There are times where only explicitly asserted elements need be mapped. In this case the element is marked with the <explicit> tag as TRUE.

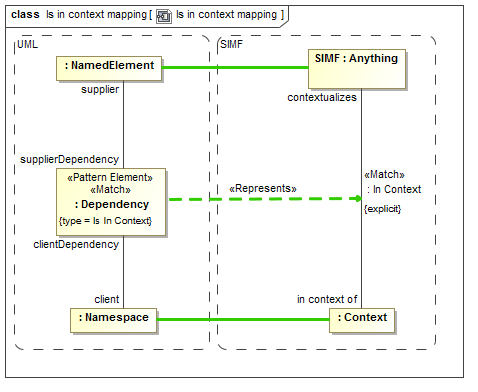


Figure Example of "explicit" pattern elements

The above example shows that the “in context of” relationship in SIMF should only be mapped to UML if it is explicitly asserted.

### Facades and Representation Computations

In some cases, it is desirable to have mapping rules as “reusable pieces” that can provide a “Face” to the conceptual model that fits better for one or more mapping rules. There is also the case where these rules fall outside of the expressive power of mapping rules and are best done in calculations (program code or fUML models).

Facades provide for making a new “face” of either a conceptual model or data model element. A Façade is a class with additional properties and/or relations that can be derived from the element it represents. Either mapping rules or computations are then used to “populate” the façade or map the façade back to what it represents.

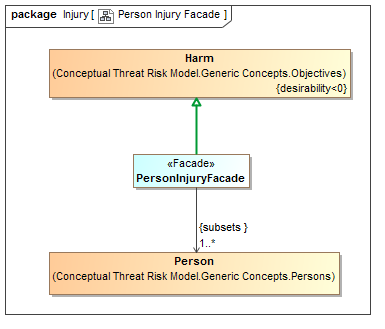


Figure Facade Example

The “PersonalInjuryFacade” above represents the concept of “Harm” but only where the harm impacts a Person. In NIEM, injury is only considered relative to a person – so this façade provides such a “View” of the conceptual model, harm restricted to personal injury. In this case no additional representation rule is required, but such a façade could also define new properties or associations that would be populated in the same way as a data model.

Note that in this case the <<Represents>> relation is applied to a generalization to assert that “PersonInjuryFacade” includes all of the features of “Harm”

Facades can also use “Computations” or Representation Rules do define their properties.

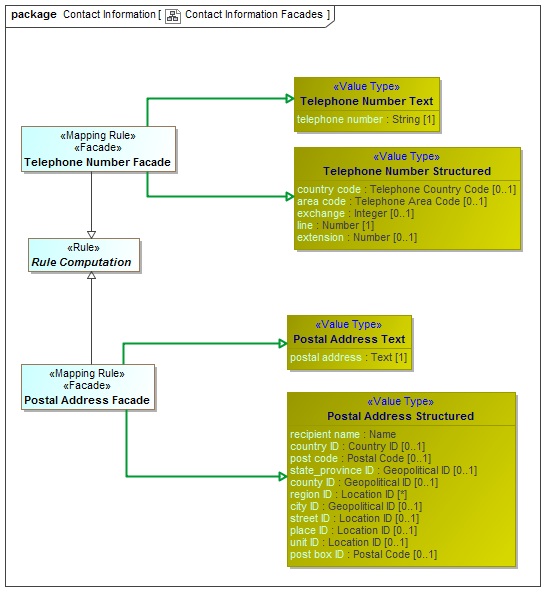


Figure Computation Facade Examples

In the above example both a telephone number façade and address façade are “computed” based on combining both a structured and unstructured representation of telephone numbers and addresses. The specific computation is external to the specification and defined by implementations. These implementations could be implemented in any language, including “ALF”, the executable language of UML.

The mapping engine is responsible for implementation of computation behavior and should update a computed Façade whenever any of its elements changes (some implementations may group such changes in a transaction).

In summary, facades and computations provide for reusability and extensibility of mappings.

### Property Chains

Rules may also be used within a conceptual model, an example being the “property chain” concept from OWL which allows a “path” through properties to be summarized by another property.

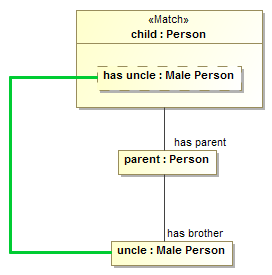
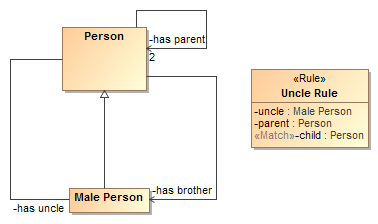


Figure Property Chain Example

In this example we see a simple model of a person with parents and male people that can be brothers or uncles. The “Uncle Rule” states that the “path” through “has parent” to “has brother” <<Map>>s to “has uncle”.

The above is one way to specify a chain, another would be with the <chain> tag of an <<Equivalent Property>> which is more compact but less powerful.

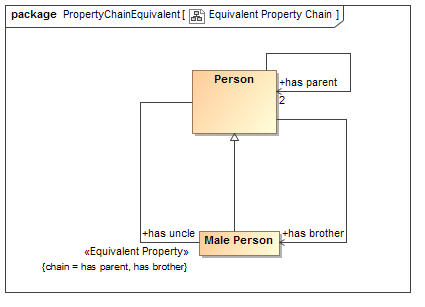


Figure Property Chain as Equivalent Property

The above example says the same thing as the property chain rule, using the more compact <chain> tag of <<Equivalent Property>>

## SIMF Rules Profile Specification

This section specifies the stereotypes specific to rules and data model mapping.

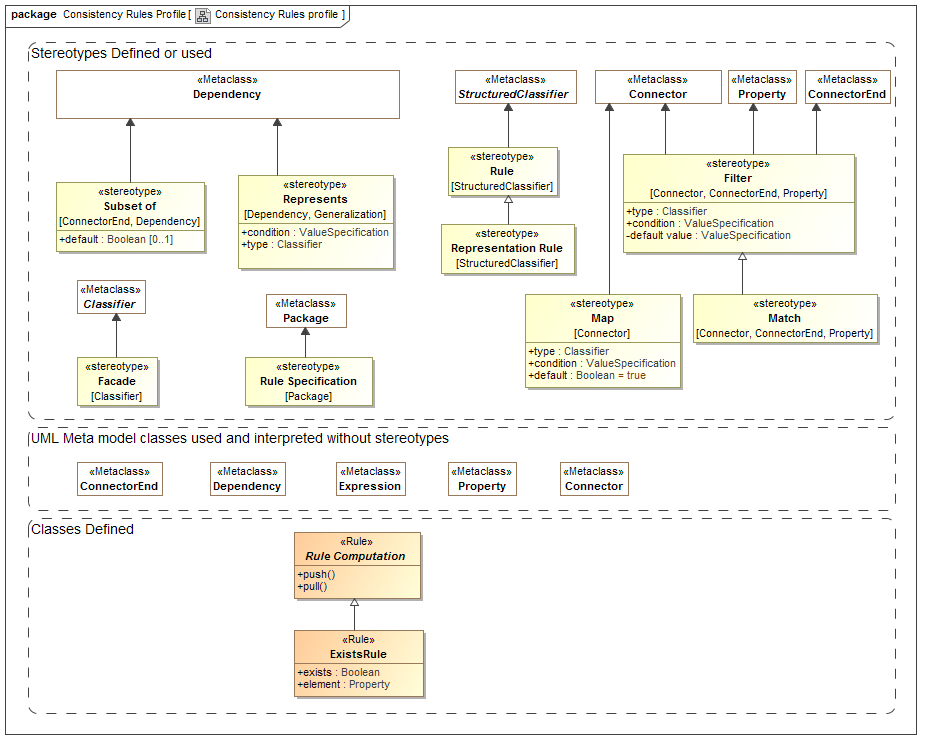


Figure Rules Profile

### Class: ExistsRule

ExistsRule is a rule to map the existence of an element to a Boolean. exists is true if element is not null. ExistsRule is used within Representation Rules.

Properties

* Exists : Boolean – true when the part mapped to element in not null.
* Element : UML::Property – reference to a property for which the NULL value corresponds to the truth value of Exists.

Example

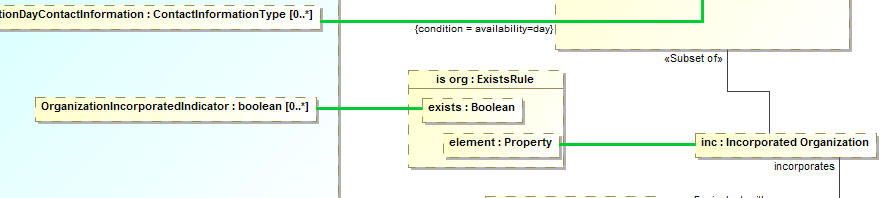


Figure Exists Example

The above will make the Boolean “OrganizationIncorporationRule” correspond with the organization being an Incorporated Organization”.

### Stereotype: Facade [Classifier]

Façade classifies an element as being a view of (facade of) one or more other elements. Facades usually define additional properties that match some external view of a conceptual model element and then ”fill in” those elements on one of several ways.

A Façade should use a generalization with a <<Represents>> stereotype to specify the class for which it is a façade.

Note that a façade may be used as a pure specification or may actually be instantiated, depending on implementation styles.

Façade’s will use one of three methods to relate the façade properties to the conceptual Model:

* A <<Representation Rule>> - where the representation rule defines mappings
* Subclassing “Representation Computation” – where the computation methods implement the façade.
* Constraints on the definition of the façade to subset the represented concept.

These methods may be combined.

Examples

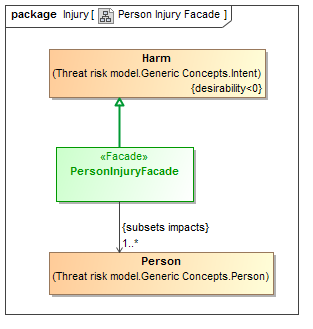


Figure Facade constraint example

The above “PersonInjuryFacade” of “Harm” will be a subset of Harm such that the harm is to a person.



Figure Computation Facade Examples

The above “Postal Address Façade” and “Telephone Number Façade” will be populated by the execution of the “push() and pull() methods as defined in “Representation Computation”. In these cases the façade will convert between a textual and structured representation of phone numbers and addresses.

