Date: 22 August 2005

Ontology Definition Metamodel

Third Revised Submission to OMG/ RFP ad/2003-03-40

Submitted by:

IBM
Sandpiper Software, Inc.

Supported by:

Adaptive, Inc.
AT&T Government Solutions
Consultative Committee for Space Data Systems (CCSDS)
Data Access Technologies
David Frankel Consulting
DSTC Pty. Ltd.
Florida Institute for Human and Machine Cognition (IHMC)
Gentleware AG
Hewlett-Packard Company
Hyperion
IKAN Group
John Deere
Mercury Computer Systems
MetaMatrix
MetLife
No Magic
SAP Labs, LLC
Stanford University, Knowledge Systems Laboratory (KSL)
UMTP

ad/2005-08-01

Copyright © 2005 IBM
Copyright © 2005 Sandpiper Software, Inc.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

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

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

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED “AS IS” AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

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

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the

event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

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

Table of Contents

1 Scope. 21

2 Conformance. 22

3 Normative References. 23

4 Terms and Definitions 25

5 Symbols 28

6 Additional Information 29

6.1 Changes to Adopted OMG Specifications 29

6.2 How to Read This Specification 29

6.3 Contributors 29

6.4 Primary Contacts 30

6.5 Acknowledgements. 31

6.6 Resolution of RFP Mandatory Requirements. 31

6.7 Optional Requirements 32

6.8 Issues To Be Discussed 33

6.9 Evaluation Criteria 34

6.10 Proof of Concept. 34

7 Usage Scenarios and Goals. 35

7.1 Introduction. 35

7.2 Perspectives 35

7.2.1 Model-Centric Perspectives 36
7.2.2 Application-Centric Perspectives..37

7.3 Usage Scenarios 39

7.4 Business Applications. 41

7.4.1 Run Time Interoperation...41
7.4.2 Application Generation 42
7.4.3 Ontology Lifecycle 42

Table of Contents

7.5 Analytic Applications 43

7.5.1 Emergent Property Discovery 43
7.5.2 Exchange of Complex Data Sets 43

7.6 Engineering Applications 44

7.6.1 Information Systems Development 44
7.6.2 Ontology Engineering 44

7.7 Goals for Generic Ontologies and Tools 45

8 Design Rationale 47

8.1 Design Principles 47

8.2 Why Not Simply Use or Extend the UML 2.0 Metamodel? 47

8.3 Component Metamodel Selection 48

8.4 Relationships among Metamodels 49

8.4.1 The Need for Translation 49
8.4.2 UML Profiles 49
8.4.3 Mappings 50
8.4.4 Mappings Are Informative, Not Normative 50

8.5 Why Common Logic over OCL? 50

8.6 Why EMOF? 51

8.7 M1 Issues 51

9 ODM Overview 53

10 The UML2 Metamodel 55

10.1 Introduction 55

10.2 Features in Common (More or Less) 55

10.2.1 UML Kernel 55
10.2.2 Class and Property - Basics 57
10.2.3 More Advanced Concepts 62
10.2.4 Summary of More-or-Less Common Features 67

10.3 OWL but not UML 68

10.3.1 Predicate Definition Language 68
10.3.2 Names 69
10.3.3 Other OWL Developments 69

Table of Contents

10.4 In UML But Not OWL . . . 69

10.4.1 Behavioral and Related Features . . . 69
10.4.2 Complex Objects . . . 70
10.4.3 Access Control . . . 70
10.4.4 Keywords . . . 70

11 The RDF Schema Metamodel . . . 71

11.1 Overview . . . 71

11.1.1 Organization of the RDFS Metamodel . . . 71
11.1.2 Design Considerations . . . 71

11.2 The Classes and Utilities Diagrams . . . 72

11.2.1 PlainLiteral . . . 73
11.2.2 RDFSClass . . . 73
11.2.3 RDFSDatatype . . . 74
11.2.4 RDFSLiteral . . . 74
11.2.5 RDFSResource . . . 75
11.2.6 RDFXMLLiteral . . . 76
11.2.7 TypedLiteral . . . 76

11.3 The Properties Diagram . . . 77

11.3.1 RDFProperty . . . 77

11.4 The Containers Diagram . . . 78

11.4.1 RDFAlt . . . 78
11.4.2 RDFBag . . . 79
11.4.3 RDFSContainer . . . 79
11.4.4 RDFSContainerMembershipProperty . . . 80
11.4.5 RDFSeq . . . 80

11.5 The Collections Diagram . . . 81

11.5.1 RDFList . . . 81

11.6 The Reification Diagram . . . 81

11.6.1 RDFStatement . . . 82

11.7 The Ontology Diagram . . . 82

11.7.1 Ontology . . . 83

11.8 Language Mappings . . . 83

11.8.1 Classes and Utilities . . . 84
11.8.2 Properties . . . 84
11.8.3 Containers . . . 85
11.8.4 Collections . . . 85

Table of Contents

11.8.5 Reification 85
11.8.6 Ontology 86

12 The OWL Metamodel 87

12.1 Overview 87

12.1.1 Organization of the OWL Metamodel 87
12.1.2 Design Considerations 88

12.2 The Classes and Restrictions Diagrams 88

12.2.1 AllValuesFromRestriction 90
12.2.2 CardinalityRestriction 90
12.2.3 ComplementClass 91
12.2.4 EnumeratedClass 91
12.2.5 HasValueRestriction 92
12.2.6 IntersectionClass 92
12.2.7 MaxCardinalityRestriction 93
12.2.8 MinCardinalityRestriction 93
12.2.9 OWLClass 94
12.2.10 OWLRestriction 95
12.2.11 SomeValuesFromRestriction 95
12.2.12 UnionClass 96

12.3 The Properties Diagram 96

12.3.1 OWLDatatypeProperty 97
12.3.2 OWLObjectProperty 97
12.3.3 Property 98

12.4 The Individuals Diagram 99

12.4.1 DatatypeSlot 100
12.4.2 Individual 101
12.4.3 ObjectSlot 102
12.4.4 OWLAllDifferent 102

12.5 The Datatypes Diagram 103

12.5.1 OWLDataRange 103

12.6 The Utilities Diagram 103

12.6.1 OWLAnnotationProperty 104

12.7 The Ontology Diagram 104

12.7.1 OWLOntology 105
12.7.2 OWLOntologyProperty 106

12.8 Language Mappings 106

12.8.1 Classes and Restrictions 108

Table of Contents

12.8.2 Properties 110
12.8.3 Individuals 110
12.8.4 Datatypes 111
12.8.5 Utilities 111
12.8.6 Ontology 111

13 The Common Logic Metamodel 113

13.1 Overview 113

13.1.1 Design Considerations 113
13.1.2 Modeling Notes 114

13.2 The Phrases Diagram 114

13.2.1 Comment 114
13.2.2 ExclusionSet 115
13.2.3 Identifier 115
13.2.4 Importation 116
13.2.5 LogicalName 117
13.2.6 Module 118
13.2.7 Phrase 118
13.2.8 Sentence 119
13.2.9 Text 121

13.3 The Terms Diagram 122

13.3.1 CommentedTerm 122
13.3.2 FunctionalTerm 122
13.3.3 SequenceVariable 123
13.3.4 Term 124

13.4 The Atoms Diagram 125

13.4.1 Atom 125
13.4.2 AtomicSentence 126
13.4.3 Equation 126

13.5 The Sentences Diagram 127

13.5.1 Biconditional 127
13.5.2 CommentedSentence 128
13.5.3 Conjunction 128
13.5.4 Disjunction 129
13.5.5 ExistentialQuantification 129
13.5.6 Implication 130
13.5.7 IrregularSentence 130
13.5.8 Negation 131
13.5.9 QuantifiedSentence 131
13.5.10 UniversalQuantification 132

13.6 The Boolean Sentences Diagram 132

13.7 The Quantified Sentences Diagram 133

Table of Contents

13.8 Summary of CL Metamodel Elements with Interpretation 133

14 The ER Metamodel 135

14.1 Overview 135

14.1.1 Organization of the ER Metamodel 135

14.2 The Model Diagram 135

14.2.1 Model 136
14.2.2 ModelElement 137
14.2.3 Package 137
14.2.4 SubjectArea 138

14.3 The Domain Diagram 139

14.3.1 AtomicDomain 140
14.3.2 Domain 140
14.3.3 DomainConstraint 140
14.3.4 ListDomain 141
14.3.5 UnionDomain 142

14.4 The Entity Diagrams 142

14.4.1 Attribute 143
14.4.2 Entity 144
14.4.3 EntityConstraint 145
14.4.4 Generalization 145

14.5 The Relationship Diagram 146

14.5.1 Relationship 147
14.5.2 Role 148

14.6 The Key Diagram 149

14.6.1 AlternateKey 149
14.6.2 ForeignKey 150
14.6.3 InversionEntry 150
14.6.4 Key 150
14.6.5 PrimaryKey 151

14.7 The Instance Diagram 151

14.7.1 AttributeInstance 152
14.7.2 EntityInstance 153
14.7.3 Extent 153
14.7.4 Instance 154
14.7.5 RelationshipInstance 154
14.7.6 RoleInstance 155

14.8 The Inheritance Diagram 155

Table of Contents

14.8.1 NamedElement 156

14.9 Examples. 157

15 The Topic Map Metamodel 161

15.1 Topic Map Constructs. 161

15.1.1 TopicMapConstruct 162
15.1.2 TopicMap 162
15.1.3 MapItem 163
15.1.4 Topic 163
15.1.5 Association 165

15.2 Scope and Type. 165

15.2.1 Scope_able 166
15.2.2 Type_able 166

15.3 Characteristics. 167

15.3.1 Characteristic 167
15.3.2 AssociationRole 167
15.3.3 Occurrence 168
15.3.4 TopicName 169
15.3.5 VariantName 169

15.4 Published Subjects 170

15.5 Example 170

16 UML Profiles for RDF Schema and OWL 173

16.1 UML Profile for RDF Schema 173

16.1.1 RDF Document Syntax (Optional) 173
16.1.2 RDF Graph Model (Optional) 180
16.1.3 RDF Schema Profile Package 186
16.1.4 RDFS Ontology 186
16.1.5 RDF Document (optional) 187
16.1.6 Classes and Utilities 190
16.1.7 Properties in RDF/S 192
16.1.8 Containers and Collections 195
16.1.9 Reification 196
16.1.10 RDF Graphs and Nodes (optional) 197

16.2 UML Profile for OWL 197

16.2.1 OWL Profile Package 198
16.2.2 OWL Ontology Document 198
16.2.3 OWL Annotation Properties 199
16.2.4 OWL Ontology Properties 201
16.2.5 OWL Classes, Restrictions, and Class Axioms 205

Table of Contents

16.2.6 Properties....217
16.2.7 Individuals....222
16.2.8 Datatypes....225

17 The Topic Map Profile 227

17.1 Stereotypes 227

17.1.1 Topic Map....227
17.1.2 Topic....227
17.1.3 Association....228
17.1.4 Characteristics....228

17.2 Abstract Bases. 229

17.2.1 TopicMapElement....229
17.2.2 Scoped Element....229
17.2.3 TypedElement....230

17.3 Example 230

18 Mapping UML to OWL 233

18.1 Overview. 233

18.2 UML to OWL Mapping 233

18.2.1 Package....233
18.2.2 Class....234
18.2.3 Association....239
18.2.4 InstanceSpecification....240

18.3 OWL to UML Mapping 242

19 ER to OWL Mapping 243

19.1 Overview. 243

19.1.1 Representation of Source and Target Models....243
19.1.2 Representation of Mapping Specifications....243

19.2 ER to OWL Mapping 244

19.2.1 ER to OWL Mapping Summary....244
19.2.2 NamedElement....245
19.2.3 Model....245
19.2.4 Entity....246
19.2.5 Attribute....247
19.2.6 Relationship and Role....247

19.3 OWL to ER Mapping 248

Table of Contents

19.3.1 OWL to ER Mapping Summary 249
19.3.2 RDFSResource 249
19.3.3 OWLOntology 250
19.3.4 OWLClass 250
19.3.5 OWLRestriction 251
19.3.6 OWLDatatypeProperty 251
19.3.7 OWLObjectProperty 251
19.3.8 OWLDataRange 252

19.4 ER Abstract Syntax. 252

19.5 ODM OWL Abstract Syntax 254

20 Mapping Topic Maps to OWL 257

20.1 Overview. 257

20.2 Topic Maps to OWL Full Mapping 257

20.2.1 Overview 257
20.2.2 Basic Constructs 257
20.2.3 Property Restriction Patterns 261
20.2.4 Type Hierarchy Pattern 262
20.2.5 Naming Patterns 262
20.2.6 Instance Patterns 263

20.3 OWL to Topic Maps Mapping 265

20.3.1 Overview 265
20.3.2 Basic Constructs 265
20.3.3 Class Hierarchy 267
20.3.4 Labels 268
20.3.5 Instances 268
20.3.6 Example 269

21 Mapping RDFS and OWL to CL 271

21.1 Overview. 271

21.2 RDFS to CL Mapping. 271

21.2.1 RDF Triples 271
21.2.2 RDF Literals 272
21.2.3 RDF URIs and Graphs 272
21.2.4 RDF Lists 273
21.2.5 RDF Schema 273
21.2.6 RDFS Semantics 274

21.3 OWL to CL Mapping 277

Table of Contents

22 References (non-normative) 289

Appendix A Foundation Ontology (M1) for RDFS and OWL 291

Appendix B A Description Logic Metamodel 293

B.1 Introduction. 293

B.2 Containers 294

B.3 Concepts and Roles. 295

B.4 Datatypes 297

B.5 Collections 298

B.6 Expressions and Constructors. 299

B.7 Examples. 302

B.8 Overview Diagram 305

Appendix C Extending the ODM 307

C.1 Extendibility 307

C.2 Metaclass Taxonomy 307

C.3 Models of General Kinds of Application Domains 308

C.4 N-ary Associations 308

Appendix D Open Issues 311

Appendix E Mappings - Informative, Not Normative . . 313

List of Figures

Figure 1 ODM Metamodels: Structure and Mappings....54
Figure 2 Key Aspects of UML Class Diagram....55
Figure 3 Simple M1 Model....56
Figure 4 M1 Model with Association Class....61
Figure 5 Example N-ary Association with Multiplicity....64
Figure 6 The Classes Diagram of the RDF Schema Metamodel....72
Figure 7 The Utilities Diagram of the RDF Schema Metamodel....73
Figure 8 The Properties Diagram of the RDF Schema Metamodel....77
Figure 9 The Containers Diagram of the RDF Schema Metamodel....78
Figure 10 The Collections Diagram of the RDF Schema Metamodel....81
Figure 11 The Reification Diagram of the RDF Schema Metamodel....82
Figure 12 The Description Diagram of the RDF Schema Metamodel....83
Figure 13 The Classes Diagram of the OWL Metamodel....89
Figure 14 The Restrictions Diagram of the OWL Metamodel....90
Figure 15 The Properties Diagram of the OWL Metamodel....97
Figure 16 The Individuals Diagram of the OWL Metamodel....100
Figure 17 The Datatypes Diagram of the OWL Metamodel....103
Figure 18 The Utilities Diagram of the OWL Metamodel....104
Figure 19 The Ontology Diagram of the OWL Metamodel....105
Figure 20 Phrases....114
Figure 21 Valid Terms in CL....122
Figure 22 Atomic Sentences....125
Figure 23 Sentences....127
Figure 24 Boolean Sentences....132
Figure 25 Quantified Sentences....133
Figure 26 The Model Diagram of the ER Metamodel....136
Figure 27 The Domain Diagram of the ER Metamodel....139
Figure 28 The Entity Diagram of the ER Metamodel....143
Figure 29 The Relationship Diagram of the ER Metamodel....147
Figure 30 The Key Diagram of the ER Metamodel....149
Figure 31 The Instance Diagram of the ER Metamodel....152
Figure 32 The Inheritance Diagram of the ER Metamodel....156
Figure 33 The Model Example Diagram of the ER Metamodel....158
Figure 34 The Instance Example Diagram of the ER Metamodel....159
Figure 35 Primary Elements in the Topic Map Metamodel....161
Figure 36 The Topic Class....164
Figure 37 Topic Name Class....169
Figure 38 Instance of Topic Map Metamodel....171
Figure 39 RDF Document Definitions....175
Figure 40 RDF Graph, Node & Statement Definitions....181
Figure 41 RDF Schema Profile Package....186
Figure 42 RDFS Ontology Package....187
Figure 43 RDFDocument Provides an Alternate Container For RDFS Vocabularies and OWL Ontologies187
Figure 44 Property hasColor Without Specified Domain....192
Figure 45 Property hasColor Without Specified Domain, Alternate Notation....193

List of Figures

Figure 46 Property hasColor - Association Class Representation 193
Figure 47 Properties With Defined Domain, Undefined Range 193
Figure 48 Property Subsetting, Notation on Property Entry for Class..........................194
Figure 49 Property Subsetting, Unidirectional Association Notation...........................194
Figure 50 Property Subsetting, Association Class Notation 194
Figure 51 Web Ontology Language (OWL) Profile Package 198
Figure 52 Stereotype Notation for owl:versionInfo Applied to an Ontology or RDF Document 201
Figure 53 Stereotype Notation for owl:backwardCompatibleWith 203
Figure 54 Stereotype Notation for owl:imports 204
Figure 55 Stereotype Notation for owl:incompatibleWith...204
Figure 56 Stereotype Notation for owl:priorVersion 205
Figure 57 owl:Cardinality - Restricted Multiplicity in Subtype 208
Figure 58 owl:Cardinality - Restricted Multiplicity in Subtype 209
Figure 59 Simple Property Redefinition Example For owl:allValuesFrom..................210
Figure 60 Property Redefinition For owl:allValuesFrom With Unidirectional Associations 210
Figure 61 Property Redefinition For owl:allValuesFrom With Association Classes 210
Figure 62 Example Using owl:hasValue Constraint 212
Figure 63 Example Using owl:intersectionOf...213
Figure 64 Example Using owl:unionOf 214
Figure 65 Example Using owl:complementOf..214
Figure 66 Example Using owl:disjointWith..215
Figure 67 Example Using owl:disjointWith With Multiple Participants......................216
Figure 68 Example Using owl:disjointWith With Common Supertype 216
Figure 69 Example Using owl:equivalentClass 217
Figure 70 Using owl:inverseOf With Bidirectional Shorthand Notation......................221
Figure 71 Using owl:inverseOf Between Association Classes 221
Figure 72 Topic Map Stereotype...227
Figure 73 Topic Stereotype 227
Figure 74 Association Stereotype 228
Figure 75 Characteristic Stereotype 228
Figure 76 TopicMapElement Stereotypes...229
Figure 77 ScopedElement Stereotypes..229
Figure 78 TypedElement Stereotypes 230
Figure 79 Example Profile 231
Figure 80 Knowledge Representation System 293
Figure 81 Basic Containment Constructs...294
Figure 82 Element Model...295
Figure 83 DataType Model 297
Figure 84 Collection Model 298
Figure 85 Specialisations of Constructor 300
Figure 86 Example One..303
Figure 87 Example Two 304
Figure 88 Complete DL Metamodel 305
Figure 89 Countable/Bulk Package Extending OWL 307

List of Figures

Figure 90 Semantic Domain Model for isPartOf Property 308
Figure 91 Metaproperty Package for OWL..309

List of Figures

List of Tables

Table 1 Summary of Compliance Points 22
Table 2 Response to RFP Mandatory Requirements 31
Table 3 Response to RFP Optional Requirements 32
Table 4 Response to RFP Issues 33
Table 5 Response to RFP Evaluation Criteria 34
Table 6 Perspectives of Applications that Use Ontologies Considered in this Analysis....35
Table 7 Usage Scenario Perspective Values 40
Table 8 Summary of Requirements 45
Table 9 Properties and Types in Simple Model 56
Table 10 Classes and Owned Properties in Simple Model 57
Table 11 Implementation of Association in Simple Model 57
Table 12 Alternative Implementation of Association in Simple Model 57
Table 13 Example Course Instance 57
Table 14 Simple Model Classes Translated to OWL 58
Table 15 Simple Model Associations Translated to OWL 59
Table 16 Sample Associations Translated to OWL 60
Table 17 Sample Model Association Classes 62
Table 18 Common Features of UML and OWL 67
Table 19 OWL Features with No UML Equivalent 68
Table 20 Mapping Classes and Utilities 84
Table 21 Mapping Properties 84
Table 22 Mapping Containers 85
Table 23 Mapping Collections 85
Table 24 Mapping Reification 85
Table 25 Mapping Description 86
Table 26 Classes and Restrictions in the OWL Syntaxes 108
Table 27 Properties in the OWL Syntaxes 110
Table 28 Individuals in the OWL Syntaxes 110
Table 29 Datatypes in the OWL Syntaxes 111
Table 30 Utilities in the OWL Syntaxes 111
Table 31 Ontology in the OWL Syntaxes 111
Table 32 CL Metamodel Summary with Interpretation 133
Table 33 RDF Documents 188
Table 34 Classes and Utilities 190
Table 35 Properties 195
Table 36 Containers and Collections 195
Table 37 Reification (Basic Model) 196
Table 38 RDF Graphs and Nodes 197
Table 39 Annotation Properties 199
Table 40 Ontology Properties 202
Table 41 Class Descriptions 206
Table 42 Properties 218
Table 43 ER to OWL Mapping Summary 244
Table 44 OWL to ER Mapping Summary 249
Table 45 Equivalent Topic Map and OWL Constructs 269
Table 46 RDF Triple to CL Mapping 271

List of Tables

Table 47 Basic RDF to CL Mapping 272
Table 48 RDFS Triple to CL Mapping..274
Table 49 RDFS Extensional Logical Form Translation 275
Table 50 RDFS/OWL to CL Metamodel Translation 278
Table 51 Foundation Ontology (M1) for RDFS and OWL 291

1 Scope

IBM and Sandpiper Software are pleased to submit this response to the ADTF's RFP for an Ontology Definition Metamodel (ODM). We believe that this specification represents the foundation for an extremely important set of enabling capabilities for Model Driven Architecture (MDA) based software engineering, namely the formal grounding for representation, management, interoperability, and application of business semantics.

The ODM specification offers a number of benefits to potential users, including:

- Options in the level of expressivity, complexity, and form available for designing and implementing conceptual models, ranging from familiar UML and ER methodologies to formal ontologies represented in description logics or first order logic
- Grounding in formal logic, through standards-based, model-theoretic semantics for the knowledge representation languages supported, sufficient to enable reasoning engines to understand, validate, and apply ontologies developed using the ODM
- Profiles and mappings sufficient to support not only the exchange of models developed independently in various formalisms but to enable consistency checking and validation in ways that have not been feasible to date
- The basis for a family of specifications that marry MDA and Semantic Web technologies to support semantic web services, ontology and policy-based communications and interoperability, and declarative, policy-based applications in general

The specification defines a family of independent metamodels, related profiles, and mappings among the metamodels corresponding to several international standards for ontology and Topic Map definition, as well as capabilities supporting conventional modeling paradigms for capturing conceptual knowledge, such as entity-relationship modeling.

The ODM is applicable to knowledge representation, conceptual modeling, formal taxonomy development and ontology definition, and enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF. ODM-based ontologies can be used to support

- interchange of knowledge among heterogeneous computer systems
- representation of knowledge in ontologies and knowledge bases
- specification of expressions that are the input to or output from inference engines

The ODM is not intended to encompass

- specification of proof theory or inference rules
- specification of translation and transformations between the notations used by heterogeneous computer systems
- free logics
- conditional logics
- methods of providing relationships between symbols in the logical “universe” and individuals in the “real world”
- issues related to computability using the knowledge representation formalisms represented in the ODM (*e.g.*, optimization, efficiency, etc.)

2 Conformance

There are three compliance points distinguished for the Ontology Definition Metamodel:

[1] **No** compliance

[2] **Compliant** compliance: Implementation fully complies with the abstract syntax, well-formedness rules, semantics and notation of the package

[3] **Interchange** compliance. The tool has compliant compliance, and can exchange metamodel instances using XMI

Note: The mapping sections of the specification were made informative just prior to publication.

Table 1 Summary of Compliance Points

Compliance Point	Valid Options
RDFS Metamodel (Basic)	Compliant , Interchange
RDFS Metamodel (Extended, Document Model)	No, Compliant , Interchange
RDFS Metamodel (Extended, Document and Graph Model)	No, Compliant , Interchange
OWL Metamodel	Compliant , Interchange
CL Metamodel	No, Compliant , Interchange
ER Metamodel	No, Compliant , Interchange
Topic Maps Metamodel	No, Compliant , Interchange
UML Profile for RDFS and OWL (Basic)	No, Compliant , Interchange
UML Profile for RDFS and OWL (Document Model)	No, Compliant , Interchange
UML Profile for RDFS and OWL (Document and Graph Models)	No, Compliant , Interchange
UML Profile for Topic Maps	No, Compliant , Interchange
Mapping from UML to OWL	No, Compliant (unidirectional, bidirectional)
Mapping from ER to OWL	No, Compliant (unidirectional, bidirectional)
Mapping from Topic Maps to OWL	No, Compliant (unidirectional, bidirectional)
Mapping from RDFS and OWL to CL	No, Compliant

3 Normative References

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

[ISO 646] ISO/IEC 646:1991, Information technology -- ISO 7-bit coded character set for information interchange

[ISO 2382] ISO/IEC 2382-15:1999, Information technology -- Vocabulary -- Part 15: Programming languages

[ISO 10646] ISO/IEC 10646:2003, Information technology -- Universal Multiple-Octet Coded Character Set (UCS)

[ISO 14977] ISO/IEC 14977, Information technology -- Syntactic metalanguage -- Extended BNF

[ISO 24707] ISO/IEC CD 24707 Information technology – Common Logic (Common Logic) – A Framework for a Family of Logic-Based Languages, 2005-03-11. Latest version is available at http://cl.tamu.edu/docs/cl/24707-for-CD-Spring-2005.doc.

[MOF] MOF 2.0 Core. Final Adopted Specification, ptc/03-10-04. Latest version is available at http://www.omg.org/docs/ptc/03-10-04.pdf.

[MOF XMI] MOF 2.0 XMI (XML Metadata Interchange). Final Adopted Specification, ptc/04-06-11. Latest version is available at http://www.omg.org/docs/ptc/04-06-11.pdf.

[OCL] UML 2.0 OCL. Final Adopted Specification, ptc/03-10-14. Latest version is available at http://www.omg.org/docs/ptc/03-10-14.pdf.

[OWL S&AS] OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 February 2004, Peter F. Patel-Schneider, Patrick Hayes, Ian Horrocks, eds. Latest version is available at http://www.w3.org/TR/owl-semantics/.

[RDF Concepts] Resource Description Framework (RDF): Concepts and Abstract Syntax. Graham Klyne and Jeremy J. Carroll, Editors. W3C Recommendation, 10 February 2004. Latest version is available at http://www.w3.org/TR/rdf-concepts/.

[RDF MIME Type] *MIME Media Types*, The Internet Assigned Numbers Authority (IANA). This document is http://www.iana.org/assignments/media-types/. The registration for application/rdf+xml is archived at http://www.w3.org/2001/sw/RDFCore/mediatype-registration.

[RDF Primer] RDF Primer. Frank Manola and Eric Miller, Editors. W3C Recommendation, 10 February 2004. Latest version is available at http://www.w3.org/TR/rdf-primer/.

[RDF Schema] RDF Vocabulary Description Language 1.0: RDF Schema. Dan Brickley and R.V. Guha, Editors. W3C Recommendation, 10 February 2004. Latest version is available at http://www.w3.org/TR/rdf-schema/.

[RDF Semantics] RDF Semantics. Patrick Hayes, Editor, W3C Recommendation, 10 February 2004. Latest version available at http://www.w3.org/TR/rdf-mt/.

[RDF Syntax] RDF/XML Syntax Specification (Revised). Dave Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available at http://www.w3.org/TR/rdf-syntax-grammar/.

[RFC2396] IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998.

[RFC2732] *RFC 2732 - Format for Literal IPv6 Addresses in URL's*, R. Hinden, B. Carpenter and L. Masinter, IETF, December 1999. This document is http://www.isi.edu/in-notes/rfc2732.txt.

[TMDM] ISO/IEC FCD 13250-2: Topic Maps – Data Model, 2005-07-13. Latest version is available at http://www.isotopicmaps.org/sam/sam-model/.

[UML2] UML 2.0 Superstructure Specification. OMG Adopted Specification, ptc/2004-10-02. FTF Convenience Document is available at http://www.omg.org/cgi-bin/doc?ptc/2004-10-02.

[UML Infra] UML 2.0 Infrastructure Specification. OMG Final Adopted Specification, ptc/03-09-15. Latest version is available at http://www.omg.org/docs/ptc/03-09-15.pdf.

[Unicode] *The Unicode Standard, Version 3*, The Unicode Consortium, Addison-Wesley, 2000. ISBN 0-201-61633-5, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions/ for the latest version and additional information on versions of the standard and of the Unicode Character Database).

[XLINK] XML Linking Language (XLink) Version 1.0, W3C Recommendation 27 June 2001, http://www.w3.org/TR/xlink/.

[XML Schema Datatypes] XML Schema Part 2: Datatypes. W3C Recommendation 02 May 2000. Latest version is available at http://www.w3.org/TR/xmlschema-2/.

[XMLNS] Namespaces in XML; W3C Recommendation, 14 January 1999. Latest version is available at http://www.w3.org/TR/1999/REC-xml-names-19990114/.

[XTM] ISO/IEC 13250-3: Topic Maps – XML Syntax. Latest version is available at http://www.isotopicmaps.org/sam/sam-xtm/.

4 Terms and Definitions

Complete MOF (CMOF)
The CMOF, or Complete MOF, Model is the model used to specify other metamodels such as UML2. It is built from EMOF and the Core::Constructs of UML. The CMOF package does not define any classes of its own. Rather, it merges packages with its extensions that together define basic metamodeling capabilities.

Common Logic (CL)
Common Logic is a first order logic framework intended for information exchange and transmission. The framework allows for a variety of different syntactic forms, called dialects, all expressible within a common XML-based syntax and all sharing a single semantics.

Computation Independent Model (CIM)
A computation independent model is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model, and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification. Some ontologies are essentially CIMs from a software engineering perspective.

Description Logics (DL)
Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies, and typically represent a decidable subset of traditional first order logic. Description logic systems have been used for building a variety of applications including conceptual modeling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding. The Web Ontology Language (OWL) is a member of the description logics family of knowledge representation languages.

Entity-Relationship (ER)
An ER (entity-relationship) diagram is a graphical modeling notation that illustrates the interrelationships between entities in a domain. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

Essential MOF (EMOF)
Essential MOF is the subset of MOF that most closely corresponds to the facilities found in object-oriented programming languages and in XML. It provides a straightforward framework for mapping MOF models to implementations such as JMI and XMI for simple metamodels. A primary goal of EMOF is to allow simple metamodels to be defined using simple concepts while supporting extensions (by the usual class extension mechanism in MOF) for more sophisticated metamodeling using CMOF.

interpretation
A relationship between individuals in a universe of discourse and the symbols and relations in a model such that the model expresses truths about the individuals.

Knowledge Interchange Format (KIF)
Knowledge Interchange Format (KIF) is a computer-oriented language for the interchange of knowledge among disparate systems. It has declarative semantics (*i.e.* the meaning of expressions in the representation can be understood without appeal to an interpreter for manipulating those expressions); it is logically comprehensive (*i.e.* it provides for the expression of arbitrary sentences in the first-order predicate calculus); it provides for the representation of knowledge about the representation of knowledge; it provides for the representation of nonmonotonic reasoning rules; and it provides for the definition of objects, functions, and relations. KIF was developed in the late 1980s and early 1990s through support of the DARPA Knowledge Sharing Effort. There are several “flavors” of KIF in use today, including the best known versions: ANSI KIF (*i.e.*, Knowledge Interchange Format dpANS, NCITS.T2/98-004, http://logic.stanford.edu/kif/dpans.html) and KIF Reference (*i.e.*, Version 3.0 of the KIF Reference Manual, http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps). For the purpose of this ODM specification, references to KIF should be considered references to the KIF 3.0 Reference Manual cited in the Non-Normative References section of this specification.

Meta-Object Facility (MOF)
The Meta Object Facility (MOF), an adopted OMG standard, provides a metadata management framework, and a set of

metadata services to enable the development and interoperability of model and metadata driven systems. Examples of these systems that use MOF include modeling and development tools, data warehouse systems, metadata repositories etc. For the purpose of this ODM specification, references to MOF should be considered references to the Meta-Object Facility 2.0 Core Specification, cited in Normative References, above.

Object Constraint Language (OCL)
The Object Constraint Language (OCL), an adopted OMG standard, is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects; *i.e.* their evaluation cannot alter the state of the corresponding executing system. For the purpose of this ODM specification, references to OCL should be considered references to the UML 2.0 Object Constraint Language Specification, cited in Normative References, above.

Ontology Definition Metamodel (ODM)
The Ontology Definition Metamodel (ODM), as defined in this specification, is a family of MOF metamodels, mappings between those metamodels as well as mappings to and from UML, and a set of profiles that enable ontology modeling through the use of UML-based tools. The metamodels that comprise the ODM reflect the abstract syntax of several standard knowledge representation and conceptual modeling languages that have either been recently adopted by other international standards bodies (*e.g.*, RDF, RDF Schema, and OWL by the W3C), are in the process of being adopted (*e.g.*, Common Logic and Topic Maps by the ISO) or are considered industry de facto standards (ER).

Platform Independent Model (PIM)
A *platform independent model* is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. Examples of platforms range from virtual machines, to programming languages, to deployment platforms, to applications, depending on the perspective of the modeler and application being modeled.

