Table of Contents

[1 SMIF Conceptual Model 2](#_Toc465791179)

[1.1 Diagram: SMIF Packages 2](#_Toc465791180)

[2 SMIF Conceptual Model::Expressions 4](#_Toc465791181)

[2.1 Diagram: Expressions 4](#_Toc465791182)

[2.2 Class Constant Reference 4](#_Toc465791183)

[2.2.1 Direct Supertypes 5](#_Toc465791184)

[2.3 Association Constant Value 5](#_Toc465791185)

[2.3.1 Association Ends 5](#_Toc465791186)

[2.4 Class Equality 5](#_Toc465791187)

[2.4.1 Direct Supertypes 5](#_Toc465791188)

[2.5 Association Equality Constraint 5](#_Toc465791189)

[2.5.1 Association Ends 5](#_Toc465791190)

[2.6 Class Evaluation 5](#_Toc465791191)

[2.6.1 Direct Supertypes 5](#_Toc465791192)

[2.7 Association Expression Context 6](#_Toc465791193)

[2.7.1 Direct Supertypes 6](#_Toc465791194)

[2.7.2 Association Ends 6](#_Toc465791195)

[2.8 Class Expression Context 6](#_Toc465791196)

[2.8.1 Direct Supertypes 6](#_Toc465791197)

[2.9 Association Expression Evaluation 6](#_Toc465791198)

[2.9.1 Association Ends 6](#_Toc465791199)

[2.10 Class Expression Node 6](#_Toc465791200)

[2.10.1 Direct Supertypes 6](#_Toc465791201)

[2.10.2 Attributes 7](#_Toc465791202)

[2.11 Class Function Call 7](#_Toc465791203)

[2.11.1 Direct Supertypes 7](#_Toc465791204)

[2.12 Association Function Called 7](#_Toc465791205)

[2.12.1 Direct Supertypes 7](#_Toc465791206)

[2.12.2 Association Ends 7](#_Toc465791207)

[2.13 Association Function Implementation 7](#_Toc465791208)

[2.13.1 Direct Supertypes 7](#_Toc465791209)

[2.13.2 Association Ends 7](#_Toc465791210)

[2.14 Class Function Type 8](#_Toc465791211)

[2.14.1 Direct Supertypes 8](#_Toc465791212)

[2.15 Class Object Operation Type 8](#_Toc465791213)

[2.15.1 Direct Supertypes 8](#_Toc465791214)

[2.16 Association OO Target 8](#_Toc465791215)

[2.16.1 Association Ends 8](#_Toc465791216)

[2.17 Association Result type 8](#_Toc465791217)

[2.17.1 Association Ends 8](#_Toc465791218)

[2.18 Class Traversal 8](#_Toc465791219)

[2.18.1 Direct Supertypes 9](#_Toc465791220)

[2.18.2 Attributes 9](#_Toc465791221)

[2.19 Association Traverse Through 9](#_Toc465791222)

[2.19.1 Association Ends 9](#_Toc465791223)

[3 SMIF Conceptual Model::Facets 10](#_Toc465791224)

[3.1 Diagram: Facets 10](#_Toc465791225)

[3.2 Class Entity to Facet Binding 10](#_Toc465791226)

[3.2.1 Direct Supertypes 11](#_Toc465791227)

[3.3 Class Phase 11](#_Toc465791228)

[3.3.1 Direct Supertypes 11](#_Toc465791229)

[3.4 Class Role 11](#_Toc465791230)

[3.4.1 Direct Supertypes 11](#_Toc465791231)

[4 SMIF Conceptual Model::Identifiers 12](#_Toc465791232)

[4.1 Diagram: Identifiers 12](#_Toc465791233)

[4.2 Association Identification 13](#_Toc465791234)

[4.2.1 Association Ends 13](#_Toc465791235)

[4.3 Class Identifier 13](#_Toc465791236)

[4.3.1 Direct Supertypes 13](#_Toc465791237)

[4.4 Association Identifier in Namespace 13](#_Toc465791238)

[4.4.1 Direct Supertypes 13](#_Toc465791239)

[4.4.2 Association Ends 13](#_Toc465791240)

[4.5 Association Identifier Preference 14](#_Toc465791241)

[4.5.1 Direct Supertypes 14](#_Toc465791242)

[4.5.2 Association Ends 14](#_Toc465791243)

[4.6 Class IRI Identifier 14](#_Toc465791244)

[4.6.1 Direct Supertypes 14](#_Toc465791245)

[4.7 Class Name 14](#_Toc465791246)

[4.7.1 Direct Supertypes 14](#_Toc465791247)

[4.8 Class Namespace 14](#_Toc465791248)

[4.8.1 Direct Supertypes 15](#_Toc465791249)

[4.9 Association Naming Relationship 15](#_Toc465791250)

[4.9.1 Direct Supertypes 15](#_Toc465791251)

[4.9.2 Association Ends 15](#_Toc465791252)

[4.10 Class Technical Identifier 15](#_Toc465791253)

[4.10.1 Direct Supertypes 15](#_Toc465791254)

[4.11 Class Term 15](#_Toc465791255)

[4.11.1 Direct Supertypes 15](#_Toc465791256)

[4.12 Class Text Identifier 16](#_Toc465791257)

[4.12.1 Direct Supertypes 16](#_Toc465791258)

[4.12.2 Attributes 16](#_Toc465791259)

[4.13 Class Unique Identifier 16](#_Toc465791260)

[4.13.1 Direct Supertypes 16](#_Toc465791261)

[4.14 Class Unique Text Identifier 16](#_Toc465791262)

[4.14.1 Direct Supertypes 16](#_Toc465791263)

[5 SMIF Conceptual Model::Kernel 17](#_Toc465791264)

[5.1 Diagram: Kernel Identifiers 17](#_Toc465791265)

[5.2 Diagram: Kernel Lexical Scope 18](#_Toc465791266)

[5.3 Diagram: Kernel Metadata 19](#_Toc465791267)

[5.4 Diagram: Kernel Properties 19](#_Toc465791268)

[5.5 Diagram: Kernel Relationships 20](#_Toc465791269)

[5.6 Diagram: Kernel Rules Summary 21](#_Toc465791270)

[5.7 Diagram: Kernel Top Level 22](#_Toc465791271)

[5.8 Diagram: Kernel Types 23](#_Toc465791272)

[5.9 Diagram: Kernel Values 24](#_Toc465791273)

[6 SMIF Conceptual Model::Lexical Scope 25](#_Toc465791274)

[6.1 Diagram: Lexical Scope 25](#_Toc465791275)

[6.2 Class Conceptual Package 25](#_Toc465791276)

[6.2.1 Direct Supertypes 26](#_Toc465791277)

[6.3 Association Definition 26](#_Toc465791278)

[6.3.1 Direct Supertypes 26](#_Toc465791279)

[6.3.2 Association Ends 26](#_Toc465791280)

[6.4 Class Include 26](#_Toc465791281)

[6.4.1 Direct Supertypes 26](#_Toc465791282)

[6.5 Class Lexical Reference 26](#_Toc465791283)

[6.5.1 Direct Supertypes 26](#_Toc465791284)

[6.6 Class Lexical Scope 26](#_Toc465791285)

[6.6.1 Direct Supertypes 26](#_Toc465791286)

[6.7 Class Logical Package 27](#_Toc465791287)

[6.7.1 Direct Supertypes 27](#_Toc465791288)

[6.8 Class Mapping Package 27](#_Toc465791289)

[6.8.1 Direct Supertypes 27](#_Toc465791290)

[6.9 Class Model 27](#_Toc465791291)

[6.9.1 Direct Supertypes 27](#_Toc465791292)

[6.10 Class Package 27](#_Toc465791293)

[6.10.1 Direct Supertypes 27](#_Toc465791294)

[6.11 Class Physical Package 27](#_Toc465791295)

[6.11.1 Direct Supertypes 27](#_Toc465791296)

[6.12 Association Prefix 27](#_Toc465791297)

[6.12.1 Direct Supertypes 27](#_Toc465791298)

[6.12.2 Association Ends 28](#_Toc465791299)

[6.13 Class Prefix 28](#_Toc465791300)

[6.13.1 Direct Supertypes 28](#_Toc465791301)

[6.14 Association Scope of Reference 28](#_Toc465791302)

[6.14.1 Association Ends 28](#_Toc465791303)

[6.15 Association Scope Reference 28](#_Toc465791304)

[6.15.1 Association Ends 28](#_Toc465791305)

[6.16 Association Statement 28](#_Toc465791306)

[6.16.1 Direct Supertypes 28](#_Toc465791307)

[6.16.2 Association Ends 28](#_Toc465791308)

[7 SMIF Conceptual Model::Mapping 30](#_Toc465791309)

[7.1 Diagram: Facades 30](#_Toc465791310)

[7.2 Diagram: Mapping Rules 31](#_Toc465791311)

[7.3 Class Computed Facade 31](#_Toc465791312)

[7.3.1 Direct Supertypes 31](#_Toc465791313)

[7.3.2 Operations 31](#_Toc465791314)

[7.4 Association Concrete Map End 32](#_Toc465791315)

[7.4.1 Association Ends 32](#_Toc465791316)

[7.5 Association Concrete Pattern Relation 32](#_Toc465791317)

[7.5.1 Association Ends 32](#_Toc465791318)

[7.6 Class Facade 32](#_Toc465791319)

[7.6.1 Direct Supertypes 32](#_Toc465791320)

[7.7 Association Map Rule Type Assertion 32](#_Toc465791321)

[7.7.1 Association Ends 32](#_Toc465791322)

[7.8 Association Mapped Property 32](#_Toc465791323)

[7.8.1 Association Ends 32](#_Toc465791324)

[7.9 Association Mapped Relationship 33](#_Toc465791325)

[7.9.1 Association Ends 33](#_Toc465791326)

[7.10 Class Mapping 33](#_Toc465791327)

[7.10.1 Direct Supertypes 33](#_Toc465791328)

[7.11 Class Mapping Pattern 33](#_Toc465791329)

[7.11.1 Direct Supertypes 33](#_Toc465791330)

[7.12 Class Match Rule 33](#_Toc465791331)

[7.12.1 Direct Supertypes 33](#_Toc465791332)

[7.12.2 Attributes 33](#_Toc465791333)

[7.13 Class Match Rule End 34](#_Toc465791334)

[7.13.1 Attributes 34](#_Toc465791335)

[7.14 Class Property End 34](#_Toc465791336)

[7.14.1 Direct Supertypes 34](#_Toc465791337)

[7.15 Association Reference Map End 34](#_Toc465791338)

[7.15.1 Association Ends 35](#_Toc465791339)

[7.16 Association Reference Pattern Relation 35](#_Toc465791340)

[7.16.1 Association Ends 35](#_Toc465791341)

[7.17 Class Relationship End 35](#_Toc465791342)

[7.17.1 Direct Supertypes 35](#_Toc465791343)

[7.18 Association Representation 35](#_Toc465791344)

[7.18.1 Association Ends 35](#_Toc465791345)

[7.19 Class Representation Rule 35](#_Toc465791346)

[7.19.1 Direct Supertypes 35](#_Toc465791347)

[7.19.2 Attributes 36](#_Toc465791348)

[7.20 Association Represented Concept 36](#_Toc465791349)

[7.20.1 Association Ends 36](#_Toc465791350)

[8 SMIF Conceptual Model::Metadata 37](#_Toc465791351)

[8.1 Diagram: Metadata 37](#_Toc465791352)

[8.2 Association Assertion Statement 38](#_Toc465791353)

[8.2.1 Direct Supertypes 38](#_Toc465791354)

[8.2.2 Association Ends 38](#_Toc465791355)

[8.3 Class Definition 38](#_Toc465791356)