### Stereotype: Filter [Property, Connector, ConnectorEnd]

<<Filter>> categorizes a connector or property within a rule as having a condition (A Boolean expression restricting the population of elements) and/or a default (an expression creating a new object where one does not exist). The expression syntax may be ALF or OCL. Filter may also provide a default value where the filtered element is null.

Filter may be used on the end of a <<Map>> to qualify the map.

Tag Definitions

* Type – a required type of the element
* Condition – an expression (ALF or OCL) returning a Boolean. If the condition is true the filtered element will hold.
* Default – an expression returning an instance to populate the element if that element is NULL.

*Please see Map Example.*

### Stereotype: Map [Connector]

<<Map>> defines a rule that maps the representation of elements between levels of abstraction, data representations or property indirections. A Map connector asserts that the values on both sides will be equivalent. A map is only asserted within the context of the enclosing rule where the “Match” pattern is valid.

A <<Map>> will also assert a conversion between types if any conversion rule exists between those types, so a map from a string property to an integer property will attempt a string/integer conversion.

**Map constraints**: The ends of a map (ConnectorEnd) may have a <<Filter>> defined to further constrain a map or the same properties may be set for the <<Map>> as a whole, in which case it applies to the conceptual model. If the filter type is set, the end must match the more constrained type specified. If the filter condition is set, the condition must hold for the map to be applied.  
**Map dependencies**: A map may also have dependencies to parts, the map must be in context of these parts. The mapping will be to the parts within the dependent parts. Dependencies are only required where the map would otherwise be indeterminate. The default context for maps to nested parts is the <<Match>> element. Map dependencies are usually required due to the non-obvious notation of UML where parts within parts simply showing parts of the type, not parts of the part.

Tag Definitions

* Type – a required type of the element in the conceptual model
* Condition – an expression (ALF or OCL) returning a Boolean. If the condition is true in the conceptual model, the map will hold.

Examples

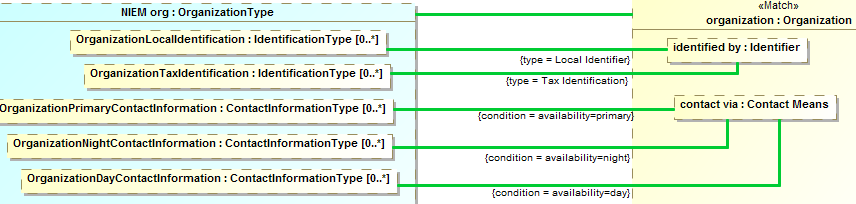


Figure Map examples

The above shows multiple “Map” connectors. Those to “identified by” will be constrained to instances where the identifier is of the indicated type. The “condition” maps will only hold to where the expressions are true or will assert the condition to be true.

### Stereotype: Rule Specification [Package]

A <<Rule Specification>> package represents rules for model consistency and mapping data between different models. Defining a package as a rule specification asserts the enclosed rules.

Properties

* URI:String (From UML) – the URI of the model

Examples



Figure Rule Specification Example

### Stereotype: Match [Connector, Property]

Match specifies an element or set of related elements in a representation rule structure that must match the model, these are the predicates of the rule. The <<Match>> elements form the foundation of the pattern to match and the root of any traversals. Match is s subtype of “Filter” and can utilize the condition and default tags. Match is bidirectional, the matching process may start at either “end” of the mapping.

Supertypes

* Filter

Example

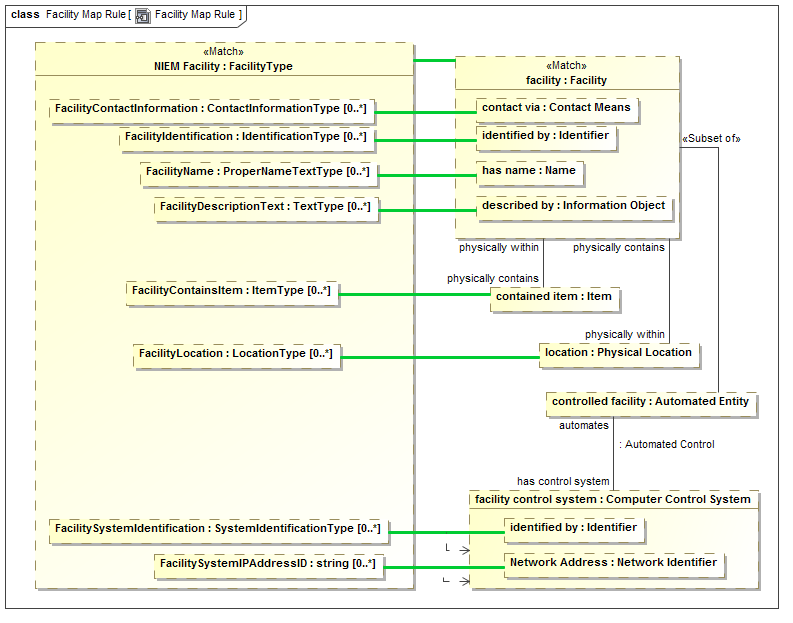


Figure Match Example

In the above example “facility : Facility” is specified as the <<Match>> part on the concept6ual model side. On projecting from the conceptual model all Facilities will match this element (since it has no other constraints). On production from “NIEM facility:FacilityType” the mapped FacilityType will be the basis for the match.

The facility will then be the root to traverse across the “physically contains”, “physically within” and “has control system” relationships.

### Class: Rule

A rule defines something which must hold true for the context of the rule.

A rule is a pattern structure described by a structured classifier that shows how elements are related. Each mapped must match, including any traversals through structures defined with properties and connectors. Such traversals are links which may also have filters to more precisely define the pattern. The mapping engine ensures that the patterns match, bidirectionally.

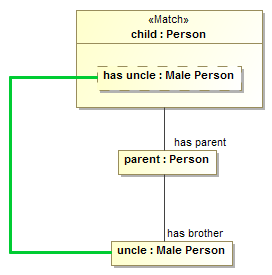
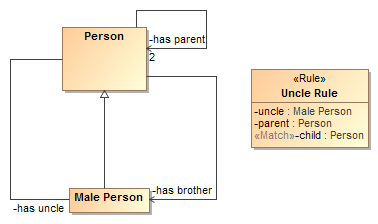


Figure Rule Example

The above defines an “Uncle Rule” that asserts that “has uncle” will have the same value as the “path” through “has parent” and “has brother”.

See also: OWL “property Chain”

### Class: Rule Computation

A rule computation is a class or facade that includes calculations as defined by push() and pull() methods. Rule computation allows for programmed computations. The implementation may be external or utilize ALF, the UML execution language. A Rule computation should <<Represents>> a class that will be the basis for computation. Facades that are to be calculated should subclass Rule Computation.

Operations

* Operation: push() - An operation called to evoke the behavior associated with a new facade element being created or modified.
* Operation: pull() - An operation called to evoke the behavior associated with a facade representing existing elements.

Please see example under “façade”

### Stereotype: Representation Rule [Structured Classifier]

A representation rule is a pattern structure described by a structured classifier that shows how both "sides" of a representation are related. Each "side" must match, including any traversals through structures defined with properties and connectors. Such traversals are links which may also have filters to more precisely define the pattern. The mapping engine must ensure that the patterns match, bidirectionally. Representation Rules may contain and contextualize other rules, in particular <<Map>> connectors are used within a representation rule to map between parts of the representation rule.



Figure Representation rule definition

A representation rule is just a class stereotyped as <<Representation Rule>>. The interesting part is the internal structure.

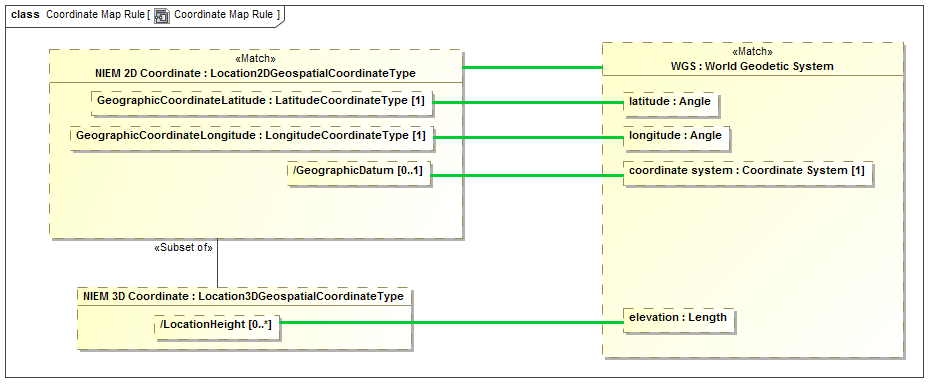


Figure Structure of example representation rule

The internals of a representation rule, shown as a UML composite structure diagram, contain parts and connectors that use the other stereotypes defined in this section. The “root” of the rule is marked with <<Match>>. The detailed work of mapping properties is done with <<Map>> rules inside of a representation rule.

### Stereotype: Represents [Realization, Dependency]

Please see the definition of <<Represents>>in the conceptual profile. This entry discusses use of <<Represents>> for mapping.

When used within a mapping specification the dependency form of <<Represents>> is used between a data model and a conceptual model. It states that the data model element is a data representation of the concept. <<Represents>> is usually the starting point for a mapping, where the “high level” types are connected.

Represented types are used to filter the relations between instances to only those relations that are valid for the represented types. For example, if any entity can be “related to” any other, and that relation is mapped to a property “relatedToIncident, with type incident, only the actual relations to an incident will be mapped – thus providing for mapping type safety.

If not other representation rules are defined for represented classes, the classes are defined to be mapped 1..1.

Tag Definitions

* Condition – a Boolean expression that must be true for the represents rule to be enforced.
* In context – other elements that must be in context for the rule to be enforced. In context may also be specified by an un-stereotyped dependency.
* Type – a constraint on the type of the represented element.

Examples

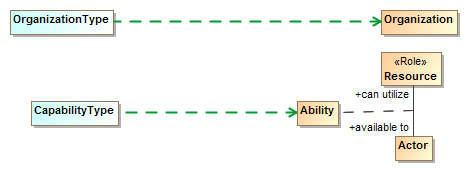


Figure Represents dependencies

The above states that OrganizationType is a data representation of an organization and that CapabilityType is a data representation of an ability ( an association class between a resource and an actor with that capability). Represented types are used to filter the relations between instances to only those relations that are valid for the represented types.