Platform Specific Model (PSM)
A *platform specific model* is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Resource Description Framework (RDF)
The Resource Description Framework (RDF) is a framework for representing information in the Web. RDF has an abstract syntax that reflects a simple graph-based data model, and formal semantics with a rigorously defined notion of entailment providing a basis for well founded deductions in RDF data. The vocabulary is fully extensible, being based on URIs with optional fragment identifiers (URI references, or URIrefs). For the purpose of this ODM specification, references to RDF should be considered references to the set of RDF recommendations available from the World Wide Web Consortium, and in particular, the RDF Concepts and Abstract Syntax recommendation, cited in Normative References, above.

RDF Schema (RDFS)
RDF's vocabulary description language, RDF Schema, is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties. The RDF vocabulary description language class and property system is similar to the type systems of object-oriented programming languages such as Java. RDF differs from many such systems in that instead of defining a class in terms of the properties its instances may have, the RDF vocabulary description language describes properties in terms of the classes of resource to which they apply. For the purpose of this ODM specification, references to RDF Schema should be considered references to the set of RDF recommendations available from the World Wide Web Consortium, and in particular, the RDF Vocabulary Description Language 1.0: RDF Schema recommendation, cited in Normative References, above.

Topic Maps (TM)
Topic Maps provide a model and grammar for representing the structure of information resources used to define topics, and the associations (relationships) between topics. Names, resources, and relationships are said to be characteristics of abstract subjects, which are called topics. Topics have their characteristics within scopes: *i.e.* the limited contexts within which the names and resources are regarded as their name, resource, and relationship characteristics. One or more interrelated documents employing this grammar is called a “topic map.” For the purpose of this ODM specification,

references to Topic Maps should be considered references to the draft ISO standard cited in Normative References, above.

traditional first order logic
The traditional algebraic (or mathematical) formulations of logic generally described by Russell, Whitehead, Peano, and Pierce, dealing with quantification, negation, and logical relations as expressed in propositions that are strictly true or false. This specifically excludes reasoning over relations and excludes using the same name as both an individual name and a relation name.

Unified Modeling Language (UML)
The Unified Modeling Language, an adopted OMG standard, is a visual language for specifying, constructing and documenting the artifacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (*e.g.*, health, finance, telecommunications, aerospace) and implementation platforms (*e.g*., J2EE, .NET). For the purpose of this ODM specification, references to UML should be considered references to the Unified Modeling Language 2.0 Infrastructure and Superstructure Specifications, cited in Normative References, above.

universe of discourse
A non-empty set over which the quantifiers of a logic language are understood to range. Sometimes called a "domain of discourse".

Web Ontology Language (OWL)
The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL has three increasingly-expressive sub-languages: OWL Lite, OWL DL, and OWL Full. For the purpose of this ODM specification, references to OWL should be considered references to the set of OWL recommendations available from the World Wide Web Consortium, and in particular, the OWL Web Ontology Language Semantics and Abstract Syntax recommendation, cited in Normative References, above.

XML Metadata Interchange (XMI)
XMI is a widely used interchange format for sharing objects using XML. Sharing objects in XML is a comprehensive solution that build on sharing data with XML. XMI is applicable to a wide variety of objects: analysis (UML), software (Java, C++), components (EJB, IDL, CORBA Component Model), and databases (CWM). For the purpose of this ODM specification, references to XMI should be considered references to the XML Metadata Interchange (XMI) 2.0 Specification, cited in Normative References, above.

eXtended Markup Language (XML)
Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. RDF and OWL build on XML as a basis for representing business semantics on the Web. Relevant W3C recommendations are cited in the RDF and OWL documents as well as those cited under Normative References, above.

5 Symbols

CIM	Computation Independent Model
CL	Common Logic
DL	Description Logics
ER	Entity-Relationship
FOL	First Order Logic
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
KIF	Knowledge Interchange Format
MOF	Meta-Object Facility 2.0
OCL	UML 2.0 Object Constraint Language
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OWL	Web Ontology Language
PIM	Platform Independent Model
PSM	Platform Specific Model
RDF	Resource Description Framework
RDFS	RDF Schema
RFP	Request for Proposal
TFOL	Traditional First Order Logic
TM	Topic Maps
UML	Unified Modeling Language 2.0
XMI	XML Metadata Interchange
XML	eXtended Markup Language

6 Additional Information

6.1 Changes to Adopted OMG Specifications

No changes to UML 2.0 or other OMG specifications are required.

6.2 How to Read This Specification

This Chapter contains contact information and explains how the proposal addresses the RFP requirements.

Chapter 7 describes the usage scenarios and goals for the Ontology Definition Metamodel.

Chapter 8 explains the design rationale for the Ontology Definition Metamodel.

Chapter 9 explains the overall structure of the metamodels constituting the Ontology Definition Metamodel.

Chapter 10 demonstrates how the requirements of ontologies can be achieved through the re-use and extension of the UML 2.0 Kernel.

Chapter 11 describes the Resource Description Framework Schema metamodel.

Chapter 12 describes the Web Ontology Language metamodel.

Chapter 13 describes the Common Logic metamodel.

Chapter 14 describes the Entity-Relationship metamodel.

Chapter 15 describes the Topic Maps metamodel.

Chapter 16 describes the UML Profile for RDF Schema and OWL.

Chapter 17 describes the UML Profile for Topic Maps.

Chapter 18 describes the mapping between UML 2.0 and OWL.

Chapter 19 describes the mapping between ER and OWL.

Chapter 20 describes the mapping between Topic Maps and OWL.

Chapter 21 describes the mapping from RDFS and OWL to CL.

Chapter 22 contains non-normative references to other work.

Appendix A describes a Foundation ontology (M1) for RDFS and OWL

Appendix B describes the Description Logics metamodel (not normative).

Appendix C describes a methodology for extending the ODM.

Appendix D contains a short list of open issues.

6.3 Contributors

The following companies submitted this specification:

- IBM
- Sandpiper Software, Inc.

The following companies and organizations support this specification:

- Adaptive, Inc.
- AT&T Government Solutions
- Consultative Committee for Space Data Systems (CCSDS)
- Data Access Technologies
- David Frankel Consulting
- DSTC Pty Ltd.
- Florida Institute for Human and Machine Cognition (IHMC)
- Gentleware AG
- Hewlett-Packard Company
- Hyperion
- IKAN Group
- John Deere
- Mercury Computer Systems
- MetaMatrix
- MetLife
- No Magic
- SAP Labs, LLC
- Stanford University, Knowledge Systems Laboratory (KSL)
- UMTP

6.4 Primary Contacts

The primary contacts for this Ontology Definition Metamodel submission are:

Dr. Daniel T. Chang
IBM Silicon Valley Lab
555 Bailey Ave
San Jose, California
Phone: +1 408 463 2319
Email: dtchang@us.ibm.com

Elisa F. Kendall
Sandpiper Software, Inc.
2053 Grant Road #162
Los Altos, California 94024 USA
Phone: +1 650 960 2456
Fax: +1 650 969 6991
Email: ekendall@sandsoft.com

6.5 Acknowledgements

The submitters wish to gratefully acknowledge the contributions of Conrad Bock, Dr. Marko Boger, Jeremy Carroll, Dr. Robert M. Colomb, Mark Dutra, Patrick Emery, David Frankel, Lars Marius Garshol, Dr. Richard Goodwin, Lewis Hart, Dr. Patrick Hayes, Professor Hajime Horiuchi, Sridhar Iyengar, Dr. Philippe Martin, Dr. Deborah McGuinness, Jishnu Mukerji, Masaharu Obayashi, Masao Okabe, Dr. Yue Pan, Dr. Kerry Raymond, Dave Reynolds, Evan Wallace, Dr. Christopher Welty, Guo Tong Xie, and Dr. Yiming Ye in the preparation of this specification.

6.6 Resolution of RFP Mandatory Requirements

This section describes how this submission meets the mandatory and optional requirements identified in the RFP.

The ODM RFP calls for a Platform Independent Model (PIM) for ontology definition and at least one mapping to a Platform Specific Model (PSM) for OWL DL. The set of metamodels defined herein does not include a single generic metamodel for ontology modeling, given the variation in definitions of "ontology" (see discussions in Chapter 7, Usage Scenarios and Goals, and in Chapter 8, Design Rationale). The RFP also indicated that a PIM for ontology definition supporting knowledge representation languages that represent fragments of predicate logic was requested. The continuum from highly constrained description logics to a language as expressive as CL is fairly rich, in fact. Thus, we have developed:

- a Platform Independent Model (PIM) representing the abstract syntax of the Web Ontology Language (OWL), which actually consists of three distinct dialects. Each dialect can be considered a platform specific variant. The DL dialect of OWL is the most generally accepted standard representing the description logics family of languages.
- a Platform Independent Model (PIM) representing the abstract syntax of Common Logic (CL), which represents a family of knowledge representation languages that are first order in nature.
- a non-normative metamodel representing many of the features common to a family of knowledge representation languages called Description Logics (the DL in OWL DL), which we include as Appendix B for reference and educational purposes.

Constraints on the OWL Full metamodel that will restrict it to support OWL DL are in work and will be included during the finalization phase of this specification. A PSM for XCL, which is an XML surface syntax for CL, will also be included during the finalization phase of this specification. Others, such as PSMs for Conceptual Graphs [CGS] or the Knowledge Interchange Format [KIF], may be considered as resources permit. In addition, axioms for mapping both OWL Full and OWL DL to CL are included in Chapter 21, highlighting some of the distinctions between the two dialects of OWL considered important for the purposes of the ODM.

The following mandatory requirements are taken from Section 6.5 in the RFP.

Table 2 Response to RFP Mandatory Requirements

Requirement	Response
6.5.1. Ontology Definition Metamodel representing the semantics of ontologies including but not necessarily limited to OWL ontologies.	The metamodels for ontology definition specified herein support ontology development in the Web Ontology Language (OWL) and Common Logic (CL), which supports expressivity to the level of first order logic.
6.5.1.1. Depict using UML.	The ODM metamodels are depicted using constructs from UML.
6.5.1.2. Use v2.x of MOF, UML and OCL.	This submission is based on v2.0 of MOF, UML and OCL. However, prototyping of some capabilities has been limited by the lack of availability of tools conformant to these specifications. In practice, the ODM is based on the EMOF subset of MOF 2.0.

Table 2 Response to RFP Mandatory Requirements

6.5.2. UML2 profile	UML2 Profiles for ontology definition for RDF Schema, OWL, and Topic Maps are included.
6.5.3. Mapping between meta-model and profile	Two-way, bounded mappings from the appropriate metamodels to OWL, supporting forward and reverse engineering, are given in Chapter 18 through Chapter 20. A one-way mapping from RDF Schema and OWL to CL is provided in Chapter 21. The region covered by these mappings reflects the spectrum of knowledge representation formalisms identified in Chapter 7, Usage Scenarios and Goals as important for the purposes of this specification.
6.5.3.1. Support for forward and reverse engineering between ontologies	Mappings from the appropriate metamodels to OWL that are two-way and bounded, supporting forward and reverse engineering, are given in Chapter 18 through Chapter 20.
6.5.4 A mapping from ODM to OWL DL that is two-way and bounded.	The RFP called for a PSM for OWL DL and a mapping from the ODM metamodel to this PSM. This submission does not include a single ODM metamodel for ontology modeling. Rather, metamodels reflecting the abstract syntax of several ontology definition and conceptual modeling languages are included. Thus, a mapping from "ODM" to OWL DL doesn't necessarily make sense. Constraints that extend the metamodel for OWL that will enable vendors to restrict their implementations to OWL DL or OWL Full, if desired, will be provided in the next revision of the specification, and the mapping from UML to OWL provided in Chapter 18 will be augmented to show how a two-way, bounded mapping from UML to OWL DL is supported.
6.5.5 XMI XSD for ODM	XMI XSD for the set of normative ODM metamodels will be provided during finalization due to the lack of availability of tools conforming to the XMI2 specification.

6.7 Optional Requirements

This section describes how this submission meets the mandatory and optional requirements identified in the RFP. The following optional requirements are taken from Section 6.6 in the RFP.

Table 3 Response to RFP Optional Requirements

6.6.1. Mapping multiple ontologies into a single UML model	It is possible to map multiple ontologies into a single UML model, by coalescing those elements identified as representing the same concepts using the mapping facilities of Chapter 18.
6.6.2 Support for round-trip engineering	Discussed in Chapter 8.

Table 3 Response to RFP Optional Requirements

6.6.3 Mapping to DAML+OIL or other ontology languages	Mappings from the ODM to the language described by each metamodel are given in the chapter containing the relevant metamodel.
6.6.4 Support for OWL Full	The metamodels, mappings, and profiles provided herein support both OWL DL and OWL Full, including support for RDF Schema. In fact, limiting support to RDF Schema is possible, including mapping an RDF description (vocabulary, graph) to CL without reference to OWL.
6.6.5 Metadata to define context and scope of an ontology.	The ODM provides facilities for representing limited metadata for ontologies, such as defining the scope of an ontology (*i.e.*, limiting that scope to a particular RDF graph or set of graphs, to an OWL ontology, or to a CL module). Support for additional metadata may include OWL annotation properties or CL text, but support for specific metadata standards, such as Dublin Core, ISO 11179, or others is not explicitly provided. This level of metadata support could be implemented in ontologies, topic maps, or other conceptual models representing those standards that leverage ODM concepts, however.

6.8 Issues To Be Discussed

The following issues have been taken from Section 6.7 in the RFP.

Table 4 Response to RFP Issues

Strategy for mapping names between ODM and OWL	Discussed in Chapter 10.
Properties in OWL not supported in UML	Discussed in Chapter 10.
Iterative development across mapped environments	Round-trip transformation issues are discussed in Chapter 8.
Exclusive use of UML profiling	The submitters do not believe that the exclusive use of UML profiling is sufficient for the expressive power needed for ODM; rather, full MOF metamodels, and in some cases, profiles, are presented. Use of a full MOF metamodel enables the use of many MDA tools such as repositories, XMI generators, HUTN generators, and Query/View/Transformation tools.
Relationship to Business Nomenclature in CWM	The relationship between the Business Nomenclature in the Common Warehouse Metamodel and the ODM will be explored during finalization, in collaboration with the CWM2 working group.

6.9 Evaluation Criteria

The following Evaluation Criteria are taken from Section 6.8 of the RFP.

Table 5 Response to RFP Evaluation Criteria

Compactness and clarity	The model presented here has been selected to minimise the number of concepts and relationships within the bounds of achieving a desired level of expressive power. In modeling, there is always a trade-off between elaborating more concepts and relationships in a very precise way versus overloading concepts and using a smaller set in a less precise way (compensating with constraints or additional natural language semantics).
Extent of mapped regions	The mappings between ODM and OWL are provided in Chapter 18. Since the ODM includes OWL Full, the mapping is complete.
Expressiveness of ODM	The CL metamodel presented in this submission has an expressive power greater than that of OWL. Support for expression of types and statements across multiple ontologies enables the co-existence of divergent beliefs, but with support to detect and resolve such divergences.

6.10 Proof of Concept

DSTC Pty Ltd. is currently engaged in a seven year research programme into Enterprise Distributed Systems Technology with major projects devoted to knowledge representation. DSTC Pty Ltd. has extensive experience in the standardization, implementation and use of MOF, XMI and UML. The DSTC has been developing MOF-based tools since 1996. DSTC has developed the following prototypes to validate parts of this specification:

- Web-KB is a non-MOF-based implementation of many of the concepts represented in this specification. It is available for live demonstration on the Internet at www.webkb.org.
- Parts of the model presented in this specification has been implemented using DSTC's dMOF product (MOF 1.3) and DSTC's TokTok product (HUTN 1.0) to validate the expressive power of the model.

IBM has developed the following tools which in part validate portions of this specification:

- IBM Semantics Toolkit is a toolkit for storage, manipulation, query, and inference of ontologies and corresponding instances. It is available for download at http://alphaworks.ibm.com/tech/semanticstk.
- EODM is a tool for manipulation of and inference over OWL ontologies and RDF vocabularies, using EMF-based Java APIs generated from the OWL and RDFS metamodels. EODM is currently being proposed as an open-source Eclipse subproject (contact: dtchang@us.ibm.com).

Sandpiper Software has been developing technologies and tools to support UML-based knowledge representation since 1999. Sandpiper has developed the following products that validate parts of this specification:

- Visual Ontology Modeler (VOM) v1.5 is a UML 1.x/MOF 1.x compliant add-in to IBM Rational Rose, enabling component-based ontology modeling in UML with support for forward and reverse engineering of OWL ontologies.
- Next generation support for UML2, MOF2, and ODM compliance for RDFS/OWL and CL ontologies, and a CL constraint editor are under development, including migration to Eclipse/EMF, IBM Rational Software Architect (RSA) and IBM Rational Software Modeler (RSM), as well as integration with other UML2-compliant modeling environments such as No Magic's MagicDraw tool.

7 Usage Scenarios and Goals

7.1 Introduction

The usage scenarios presented herein highlight characteristics of ontologies that represent important design considerations for ontology-based applications. They also motivate some of the features and functions of the ODM and provide insight into when users can limit the expressivity of their ontologies to a description logics based approach, as well as when additional expressivity, for example from first order logic, might be needed. This set of examples is not intended to be exhaustive. Rather, the goal is to provide sufficiently broad coverage of the kinds of applications the ODM is intended to support so that ODM users can make informed decisions when choosing what parts of the ODM to implement to meet their development requirements and goals.

This analysis can be compared with a similar analysis performed by the W3C Web Ontology Working Group (W3C 2003). We believe that the six use cases and eight goals considered in W3C (2003) provide additional, and in some cases overlapping, examples, usage scenarios and goals for the ODM.

7.2 Perspectives

In order to ensure a relatively complete representation of usage scenarios and their associated example applications, we evaluated the coverage by using a set of perspectives that characterize the domain. Table 6 provides an overview of these perspectives.

Table 6 Perspectives of Applications that Use Ontologies Considered in this Analysis

Perspective	One Extreme	Other Extreme
Level of Authoritativeness	Least authoritative, broader, shallowly defined ontologies	Most authoritative, narrower, more deeply defined ontologies
Source of Structure	Passive (Transcendent) – structure originates outside the system	Active (Immanent) – structure emerges from data or application
Degree of Formality	Informal, or primarily taxonomic	Formal, having rigorously defined types, relations, and theories or axioms
Model Dynamics	Read-only, ontologies are static	Volatile, ontologies are fluid and changing.
Instance Dynamics	Read-only, resource instances are static	Volatile, resource instances change continuously
Control / Degree of Manageability	Externally focused, public (little or no control)	Internally focused, private (full control)
Application Changeability	Static (with periodic updates)	Dynamic
Coupling	Loosely-coupled	Tightly-coupled
Integration Focus	Information integration	Application integration
Lifecycle Usage	Design Time	Run Time

An ontology is a component of the design of a software system, produced as part of a software development process. The ontology itself is a specification of a conceptualization in some area. Different areas have different sorts of conceptualizations. They also differ in the costs and benefits associated with different specifications. The perspectives associated with the conceptualizations and the organization that produce the specifications are called *model centric*.

On the other hand, the ontology is used in the software development process in different ways. The perspectives that take account of how the ontology participates in the software development process are called *application centric*.

7.2.1 Model-Centric Perspectives

The model centric perspectives characterize the ontologies themselves and are concerned with the structure, formalism and dynamics of the ontologies, they are:

- Level of Authoritativeness
- Source of Structure
- Degree of Formality
- Model Dynamics
- Instance Dynamics

Each of these are discussed, in turn, below.

Level of Authoritativeness

The conceptualization from which an ontology is developed is always produced by someone. If the ontology is developed by the institution which is responsible for producing the conceptualization, then it is definitive, therefore *highly authoritative*. If the ontology is developed by an organization distant from the producing institution, it is generally *not very authoritative*.

Highly authoritative ontologies are part of the institutional environment of the organizations which will use them. If the conceptualization is complex, it often pays to develop the specification in great depth. But the authority of the responsible institution is limited, so the specification will generally have sharp boundaries, so will be relatively narrow. Ontologies which are not authoritative are often broad, since the creator can pick the most accessible concepts from many conceptualizations, but generally not very deep. The creator may not have access to the detail, or to the current definitive detail. The ontology cannot be relied upon by its users, so will generally not attract the resources to be developed in great detail.

SNOMED[1] is a very large and authoritative ontology. The periodic table of the elements is very authoritative, but small. However, it can be safely used as a component of larger ontologies in physics or chemistry. Ontologies used for demonstration or pedagogic purposes, like the Wine Ontology[2], are not very authoritative. Table 6 can be seen as an ontology which at present is not very authoritative. Should the classifications gain wide use in the ontology community, the ontology in Table 6 would become more authoritative.

Source of Structure

An ontology is a structure eventually implemented in software of some kind. In some cases, the structure is essentially the rules of the game. It is impossible to interoperate without using the published structure, and there are no actions which allow the structure to be changed. If the organization responsible for the ontology decides a change

---

1. http://www.snomed.org
2. http://www.w3.org/2001/sw/WebOnt/guide-src/wine.owl

is needed, it will be made as a software update and published as a new version of the ontology. An ontology that defines the rules of the game is called *transcendent*. SNOMED is a transcendent ontology defined by the various governing bodies of medicine. E-commerce exchanges are generally supported by transcendent ontologies.

On the other hand, the structure can be defined by patterns arising from the data produced by the interoperations, inferred using applications. This sort of ontology is called *immanent*. There are many applications where the ontologies are immanent. For example, consider a news feed. A useful structure to the users of the news feed is the topics a news item relates to. These topics arise from the news itself. Most changes in the structure of topics are fairly minor. A new person is elected as the head of government, but the government stays the same, for example. But when something new happens, the structure of the topics can change radically. The outbreak of a war can do this, as can the introduction of a new technology like the World-Wide Web or mobile telephones.

The applications extracting the structure from ongoing interoperations are generally statistical, often called some form of data mining. Besides news they are widely used in customer relationship management (think of the suggestions Amazon.com makes), search engines, and security applications. An example of the last is the detection of unusual patterns of credit card activity which may be indicators of fraudulent use.

Degree of Formality

Degree of formality refers to the level of formality of the specification of the conceptualization, ranging from *highly informal* or taxonomic in nature, where the ontologies may be tree-like, involving inheritance relations, to semantic networks, which may include complex subclass/superclass relations but no formal axiom expressions, to ontologies containing both subclass/superclass relations and *highly formal* axioms that explicitly define concepts. SNOMED is taxonomic, as is the Standard Industrial Classification system (SIC) used by the US Department of Labor Statistics, while engineering ontologies like Gruber and Olsen (1994) are highly formal.

Model Dynamics

All ontologies have structure. It is often necessary to change the structure. If the ontology is transcendent, the responsible organization may decide to make a change, while if the ontology is immanent, new patterns may arise in the interoperation data. The question is, how often is the structure changed? One extreme in the model dynamics dimension is *stable* or *read-only*. The ontology rarely changes its structure. The Periodic Table is very stable in its structure, as are generally the rules to any game. SNOMED is pretty stable, as is the SIC (the SIC is in process of being replaced by the North American Industry Classification System or NAICS, after 60 years, due to structural changes in the American economy in that period).

The other extreme in model dynamics is ontologies whose structure is *volatile*, changing often. An ontology supporting tax accounting in Australia would be volatile at the model level, since the system of taxation institutional facts can change, sometimes quite radically, with any Budget.

Instance Dynamics

An ontology is generally specified as a system of classes and properties (the structure) which is populated by instances (the extents). As with model dynamics, the instances in an ontology can be *stable* (*read-only) or* volatile. The Periodic Table is stable at the instance level (e.g. particular elements) as well as the model level (e.g. classes like noble gasses or rare earths). New elements are possible but rarely discovered. On the other hand, an ontology supporting an e-commerce exchange would be volatile at the instance level but possibly not at the model level. Z39.50 based applications are very stable in their model dynamics, but very volatile in their instance dynamics. Libraries are continually turning over their collections.

7.2.2 Application-Centric Perspectives

Application centric perspectives are concerned with how applications use and manipulate the ontologies, they are:

- Control / Degree of Manageability

- Application Changeability
- Coupling
- Integration Focus
- Lifecycle Usage

Control / Degree of Manageability

An ontology, like any piece of software, is subject to change. The issue in this dimension is who decides when and how much change to make. One extreme is when the body responsible for the ontology has sole decision on change (*internally focused*). The SIC is internally focused. Change is required because the structure of the US economy has changed over the years, but the Bureau of Labor Statistics makes the decision as to how to change and when the change is introduced.

The other extreme is when changes to the ontology are mandated by outside agencies (*externally focused*). In the US, ontologies in the finance industry were required to change by the Sarbanes-Oxley Act of 2002, and changes in ontologies in many areas were mandated by the Patriot Act, passed shortly after the World Trade Center attacks in 2001. An ontology on taxation matters managed by a trade association of accountants is subject to change as the relevant taxation acts are changed by governments.

Application Changeability

An ontology is eventually implemented in applications. The applications may be developed once, as for an e-commerce exchange (*static*). Of course there may be periodic updates. On the other extreme the applications may be constructed dynamically on the fly (*dynamic*), as in an application that composes web services at run time. In this case, the applications available to the end user come and go according to user requests.

Coupling

An ontology is generally implemented many times in many applications. The issue in this dimension is how closely coupled the applications are to each other. The applications in an e-commerce exchange are *tightly coupled* to each other, since they must interoperate at run time. At the other extreme, the applications using the Periodic Table or the Engineering Mathematics ontology may have nothing in common at run time. They are *loosely coupled*, solely because they share a component.

Integration Focus

Some ontologies specify the structure of interoperation but not the content. Z39.50 exclusive of the use attribute sets is a good example. The MPEG-21 multimedia framework is another example. It specifies the structure of multimedia objects without regard for their content. This extreme is called *application integration*, because they can be used to link programs together so that the output of one is a valid input for the other.

Other ontologies specify content. The ontology may be used to specify the structure of a shared database, so that the different players can exchange information about shared objects by withdrawing them from the shared database, updating them, and replacing them in the shared database. This extreme is called *information integration*.

An application can have both application and information focus.

Lifecycle Usage

An ontology is implemented in application software. In some cases, the application is used in the specification or design phases of the software life cycle and never again referred to explicitly. Use of the Periodic Table or Engineering Mathematics ontology in the specification of an engineering or scientific application is an example of

this extreme. The ontology is used at *design time*. In other cases, the ontology is used continually in the operation of the application. In a large e-commerce exchange, the exchange may check every message to see whether it conforms to the ontology and if so, what version. The message is then sent to the recipient with a certification, therefore relieving the players from having to do the checks themselves. In this case, the ontology is used at *run time*.

7.3 Usage Scenarios

As might be expected, some of these perspectives tend to correlate across different applications, forming application areas with similar characteristics. Our analysis, summarized in Table 7, has identified three major clusters of application types that share some set of perspective values:

- Business Applications are characterized by having transcendent source of structure, a high degree of formality and external control relative to nearly all users.
- Analytic Applications are characterized by highly changeable and flexible ontologies, using large collections of mostly read-only instance data.
- Engineering Applications are characterized by again having transcendent source of structure, but as opposed to business applications their users control them primarily internally and they are considered more authoritative.

Table 7 Usage Scenario Perspective Values

Use Case Clusters		Characteristic Perspective Values						
		Model Centric					Application Centric	
	Description	Authoritativeness	Structure	Formalit y	Model Dynamics	Instance Dynamics	Control	Change-ability
7.4	Business Applications		From Outside	Formal			External	
7.4.1	Run-time Interoperation	Least/Broad	From Outside	Formal	Read-Only	Volatile	External	Static
7.4.2	Application Generation	Most/Deep	From Outside	Formal	Read-Only	Read-Only	External	Static
7.4.3	Ontology Lifecycle	Middle/Broad& Deep	From Outside	Semi-Formal / Formal	Read-Only	Read-Only	External	Static
7.5	Analytic Applications				Volatile	Read-Only		Dynamic
7.5.1	Emergent Property Discovery	Broad & Deep	From Inside	Informal	Volatile	Read-Only	Internal & External	Dynamic
7.5.2	Exchange of Complex Data Sets	Broad & Deep	From Inside	Informal	Volatile	Read-Only/ Volatile	Internal & External	Dynamic
7.6	Engineering Application	Broad & Deep	From Outside				Internal	
7.6.1	Information System Development	Broad & Deep	From Outside	Semi-Formal / Formal	Read-Only	Volatile	Internal	Changeable
7.6.2	Ontology Engineering	Broad & Deep	From Outside	Semi-Formal / Formal	Volatile	Volatile	Internal	Changeable

7.4 Business Applications

7.4.1 Run Time Interoperation

Externally focused information interoperability applications are typically characterized by strong de-coupling of the components realizing the applications. They are focused specifically on information rather than application integration (and here we include some semantic web service applications, which may involve composition of vocabularies, services and processes but not necessarily APIs or database schemas). Because the community using them must agree upon the ontologies in advance, their application tends to be static in nature rather than dynamic.

Perspectives that drive characterization of these scenarios include:

- The ontology must be sufficiently authoritative to support the investment.
- Whether the control is external to the community members.
- Whether or not there is a design time component to ontology development and usage
- Whether or not the knowledge bases and information resources that implement the ontologies are modified at run time (since the source of structure remains relatively unchanged in these cases, or the ontologies are only changed in a highly controlled, limited manner).

These applications may require mediation middleware that leverages the ontologies and knowledge bases that implement them, potentially on either side of the firewall – in next generation web services and electronic commerce architectures as well as in other cross-organizational applications, for example:

- For semantically grounded information interoperability, supporting highly distributed, intra- and inter-organizational environments with dynamic participation of potential community members, (as when multiple emergency services organizations come together to address a specific crisis), with diverse and often conflicting organizational goals.
- For semantically grounded discovery and composition of information and computing resources, including Web services (applicable in business process integration and grid computing).

In electronic commerce exchange applications based on state-full protocols such as EDI or Z39.50, where there are multiple players taking roles performing acts by sending and receiving messages whose content refers to a common world.

In these cases, we envision a number of agents and/or applications interoperating with one another using fully specified ontologies. Support for query interoperation across multiple, heterogeneous databases is considered a part of this scenario.

While the requirements for ontologies to support these kinds of applications are extensive, key features include:

- the ability to represent situational concepts, such as player/actor – role – action – object – state,
- the necessity for multiple representations and/or views of the same concepts and relations, and
- separation of concerns, such as separating the vocabularies and semantics relevant to particular interfaces, protocols, processes, and services from the semantics of the domain.
- Service checking that messages commit to the ontology at run time. These communities can have thousands of autonomous players, so that no player can trust any other to send messages properly committed to the ontology.

7.4.2 Application Generation

A common worldview, universe of discourse, or domain is described by a set of ontologies, providing the context or situational environment required for use by some set of agents, services, and/or applications. These applications might be internally focused in very large organizations, such as within a specific hospital with multiple, loosely coupled clinics, but are more likely multi- or cross-organizational applications. Characteristics include:

- Authoritative environments, with tighter coupling between resources and applications than in cases that are less authoritative or involve broader domains, though likely on the “looser side” of the overall continuum.
- Ontologies shared among organizations are highly controlled from a standards perspective, but may be specialized by the individual organizations that use them within agreed parameters.
- The knowledge bases implementing the ontologies are likely to be dynamically modified, augmented at run time by new metadata, gathered or inferred by the applications using them.
- The ontologies themselves are likely to be deeper and narrower, with a high degree of formality in their definition, focused on the specific domain of interest or concepts and perspectives related to those domains.

For example:

- Dynamic regulatory compliance and policy administration applications for security, logistics, manufacturing, financial services, or other industries.
- Applications that support sharing clinical observation, test results, medical imagery, prescription and non-prescription drug information (with resolution support for interaction), relevant insurance coverage information, and so forth across clinical environments, enabling true continuity of patient care.

Requirements:

- The ontologies used by the applications may be fully specified where they interoperate with external organizations and components, but not necessarily fully specified where the interaction is internal.
- Conceptual knowledge representing priorities and precedence operations, time and temporal relevance, bulk domains where individuals don’t make sense, rich manufacturing processes, and other complex notions may be required, depending on the domain and application requirements.

7.4.3 Ontology Lifecycle

In this scenario we are concerned with activity, which has as its principle objectives conceptual knowledge analysis, capture, representation, and maintenance. Ontology repositories should be able to support rich ontologies suitable for use in knowledge-based applications, intelligent agents, and semantic web services. Examples include:

- Maintenance, storage and archiving of ontologies for legal, administrative and historical purposes,
- Test suite generation, and
- Audits and controllability analysis.

Ontological information will be included in a standard repository for management, storage and archiving. This may be to satisfy legal or operations requirements to maintain version histories.

These types of applications require that Knowledge Engineers interact with Subject Matter Experts to collect knowledge to be captured. UML models provide a visual representation of ontologies facilitating interaction. The existence of meta-data standards, such as XMI and ODM, will support the development of tools specifically for Quality Assurance Engineers and Repository Librarians.

Requirements implications:

- Full life-cycle support will be needed to provide managed and controlled progression from analysis, through design, implementation, test and deployment, continuing on through the supported systems maintenance period.
- Part of the lifecycle of ontologies must include collaboration with development teams and their tools, specifically in this case configuration and requirements management tools. Ideally, any ontology management tool will also be ontology aware.
- It will provide an inherent quality assurance capability by providing consistency checking and validation.
- It will also provide mappings and similarity analysis support to integrate multiple internal and external ontologies into a federated web.

7.5 Analytic Applications

7.5.1 Emergent Property Discovery

By this we mean applications that analyze, observe, learn from and evolve as a result of, or manage other applications and environments. The ontologies required to support such applications include ontologies that express properties of these external applications or the resources they use. The environments may or may not be authoritative; the ontologies they use may be specific to the application or may be standard or utility ontologies used by a broader community. The knowledge bases that implement the ontologies are likely to be dynamically augmented with metadata gathered as a part of the work performed by these applications. External information resources and applications are accessed in a read-only mode.