[8.3.1 Direct Supertypes 38](#_Toc465791357)

[8.3.2 Attributes 38](#_Toc465791358)

[8.4 Association Definition Relationship 38](#_Toc465791359)

[8.4.1 Direct Supertypes 39](#_Toc465791360)

[8.4.2 Association Ends 39](#_Toc465791361)

[8.5 Class Information Source 39](#_Toc465791362)

[8.5.1 Direct Supertypes 39](#_Toc465791363)

[8.6 Class Metadata 39](#_Toc465791364)

[8.6.1 Direct Supertypes 39](#_Toc465791365)

[8.7 Association Metadata relationship 39](#_Toc465791366)

[8.7.1 Association Ends 39](#_Toc465791367)

[8.8 Association Record of a thing 39](#_Toc465791368)

[8.8.1 Association Ends 40](#_Toc465791369)

[8.9 Association Source of Information 40](#_Toc465791370)

[8.9.1 Association Ends 40](#_Toc465791371)

[8.10 Class Statement 40](#_Toc465791372)

[8.10.1 Direct Supertypes 40](#_Toc465791373)

[8.10.2 Attributes 40](#_Toc465791374)

[9 SMIF Conceptual Model::Patterns 41](#_Toc465791375)

[9.1 Diagram: Patterns 41](#_Toc465791376)

[9.2 Association Defining Relative Property 42](#_Toc465791377)

[9.2.1 Association Ends 42](#_Toc465791378)

[9.3 Class Focus Property 42](#_Toc465791379)

[9.3.1 Direct Supertypes 42](#_Toc465791380)

[9.4 Association Match Rules 42](#_Toc465791381)

[9.4.1 Direct Supertypes 42](#_Toc465791382)

[9.4.2 Association Ends 42](#_Toc465791383)

[9.5 Class Part Property 42](#_Toc465791384)

[9.5.1 Direct Supertypes 42](#_Toc465791385)

[9.5.2 Attributes 42](#_Toc465791386)

[9.6 Class Pattern 43](#_Toc465791387)

[9.6.1 Direct Supertypes 43](#_Toc465791388)

[9.7 Class Pattern of Type 43](#_Toc465791389)

[9.7.1 Direct Supertypes 43](#_Toc465791390)

[9.8 Association Pattern Properties 43](#_Toc465791391)

[9.8.1 Direct Supertypes 43](#_Toc465791392)

[9.8.2 Association Ends 43](#_Toc465791393)

[9.9 Class Pattern Property 43](#_Toc465791394)

[9.9.1 Direct Supertypes 44](#_Toc465791395)

[9.9.2 Attributes 44](#_Toc465791396)

[9.10 Class Pattern Property Subset 44](#_Toc465791397)

[9.10.1 Attributes 44](#_Toc465791398)

[9.11 Class Pattern Relationship 44](#_Toc465791399)

[9.11.1 Direct Supertypes 44](#_Toc465791400)

[9.11.2 Attributes 44](#_Toc465791401)

[9.12 Association Pattern Relationship 45](#_Toc465791402)

[9.12.1 Direct Supertypes 45](#_Toc465791403)

[9.12.2 Association Ends 45](#_Toc465791404)

[9.13 Class Quantified Property 45](#_Toc465791405)

[9.13.1 Direct Supertypes 45](#_Toc465791406)

[9.13.2 Attributes 45](#_Toc465791407)

[9.14 Class Relative Property 45](#_Toc465791408)

[9.14.1 Direct Supertypes 45](#_Toc465791409)

[9.14.2 Attributes 45](#_Toc465791410)

[9.15 Association Subject of Pattern Relationship 46](#_Toc465791411)

[9.15.1 Direct Supertypes 46](#_Toc465791412)

[9.15.2 Association Ends 46](#_Toc465791413)

[9.16 Class Type Pattern Property 46](#_Toc465791414)

[9.16.1 Enumeration Pattern Element Strength 46](#_Toc465791415)

[9.16.2 Enumeration Quantifier 47](#_Toc465791416)

[10 SMIF Conceptual Model::Properties 48](#_Toc465791417)

[10.1 Diagram: Properties 48](#_Toc465791418)

[10.2 Class Annotation Property 48](#_Toc465791419)

[10.2.1 Direct Supertypes 49](#_Toc465791420)

[10.3 Association Bound Individual 49](#_Toc465791421)

[10.3.1 Association Ends 49](#_Toc465791422)

[10.4 Association Bound Property 49](#_Toc465791423)

[10.4.1 Direct Supertypes 49](#_Toc465791424)

[10.4.2 Association Ends 49](#_Toc465791425)

[10.5 Association Bound Subject 49](#_Toc465791426)

[10.5.1 Association Ends 49](#_Toc465791427)

[10.6 Class Characteristic Binding 50](#_Toc465791428)

[10.6.1 Direct Supertypes 50](#_Toc465791429)

[10.7 Class Characteristic Type 50](#_Toc465791430)

[10.7.1 Direct Supertypes 50](#_Toc465791431)

[10.8 Association Properties Relationship 50](#_Toc465791432)

[10.8.1 Association Ends 50](#_Toc465791433)

[10.9 Class Property Binding 51](#_Toc465791434)

[10.9.1 Direct Supertypes 51](#_Toc465791435)

[10.10 Class Property Type 51](#_Toc465791436)

[10.10.1 Direct Supertypes 52](#_Toc465791437)

[11 SMIF Conceptual Model::Relationships 53](#_Toc465791438)

[11.1 Diagram: Relationships 53](#_Toc465791439)

[11.2 Class Relationship 54](#_Toc465791440)

[11.2.1 Direct Supertypes 54](#_Toc465791441)

[11.3 Class Relationship Type 54](#_Toc465791442)

[11.3.1 Direct Supertypes 54](#_Toc465791443)

[12 SMIF Conceptual Model::Rules 55](#_Toc465791444)

[12.1 Diagram: General Rules 55](#_Toc465791445)

[12.2 Diagram: Property Constraints 56](#_Toc465791446)

[12.3 Diagram: Rules in Context 57](#_Toc465791447)

[12.4 Diagram: Rules Summary 58](#_Toc465791448)

[12.5 Diagram: Type Constraints 59](#_Toc465791449)

[12.6 Class Conditional Rule 59](#_Toc465791450)

[12.6.1 Direct Supertypes 59](#_Toc465791451)

[12.6.2 Attributes 60](#_Toc465791452)

[12.7 Class Covering Constraint 60](#_Toc465791453)

[12.7.1 Direct Supertypes 60](#_Toc465791454)

[12.8 Association Covering Constraint 60](#_Toc465791455)

[12.8.1 Association Ends 60](#_Toc465791456)

[12.9 Class Disjoint 60](#_Toc465791457)

[12.9.1 Direct Supertypes 60](#_Toc465791458)

[12.10 Class Enumerated 61](#_Toc465791459)

[12.10.1 Direct Supertypes 61](#_Toc465791460)

[12.11 Class Equivalent 61](#_Toc465791461)

[12.11.1 Direct Supertypes 61](#_Toc465791462)

[12.12 Class Facet Classification Constraint 61](#_Toc465791463)

[12.12.1 Direct Supertypes 61](#_Toc465791464)

[12.13 Association Generalization 61](#_Toc465791465)

[12.13.1 Association Ends 61](#_Toc465791466)

[12.14 Class Generalization Constraint 62](#_Toc465791467)

[12.14.1 Direct Supertypes 62](#_Toc465791468)

[12.14.2 Attributes 62](#_Toc465791469)

[12.15 Class Multiplicity Constraint 62](#_Toc465791470)

[12.15.1 Direct Supertypes 63](#_Toc465791471)

[12.15.2 Attributes 63](#_Toc465791472)

[12.16 Association Multiplicity Reference 63](#_Toc465791473)

[12.16.1 Association Ends 63](#_Toc465791474)

[12.17 Association Multiplicity Target 64](#_Toc465791475)

[12.17.1 Direct Supertypes 64](#_Toc465791476)

[12.17.2 Association Ends 64](#_Toc465791477)

[12.18 Class Property Constraint 64](#_Toc465791478)

[12.18.1 Direct Supertypes 64](#_Toc465791479)

[12.19 Class Property Transitivity Constraint 64](#_Toc465791480)

[12.19.1 Direct Supertypes 64](#_Toc465791481)

[12.20 Association Property Type 64](#_Toc465791482)

[12.20.1 Association Ends 64](#_Toc465791483)

[12.21 Class Property Type Constraint 65](#_Toc465791484)

[12.21.1 Direct Supertypes 65](#_Toc465791485)

[12.21.2 Attributes 65](#_Toc465791486)

[12.22 Class Rule 65](#_Toc465791487)

[12.22.1 Direct Supertypes 65](#_Toc465791488)

[12.23 Association Rule Constrains 65](#_Toc465791489)

[12.23.1 Association Ends 65](#_Toc465791490)

[12.24 Association Rule Subsumption 65](#_Toc465791491)

[12.24.1 Association Ends 66](#_Toc465791492)

[12.25 Association Specialization 66](#_Toc465791493)

[12.25.1 Direct Supertypes 66](#_Toc465791494)

[12.25.2 Association Ends 66](#_Toc465791495)

[12.26 Class Type Constraint 66](#_Toc465791496)

[12.26.1 Direct Supertypes 66](#_Toc465791497)

[12.27 Association Unique Set 66](#_Toc465791498)

[12.27.1 Association Ends 66](#_Toc465791499)

[12.28 Class Uniqueness Constraint 67](#_Toc465791500)

[12.28.1 Direct Supertypes 67](#_Toc465791501)

[12.28.2 Attributes 67](#_Toc465791502)

[13 SMIF Conceptual Model::Situations 68](#_Toc465791503)

[13.1 Diagram: Situations 68](#_Toc465791504)

[13.2 Class Actual Situation 68](#_Toc465791505)

[13.2.1 Direct Supertypes 68](#_Toc465791506)

[13.3 Class Situation 69](#_Toc465791507)

[13.3.1 Direct Supertypes 69](#_Toc465791508)

[13.4 Class Situation Type 69](#_Toc465791509)

[13.4.1 Direct Supertypes 69](#_Toc465791510)

[14 SMIF Conceptual Model::Structures 70](#_Toc465791511)

[14.1 Diagram: Records 70](#_Toc465791512)

[14.2 Diagram: Structures 71](#_Toc465791513)

[14.3 Class Record 71](#_Toc465791514)

[14.3.1 Direct Supertypes 71](#_Toc465791515)

[14.4 Class Record Type 71](#_Toc465791516)

[14.4.1 Direct Supertypes 72](#_Toc465791517)

[14.5 Association Record types for a type 72](#_Toc465791518)

[14.5.1 Association Ends 72](#_Toc465791519)

[14.6 Class Structure 72](#_Toc465791520)

[14.6.1 Direct Supertypes 72](#_Toc465791521)

[14.7 Class Structure Property Binding 72](#_Toc465791522)

[14.7.1 Direct Supertypes 72](#_Toc465791523)

[14.8 Class Structure Type 72](#_Toc465791524)

[14.8.1 Direct Supertypes 72](#_Toc465791525)

[14.9 Class Structured Property Type 72](#_Toc465791526)

[14.9.1 Direct Supertypes 73](#_Toc465791527)

[15 SMIF Conceptual Model::Top level 74](#_Toc465791528)

[15.1 Diagram: Top Level 74](#_Toc465791529)

[15.2 Class Actual Entity 75](#_Toc465791530)

[15.2.1 Direct Supertypes 75](#_Toc465791531)

[15.3 Association Assertion 75](#_Toc465791532)

[15.3.1 Association Ends 75](#_Toc465791533)

[15.4 Class Context 76](#_Toc465791534)

[15.4.1 Direct Supertypes 77](#_Toc465791535)