### Stereotype: Subset of [Connector End]

In a pattern or mapping rule, <<Subset of>> defines a part (UML Property) that represents a subset of another part. The subset may be constrained by a more specific type, expressions or required cardinalities. The <<Subset of>> stereotype is applied to the end of a connector that is the superset of the subset part.

A part within a structured classifier may hold any number of instances, including zero. The superset property is assumed to hold some value for which the subset property is a constraint, forming the subset. The subset may be constrained by its type (usually a subtype or role of the superset) or by required associations or filter conditions.

Tag Definitions

* Default:Boolean – indicates that the subset be a subset that excludes all other subsets of the same part.

Examples

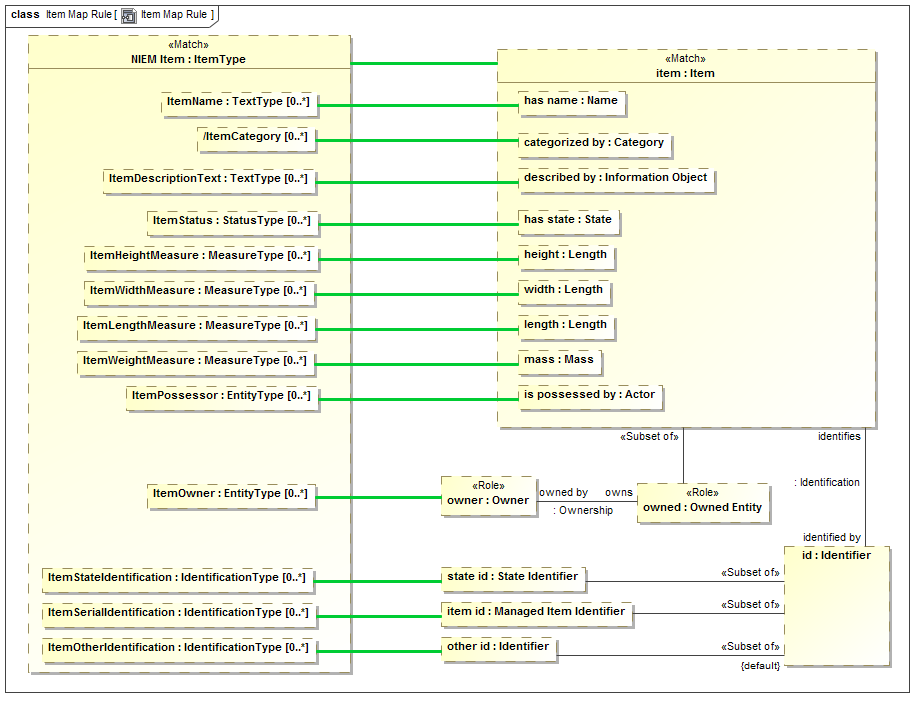


Figure Subsets Example

The above uses subsets for the following:

* “owns :Owned Entity” – is populated when item has a role of Owned Entity
* “state id” is a subset of “id” when the id has the type “State Identifier”.
* “item id” is a subset of “id” when “id” has type “Managed Item Identifier”
* “other id” is a subset of “id” less the subsets “state id” and “item id”.

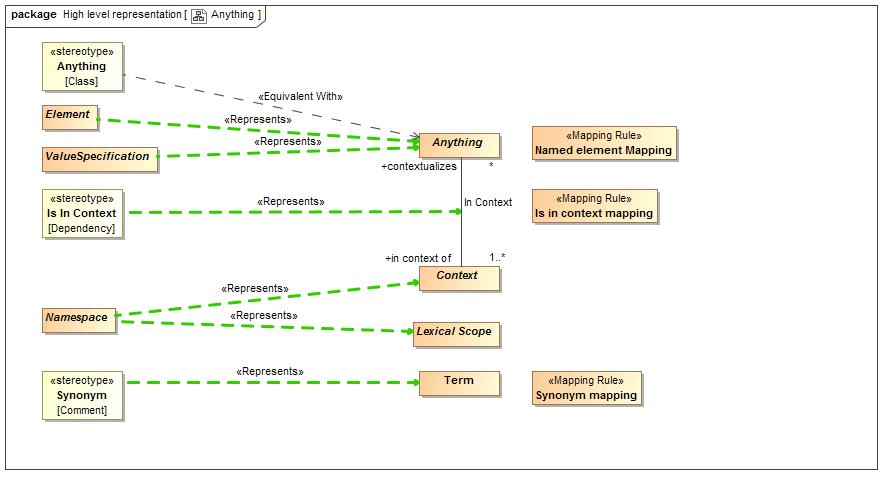
# Profile mapping to SIMF Model (Normative)

The following diagrams summarize the mapping which is further defined in the UML mapping model.

## SIMFProfileToModelMapping::High level representation

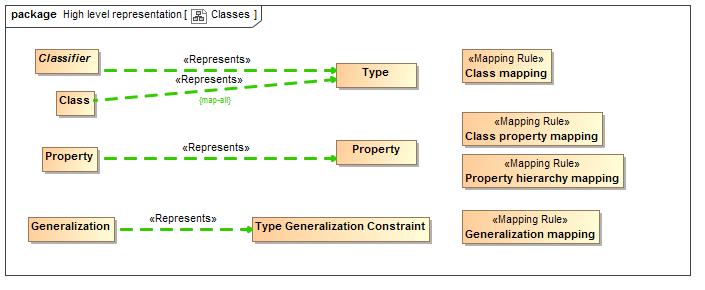
The following diagrams show the <<Represents>> rules defined between the profile and the SIMF model.