- Semantically grounded knowledge discovery and analysis (e.g., financial, market research, intelligence operations)
- Semantics assisted search of data stored in databases or content stored on the Web (e.g., using domain ontologies to assist database search, using linguistic ontologies to assist Web content search)
- Semantically assisted systems, network, and / or applications management.
- Conflict discovery and prediction in information resources for self-service and manned support operations (e.g., technology call center operations, clinical response centers, drug interaction)

What these have in common is that the ontology is typically not directly expressed in the data of interest, but represents theories about the processes generating the data or emergent properties of the data. Requirements include representation of the objects in the ontology as rules, predicates, queries or patterns in the underlying primary data.

7.5.2 Exchange of Complex Data Sets

Applications in this class are primarily interested in the exchange of complex (multi-media) data in scientific, engineering or other cooperative work. The ontologies are typically used to describe the often complex multimedia containers for data, but typically not the contents or interpretation of the data, which is often either at issue or proprietary to particular players. (The OMG standards development process is an example of this kind of application.)

Here the ontology functions more like a rich type system. It would often be combined with ontologies of other kinds (for example an ontology of radiological images might be linked to SNOMED for medical records and insurance reimbursement purposes).

Requirements include

- Representation of complex objects (aggregations of parts)
- Multiple inheritance where each semantic dimension or facet can have complex structure.

- Tools to assemble and disassemble complex sets of scientific and multi-media data.
- Facilities for mapping ontologies to create a cross reference. These do not need to be at the same level of granularity. For the purposes of information exchange, the lower levels of two ontologies may be mapped to a higher level common abstraction of a third, creating a sort of index.

7.6 Engineering Applications

The requirements for ontology development environments need to consider both externally and internally focused applications, as externally focused but authoritative environments may require collaborative ontology development.

7.6.1 Information Systems Development

The kinds of applications considered here are those that use ontologies and knowledge bases to support enterprise systems design and interoperation. They may include:

- methodology and tooling, where an application actually composes various components and/or creates software to implement a world that is described by one or more component ontologies.
- Semantic integration of heterogeneous data sources and applications (involving diverse types of data schema formats and structures, applicable in information integration, data warehousing and enterprise application integration).
- Application development for knowledge based systems, in general.

In the case of model-based applications, extent-descriptive predicates are needed to provide enough meta-information to exercise design options in the generated software (e.g., describing class size, probability of realization of optional classes). An example paradigm might reflect how an SQL query optimizer uses system catalog information to generate a query plan to satisfy the specification provided by an SQL query. Similar sorts of predicates are needed to represent quality-type meta-attributes in semantic web type applications (comprehensiveness, authoritativeness, currency).

7.6.2 Ontology Engineering

Applications in this class are intended for use by an information systems development team, for utilization in the development and exploitation of ontologies that make implicit design artifacts explicit, such as ontologies representing process or service vocabularies relevant to some set of components. Examples include:

- Tools for ontology analysis, visualization, and interface generation.
- Reverse engineering and design recovery applications.

The ontologies are used throughout the enterprise system development life cycle process to augment and enhance the target system as well as to support validation and maintenance. Such ontologies should be complementary to and augment other UML modeling artifacts developed as part of the enterprise software development process. Knowledge engineering requirements may include some ontology development for traditional domain, process, or service ontologies, but may also include:

- Generation of standard ontology descriptions (e.g., OWL) from UML models.
- Generation of UML models from standard ontology descriptions (e.g., OWL).
- Integration of standard ontology descriptions (e.g., OWL) with UML models.

Key requirements for ontology development environments supporting such activities include:

- Collaborative development

- Concurrent access and ontology sharing capabilities, including configuration management and version control of ontologies in conjunction with other software models and artifacts at the atomic level within a given ontology, including deprecated and deleted ontology elements
- Forward and reverse engineering of ontologies throughout all phases of the software development lifecycle
- Ease of use, with as much transparency with respect to the knowledge engineering details as possible from the user perspective
- Interoperation with other tools in the software development environment; integrated development environments
- Localization support
- Cross-language support (ontology languages as opposed to natural or software languages, such as generation of ontologies in the RDF(S)/OWL family of description logics languages, or in the Knowledge Interchange Format (KIF) where first or higher order logics are required)
- Support for ontology analysis, including deductive closure; ontology comparison, merging, alignment and transformation
- Support for import/reverse engineering of RDBMS schemas, XML schemas and other semi-structured resources as a basis for ontology development

7.7 Goals for Generic Ontologies and Tools

The diversity of the usage scenarios illustrates the wide applicability of ontologies within many domains. Table 8 brings these requirements together. To address all of these requirements would be an enormous task, beyond the capacity of the ODM development team. The team is therefore concentrating on the most widely applicable and most readily achievable goals. The resulting ODM will be not a final solution to the problem, but will be intended as a solid start which will be refined as experience accumulates.

Table 8 Summary of Requirements

Requirement	Section
Structural features	
Support ontologies expressed in existing description logic, (e.g. OWL/DL) and higher order logic languages (e.g. OWL Full and KIF), as well as emerging and new formalisms.	7.4.2 7.5.1 7.6.2
Represent complex objects as aggregations of parts	7.5.2
Multiple inheritance of complex types	7.5.2
Separation of concerns	7.4.1
Full or partial specification	7.4.2
Model-based architectures require extent-descriptive predicates to provide a description of a resource in an ontology, then generating a specific instantiation of that resource.	7.6.1
Efficient mechanisms will be needed to represent large numbers of similar classes or instances.	7.4.1
Generic content	
Support physical world concepts, including time, space, bulk or mass nouns like ‘water’, and things that do not have identifiable instances.	7.4.2

Table 8 Summary of Requirements

Support object concepts that have multiple facets of representations, e.g., conceptual versus representational classes.	7.4.1
Provide a basis for describing stateful representations, such as finite state automaton to support an autonomous agent's world representation.	7.4.1
Provide a basis for information systems process descriptions to support interoperability, including such concepts as player, role, action, and object.	7.4.1
Other generic concepts supporting particular kinds of domains	7.4.2
Run-time tools	
Tools to assemble and disassemble complex sets of scientific and multi-media data.	7.5.2
Service to check message commitment to ontology	7.4.1
Design-time tools	
Full life-cycle support	7.4.3 7.6.2
Support for collaborative teams	7.4.3 7.6.2
Ease of use, transparency with respect to details	7.6.2
Support for modules and version control.	7.4.3
Consistency checking and validation, deductive closure	7.4.3 7.6.2
Mappings and similarity analysis	7.4.3 7.5.2 7.6.2
Interoperation with other tools, forward and reverse engineering	7.6.2
Localization support	7.6.2

The table classifies the requirements into

- structural features – knowledge representation requirements
- generic content – aspects of the world common to many applications
- run-time tools – use of the ontology during interoperation
- design-time tools – needed for the design of ontologies

Associated with each requirement are the usage scenario from which it mainly arises.

8 Design Rationale

8.1 Design Principles

The ODM uses the design principles, such as modularity, layering, partitioning, extensibility and reuse, that are articulated in the UML Infrastructure document [UML Infra].

8.2 Why Not Simply Use or Extend the UML 2.0 Metamodel?

An ontology is a conceptual model, and shares characteristics with more traditional data models. The UML Class Diagram is a rich representation system, widely used, and well-supported with software tools. Why not simply use UML for representing ontologies?

OWL concepts, particularly those of OWL DL, represent an implementation of a subset of traditional first order logic called Description Logics (DL), and are largely focused on sets and mappings between sets in order to support efficient, automated inference. UML class diagrams are also based in set semantics, but these semantics are not as complete; additionally, in UML, not as much care is taken to ensure the semantics are followed sufficiently for the purposes of automatic inference. This can potentially be rectified with OCL, which is part of UML 2.0. The issues can be categorized by cases where UML is overly restrictive, not restrictive enough, or simply doesn't provide the explicit construct necessary. For example:

- UML disjointedness requires disjoint classes to have a common super-type, which is not the case in OWL (aside from the fact that all OWL classes are ultimately subclasses of owl:Thing, and similarly that all classes in RDF Schema are resources).
- To model set intersection in UML one might consider using multiple inheritance, but this still allows an instance of both super-classes to be omitted from the subclass, which is not permitted in OWL.
- There is no UML construct for set complement.

The lack of reliable set semantics and model theory for UML prevents the use of automated reasoners on UML models. Such a capability is important to applying Model Drive Architecture to systems integration. A reasoner can automatically determine if two models are compatible, assuming they have a rigorous semantics and axioms are defined to relate concepts in the various systems.

Another distinction is in the ability to fully specify individuals apart from classes, and for individuals to have properties independently of any class they might be an instance of in OWL. In this regard, UML shows it software heritage, in which it is not possible for an instance to exist without a class to define its structure, a characteristic that derives from classes used as abstractions of memory layout. It is not hard to work around this using singleton classes as proposed in the profile, but for methodologies that start with instances and derive classes from them, this is clutter obviously introduced from a practice in which the reverse is the norm.

In OWL Full, it is also common to reify individuals as classes. OWL Full allows classes to have instances which are themselves classes or properties; classes and properties can be the domains of other properties. Elements of an ontology frequently cross meta-levels, and may represent the equivalent of multiple meta-levels depending on the domain, application, usage model, and so forth. Ontologists frequently want to see a combination of these classes and individuals on the same diagram, and find it unnatural if they cannot. Many software languages reify classes, but UML has been only half-hearted in supporting this mechanism. One can also work around this, however, as shown in the profile. The four-layer meta level architecture that UML resides in does not restrict class reification, even though it is often confused with reification. Classes and instances can reside on a single level of the architecture, at least if UML is used to describe that layer.

Additionally, while some claim that UML would need to support properties independently of classes to be used in the OWL style, this is not actually the case. In fact, independent properties in OWL are semantically equivalent to properties on owl:Thing, which is directly translatable to UML using a model library, corresponding to the one proposed in the Foundation Ontology given in Appendix A. OWL does not require the use of owl:Thing for properties without defined domains, but this is really just syntactic sugar. Note that the same is true when RDF vocabularies are developed without using any OWL constructs; for the purposes of this specification, the model library should be used in either case.

The above problems could potentially be addressed in a revision of UML. However, the RFP to which this submission is responding did not call for that.

8.3 Component Metamodel Selection

A trigger for the call for development of an ODM was the development by the World-Wide Web Consortium of a set of languages that form the foundation of the Semantic Web, including the Resource Description Framework (RDF), RDF Schema, and the Web Ontology Language (OWL). In addition, there have been many other ontology language development efforts, including International Standards Organization (ISO) projects for Topic Maps and Common Logic (CL). Topic Maps is a metalanguage designed to express the “aboutness” of an information structure with key model elements *topic* and *association*. Common Logic represents a family of knowledge representation languages. Common Logic, or CL, is a first order logic, analogous to predicate calculus, and is the successor to KIF (Knowledge Interchange Format). Both Topic Maps and CL have XML serializations, and were designed to express semantics for knowledge exchanged over the World Wide Web. These languages overlap with some parts of OWL as might be expected, but are used for different purposes and have different or no requirements for automated reasoning. CL is more expressive than OWL, and is better suited to applications involving declarative representation of rules and formulas, for example.

As an initial part of the ODM development process, the team determined that understanding the requirements for ontology development using ODM metamodels was essential to establishing the ODM architecture and selecting an appropriate set of languages to be incorporated in the specification. The results of this requirements analysis are summarized in Chapter 7, Usage Scenarios and Goals. The set of languages represented, the architecture, and potential extensions currently envisioned developed as a direct consequence of this effort. This includes the notion that organizations developing ontologies may need to leverage pre-existing data and process models represented in UML, Entity-Relationship (ER), or another modeling language, even if the development effort itself is conducted using an ODM metamodel. For some possible extensions to better support some classes of ontologies, see Appendix C, Extending the ODM.

A significant exception is immanent ontologies, whose structure is derived from the information being exchanged as distinguished from transcendent ontologies, whose structure is provided a priori by schemas and the like. News feeds, results of data mining, and intelligence applications are examples of immanent ontologies, while e-commerce exchanges, engineering applications, and controlled vocabularies generally are transcendent. Immanent ontologies are represented by at least collections of terms, but often also by some numeric representation of the relationship among terms: co-occurrence matrices, conditional probabilities of co-occurrence, and eigenvectors of co-occurrence matrices, for example. These kinds of applications have not attracted the development of standardized representation structures as have transcendent ontologies. The ODM team considered that it was outside the scope of the RFP to innovate in areas such as immanent ontology development without existing standard representations.

8.4 Relationships among Metamodels

8.4.1 The Need for Translation

The various metamodels in the ODM are treated equally, in that they are generally independent of each other. It is not necessary to understand or be aware of the others to understand any one in particular. The one exception to this is that the metamodel for OWL extends the metamodel for RDF/S, as the OWL language itself extends the RDF/S language.

However, in an ontology development project it might be necessary to use several of the metamodels, and to represent a given fragment of an ontology component in more than one. For example, consider a large e-commerce exchange project. The developers might choose to represent the ontology specifying the shared world governing the exchange in OWL. But the exchange might have evolved from a single large company's electronic procurement system (as was the case for example with the General Electric Global Exchange Service [GE]). The original procurement system might have been designed using UML, so that it would be a significant saving in development cost to be able to translate the UML specification to OWL as a starting point for development of the ontology.

Once such an exchange is operating, it may have thousands of members, each of which will have its own information system performing a variety of tasks in addition to interoperating through the exchange. These systems are all autonomous, and the exchange has no interest in how they generate and interpret the messages they use to interoperate so long as they commit to the ontology. Let us assume that the various members have systems with data models in UML or dialects of the ER model. A given member will need to subscribe to at least a fragment of the ontology and make sure its internal data model conforms to the fragment. It would therefore be an advantage to be able to translate a fragment of the ontology to UML or ER to facilitate the member making any changes to its internal operations necessary for it to commit to the ontology. Alternatively, a member might have a large investment in UML and would like the development to leverage UML experience and UML tools to make at least a first approximation to alignment with the OWL model.

It is extremely important for those leveraging existing artifacts for ontology development to understand that "what makes a good object-oriented software component model" does not necessarily make a good ontology. Once a particular UML or ER model has been translated to RDFS/OWL, for example, care needs to be taken to ensure that the resultant model will result in the desired assertions in a knowledge base. Significant restructuring is often required, in other words.

The ODM therefore needs to provide facilities for making relationships among instances of its metamodels, including UML. There are two ways to accomplish this: UML profiles and mappings.

8.4.2 UML Profiles

The goal of a UML profile from the ODM perspective is to provide a bridge between the UML and knowledge representation communities on a well-grounded, semantic basis, with a broader goal of relating software and logical approaches to representing information. Profiles facilitate implementation using common notation on existing UML tools. They support renaming and specializing UML model elements in consistent ways, so that an instance of a UML model can be seen as an extended metamodel. Profiles allow a developer to leverage UML experience and tools while moving to integrating with an ontology represented in another metamodel.

We have provided such profiles for the Topic Maps, RDFS and OWL metamodels, as one of the primary goals that emerged from our use case development work was to enable use of existing UML tools for ontology modeling. The profiles provided in Chapter 16, UML Profiles for RDF Schema and OWL, and in Chapter 17, The Topic Map Profile, were designed specifically for use in UML 2.0 tools. A profile for Common Logic is under consideration as an extension to this specification, as potential applications for its use in business semantics and production rules applications were identified late in the specification development process.

8.4.3 Mappings

Working with multiple metamodels will often require a model element by model element translation of model instances from one metamodel to another. We have seen that UML profiling has limited capability in representing ODM metamodels in UML. We therefore need to specify mappings from one metamodel to another.

There is a parallel RFP in the OMG called QVT (Query/View/Transform) which will provide a standardized MOF-based platform for mapping instances of MOF metamodels from one metamodel to another [QVT]. Although the QVT specification is not yet final, it appears to be sufficiently mature that we have used it to define the mappings in the ODM.

Translation between metamodels has the fundamental problem that there may not be a single and separate model element in the target corresponding to each model element in the source (indeed, if the metamodels are not simply syntactic variations, this would be the normal situation). We will call this situation *structure loss*. Some of the issues involved with structure loss and what to do about it using one of the earlier QVT proposals are discussed in [MSDW].

An overview of the mapping strategy used in the ODM is illustrated in Chapter 9. Note that there are mappings from each metamodel to and from OWL Full, except for Common Logic (CL) for which there is only a mapping from OWL Full. A lossy, reverse mapping defined in QVT from CL to OWL, and bi-directional mappings between UML and CL are planned, and will be added either during finalization or through an RFP/RFC process.

8.4.4 Mappings Are Informative, Not Normative

The RFP (Section 6.2) calls for a language mapping from the ODM to OWL. In chapter 9, The ODM is shown as having metamodels for several languages (RDFS/OWL, Topic Maps, Common Logic and Entity-Relationship Models) tied together by mappings to and from OWL (including UML to and from OWL). Common Logic is the exception, with mappings from OWL to CL only.

An argument for the infeasibility of normative mappings is presented in Appendix M. In a nutshell, the mappings the ODM can provide are very general. Due to the very different scope and structure of the systems metamodeled, mappings based solely on the general structure of the languages will often lead to less than ideal choices for mapping some structures. Any particular mapping project will have additional constraints arising from the structure of the particular models to be mapped and the purposes of the project, so will very likely make different mapping choices than those in the ODM. An industry of consultants will likely arise, adding value by exactly this activity. They can use the ODM mappings as a takeoff point, and as an aid to understanding the comparative model structure, so the ODM mappings have value as informative, but not as normative.

8.5 Why Common Logic over OCL?

Common Logic (CL) is qualitatively different from some of the other metamodels in that it represents a dialect of traditional first order logic, rather than a modeling language. UML already supports a constraint (rule) language, which includes similar logic features, OCL [OCL], so why not use it?

The short answer to that question is that the ODM does include OCL in the same way it includes UML. Unfortunately, just as UML lacks a formal model theoretic semantics, OCL also has neither a formal model theory nor a formal proof theory, and thus cannot be used for automated reasoning (today). Common Logic, on the other hand, has both, and therefore can be used either as an expression language for ontology definition or as an ontology development language in its own right.

CL represents work that has been ongoing in the knowledge representation and logic community for a number of years. It is a second-generation language intended to have an extremely concise kernel for efficient reasoning, has a surface syntax for use with Semantic Web applications, and is rooted in the Knowledge Interchange Format (KIF Reference Manual v3.0 was published in 1992) as well as in other knowledge representation formalisms. It is also a committee draft standard (24707) in JTC 1 / SC32 of the ISO/IEC standards community.

Our original work with regard to the metamodel was done with active participation of the CL language authors, and sought to be true to the abstract syntax of the CL language to the extent possible. Our intent was to enable ontologies developed using the ODM to be operated on by DL and CL reasoners downstream. There are a number of such reasoners available today, including FaCT, Racer, Cerebra, and others from the DL community, as well as KIF-based reasoners such as Stanford's Java Theorem Prover (JTP), OntologyWorks, and so forth, which ODM users can leverage for model consistency checking, model validation, and for applications development.

Finally, given that the ODM includes mappings among the metamodels for the modeling languages, why not include mappings between OCL and CL? Such a mapping should in principle be possible, but both languages are very rich. A mapping between them must deal with concerns about issues related to unintended semantics, the ability to write complex expressions involving multiple variables that preserve quantifier scope, and so forth. These issues are very important from a reasoning perspective, and thus our approach needs to be well developed and tested using both OCL and CL reasoners if we are to go down that path. This represents a longer term activity that may be taken up in the Ontology PSIG if there is sufficient commercial interest in doing so.

8.6 Why EMOF?

The RFP called for a MOF 2 metamodel for the ODM. MOF 2 has two flavors, EMOF (Essential MOF) and CMOF (Complete MOF), with EMOF being equivalent to a subset of CMOF. We have used EMOF for the ODM for two reasons:

- The advantage of using EMOF is that the modeling tools available during ODM development, such as IBM Rational Rose, support EMOF (or close to it) but not CMOF. It was therefore possible to use such tools to define ODM metamodels. At present, the newness of CMOF means that CMOF facilities are not supported by most tools. Therefore use of CMOF facilities imposes a significant burden.
- The ODM metamodels can be represented in EMOF without sacrificing major syntactic or semantic considerations.

On the other hand, some of the possible extensions discussed in Appendix C do require CMOF facilities. Use of EMOF in the development of the ODM does not preclude extensions to CMOF as might be advantageous, and as the tools evolve to support it.

8.7 M1 Issues

The ODM team encountered some issues in developing MOF-based metamodels for the W3C languages RDF, RDFS and OWL, and to a lesser extent the ISO language Topic Maps. A MOF-based metamodel has a strict separation of meta-levels. The number and designation of meta-levels is changed in MOF2 from MOF 1.4, but the issue can be described in the MOF1.4 designations:

- M3 – the MOF
- M2 – a MOF class model, specifying the classes and associations of the system being modeled, the structure of OWL for example.
- M1 – an instance of an M2 model, describing a particular instance of the system being modeled, a particular OWL ontology, for example.
- M0 – ground individuals. A population of instances of the classes in a particular OWL ontology, for example.

RDFS and OWL are defined as specializations of RDF. RDF has natively a very simple model. There are resources and properties. The entire structure of RDF, RDFS and OWL is defined in terms of instances of resources, properties, and other structures like classes, which are defined in terms of built-in resources and properties. In fact, even property is formally defined as an instance of resource, and resource (the set of resources) is itself an instance of resource. These languages are self-referential in a way that a native MOF metamodel could never be.

The same is true to a lesser degree of Topic Maps. Although the ISO standard provides a Topic Map Data Model, some important constructs like class and subclass are defined as published subjects, which are instances of topics. Topics are defined at the M2 level, so published subjects are M1 objects.

The Topic Maps metamodel in the ODM deals with the M1 problem by having an M2 structure following the published Topic Map Data Model, with a note detailing the built-in M1 published subjects, but this approach does not suit the W3C languages. In the ODM we have modeled RDF, RDF Schema, and OWL at the M2 level, following the published abstract syntax for them. Certain built-in RDF/S and OWL constructs have relevance at multiple MOF meta-levels. Some of these, such as annotation properties including rdfs:seeAlso, are included as M2 elements in the RDFS Metamodel; others, such as ontology properties including owl:priorVersion, are included as M2 elements in the OWL Metamodel.

Some important constructs, however, are not appropriate to model at all at the M2 level. These are provided in an ontology as an M1 model library (given in Appendix A, Foundation Ontology (M1) for RDFS and OWL), and include:

- Two built-in classes - owl:Thing and owl:Nothing
- The built-in empty list - rdf:Nil
- The set of XML Schema datatypes that are supported in RDF/S and OWL - xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte and xsd:positiveInteger

9 ODM Overview

As introduced briefly in the RFP [ODM RFP], ontology is a discipline rooted in philosophy and formal logic, introduced by the Artificial Intelligence community in the early to mid-80s to describe real world concepts that are independent of specific applications. Over the past two decades, knowledge representation methodologies and technologies have subsequently been used in other branches of computing where there is a need to represent and share contextual knowledge independently of applications.

The following definition was adopted from the RFP:

> An ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge. An ontology can range in expressivity from a Taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a Thesaurus (words and synonyms), to a Conceptual Model (with more complex knowledge), to a Logical Theory (with very rich, complex, consistent and meaningful knowledge).

This definition, and the analysis presented in Chapter 7, led to the determination that the ODM would ultimately include six metamodels (five that are normative, and one that is informative). These are grouped logically together according to the nature of the representation formalism that each represents: formal first order and description logics, structural and subsumption / descriptive representations, and traditional conceptual or object-oriented software modeling.

At the core are two metamodels that represent formal logic languages: DL (Description Logics, which, although it is non-normative, is included as informative for those unfamiliar with description logics, [BCMNP]) and CL (Common Logic, [ISO 24707]), a declarative first-order predicate language. While the heritage of these languages is distinct, together they cover a broad range of representations that lie on a continuum ranging from higher order, modal, probabilistic and intentional representations to very simple taxonomic expression.

There are three metamodels that represent more structural or descriptive representations that are somewhat less expressive in nature than CL and some DLs. These include metamodels of the abstract syntax for RDFS [RDF Schema], OWL [OWL Reference; OWL S&AS], and TM (Topic Maps, [TMDM]). RDFS, OWL and TM are commonly used in the semantic web community for describing vocabularies, ontologies and topics, respectively.

Two additional metamodels considered essential to the ODM represent more traditional, software engineering approaches to conceptual modeling: UML2 [UML2, UML Infra] and ER (Entity Relationship) diagramming. UML and ER methodologies are the two most widely used modeling languages in software engineering today, particularly for conceptual or logical modeling. Interoperability with and use of intellectual capital developed in these languages as a basis for ontology development and further refinement is a key goal of the ODM. Since UML2 is an adopted OMG standard, we simply reference it in the ODM.

Three UML profiles have been identified for use with the ODM: UML4RDFS, UML4OWL and UML4TM. These enable the use of UML notation (and tools) for ontology modeling and facilitate generation of corresponding ontology descriptions in RDFS, OWL and TM, respectively.

In addition, in order to support the use of legacy models as a starting point for ontology development, and to enable ODM users to make design trade-offs in expressivity based on application requirements, mappings among a number of the metamodels are provided. As discussed in Section 8.4.3, these mappings are expressed in the MOF QVT Relations Language. To avoid an n-squared set of mappings, the ODM includes direct mappings to and from OWL for UML, ER and Topic

Maps.

CL is an exception to this strategy. CL is much more expressive than the other metamodels, and is therefore much more difficult to map into the other metamodels. CL can be used to define constraints and predicates that cannot be expressed (or are difficult to express) in the other metamodels. Some predicates might be specified in a primary metamodel, for example, in OWL, and refined or further constrained in CL. The relevant elements of the M1 model expressed in the primary metamodel will be mapped into CL. Thus, uni-directional mappings (to CL), only, are included or planned at this time.

Figure 1 shows the organization of the metamodels, with the current and intended mapping components indicated, with RDFS and OWL grouped, as shown, for mapping purposes.

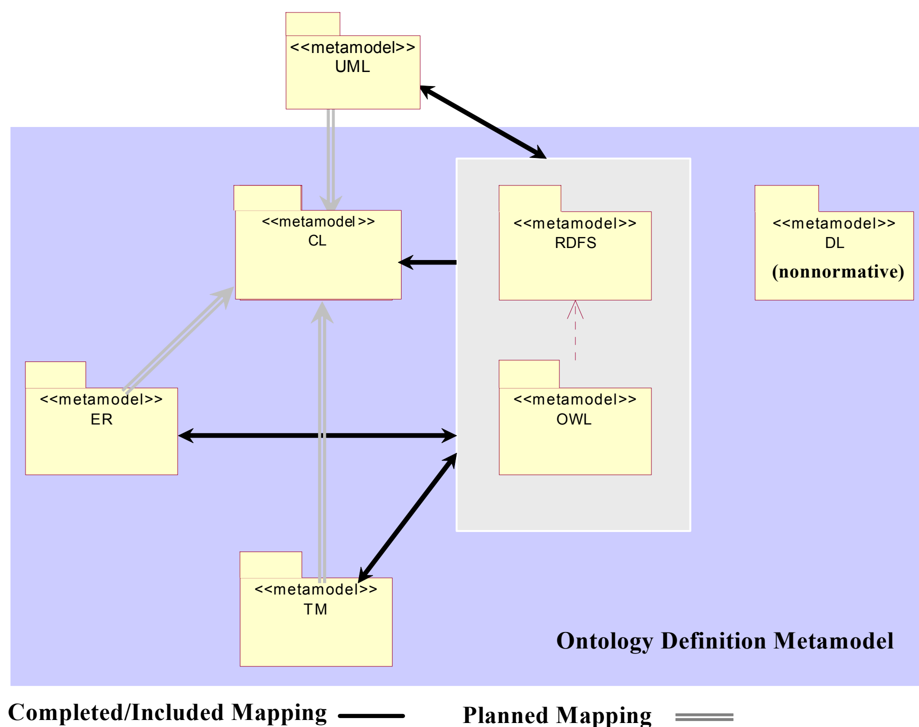


Figure 1 ODM Metamodels: Structure and Mappings

10 The UML2 Metamodel

10.1 Introduction

This chapter is intended to show how UML compares with the mandated ontology representation language OWL, partly to motivate the development of the ODM as opposed to a blanket recommendation that people use UML for ontology representation. It compares the features of OWL Full (as summarized in OWL Web Ontology Language Overview [OWL OV]) with the features of UML 2.0 [UML2]. It first looks at the features the two have in common, although sometimes represented differently, then the features in one but not the other. Little attempt is made to distinguish the features of OWL Lite or OWL DL from those of OWL Full. This overview ignores secondary features such as headers, comments and version control. In the features in common, a sketch is given of the translation from a model expressed in UML to an OWL expression. In several cases, there are alternative ways to translate UML constructs to OWL constructs. This chapter selects a particular way in each case, but the translation is not intended to be normative. In particular applications other choices may be preferable.

The possible translation of OWL to UML is not considered in this chapter, but is covered in Chapter 18.

UML models are organized in a series of metalevels: M3, M2, M1 and M0, as follows:

- M3 is the MOF, the universal modeling language in which modeling systems are specified.
- M2 is the model of a particular modeling system. The UML metamodel is an M2 construct, as it is specified in the M3 MOF.
- M1 is the model of a particular application represented in a particular modeling system. The UML Class diagram model of an order entry system is an M1 construct expressed in the M2 metamodel for the UML Class diagram.
- M0 is the population of a particular application. The population of a particular order entry system at a particular time is an M0 construct.

10.2 Features in Common (More or Less)

10.2.1 UML Kernel

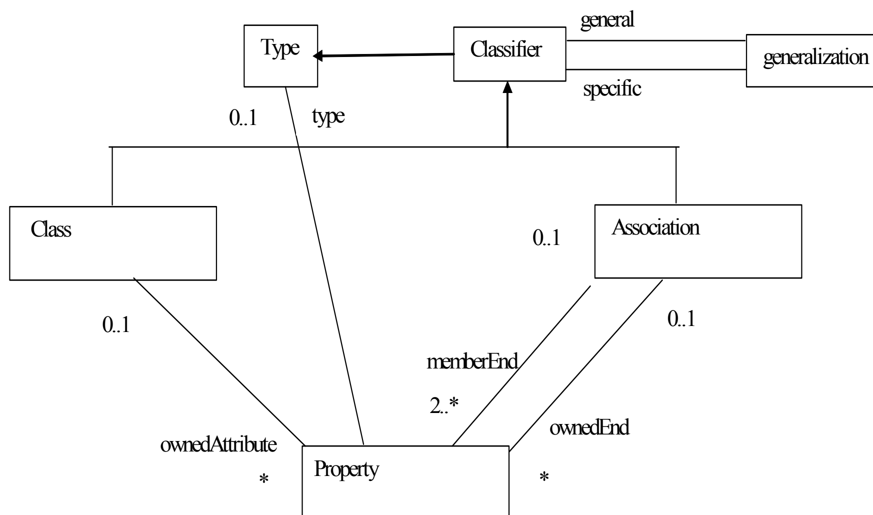


Figure 2 Key Aspects of UML Class Diagram

The structure of UML is formally quite different from OWL. What we are trying to do is to understand the relationship between an M1/M0 model in UML and the equivalent model in OWL, so we need to understand how the M1 model is represented in the M2 structure shown. First, a few observations from Figure 2.

- Most of the content of a UML model is in the M1 specification. The M0 model can be anything that meets the specification of the M1 model.
- There is no direct linkage between Association and Class. The linkage is mediated by Property.
- A Property is a structural feature (not shown), which is typed. The M1 model is built from structural features.
- Both Class and Association are types.
- A class can have a property which is the structural feature that implements it.
- A property may or may not be owned by one or more classes. A property owned by at least one class is called *navigable*[3]. A property owned by no class is called *not navigable*[4]. Associations can have navigable ends.

It will help if we represent a simple M1 model in this structure (Figure 3).

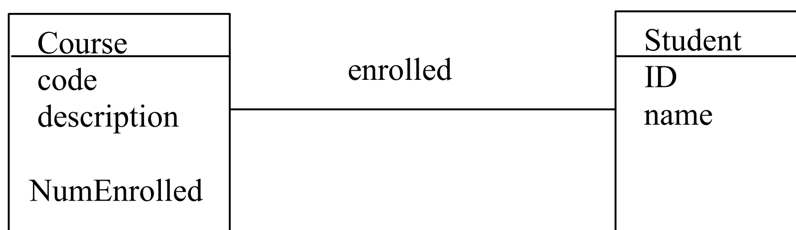


Figure 3 Simple M1 Model

The properties with their types are:

Table 9 Properties and Types in Simple Model

Property	Type
code	CourseIdentifier
description	string
NumEnrolled	integer
ID	StudentIdentifier
name	string

The classes are: Course, Student.

3. Called a member end in the Classes diagram of the UML superstructure
4. Called an owned end in the Classes diagram of the UML superstructure

Classes are represented by sets of *ownedAttribute* properties:

Table 10 Classes and Owned Properties in Simple Model

Class	ownedAttribute Properties
Course	code, description, NumEnrolled
Student	ID, name

Associations are: enrolled

The association can be modeled in a number of different ways, depending on how classes are implemented. If classes are implemented as in Table 10, one way is as the disjoint union of the owned attributes of the two classes.

Table 11 Implementation of Association in Simple Model

Association	Implementation
enrolled	code, description, NumEnrolled, ID, name

But there are other ways to implement a class. If it is known that the property *code* identifies instances of *Course* and that the property *ID* identifies instances of *Student*, then an alternative implementation of *enrolled* is:

Table 12 Alternative Implementation of Association in Simple Model

Association	Implementation
enrolled	code, ID

In this case, the properties *code* and *ID* would be of type *Course* and *Student* respectively.