[15.5 Association Extent of Context 77](#_Toc465791536)

[15.5.1 Association Ends 77](#_Toc465791537)

[15.6 Class Identifiable Entity 77](#_Toc465791538)

[15.6.1 Direct Supertypes 78](#_Toc465791539)

[15.7 Association Negation 79](#_Toc465791540)

[15.7.1 Association Ends 79](#_Toc465791541)

[15.8 Class Proposition 79](#_Toc465791542)

[15.8.1 Direct Supertypes 79](#_Toc465791543)

[15.9 Class Temporal Entity 80](#_Toc465791544)

[15.9.1 Direct Supertypes 80](#_Toc465791545)

[15.10 Class Thing 80](#_Toc465791546)

[16 SMIF Conceptual Model::Types 82](#_Toc465791547)

[16.1 Diagram: Type-instance 82](#_Toc465791548)

[16.2 Diagram: Types 83](#_Toc465791549)

[16.3 Class Entity Type 83](#_Toc465791550)

[16.3.1 Direct Supertypes 84](#_Toc465791551)

[16.4 Association Extent of Type 84](#_Toc465791552)

[16.4.1 Direct Supertypes 84](#_Toc465791553)

[16.4.2 Association Ends 84](#_Toc465791554)

[16.5 Class Facet 84](#_Toc465791555)

[16.5.1 Direct Supertypes 84](#_Toc465791556)

[16.6 Class Intersection Type 85](#_Toc465791557)

[16.6.1 Direct Supertypes 85](#_Toc465791558)

[16.7 Class Type 85](#_Toc465791559)

[16.7.1 Direct Supertypes 85](#_Toc465791560)

[16.7.2 Associations 85](#_Toc465791561)

[16.8 Class Union Type 85](#_Toc465791562)

[16.8.1 Direct Supertypes 86](#_Toc465791563)

[17 SMIF Conceptual Model::Values 87](#_Toc465791564)

[17.1 Diagram: Values 88](#_Toc465791565)

[17.2 Class Base Unit Type 89](#_Toc465791566)

[17.2.1 Direct Supertypes 89](#_Toc465791567)

[17.3 Class Quantity kind 89](#_Toc465791568)

[17.3.1 Direct Supertypes 89](#_Toc465791569)

[17.4 Class Quantity Value 89](#_Toc465791570)

[17.4.1 Direct Supertypes 90](#_Toc465791571)

[17.4.2 Attributes 90](#_Toc465791572)

[17.5 Association Referenced System of Units 90](#_Toc465791573)

[17.5.1 Direct Supertypes 90](#_Toc465791574)

[17.5.2 Association Ends 90](#_Toc465791575)

[17.6 Class Structured Value 90](#_Toc465791576)

[17.6.1 Direct Supertypes 90](#_Toc465791577)

[17.7 Class Structured Value Type 90](#_Toc465791578)

[17.7.1 Direct Supertypes 90](#_Toc465791579)

[17.8 Class System of Units 91](#_Toc465791580)

[17.8.1 Direct Supertypes 91](#_Toc465791581)

[17.9 Class Unit Type 91](#_Toc465791582)

[17.9.1 Direct Supertypes 91](#_Toc465791583)

[17.9.2 Attributes 91](#_Toc465791584)

[17.10 Class Value 92](#_Toc465791585)

[17.10.1 Direct Supertypes 92](#_Toc465791586)

[17.11 Class Value Type 92](#_Toc465791587)

[17.11.1 Direct Supertypes 92](#_Toc465791588)

[Appendix A: Primitive Type 93](#_Toc465791589)

[17.12 Boolean 93](#_Toc465791590)

[17.13 Integer 93](#_Toc465791591)

[17.14 Number 93](#_Toc465791592)

[17.15 Percentage 93](#_Toc465791593)

[17.16 Real Number 93](#_Toc465791594)

[17.17 Text 93](#_Toc465791595)

[Appendix B: Data Type 94](#_Toc465791596)

[17.18 Data Value 94](#_Toc465791597)

[17.19 Measurement Value 94](#_Toc465791598)

[17.20 Numeric 94](#_Toc465791599)

[17.21 Primitive Value 94](#_Toc465791600)

[Appendix C: Tree 95](#_Toc465791601)

[17.22 Class Hierarchy 95](#_Toc465791602)

[17.23 Enumeration Hierarchy 96](#_Toc465791603)

# SMIF Conceptual Model

## Diagram: SMIF Packages



1. SMIF Packages

# SMIF Conceptual Model::Expressions

Expressions define computations across SMIF models.

## Diagram: Expressions



1. Expressions

Expressions define computations

## Class Constant Reference

A calculation that returns a thing identified by <has value>.  
  
[FIBO] Constant  
[FUML] LiteralSpecification where subtype of literal is determined by the type of <has value>.   
-LiteralInteger->type is Integer or a subtype  
-LiteralReal-> type is not integer or a subtype  
-LiteralBoolean->type is Boolean  
-LiteralString->type is Text

### Direct Supertypes

[Expression Node](#_f9bba899ada544a47c36bb071e9024f5)

## Association Constant Value

Relationship defining a link to a constant value within an expression.

### Association Ends

has value : [Thing](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



A constant value referenced in an expression.

referenced by : [Constant Reference](#_f3f61859903284f1b00fc6feee0b33f8) [\*]



Referencing constant expression node.

## 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)

## Association Equality Constraint

Relationship defining set of things that will be evaluated for equality.

### Association Ends

has equal : [Thing](#_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.

has equality : [Equality](#_99ee84fc373e5bb5ae6febaa538452e1) [\*]



Equality constraints for a thing.

## 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)

## Association Expression Context

Context in which an expression will be evaluated.

### Direct Supertypes

[Extent of Context](#_52c887644007b8e51a1f6e976113707a)

### Association Ends

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



Context of evaluation and namespace resolution for an expression.

contextualizes : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]



Expressions referencing a context.

## 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

[Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)

## Association Expression Evaluation

Relationship defining the expression that will be evaluated by an evaluation.

### Association Ends

evaluates : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [1]



The expression node "head" an evaluation evaluates.

evaluated by : [Evaluation](#_764178c56513beb91e5b5964ec31da8e) [\*]



Evaluations of an expression node.

## Class Expression Node

An abstract class representing 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.  
  
[FIBO] Expression  
[UML] Expression

### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc)

### Attributes

expression text : [Text](#_f8b3b1f5ed63755061811cd6b69bc24f) [0..1]



Textual expression of the expression which is further refined by subtypes of expression.  
[UML] StringExpression

expression text language : [Text](#_f8b3b1f5ed63755061811cd6b69bc24f) [0..1]



expression language used for the expression text

## 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](#_e60871f18b94666411d0d4023a66bd0b)

## Association Function Called

Relationship defining the function (a type) called by a function call.

### Direct Supertypes

[Extent of Type](#_7930d7b301f56f0155603422a27ad833)

### Association Ends

calls : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [1]



Function called

is used by : [Function Call](#_db3e44e523a232e5b77a133d74842e81) [\*]



Function calls using a function declaration.

## Association Function Implementation

Relationship defining the implementation of a function by an expression.

### Direct Supertypes

[Expression Context](#_6c832196fd78ec5da9e6b1ddd1779adf)

### Association Ends

implemented by : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [0..1]



Expression which defines the implementation of a function.

implements : [Function Type](#_cff99d2f22ee84a9e95ea582786a897b) [0..1]



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.  
  
[FUML] Operation where ownedParameter corresponds with <has property> and type corresponds with <resulting type>.

### Direct Supertypes

[Expression Context](#_d847ee03faa23264a18dd452d21972fc), [Structure Type](#_3b0c6b335aca4015ef569068da1bec31)

## Class Object Operation Type

An operation bound to a specific "receiver" in the "Object Oriented" sense.  
[FUML] Operation

### Direct Supertypes

[Function Type](#_cff99d2f22ee84a9e95ea582786a897b)

## Association OO Target

Relationship defining the "target" type of an object oriented function.

### Association Ends

receiver : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property that is the receiver of an object operation.   
[UML] class (of Operation)

received by : [Object Operation Type](#_e6c2e5d52e1652a6c3d27d411345c754) [\*]



The Object Operation for which a receiver is defined.

## Association Result type

Relationship defining the type or types returned by an expression evaluation.

### Association Ends

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



Type of the result of a function  
[UML] type (of an operation or expression).

returned by : [Expression Context](#_d847ee03faa23264a18dd452d21972fc) [\*]



Method returning a type.

## 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](#_e60871f18b94666411d0d4023a66bd0b)

### Attributes

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.

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.

## Association Traverse Through

Relationship defining the property of the current context which will be traversed.

### Association Ends

traverses through : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]



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

traversed by : [Traversal](#_0492440b12b90a76377a15324efa2182) [\*]



Traversals through a property.

# SMIF Conceptual Model::Facets

The facet package defines facets, roles and phases. Types that "mix in" to other types in a specific context or timeframe.

## Diagram: Facets



1. Facets

## Class Entity to Facet Binding

Facet of Entity binds a particular entity to a facet. In the case of a role, it states that an entity plays the role. In the case of a phase, it states that an entity has that phase and that it is a phase of that entity.  
Facet of Entity is a kind of contextual categorization in that the entity assumes all of the characteristics of the facet where the Facet of Entity is asserted. E.g. if Joe has a policeman role, Joe is a policeman.  
  
Facet of entity may only relate entities that have a type compatible with the type of the facet, as defined by a Facet Classification Rule.  
  
[FIBO] (for roles of actors) AgentInRole.  
[FIBO] (for roles of anything else) ThingInRole

### Direct Supertypes

[Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166)

## Class Phase

A phase (or state) is a static 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.  
A Phase is a situation in that there is a situation coincident with each phase.  
  
[Guizzardi] (Phased-Sortal): Let PS be a universal and let S be a  
substance sortal specialized (restricted by) PS. Now, let extw(~PS) = extw(S) \ extw(PS)  
be the complement of the extension of PS in world w. In this formula, the  
symbol \ represents the set theoretical operation of set difference. The  
universal PS is a phased-sortal iff for all worlds w ∈ W, there is a w ∈ W such  
that extw(PS) ∩ extw(~PS) ≠ ∅

### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013), [Situation Type](#_50241f5936e61055293ca95f860768d8)

## Class Role

A role is a facet type that defines a specific purpose or behavior of a class of things. E.g. teacher, policeman, or employer.  
[FIBO] Role. Note that partyInRole or thingInRole are implied by classification of a thing.

### Direct Supertypes

[Facet](#_3b2e69eb6121d1e3a1180bbe8ee64013)

# SMIF Conceptual Model::Identifiers

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

## Diagram: Identifiers



1. Identifiers

An identifier that can be represented as text. The text is in the "value" property.  
  
[IDEAS] Sign: An Individual that signifies a Thing.

## Association Identification

Relationship defining an identifier for an entity.  
  
[IDEAS] namedBy: A couple that asserts that a Name describes a Thing.  
  
[ISO 1087] Designation

### Association Ends

identifies : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [1]



The entity an identifier identifies.  
  
[FIBO] identifies: is the relationship between something and that which provides a unique reference for it  
  
[ISO 1087] designator: representation of a concept (3.2.1) by a sign which  
denotes it

identified by : [Identifier](#_095e3f15be2ed98da1f28f354699da01) [\*]



An identifier for an <Entity>.  
[FIBO] hasDenotation

## Class Identifier

An identifier is any value that is used to distinguish an entity from other entities. Note that any identifier may be contextualized by one or more context, including language context. Identifiers are a “sign” for an identity where identity is an abstraction of individuality that is the basis for identifiers.  
  