### Diagram: Anything



1. Anything

### Diagram: Classes



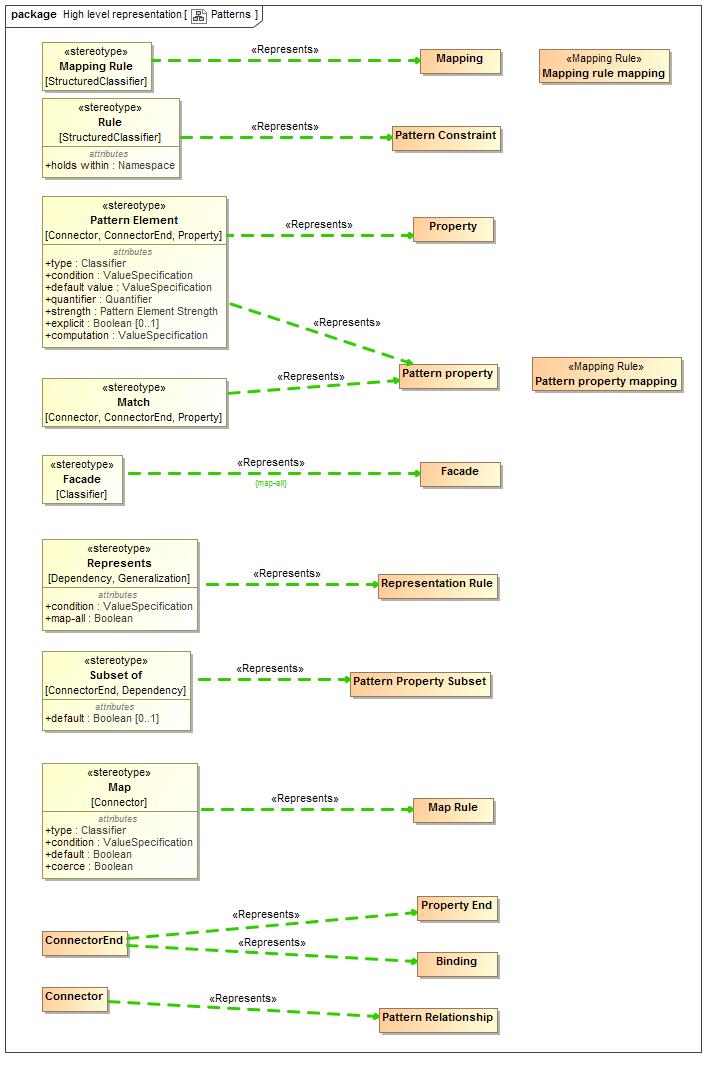
1. Classes

### Diagram: Lexical Structure



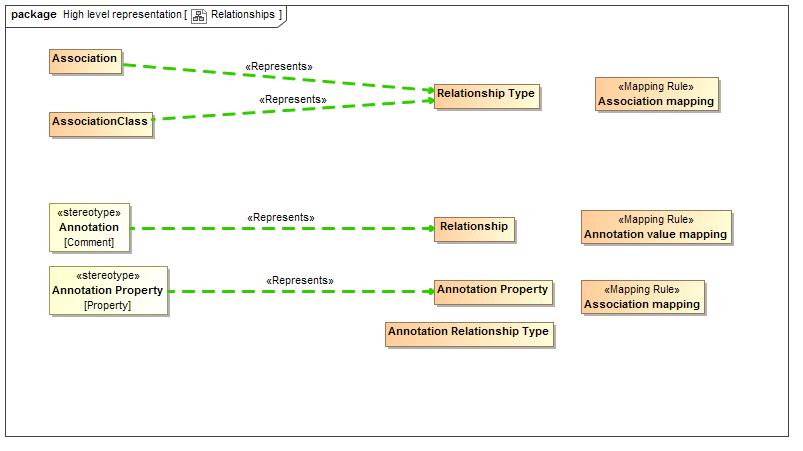
1. Lexical Structure

### Diagram: Patterns



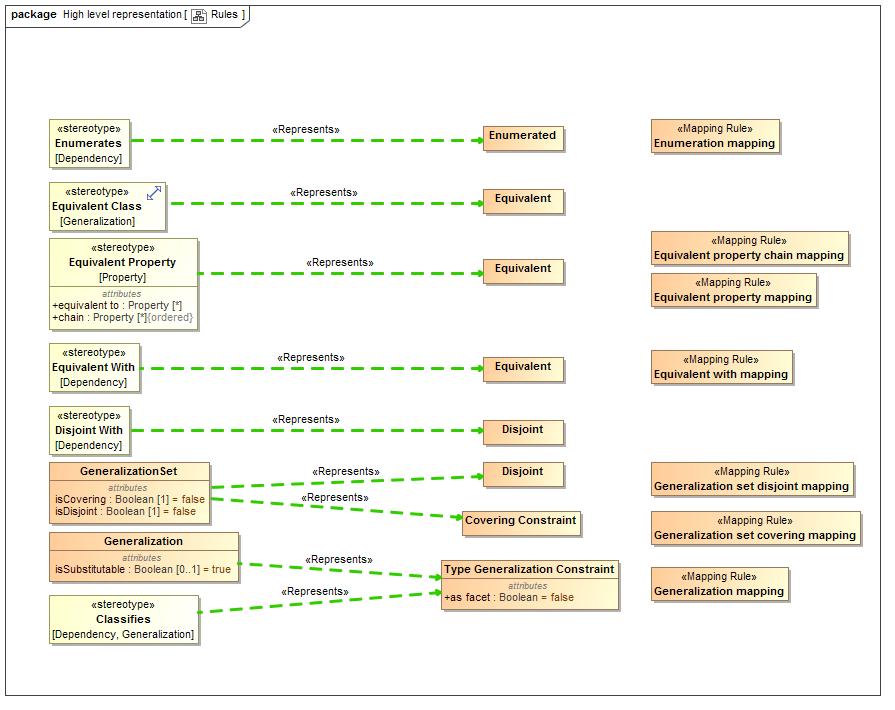
1. Patterns

### Diagram: Relationships



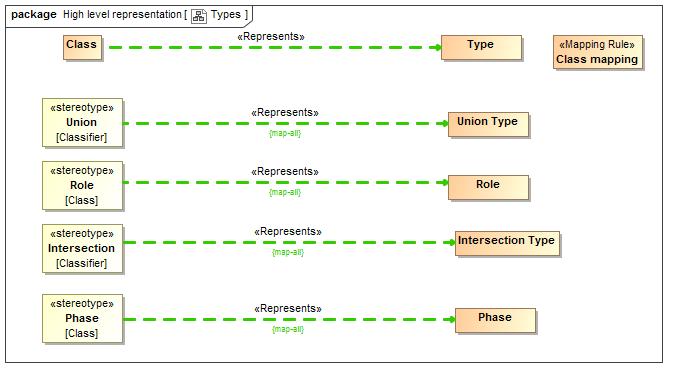
1. Relationships

### Diagram: Rules



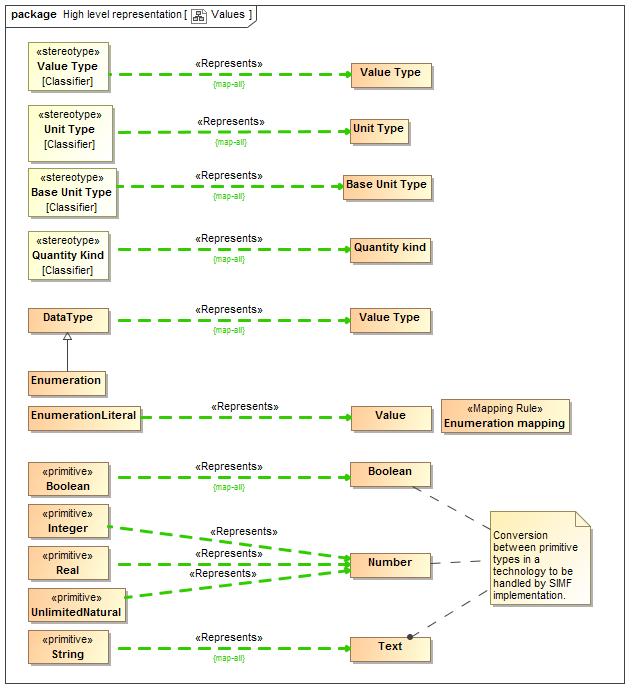
1. Rules

### Diagram: Types



1. Types

### Diagram: Values



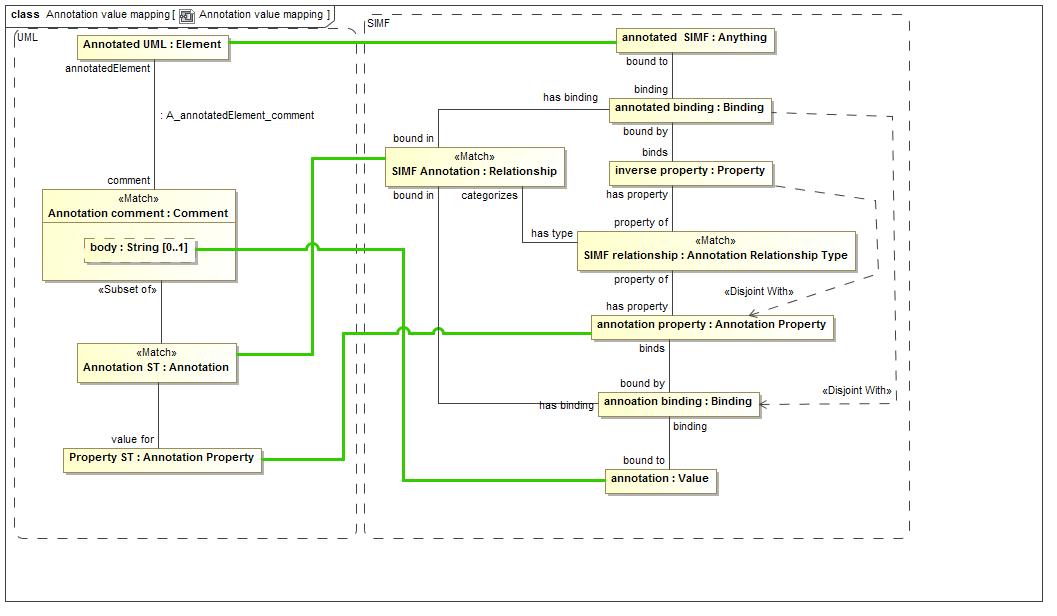
1. Values

## SIMFProfileToModelMapping::Mapping rules

The following are the mapping rules that hold for mapping the SIMF profile to the SIMF model.

## Class Annotation value mapping

Annotation value mapping defines a direct correspondence between a UML <<Annotation>> stereotype of a comment and a SIMF Relationship typed by a Annotation Relationship Type. It then creates the properties for the Annotation Relationship Type and binds the subject of the annotation to the annotated element and the annotation property to the string "body" value of the annotation.  
Note that an annotation defines a property instance, it does not define a new property.



1. Annotation value mapping

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg Annotation ST : [Annotation](#_f98249aabdb650274599d4c673302390)

-1626698621.jpg Annotation comment : [Comment](#_0be8406951cdfda82f00f79328cf4efc)

-1626698621.jpg annotation property : [Annotation Property](#_630a40beaf4bbbac635cfcacaaf353d1)

-1626698621.jpg SIMF relationship : [Annotation Relationship Type](#_7a2b4b15c2377efe429d3d9e5b30b859)

-1626698621.jpg annotation : [Value](#_a739673c8d53da123e392b7e5059ceec)

-1626698621.jpg annoation binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64)

-1626698621.jpg Annotated UML : [Element](#_231afe47f3f37d3808096b36c28b4ded)

-1626698621.jpg Property ST : [Annotation Property](#_78cccb8676a335918ea57bcbf344cfcc)

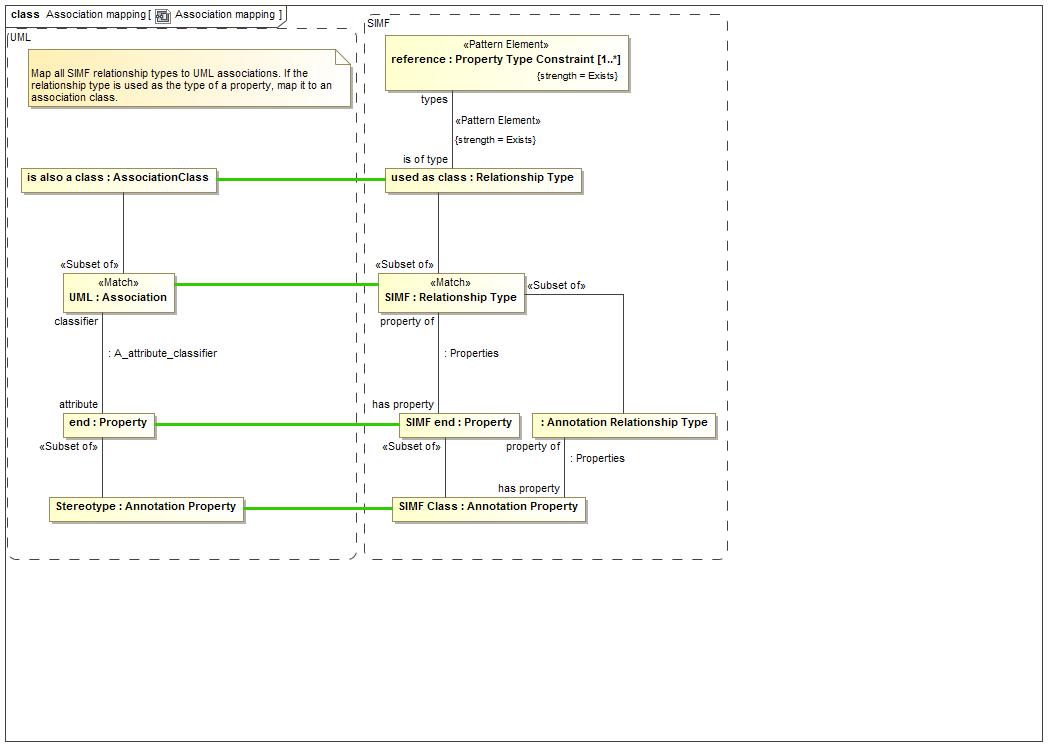
-1626698621.jpg inverse property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg annotated binding : [Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64)

-1626698621.jpg annotated SIMF : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-1626698621.jpg SIMF Annotation : [Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166)

## Class Association mapping



1. Association mapping

The Association mapping draws a direct correspondence between a UML association and a SIMF Relationship Type. It maps each property of the association to a SIMF property.  
SIMF does not distinguish between associations and association classes (all associations are essentially classes). As a convention, a UML association class will be created only when the association is used as an type of some other property.  
For annotations, UML <<Annotation Property>> corresponds with a SIMF Annotation Property as a property of a Annotation Relationship Type.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg UML : [Association](#_61d458d8946b919c7c2590e9d6cc0a57)

-1626698621.jpg is also a class : [AssociationClass](#_f7d352f165fcac36e06c87812e6c2546)

-1626698621.jpg SIMF : [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a)

-1626698621.jpg SIMF end : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg end : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg used as class : [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a)

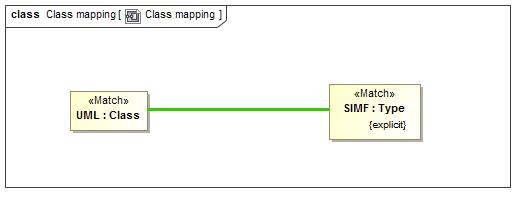
-1626698621.jpg reference : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [1..\*]

-1626698621.jpg Stereotype : [Annotation Property](#_78cccb8676a335918ea57bcbf344cfcc)

-1626698621.jpg SIMF Class : [Annotation Property](#_630a40beaf4bbbac635cfcacaaf353d1)

-1626698621.jpg : [Annotation Relationship Type](#_7a2b4b15c2377efe429d3d9e5b30b859)

## Class Class mapping



1. Class mapping

UML Classes correspond directly to explicitly asserted SIMF types.