10.2.2 Class and Property - Basics

Both OWL and UML are based on classes. A **class** in OWL is a set of **instances**. A class in UML is a more general construct, but one of its uses is as a set of instances. The set of instances associated at a particular time with a class is called the class' **extent**. There are subtle differences between OWL classes and UML classes which represent sets.

In UML the extent of a class is an M0 object consisting of instances. (Instances may be specified at the M1 level in a model library, but they are equivalent to M0 objects.) An instance consists of a set of slots each of which contains a value drawn from the type of the property of the slot. The instance is associated with one or more classifiers. An instance of the class *Course* might be:

Table 13 Example Course Instance

Classifier	code	title	NumEnrolled
Course	INFS3101	Ontology and the Semantic Web	0

But the M0 implementation of a class is not fully constrained. An equally valid instance of *Course* would be the name *INFS3101*, if it were decided that the name would identify an instance of the class. The remainder of the slots could be filled dynamically from other properties of the class.

In OWL, the extent of a class is a set of individuals, which are concretely represented. Individual is defined independently of classes. There is a universal class *Thing* whose extent is all individuals in a given OWL model, and all classes are subclasses of *Thing*. The main difference between UML and OWL in respect of instances is that in

OWL an individual may be an instance of Thing and not necessarily any other class, so could be outside the system in a UML model. It is of course possible to include a universal class in an M1 model library, but this would be sufficiently unusual to be problematic, whereas the concept is central to OWL.

An OWL class is declared by assigning a name to the relevant type. For example

```
<owl:Class rdf:ID=”Course”/>
```

An individual is at bottom an RDFS resource, which is essentially a name, so the individual INFS3101 will be declared with something like

```
<owl:Thing rdf:ID=“INFS3101”/>
```

Relationships among classes in OWL are called **properties**. That the class *course* has the relationship with the class *student* called *enrolled*, which was represented in the UML model as the association *enrolled*, is represented in OWL as a property

```
<owl:ObjectProperty rdf:ID = “enrolled”/>
```

Properties are not necessarily tied to classes. By default, a property is a binary relation between *Thing* and *Thing*.

So, in order to translate the M1 model of Figure 3 to OWL, UML Class goes to owl:Class.

Table 14 Simple Model Classes Translated to OWL

Class	Owned attributes	OWL equivalent
Course	code, description, NumEnrolled	`<owl:Class rdf:ID="Course"/>`
Student	ID, name	`<owl:Class rdf:ID="Student"/>`

The relationships among classes represented in OWL by owl:ObjectProperty and owl:DatatypeProperty come from two different sources in the UML model. One source is the M2 association *ownedAttribute* between Class and Property, which generates the representation of a class as a bundle of owned attributes as in Table 10. A M1 instance of *Class ownedAttribute Property* would translate as properties whose domain is *Class* and whose range is the type of *Property*. The UML *ownedAttribute* instance would translate to owl:ObjectProperty if the type of *Property* were a UML Class, and owl:DatatypeProperty otherwise. The translation ofTable 10 is shown in Table 15. Note that UML *ownedAttribute* M2 associations are distinct, even if *ownedAttributes* have the same name associated with different classes. The owl property names must therefore be unique. One way to do this is to use a combination of the class name and the owned property name. Note also that since instances of *ownedAttribute* are always relationships among types, the equivalent OWL properties all have domain and range specified.

An alternative way to give domain and range to OWL properties is to use restriction to allValuesFrom the range class when the property is applied to the domain class. This is probably a more natural OWL specification. However, since all OWL properties arising from a UML model are distinct, the method employed in this chapter is adequate. Should a translation of a UML model be intended as a base for further development in OWL, an appropriate translation can be employed (see section 18 UMLtoOWL translation???).

Table 15 Simple Model Associations Translated to OWL

Class	Owned property	Type of owned property	OWL equivalent
Course	code	CourseID	<owl:ObjectProperty rdf:ID="CourseCode"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="CourseID"/> </owl:ObjectProperty>
	description	string	<owl:DatatypeProperty rdf:ID="CourseDescription"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/> </owl:DatatypeProperty>
	Num Enrolled	integer	<owl:DatatypeProperty rdf:ID="CourseEnrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/> </owl:DatatypeProperty>
Student	ID	StudentIdent	<owl:ObjectProperty rdf:ID="StudentID"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="StudentIdent"/> </owl:ObjectProperty>
	name	string	<owl:DatatypeProperty rdf:ID="StudentName"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/> </owl:DatatypeProperty>

Note that the translation in Table 15 assumes that a single name is an identifier for instances of the corresponding class. This is not always true. That is there are cases in which a relational database implementation would use a compound key to identify an instance of a class. Since OWL individuals are always unitary names, the translation of the UML class would construct a unitary name from the values of the individual properties. For example, if the association *enrolled* were treated as a class (UML association class), its representing property might be a concatenation of Course.code and Student.id, so that the link for student 1234 enrolled in course INFS3101 might be translated to an OWL individual with name a globalized equivalent of 1234.INFS3101. Alternatively, a system-defined name could be assigned, linked to each name in the compound key by system-defined properties.

The second source of owl properties in a UML M1 model is the M1 population of the M2 class *association*. A binary UML association translates directly to an owl:ObjectProperty. The translation of Table 12 is given in Table 16. Note that since associations in UML are always between types, the OWL property always has domain and range specified. If the association name occurs more than once in the same model, it must be disambiguated in the OWL translation, for example by concatenating the member names to the association name.

Table 16 Sample Associations Translated to OWL

Association	Member 1 Property Type	Member 2 Property Type	OWL equivalent
enrolled	Course	Student	<owl:ObjectProperty rdf:ID="enrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="Student"/> </owl:ObjectProperty>

Both languages support the **subclass** relationship (OWL rdfs:subClassOf, UML generalization). Both also support **subproperties** (UML generalization of association or meta-associations among properties like subsetting or redefining). UML defines generalization at the supertype *classifier*, while in OWL subtype and subproperty are separately but identically defined.

The translation from UML to OWL is straightforward. If <S, G> is an M1 instance of the UML M2 association *generalization* (S is a subclassifier of G), then if both S and G are classes and TS, TG are respectively the types of the identifying owned property of S, G respectively, the OWL equivalent is the addition of the clause

```
<rdfs:subClassOf rdf:resource="TG"/>
```

to the definition of the OWL class TS. Similarly if S and G are both associations, the owl equivalent is the addition of the clause

```
<rdfs:subPropertyOf rdf:resource="G"/>
```

to the definition of the OWL object property S. Note that subassociations can be defined in a number of ways, including by OCL.

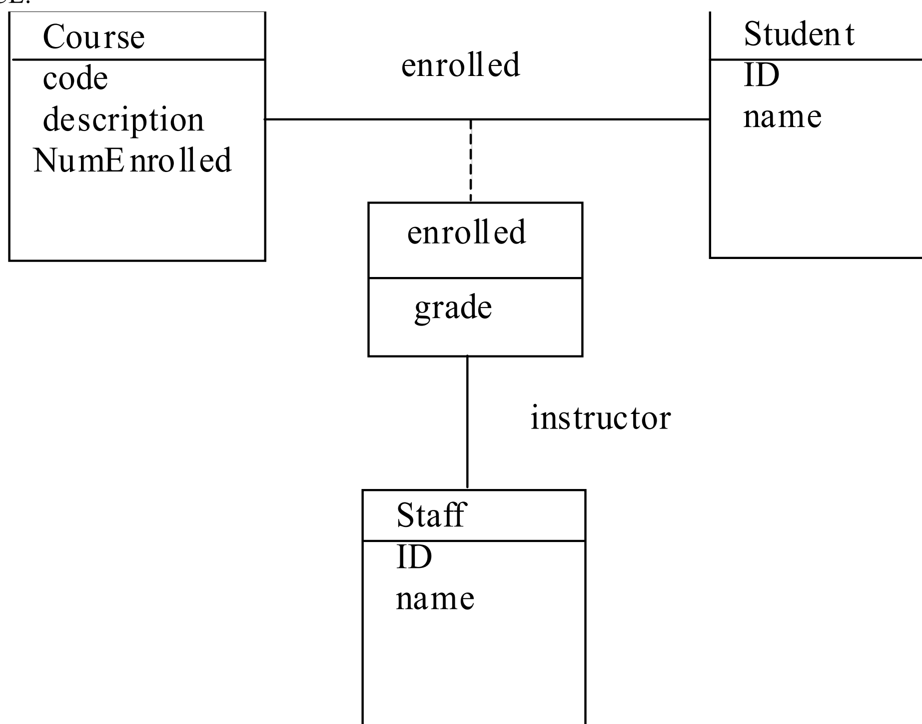


Figure 4 M1 Model with Association Class

An association in UML can be N-ary. It can have a possibly non-navigable end (*ownedEnd*). An association can also be a class (*association class*), so can participate in further associations. In OWL DL, classes and properties are disjoint, but in OWL Full they are overlapping. However, there is limited syntactic mechanism in the documents so far published to support this overlap. There is an advantage in translating these more complex associations to structures supported by OWL DL. In any case, the translations proposed are not normative, so those responsible for a particular application can use more powerful features of OWL if there is an advantage to doing so.

Our proposal takes advantage of the fact that an N-ary relation among types T_1 ... T_N, or an association class with attributes, is formally equivalent to a set R of identifiers together with N projection functions P_1, ..., P_N, where P_i:R -> T_i. Thereby N-ary UML associations are translated to OWL classes with bundles of binary functional properties.

Figure 4 extends the model of Figure 3 by making *enrolled* an association class whhich owns an attribute *grade*. The association class *enrolled* is a member end of an association *instructor*, whose other member end is *staff*. Some students enrolled in a given course may be assigned to one staff member as instructor, some as another.

The model of Figure 4 is represented in table form in Table 17. The association class *enrolled* is represented by its

Table 17 Sample Model Association Classes

Association	Parts	Type
enrolled	end 1	Course
	end 2	Student
	attri-bute	Grade
	Reifi-cation	enrolledR
instructor	end 1	enrolledR
	end 2	Staff

two end classes, *Course* and *Student*, the attribute of the association class *Grade*, and by an owned attribute *enrolledR* which implements the association class as a class, in the same way as in Table 11 and Table 12.

The implementation of *enrolled* and *Instructor* in Table 17 is translated into OWL as follows:

```
<owl:Class rdf:ID="enrolled" / >
<owl:FunctionalProperty rdf:ID="enrolledCourse">
     <rdfs:domain rdf:resource="enrolled/>
     <rdfs: range rdf:resource="Course"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledStudent">
     <rdfs:domain rdf:resource=enrolled/>
     <rdfs: range rdf:resource="Student"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledGrade">
     <rdfs:domain rdf:resource=enrolled/>
     <rdfs: range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledenrolledR">
     <rdfs:domain rdf:resource=enrolled/>
     <rdfs: range rdf:resource=enrolledR/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="instructor">
     <rdfs:domain rdf:resource=enrolledR/>
     <rdfs: range rdf:resource=Staff/>
</owl:FunctionalProperty >
```

10.2.3 More Advanced Concepts

There are a number of more advanced concepts in both UML and OWL. In the cases where the UML concept occurs in OWL, the translation is often quite straightforward, so will not always be shown.

Both languages support a module structure, called **package** in UML and **ontology** in OWL. The translation of package to ontology is straightforward. Both languages also support **namespaces**.

Both UML and OWL support a fixed defined extent for a class (OWL **oneOf**, UML **enumeration**). Note that in UML enumeration is a datatype rather than a class.

UML has the option for associations to have distinguished ends which can be **navigable** or **non-navigable**. A navigable property is one which is owned by a class or optionally an association, while a non-navigable is not (an integer, say). OWL properties always are binary and have distinguished ends called **domain** and **range**. A UML binary association with one navigable end and one non-navigable end will be translated into a property whose domain is the navigable end. A UML binary association with two navigable ends will be translated into a pair of OWL properties, where one is **inverseOf** the other.

A key difference is that in OWL a property is defined by default as having range and domain both *Thing*. A given property therefore can in principle apply to any class. So a property name has global scope and is the same property wherever it appears. In UML the scope of a property is limited to the subclasses of the class on which it is defined. A UML association name can be duplicated in a given diagram, with each occurrence having a different semantics. It is possible, though not customary, to include a universal superclass in an M1 model library. This is sufficiently unusual that it is not clear what the current toolsets would do with it.

An OWL individual can therefore be problematic a UML model. UML has a facility **dynamic classification** which allows an instance of one class to be changed into an instance of another, which captures some of the features of Individual, but an object must always be an instance of some class. UML models rarely include universal classes.

Both languages allow a class to be a subclass of more than one class (**multiple inheritance**). Both allow subclasses of a class to be declared **disjoint**. (In OWL, all classes are subclasses of *Thing*, so any pair of classes can be declared disjoint.) UML allows a collection of subclasses to be declared to **cover** a superclass, that is to say every instance of the superclass is an instance of at least one of the subclasses. The corresponding OWL construct is the declare the superclass to be the union of the subclasses, using the construct **unionOf**.

UML has a strict separation of metalevels, so that the population of M1 classes is distinct from the population of M0 instances and also the M1 model libraries. OWL Full permits classes to be instances of other classes. UML only models classes of classes in the context of declaration of disjoint or covering powersets.

In OWL, a property when applied to a class can be constrained by cardinality restrictions on the domain giving the minimum (**minCardinality**) and maximum (**maxCardinality**) number of instances which can participate in the relation. In addition, an OWL property can be globally declared as functional (**functionalProperty**) or inverse functional (**inverseFunctional**). A functional property has a maximum cardinality of 1 on its range, while an inverse functional property has a maximum cardinality of 1 on its domain. In UML an association can have minimum and maximum cardinalities (**multiplicity**) specified for any of its ends. OWL allows individual-valued properties (objectProperty) to be declared in pairs, one the inverse of the other.

So if a binary UML association has a multiplicity on a navigable end, the corresponding OWL property will have the same multiplicity. If a binary UML association has a multiplicity on its both ends, then the corresponding OWL property will be an inverse pair, each having one of the multiplicity declarations.

For an N-ary UML association, multiplicities are more problematic. For example, in Figure 5, the multiplicities show that given instances of *event*, *Olympiad* and *competitor*, there is at most one instance of *result*; given instances of *event*, *Olympiad* and *result* there is at most one instance of *competitor*; given instances of *Olympiad*, *competitor* and *result* there may be many instances of event (an athlete may compete at several events in the same Olympiad and finish in the same place in each); and given instances of *event*, *competitor* and *result* there may be many instances of *Olympiad* (an athlete may compete in the same event at several Olympiads and finish in the same place in each). For an N-ary UML association, any multiplicity associated with one of its end classes will apply to the OWL property translating the corresponding projection from the association class to the translated end class.

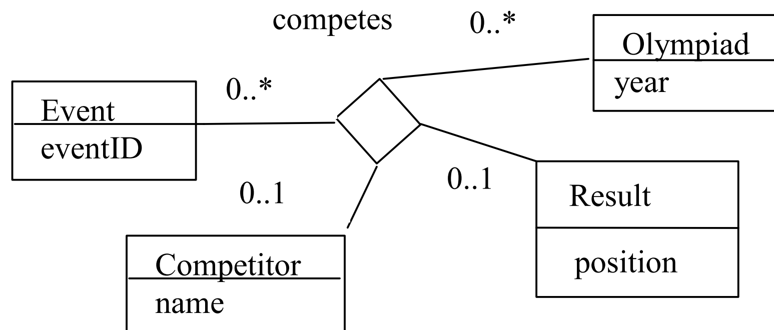


Figure 5 Example N-ary Association with Multiplicity

The N-ary association in Figure 5 would be translated as a class *competes* whose instances are instances of links in the association, and four properties whose domain is *competes* and whose ranges are the classes attached to the member ends of the association. Since one instance of a link includes only one instance of the class at each member end, all the properties are functional. The multiplicities on the UML diagram do not translate to OWL in a straightforward way.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY owl   "http://www.w3.org/2002/07/owl#">
<!ENTITY rdf   "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs  "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY xsd   "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;"
      xmlns:owl="&owl;"
        xmlns:xsd="&xsd;"/>

<owl:Class rdf:ID="competes">
     <owl:subClassOf>
        <owl:Restriction>
       <owl:onProperty rdf:resource="competesEvent"/>
          <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
     </owl:subClassOf>
     <owl:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="competesCompetitor"/>
          <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
     </owl:subClassOf>
     <owl:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#competesOlympiad"/>
          <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
     </owl:subClassOf>
     <owl:subClassOf>
        <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="#competesResult"/>
           <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
         </owl:Restriction>
      </owl:subClassOf>
      <owl:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#competesResult"/>
           <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
         </owl:Restriction>
      </owl:subClassOf>
</owl:Class>
<owl:FunctionalProperty rdf:ID="competesEvent">
 <rdfs:domain rdf:resource="#competes"/>
   <rdfs:range rdf:resource="#Event"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesCompetitor">
  <rdfs:domain rdf:resource="#competes"/>
     <rdfs:range rdf:resource="#Competitor"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesOlympiad">
  <rdfs:domain rdf:resource="#competes"/>
      <rdfs:range rdf:resource="#Olympiad"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesResult">
  <rdfs:domain rdf:resource="#competes"/>
     <rdfs:range rdf:resource="&xsd;string> >
</owl:FunctionalProperty>
</rdf:RDF>
```

In UML, multiplicities can be defined on both ends of an association. In OWL, general multiplicities apply to the range instances associated with a given domain instances. In both cases, multiplicities can be strengthened when associations/properties are applied to subclasses.

Note that the class might be the domain of a property for which the individual might not have a value. This can happen if the mincardinality of the domain of the property is 0, in which case the property is optional (or partial) for that class. The same can happen in UML. An instance of a class is constrained to participate only in properties which are mandatory, minimum cardinality > 0. So an instance can lack optional properties. (The somewhat strange construct maxCardinality < minCardinality is syntactically correct in OWL and has the semantics that the property has no instances. It can occur where multiple autonomous ontologies are merged, for example.)

However, even if the property is mandatory (mincardinality > 0 and maxcardinality >= mincardinality), there may not be definite values for the property. Consider a class (K) for which a property (P) is mandatory. In this case, the individual (I) must satisfy the predicate

[M]: I instance of K -> exists X such that P(I) = X.

It is not required in OWL that there be a constant C such that X = C. All horses have color, but we may not know what color a particular horse has.

In UML, there is a strict separation between the M1 and M0 levels. At the M1 level, that an association is mandatory (minimum cardinality greater than 0) is exactly the predicate [M]. Any difference between UML and OWL must come from the treatment of the model of the M1 theory at the M0 level. In practice, M0 models in UML applications tend to be Herbrand models implemented by something like an SQL database manager. For these cases, if we know a horse has a color, then we know what color it has. To the extent that UML tools and modelling build this expectation into products, conflict can occur when interoperating with an OWL ontology.

But UML does not mandate M0 models to be Herbrand models. In particular SQL-92 supports the Null value construct, which has multiple interpretations, including “value exists but is not known”. Some years ago, CJ Date proposed a zoo of nulls with specific meanings, including “value exists but is not known”, and there have been proposals by Ray Reiter and others for databases with either existentially quantified variables in the data or which reason with the M1 theory for existentially quantified queries. It is possible for a particular application to introduce a special constant “unknown” into a class, which is treated specially by the programs. UML does not forbid an implementation of a class model in one of these ways. So there is no difference in principle between UML and OWL for properties which are declared to have minCardinality greater than 0 (and maxCardinality >= minCardinality) for a class.

Note that a consequence of this possible indeterminacy, it may not be possible to compute a transitive closure for a property across several ontologies, even if they share individuals.

An OWL property can have its range restricted when applied to a particular class, either that the range is limited to a class (subclass of *range* if declared) (**allValuesFrom**) or that the range must intersect a class (**someValuesFrom**). UML permits these and other restrictions using the facilities **specializes** or **refines**.

OWL allows properties to be declared symmetric (**SymmetricProperty**) or transitive (**TransitiveProperty**). In both cases, if the domain and range are not type compatible, the property is empty. UML uses OCL for this purpose.

OWL permits declaration of a property whose value is the same for all instances of a class, so the property value is in effect attached to the class (OWL DL property declared as allValuesFrom a singleton set for that class). OWL full allows properties to be directly assigned to classes without special machinery. If class A is an instance of class B, then a property P whose domain includes B will designate a value P(A) which applies to the class A so can be derived for all instances of A.

UML allows a property to be **derived** from other model constructs, for example a composition of associations or from a generalization. That a property is derived can be represented as an annotation in OWL. The actual derivation rule is problematic. Derivation rules in UML are expressed in OCL, and there is no general translation of OCL to OWL.

A classifier in UML can be declared **abstract**. An abstract classifier typically cannot be instantiated, but may be a superclass of concrete classifiers. There is no OWL equivalent for this.

Two different objects modeled in UML may have dependencies which are not represented by UML named (model) elements, so that a change in one (the supplier) requiring a change in the other (the client) will not be signaled by for example association links. Two such objects may be declared **dependent**. There are a number of subclasses of dependency, including abstraction, usage, permission, realization and substitution. OWL does not have a comparable feature except as annotations, but RDF, the parent of OWL, permits an RDF:property relation between very general elements classified by RDFS:Class. Therefore, a dependency relationship between a supplier and client UML model element will be translated to a reserved name RDF:Property relation whose domain and range are both RDF:Class. Population of the property will include the individuals which are the target of the translation of the supplier and client named elements.

10.2.4 Summary of More-or-Less Common Features

This section has described features of UML and OWL which are in most respects similar. Table 18 summarizes the features of UML in this feature space, giving the equivalent OWL features. UML features are grouped in clusters which translate to a single OWL feature or a cluster of related OWL features. The column *Package* shows the section of the UML Superstructure document [UML2] where the relevant features are documented.

Table 18 Common Features of UML and OWL

UML features	Package	OWL features	Comment
class, property ownedAttribute, type[a]	7.3.7 Classes 7.3.8 Classifiers 7.3.32 Multiplicities	class	
instance	7.3.22 Instances	individual	OWL individual independent of class
ownedAttribute, binary association	7.3.7 Classes	property	OWL property can be global
subclass, generalization	7.3.7 Classes 7.3.8 Classifiers	subclass subproperty	
N-ary association, association class	7.3.7 Classes 7.3.4 Association Classes	class, property	
enumeration	7.3.11 Datatypes	oneOf	
navigable, non-navigable	7.3.7 Classes	domain, range	
disjoint, cover	7.3.21 Generalization sets	disjointWith, unionOf	
multiplicity	7.3.32 Multiplicities	minCardinality maxCardinality inverseOf	OWL cardinality declared only for range
derived	7.3.7 Classes	no equivalent	
package	7.3.37 Packages	ontology	
dependency	7.3.12 Dependencies	reserved name RDF:property	
abstract classifier	7.3.8 Classifiers	no equivalent	

a. This cell summarizes the relationship between UML class and OWL class mediated by property, ownedAttribute and type. It does not signify that the latter three are themselves translated to OWL class.

All of the UML features considered in the scope of the ODM have more-or-less satisfactory OWL equivalents. Some OWL features in this feature space have no UML equivalent, so are omitted from Table 18. They are summarized in Table 19. Besides the small differences in the features in the feature space common to UML and OQL, there are some

Table 19 OWL Features with No UML Equivalent

OWL features with no UML equivalent
Thing, global properties, autonomous individual
allValuesFrom, someValuesFrom
SymmetricProperty, TransitiveProperty
Classes as instances

more general differences described in the next section.

10.3 OWL but not UML

10.3.1 Predicate Definition Language

OWL permits a subclass to be declared using subClassOf or to be inferred from the definition of a class in terms of other classes. It also permits a class to be defined as the set of individuals which satisfy a restriction expression. These expressions can be a boolean combination of other classes (**intersectionOf**, **unionOf**, **complementOf**), or property value restriction on properties (requirement that a given property have a certain value – **hasValue**). The property **equivalentClass** applied to restriction expressions can be used to define classes based on property restrictions.

For example, the class definition[5]

```
<owl:Class rdf:ID=”TexasThings”>
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource=”#locatedIn” />
            <owl:allValuesFrom rdf:resource=”#TexasRegion” />
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

Defines the class *TexasThings* as a subclass of the domain of the property *locatedIn*. These individuals are precisely those for which the range of *locatedIn* is in the class *TexasRegion*. Given that we know an individual to be an instance of *TexasThings*, we can infer that it has the property *locatedIn*, and all of the values of *locatedIn* associated with it are instances of *TexasRegion*. Conversely, if we have an individual which has the property *locatedIn* and all of the values of *locatedIn* associated with that individual are in *TexasRegion*, we can infer that the individual is an instance of *TexasThings*.

Because it is possible to infer from the properties of an individual that it is a member of a given class, we can think of the complex classes and property restrictions as a sort of predicate definition language.

UML provides but does not mandate the predicate definition language OCL. Note that a subsumption reasoner could be built for UML. But because UML is strongly typed, it could work in the way mandated for OWL only if there were a universal superclass provided in the model library, which is rarely provided in practice.

5. OWL Web Ontology Language Guide http://www/w3/org/TR/2003/PR-owl-guide-20031215/ section 3.4.1

OCL and CL (Common Logic) are two predicate definition languages which are relevant to the ODM. Both are more expressive than the complex class and property restriction expressions of OWL Full. There are also other predicate definition languages of varying expressive powers which particular applications might wish to use.

The ODM does not mandate any particular predicate definition language, but will provide a place for a package enabling the predicate definition language of choice for an application. In particular, the ODM includes a metamodel for CL.

10.3.2 Names

A common assumption in computing applications is that within a namespace the same name always refers to the same object, and that different names always refer to different objects (the **unique name assumption**). As a consequence, given a set of names, one can count the names and infer that the names refer to that number of objects.

Names in OWL do not by default satisfy the unique name assumption. The same name always refers to the same object, but a given object may be referred to by several different names. Therefore counting a set of names does not warrant the inference that the set refers to that number of objects. Names, however, are conceptually constants, not variables.

OWL provides features to discipline names. The unique name assumption can be declared to apply to a set of names (**allDifferent**). One name can be declared to refer to the same object as another (**sameAs**). One name can be declared to refer to something different from that referred to by any of a set of names (**differentFrom**).

Two classes can be stated to be equivalent (**equivalentClass**) and two properties can be stated to be equivalent (**equivalentProperty**). Equivalent classes have the same extents, equivalent properties link the same pairs.

UML supports named elements with namespaces at the M1 level. Although a UML class may be defined to contain a definite collection of names, names at the M0 level are not prescribed except their namespaces are inherited from the M1 declarations. Applications modeled in UML are frequently implemented using systems like SQL which default the unique name assumption, but this is not mandated. UML places no constraints on names at the M0 level.

In particular, it is permitted for applications modeled in UML to be implemented at the M0 level using names which are variables. Note that the UML constraint language OCL uses variables. OWL does not support variables at all.

10.3.3 Other OWL Developments

There are a number of developments related to OWL which are not yet finalized, including SWRL Semantic Web Rule Language and OWL services. These are considered out of scope for the ODM. A translation of an out-of-scope model element will be to a comment in the OWL target.

10.4 In UML But Not OWL

10.4.1 Behavioral and Related Features

UML allows the specification of behavioral features, which declare capabilities or resources. One use of behavioral features is to calculate property values. Behavioral features can be used in the OCL that derives properties. Facilities of UML supporting programs include **operations**, which describe the parameters of methods; **responsibilities**, which specify which class is responsible for what action; **static operations**, which are operations attached to a class like static attributes; **interface classes**, which specify among other things operation features; **qualified associations**, which are a special kind of ternary relation; and **active classes**, which are classes each instance of which controls its own thread of execution control.

ODM omits these features of UML.

10.4.2 Complex Objects

UML supports various flavors of the part-of relationship between classes. In general, a class (of parts) can have a part-of relationship with more than one class (of wholes). One flavor (**composition**) specifies that every instance of a given class (of parts) can be a part of at most one whole. Another (**aggregation**) specifies that instances of parts can be shared among instances of wholes.

Composite structures defined in classes specify runtime instances of classes collaborating via **connectors**. They are used to hierarchically decompose a class into its internal structure which allows a complex objects to be broken down into parts. These diagrams extend the capabilities of class diagrams, which do not specify how internal parts are organized within a containing class and have no direct means of specifying how interfaces of internal parts interact with its environment.

Ports model how internal instances are to be organized. Ports define an interaction point between a class and its environment or a class and its contents. They allow you to group the required and provided interfaces into logical interactions that a component has with the outside world. **Collaboration** provides constructs for modeling roles played by connectors.

Although not strictly part of the complex object feature set, the feature **template** (parameterized class) is most useful where the parameterized class is complex. One could for example define a multimedia object class for movies, and use it as a template for a collection of classes of genres of movie, or a complex object giving the results of the instrumentation on a fusion reactor which would be a template for classes containing the results of experiments with different objectives.

Although it is recognized that there is a need for facilities to model mereotopological relationships in ontologies, there does not seem to be sufficient agreement on the scope and semantics of existing models for inclusion of specific mereotopological modeling features into the ODM at this stage.

These modeling elements will be translated to properties or classes as ownedAttributes or association ends. The target elements will be annotated with appropriate comments.

10.4.3 Access Control

UML permits a property to be designated **read-only**. It also allows classes to have **public** and **private** elements.

ODM omits access control features.

10.4.4 Keywords

UML has keywords which are used to extend the functionality of the basic diagrams. They also reduce the amount of symbols to remember by replacing them with standard arrows and boxes and attaching a <<keyword>> between guillements. A common feature that uses this is <<interfaces>>.

ODM omits this feature.

11 The RDF Schema Metamodel

11.1 Overview

The RDF Schema (RDFS) Metamodel is a MOF2 compliant metamodel that allows a user to define ontology models using the same terminology and concepts as those defined in RDF Schema [RDF Schema].

RDF Schema is the vocabulary description language for RDF (Resource Definition Framework) which is a language for representing information about resources in terms of properties. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. RDF Schema specifies mechanisms that may be used to name and describe properties and the classes of resource they describe. It defines classes and properties that may be used to describe classes, properties and other resources.

11.1.1 Organization of the RDFS Metamodel

The RDFS Metamodel uses diagrams to control complexity and promote understanding.

The classes and associations are grouped and illustrated in the following diagrams:

- Classes
  Contains classes and associations that can be used to define RDF classes and datatypes.
- Properties
  Contains classes and associations that can be used to define RDF properties.
- Containers
  Contains classes and associations that can be used to define RDF containers and their members.
- Collections
  Contains classes and associations that can be used to define RDF collections (i.e., lists) and their members.
- Reification
  Contains classes and associations that can be used to define RDF statements.
- Utilities
  Contains classes and associations that can be used to define utility RDF properties.
- Ontology
  Contains classes and associations that can be used to define the scope of an RDFS ontology.

11.1.2 Design Considerations

Metamodel Constructs

Classes defined in RDF Schema are represented by MOF classes. Properties defined in RDF Schema are represented by MOF associations.

Note:

RDF properties are first-class entities with unique identifiers. In addition, an RDF property can be a subproperty of another RDF property. MOF associations, on the other, are not first-class entities. They are defined between two MOF classes and their role names are locally scoped. In addition, in EMOF, a MOF association cannot be a subassociation of another MOF association. Therefore, there is an inherent impedance mismatch between RDF Schema and EMOF. Naming and textual description are used to overcome this impedance mismatch.

Naming

Classes and properties defined in RDF Schema have URI based unique names (e.g., rdfs:Class, rdf:Property, rdfs:subClassOf, rdf:type). Names of MOF classes are package qualified rather than globally scoped, as is the case with conventional XML uniform resource identifiers (URIs). In fact, rdfs:Class, rdf:Property, and other names specified in the RDF specifications are actually abbreviations for URIs using conventional namespace prefixes and concatenation. To make matters worse, names of MOF association roles are local to the MOF classes where they are defined. To overcome this impedance mismatch, prefixes are used in naming MOF classes and MOF properties that directly represent RDFS classes and RDFS properties, respectively. For example, RDFSClass represents rdfs:Class and RDFProperty represents rdf:Property. Vocabulary, which does not have a prefix, represents something which is not explicitly defined in RDF Schema. An optional, explicit approach to representing URIs and URI references, per the RDF and XML specifications, is given in Chapter 16, UML Profiles for RDF Schema and OWL.

11.2 The Classes and Utilities Diagrams

The Classes diagram is shown in Figure 6. The Utilities diagram is shown in Figure 7. These two diagrams are shown together to facilitate the specification of RDFSResource.