[IDEAS] Name: A Representation that identifies a Thing.  
[FIBO] Identifier  
[CL] Term: expression which denotes an individual, consisting of either a name or, recursively, a function term applied to a sequence of arguments, which are themselves terms

### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

## Association Identifier in Namespace

Relationship defining the namespace within which a unique identifier is defined and unique.  
  
[ISO 1087] monosemy: relation between designations (3.4.1) and concepts (3.2.1) in a given language in which one designation only relates to one concept

### Direct Supertypes

[Definition](#_3f8ee3c0c2369667c3f31d50e0ff6f83)

### Association Ends

unique within : [Namespace](#_9c5aa7f24b9d67e77921e06d105205c0) [1]



The namespace in which an identifier is defined and has a unique value.  
[FUML] memberNamespace

scopes identifier : [Unique Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26) [\*]



An Identifier defined within the scope of a namespace.  
[FUML] member

## Association Identifier Preference

Relationship defining the preferred identifier for an entity.  
  
[ISO 1087] preferred term: term (3.4.3) rated according to the scale of the term acceptability rating (3.4.14) as the primary term for a given concept (3.2.1)

### Direct Supertypes

[Identification](#_5a0c9611d1c64dcbc0f89b5299e112ed)

### Association Ends

has preferred : [Identifier](#_095e3f15be2ed98da1f28f354699da01) [0..1]



Default identifier to use for an entity.  
Where multiple identifiers are preferred in differing context any method for selecting the most preferred identifier is implementation specific and not specified by this standard.  
[FUML] NamedElement.name: Note: An Identifier that is <preferred for> an entity is equivalent to the name of a named element.

preferred for : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [0..1]



The entity an identifier is preferred for.

## Class IRI Identifier

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

### Direct Supertypes

[Technical Identifier](#_64e77adb9e39a94e091d321b1b778074)

## Class Name

A word or set of words by which a person, animal, place, or thing is known, addressed, or referred to. Names are not necessarily unique.  
  
[IDEAS] Name: A Representation that identifies a Thing.  
  
[CL] Name: Names and sequence markers are disjoint syntax categories, and each is disjoint from all other syntax categories.

### Direct Supertypes

[Text Identifier](#_380248073543af7bed8363f2b34ad5f7)

## Class Namespace

A namespace is a context that provides a way to make identifiers unique and identify exactly one entity. For example, the Virginia driver's license division provides unique driver's license numbers.  
  
Similar to [IDEAS] UniqueNamingScheme: A NamingScheme where different Names will not contain tokens of the same Representation Type.   
Note: SMIF identifiers are not instances of their namespace.  
  
[FIBO] IdentificationScheme: system for allocating identifiers to objects  
  
[ISO 1087] terminology 1: set of designations (3.4.1) belonging to one special language (3.1.3)  
  
[FUML] Namespace  
  
[CL] Vocabulary

### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

## Association Naming Relationship

Relationship defining a human meaningfully name for an entity.

### Direct Supertypes

[Identification](#_5a0c9611d1c64dcbc0f89b5299e112ed)

### Association Ends

names : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [1..\*]



An entity named by a name.

has name : [Name](#_afe5a48976a2df078be9473827611fb8) [\*]



A human meaningful name for an entity.  
[FIBO] hasName: that by which some thing is known; may apply to anything

## Class Technical Identifier

A technical identifier is defined within a technical system, information structure or system of systems for references and identity within that system or information element. Such identifiers may have no meaning outside of that system.  
  
Typical technical identifiers include inter document "refs", record numbers, etc. The system should be referenced as the namespace.

### Direct Supertypes

[Unique Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

## Class Term

A word, phrase or name used by stakeholders to uniquely identify entities.   
  
[ISO 1087] term: verbal designation of a general concept in a specific subject field.

### Direct Supertypes

[Name](#_afe5a48976a2df078be9473827611fb8), [Unique Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

## Class Text Identifier

A code or other simple value that can be represented as text, identifying something as defined in some namespace. Simple identifiers may be codes, names, numbers or compound values.  
[NIEM] IdentificationType (IdentificationID=value)

### Direct Supertypes

[Identifier](#_095e3f15be2ed98da1f28f354699da01)

### Attributes

value : [String](#_e8a6ce315d976318da3ab784a645ea44)



Text value of an identifier

## Class Unique Identifier

A unique identifier is an entity used to uniquely identify something. The identified thing is referenced by what the identifier <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

[Identifier](#_095e3f15be2ed98da1f28f354699da01)

## Class Unique Text Identifier

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

### Direct Supertypes

[Text Identifier](#_380248073543af7bed8363f2b34ad5f7), [Unique Identifier](#_18f8ef1b23e6cdf9278bd94f24f73c26)

# SMIF Conceptual Model::Kernel

The kernel subsets the SMIF classes. The diagrams in this package illustrate the concrete classes that are used to define the SMIF language.  
Note that shaded classes are not instantiated in the kernel and may be "flattened". Specifications for each class and association are defined in the corresponding package for that concept.

## Diagram: Kernel Identifiers



1. Kernel Identifiers

An identifier that can be represented as text. The text is in the "value" property.  
  
[IDEAS] Sign: An Individual that signifies a Thing.

## Diagram: Kernel Lexical Scope



1. Kernel Lexical Scope

## Diagram: Kernel Metadata



1. Kernel Metadata

## Diagram: Kernel Properties



1. Kernel Properties

## Diagram: Kernel Relationships



1. Kernel Relationships

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

## Diagram: Kernel Rules Summary



1. Kernel Rules Summary

This diagram shown a summary of the primary rules.

## Diagram: Kernel Top Level



1. Kernel Top Level

Diagram showing summary of top level classes and significant subtypes.

## Diagram: Kernel Types



1. Kernel Types

## Diagram: Kernel Values



1. Kernel Values

# SMIF Conceptual Model::Lexical Scope

Lexical scope defines the structure of models and the ownership of model elements.

## Diagram: Lexical Scope



1. Lexical Scope

## Class Conceptual Package

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

### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

## Association Definition

Relationship defining the set of elements defined within a lexical scope.

### Direct Supertypes

[Extent of Context](#_52c887644007b8e51a1f6e976113707a)

### Association Ends

defines : [Thing](#_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.  
[FUML] ownedElement, ownedMember

defined in : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [1]



Lexical scope defining model elements.  
[UML]owner

## Class Include

An "Include" is an external scope that is visible and asserted by the owning lexical scope.   
[FUML] PackageImport  
[CL] Importation: An importation contains a name. The intention is that the name identifies a piece of Common Logic content represented externally to the text, and the importation re-asserts that content in the text.

### Direct Supertypes

[Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0)

## 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)

## 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.  
[CL] Text: A text is a set, list, or bag of phrases. A piece of text shall optionally be identified by a name.

### Direct Supertypes

[Namespace](#_9c5aa7f24b9d67e77921e06d105205c0)

## Class Logical Package

A model of information about systems independent of technical representation.

### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

## Class Mapping Package

A model defining relationships between other models.

### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

## 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)

## Class Package

A model element that provides a definitional scope for other model elements. A package may be represented as a "graph".  
  
[ISO 1087] concept system: system of concepts set of concepts (3.2.1) structured according to the  
relations among them  
  
[FUML] Package. FUML ownedMember corresponds with SMIF <defines>. FUML "nestedPackage" corresponds with "defines" where the element defined is a package.  
  
[CL] Module: A module consists of a name, an optional set of names called the exclusion set, and a text called the body text.

### Direct Supertypes

[Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

## Class Physical Package

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

### Direct Supertypes

[Package](#_0506f167988dfda7ae188b66aefe4f05)

## Association Prefix

Relationship defining the prefix for a package.

### Direct Supertypes

[Identification](#_5a0c9611d1c64dcbc0f89b5299e112ed)

### Association Ends

has prefix : [Prefix](#_f02d6606b404e367c8d0a72afd7f68e5) [0..1]



An abbreviation that can be used to identify a package.

prefix of : [Package](#_0506f167988dfda7ae188b66aefe4f05) [1]



An abbreviation for a package.

## Class Prefix

A technical abbreviation for a package.

### Direct Supertypes

[Unique Text Identifier](#_c9d4914a019b89a37f1f18103ebaf817)

## Association Scope of Reference

Relationship defining internal or external context that are referenced by a lexical scope using a lexical reference.

### Association Ends

Referenced scope : [Context](#_66d62b068053cee3464e1e03e6035eed) [1]



A referenced context, potentially in another model, that provides visibility to the elements in that context.  
[FUML] importedPackage

referenced by : [Lexical Reference](#_0315319befc74caa0a2a7d36cff333c0) [\*]



References to a context.

## Association Scope Reference

Relationship defining references for a scope.

### Association Ends

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



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

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



A lexical scope that is extended by a lexical reference.  
[FUML] importingNamespace

## Association Statement

Relationship defining the set of elements defined within and asserted by a lexical scope.

### Direct Supertypes

[Definition](#_3f8ee3c0c2369667c3f31d50e0ff6f83)

### Association Ends

states : [Thing](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



<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.

stated by : [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151) [0..1]



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

# SMIF Conceptual Model::Mapping

Mapping rules define how data represents concepts or how 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)

### Operations

public push ()



An operation called to evoke the behavior associated with a new facade element being created or modified. Push asserts the more concrete type based on a reference type.

public pull ()



An operation called to evoke the behavior associated with a facade representing existing elements. Pull asserts the reference type based on a more concrete type.

## Association Concrete Map End

Relationship to the more concrete end of a match rule.

### Association Ends

concrete end : [Match Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



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

match from : [Match Rule](#_63d69e49de8214503f0947e7f9dbc652) [0..1]



Mapping rule owning a "concrete" end.

## Association Concrete Pattern Relation

Relationship between a mapping and a pattern of the more concrete concepts to be mapped.

### Association Ends

concrete pattern : [Mapping Pattern](#_3f2e4e9acce53a41e40eecfb8ae833ae) [1]



The pattern in a mapping that represents a pattern of the "more concrete" data type.

concrete mapping : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [1]



Mapping for which a more concrete pattern is defined.

## 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)

## Association Map Rule Type Assertion

Relationship defining more concrete types that shall be asserted for an end of a match rule.

### Association Ends

asserted type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



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

asserted by : [Match Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [\*]



Map rule and that asserts a type

## Association Mapped Property

Relationship defining the property that is the source or target of a mapping

### Association Ends

maps property : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [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.

property end : [Property End](#_7decbae485fa9be88d63fca6f68547ec) [\*]



Map rule end for a property

## Association Mapped Relationship

Relationship defining the relationship pattern that is the source or target of a mapping

### Association Ends

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.

relationship end : [Relationship 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 in a "concrete" pattern may be defined to be a representation (data about) a concept in a "reference" pattern.  
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.  
Note that the roles of "concrete" and "reference" may or may not reflect different levels of abstraction and in some cases the choice may be arbitrary.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## Class Mapping Pattern

A pattern used within a mapping specification as a pattern of the more concrete or more abstract concepts.

### Direct Supertypes

[Pattern of Type](#_d887c32e4bfb53e43fcdbf0a0fa25c0f)

## Class Match 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)

### Attributes

coerce : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

default : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



True if the map should be enforced only if no other maps or prior values exist for the related ends.

## Class Match 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.  
Match 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.

### Attributes

condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)



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

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.

## Class Property End

An end of a match 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

[Match Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

## Association Reference Map End

Relationship to the reference end of a match rule.

### Association Ends

reference end : [Match Rule End](#_0d8a19bfafdae6e590a12e54ebcff122) [1]



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

match to : [Match Rule](#_63d69e49de8214503f0947e7f9dbc652) [0..1]



Mapping rule owning a reference" end.

## Association Reference Pattern Relation

Relationship between a mapping and a pattern of the more abstract concepts to be mapped.