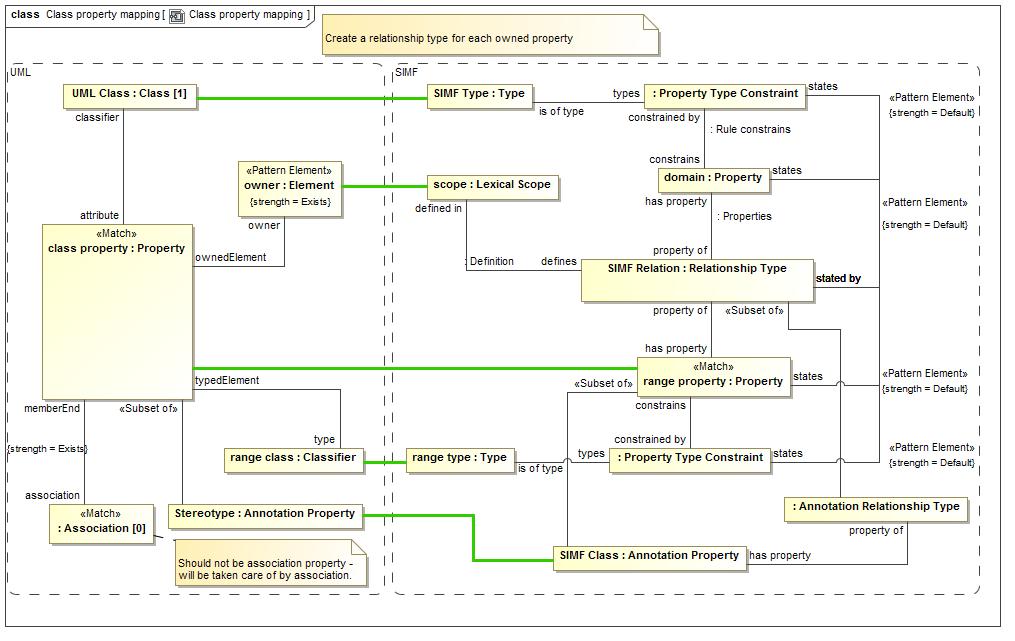
**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg UML : [Class](#_9bd81329febf6efe22788e03ddeaf0af)

## Class Class property mapping



1. Class property mapping

Properties of UML classes correspond with a pattern involving a relationship. SIMF conceptual models to not define properties directly on types but each is an independent relationship type.   
Each UML Property corresponds with a SIMF property with a type that matches the UML property type. The SIMF property becomes a property of a Relationship Type with a "domain" property corresponding to the UML class owning the property.   
UML properties marked as an <Annotation Property>> Correspond to a SIMF Annotation Property and further classify the relationship as a Annotation Relationship Type.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF Type : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg UML Class : [Class](#_9bd81329febf6efe22788e03ddeaf0af) [1]

-1626698621.jpg class property : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg range property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg : [Association](#_61d458d8946b919c7c2590e9d6cc0a57) [0]

-1626698621.jpg : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = false

-1626698621.jpg SIMF Relation : [Relationship Type](#_adbc34bb07fb06a9fcdba6f84fc4c37a)

-1626698621.jpg domain : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a)

-1626698621.jpg range class : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8)

-1626698621.jpg : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a)

-1626698621.jpg owner : [Element](#_231afe47f3f37d3808096b36c28b4ded)

-1626698621.jpg scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

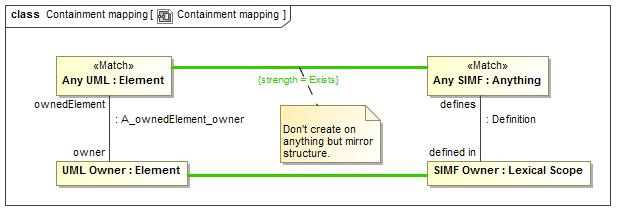
-1626698621.jpg range type : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg Stereotype : [Annotation Property](#_78cccb8676a335918ea57bcbf344cfcc)

-1626698621.jpg SIMF Class : [Annotation Property](#_630a40beaf4bbbac635cfcacaaf353d1)

-1626698621.jpg : [Annotation Relationship Type](#_7a2b4b15c2377efe429d3d9e5b30b859)

## Class Containment mapping



1. Containment mapping

Containment mapping forces the UML ownership structure and the SIMF lexical structure to match.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

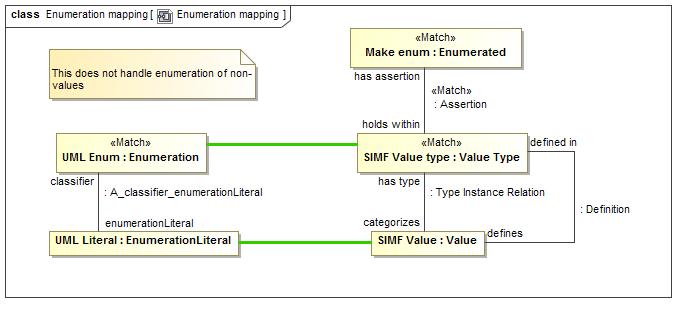
-1626698621.jpg SIMF Owner : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

-1626698621.jpg Any UML : [Element](#_231afe47f3f37d3808096b36c28b4ded)

-1626698621.jpg Any SIMF : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-1626698621.jpg UML Owner : [Element](#_231afe47f3f37d3808096b36c28b4ded)

## Class Enumeration mapping



1. Enumeration mapping

Enumeration mapping draws a correspondence between UML Enumerations and a SIMF Value Type with an "Enumerated" constraint.  
While SIMF may enumerate non-value types UML does not support this semantic.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg UML Enum : [Enumeration](#_2958e521f3da6b41059c4369a34a2a23)

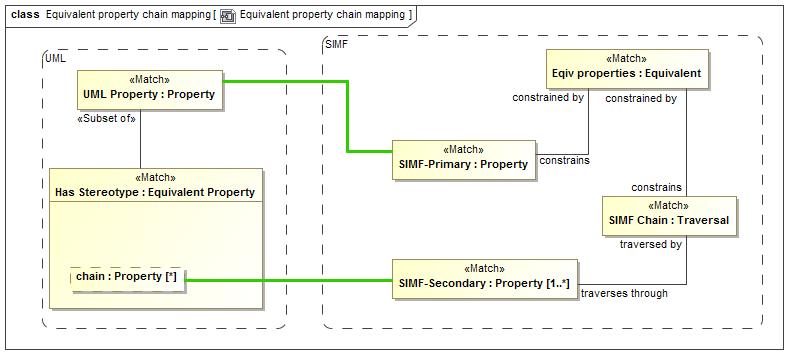
-1626698621.jpg UML Literal : [EnumerationLiteral](#_3aa592257b6491c626c04ffe97b5c062)

-1626698621.jpg SIMF Value type : [Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

-1626698621.jpg Make enum : [Enumerated](#_2ec7764a8ba3b29c599c56dc46fe72f6)

-1626698621.jpg SIMF Value : [Value](#_a739673c8d53da123e392b7e5059ceec)

## Class Equivalent property chain mapping



1. Equivalent property chain mapping

Equivalent property chain mapping maps the UML stereotype <<Equivalent Property>> with a "chain" tag to a SIMF "Equivalent" constraint constraining a SIMF Traversal of that chain.  
The stereotyped property corresponds with the <constrains> property of the Equivalent constraint.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg Has Stereotype : [Equivalent Property](#_36f5b7605ddbf56b1831c49db7afc07c)

-1626698621.jpg SIMF-Secondary : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

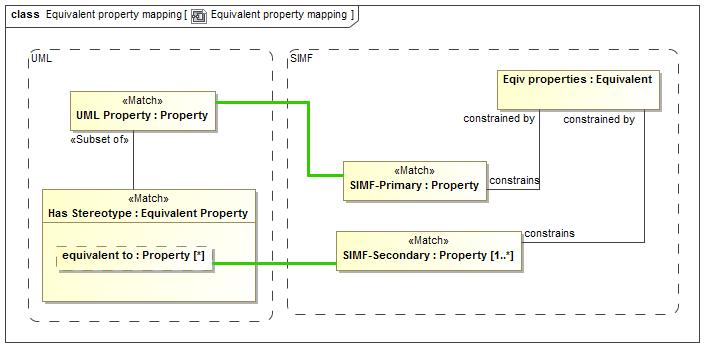
-1626698621.jpg Eqiv properties : [Equivalent](#_10a243d8b2330fdc0ff939b6c6625e89)

-1626698621.jpg UML Property : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg SIMF-Primary : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg SIMF Chain : [Traversal](#_0492440b12b90a76377a15324efa2182)

## Class Equivalent property mapping



1. Equivalent property mapping

Equivalent property chain mapping maps the UML stereotype <<Equivalent Property>> with a <equivalent to> tag to a SIMF "Equivalent" constraint constraining a SIMF Secondary property.  
The stereotyped property corresponds with the <constrains> property of the Equivalent constraint.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg Eqiv properties : [Equivalent](#_10a243d8b2330fdc0ff939b6c6625e89)