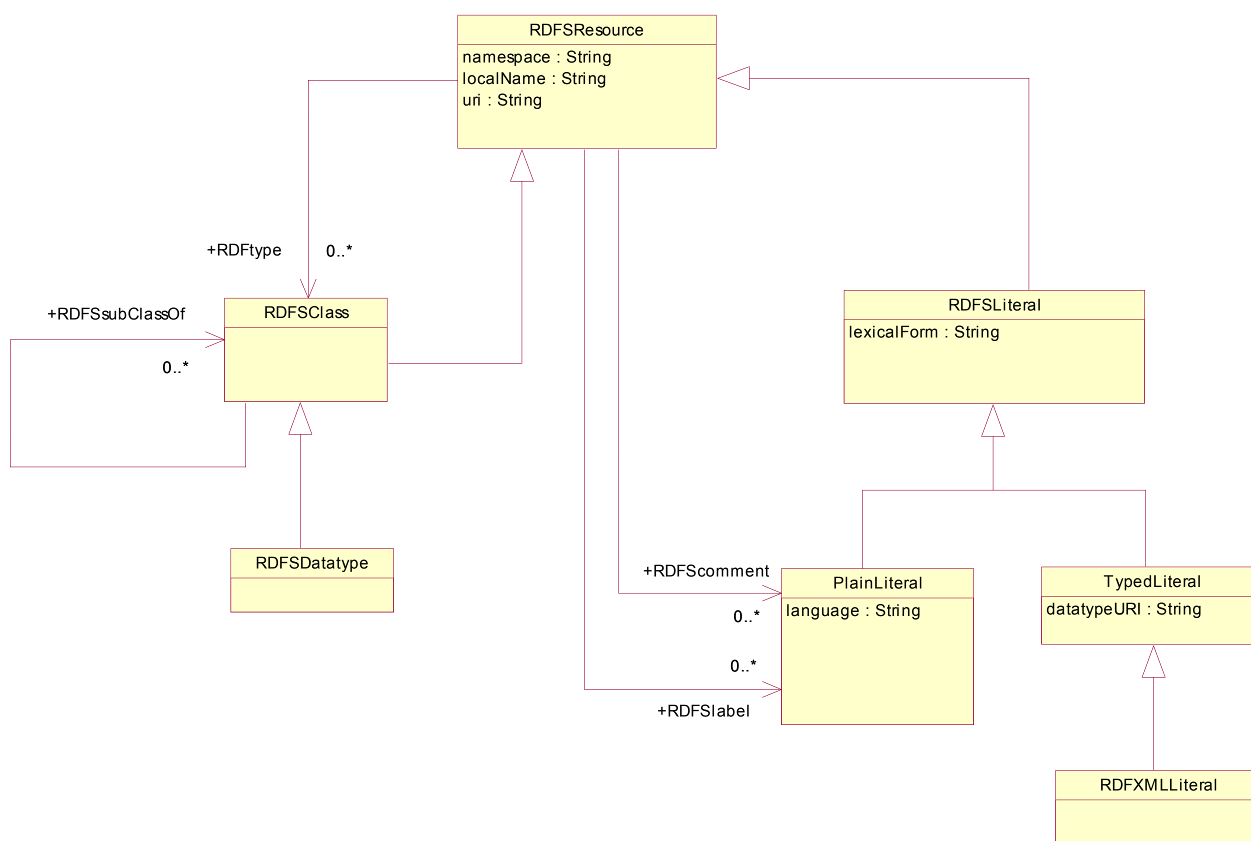


Figure 6 The Classes Diagram of the RDF Schema Metamodel

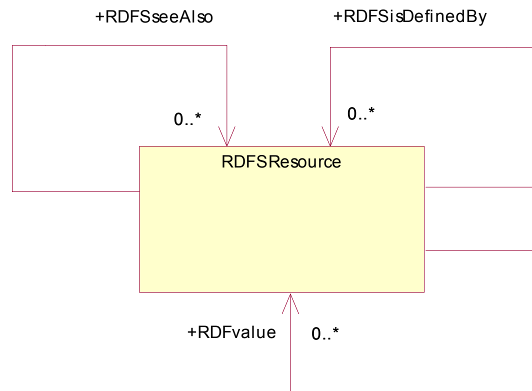


Figure 7 The Utilities Diagram of the RDF Schema Metamodel

11.2.1 PlainLiteral

This is the class of plain literal values such as strings.

Description

This is the class of plain literal values such as strings. Plain literals consist of a Unicode string in Normal Form C and an optional language tag.

Attributes

- language: String [0..1]
  The optional language tag used for plain literals.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See description.

11.2.2 RDFSClass

This is the class of resources that are classes.

Description

This is the class of resources that are classes. Classes provide an abstraction mechanism for grouping resources with similar characteristics

Attributes

No additional attributes.

Associations

- subClassOf: RDFSClass [0..*]
  This is an instance of RDFProperty that is used to state that all the instances of one class are instances of another. If the class C1 is defined as a subclass of class C2, then the set of instances of C1 should be a subset of the set of instances of C2. This property is transitive.

Constraints

No additional constraints.

Semantics

The purpose of a class is to provide an abstraction mechanism for grouping resources. A resources can be an instance of more than one class.

11.2.3 RDFSDatatype

This is the class of datatypes.

Description

This is the class of datatypes. It is needed to convert typed literals between lexical and value forms.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

[1] The inherited URI attribute is obligatory for members of RDFSDatatype.

Semantics

Instances of datatypes are typed literals, i.e., each is a subclass of RDFSLiteral. The following XML Schema simple types [XML Schema Datatypes] can be used as built-in datatypes: string, boolean, decimal, float, double, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, hexBinary, base64Binary, anyURI, normalizedString, token, language, NMTOKEN, Name, NCName, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, and positiveInteger. The URI reference of these datatypes is of the form:

```
http://www.w3.org/2001/XMLSchema#NAME
```

11.2.4 RDFSLiteral

This is the class of literal values such as strings and integers.

Description

This is the class of literal values such as strings and integers. Literals may be plain or typed. Plain literals consist of a Unicode string in Normal Form C and an optional language tag. Typed literals consist of a lexical representation, but without the optional language tag.

Attributes

- lexicalForm: String
  A Unicode string in Normal Form C.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See description.

11.2.5 RDFSResource

All things described by RDF are called resources.

Description

All things described by RDF are called resources. This is the class of everything. All other classes are subclasses of this class.

Attributes

- localName: String [0..1]
  The local name of the RDF resource.

- namespace: String [0..1]
  The namespace of the RDF resource.

- uri: String [0..1]
  The unique identifier of the RDF resource. It may be constructed from the namespace and localName of the RDF resource (if both are present), and, in most cases, the namespace and localName can be derived from the uri.

Associations

- RDFScomment: PlainLiteral [0..*]
  This is an instance of RDFProperty that may be used to provide a human-readable description of a resource.

- RDFSisDefinedBy: RDFSResource [0..*]
  This is an instance of RDFProperty that is used to indicate a resource defining this resource. It is a subproperty of RDFSseeAlso.

- RDFSlabel: PlainLiteral [0..*]
  This is an instance of RDFProperty that may be used to provide a human-readable version of a resource's name.

- RDFSmember: RDFSResource [0..*]
  This is an instance of RDFProperty that is a super-property of all the container membership properties. (See Figure 9.)

- RDFSseeAlso: RDFSResource [0..*]
  This is an instance of RDFProperty that is used to indicate a resource that might provide additional information about this resource.
- RDFtype: RDFSClass [0..*]
  This is an instance of RDFProperty that is used to state that a resource is an instance of a class.
- RDFvalue: RDFSResource [0..*]
  This is an instance of RDFProperty that may be used in describing structured values.

Constraints

None.

Semantics

The localName attribute is used for identification of an RDF resource within the namespace in which it is defined. The uri attribute is used for unique identification of an RDF resource globally. In general, the uri attribute is constructed from the namespace and localName attributes, or vice versa. Note that these attributes have a multiplicity of [0..1] which provides for the possibility of the absence of an identifier.

11.2.6 RDFXMLLiteral

This is the class of XML literal values.

Description

This is the class of XML literal values. It is an instance of RDFSDatatype.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See description.

11.2.7 TypedLiteral

This is the class of typed literal values such as integers.

Description

This is the class of typed literal values such as integers. Typed literals consist of a lexical representation and an optional datatype URI. A typed literal is an instance of a RDFSDatatype.

Attributes

- datatypeURI: String [0..1]
  The optional datatype URI for typed literals.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See description.

11.3 The Properties Diagram

The Properties diagram of the RDF Schema metamodel is shown in Figure 8.

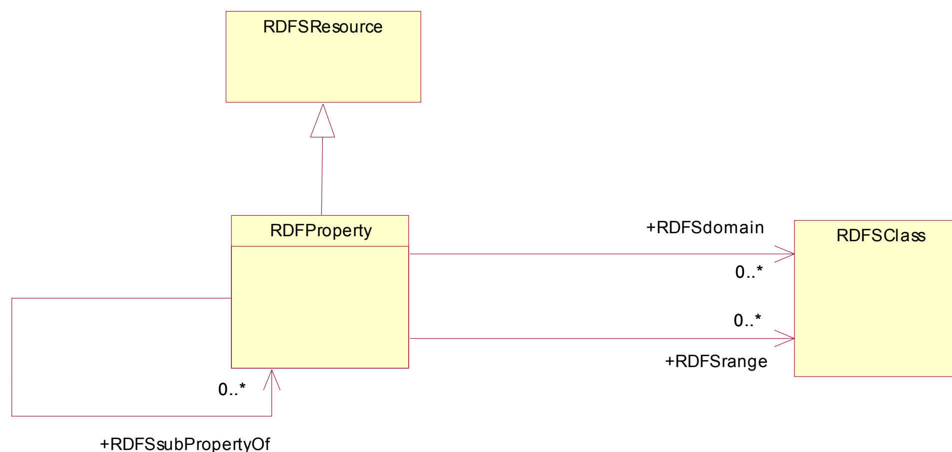


Figure 8 The Properties Diagram of the RDF Schema Metamodel

11.3.1 RDFProperty

This is the class of properties.

Description

This is the class of properties. A property relates resources to resources or literals. Every property is associated with a set of instances, called the property extension. Instances of properties are subject-object pairs or subject-value pairs.

Attributes

No additional attributes.

Associations

- RDFSdomain: RDFSClass [0..*]
  This is an instance of RDFProperty that is used to state that any resource that has a given property is an instance of one or more classes.
- RDFSrange: RDFSClass [0..*]

This is an instance of RDFProperty that is used to state that the values of a property are instances of one or more classes.

- RDFSsubPropertyOf: RDFProperty [0..*]
  This is an instance of RDFProperty that is used to state that all resources related by one property are also related by another. If a property P1 is a subproperty of property P2, then all pairs of resources which are related by P1 are also related by P2. The term super-property is often used as the inverse of subproperty.

Constraints

No additional constraints.

Semantics

A property relates resources to resources or literals. A property can be declared with or without specifying its domain (i.e., classes which the property can apply to) or range (i.e., classes or datatypes that the property may have value from).

Properties may be specialized (subPropertyOf). The existence of an instance of a specializing property implies the existence of an instance of the specialized property, relating the same set of resources.

11.4 The Containers Diagram

The Containers diagram of the RDF Schema metamodel is shown in Figure 9.

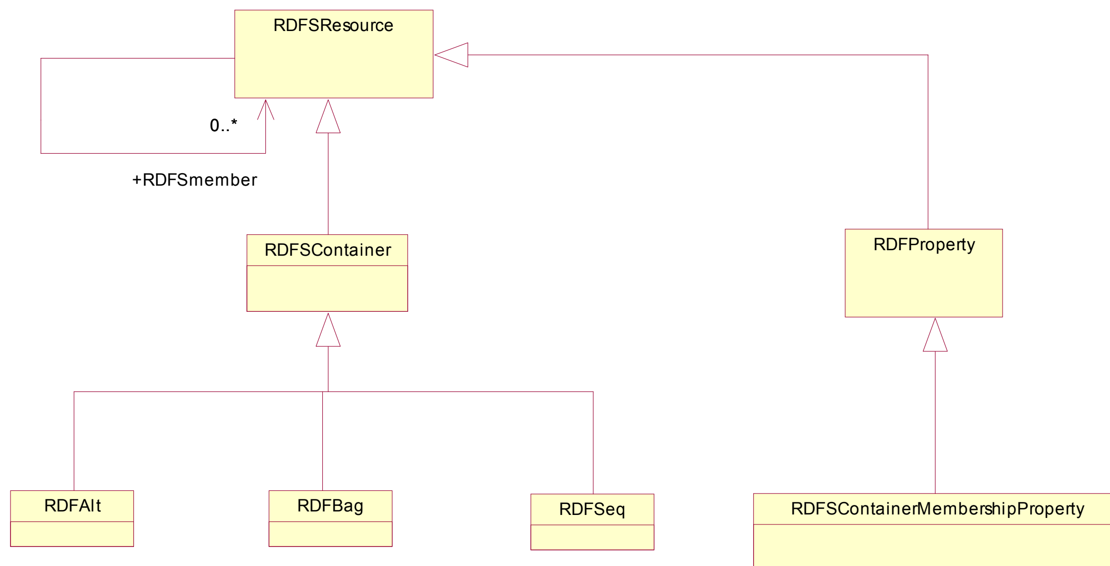


Figure 9 The Containers Diagram of the RDF Schema Metamodel

11.4.1 RDFAlt

This is the class of RDF “Alternative” containers.

Description

This is the class of RDF "Alternative" containers. It is used conventionally to indicate that typical processing will be to select one of the members of the container. The first member of the container is the default choice.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

11.4.2 RDFBag

This is the class of RDF "Bag" containers.

Description

This is the class of RDF "Bag" containers. It is used conventionally to indicate that the container is intended to be unordered.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

11.4.3 RDFSContainer

This is a super-class of RDF container classes.

Description

This is a super-class of RDF container classes. RDF containers are resources that are used to represented containers.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

The same resource may appear in a container more than once. A property of a container is not necessarily a property of all of its members.

11.4.4 RDFSContainerMembershipProperty

This class has as instances the properties that are used to state that a resource is member of a container.

Description

This class has as instances the properties that are used to state that a resource is a member of a container. Each instance of this class is a subproperty of the RDFSmember property.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

Container membership properties may be applied to resources other than containers.

11.4.5 RDFSeq

This is the class of RDF "Sequence" containers.

Description

This is the class of RDF "Sequence" containers. It is used conventionally to indicate that the numerical ordering of the container membership properties of the container is intended to be significant.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

11.5 The Collections Diagram

The Collections diagram of the RDF Schema metamodel is shown in Figure 10.

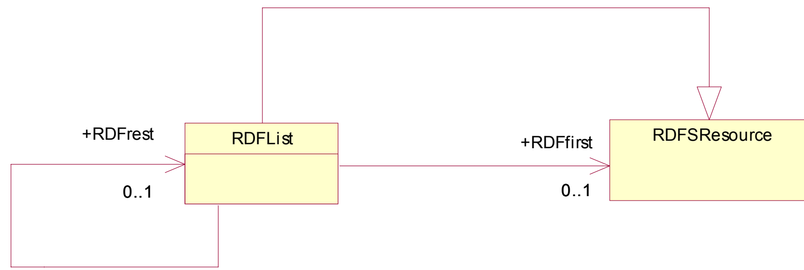


Figure 10 The Collections Diagram of the RDF Schema Metamodel

11.5.1 RDFList

This class represents descriptions of lists and other list-like structures.

Description

This class represents descriptions of lists and other list-like structures.

Attributes

No additional attributes.

Associations

- RDFfirst: RDFSResource [0..1]
  The first element of this list.
- RDFrest: RDFList [0..1]
  The list that consists of the rest of the elements of this list.

Constraints

No additional constraints.

Semantics

rdf:Nil is a predefined instance of rdf:List which explicitly denotes the termination of an rdf:List. Since rdf:Nil is at the model level, it is not explicitly represented.

11.6 The Reification Diagram

The Reification diagram of the RDF Schema metamodel is shown in Figure 11.

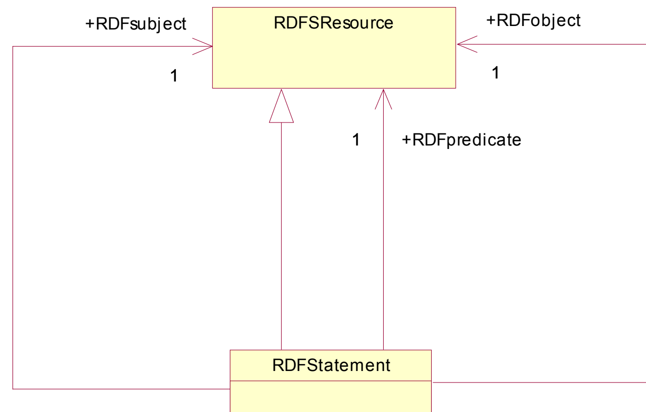


Figure 11 The Reification Diagram of the RDF Schema Metamodel

11.6.1 RDFStatement

This class is intended to represent the class of RDF statements.

Description

This class is intended to represent the class of RDF statements.

Attributes

No additional attributes.

Associations

- RDFobject: RDFSResource [1]
  The is an instance of RDFProperty that is used to state the object of a statement.
- RDFpredicate: RDFProperty [1]
  The is an instance of RDFProperty that is used to state the predicate of a statement.
- RDFsubject: RDFSResource [1]
  The is an instance of RDFProperty that is used to state the subject of a statement.

Constraints

No additional constraints.

Semantics

See Description.

11.7 The Ontology Diagram

The Ontology diagram of the RDF Schema metamodel is shown in Figure 12.

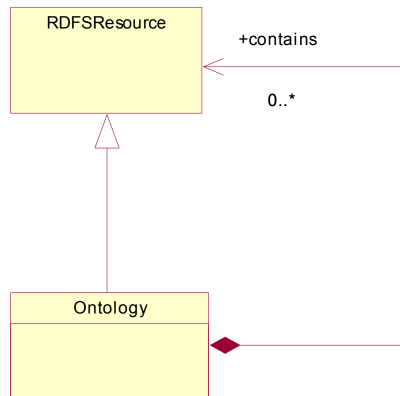


Figure 12 The Description Diagram of the RDF Schema Metamodel

11.7.1 Ontology

An ontology consists of the various classes and properties that can be used to describe and represent a domain of knowledge.

Description

An ontology consists of the various classes and properties that can be used to describe and represent a domain of knowledge. Classes represent concepts within a domain or across domains, and properties represent the relationships among them.

Attributes

None.

Associations

- contains: RDFSResource [0..*]
  The resources owned by this ontology.

Constraints

None.

Semantics

An ontology provides a container and a namespace for resources. If an ontology is removed, so are the resources owned by it.

11.8 Language Mappings

This section specifies the mappings of the constructs in the RDFS metamodel to elements of the RDF Schema language syntax [RDF Primer].

11.8.1 Classes and Utilities

Table 20 Mapping Classes and Utilities

RDFS Metamodel	[RDF Schema]
RDFSResource	rdfs:Resource
RDFScomment	rdfs:comment
RDFSisDefinedBy	rdfs:isDefinedBy
RDFSlabel	rdfs:label
RDFSmember	rdfs:member
RDFSseeAlso	rdfs:seeAlso
RDFtype	rdf:type
RDFvalue	rdf:value
RDFSClass	rdfs:Class
RDFSsubClassOf	rdfs:subClassOf
RDFSDatatype	rdfs:Datatype
RDFSLiteral	rdfs:Literal
PlainLiteral	rdfs:Literal
TypedLiteral	rdfs:Literal
RDFXMLLiteral	rdf:XMLLiteral

11.8.2 Properties

Table 21 Mapping Properties

RDFS Metamodel	[RDF Schema]
RDFProperty	rdf:Property
RDFSdomain	rdfs:domain
RDFSrange	rdfs:range
RDFSsubPropertyOf	rdfs:subPropertyOf

11.8.3 Containers

Table 22 Mapping Containers

RDFS Metamodel	**[RDF Schema]**
RDFSContainer	rdfs:Container
RDFSContainer MembershipProperty	rdfs:Container MembershipProperty
RDFAlt	rdf:Alt
RDFBag	rdf:Bag
RDFSeq	rdf:Seq

11.8.4 Collections

Table 23 Mapping Collections

RDFS Metamodel	**[RDF Schema]**
RDFList	rdf:List
RDFfirst	rdf:first
RDFrest	rdf:rest

11.8.5 Reification

Table 24 Mapping Reification

RDFS Metamodel	**[RDF Schema]**
RDFStatement	rdf:Statement
RDFpredicate	rdf:predicate
RDFobject	rdf:object
RDFsubject	rdf:subject

11.8.6 Ontology

Table 25 Mapping Description

RDFS Metamodel	[RDF Schema]
Ontology	rdf:RDF

12 The OWL Metamodel

12.1 Overview

The OWL Metamodel is a MOF2 compliant metamodel that allows a user to define ontology models using the same terminology and concepts as those defined in OWL [OWL S&AS] [OWL S&AS]. The OWL Metamodel is an extension of the RDF Schema metamodel.

OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is a language extension of RDF Schema. OWL provides three increasingly expressive sublanguages designed for use by specific communities of users and implementors:

- OWL Lite
  It supports those users primarily needing a classification hierarchy and simple constraint features. For example, OWL Lite only permits cardinality constraint values of 0 or 1.
- OWL DL
  It supports those users who want the maximum expressiveness without losing computational completeness and decidability of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class cannot also be an individual or property, a property cannot also be a class or individual). OWL DL is so named because of its correspondence with description logics.
- OWL Full
  It is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

12.1.1 Organization of the OWL Metamodel

The OWL Metamodel uses diagrams to control complexity and promote understanding.

The classes and associations are grouped and illustrated in the following diagrams:

- Classes
  Contains classes and associations that can be used to define OWL classes.
- Restrictions
  Contains classes and associations that can be used to define OWL restrictions.
- Properties
  Contains classes and associations that can be used to define OWL properties.
- Individuals
  Contains classes and associations that can be used to define OWL individuals.
- Datatypes
  Contains classes and associations that can be used to define OWL datatypes.
- Utilities
  Contains classes and associations that can be used to define utility OWL classes.
- Ontology
  Contains classes and associations that can be used to define the properties of an OWL ontology.

12.1.2 Design Considerations

Metamodel Constructs

Classes defined in OWL are represented by MOF classes. Properties defined in OWL are represented by MOF associations.

Note:

See the corresponding note in the RDF Schema metamodel.

Naming

As in the RDF Schema metamodel, prefixes are used in naming MOF classes and MOF properties that directly represent OWL classes and OWL properties, respectively. For example, OWLClass represents owl:Class and OWLimports represents owl:imports. Instance, which does not have a prefix, represents something which is not explicitly defined in OWL.

12.2 The Classes and Restrictions Diagrams

The Classes diagram of the OWL metamodel is shown in Figure 13. The Restrictions diagram is shown in Figure 14. These two diagrams are logically related and, therefore, are grouped together here for discussion.

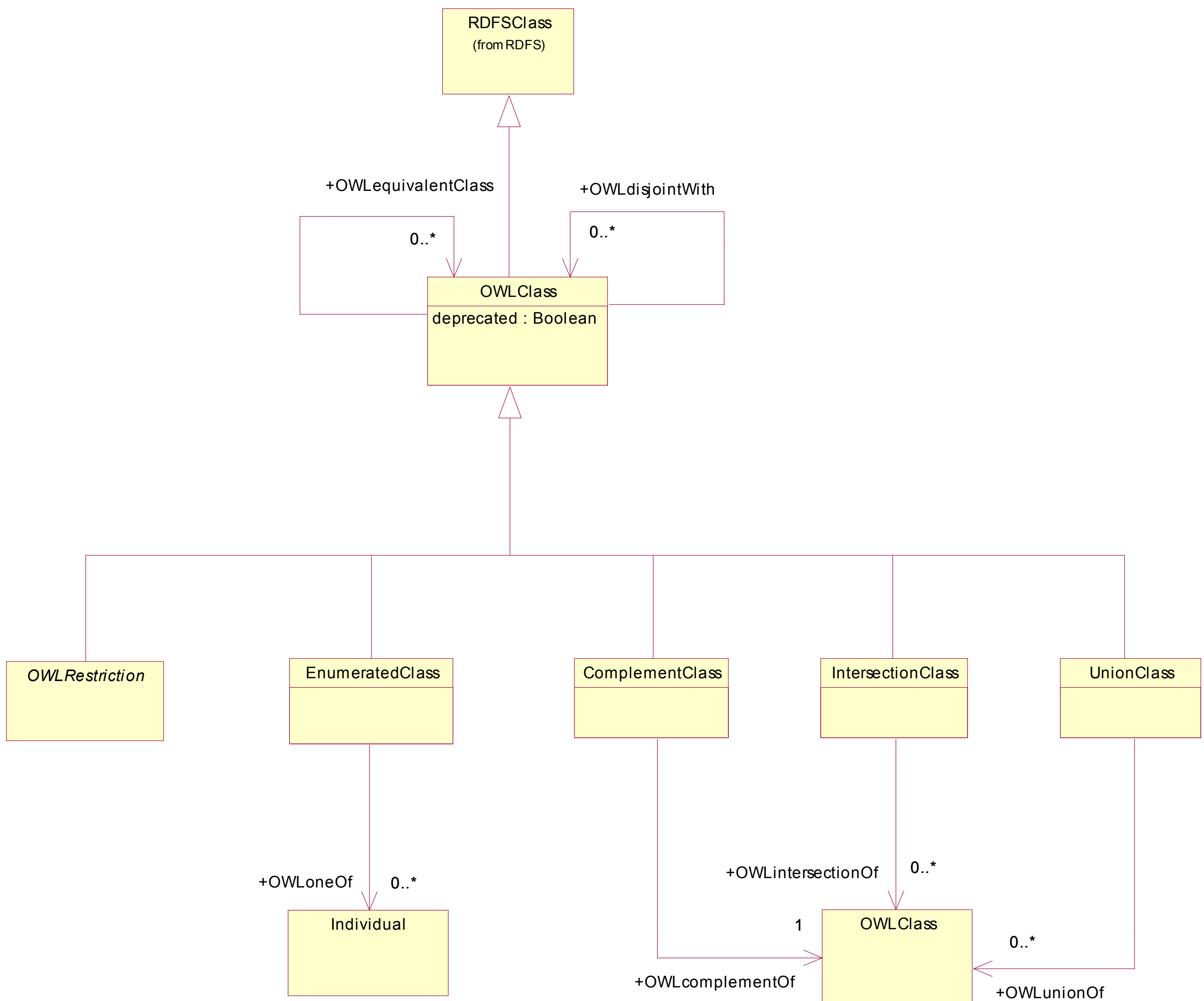


Figure 13 The Classes Diagram of the OWL Metamodel

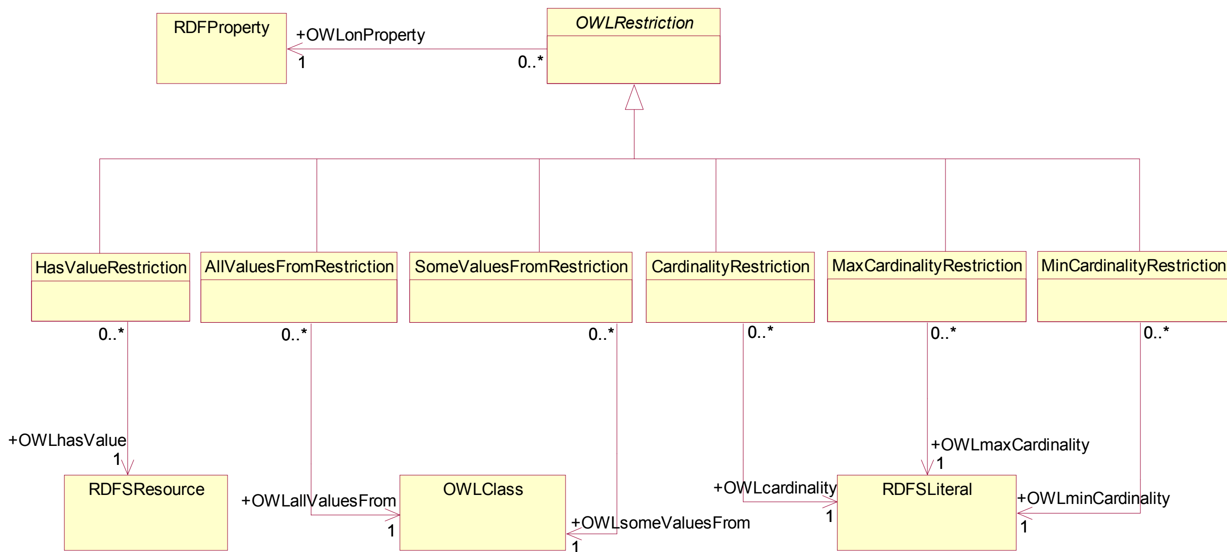


Figure 14 The Restrictions Diagram of the OWL Metamodel

12.2.1 AllValuesFromRestriction

A specific kind of value constraint restriction that describes a class of all individuals for which all values of the property under consideration are members of the class extension of the specified class.

Note: AllValuesFromRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of value constraint restriction that describes a class of all individuals for which all values of the property under consideration are members of the class extension of the specified class.

Attributes

No additional attributes.

Associations

- OWLallValuesFrom: OWLClass [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals for which all values of the property under consideration are members of the class extension of the specified class.

Constraints

No additional constraints.

12.2.2 CardinalityRestriction

A specific kind of cardinality constraint restriction that describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Note: CardinalityRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of cardinality constraint restriction that describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Attributes

No additional attributes.

Associations

- OWLcardinality: RDFSLiteral [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Constraints

- The datatype of the RDFSLiteral for "OWLcardinality" associations must be TypedLiteral, and further must be of type xsd:NonNegativeInteger.

12.2.3 ComplementClass

A class description that describes the complement of another class description.

Note: ComplementClass is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A class description that describes the complement of another class description.

Attributes

No additional attributes.

Associations

- OWLcomplementOf: OWLClass [1]
  This is an instance of RDFProperty that defines this class to be one whose class extension contains exactly those individuals that do not belong to the class extension of the specified class.

Constraints

No additional constraints.

Semantics

See description.

12.2.4 EnumeratedClass

Description

A class description that describes an exhaustive enumeration of individuals.

Note: EnumeratedClass is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Attributes

No additional attributes.

Associations

- OWLoneOf: Individual [0..*]
  This is an instance of RDFProperty that is used to define a class description of the "enumeration" kind. The specified individuals are the instances of this class. This enables a class to be described by exhaustively enumerating its instances.

Constraints

No additional constraints.

Semantics

See description. Note that the elements of an enumerated class are not necessarily unique and no unique names assumption applies.

12.2.5 HasValueRestriction

A specific kind of value constraint restriction that describes a class of all individuals for which the property under consideration has at least one value semantically equivalent to the specified resource.

Note: HasValueRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of value constraint restriction that describes a class of all individuals for which the property under consideration has at least one value semantically equivalent to the specified resource.

Attributes

No additional attributes.

Associations

- OWLhasValue: RDFSResource [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals for which the property under consideration has at least one value semantically equivalent to the specified resource.

Constraints

No additional constraints.

12.2.6 IntersectionClass

A class description that describes the intersection of two or more class descriptions.

Note: IntersectionClass is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A class description that describes the intersection of two or more class descriptions.

Attributes

No additional attributes.

Associations

- OWLintersectionOf: OWLClass [0..*]
  This is an instance of RDFProperty that defines this class to be one whose class extension contains precisely those individuals that are members of the class extension of all specified classes.

Constraints

No additional constraints.

Semantics

See description.

12.2.7 MaxCardinalityRestriction

A specific kind of cardinality constraint restriction that describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Note: MaxCardinalityRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of cardinality constraint restriction that describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Attributes

No additional attributes.

Associations

- OWLmaxCardinality: RDFSLiteral [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Constraints

- The datatype of the RDFSLiteral for “OWLmaxCardinality” association must be TypedLiteral, and further must be of type xsd:NonNegativeInteger.

12.2.8 MinCardinalityRestriction

A specific kind of cardinality constraint restriction that describes a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Note: MinCardinalityRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of cardinality constraint restriction that describes a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Attributes

No additional attributes.

Associations

- OWLminCardinality: RDFSLiteral [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property under consideration, where N is the value of the cardinality constraint.

Constraints

- The datatype of the RDFSLiteral for OWLminCardinality" association must be TypedLiteral, and further must be of type xsd:NonNegativeInteger.

12.2.9 OWLClass

Description

Classes provide an abstraction mechanism for grouping individuals with similar characteristics. Every class is associated with a set of individuals, called the class extension. The individuals in the class extension are called the instances of the class. Classes are described through "class descriptions", which can be combined into "class axioms". There are six types of class descriptions:

1. a class identifier
2. an exhaustive enumeration of individuals
3. a property restriction
4. the complement of a class description
5. the intersection of two or more class descriptions
6. the union of two or more class descriptions

Attributes

- deprecated: Boolean
  This specifies if this class is deprecated. A deprecated class denotes a class that is preserved for backward compatibility purposes, but may be phased out in the future. Deprecated classes should not be used in new instances that commit to the ontology. This allows an ontology to maintain backward compatibility while phasing out an old vocabulary. Deprecation should be used in combination with backward compatibility. That is, in addition to indicating the class (the old version) that has been deprecated, one should further indicate the class (the new version) that should be used in its place (e.g., by specifying an "OWLequivalentClass").

Associations

- OWLdisjointWith: OWLClass [0..*]
  This is an instance of RDFProperty that asserts the class extensions of this class and any of the specified classes have no individuals in common. Declaring two classes to be disjoint is a partial definition: it imposes a necessary

but not sufficient condition on this class.

- OWLequivalentClass: OWLClass [0..*]
  This is an instance of RDFProperty that asserts the specified classes have the same class extension as this class.

Constraints

No additional constraints.

Semantics

The purpose of a class is to provide an abstraction mechanism for classifying individuals. Common characteristics of individuals can be specified on the class using property restrictions.

Individuals of a class do not have to have values for each property that can apply to that class. Also, individuals can be an instance of more than one class.

owl:Thing and owl:Nothing are two predefined instances of owl:Class. owl:Thing represents all individuals and owl:Nothing is the complement of owl:Thing. Since these are at the model level, they are not explicitly represented in the metamodel but are given in the model library provided in Appendix A, Foundation Ontology (M1) for RDFS and OWL.

12.2.10 OWLRestriction

Description

A restriction is a special kind of class. It is frequently, though not always anonymous, and represents a class of all individuals that satisfy certain property restriction.

There are two kinds of property restrictions: value constraints and cardinality constraints. A value constraint puts constraints on the range of the property when applied to this particular class description. A cardinality constraint puts constraints on the number of values a property can take, in the context of this particular class description.

Attributes

No additional attributes.

Associations

- OWLonProperty: RDFProperty
  This is an instance of RDFProperty that indicates that the specified property is the property under consideration.