### Association Ends

reference pattern : [Mapping Pattern](#_3f2e4e9acce53a41e40eecfb8ae833ae) [1]



The pattern in a mapping that represents a pattern of the "more abstract" set of concepts.

reference mapping : [Mapping](#_551417ad3c6e740d8b880bee8085a718) [1]



Mapping for which a more abstract pattern is defined.

## Class Relationship End

An end of a match rule where the end maps to a relationship defined in the pattern, the instances of the relationship between the properties will be mapped.  
The relationship must be defined in one of the mapped patterns of the mapping (concrete pattern or reference pattern).

### Direct Supertypes

[Match Rule End](#_0d8a19bfafdae6e590a12e54ebcff122)

## Association Representation

More concrete type that represents information about the represented concept of a representation rule.

### Association Ends

represented by : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The representation of a concept in a more specific form

represents rule : [Representation Rule](#_e2101d86d43ebb1c6717af7a9f48ebc1)



Rule defining a representation of a type.

## 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)

### Attributes

condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5)



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

map all : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



Specifies 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.

## Association Represented Concept

More abstract type that is <represented by> a more concrete type of a representation rule.

### Association Ends

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



A more general or abstract concept that is being represented.

concept rule : [Representation Rule](#_e2101d86d43ebb1c6717af7a9f48ebc1)



Rule defining a concept that is represented by another, more concrete, concept.

# SMIF Conceptual Model::Metadata

Metadata defines data about model elements (their source, definition or trust), which can be differentiated from model elements about the subject domain.

## Diagram: Metadata



1. Metadata

## Association Assertion Statement

Relationship defining the original statement, speech act or information artifact that asserted something in a model.

### Direct Supertypes

[Metadata relationship](#_b8c1a19be638ded573b2848c849fee69)

### Association Ends

was stated in : [Statement](#_fa97e8600fa5d7e45f0100b981e94ee8) [\*]



Metadata representing the speech act, document or other record where a statement captured in a model was made.

resulted in : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)



Statement made in a statement by an information source.

## 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.  
  
[ISO 1087] definition: representation of a concept (3.2.1) by a descriptive statement which serves to differentiate it from related concepts  
  
[FUML] Comment (where body corresponds with "text definition").

### Direct Supertypes

[Metadata](#_083d03a8bb38e1a0cab92a7dc3f1cf03)

### Attributes

text definition : [Text](#_f8b3b1f5ed63755061811cd6b69bc24f)



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

external reference : [IRI Identifier](#_f904ff1da5bfc3387d892b7e0fe9ecb1)



A reference to an external information resource that further defines something.  
[FIBO] ReferenceDOcument

external term : [Term](#_1945edd0888993a52c5dc6467a7b3ef8)



Specific term in an external resource that further defines something.

summary description : [Text](#_f8b3b1f5ed63755061811cd6b69bc24f)



A short description of something.

## Association Definition Relationship

Relationship between a thing and its definitions.

### Direct Supertypes

[Metadata relationship](#_b8c1a19be638ded573b2848c849fee69)

### Association Ends

defines : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [1]



Some thing described by a definition.  
[FIBO] defines  
[FUML]annotatedElement

defined by : [Definition](#_1a6d88e097d757268d09f68af82fbd34) [\*]



An informal description of something.  
[FIBO] hasDefinition  
[UML] comment  
[FUML] ownedComment

## Class Information Source

Metadata defining the origin or provenance of a set of statements in a model or data.  
Note that the source could be a human, an organization, a mapping or other automated processes.

### Direct Supertypes

[Actual Entity](#_e075b03ae73f89f5fcb1481cd5a16cbe), [Metadata](#_083d03a8bb38e1a0cab92a7dc3f1cf03)

## Class Metadata

Information about the source, provenance or origin of information. Metadata may be a managed entity, providing for provenance.  
[NIEM] MetadataType

### Direct Supertypes

[Record](#_8b38efa9c56da3bc8ecb501e56419e41)

## Association Metadata relationship

Relationship between something and metadata about that thing; data about data.

### Association Ends

metadata about : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]



The subject of metadata, the entity described by the metadata.

has metadata : [Metadata](#_083d03a8bb38e1a0cab92a7dc3f1cf03) [\*]



Metadata associated with (data about the information concerning) the subject entity.

## Association Record of a thing

Relationship between a thing and records (or information) about that thing.  
Note that in SMIF, things refer to the actual thing they represent, not data about it (unless the type is a record, in which case the "thing" is the data). This relationship recognizes that both a thing and data about the thing are things.  
  
[IDEAS] describedBy: A representedBy that asserts that a Description describes a Thing.

### Association Ends

about : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [\*]



The thing described by a record.

has record : [Record](#_8b38efa9c56da3bc8ecb501e56419e41) [\*]



A record about something.

## Association Source of Information

Relation defining an entity making a statement represented within a model. E.g. the person or organization that made a statement.  
  
[ISO 1087] source identifier: information in a terminological entry (3.8.2) which indicates the source documenting the terminological data (3.8.1)

### Association Ends

made statement : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [1..\*]



Metadata representing statements made by an authoritative source.   
Sources may be people, organizations, documents, information systems, etc.

has authoritative source : [Information Source](#_1e4f4f7a2bb7525a97cda0bc61f02036) [\*]



Metadata representing the authority behind a statement - who or what made a statement captured in a model.

## Class Statement

Statements provide metadata as to the source of information - who or what said it.  
This source of the information may be captured using "InformationSource" metadata about the metadata.

### Direct Supertypes

[Metadata](#_083d03a8bb38e1a0cab92a7dc3f1cf03)

### Attributes

statement date and time : [Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)



Metadata representing the date and time the statement was made or modified.

version : [Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)



Metadata representing an identifier for a version of information.

transaction id : [Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)



Identifier for an act or transaction creating or modifying information.

# SMIF Conceptual Model::Patterns

Patterns are templates for structures or compositions of things that may then be expressed as instances of the pattern.

## Diagram: Patterns



1. Patterns

## Association Defining Relative Property

Relationship connecting a property with a relative property such that the relative property defines additional semantics for the <relative to> property, within the context of the pattern.

### Association Ends

relative to : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



Property for which a relative property in a pattern defines additional semantics.

is defined by : [Relative Property](#_ea6776cff1b39b709bc823e54dd8cd52) [\*]



A property of a pattern that defines additional semantics of the <relative to> property.

## Class Focus Property

A property representing the extent of the subject type within the context of the owning pattern.  
The type of the property is asserted be the same as the subject type of the pattern.

### Direct Supertypes

[Quantified Property](#_ef5e0064e356df6ead8b83395eedd22c), [Type Pattern Property](#_e31b27a7745289f6f0e539acb4a8b867)

## Association Match Rules

Relationship defining the match rules for a mapping.

### Direct Supertypes

[Rule Constrains](#_9562d6c08cbe5eb32022ec9309bb6160), [Statement](#_ae63cfff50cedcc072b5771554ea61a3)

### Association Ends

has map rule : [Match Rule](#_63d69e49de8214503f0947e7f9dbc652) [\*]



Map rule that is asserted by a mapping.

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



Mapping containing a map rule.

## Class Part Property

A relative pattern property representing a part of the subject type. Additional relations and rules may be made about the part. A type with parts is by its nature a composition.  
The relationship(s) traversed by the path expression, resulting the relative part, define a composite relationship.

### Direct Supertypes

[Relative Property](#_ea6776cff1b39b709bc823e54dd8cd52), [Type Pattern Property](#_e31b27a7745289f6f0e539acb4a8b867)

### Attributes

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"

## 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 properties 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.  
  
[DTV] general situation kind: situation kind that is not an individual situation kind  
Note:This concept is defined in contrast to ‘individual situation kind’ not because there is any characteristic that distinguishes ‘general situation kind’ from ‘situation kind’.  
Note:A situation kind is a general situation kind if it can be exemplified by more than one Event in some possible world, even when it cannot have more than one Event in the possible world chosen to be the universe of discourse.  
  
[UML] StructuredClassifier. Also Similarity with TemplateSignature

### Direct Supertypes

[Situation](#_8c517cf1950741c0f89edebf828214cc), [Situation Type](#_50241f5936e61055293ca95f860768d8)

## Class Pattern of Type

A pattern of type defines a set of properties and relationships that must hold true for all instances of a type. Where the pattern includes parts, the subject type is a composition.  
Patterns augment the semantics of th subject type in the context of the pattern.

### Direct Supertypes

[Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f)

## Association Pattern Properties

Relationship defining properties within a pattern.

### Direct Supertypes

[Statement](#_ae63cfff50cedcc072b5771554ea61a3)

### Association Ends

owns property : [Pattern Property](#_4d83e476040c7444758dda440d3096fc) [\*]



A property defined within the context of a pattern that is used as part of the patterns definition.  
[UML] ownedAttribute

has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1]



Pattern owning a pattern property.

## 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.  
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.  
  
[UML] Similarity with TemplateParameter

### Direct Supertypes

[Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

### Attributes

condition : [Expression Node](#_f9bba899ada544a47c36bb071e9024f5) [\*]



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

explicit : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

## 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.

### Attributes

defaut : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

## 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.  
[UML] Connector. (type = has type). Each ConnectorEnd corresponds with a Structured Property Binding.

### Direct Supertypes

[Relationship](#_f7a7f80baaeb7cc3f36c45e96eacd166)

### Attributes

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.

explicit : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

## Association Pattern Relationship

Definition of a relationship that must hold within all instances of a pattern.

### Direct Supertypes

[Statement](#_ae63cfff50cedcc072b5771554ea61a3)

### Association Ends

owns relationship : [Pattern Relationship](#_ef18e024624b062995d1bb5acd223c15) [\*]



A relationship defined within the context of a pattern. The semantics of the relationship is further defined by "has strength".  
Note that a pattern may also assert other propositions, including other rules and relationships.  
[UML] ownedConnector

has owning pattern : [Pattern](#_8d9c945b6f864c34fdd7a91d4d62755f) [1]



Pattern owning a pattern relationship.

## Class Quantified Property

A quantified property defines a set of things resulting from the extent of the property type filtered by the quantifier. The resulting set may then be used in relationships or other assertions within the owning pattern.

### Direct Supertypes

[Pattern Property](#_4d83e476040c7444758dda440d3096fc)

### Attributes

quantifier : [Quantifier](#_b82d09af5a9584abc6560d3bfb03e524) [1]



A quantified property defines a quantification within a pattern. The quantifier defines the set of things that will populate the pattern property for all instances of the pattern. Default is "All".  
  
e.g. for all people p: People is the context and P is the quantified property. In SMIF 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.

## Class Relative Property

A relative property defines additional semantics of the <relative to> property in the context of the owning pattern.

### Direct Supertypes

[Pattern Property](#_4d83e476040c7444758dda440d3096fc)

### Attributes

path expression : [Traversal](#_0492440b12b90a76377a15324efa2182) [0..1]



Path expression 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.

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



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

## Association Subject of Pattern Relationship

Relationship defining the subject pattern of a type specific pattern.

### Direct Supertypes

[Assertion](#_98ff7066ce9f28f3ab4a80f88bc3fddc)

### Association Ends

asserts pattern : [Pattern of Type](#_d887c32e4bfb53e43fcdbf0a0fa25c0f) [0..\*]



A pattern asserted for all instances of a type. Where the pattern includes parts, the type defines a composition.

subject type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The type which is the context of a pattern of type. The pattern is "about" the subject type.

## Class Type Pattern Property

Type Pattern property is an abstract supertype that provides for a restriction that parts and focus properties must be owned by a pattern of a type.

### 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 SMIF Conceptual Model::Patterns

public enum Pattern Element Strength

{Select, Assert, Default, Exists}

#### Literals

Select



Select is used in query and mapping patterns, all elements of the classified type that match the pattern are selected as instances of the pattern.  
Select may be considered a qualified "All". Select 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>=Select must hold between the selected properties for the pattern to be asserted.