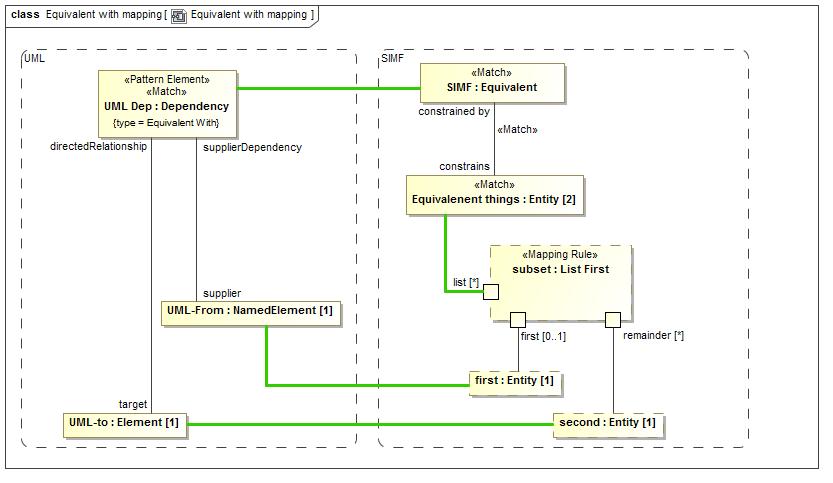
-1626698621.jpg Has Stereotype : [Equivalent Property](#_36f5b7605ddbf56b1831c49db7afc07c)

-1626698621.jpg UML Property : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg SIMF-Primary : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg SIMF-Secondary : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]

## Class Equivalent with mapping



1. Equivalent with mapping

Equivalent with draws a correspondence between a UML dependency stereotyped as <<Equivalent With>> and a SIMF equivalent constraint with exactly 2 constrained elements. The first element is mapped to the dependency supplier and the second element to the dependency target.  
The "List First" rule is used to divide the list elements.  
Note that SIMF Equivalent constraints with more than 2 constrained elements will map to a generalization set.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg UML Dep : [Dependency](#_90a95d6639a7bbbeff7f36a7ec8f3b10)

-1626698621.jpg SIMF : [Equivalent](#_10a243d8b2330fdc0ff939b6c6625e89)

-1626698621.jpg Equivalenent things : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [2]

-1626698621.jpg UML-to : [Element](#_231afe47f3f37d3808096b36c28b4ded) [1]

-1626698621.jpg UML-From : [NamedElement](#_705b0620812f5826ea76b12f804e54bb) [1]

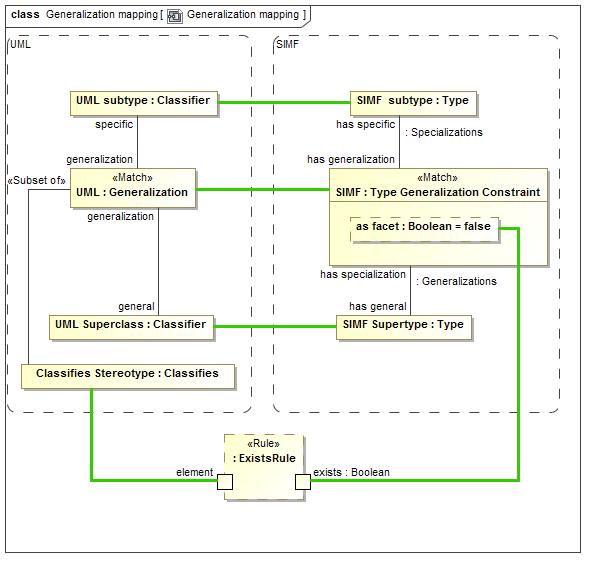
-1626698621.jpg first : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [1]

-1626698621.jpg second : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [1]

-1626698621.jpg subset : [List First](#_484cfaf11b717803a5dfed5261713884)

## Class Generalization mapping

The generalization mapping rule draws a direct correspondence between a UML Generalization and a SIMF Type Generalization Constraint.  
  
The UML <<Classifies>> constraint corresponds with the "as facet" boolean of the SIMF constraint.



1. Generalization mapping

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF subtype : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg UML subtype : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8)

-1626698621.jpg UML Superclass : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8)

-1626698621.jpg UML : [Generalization](#_e60de48d98369f3b6276ac9d9422383e)

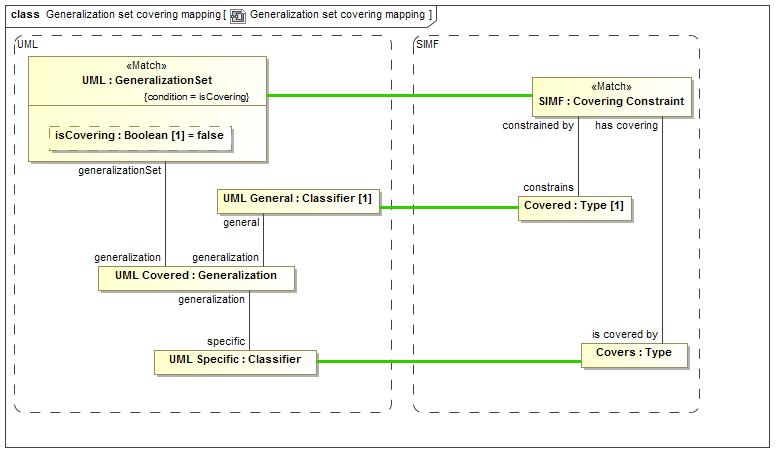
-1626698621.jpg SIMF : [Type Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf)

-1626698621.jpg SIMF Supertype : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg Classifies Stereotype : [Classifies](#_34e602a3e6c4f12fd1baa7ab4b38b4de)

-1626698621.jpg : [ExistsRule](#_4dcf4b5884925d444819a469fd70ce00)

## Class Generalization set covering mapping



1. Generalization set covering mapping

Generalization set covering constraint draws a correspondence between a generalization set where isCovering is true and a SIMF Covering Constraint.  
The generalization set has a set of UML covered <<Generalization>>s, all of which must have the same <general> Classifier, which is mapped to the <holds within> type.  
The <specific> classifiers correspond to a set of <is covered by> types.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg UML : [GeneralizationSet](#_ff83bc3640ce7e2f1d38966bb69d76df)

-1626698621.jpg UML Covered : [Generalization](#_e60de48d98369f3b6276ac9d9422383e)

-1626698621.jpg SIMF : [Covering Constraint](#_507049575ebfa9f535e8f25db14a0760)

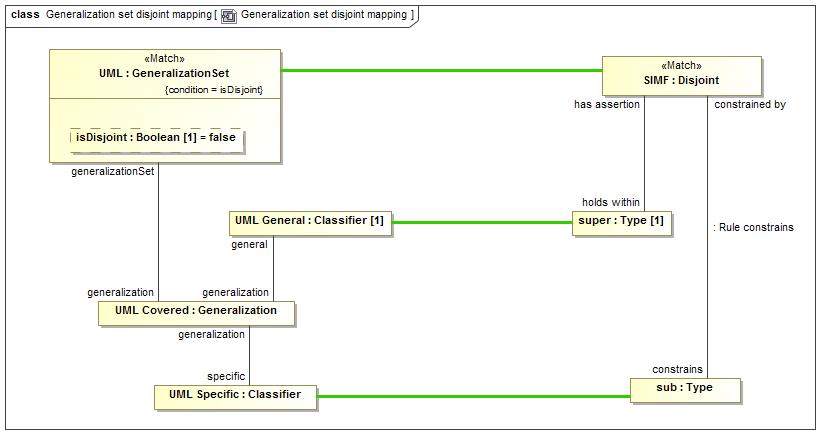
-1626698621.jpg UML General : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8) [1]

-1626698621.jpg UML Specific : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8)

-1626698621.jpg Covered : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

-1626698621.jpg Covers : [Type](#_dfe1514224ca21cedba7b2b29802db50)

## Class Generalization set disjoint mapping



1. Generalization set disjoint mapping

Generalization set disjoint constraint draws a correspondence between a generalization set where isDisjoint is true and a SIMF Disjoint Constraint.  
The generalization set has a set of UML covered <<Generalization>>s, all of which must have the same <general> Classifier, which is mapped to the <holds within> type.  
The <specific> classifiers correspond to a set of <is covered by> types.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg UML : [GeneralizationSet](#_ff83bc3640ce7e2f1d38966bb69d76df)

-1626698621.jpg UML Covered : [Generalization](#_e60de48d98369f3b6276ac9d9422383e)

-1626698621.jpg UML Specific : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8)

-1626698621.jpg SIMF : [Disjoint](#_141639bfc7d23be7f533db476cadd0ff)

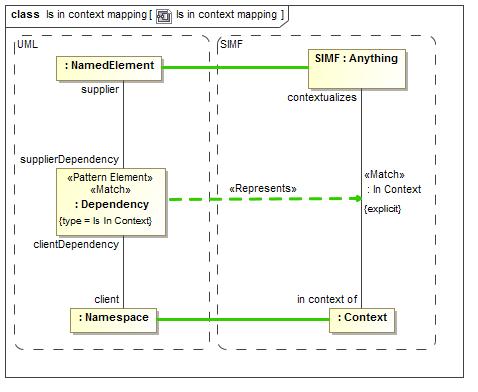
-1626698621.jpg UML General : [Classifier](#_d64cbb1ac021e55aa02cfbb633a5e4a8) [1]

-1626698621.jpg sub : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg super : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]

-1626698621.jpg : [Generalization set disjoint mapping](#_1159e23f5dd1fa271eae56eb2320f7e3)

## Class Is in context mapping



1. Is in context mapping

Is in contest mapping draws a correspondence between a UML dependency with a <<Is In CContext>> stereotype and an explicit "In Context" Relationship. The relationship is Explicit if it has been explicitly asserted, not inferred.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-1626698621.jpg UML : [Element](#_231afe47f3f37d3808096b36c28b4ded)

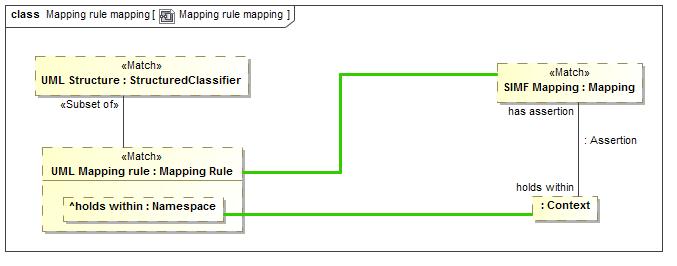
-1626698621.jpg : [Context](#_66d62b068053cee3464e1e03e6035eed)