Constraints

No additional constraints.

Semantics

See Description, above.

12.2.11 SomeValuesFromRestriction

A specific kind of value constraint restriction that describes a class of all individuals for which at least one value of the property under consideration is a member of the class extension of the specified class.

Note: SomeValuesFromRestriction is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A specific kind of value constraint restriction that describes a class of all individuals for which at least one value of the property under consideration is a member of the class extension of the specified class.

Attributes

No additional attributes.

Associations

- OWLsomeValuesFrom: RDFSClass [1]
  This is an instance of RDFProperty that is used to describe a class of all individuals for which at least one value of the property under consideration is a member of the class extension of the specified class.

Constraints

No additional constraints.

12.2.12 UnionClass

A class description that describes the union of two or more class descriptions.

Note: UnionClass is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A class description that describes the union of two or more class descriptions.

Attributes

No additional attributes.

Associations

- OWLunionOf: OWLClass [0..*]
  This is an instance of RDFProperty that defines this class to be one whose class extension contains precisely those individuals that are members of at least one of the specified classes.

Constraints

No additional constraints.

Semantics

See description.

12.3 The Properties Diagram

The Properties diagram of the OWL metamodel is shown in Figure 15.

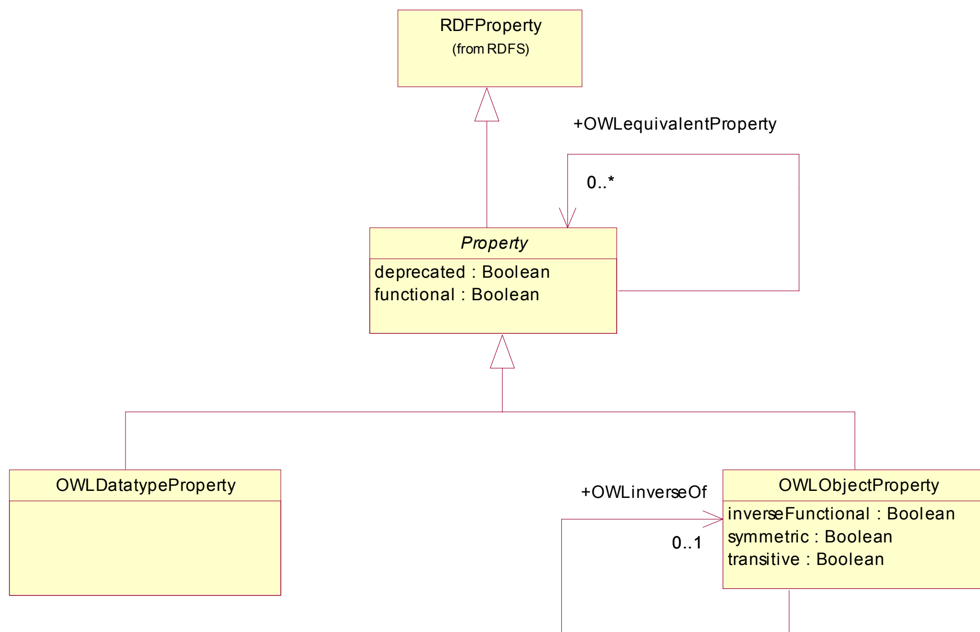


Figure 15 The Properties Diagram of the OWL Metamodel

12.3.1 OWLDatatypeProperty

A datatype property relates individuals to data values.

Description

A datatype property relates individuals to data values. Datatype properties provide relationships between instances of classes and instances of data ranges.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

In OWL DL, datatype properties must be disjoint with object properties, annotation properties, and ontology properties. The values of datatype properties are data values, which may or may not be typed. The values of object properties are individuals.

12.3.2 OWLObjectProperty

An object property relates individuals to individuals.

Description

An object property relates individuals to individuals. Object properties provide relationships between instances of two classes.

Attributes

- inverseFunctional: boolean
  This specifies if this property is inverse functional. An inverse-functional property is an object property whose value uniquely determines some individual. Inverse-functional properties resemble the notion of a key in databases.
- symmetric: boolean
  This specifies if this property is symmetric. A symmetric property is an object property for which holds that if the pair (x, y) is an instance of the property, then the pair (y, x) is also an instance of the property.
- transitive: boolean
  This specifies if this property is transitive. A transitive property is an object property for which holds that if the pair (x, y) is an instance of the property, and the pair (y, z) is also instance of the property, then the pair (x, z) is also an instance of the property.

Associations

- OWLinverseOf: OWLObjectProperty [0..1]
  This is an instance of RDFProperty that states the specified object property is the inverse of this object property. This association is symmetric. That is, if A is an inverse of B, then B is an inverse of A.

Constraints

No additional constraints.

Semantics

In OWL DL, datatype properties must be disjoint with object properties, annotation properties, and ontology properties. The values of datatype properties are data values, which may or may not be typed. The values of object properties are individuals. A symmetric property is its own inverse.

12.3.3 Property

A property relates individuals to data values or individuals.

Description

A property relates individuals to data values or individuals. Property is an abstract class.

Attributes

- deprecated: Boolean
  This specifies if this property is deprecated. A deprecated property denotes a property that is preserved for backward compatibility purposes, but may be phased out in the future. Deprecated properties should not be used in new instances that commit to the ontology. This allows an ontology to maintain backward compatibility while phasing out an old vocabulary. Deprecation should be used in combination with backward compatibility. That is, in addition to indicating the property (the old version) that has been deprecated, one should further indicate the property (the new version) that should be used in its place (e.g., by specifying an "OWLequivalentProperty").
- functional: boolean
  This specifies if this property is functional. A functional property is a property that can have only one (unique) value for each individual.

Associations

- OWLequivalentProperty: OWLDatatypeProperty [0..*]
  This is an instance of RDFProperty that states the specified datatype properties have the same property extension with this datatype property. It is a sub-property of RDFSsubPropertyOf.

Constraints

No additional constraints.

Semantics

See Description.

12.4 The Individuals Diagram

The Individuals diagram of the OWL metamodel is shown in Figure 16.

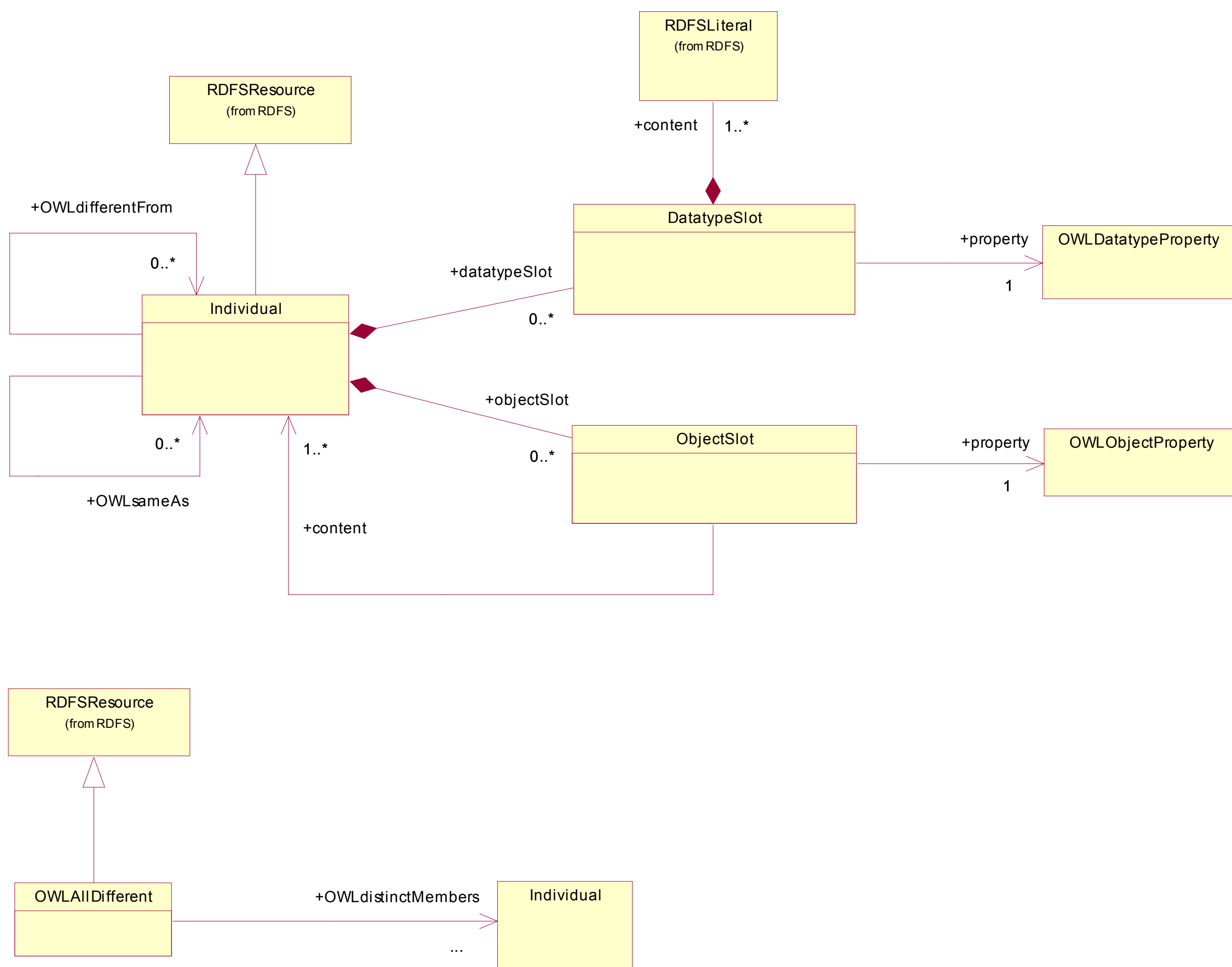


Figure 16 The Individuals Diagram of the OWL Metamodel

12.4.1 DatatypeSlot

A datatype slot of an individual.

Note: DatatypePropertyValue is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

A datatype slot of an individual.

Attributes

No additional attributes.

Associations

- content: RDFLiteral [1..*]
  This identifies the content of this value.

- property: OWLDatatypeProperty
  This identifies the datatype property that this value is for.

Constraints

None.

Semantics

See description.

12.4.2 Individual

An instance of a class.

Note: Individual is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

An instance of a class.

Attributes

No additional attributes.

Associations

- datatypeSlot: DatatypeSlot [0..*]
  This identifies the datatype slots of this individual.

- objectSlot: ObjectSlot [0..*]
  This identifies the object slots of this individual.

- OWLdifferentFrom: Individual [0..*]
  This is an instance of RDFProperty that links this individual to an individual on the specified list and indicates that the two individuals refer to two different things: the individuals are different.

- OWLsameAs: Individual [0..*]
  This is an instance of RDFProperty that links this individual to an individual on the specified list and indicates that the two individuals refer to the same thing: the individuals are identical.

Constraints

The RDFtype of an individual must be an OWLClass.

Semantics

An individual may be an instance of zero or more classes.

12.4.3 ObjectSlot

An object slot of an individual.

Note: ObjectSlot is not an explicit OWL language construct. Therefore, its name is not prefixed with OWL.

Description

An object slot of an individual.

Attributes

No additional attributes.

Associations

- content: Individual [1..*]
  This identifies the content of this value.

- property: OWLObjectProperty
  This identifies the datatype property that this value is for.

Constraints

None.

Semantics

See description.

12.4.4 OWLAllDifferent

A special construct which links an instance of AllDifferent to a list of individuals. The intended meaning is that all individuals in the list are all different from each other.

Description

A special construct which links an instance of AllDifferent to a list of individuals. The intended meaning is that all individuals in the list are all different from each other.

Attributes

No additional attributes.

Associations

- OWLdistinctMembers: OWLThing [2..*]
  This is an instance of RDFProperty that specifies all individuals in the specified list are all different from each other.

Constraints

No additional constraints.

Semantics

AllDifferent provides a convenient way to represent OWLdifferentFrom associations among individuals.

12.5 The Datatypes Diagram

The Datatypes diagram of the OWL metamodel is shown in Figure 17.

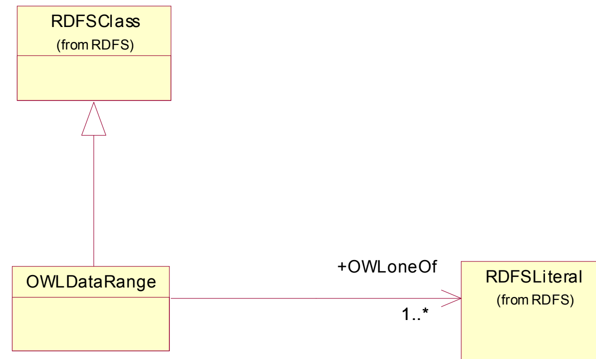


Figure 17 The Datatypes Diagram of the OWL Metamodel

12.5.1 OWLDataRange

A data range represents a range of data values.

Description

A data range represents a range of data values. It can be either a datatype or a set of data values.

Attributes

No additional attributes.

Associations

- OWLoneOf: RDFSLiteral [1..*]
  This is an instance of RDFProperty that specifies the set of data values of this data range.

Constraints

No additional constraints.

Semantics

Data ranges are used to specify the range of datatype properties.

12.6 The Utilities Diagram

The Utilities diagram of the OWL metamodel is shown in Figure 18.

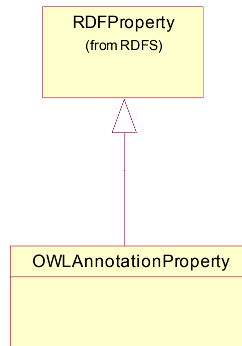


Figure 18 The Utilities Diagram of the OWL Metamodel

12.6.1 OWLAnnotationProperty

An annotation property relates an ontology, a class, or a property to an annotation.

Description

An annotation property relates an ontology, a class, or a property to an annotation. Annotation properties can be predefined or user defined. The following are predefined annotation properties: OWLversionInfo.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

An annotation adds no semantics to the annotated resource, but may represent information useful to the user.

12.7 The Ontology Diagram

The Ontology diagram of the OWL metamodel is shown in Figure 19.

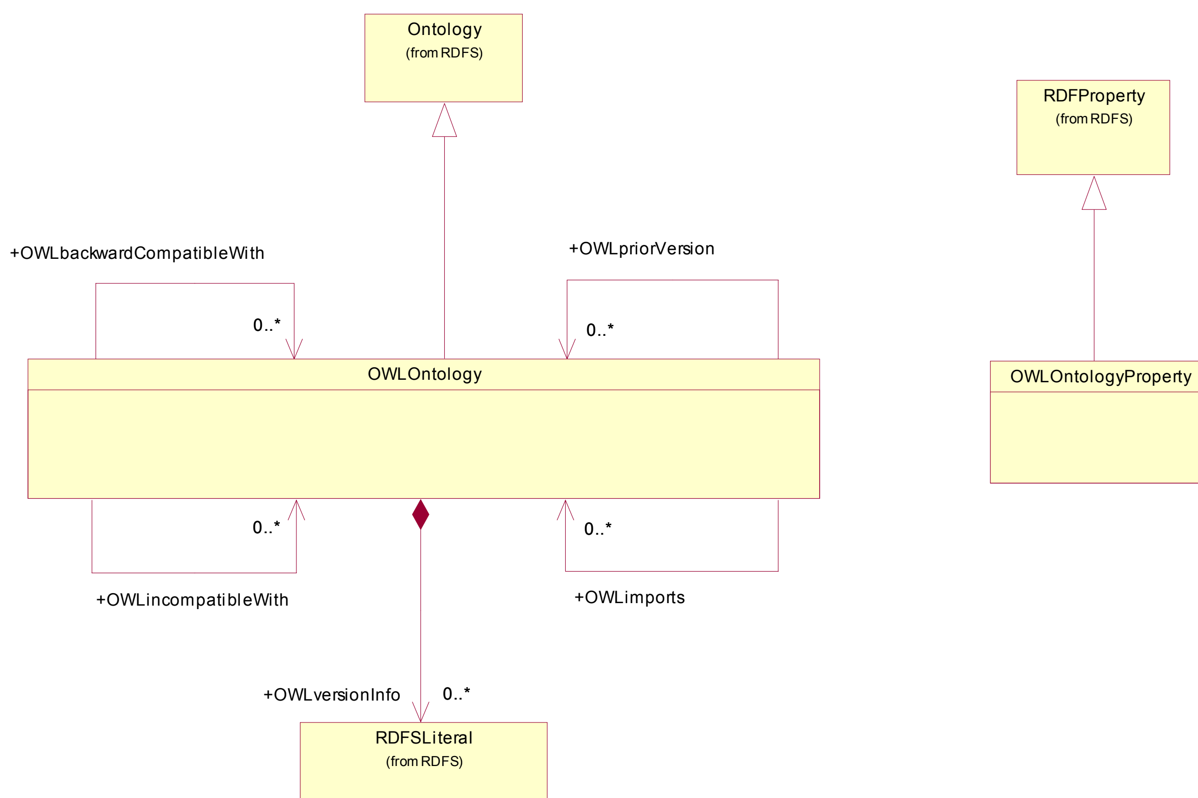


Figure 19 The Ontology Diagram of the OWL Metamodel

12.7.1 OWLOntology

An ontology consists of the various classes and properties that can be used to describe and represent a domain of knowledge.

Description

An ontology consists of the various classes and properties that can be used to describe and represent a domain of knowledge. Classes represent concepts within a domain or across domains, and properties represent the relationships among them.

Attributes

No additional attributes.

Associations

- OWLbackwardCompatibleWith: OWLOntology [0..*]
  This is an instance of OWLOntologyProperty that identifies the specified ontology as a prior version of this ontology, and further indicates that it is backward compatible with the prior version. In particular, this indicates that all identifiers from the prior version have the same intended interpretations in this version.

- OWLimports: OWLOntology [0..*]
  This is an instance of OWLOntologyProperty that identifies the specified ontologies containing definitions whose meanings are considered to be part of the meaning of this ontology. This association is transitive. That is, if ontol-

ogy A imports B, and B imports C, then A imports both B and C. Importing an ontology into itself is considered a null action, so if ontology A imports B and B imports A, then they are considered to be equivalent.

- OWLincompatibleWith: OWLOntology [0..*]
  This is an instance of OWLOntologyProperty that indicates this ontology is a later version of the specified ontology, but is not backward compatible with it.
- OWLpriorVersion: OWLOntology [0..*]
  This is an instance of OWLOntologyProperty that identifies the specified ontology as a prior version of this ontology.
- OWLversionInfo: RDFSLiteral [0..*]
  This is an instance of OWLAnnotationProperty that indicates the version information of this ontology.

Constraints

- For any two ontologies, only one of the two associations OWLincompatibleWith and OWLbackwardCompable-With may relate them together.

Semantics

An ontology provides a container and a namespace for resources. If an ontology is removed, so are the resources owned by it.

12.7.2 OWLOntologyProperty

An ontology property relates ontologies to other ontologies.

Description

Ontology properties relate ontologies to other ontologies. The following are built-in ontology properties: OWLbackwardCompatibleWith, OWLimports, OWLincompatibleWith, and OWLpriorVersion.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

- The uri of an ontology property cannot be the same as that of an annotation property, a datatype property, or an object property.

Semantics

Except for built-in ontology properties (OWLbackwardCompatibleWith, OWLimports, OWLincompatibleWith, and OWLpriorVersion), the semantics of ontology properties are user defined.

12.8 Language Mappings

This section specifies the mappings of the constructs in the OWL metamodel to elements of two OWL language syntax:

- RDF/XML syntax [OWL S&AS]

- XML Presentation syntax [OWL XML Syntax], which is defined as a dialect similar to OWL Abstract Syntax [OWL S&AS] *Note*: [OWL XML Syntax] is slightly out-of-date compared to [OWL S&AS].

12.8.1 Classes and Restrictions

Table 26 Classes and Restrictions in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
OWLClass	owl:Class	owlx:Class
deprecated (attribute)	owl:DeprecatedClass	deprecated (attribute)
OWLdisjointWith	owl:disjointWith	owlx:DisjointClasses
OWLequivalentClass	owl:equivalentClass	owlx:EquivalentClasses
EnumeratedClass	n/a	owlx:EnumeratedClass
OWLoneOf	owl:oneOf	owlx:OneOf
ComplementClass	n/a	n/a
OWLcomplementOf	owl:complementOf	owlx:ComplementOf
IntersectionClass	n/a	n/a
OWLintersectionOf	owl:intersectionof	owlx:IntersectionOf
UnionClass	n/a	n/a
OWLunionOf	owl:unionOf	owlx:UnionOf
OWLRestriction (abstract)	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLonProperty	owl:onProperty	property (attribute)
AllValuesFrom-Restriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLallValuesFrom	owl:allValuesFrom	owlx:allValuesFrom
HasValueRestriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLhasValue	owl:hasValue	owlx:hasValue
SomeValuesFrom-Restriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLsomeValuesFrom	owl:someValuesFrom	owlx:someValuesFrom
CardinalityRestriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLcardinality	owl:cardinality	owlx:cardinality

Table 26 Classes and Restrictions in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
MaxCardinality-Restriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLmaxCardinality	owl:maxCardinality	owlx:maxCardinality
MinCardinality-Restriction	owl:Restriction	owlx:DataRestriction owlx:ObjectRestriction
OWLminCardinality	owl:minCardinality	owlx:minCardinality

12.8.2 Properties

Table 27 Properties in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
OWLDatatypeProperty	owl:DatatypeProperty	owlx:DatatypeProperty
deprecated (attribute)	owl:DeprecatedProperty	deprecated (attribute)
functional (attribute)	owl:FunctionalProperty	functional (attribute)
OWLequivalentProperty	owl:equivalentProperty	owlx:EquivalentProperties
OWLObjectProperty	owl:ObjectProperty	owlx:ObjectProperty
deprecated (attribute)	owl:DeprecatedProperty	deprecated (attribute)
functional (attribute)	owl:FunctionalProperty	functional (attribute)
inverseFunctional (attribute)	owl:InverseFunctionalProperty	inverseFunctional (attribute)
symmetric (attribute)	owl:SymmetricProperty	symmetric (attribute)
transitive (attribute)	owl:TransitiveProperty	transitive (attribute)
OWLequivalentProperty	owl:equivalentProperty	owlx:EquivalentProperties
OWLinverseOf	owl:inverseOf	inverseOf (attribute)

12.8.3 Individuals

Table 28 Individuals in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
Individual	n/a	owlx:Individual
OWLdifferentFrom	owl:differentFrom	owlx:DifferentIndividuals
OWLsameAs	owl:sameAs	owlx:SameIndividuals
DatatypeSlot	n/a	owlx:DataPropertyValue
ObjectSlot	n/a	owlx:ObjectPropertyValue
OWLAllDifferent	owl:AllDifferent	owlx:DifferentIndividuals
OWLdistinctMembers	owl:distinctMembers	n/a

12.8.4 Datatypes

Table 29 Datatypes in the OWL Syntaxes

OWL Metamodel	[OWLReference]	[OWL XML Syntax]
OWLDataRange	owl:DataRange	owlx:OneOf

12.8.5 Utilities

Table 30 Utilities in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
OWLAnnotationProperty	owl:AnnotationProperty	n/a

12.8.6 Ontology

Table 31 Ontology in the OWL Syntaxes

OWL Metamodel	[OWL Reference]	[OWL XML Syntax]
OWLOntology	rdf:RDF owl:Ontology	owlx:Ontology
OWLbackwardCompatibleWith	owl:backwardCompatibleWith	owlx:BackwardCompatibleWith
OWLimports	owl:imports	owlx:Imports
OWLincompatibleWith	owl:incompatibleWith	owlx:IncompatibleWith
OWLpriorVersion	owl:priorVersion	owlx:PriorVersion
OWLversionInfo	owl:versionInfo	owlx:VersionInfo
OWLOntologyProperty	owl:OntologyProperty	n/a

13 The Common Logic Metamodel

13.1 Overview

Common Logic (CL) is a first-order logical language intended for information exchange and transmission over an open network [ISO 24707]. It allows for a variety of different syntactic forms, called dialects, all expressible within a common XML-based syntax and all sharing a single semantics. The language has declarative semantics, which means that it is possible to understand the meaning of expressions written in CL without requiring an interpreter to manipulate those expressions. CL is logically comprehensive – at its most general, it provides for the expression of arbitrary logical expressions. CL has a purely first-order semantics, and satisfies all the usual semantic criteria for a first-order language, such as compactness and the downward Skolem-Löwenheim property.

Motivation for its consideration as an integral component of the Ontology Definition Metamodel (ODM) includes:

- The potential need by ontologists using the ODM to be able to represent constraints and rules with expressiveness beyond that supported by description logics (*e.g.*, for composition of semantic web services), as highlighted in Chapter 7, Usage Scenarios and Goals.
- The availability of normative mappings from CL to syntactic forms for several commonly used knowledge representation standards, defined in [ISO 24707], including the Knowledge Interchange Format [KIF] and Conceptual Graphs [CGS].
- The availability of a normative XML-based surface syntax for CL, called XCL (also defined in [ISO 24707]), which dramatically increases its potential for use in web-based applications.
- The availability of a direct mapping from the Web Ontology Language (OWL)[OWL S&AS] to CL, such that CL reasoners can leverage both the ontologies expressed in OWL and constraints written in CL to solve a wider range of problems than can be addressed by OWL alone (see Chapter 21, Mapping RDFS and OWL to CL).

In general, first order logic provides the basis for most commonly used knowledge representation languages, including relational databases; more application domains have been formalized using first order logic than any other formalism – its meta-mathematical properties are thoroughly understood. CL in particular provides a modern form of first order logic that takes advantage of recent insights in some of these application areas including the semantic web.

First order logic can also provide the formal grounding for business semantics. Although work on the OMG's Business Semantics For Business Rules (BSBR) RFP has been done in parallel with the ODM to date, recently, there has been significant effort to leverage CL as the first order logic basis for the Semantics of Business Vocabulary and Business Rules (SBVR) submission to this RFP. This revision of the specification supports irregular sentences, a recent addition to the abstract syntax of CL required for the SBVR modality representations, for example. Subsequent versions of both specifications will be amended to accommodate additional interoperability requirements to the extent possible.

13.1.1 Design Considerations

The CL Metamodel is defined per [ISO 24707], and was developed with the help of the CL language authors to be a comprehensive and accurate representation of the abstract syntax of CL. As indicated in Chapter 8, Design Rationale, a decision was made not to depend on the OCL 2.0 Metamodel specifically because such a dependency would introduce unnecessary complexity and semantics that may be inconsistent with the simplicity, efficiency, and formal semantics of CL. Inconsistencies in the semantics can have unintended consequences for downstream reasoning, limiting the utility of an ODM-based application that leverages the CL metamodel. A mapping between CL and OCL may be considered in subsequent versions of the ODM if requirements for such a mapping are identified. Such a mapping would require validation through the use of CL and OCL-based reasoning engines, which may not be available prior to finalization of this specification.

To date, although a number of proposals have been put forth to the W3C for a rule language for OWL, there is no formal recommendation available from the W3C today. Such a standard may be considered for integration with, or as an additional candidate for mapping to, the CL metamodel through a subsequent RFP.

The complete syntax and formal semantics for CL are documented in [ISO 24707] and are considered essential to understanding the expressions that might be imported, managed, manipulated, or generated by any ODM/CL-compliant tool.

13.1.2 Modeling Notes

All of the OCL constraints documented below have been validated using OCL tools.

13.2 The Phrases Diagram

Phrases provide mechanisms for grouping and scoping the elements that constitute an ontology (or set of constraints associated with an OWL ontology), authored in Common Logic or any of its syntactic variants. An overview of the top-level elements of the CL metamodel is provided in Figure 20.

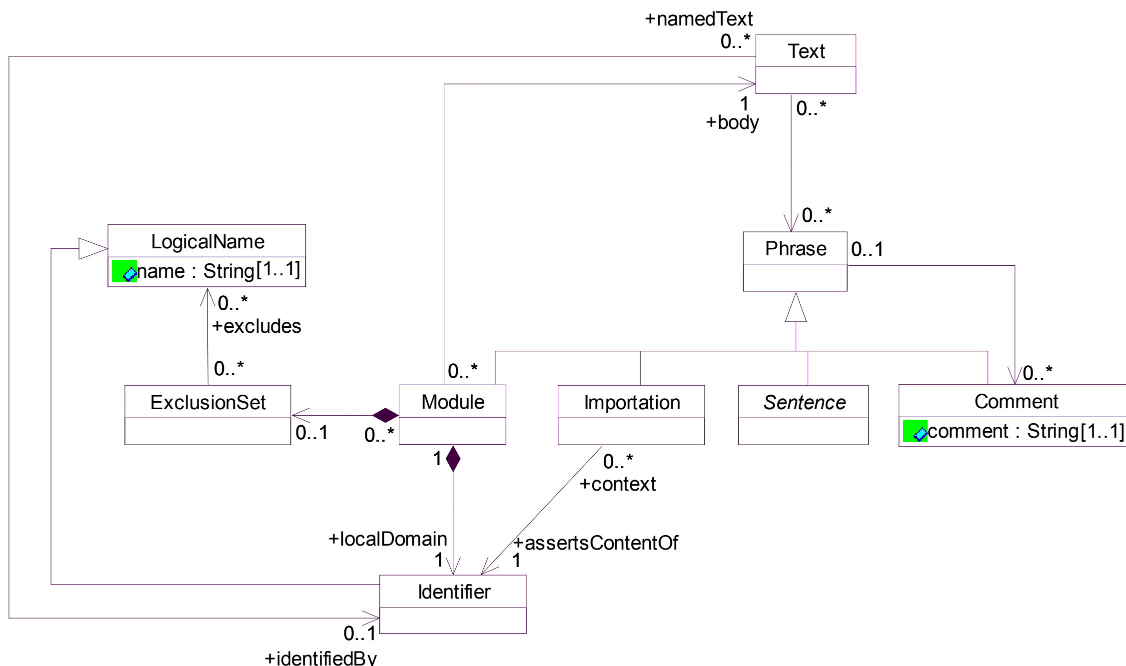


Figure 20 Phrases

13.2.1 Comment

Description

A Comment provides the facility for commenting on a particular phrase or set of sentences.

Attributes

- comment: String [1..1] – the character string that is the comment on the phrase

Associations

- commentedPhrase: Phrase [0..1] in association CommentedPhrase – the phrase about which the comment applies
- Specialize Class Phrase

Constraints

None.

Semantics

None.

13.2.2 ExclusionSet

Description

A module may optionally have an exclusion list of names whose denotations are considered to be excluded from the domain (*i.e.*, while the names may be considered part of the universe of discourse, they are not considered part of the local domain of discourse).

Attributes

None.

Associations

- excludes: LogicalName [0..*] in association ExcludedNames – the names that are members of the ExclusionSet
- module: Module [0..*] in association ModuleExcludes – the module(s) that excludes this set of names

Constraints

None.

Semantics

An ExclusionSet essentially represents some set of non-denoting names as they relate to a particular domain of discourse. See [ISO 24707] for additional detail.

13.2.3 Identifier

Description

Dialects intended for transmission of content on a network should not impose arbitrary or unnecessary restrictions on the form of names, and must provide for names to be used as identifiers of common logic texts. An identifier is a name explicitly used to identify a module or piece of common logic text.

Attributes

None.

Associations

- context: Importation [0..*] in association NameForImportation – links an identifier to an importation that references it
- module: Module [1] in association ModuleName – links an identifier to the module it names

- namedText: Text [0..1] in association NameForText – links an identifier to the text it names
- Specialize Class LogicalName

Constraints

None.

Semantics

Names used to name texts on a network are understood to be *rigid* and to be *global* in scope, so that the name can be used to identify the thing named – in this case, the Common Logic text or module – across the entire communication network. (See [RFC2396] for further discussion.) A name which is globally attached to its denotation in this way is an *identifier*, and is typically associated with a system of conventions and protocols which govern the use of such names to identify, locate and transmit pieces of information across the network on which the dialect is used. While the details of such conventions are beyond the scope of this specification, we can summarize their effect by saying that the act of publishing a named Common Logic text (or module) is intended to establish the name as a rigid identifier of the text, and Common Logic acknowledges this by requiring that *all* interpretations shall conform to such conventions when they apply to the network situation in which the publication takes place.

Note that in the case of an importation, the name serves to identify the module, which is accomplished through a double interpretation in the semantics. The 'import' condition is that (import x) is true in I just when I(I(x)) is true. In other words, interpreting an identifier gets what it denotes. If the name happens to be an CL ontology (I(x) is an ontology), then interpreting it again I(I(x)) returns a truth-value; thus, (import x) says that x is an ontology which *this* ontology (the one doing the importing) asserts to be true.

13.2.4 Importation

Description

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.

Attributes

None.

Associations

- assertsContentOf: Identifier [1] in association NameForImportation – the name of the module to be imported; the name argument of an importation will usually be a URI.
- Specialize Class Phrase

Constraints

None.

Semantics

An import construction requires that we assume the existence of a global module-naming convention, and that module names refer to entities in formal interpretations. Common Logic uses the same semantic web conventions used in RDF and OWL, based on W3C recommendation for representing namespaces in XML [XMLNS]. The meaning of an importation phrase is that the name it contains shall be understood to identify some Common Logic content, and the importation is true just when that content is true. Thus, an importation amounts to a virtual ‘copying’ of some Common Logic content from one ‘place’ to another. This idea of ‘place’ and ‘copying’ can be understood

only in the context of deploying logical content on a communication network. A *communication network*, or simply a *network*, is a system of agents which can store, publish or process common logic text, and can transmit common logic text to one another by means of information transfer protocols associated with the network.

13.2.5 LogicalName

Description

A *logical name* is any lexical token, or character string, which is understood to denote something in the domain of discourse. Part of the design philosophy of CL is to avoid syntactic distinctions between name types, allowing ontologies freedom to use names without requiring mechanisms for syntactic alignment. Names are primitive referring elements in CL, and refer to elements of a particular ontology, such as module names, role names, relations, or numerals.