Assert



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

Default



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

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 SMIF Conceptual Model::Patterns

public enum Quantifier

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

#### Literals

None



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

There Exists



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

Exactly One



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

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".

All



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

# SMIF Conceptual Model::Properties

Properties define the most granular connections between entities or values. Properties may be used as the ends of relationships, to represent individual characteristics or as elements of a data structure.

## Diagram: Properties



1. Properties

## 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

[Characteristic Type](#_9ee6787ca6750591f74aeb580057443b)

## Association Bound Individual

Relationship defining the thing bound to a subject based on a bound property - the "object" of the property binding.

### Association Ends

binds : [Thing](#_a52cb0ff6e414b3170b58afe10b6afcb) [1]



The thing bound to a property in a specific situation. E.g. if the weight of truck-XYZ is 4500 LBS, the bound individual would be "4500 LBS".  
[FUML] value

bound in : [Property Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



Bindings in which a thing participates.

## Association Bound Property

Relationship defining the property type that defines the semantics of a property binding. E.g. if the weight of truck-XYZ is 4500 LBS, the bound property could be "has weight".

### Direct Supertypes

[Extent of Type](#_7930d7b301f56f0155603422a27ad833)

### Association Ends

has binding : [Property Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



Bindings referencing a property.

bound by : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1]



The property a binding binds a thing to.  
[FUML] definingFeature

## Association Bound Subject

Relationship defining the subject of a bound property. Where the subject is a relationship, the relationship becomes transparent and the applicable subject(s) are the other ends of the relationship. E.g. if the weight of truck-XYZ is 4500 LBS, the bound subject would be Truck-XYZ".

### Association Ends

has binding : [Property Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64) [\*]



Bindings asserted for properties within a situation.

bound to : [Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d) [1]



The subject of a property binding.  
[FUML] owningInstance (note that in SMIF the owner and subject may not be the same). Where the are the same, the semantics are the same as FUML.

## Class Characteristic Binding

A characteristic of a specific thing, e.g. the color of Pump-1234 in the <bound to> entity. A characteristic is a "first class" element and may participate in relationships and have annotations.  
  
[IDEAS] measureOfIndividual: A typeInstance that asserts an Individual is an instance of a Measure - i.e. the Individual "has" a property corresponding to the Measure.  
  
[ISO 1087] characteristic: abstraction of a property of an object (3.1.1) or of a  
set of objects  
  
[Guizzardi] quality(x) =def ∃!U qualityUniversal(U) ∧ (x::U)

### Direct Supertypes

[Actual Situation](#_318306db8339a16351b356169444c6ed), [Property Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64)

## Class Characteristic Type

A kind of characteristic of a type of thing may have, e.g. paint may have a color. Characteristic kind is the type of characteristic bindings which are "first class" elements and may participate in relationships and have meta-characteristics.  
  
[IDEAS] Property: An IndividualType whose members all exhibit a common trait or feature. Often the Individuals are states having a property (the state of being 18 degrees centigrade), where this property can be a CategoricalProperty (qv.) or a DispositionalProperty (qv.).  
  
[ISO 1087] type of characteristics: category of characteristics (3.2.4) which serves as the criterion of subdivision when establishing concept systems. NOTE The type of characteristics colour embraces characteristics (3.2.4) being red, blue, green, etc. The type of characteristics material embraces characteristics made of wood, metal, etc.  
  
[FIBO] Simple Property: Simple Properties are assertions about things in a class, which may be framed in terms of some simple type of information.   
  
  
[Guizzardi] qualityUniversal(U) =def intrinsicMomentUniversal(U) ∧ ∃!x QS(x) ∧ assoc(x,U)

### Direct Supertypes

[Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d), [Situation Type](#_50241f5936e61055293ca95f860768d8)

## Association Properties Relationship

Relationship defining the set of properties defined for a type.  
Where the <property of> type is a relationship type, the "subject" of the property is the other ends (properties) of the relationship.  
Where the <property of> type is not a relationship, the subject of the property is the <property of> type.

### Association Ends

has property : [Property Type](#_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.  
[FUML] feature  
[UML] memberEnd. attribute (of classifier).

property of : [Type](#_dfe1514224ca21cedba7b2b29802db50) [0..1]



Type for which a property is relevant. The domain of the property.   
[FUML] featuringClassifier

## Class Property Binding

A property value binding binds a particular thing (the value) to a situation based on a defined property.  
  
For relations (which are structures), this is also known as an "association end".  
  
Where <binds> is an expression evaluation, the property value shall evaluate to the evaluation of the expression.  
  
Where <binds> is a property, the property value shall be the property values bound to that property in <bound to> situation.  
  
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 bound thing will be asserted.  
  
The type of the <bound to> structure must (directly or indirectly) have the type the <bound by> properties <property of> type.  
  
[FUML] Slot (Noting that in SMIF the binding may or may not be owned by the subject, depending on the subtype of property).  
  
[CL] Binding:

### Direct Supertypes

[Thing](#_a52cb0ff6e414b3170b58afe10b6afcb)

## Class Property Type

A property type defines the way in which instances of a type participate in (or, are involved in) instances of another type (including relationships). Sometimes called a variable, argument or role.  
In a conceptual model the terms associated with a property kind are typically "verb phrases" defining how instances of the involved type participate in the situation or relationship.  
In a record (data structure) the property is a "slot" of a record and may have a term which is a noun or verb phrase.  
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.  
  
[Guizzardi] MomentUniversal(U) =def Universal(U) ∧ ∀x (x::U → Moment(x))  
  
[FUML] Parameter where owner is operation. Otherwise Property.  
[UML] Property. LAll typed elements in SMIF are Property Types. redefinedProperty and subsets corresponds with a GeneralizationConstraint.  
  
[CL] Operator: distinguished syntactic role played by a specified component within a functional term

### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

# SMIF Conceptual Model::Relationships

Relationships are primitive facts about anything, relating individuals through properties of the relationships. Relationships have their semantics described by a relationship type. The ends of relationships are defined by "structured property type", a relationship may have any number of "ends".

## Diagram: Relationships



1. Relationships

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

## Class Relationship

A relationship defines a condition involving related things. A relationship may be asserted within a context as true or false within that context. Each relationship type has a number of bindings of which do not change for the life of the relationship..   
A relationship may be true or false within its context (including a timeframe) but is atomic in its truth value.  
Relationships may participate in (be bound to) other relationships and as such bindings involving a relationship may change over time. That is, relationships are "first class" objects.  
  
[IDEAS] tuple: A relationship between two or more things.  
Note: SMIF allows one end of a relationship.

### Direct Supertypes

[Actual Situation](#_318306db8339a16351b356169444c6ed), [Structure](#_e60871f18b94666411d0d4023a66bd0b)

## Class Relationship Type

A relationship type defines a type of condition, the relationship, involving related things. A relationship may be asserted within a context as true or false within that context. Each relationship type has a number of <has property> "structured property type" properties which describe the role of the related things with respect to the relationship, values of which uniquely do not change for the life of the relationship.   
A relationship may be true or false within its context (including a timeframe) but is atomic in its truth value.  
Relationships may participate in (be bound to) other relationships and as such bindings involving a relationship may change over time.   
The terms for properties of a relationship in a conceptual model are typically verb phrases, connecting the relationship with the related types.  
  
[FIBO] Mediating Thing  
  
[IDEAS] TupleType: The Powertype of tuple.  
  
[FUML] Association where memberEnd corresponds with <has property>. Note that SMIF relationships are "first class" and may also be considered to correspond to an association class where there are any properties or other relationships referencing the subject relationship.  
  
[UML] AssociationClass (note that "end ownership" is meaningless in SMIF).  
  
[Guizzardi2015] Relator: endurants of a special kind, with the power of connecting (mediating) other endurants. Note: Guissardi "mediation" corresponds with relationship properties.

### Direct Supertypes

[Situation Type](#_50241f5936e61055293ca95f860768d8), [Structure Type](#_3b0c6b335aca4015ef569068da1bec31)

# SMIF Conceptual Model::Rules

Rules define constraints or behaviors that are asserted in specified context.

## Diagram: General Rules



1. General Rules

## Diagram: Property Constraints



1. Property Constraints

This diagram focuses on rules about properties.

## Diagram: Rules in Context



1. Rules in Context

This diagram shows how rules are propositions that may be asserted within any context to apply to any other context, thus realizing the "open world assumption".

## Diagram: Rules Summary



1. Rules Summary

This diagram shown a summary of the primary rules.

## Diagram: Type Constraints



1. Type Constraints

This diagram focuses on rules about types (note that property types are also types).

## Class Conditional Rule

A rule with a general expression as a condition that applies to what the rule <constrains>. Where asserted, the condition must be true.  
[UML] Constraint where "context" corresponds with <holds within> and "constrainedElement" corresponds with "constrains". "specification" corresponds with "condition".

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

### Attributes

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.  
[UML] GeneralizationSet with isCovering=TRUE. "constrains" corresponds with the common "general" of each Generalization". "is covered by" corresponds with each "special" of each generalization.

### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

## Association Covering Constraint

Relationship defining the types covered by a covering constraint.

### Association Ends

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.

has covering : [Covering Constraint](#_507049575ebfa9f535e8f25db14a0760) [\*]



Covering constraints of a type.

## Class Disjoint

Disjoint is a rule that the things denoted by what the rule <constrains> do not and may not denote any of the same set of things.  
When applied to a context (including types) all elements contextualized are included in the set of disjoint individuals.  
  
[FIBO] Mutually Exclusive sets  
  
[IDEAS] PartitionOfSetOfDisjointIndividuals: A FusionOfSetOfIndividuals whose fusioned Type is a SetOfDisjointIndividuals.  
  
[UML] [UML] GeneralizationSet with isDisjoint=TRUE. "constrains" corresponds with "is covered by" of each "special" of each generalization. Note the SMIF does not require that disjoint elements have a common supertype, one may be inferred for UML mapping.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## 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 in <holds within>.  
  
[FIBO] Selections of Things  
[FUML] Wen constraining a type, corresponds with [FUML] "Enumeration". SMIF enumerations are not limited to literals. The "ownedLiteral" corresponds with all elements owned by <holds within>.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## Class Equivalent

Equivalent is a rule that the things the rule <constraints> denote the same set of things. When applied to a context (including types) each thing the context contextualizes is included in the set of equivalent things.  
  
Related to\*: [ISO 1087] synonymy: relation between or among terms (3.4.3) in a given language representing the same concept (3.2.1)  
  
Related to\*: [ISO 1087] equivalence: relation between designations (3.4.1) in different languages representing the same concept (3.2.1)  
  
\* SMIF relates concepts, not terms. synonymy may also be represented by multiple terms for the same concept.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## Class Facet Classification Constraint

A Facet Classification Constraint asserts that 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" is a facet of a person.   
  
[FIBO] isPlayedBy

### Direct Supertypes

[Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf)

## Association Generalization

Relationship defining the general type of a generalization constraint.  
  
[ISO 1087] generic concept: concept (3.2.1) in a generic relation (3.2.21) having the narrower intension (3.2.9)

### Association Ends

has general : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The general type in the Generalization rule.  
  
[ISO 1087] concept (3.2.1) in a generic relation (3.2.21) having the broader intension (3.2.9)  
  
[FUML] General (Where redefines is false or not defined)  
[FUML] RedefinableElement.redefinedElement (Where redefines is true)

has specialization : [Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]



Specialization rules for a type.

## Class 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.  
  
[IDEAS] superSubtype: A couple relating two Types which asserts that one type is a subset of the other.   
  