-1626698621.jpg : [Dependency](#_90a95d6639a7bbbeff7f36a7ec8f3b10)

-1626698621.jpg : [NamedElement](#_705b0620812f5826ea76b12f804e54bb)

-1626698621.jpg : [Namespace](#_b3ba0fe968ce39dcfc6fe8cc0f1b02da)

## Class Mapping rule mapping



1. Mapping rule mapping

The Mapping rule mapping draws a correspondence between any UML StructuredClassifier marked as a <<Mapping Rule>> and a SIMF Mapping. The <holds within> tag then maps to the context the mapping holds within.  
Any structured classifier may be created from SIMF.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

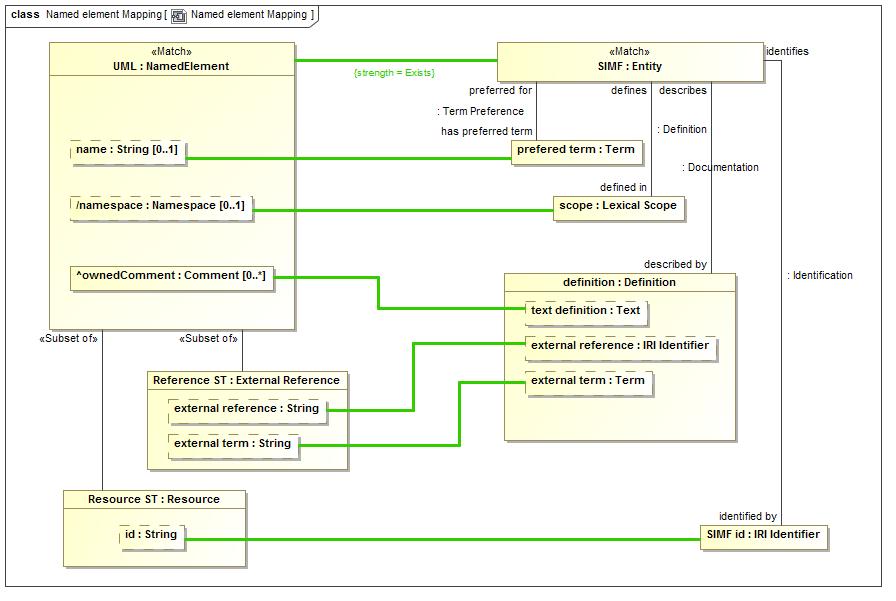
-1626698621.jpg SIMF Mapping : [Mapping](#_551417ad3c6e740d8b880bee8085a718)

-1626698621.jpg UML Mapping rule : [Mapping Rule](#_88466830d981762a31ea6c9ac097c68a)

-1626698621.jpg : [Context](#_66d62b068053cee3464e1e03e6035eed)

-1626698621.jpg UML Structure : [StructuredClassifier](#_28dc1c7ea2fc239d1b7b6e4a9985a371)

## Class Named element Mapping



1. Named element Mapping

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg : [Anything](#_a52cb0ff6e414b3170b58afe10b6afcb)

-1626698621.jpg prefered term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8)

-1626698621.jpg SIMF : [Entity](#_eb8398b5a178c638b98597120ec51c4d)

-1626698621.jpg UML : [NamedElement](#_705b0620812f5826ea76b12f804e54bb)

-1626698621.jpg scope : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

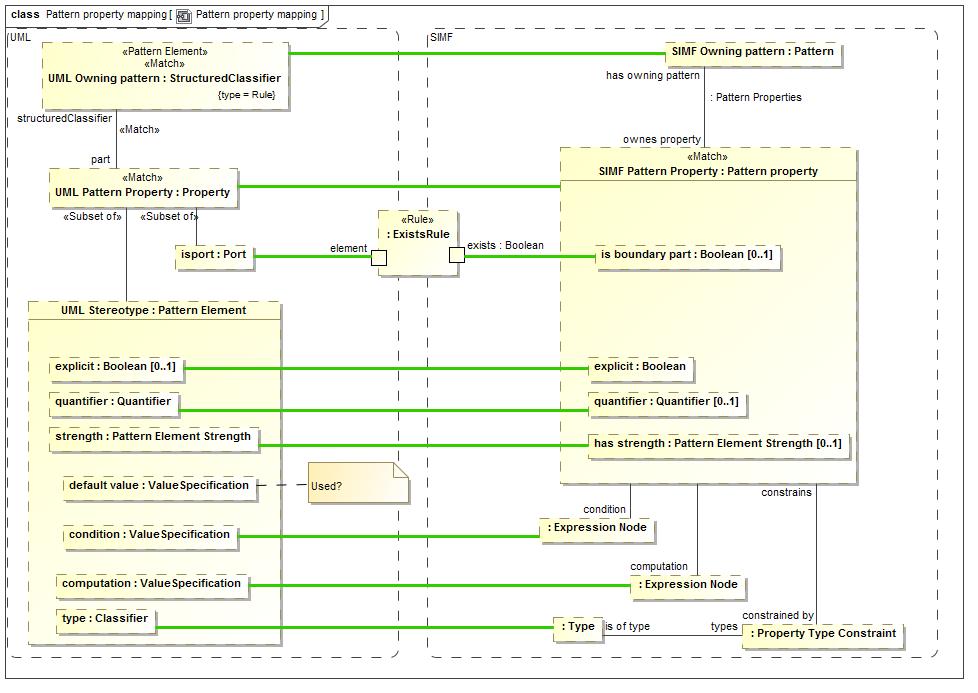
-1626698621.jpg definition : [Definition](#_1a6d88e097d757268d09f68af82fbd34)

-1626698621.jpg Resource ST : [Resource](#_7eb9b01488b85cfcd94ce909e4efc52d)

-1626698621.jpg SIMF id : [IRI Identifier](#_f904ff1da5bfc3387d892b7e0fe9ecb1)

-1626698621.jpg Reference ST : [External Reference](#_57d5acead118af36f52c313850902bc4)

## Class Pattern property mapping



1. Pattern property mapping

Pattern property mapping makes a correspondence between any UML property that is <part> of a <Rule> and a SIMF Pattern Property.  
The owning StructuredClassifier corresponds with the pattern that owns the Pattern Property.  
If the UML Property is stereotyped as a <<Pattern Element>> the tags correspond with the explicit, quantifier, condition, computation and has strength properties of the SIMF pattern property.  
The <type> of the UML Pattern Element is constrained to be a required <is of type> of the pattern property.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF Pattern Property : [Pattern property](#_4d83e476040c7444758dda440d3096fc)

-1626698621.jpg UML Pattern Property : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg UML Owning pattern : [StructuredClassifier](#_28dc1c7ea2fc239d1b7b6e4a9985a371)

-1626698621.jpg UML Stereotype : [Pattern Element](#_2ab083e7636c6f2e537ce538d10aef76)

-1626698621.jpg SIMF Owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f)

-1626698621.jpg : [Type](#_dfe1514224ca21cedba7b2b29802db50)

-1626698621.jpg : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a)

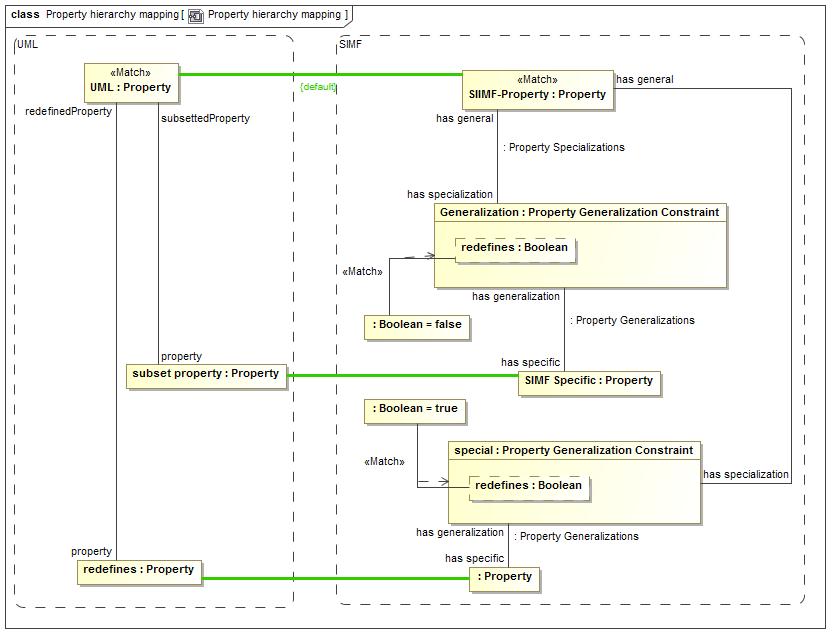
-1626698621.jpg isport : [Port](#_60aaf44d4b562252c04db7f98497e9aa)

-1626698621.jpg : [ExistsRule](#_4dcf4b5884925d444819a469fd70ce00)

-1626698621.jpg : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

-1626698621.jpg : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

## Class Property hierarchy mapping



1. Property hierarchy mapping

Property hierarchy mapping, for properties that have been mapped in other ways, maps UML subsetted properties to a Property Generalization Constraint with redefines=false.  
It also maps UML redefines properties to a Property Generalization Constraint with redefines=true.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg Generalization : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab)

-1626698621.jpg UML : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg redefines : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg subset property : [Property](#_5ad234cb2cde4266195252a23ca7d84e)

-1626698621.jpg SIIMF-Property : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

-1626698621.jpg SIMF Specific : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

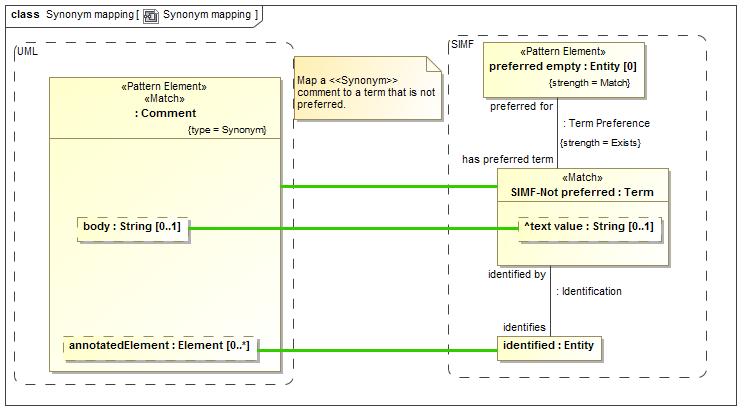
-1626698621.jpg : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = false

-1626698621.jpg special : [Property Generalization Constraint](#_8453c07e3c7aaeab8e715f3569b782ab)

-1626698621.jpg : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f) = true

-1626698621.jpg : [Property](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

## Class Synonym mapping



1. Synonym mapping

Synonym mapping maps a UML Comment with a <<Synonym>> stereotype to a SIMF Term that identifies the element that is annotated by the comment in UML.  
A Synonym term is not a preferred term.