Dialects intended for use on the Web should allow Universal Resource Identifiers and URI references [RDF Syntax] to be used as names. Common Logic dialects should define names in terms of Unicode [ISO 10646] conventions.

Attributes

- name: String [1..1] – the character string symbolizing the name

Associations

- exclusionSet: ExclusionSet [0..*] in association ExcludedNames – the optional exclusion list referring to the name
- quantifiedSentence: QuantifiedSentence [0..1] in association NameBoundByQuantifier – associates an optional name (variable) in quantified sentences
- Specialize Class Term

Constraints

[1] The lexical syntax for several CL dialects identifies a number of rules for specifying valid names that cannot be expressed in OCL, and are thus delegated to CL parsers (such as identification of special characters that cannot be embedded in names, the requirement for conformance to Unicode conventions, additional constraints on logical names that are URIs or URI references, and so forth).

[2] LogicalName and SequenceVariable are mutually exclusive.

Semantics

The only undefined terms in the CL are name and sequence variable. The only required constraint on these is that they must be exclusive. Common Logic does not require names to be distinguished from variables, nor does it require names to be partitioned into distinct classes such as relation, FunctionalTerm or individual names, or impose sortal restrictions on names. Particular Common Logic dialects may make these or other distinctions between subclasses of names, and impose extra restrictions on the occurrence of types of names or terms in expressions - for example, by requiring that bound names be written with a special variable prefix, as in KIF, or with a particular style, as in Prolog; or by requiring that operators be in a distinguished category of relation names, as in conventional first-order syntax.

A dialect may impose particular semantic conditions on some categories of names, and apply syntactic constraints to limit where such names occur in expressions. For example, the core syntax treats numerals as having a fixed denotation, and prohibits their use as identifiers. A dialect may require some names to be non-denoting names. This requirement may be imposed by, for example, partitioning the vocabulary, or by requiring names which occur in certain syntactic positions to be non-denoting. A dialect with non-denoting names is called segregated.

13.2.6 Module

Description

A module consists of a name, an optional set of names called an *exclusion set*, and a text called the *body text*. The module name indicates the local domain of discourse in which the text is understood; the exclusion list indicates any names in the text which are excluded from the local domain (*i.e.*, variables whose scope is external to the local domain). A module has a global name in the form of a Uniform Resource Identifier [RDF Syntax] or a URI reference.

Attributes

None.

Associations

- body: Text [1] in association ModuleBody – the body, or set of phrases, that are contained in the module
- exclusionSet: ExclusionSet [0..1] in association ModuleExcludes – the optional set of names, or exclusion list, associated with a given module
- localDomain: Identifier [1] in association ModuleName – the logical name associated with a module (for most applications, particularly those that are web based, module names must be unique)
- Specialize Class Phrase

Constraints

In cases where CL is used to define ontologies for the Web, module names take the form of Uniform Resource Identifiers [RDF Syntax] or URI references, and are global (thus must be unique).

Semantics

A module provides the scoping mechanism for an CL ontology, corresponding to an RDF graph [RDF Primer] or document, or to an OWL ontology. The name of a module should be the name of the corresponding RDF document in cases where CL constraints are associated with an RDFS/OWL ontology, and has the same URI or URI reference (*i.e.*, that of the RDFS/OWL ontology).

The CL syntax provides for modules to state an intended domain of discourse, to relate modules explicitly to other domains of discourse, and to express intended restrictions on the syntactic roles of symbols. This feature is critical to component-based ontology (or micro-theory) construction, and therefore relevant to any MDA-based authoring environment.

13.2.7 Phrase

Description

A phrase is either a comment, or a module, or a sentence, or an importation, or a phrase with an attached comment.

Attributes

None.

Associations

- commentForPhrase: Comment [0..*] in association CommentedPhrase – optional comment(s) associated with the phrase
- text: Text [0..*] in association PhraseForText – the text(s) in which the phrase occurs

Constraints

[1] Module, Importation, Sentence, and Comment are specializations of Phrase and form a disjoint partition, as follows:

```
context Phrase inv XOR:
   (self.oclIsKindOf(Module) xor self.oclIsKindOf(Importation)) and
   (self.oclIsKindOf(Module) xor self.oclIsKindOf(Sentence)) and
   (self.oclIsKindOf(Module) xor self.oclIsKindOf(Comment)) and
   (self.oclIsKindOf(Importation) xor self.oclIsKindOf(Sentence)) and
   (self.oclIsKindOf(Importation) self.oclIsKindOf(Comment)) and
   (self.oclIsKindOf(Sentence) xor self.oclIsKindOf(Comment))
```

Semantics

Sequence variables take Common Logic beyond first-order expressiveness. A sequence variable stands for an arbitrary sequence of arguments. Since sequence variables are implicitly universally quantified, any expression containing a sequence variable has the same semantic import as the *infinite* conjunction of all the expressions obtained by replacing the sequence variable by a finite sequence of names, all universally quantified at the top (phrase) level. Significant detail is provided in [ISO 24707] on the use of such variables and their scope.

13.2.8 Sentence

Description

CL provides facilities for expressing several kinds of sentences, including atomic sentences as well as compound sentences built up from atomic sentences or terms with a set of logical constructors. CL sentences are Phrases, as stated above and as shown in Figure 20.

The convention used in CL for expressing sentences differs from the approach taken in the informative DL metamodel. In the DL case, constructors are uniquely defined, whereas in CL the constructors are an integral part of the sentence, named for the kind of construction used in the sentence.

Attributes

None.

Associations

- biconditional: Biconditional [0..1] in association LvalueForBiconditional – associates a sentence as the “lvalue” (or left-hand side) of an Biconditional or biconditional relation
- biconditional: Biconditional [0..1] in association RvalueForBiconditional – associates a sentence as the “rvalue” (or right-hand side) of an Biconditional or biconditional relation
- comment: CommentedSentence [0..1] in association CommentForSentence – provides the facility for commenting any given CL sentence
- conjunction: Conjunction [0..1] in association Conjunction – associates a sentence to its conjuncts in a conjunction
- disjunction: Disjunction [0..1] in association Disjunction – associates a sentence to its disjuncts in a disjunction
- implication: Implication [0..1] in association AntecedentForImplication – associates a sentence as the antecedent of an implication
- implication: Implication [0..1] in association ConsequentForImplication – associates a sentence as the consequent of an implication

- negation: Negation [0..1] in association NegationSentence – associates a sentence with a negation
- quantification: QuantifiedSentence [0..1] in association QuantificationForSentence – associates a sentence (body) with a quantifier and optional bound variables
- Specialize Class Phrase

Constraints

The partition formed by the subclasses of Sentence is disjoint:

```
context Sentence inv DisjointPartition:
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(Disjunction)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(Negation)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(Implication)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(Biconditional)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(UniversalQuantification)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(Conjunction) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(Negation)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(Implication)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(Biconditional)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(UniversalQuantification)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(Disjunction) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(Implication)) and
    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(Biconditional)) and
    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(UniversalQuantification)) and
    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(Negation) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(Implication) xor self.oclIsKindOf(Biconditional)) and
    (self.oclIsKindOf(Implication) xor self.oclIsKindOf(UniversalQuantification)) and
    (self.oclIsKindOf(Implication) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(Implication) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(Implication) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(Biconditional) xor self.oclIsKindOf(UniversalQuantification)) and
    (self.oclIsKindOf(Biconditional) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(Biconditional) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(Biconditional) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(UniversalQuantification) xor self.oclIsKindOf(ExistentialQuantification)) and
    (self.oclIsKindOf(UniversalQuantification) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(UniversalQuantification) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(ExistentialQuantification) xor self.oclIsKindOf(Atom)) and
    (self.oclIsKindOf(ExistentialQuantification) xor self.oclIsKindOf(CommentedSentence)) and

    (self.oclIsKindOf(Atom) xor self.oclIsKindOf(CommentedSentence))
```

Semantics

The semantics of Common Logic is defined in terms of a satisfaction relation between Common Logic text and structures called *interpretations*. All dialects **must** apply these semantic conditions to all common logic expressions, that is, to any of the forms given in the abstract syntax. They **may** in addition apply further semantic conditions to subclasses of common logic expressions, or to other expressions.

A vocabulary is a set of names and sequence variables. The vocabulary of a Common Logic text is the set of names and sequence variables which occur in the text. In a segregated dialect, vocabularies are partitioned into denoting names and non-denoting names.

An interpretation I of a vocabulary V is a set U_I , the *universe*, with a distinguished nonempty subset D_I, the domain of discourse, or simply *domain*, and four mappings: rel_I from U_I to subsets of D_I*, fun_I from U_I to FunctionalTerms D_I*->D_I, (which we will also consider to be the set D_I* x D_I), int_I from names in V to U_I, and seq_I from sequence variables in V to D_I*. If the dialect is segregated, then int_I(x) is in D_I if and only if x is a denoting name. If the dialect recognizes irregular sentences, then they are treated as names of propositions, and int_I also includes a mapping from the irregular sentences of a text to the truthvalues {true, false}.

Intuitively, D_I is the domain of discourse containing all the individual things the interpretation is 'about' and over which the quantifiers range. U_I is a potentially larger set of things which might also contain entities which are not in the universe of discourse. All names are interpreted in the same way, whether or not they are understood to denote something in the domain of discourse; this is why there is only a single interpretation mapping applying to all names regardless of their syntactic role. In particular, rel_I(x) is in D_I* even when x is not in D. When considering only segregated dialects, the universe outside the domain may be considered to contain names and can be ignored; when considering only unsegregated dialects, the distinction between universe and domain is unnecessary. The distinction is required in order to give a uniform treatment of all dialects. Irregular sentences are treated as though they were arbitrary propositional variables.

Additional details of the semantics of Common Logic are given in [ISO 24707].

13.2.9 Text

Description

Text is a collection of phrases, each of which is a sentence, a module, an importation, or a comment. A module is a named piece of text with an optional exclusion set containing names considered to be outside the domain of discourse for the module.

Attributes

None.

Associations

- identifiedBy: Identifier [0..1] in association NameForText – links a text with an identifier in a named text
- phrase: Phrase [0..*] in association PhraseForText – the phrase(s) or sentence(s) that comprise the text
- moduleForBody: Module [0..*] in association ModuleBody – the module(s) owning the text

Constraints

None.

Semantics

The semantics of CL is defined conventionally in terms of a *satisfaction* relation between CL text and structures called *interpretations*. A discussion of the semantics regarding text interpretation is given in the ISO 24707 Common Logic specification, including distinctions in quantifier scope, features enabling structured relationships among modules, closed-world and unique naming issues, and so forth.

13.3 The Terms Diagram

The Terms Diagram, shown in Figure 21 provides additional insight into the core syntactic elements of Common Logic. These include names, commented terms, and term sequences (FunctionalTerms).

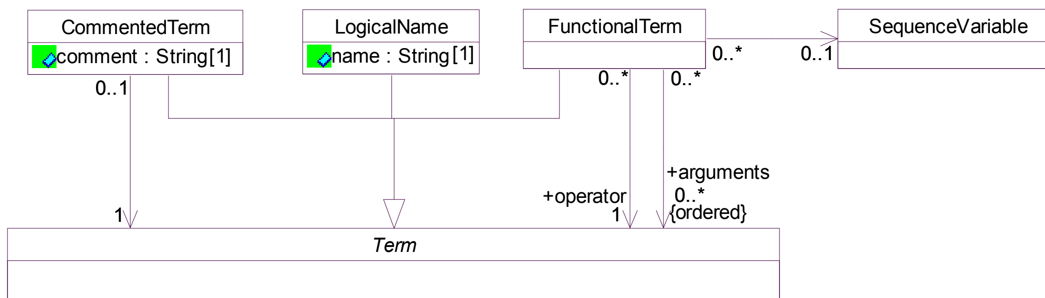


Figure 21 Valid Terms in CL

13.3.1 CommentedTerm

Description

Terms may have an attached comment.

Attributes

- comment: String [1] – supports comments on individual terms (or names)

Associations

- term: Term [1] in association CommentForTerm – links the comment to the term
- Specialize Class Term

Constraints

None.

Semantics

None.

13.3.2 FunctionalTerm

Description

A FunctionalTerm consists of a term, called the *operator*, and a term sequence called the *argument sequence*, containing terms called *arguments*.

Attributes

None.

Associations

- arguments: Term [0..*] in association ArgumentSequence – links zero or more additional terms (*i.e.*, arguments) to a functional term
- operator: Term [1] in association OperatorForTerm – links an operator to a functional term
- sequenceVariable: SequenceVariable [0..1] in association SequenceVariableForFunction – augments the argument list for the functional term with an optional sequence variable
- Specialize Class Term

Constraints

The argument sequence of a functional term is ordered.

Semantics

See additional discussion of the semantics of functional term in CL in [ISO 24707].

13.3.3 SequenceVariable

Description

A term sequence is a finite sequence of terms and an optional sequence variable. A sequence may be empty or may consist of a single sequence variable. Only one sequence variable may occur in a term sequence. Finite sequences play a central role in CL syntax and semantics. Atomic sentences consist of an application of one term, denoting a relation, to a finite sequence of other terms. Such argument sequences may be empty, but they must be present in the syntax, as an application of a relation term to an empty sequence does not have the same meaning as the relation term alone.

Attributes

None.

Associations

- function: FunctionalTerm [0..1] in association SequenceVariableForFunction – links an optional sequence variable to the functional term
- relation: AtomicSentence [0..1] in association SequenceVariableForRelation – links an optional sequence variable to the relation or atomic sentence

Constraints

None.

Semantics

Sequence variables take Common Logic beyond first-order expressiveness. A sequence variable stands for an arbitrary sequence of arguments. Since sequence variables are implicitly universally quantified, any expression containing a sequence variable has the same semantic import as the *infinite* conjunction of all the expressions obtained by replacing the sequence variable by a finite sequence of names, all universally quantified at the top (phrase) level.

This ability to represent infinite sets of sentences in a finite form means that Common Logic with sequence variables is not compact, and therefore not first-order; for clearly the infinite set of sentences corresponding in meaning to a simple atomic sentence containing a sequence variable is logically equivalent to that sentence and so entails it, but no

finite subset of the infinite set does. However, the intended use of sentences containing sequence variables is to act as axiom schemata, rather than being posed as goals to be proved, and when they are restricted to this use the resulting logic is compact. Also, even without this restriction, Common Logic is finitely complete, in the sense there are inference schemes which can derive T from S if S entails T and S is finite. Since Common Logic sentences can express the same content as infinite sets of conventional first-order sentences, the limitation to finite antecedents is less restrictive than it might seem; in fact, this completeness is a strengthening of Gödel's classical first-order completeness result.

A CL dialect which provides only some of the common logic forms may be described as a *partial* common logic dialect, or as *partially conformant*. In particular, a dialect which does not provide for term sequences with a sequence variable, but is otherwise fully conformant, is a *compact* dialect, and may be described as a *fully conformant compact dialect* if it provides for all other constructions in the abstract syntax. Additional discussion on the semantics and use of sequence variables in CL is provided in the [ISO 24707] Common Logic specification.

13.3.4 Term

Description

A term is either a name or a functional term, or a term with an attached comment. A term sequence is a finite sequence of terms and an optional sequence variable. A sequence may be empty or may consist of a single sequence variable. Only one sequence variable may occur in a term sequence.

Attributes

None.

Associations

- atomicSentence: AtomicSentence [0..*] in association ArgumentsForAtomicSentence – links an argument sequence to the relation that the arguments participate in
- atomicSentence: AtomicSentence [0..*] in association PredicateForAtomicSentence – links a predicate to the relation that it is a part of
- commentedTerm: CommentedTerm [0..1] in association CommentForTerm – provides the facility for commenting any CL term
- function: FunctionalTerm [0..*] in association ArgumentSequence – links an argument sequence to the function that the arguments participate in
- function: FunctionalTerm [0..*] in association OperatorForFunction – links an operator to the FunctionalTerm that it is a part of
- equation: Equation [0..*] in association LvalueForIdentity – links the term representing the 'lvalue' to an equation
- equation: Equation [0..*] in association RvalueForIdentity – links the term representing the 'rvalue' to an equation

Constraints

The LogicalName / CommentedTerm / FunctionalTerm partition is disjoint.

```
context Term inv DisjointPartion:
    (self.oclIsKindOf(LogicalName) xor self.oclIsKindOf(CommentedTerm)) and
    (self.oclIsKindOf(LogicalName) xor self.oclIsKindOf(FunctionalTerm)) and
    (self.oclIsKindOf(CommentedTerm) xor self.oclIsKindOf(FunctionalTerm))
```

Semantics

See additional discussion of the semantics of terms in CL in [ISO 24707].

13.4 The Atoms Diagram

Atomic sentences are similar in structure to terms, as shown in Figure 22. Equations are considered to be atomic sentences. Equations are distinguished as a special category because of their special semantic role and special handling by many applications.

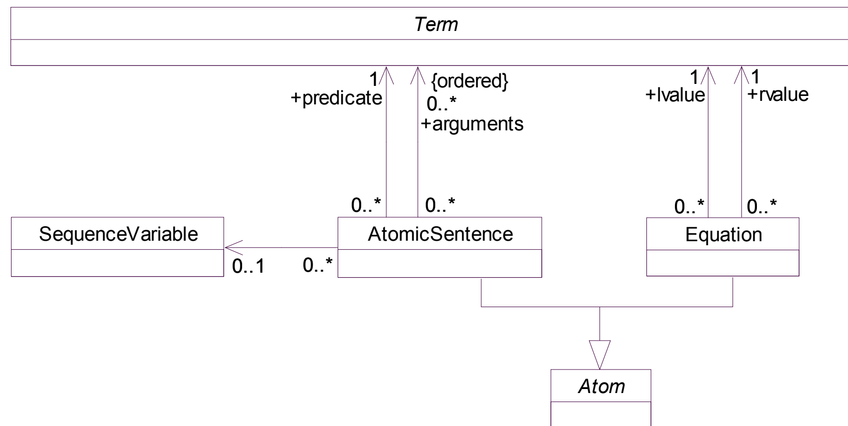


Figure 22 Atomic Sentences

13.4.1 Atom

Description

An atom is either an *equation* containing two *arguments* which are terms, or consists of a term, called the *predicate*, and term sequence called the *argument sequence*, containing terms called *arguments* of the atom. *Dialects which use a name to identify equality may consider it to be a predicate.*

Attributes

None.

Associations

- Specialize Class Sentence

Constraints

[1] The AtomicSentence/Equation partition is disjoint.

```
context Atom inv DisjointPartition:
          (self.oclIsKindOf(AtomicSentence) xor self.oclIsKindOf(Equation))
```

Semantics

An atom, or atomic sentence, asserts that a relation holds between arguments. Its general syntactic form is that of a relation term applied to an argument sequence.

13.4.2 AtomicSentence

Description

An atomic sentence consists of a relation term (predicate) applied to an argument sequence.

Attributes

None.

Associations

- arguments: Term [0..*] in association ArgumentsForAtomicSentence – links an argument sequence to the relation that they participate in
- predicate: Term [1] in association PredicateForAtomicSentence – links a predicate to the relation (atomic sentence) it participates in
- sequenceVariable: SequenceVariable [0..1] in association SequenceVariableForRelation – augments the argument list for the relation with an optional sequence variable
- Specialize Class Atom

Constraints

None.

Semantics

See additional discussion of the semantics of relations in CL in [ISO 24707].

13.4.3 Equation

Description

An equation asserts that its arguments are equal and consists of exactly two terms.

Attributes

None.

Associations

- lvalue: Term [1] in association LvalueForIdentity – associates a term as the “lvalue” of the equation (identity relation)
- rvalue: Term [1] in association RvalueForIdentity – associates a term as the “rvalue” of the equation (identity relation)
- Specialize Class Atom

Constraints

None.

Semantics

Equations are distinguished as a special category because of their special semantic role and special handling by many applications. See additional discussion of the semantics of equations in CL in [ISO 24707].

13.5 The Sentences Diagram

As shown in Figure 23, a sentence is either an atom, a boolean or quantified sentence, an irregular sentence, or a sentence with an attached comment, or an irregular sentence. *The current specification does not recognize any irregular sentence forms. They are included in the abstract syntax to accommodate future extensions to Common Logic, such as modalities.*

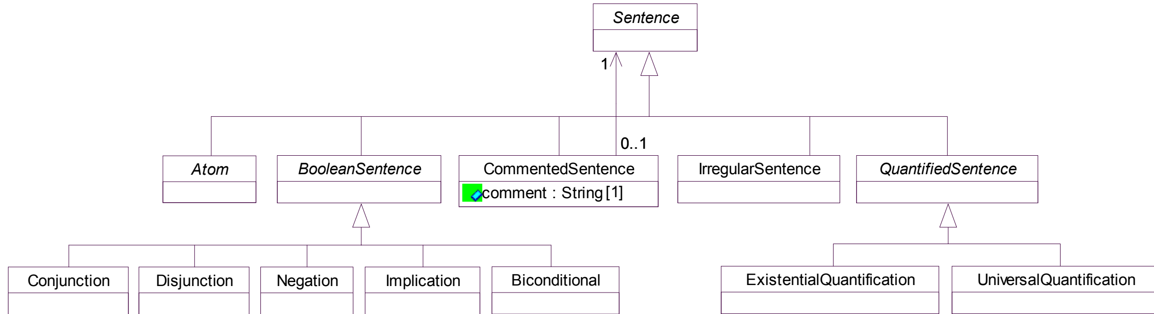


Figure 23 Sentences

13.5.1 Biconditional

Description

A Biconditional (or equivalence relation), consisting of `(iff s1 s2)`, asserts that it is

```
true if I(s1) = I(s2), otherwise false.
```

Essentially, this means that each of the two sentences implies the other.

Attributes

None.

Associations

- lvalue: Sentence [1] in association LvalueForBiconditional – associates exactly one sentence as the ‘lvalue’ of the expression
- rvalue: Sentence [1] in association RvalueForBiconditional – associates exactly one sentence as the ‘rvalue’ of the expression
- Specialize Class BooleanSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.2 CommentedSentence

Description

Provides the capability of commenting sentences as well as commented sentences.

Attributes

- comment: String [1] – represents the comment about the sentence

Associations

- sentence: Sentence [1] in association CommentForSentence – associates exactly one sentence as the argument of the expression
- Specialize Class Sentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in the specification.

13.5.3 Conjunction

Description

A conjunction, consisting of a set of conjuncts, `(and` s_1 `...` s_n`)`, asserts that it is

`false if I(`s_i`) = false  for some i in 1 … n, otherwise true.`

Essentially, a conjunction means that all its components are true. Note that *true* is defined as the empty case of a conjunction – there are no explicit definitions of true and false in CL. These definitions are conventional in formal logic and knowledge representation work.

Attributes

None.

Associations

- conjuncts: Sentence [0..*] in association Conjunction – associates zero or more sentences as conjuncts of the expression
- Specialize Class BooleanSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.4 Disjunction

Description

A disjunction, consisting of a set of disjuncts, `(or s1 ... sn)`, asserts that it is

`true if I(si) = true for some i in 1 … n, otherwise false.`

Essentially, a disjunction means that at least one of its components is true. Note that *false* is defined as the empty case of a disjunction.

Attributes

None.

Associations

- disjuncts: Sentence [0..*] in association Disjunction – associates zero or more sentences as disjuncts of the expression
- Specialize Class BooleanSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.5 ExistentialQuantification

Description

An existentially quantified sentence, consisting of `(exists (var) body)`, asserts that if for some name map `B` on `{var}, IB(body) = true, then true; otherwise, false`. An existentially quantified sentence means that its body is *true for some interpretation of its variables*. It consists of a sequence of bound names and a body that is a sentence.

Attributes

None.

Associations

- Specialize Class QuantifiedSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.6 Implication

Description

An implication, consisting of `(implies s1 s2)`, asserts that it is

```
false if I(s1) = true and I(s2) = false, otherwise true.
```

Essentially, this means that the *antecedent* implies the *consequent*.

Attributes

None.

Associations

- antecedent: Sentence [1] in association AntecedentForImplication – associates exactly one sentence as the antecedent of the expression
- consequent: Sentence [1] in association ConsequentForImplication – associates exactly one sentence as the consequent of the expression
- Specialize Class BooleanSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.7 IrregularSentence

Description

Provides the placeholder for irregular sentences in the metamodel, potentially for use with modal sentence requirements for the Semantics for Business Vocabularies and Rules (SBVR) specification.

Attributes

None.

Associations

- Specialize Class Sentence

Constraints

None.

Semantics

None.

13.5.8 Negation

Description

A negation, consisting of `(not s)`, asserts that it is `true if I(s) = false, otherwise false`. Essentially, a negation means that its inner sentence is false.

Attributes

None.

Associations

- sentence: Sentence [1] in association NegationSentence – associates exactly one sentence as the argument of the expression
- Specialize Class BooleanSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.9 QuantifiedSentence

Description

QuantifiedSentence is an abstract class representing quantified sentences – it was introduced primarily as a notational convenience for the purposes of simplifying the metamodel. Quantifiers may bind any number of variables; bound variables may be restricted to a named category.

Attributes

None.

Associations

- body: Sentence [1] in association QuantificationForSentence – associates exactly one sentence (body) with the expression
- boundName: LogicalName [0..*] in association NameBoundByQuantifier – associates zero or more bound variables with the expression
- Specialize Class Sentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.5.10 UniversalQuantification

Description

A universally quantified sentence, consisting of `(forall (var) body)`, asserts that if for every name map `B` on `{var}, IB(body) = true, then true; otherwise, false`. A universally quantified sentence means that its body is true for any interpretation of its variables. It consists of a sequence of bound names and a body that is a sentence.

Attributes

None.

Associations

- Specialize Class QuantifiedSentence

Constraints

None.

Semantics

See additional discussion of the semantics of sentences in CL in [ISO 24707].

13.6 The Boolean Sentences Diagram

A Boolean sentence has a type, called a *connective*, and a number of sentences called the *components* of the Boolean sentence, as shown in Figure 24. The number depends on the particular type. Every common logic dialect **must** distinguish the *conjunction*, *disjunction*, *negation*, *implication* and *biconditional* types with respectively any number, any number, one, two and two components.

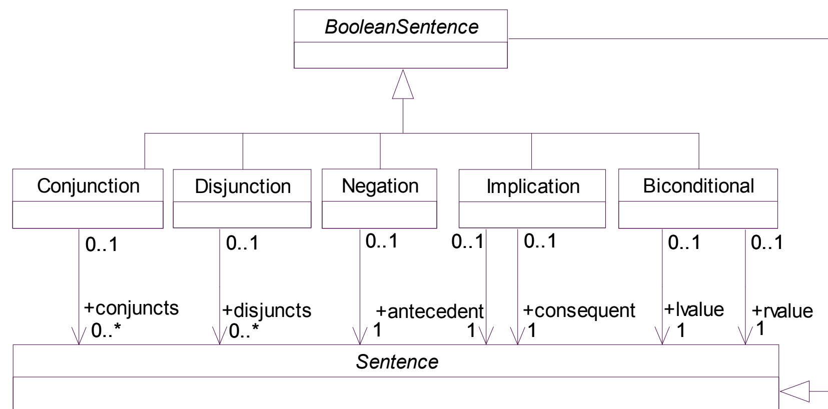


Figure 24 Boolean Sentences

13.7 The Quantified Sentences Diagram

A quantified sentence has a type, called a *quantifier*, and a set of names called the *bound names*, and a sentence called the *body* of the quantified sentence, as shown in Figure 25. Every common logic dialect **must** distinguish the *universal* and the *existential* types of quantified sentence. Any occurrence of a bound name in the body is said to be *bound in* the body; any name which occurs in the body and is not bound in the body is *free in* the body.

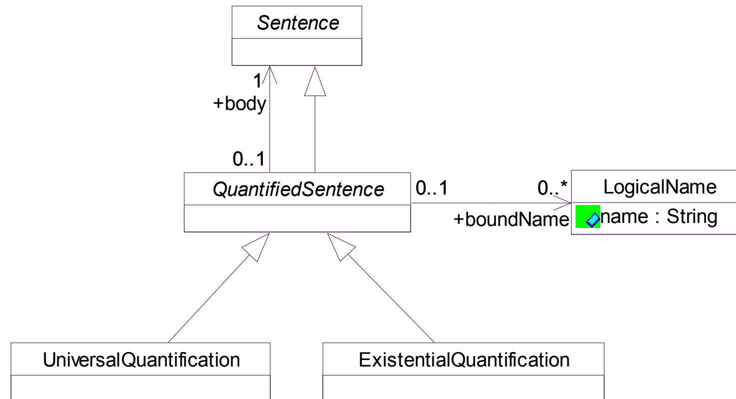


Figure 25 Quantified Sentences

13.8 Summary of CL Metamodel Elements with Interpretation

Table 32 below presents a summary of the elements in the metamodel (not exhaustive) with the corresponding elements of the core abstract syntax and their interpretation, derived from Table 1 and the summary given in section 6.5 of the [ISO 24707] Common Logic Specification.

Table 32 CL Metamodel Summary with Interpretation

CL Metamodel Element(s)	CL Core Syntax	Interpretation
	If E is an expression of the form	then I(E) =
LogicalName	name N	int_I(N)
SequenceVariable	sequence variable s	seq_I(s)
Term, FunctionalTerm, AtomicSentence	term sequence t1...tn: [t1]...[tn]	<I(t_1)...I(t_n)>
Term, FunctionalTerm, AtomicSentence	term sequence t_1...t_n with sequence variable s: [t_1]...[t_n][s]	<I(t_1)...I(t_n)>;I(s)
Equation	term which is an equation containing terms t_1, t_2: (= [t_1][t_2])	true if I[t_1] = I[t_2], otherwise false
FunctionalTerm	term with operator o and term sequence s: ([o][s])	fun_I(I(o))(I(s)), i.e., the x such that <(I(s),x)> is in (I(o))

Table 32 CL Metamodel Summary with Interpretation

CL Metamodel Element(s)	CL Core Syntax	Interpretation
Atom, AtomicSentence	atom with predicate p and term sequence s: ([p][s])	true if I(s) is in rel_I(I(p)), otherwise false
BooleanSentence, Negation	boolean sentence of type negation and component c: (not [c])	true if I(c) = false, otherwise false
BooleanSentence, Conjunction	boolean sentence of type conjunction and components c_1...c_n: (and [c_1]...[c_n])	true if I(c_1) = ... = I(c_n) = true, otherwise false
BooleanSentence, Disjunction	boolean sentence of type disjunction and components c_1...c_n: (or [c_1]...[c_n])	false if I(c_1) = ... = I(c_n) = false, otherwise true
BooleanSentence, Implication	boolean sentence of type implication and components c_1, c_2: (implies [c_1][c_2])	false if I(c_1) = true and I(c_2) = false, otherwise true
BooleanSentence, Biconditional	boolean sentence of type biconditional and components c_1, c_2: (iff [c_1][c_2])	true if I(c_1) = I(c_2), otherwise false
QuantifiedSentence, UniversalQuantification	quantified sentence of type universal and set of names N and body B: (forall ([N])[B])	true if for every name map A on N, I[A](B) is true; otherwise false
QuantifiedSentence, ExistentialQuantification	quantified sentence of type existential and set of names N and body B: (exists ([N])[B])	false if for every name map A on N, I[A](B) is false; otherwise true
Sentence, IrregularSentence	irregular sentence [S]	int_I(S)
Phrase, Sentence	phrase which is a sentence: [S]	true if for every sequence map B on the set of sequence variables in S, I[B](S) is true; otherwise false
Phrase, Importation	phrase which is an importation containing name N: (cl:imports [N])	true if I(*text*(I(N))) = true, otherwise false.
Module, ExclusionSet, Text	module with name N, exclusion list L, and body text B: (cl:module [N] (cl:excludes [L])[B])	true if [I<L](B) = true and ext(I(N)) = $D_{[I<L]}$*, otherwise false.
Text	text containing phrases s_1...s_n: [S_1]...[s_n]	true if I(S_1) = … = I(S_n) = true, otherwise false.

14 The ER Metamodel

14.1 Overview

The ER (Entity Relationship) Metamodel is a MOF2 compliant metamodel that allows users to define conceptual or ontology models using the terminology and concepts of entities and relationships. Conceptual (or Ontology) modeling deals with the question on how to describe in a declarative and reusable way the domain information of enterprises/applications, their relevant vocabulary, and how to constrain the use of the data, by understanding what can be drawn from it.

The ER diagram has been widely used as a means for describing conceptual or logical models. The ER model was a precursor of today's object models (e.g., UML) or ontology models (e.g., RDFS, OWL) and is probably the first data model to have the adjective "semantic" applied to it.

14.1.1 Organization of the ER Metamodel

The ER Metamodel uses diagrams to control complexity and promote understanding.

The classes and associations are grouped and illustrated in the following diagrams:

- Model
  Contains classes and associations that can be used to define the scope and elements of an ER model.
- Domain
  Contains classes and associations that can be used to define ER domains.
- Entity
  Contains classes and associations that can be used to define ER entities, including generalization.
- Relationship
  Contains classes and associations that can be used to define ER relationships and roles.
- Key
  Contains classes and associations that can be used to define ER keys.
- Instance
  Contains classes and associations that can be used to define ER entity instances and relationship instances.
- Inheritance
  Shows the inheritance hierarchy among all classes in the ER metamodel.

14.2 The Model Diagram

The Model diagram of the ER metamodel is shown in Figure 26.