[ISO 1087] generic relation: genus-species relation relation between two concepts (3.2.1) where the intension (3.2.9) of one of the concepts includes that of the other concept and at least one additional delimiting characteristic (3.2.7)  
  
[FIBO] Inheritance  
  
[UML] Generalization  
  
[Guizzardi] (Specialization relation): Let F and G be two universals such that F is a specialization of G. Then, for all w ∈ W we have that extw(F) ⊆ extw(G)

### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

### Attributes

redefines : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



Defines the generalization as a redefinition, subsuming the more general type.  
  
Where <redefines> is true the more specific type subsumes the more general type in the definition context. In this case the more general and more specific sets are equivalent. A type 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 type represents a subset of the more general property.  
  
Redefinition is most often used with properties (as defined in UML) but may also be applied to other types.

## Class Multiplicity Constraint

A Multiplicity constraint constrains the number of bindings <multiplicity of> types (including property types) may have in a particular instance of the constrained type.   
  
For a property type, The number of instances bound to a property for the set of instances bound to <with respect to> shall be limited by the minimum and maximum number of the multiplicity.   
  
For non-property types, the multiplicity shall apply to the extent of the type as described by <classifies>.  
  
[IDEAS] superSubType  
  
[FUML] MultiplicityElement: Note: Multiplicity Constraint constraining a type has semantics included in to UML MultiplicityElement.

### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

### Attributes

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



Minimum number in a set as constrained by a multiplicity.  
[FUML] MultiplicityElement.lowerValue

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



Maximum number in a set as constrained by a multiplicity.  
[FUML] MultiplicityElement.upperValue

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).

is sufficent : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

## Association Multiplicity Reference

Multiplicity may be defined between things. E.g. there are 2 wheels on a motorcycle. This is most often required where relationships have more than 2 ends.   
Multiplicity reference defines the "from" side of such a multiplicity (e.g. the motorcycle).

### Association Ends

with respect to : [Type](#_dfe1514224ca21cedba7b2b29802db50) [\*]



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, data structures and relationships.

respect of : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]



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

## Association Multiplicity Target

Relationship defining the type a multiplicity rule applies to. Note that properties are types and may also have multiplicity constraints.

### Direct Supertypes

[Rule Constrains](#_9562d6c08cbe5eb32022ec9309bb6160)

### Association Ends

multiplicity of : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



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

has multiplicity : [Multiplicity Constraint](#_4ff1432ad36ac8beb6cbb7e9323d9f24) [\*]



Multiplicity constraint of a type or property.

## Class Property Constraint

Abstract supertype for constraints that constrain properties types.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## Class Property Transitivity Constraint

A transitive property defined by <constrains> 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)

## Association Property Type

Relationship defining the type of a property.

### Association Ends

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>.

properties of type : [Property Type Constraint](#_3e4d7d36a3b2cb4c9b85c1b88930178a) [\*]



Properties typed by a type

## Class Property Type Constraint

A property type constraint 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.  
  
[FUML] TypedElement.type: Note: A property type constraint applied to a property has the same semantics as a UML TypedElement.

### Direct Supertypes

[Property Constraint](#_91be190e8014514a597300b286148d3e)

### Attributes

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.

## 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 or what the rule is constraining.

### Direct Supertypes

[Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1)

## Association Rule Constrains

Relationship defining the entity constrained by a rule. Where no constrained entity is specified, all entities are constrained with the scope of <holds within> are constrained.

### Association Ends

constrains : [Identifiable 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.

constrained by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]



Rules applying to an entity.

## Association Rule Subsumption

Relationship defining rule subsumption. When a rule subsumes another the subsumed rule will not apply (fire) if the <subsumed by> rules applies (fires).

### Association Ends

subsumes : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]



When a 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.

subsumed by : [Rule](#_82919e40af9ad2e13647e9d37bbf0956) [\*]



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

## Association Specialization

Relationship defining the specific type of a generalization constraint.

### Direct Supertypes

[Rule Constrains](#_9562d6c08cbe5eb32022ec9309bb6160)

### Association Ends

has specific : [Type](#_dfe1514224ca21cedba7b2b29802db50) [1]



The specific type in a generalization rule.  
[ISO 1087] generic concept: concept (3.2.1) in a generic relation (3.2.21) having the narrower intension (3.2.9)  
[FUML] specific

has generalization : [Generalization Constraint](#_5f3998cf1a072f724861db93cee66cbf) [\*]



Generalization rules for a type

## Class Type Constraint

A constraint of a type, including Relationships types.

### Direct Supertypes

[Rule](#_82919e40af9ad2e13647e9d37bbf0956)

## Association Unique Set

Relationship defining the set of properties that uniquely identify an instance of the constrained type.

### Association Ends

has unique : [Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d) [1..\*]



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

has uniqueness constraint : [Uniqueness Constraint](#_982e84b7afc784b4d0aa763204953a3d) [\*]



Uniqueness constraints for a property.

## Class Uniqueness Constraint

A constraint that, within the <constrains> 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.   
Note: Uniqueness may be used to define a "key".

### Direct Supertypes

[Type Constraint](#_ded47679f07683882f8f128d6911711a)

### Attributes

is primary identity : [Boolean](#_6119a00b0834641b9fe3f5ae9f58237f)



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

# SMIF Conceptual Model::Situations

A situation is a particular configuration of things and their relations including spatial, temporal, and logical connections between those things valid over a period of time. Situations form the basis of all complex, time dependent entities.

## Diagram: Situations



1. Situations

## Class Actual Situation

An actual situation is an individual situation that actually exists, happened in the past or may exist in some possible world, not a template or process definition. Such situations must exist for a time interval, however there are no constraints on such a time interval - from an instant to the life of the universe.  
  
DTV: Occurrence: state of affairs that is a happening in the universe of discourse

### Direct Supertypes

[Actual Entity](#_e075b03ae73f89f5fcb1481cd5a16cbe), [Situation](#_8c517cf1950741c0f89edebf828214cc)

## Class Situation

A situation is an identifiable entity composed of an arrangement of entities and the relations between them over a time interval. Situations are propositions and may be asserted as true or false in some context. Situations may change over time, unless otherwise constrained. As an identifiable entity, situations may participate in relationships, thus situations are "first class" elements in SMIF.  
  
[SBVR] "State of affairs"

### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed), [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151), [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1), [Temporal Entity](#_f2afd42e2b6e88484b5534f68f8549c1)

## Class Situation Type

A situation type defines a kind of identifiable arrangement of individuals, assertions and the relations between them over a timespan. As an identifiable entity, situations may participate in other situations and relationships by being bound to properties of those situations or relationships with bindings, thus situations are “first class” entities in a SMIF model.  
The roles or behaviors things (any entity or value) may play in a situation are identified as properties of the situation type.  
Entity types and roles may also be situation types.  
Syn. Type of a state of affairs.  
A situation type may have properties such that instances, may bind things to structures based on properties.  
Things may be bound to a structure (i.e. play a role in the structure) via properties. Things bound to properties of a structure may change over time, unless otherwise constrained.  
  
[DTV] situation kind: state of affairs that may or may not happen in some possible world

### Direct Supertypes

[Entity Type](#_a09117831b97c480bde825e7cd3696eb)

# SMIF Conceptual Model::Structures

Structures provide for things with properties (or variables) that are part of their logical or physical composition.  
  
Records represent structured data, usually used for logical or physical data models, not conceptual models.

## Diagram: Records



1. Records

## Diagram: Structures



1. Structures

## 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.  
  
[IDEAS] A Representation that describes a Thing

### Direct Supertypes

[Actual Situation](#_318306db8339a16351b356169444c6ed), [Structure](#_e60871f18b94666411d0d4023a66bd0b)

## 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](#_50241f5936e61055293ca95f860768d8), [Structure Type](#_3b0c6b335aca4015ef569068da1bec31)

## Association Record types for a type

Relationship defining types of records for a type.

### Association Ends

about type : [Type](#_dfe1514224ca21cedba7b2b29802db50) [0..1]



Thing for which a record exists

recording typs : [Record Type](#_d2ebf1b96697234b6aef9b3bfac15784) [\*]



Record for a thing.

## Class Structure

A structure can "own" property bindings and is the abstract supertype of structured values and records.  
The only purpose for structure in the model is to differentiate records (which are identifiable entities) form structured values.

### Direct Supertypes

[Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)

## Class Structure Property Binding

A structured element binding defines a value for a particular property of a particular structure.   
Similar to an OWL triple, a structure property binding does not have independent identity.  
[UML] ConnectorEnd (when used in a pattern).

### Direct Supertypes

[Property Binding](#_e829344c78ea1a9e5e18c7bc51ff8f64)

## Class Structure Type

Structure type is any type that can "own" properties - record types, relationships and structured value type.

### Direct Supertypes

[Entity Type](#_a09117831b97c480bde825e7cd3696eb)

## Class Structured Property Type

A structured element type is a property definition defined as a composite part of a structured type - most often used in data structures and relationships.  
  
[FIBO] Relationship Property  
[FUML] StructuralFeature

### Direct Supertypes

[Property Type](#_aec2b4f875c8e48059ff0f3cf4fdb05d)

# SMIF Conceptual Model::Top level

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

## Diagram: Top Level



1. Top Level

Diagram showing summary of top level classes and significant subtypes.

## Class Actual Entity

An actual entity is an identifiable and individual person, specific object, process, agreement, etc. Actual Individuals do not have to be physical but do not include types, categories or values.  
A more specific class of thing (e.g., Person) is intended to refine the classification of the individual thing.  
Individuality (or selfhood) is the state or quality of being an individual; particularly of being separate from other individuals and possessing identity. Actual entities typically have a lifetime and some individuals may change over that lifetime. Individuals may have parts that together help define the individual but may change over time.   
"Actual" does not imply current existence.  
  
[BFO] "Endurant" in [BFO] and other ontologies  
[ISO 1087] individual concept: concept (3.2.1) which corresponds to only one object   
  
[UML] Loose correspondence with "InstanceSpecification". SMIF instances are direct instances of their types, there is no "indirection" through value specification as their is in UML.  
  
[Guizzardi] (individual concept)  
Let M = 〈W, D〉 and U = U w W D w ∈ ( ) .  
An individual concept i in M is function from W into U, such that i(w) ∈ D(w) in all worlds. For a given model structure M we define I as a set of individual concepts defined for that structure.  
  
[CL] Individual: one element of the universe of discourse



1. Actual Entity Detail

### Direct Supertypes

[Temporal Entity](#_f2afd42e2b6e88484b5534f68f8549c1)

## Association Assertion

An assertion relationship between a context and the propositions asserted within that context. The <asserts> proposition is asserted (defined as "true") for all things contextualized by the <holds within> context. Assertion of truth is not absolute, it is relative to the context. For example, something could be asserted within a context where that entire context is asserted to be false.

### Association Ends

asserts : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*]



Proposition that is asserted (must be true) for anything contextualized by a context.  
As types are a context, types may assert a proposition for their instances.

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 <contextualizes> things that are related in some way.  
  
A <Context> also <asserts> propositions that hold for all things the context <contextualizes>, thus providing the link between an assertion and the set of things asserted. Likewise a context <negates> propositions that are false within the context.  
  
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.  
  
[CL] Sort: any subset of the universe of discourse over which some quantifier is allowed to range  
  
[ISO 1087] concept field: unstructured set of thematically related concepts (3.2.1)



1. Context Detail

### Direct Supertypes

[Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)

## Association Extent of Context

The association between a context and the set of things contextualized by that context, defining the extent of the context, a set.  
  