**package** SIMFProfileToModelMapping::Mapping rules

### Attributes

-1626698621.jpg SIMF-Not preferred : [Term](#_1945edd0888993a52c5dc6467a7b3ef8)

-1626698621.jpg identified : [Entity](#_eb8398b5a178c638b98597120ec51c4d)

-1626698621.jpg preferred empty : [Entity](#_eb8398b5a178c638b98597120ec51c4d) [0]

-1626698621.jpg : [Comment](#_0be8406951cdfda82f00f79328cf4efc)

# Concept Index

All, 67

annoation binding, 163

annotated SIMF, 164

annotated binding, 164

Annotated UML, 163

annotation, 163

Annotation comment, 163

annotation property, 163

Annotation Property, 71

Annotation Relationship Type, 72

Annotation ST, 163

Annotation value mapping, 163

Any SIMF, 167

Any UML, 167

Anything, 87

as facet, 80

Assert, 66

asserted by, 59

asserted type, 59

Assertion, 88

Association mapping, 164

at once, 75

Base Unit Type, 96

binding, 83

Binding, 82

binds, 83

Boolean, 97

bound by, 83

bound in, 83

Bound in situation, 83

Bound individual, 83

Bound property, 83

bound to, 83

calls, 42, 43

categorizes, 93, 94

Class mapping, 165

class property, 166

Class property mapping, 166

Classifies, 126

Classifies Stereotype, 172

coerce, 58

computation, 59, 65

Computed Facade, 57

Computed Fact, 40

concept rule, 61

Conceptual Model, 126

Conceptual Package, 51

condition, 59, 61, 65, 73

Conditional Constraint, 73

Constant Reference, 40

Constant Value, 41

constrained by, 79, 89

constrains, 76, 78, 79

Containment mapping, 167

Context, 88

contextualizes, 41, 88, 89

Covered, 173

Covering Constraint, 74

Covers, 173

Default, 66

defaut, 65

defined in, 52, 88

defined within system, 99

defines, 52, 53

definition, 177

Definition, 51, 52

defult, 58

described by, 52, 88

describes, 52

Disjoint, 74

Disjoint with, 127

Documentation, 52

domain, 166

end, 164

Entity, 88

Enumerated, 74

Enumerates, 127

Enumeration mapping, 168

Eqiv properties, 169, 170

Equality, 41

Equivalenent things, 171

Equivalent, 74

Equivalent property chain mapping, 169

Equivalent property mapping, 170

Equivalent with, 128

Equivalent with mapping, 171

evaluates, 41

evaluates in, 41, 42

Evaluation, 41

evaluted by, 42

Exactly One, 66

Exists, 66

ExistsRule, 147

explicit, 65

Expression Context, 41

Expression Node, 42

expression text, 42

expression text language, 42

extends scope, 53, 55

external reference, 52

external term, 52

Facade, 58, 148

Facet, 92

Filter, 149

first, 171

from, 58

from rue, 58

Function Call, 42

Function Implementation, 42

Function Type, 43

Generalization, 179

Generalization mapping, 171

Generalization set covering mapping, 173

Generalization set disjoint mapping, 174

Generalizations, 74

has assertion, 88

has binding, 83, 85

has covering, 94

has equal, 41

has general, 75, 77, 80

has generalization, 77, 79, 84, 93

has identifier, 47, 53

has map rule, 60, 63

has multiplicity, 76, 80

has owning pattern, 64, 65, 66

has preferred term, 89

has prefix, 54

has property, 83, 85

has specialization, 75, 77, 84, 93

has specific, 77, 79, 80

Has Stereotype, 169, 170

has strength, 65

has supertype, 94

has type, 88, 94

has unique, 80, 81

has uniqueness constraint, 80

has value, 40, 41

Has value, 134

holds within, 88, 89

Identification, 46

identified, 180

identified by, 46, 89

Identifier, 46

identifies, 46, 47

implemented by, 43

implements, 42, 43

In Context, 89

in context of, 88, 89

Include, 52

Intersection, 128

Intersection Type, 93

inverse, 44

inverse property, 163

IRI Identifier, 47

is also a class, 164

is boundary part, 64

is composite part, 64

is covered by, 74

Is in context, 129

Is in context mapping, 175

is of type, 78, 84

is primary identity, 80

is sufficent, 75

is used by, 43

isport, 178

Lexical Reference, 52

Lexical Scope, 53

Logical Package, 53

Make enum, 168

Map, 149

map all, 61

map from, 61

Map Rule, 58

Map Rule End, 58

map rule of, 58, 63

Map Rule Type Assertion, 59

Map rules, 63

map to, 58

Mapped Property, 59

Mapped Relationship, 59

Mapping, 60

Mapping Package, 53

Mapping rule mapping, 176

maps property, 59, 60

maps relationship, 59, 61

Match, 66, 151

maximum number, 75

mininum number, 75

Model, 54

Most, 66

Multiplicity Constraint, 75

multiplicity of, 75, 76

Multiplicity Perspective, 75

Multiplicity Target, 76

Named element Mapping, 177

Namespace, 47

None, 66

Number, 97

Object Management Group, Inc. (OMG), xii

Object Operation Type, 43

offset, 99

OMG specifications, xii

OO Target, 43

owner, 167

ownes property, 64

ownes relationship, 64, 66

Package, 54

Pattern, 63

Pattern Constraint, 76

Pattern Element Strength, 66

Pattern Properties, 64

Pattern property, 64

Pattern property mapping, 178

Pattern Property Subset, 65

Pattern Relationship, 65

Pattern Relationships, 66

Phase, 93, 129

Physical Package, 54

prefered term, 177

preferred empty, 180

preferred for, 90

Prefix, 54

prefix of, 54, 55

prerequisite type, 78

Primitive Value, 97

Properties, 83

Property, 84

Property Chains, 145

Property Constraint, 76

property end, 59

Property End, 60

Property Generalization Constraint, 76

Property Generalizations, 77

Property hierarchy mapping, 179

property of, 84

Property Specializations, 77

Property ST, 163

Property Transitivity Constraint, 77

Property Type, 78

Property Type Constraint, 78

Proposition, 89

quantifier, 64

Quantifier, 66

Quantity kind, 97

Quantity Kind, 129

Quantity Value, 98

range class, 166

range property, 166

range type, 167

ratio, 99

recieved by, 44

reciever, 43, 44

Record, 68

Record Type, 68

redefines, 77, 179

reference, 165

Reference ST, 177

referenced by, 41, 55

Referenced scope, 53, 55

references, 53, 55

Relationship, 72

relationship end, 60

Relationship Type, 72

Representation, 60

Representation Rule, 60, 152

represented by, 60, 61

represented concept, 61

Represented Concept, 61

Represents, 130, 153

represents rule, 60

Resource ST, 177

respect of, 76

resulting type, 42, 44

Return type, 44

returned by, 44

Role, 93, 131

Rule, 78, 151

Rule Computation, 152

Rule constrains, 79

Rule Specification, 150

scope, 167, 177

Scope, 55

Scope Reference, 55

second, 171

SIIMF-Property, 179

SIMF, 164, 165, 171, 172, 173, 174, 175, 177

SIMF subtype, 172

SIMF Annotation, 164

SIMF Chain, 169

SIMF Class, 165, 167

SIMF end, 164

SIMF id, 177

SIMF Mapping, 176

SIMF Owner, 167

SIMF Owning pattern, 178

SIMF Pattern Property, 178

SIMF Relation, 166

SIMF relationship, 163

SIMF Specific, 179

SIMF Supertype, 172

SIMF Type, 166

SIMF Value, 168

SIMF Value type, 168

SIMF-Not preferred, 180

SIMF-Primary, 169, 170

SIMF-Secondary, 169, 170

Situation, 84

Situation Type, 85

Source Reference, 131

special, 180

Specializations, 79

stated by, 55, 88

Statement, 55

states, 53, 55

Stereotype, 165, 167

Structure, 85

Structured Type, 85

sub, 174

subset, 171

Subset of, 154

subset property, 179

subsumed by, 79

subsumes, 78

Sufficient, 131

super, 174

symbol, 99

Synonym mapping, 180

System of Units, 98

Term, 47

Term Preference, 89

Text, 98

text definition, 52

Text Identifier, 47

text value, 97

There Exists, 66

to, 58, 61

to rule, 61

Traversal, 44

traverse through, 44

traverse to relation, 44

traversed by, 45

traverses through, 44, 45

Type, 93

Type Constraint, 79

Type End, 61

Type Generalization Constraint, 79

Type Instance Relation, 94

Type or Property, 80

types, 78, 93

typographical conventions, xiii

UML, 164, 165, 172, 173, 174, 175, 177, 179

UML Class, 166

UML Covered, 173, 174

UML Dep, 171

UML Enum, 168

UML General, 173, 174

UML Literal, 168

UML Mapping rule, 176

UML Owner, 167

UML Owning pattern, 178

UML Pattern Property, 178

UML Property, 169, 170

UML Specific, 173, 174

UML Stereotype, 178

UML Structure, 176

UML subtype, 172

UML Superclass, 172

UML-From, 171

UML-to, 171

Union, 132

Union Type, 94

Unique Set, 80

unique within, 47

Uniqueness Constraint, 80

Unit, 132

unit of system, 98

unit reference, 99

Unit Type, 98

used as class, 165

Value, 99, 133

Value Type, 100

with respect to, 75

1. ECSS is an initiative established to develop a coherent, single set of user-friendly standards for use in all European space activities [ECSS]. [↑](#footnote-ref-1)
2. Not to be confused with how “domain” is used in the intelligence community, which refers to levels of security [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/Mars\_Climate\_Orbiter [↑](#footnote-ref-3)