[ISO 1087] extension: totality of objects (3.1.1) to which a concept (3.2.1)  
corresponds

### Association Ends

contextualizes : [Thing](#_a52cb0ff6e414b3170b58afe10b6afcb) [\*]



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

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



A <Context> that contextualizes a thing making what it <contextualizes> subject to the propositions referenced by <has assertion> of the context.  
A thing may be <in context of> one or more context.  
[FIBO] hasContext

## Class Identifiable Entity

An identifiable entity is any identifiable thing other than values, this includes individuals, types, axioms, situations, speech acts, information structures, etc.  
Identifiable entities always have some kind of identity and may have identifiers. Note that identity is an abstraction that may have representation in models as any number of identifiers, also known ad a "sign".



1. Identifiable Entity Detail

### Direct Supertypes

[Thing](#_a52cb0ff6e414b3170b58afe10b6afcb)

## Association Negation

An assertion relationship between a context and the propositions negated (FALSE) within that context. The <negates> proposition is asserted as FALSE for all things contextualized by the <negated within> context. Assertion or negation of truth is not absolute, it is relative to the context.

### Association Ends

negates : [Proposition](#_3bd7c7d249201ad6f2447c6d182ba7f1) [\*]



Proposition that is negatively asserted (must be FALSE) for anything contextualized by a context.  
As types are a context, types may assert or negate a proposition for their instances.

negated within : [Context](#_66d62b068053cee3464e1e03e6035eed) [\*]



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

## Class Proposition

A proposition is statement, or condition with a truth value (true or false) that can be determined or asserted with some level of confidence (assessment of confidence being outside of this specification).  
All "facts", statements, speech acts, relationships and rules are propositions.  
Propositions may be asserted to be true within a context which they <holds within>.  
For a situation, the proposition is true if the situation is actual (i.e., takes place, obtains).  
  
[SBVR] the state of affairs is posited by the proposition and if the state of affairs were actual, the proposition would be true  
[CL] Sentence: unit of logical text which is true or false, i.e. which is assigned a truth-value in an interpretation



1. Proposition Detail

### Direct Supertypes

[Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)

## Class Temporal Entity

A temporal is anything that has a timespan. Temporal things may have temporal relationships with other temporal things.  
  
Note that relationships defined for [DTV] Time Intervals may be specified for <temporal Entity> but are not specified in SMIF.



1. Temporal Entity Detail

### Direct Supertypes

[Identifiable Entity](#_eb8398b5a178c638b98597120ec51c4d)

## Class Thing

Any thing or value that does or may exist in any possible world. Thing is the supertype of all types and may therefore participate in unbounded relations.  
Instances of Thing are referred to as "a thing" in this model.  
  
[IDEAS] Thing  
[OWL] Thing  
[ISO 1087] object: anything perceivable or conceivable  
[FIBO] Thing  
[Guizzardi] Thing  
[FUML] Element



1. Thing Detail

# SMIF Conceptual Model::Types

Types 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 Entity Type

A type of an identifiable entity. All concrete entity instances must have at least one entity type. Entity type may be mixed with other types to fully define an entity.  
  
[FUML] Classifier  
  
[Guizzardi] A Rigid Universal.  
(Rigid Universal): A universal G is rigid (or modally constant) iff for any w,w ∈ W 3. extw(G) = extw(G) Putting definitions 4.1 and 4.3 together, we have that for any rigid universal G the following is true 4. ext(G) = extw(G), for all w ∈ W A rigid universal is one that applies to its instances necessarily, i.e., in every possible world. Every substance sortal G is a rigid universal. Examples of non-rigid Sortals include universals such as Boy and Adult Man in the example  
discussed above, but also Student, Employee, Caterpillar and Butterfly, Philosopher, Writer, Alive and Deceased. Actually, these examples of sortals are not only non-rigid, but they are anti-rigid. Non-rigidity is the simple logical negation of rigidity, i.e., a universal is non-rigid if it does not apply necessarily to at least one of its instances. In contrast, a universal is antirigid if it does not apply necessarily to all its instances. An example of a non-rigid universal is Seatable. Suppose the instances of Seatable include a  
particular chair x and a particular crate c. In this case, whilst x instantiates Seatable necessarily, this is not the case for c: a crate can cease to be steady to afford sitting but still be the same crate.  
Non-rigidity and anti-rigidity of universals are defined formally in the  
sequel:

### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

## Association Extent of Type

The relation between a type and the things that type categorizes, the instances which defines the extent of the type, a set.  
  
[IDEAS] typeInstance: A couple that asserts that a Thing is a member of a Type.  
  
[Guizzardi] (Extension functions): Let W be a non-empty set of  
possible worlds and let w ∈ W be a specific world. The extension function extw(G) maps a universal G to the set of its instances in world w. The extension function ext(G) provides a mapping to the set of instances of the  
universal G that exist in all possible worlds, such that ext(G) = U w∈W w ext (G)

### Direct Supertypes

[Extent of Context](#_52c887644007b8e51a1f6e976113707a)

### Association Ends

categorizes : [Thing](#_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.  
[FIBO] classifies

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



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

## Class Facet

A facet is a "mix in" type that defines an aspect of something but does not define the identity or "fundamental" (A.K.A. "Rigid") type 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 to define that things identity.  
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 a Facet Generalization Constraint. E.g. Policeman is a role of a person.  
  
[Guizzard] (Non-Rigid Universal): A universal G is non-rigid iff for a w ∈ W There is an x such that x ∈ extw(G), and there is a w∈ W such that x ∉extw(G)

### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

## 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)

## Class Type

A <Type> is a categorization of any thing based on specific criteria. The specific criteria may or may not be formalized in a model.  
A <Type> <categorizes> a set of <Thing>s which comprises the "extent" of the type.  
A <Type> is a <Context> where the things it <categorizes> are <in the context> of the <Type>.  
  
[IDEAS] Type: A set (or class) of Things.   
[ISO 1087] general concept: concept (3.2.1) which corresponds to two or more objects (3.1.1) which form a group by reason of common properties  
[FIBO] Classifier: a standardized classification or delineation for something, per some scheme for such delineation, within a specified context  
[FUML] Type  
[CL] Type:: logical framework in which expressions in the logic are classified into syntactic or lexical categories (types) and restricted to apply only to arguments of a fixed type

### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed), [Lexical Scope](#_693daf0a0de3f4b82a04aee474c3f151)

### Associations

has supertype []



Supertypes(s) of a type as defined by generalization rules.  
  
All statements made about the supertype are true for the subtype. The extent (categorizes) of the subtype is a subset of the extent of the supertype.  
  
Has supertype is a a derived association based on generalization rules.

## 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.   
  
[FIBO] Logical Unions  
  
[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)

# SMIF 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 SMIF language provides specific support for specifying quantity kinds and unit types in conceptual, logical and physical models. The SMIF mapping rules may then perform the appropriate unit conversions.

The foundation of information specification in SMIF 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 SMIF 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

**SMIF concepts of quantities and units**

SMIF 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.

SMIF defines three types to realize the above goals: Quantity Kind, Unit Type, Base Unit Type. SMIF 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 SMIF 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 SMIF 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 SMIF 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 SMIF 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   
  
[FIBO] (type of) Base Unit: a measurement unit that is defined by a system of units to be the reference measurement unit 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)

## 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.  
  
[FIBO] QuantityKind: a categorization type for “quantity” that characterizes quantities as being mutually comparable

### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

## 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 subclass of a 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.  
  
[IDEAS] ScaleMapping: A CoupleType whose members are all the couples linking MeasurePoints to RealNumbers. The CoupleType (i.e. the set of couples) represents the scale.  
  
e.g. 5cm is an instance of the unit type "Centimeter"  
  
[FIBO] QuantityValue: number and measurement unit together giving magnitude of a quan-tity  
  
[Guizzardi] (quale): A point in a n-dimensional quality domain can  
be represented as a vector v = x1xn where each xi represents each of the  
integral dimensions that constitute the domain. A multidimensional quale is  
therefore the vector representing the several quality dimensions that are  
mutually dependent in a quality domain.

### Direct Supertypes

[Value](#_a739673c8d53da123e392b7e5059ceec)

### Attributes

value : [Numeric](#_49817c2cd3739b3fa65ff747c68620db)



Numeric value of a quantity

## Association Referenced System of Units

Relationship between a system of units and the set of unit types defined within that system.

### Direct Supertypes

[Extent of Context](#_52c887644007b8e51a1f6e976113707a)

### Association Ends

defined within system : [System of Units](#_7e22047cc4643bdb106af5bc777cd98a) [0..1]



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

unit of system : [Unit Type](#_9a97d5f73bf658c81147f5fab194bf88) [\*]



Unit type defined within a system of units

## Class Structured Value

A value that may have sub-elements (owned properties) defined as "structure property type".

### Direct Supertypes

[Structure](#_e60871f18b94666411d0d4023a66bd0b), [Value](#_a739673c8d53da123e392b7e5059ceec)

## Class Structured Value Type

A structured value type is a type of value that has parts represented as properties - also used for "data types" and forms.

### Direct Supertypes

[Structure Type](#_3b0c6b335aca4015ef569068da1bec31), [Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

## 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.  
  
[FIBO] SystemOfUnits: a set of measurement units associated with a system of quantities, together with a set of rules that assign one measurement unit to be the base unit for each base quantity in the system of quantities and a set of rules for the derivation of other units from the base units

### Direct Supertypes

[Context](#_66d62b068053cee3464e1e03e6035eed)

## 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 or other units.  
  
Note that unit types are not units, but the type of quantity values expressed in a common unit as defined in [JCGM 200:2008].  
  
[IDEAS] MeasureCategory: A MeasureType whose members are recognized types of MeasureInstance.

### Direct Supertypes

[Value Type](#_b08132d9b30f1d47632a28aa6e4894bf)

### Attributes

ratio : [Real](#_aef4bcae5ebc35dd9653214547b3e3cc)



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

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.

symbol : [String](#_e8a6ce315d976318da3ab784a645ea44)



The accepted symbol for the unit referenced by the unit type

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.

## Class Value

A Value is an atomic. immutable piece of information without a specific lifetime or identity independent of the value. Values include numbers, strings and other atomic "primitive" data. Values also include structured values, which are immutable.  
  
In UML values may be defined by the name of an instance specification with a value type.  
  
[IDEAS] Representation: A SignType where all the individual Signs are intended to signify the same Thing.

### Direct Supertypes

[Thing](#_a52cb0ff6e414b3170b58afe10b6afcb)

## 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.  
  
[IDEAS] RepresentationType: A Type that is the Powertype of Representation.  
  
[FUML] DataType

### Direct Supertypes

[Type](#_dfe1514224ca21cedba7b2b29802db50)

# Primitive Type

## Boolean

A value that may be true or false.  
  
[FIBO] boolean

## Integer

A number with no fraction.  
  
[IDEAS] Integer: A RationalNumber that can be written without a fractional or decimal component.   
  
[FIBO] integer

## Number

An arithmetical value representing a particular quantity and used in counting and making calculations and for showing order in a series or for identification.  
  
[IDEAS] A Type that is a number - i.e. a RealNumber or an Integer  
  
[FIBO] number  
[NIEM] NumericType

## Percentage

A unitless measure from 0 to 1.  
  
[FIBO] PercentageValue  
[NIEM] PercentType

## Real Number

Any number that may have a fraction.  
  
[IDEAS] RealNumber: A Type that is a Dedekind cut of the set of rational numbers.

## 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.  
  
[FIBO] Text  
[NIEM] TextType

# Data Type

## Data Value

Data value is equivalent with value and only required due to UML restriction that there can be no supertype of a data type and a class.

## Measurement Value

The value of a measure - e.g. 5 MM. Measurement values may be scalar or non-scalar.

## Numeric

The concept of a numeric value or set of numeric values that may be expressed in any way. This includes "structures" for representing non-scalar values.

## Primitive Value

A simple value with no parts that can be represented as a string.  
[FUML] PrimitiveType

# Tree

## Class Hierarchy

## Enumeration Hierarchy