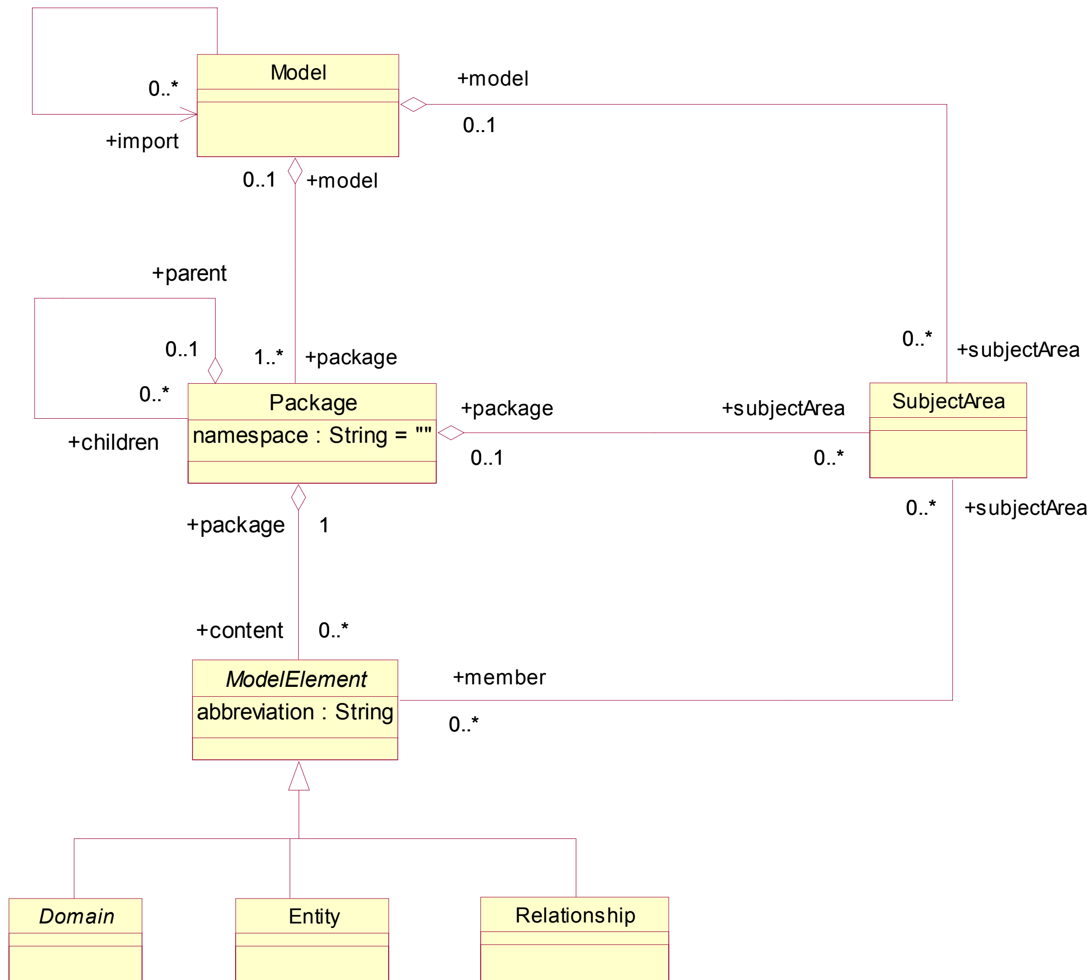


Figure 26 The Model Diagram of the ER Metamodel

14.2.1 Model

A model consists of the various modeling elements (entities, relationships and domains) that can be used to describe and represent things of interest to an enterprise.

Description

A model consists of the various modeling elements (entities, relationships and domains) that can be used to describe and represent things of interest to an enterprise. Entities represent things within a subject area or across areas, and relationships represent the associations between them. Domains represent logical data types.

Attributes

No additional attributes.

Associations

- import: Model [0..*]
  This specifies the models that are imported by this model. All model elements in the imported models may be used in this model for definition of new model elements.
- package: Package [1..*]
  This specifies the packages that are owned by this model. A package may contain subpackages.
- subjectArea: SubjectArea [0..*]
  This specifies the subject areas that are owned by this model.

Constraints

No additional constraints.

Semantics

A model provides a container for modeling elements. If a model is removed, so are the modeling elements owned by it. A model owns modeling elements through packages.

14.2.2 ModelElement

A model element is a common superclass of entity, relationship and domain.

Description

A model is a common superclass of entity, relationship and domain. ModelElement is an abstract metaclass.

Attributes

- abbreviation: String
  This represents the default abbreviation for the name of this model element.

Associations

- package: Package [1]
  This represents the package that owns this model element.
- subjectArea: SubjectArea [0..*]
  This specifies the subject areas that this model element is a member of.

Constraints

No additional constraints.

Semantics

See Description.

14.2.3 Package

A package provides a mechanism for organizing model elements in a model.

Description

A package provides a mechanism for organizing model elements in a model. A package may contain subpackages, organized as a hierarchy.

Attributes

- namespace: String =""
  An URI representing the namespace that this package belongs to. All model elements owned by this package belong to the namespace.

Associations

- children: Package [0..*]
  This specifies the packages that are the children of this package.
- model: Model [0..1]
  This specifies the model that owns this package. Only root packages are owned by a model
- content: ModelElement [0..*]
  This specifies the model elements that are owned by this package.
- parent: Package [0..1]
  This specifies the package that is the parent of this package. Root packages (i.e., packages owned by the model) do not have parent.
- subjectArea: SubjectArea [0..*]
  This specifies the subject areas that are owned by this package.

Constraints

No additional constraints.

Semantics

A package provides provides a mechanism for organizing model elements in a model. If a package is removed, so are the model elements owned by it. A package is either owned by a model (in which case it is a root package) or it has parent.

14.2.4 SubjectArea

A subject area provides a mechanism for logically grouping or classifying model elements in a model.

Description

A subject area provides a mechanism for logically grouping or classifying model elements in a model.

Attributes

No additional attributes.

Associations

- model: Model [0..1]
  This specifies the model that owns this subject area.
- member: ModelElement [0..*]
  This specifies the model elements that are members of this subject area.
- package: Package [0..1]
  This specifies the package that owns this subject area.

Constraints

No additional constraints.

Semantics

A subject area provides a mechanism for logically grouping or classifying model elements in a model. It is owned by either a model or a package..

14.3 The Domain Diagram

The Domain diagram of the ER metamodel is shown in Figure 27.

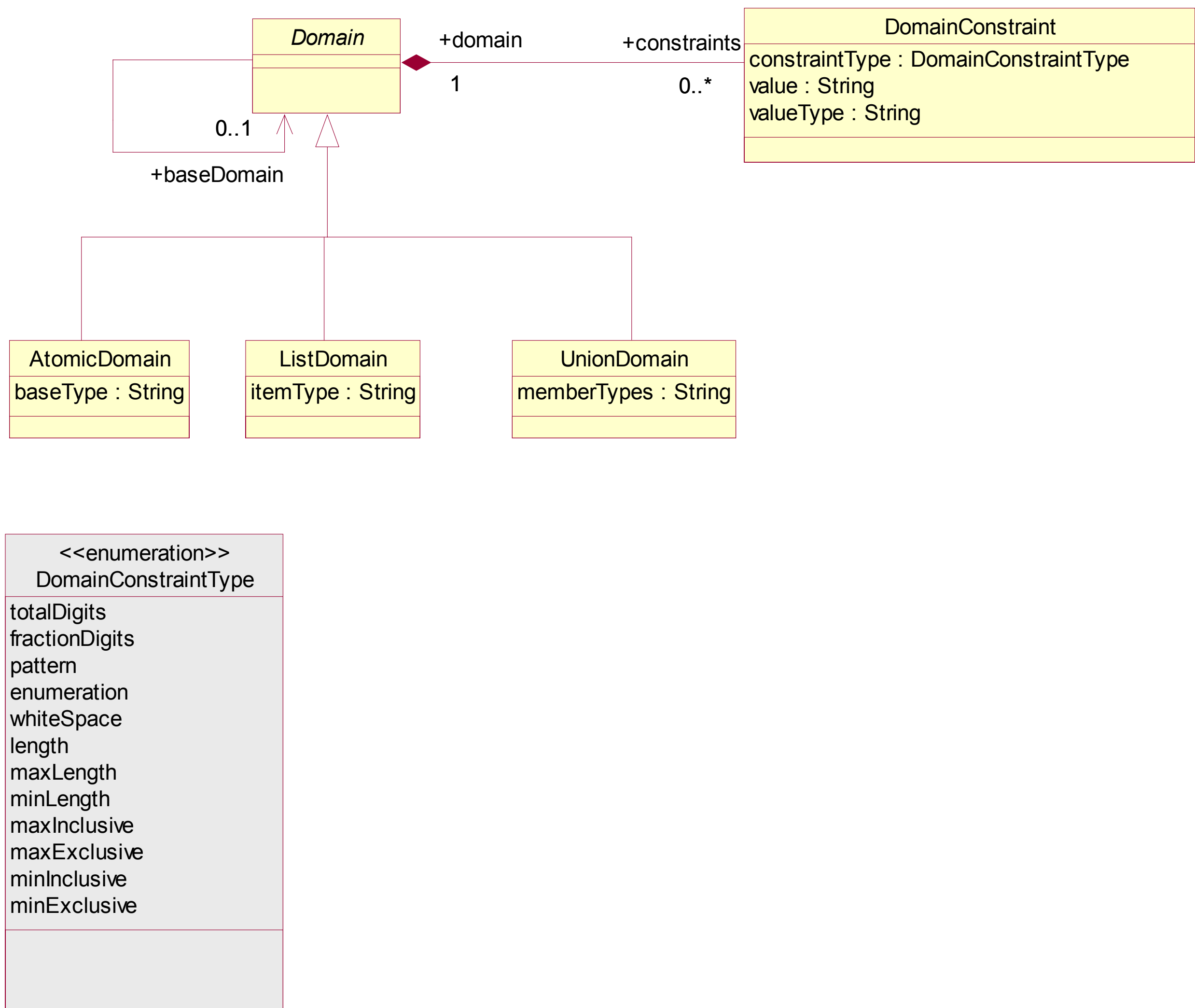


Figure 27 The Domain Diagram of the ER Metamodel

14.3.1 AtomicDomain

Atomic domains are those having values which are regarded as being indivisible.

Description

Atomic domains are those having values which are regarded as being indivisible. Atomic domains restrict, in a manner described by their constraints, the value space of the datatype identified via the baseType attribute.

Attributes

- baseType: String
  This identifies the base datatype of this atomic domain. A base datatype is an XML Schema primitive datatype [XML Schema Datatypes].

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

14.3.2 Domain

Domain represents user-defined datatypes and can be used as the type of Attributes.

Description

Domain represents user-defined datatypes and can be used as the type of Attributes. Domain is an abstract metaclass.

Attributes

No additional attributes.

Associations

- baseDomain: Domain [0..1]
  This identifies the base domain that this domain is derived from.

- constraints: DomainConstraints [0..*]
  This specifies the constraints that applies to this domain.

Constraints

No additional constraints.

Semantics

See Description.

14.3.3 DomainConstraint

Domain constraints are used to restrict the value space of datatypes.

Description

Domain constraints are used to restrict the value space of datatypes. Constraining the value space consequently constrains the lexical space. A value space is the set of values for a given datatype, whereas a lexical space is the set of valid literals for a datatype.

Attributes

- constraintType: DomainConstraintType
  This identifies the type of this domain constraint. There are twelve types of domain constraints: totalDigits, fractionDigits, pattern, enumeration, whiteSpace, length, maxLength, minLength, maxInclusive, maxExclusive, minInclusive, minExclusive. The allowed types depend on the kind of domain (AtomicDomain, ListDomain, or UnionDomain) and the baseType (in the case of AtomicDomain). See [XML Schema Datatypes] for description.

- value: String [1..*]
  This specifies the value(s) of this domain constraint .

- valueType: String
  This specifies the datatype of the value(s) of this domain constraint. The allowed datatype must be an XML Schema primitive datatype and it depends on the type of domain constraint as follows [XML Schema Datatypes]:
  totalDigits - positiveInteger
  fractionDigits - nonNegativeInteger
  pattern - string (regular expression)
  enumeration - baseType of AtomicDomain
  whiteSpace - string (‘preserve’, ‘replace’, ‘collapse’)
  length - nonNegativeInteger
  maxLength - nonNegativeInteger
  minLength - nonNegativeInteger
  maxInclusive - baseType of AtomicDomain
  maxExclusive - baseType of AtomicDomain
  minInclusive - baseType of AtomicDomain
  minExclusive - baseType of AtomicDomain

Associations

- domain: Domain
  This identifies the domain that owns this domain constraint.

Constraints

No additional constraints.

Semantics

See Description.

14.3.4 ListDomain

List domains are those having values each of which consists of a finite-length (possibly empty) sequence of atomic values.

Description

List domains are those having values each of which consists of a finite-length (possibly empty) sequence of atomic values. The value space of a list domain is a set of finite-length sequences of atomic values. The lexical space of a list datatype is a set of literals whose internal structure is a space-separated sequence of literals of the type of the items in the list.

Attributes

- itemType: String
  This identifies the datatype of the item(s) that may participate in the list. It may be an XML Schema primitive datatype or an atomic domain.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description. The following types of domain constraint apply to list domain: pattern, enumeration, whiteSpace, length, maxLength, minLength.

14.3.5 UnionDomain

Union domains are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of one or more other domains.

Description

Union domains are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of one or more other domains.

Attributes

- memberType: String [1..*]
  This identifies the datatype of the member(s) that may participate in the union. It may be an XML Schema primitive datatype or an atomic domain..

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

14.4 The Entity Diagrams

The Entity diagram of the ER metamodel is shown in Figure 28.

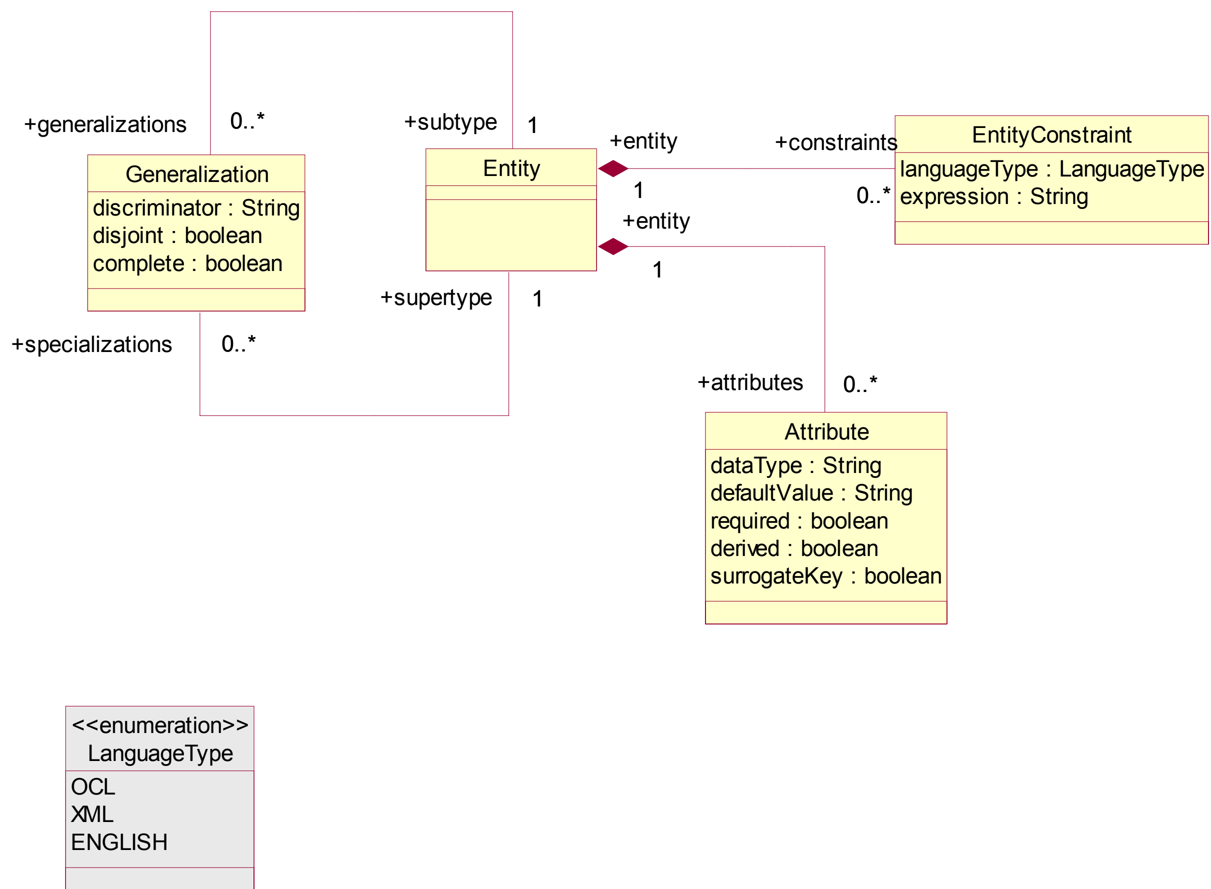


Figure 28 The Entity Diagram of the ER Metamodel

14.4.1 Attribute

An attribute represents a common characteristic of some entity instances.

Description

An attribute represents a common characteristic of some entity instances. It captures a single piece of information about the entity instance.

Attributes

- datatype: String
  This specifies the datatype of this attribute.

- defaultValue: String
  This specifies the default value of this attribute.

- derived: Boolean
  This specifies whether this attribute is derived. The default is “false”, i.e., an attribute is not derived by default.

Derived attributes are those that are computed from other attributes.

- required: Boolean
  This specifies whether this attribute is required. A required attribute must have a value for each entity instance that has it as a characteristic.

- surrogateKey: Boolean
  This specifies whether this attribute is a surrogate key. A surrogate key is an arbitrary number that is assigned to an entity instance to uniquely identify it within an entity. A surrogate key is often the best choice for a primary key.

Association

- entity: Entity
  This specifies the entity that owns this attribute.

Constraints

No additional constraints.

Semantics

See description.

14.4.2 Entity

Entities represents persons, places, or things that have common characteristics.

Description

Entities represents persons, places, or things that have common characteristics. Entities provide an abstraction mechanism for grouping things with common characteristics

Attributes

No additional attributes.

Associations

- attribute: Attribute [0..*]
  This specifies the attributes that are owned by this entity.

- constraints: EntityConstraint [0..*]
  This specifies the constraints that apply to this entity.

- generalizations: Generalization [0..*]
  This specifies the generalizations for this entity. These generalizations navigate to more general entities in the generalization hierarchy.

- keys: Key [0..*]
  This specifies the keys that are owned by this entity.

- relationships: Relationship [0..*]
  This specifies the relationships that are (logically) owned by this entity.

- role: Role [0..*]
  This specifies the roles that this entity plays.

- specializations: Generalization [0..*]

This specifies the specializations for this entity. These specializations navigate to more specific entities in the generalization hierarchy.

Constraints

No additional constraints.

Semantics

The purpose of an entity is to provide an abstraction mechanism for grouping things with common characteristics. A thing can be an instance of one and only one entity.

14.4.3 EntityConstraint

Entity constraints are used to restrict the instances of entities.

Description

Entity constraints are used to restrict the instances of entities. All instances of an entity must satisfy its entity constraints.

Attributes

- expression: String
  This specifies the content of this entity constraint.
- languageType: LanguageType
  This specifies the language used to express the content of this entity constraint. The allowed languages are: English, OCL and XML.

Association

- entity: Entity
  This specifies the entity that owns this entity constraint.

Constraints

No additional constraints.

Semantics

See description.

14.4.4 Generalization

A generalization is a taxonomic relationship between a more general (supertype) entity and a more specific (subtype) entity.

Description

A generalization is a taxonomic relationship between a more general (supertype) entity and a more specific (subtype) entity. Each instance of the subtype entity is also an instance of the supertype entity. Thus, the subtype entity indirectly has attributes of the supertype entity.

Attributes

- complete: Boolean
  This specifies whether this generalization is complete. A complete generalization indicates that all possible sub-

type entities are included in the generalization structure. An incomplete generalization indicates that there may be other subtype entities that have not yet been discovered.

- discriminator: String
  This specifies the subtype discriminator for this generalization. The subtype discriminator is an attribute of the supertype entity. It may be used to distinguish one subtype entity from another.
- disjoint: Boolean
  This specifies whether this generalization is disjoint. In a disjoint generalization, each instance in the supertype can relate to one and only one subtype. In a non-disjoint generalization, each instance in the supertype can relate to one or more subtypes.

Association

- subtype: Entity
  This references the more specific (subtype) entity in this generalization relationship.
- supertype: Entity
  This references the more general (supertype) entity in this generalization relationship.

Constraints

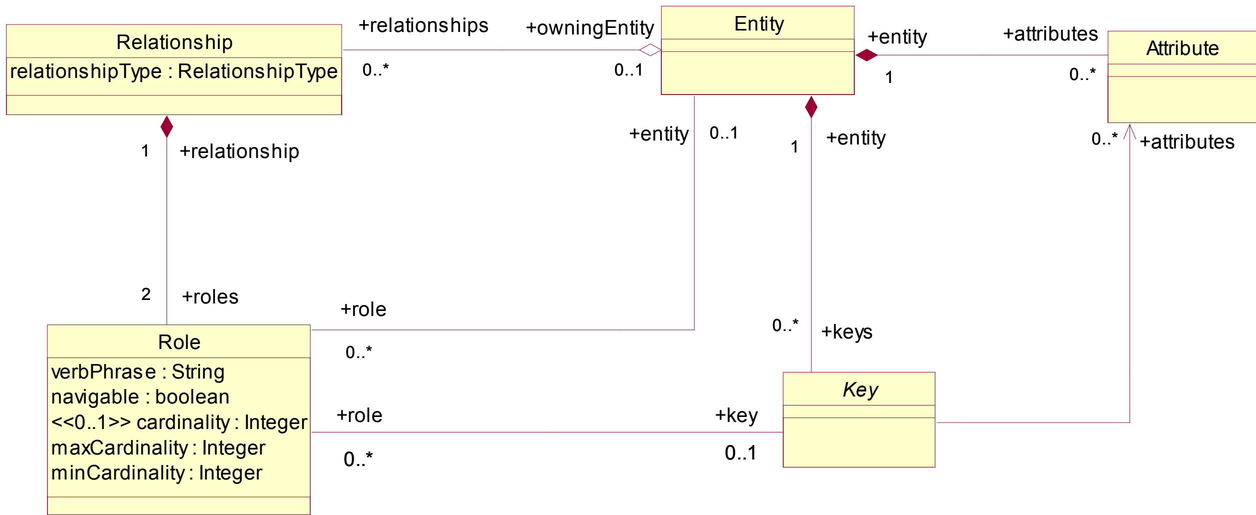
No additional constraints.

Semantics

See description.

14.5 The Relationship Diagram

The Relationship diagram of the ER metamodel is shown in Figure 29.



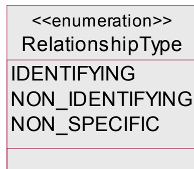


Figure 29 The Relationship Diagram of the ER Metamodel

14.5.1 Relationship

Relationships represent connections, links, or associations between two or more entities.

Description

Relationships represent connections, links, or associations between two or more entities. Relationships are binary in nature, and may be bi-directional or uni-directional.

Attributes

- relationshipType: RelationshipType
  This specifies the type of this relationship. Allowed relationship types are: IDENTIFYING, NON_IDENTIFYING, and NON_SPECIFIC. An identifying relationship is one whereby an instance of the child entity is identified through its association with a parent entity.

Associations

- role: Role [2]

This specifies the roles owned by this relationship.

- owningEntity: Entity [0..1]
  This specifies the entity that owns this relationship.

Constraints

No additional constraints.

Semantics

See Description.

14.5.2 Role

This represents the role that an entity plays in a relationship.

Description

This represents the role that an entity plays in a relationship.

Attributes

- cardinality: Integer [0..1]
  This is used to describe a role that has exactly N distinct entities participating in a relationship, where N is the value of the cardinality constraint.
- maxCardinality: Integer
  This is used to describe a role that has at most N distinct entities participating in a relationship, where N is the value of the maximum cardinality constraint.
- minCardinality: Integer
  This is used to describe a role that has at least N distinct entities participating in a relationship, where N is the value of the minimum cardinality constraint.
- navigable: Boolean
  This specifies whether this role is navigable. For a uni-directional relationship only one of the roles is navigable.
- verbPhrase: String
  This specifies a verb phrase that can be used instead of the name of this role in text so that the text will be more human readable.

Associations

- entity: Entity [0..1]
  This specifies the entity that plays this role.
- key: Key [0..1]
  This specifies the key that is associated with this role. The key belongs to the entity that plays this role.
- relationship: Relationship
  This specifies the owning relationship of this role.

Constraints

No additional constraints.

Semantics

See Description.

14.6 The Key Diagram

The Key diagram of the ER metamodel is shown in Figure 30.

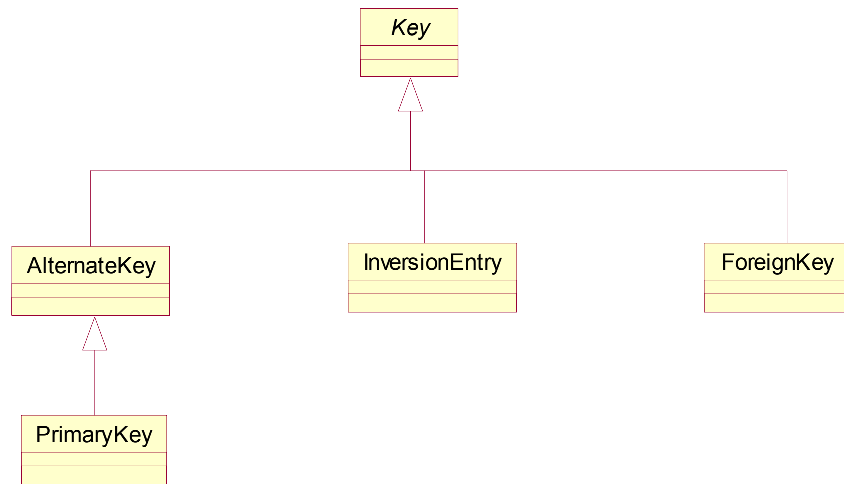


Figure 30 The Key Diagram of the ER Metamodel

14.6.1 AlternateKey

Alternate keys are keys that meet the requirements for being a primary key.

Description

Alternate keys are keys that meet the requirements for being a primary key. An alternate key is an attribute or attributes that uniquely identify an instance of an entity.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

14.6.2 ForeignKey

A foreign key identifies a set of attributes in one entity instance that uniquely identifies an instance of another entity containing a matching primary key.

Description

A foreign key identifies a set of attributes in one entity instance that uniquely identifies an instance of another entity containing a matching primary key.

Attributes

No additional attributes.

Associations

No additional attributes.

Constraints

No additional constraints.

Semantics

See Description.

14.6.3 InversionEntry

An inversion entry is an attribute or attributes that do not uniquely identify an instance of an entity, but nonetheless are often used to identify instances of the entity.

Description

An inversion entry is an attribute or attributes that do not uniquely identify an instance of an entity, but nonetheless are often used to identify instances of the entity. Examples are people’s name.

Attributes

No additional attributes.

Associations

No additional attributes.

Constraints

No additional constraints.

Semantics

See Description.

14.6.4 Key

A key is an attribute or attributes that identifies an instance of an entity.

Description

A key is an attribute or attributes that identifies an instance of an entity. A key may or may not be unique. Key is an abstract class.

Attributes

No additional attributes.

Associations

- attribute: Attribute [0..*]
  This specifies the attributes that are members of this key.
- entity: Entity
  This specifies the entity that owns this key.
- role: Role [0..*]
  This specifies the role that is associated with this key.

Constraints

No additional constraints.

Semantics

See Description.

14.6.5 PrimaryKey

A primary key is an attribute or attributes that uniquely identifies an instance of an entity.

Description

A primary key is an attribute or attributes that uniquely identifies an instance of an entity. An entity has only one primary key.

Attributes

No additional attributes.

Associations

No additional associations.

Constraints

No additional constraints.

Semantics

See Description.

14.7 The Instance Diagram

The Instance diagram of the ER metamodel is shown in Figure 31.

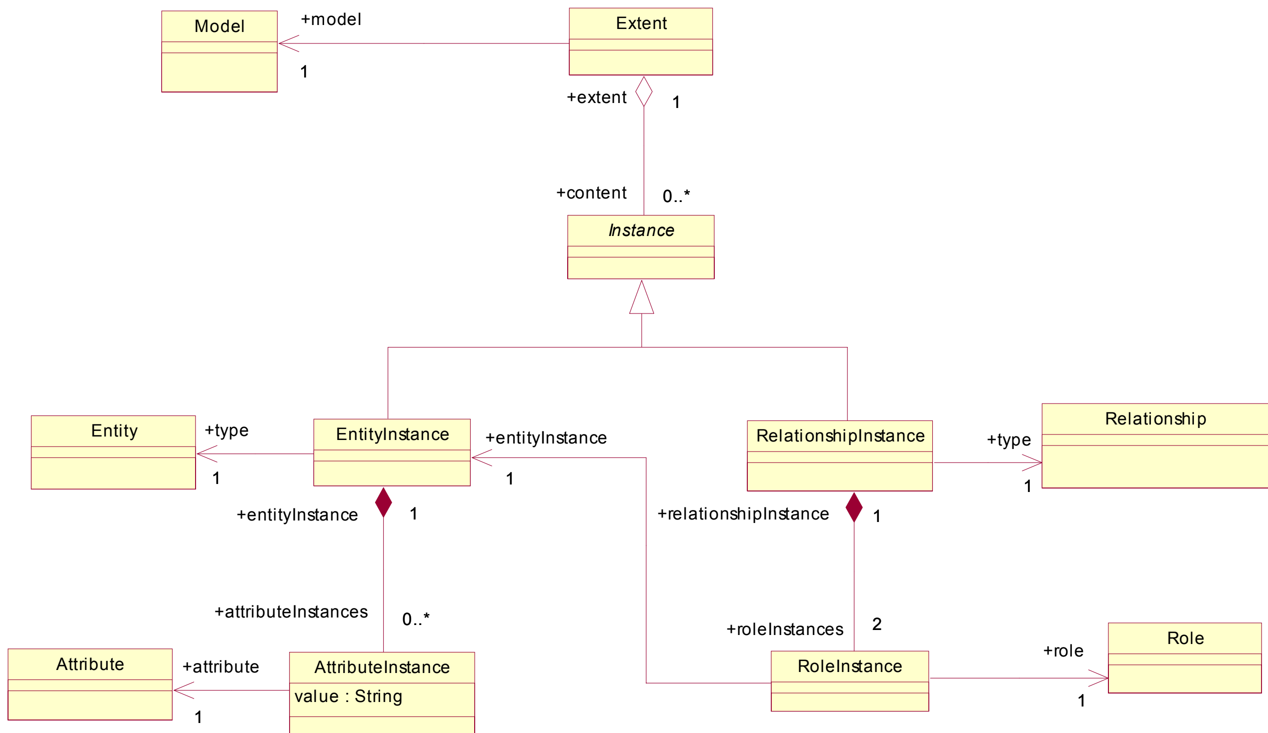


Figure 31 The Instance Diagram of the ER Metamodel

14.7.1 AttributeInstance

This is used to represent the value of an attribute.

Description

This is used to represent the value of an attribute.

Attributes

- value: String
  This contains the literal value of this attribute instance.

Associations

- attribute: Attribute
  This identifies the attribute that this attribute instance is for.

- entityInstance: EntityInstance
  This identifies the entity instance that owns this attribute instance.

Constraints

No additional constraints.

Semantics

See Description.

14.7.2 EntityInstance

This represents an instance of an entity.

Description

This represents an instance of an entity. An entity instance can only be an instance of one entity.

Attributes

No additional attributes.

Associations

- attributeInstances: AttributeInstance [0..*]
  This specifies the attribute instances that are owned by this entity instance.
- type: Entity
  This specifies the entity that is the type of this entity instance.

Constraints

No additional constraints.

Semantics

See Description.

14.7.3 Extent

This represents the collection of entity instances and relationship instances that have been instantiated from an ER model.

Description

This represents the collection of entity instances and relationship instances that have been instantiated from an ER model.

Attributes

No additional attributes.

Associations

- model: Model
  This identifies the Model from which the entity instances and relationship instances are instantiated.
- content: Instance [0..*]
  This identifies the entity instances and relationship instances that are members of this extent.

Constraints

No additional constraints.

Semantics

See Description.

14.7.4 Instance

This represents an entity instance or a relationship instance.

Description

This represents an entity instance or a relationship instance. Instance is a common superclass of EntityInstance and RelationshipInstance. It is an abstract metaclass.

Attributes

No additional attributes.

Associations

- extent: Extent
  The identifies the extent that owns this instance.

Constraints

No additional constraints.

Semantics

See Description.

14.7.5 RelationshipInstance

This represents an instance of a relationship.

Description

This represents an instance of a relationship. A relationship instance can only be an instance of one relationship.

Attributes

No additional attributes.

Associations

- roleInstances: RoleInstance [2]
  This identifies the role instances that are owned by this relationship instance. Role instances are used to reference the entity instances that participate in a relationship.

- type: Relationship
  This specifies the relationship that is the type of this relationship instance.

Constraints

No additional constraints.

Semantics

See Description.

14.7.6 RoleInstance

This is used to represent the value of a role.

Description

This is used to represent the value of a role.

Attributes

No additional attributes.

Associations

- entityInstance: EntityInstance
  This identifies the entity instance that plays the role that this role instance is for.
- relationshipInstance: RelationshipInstance
  This identifies the relationship instance that owns this role instance.
- role: Role
  This identifies the role that this role instance is for.

Constraints

No additional constraints.

Semantics

See Description.

14.8 The Inheritance Diagram

The Inheritance diagram of the ER metamodel is shown in Figure 32.

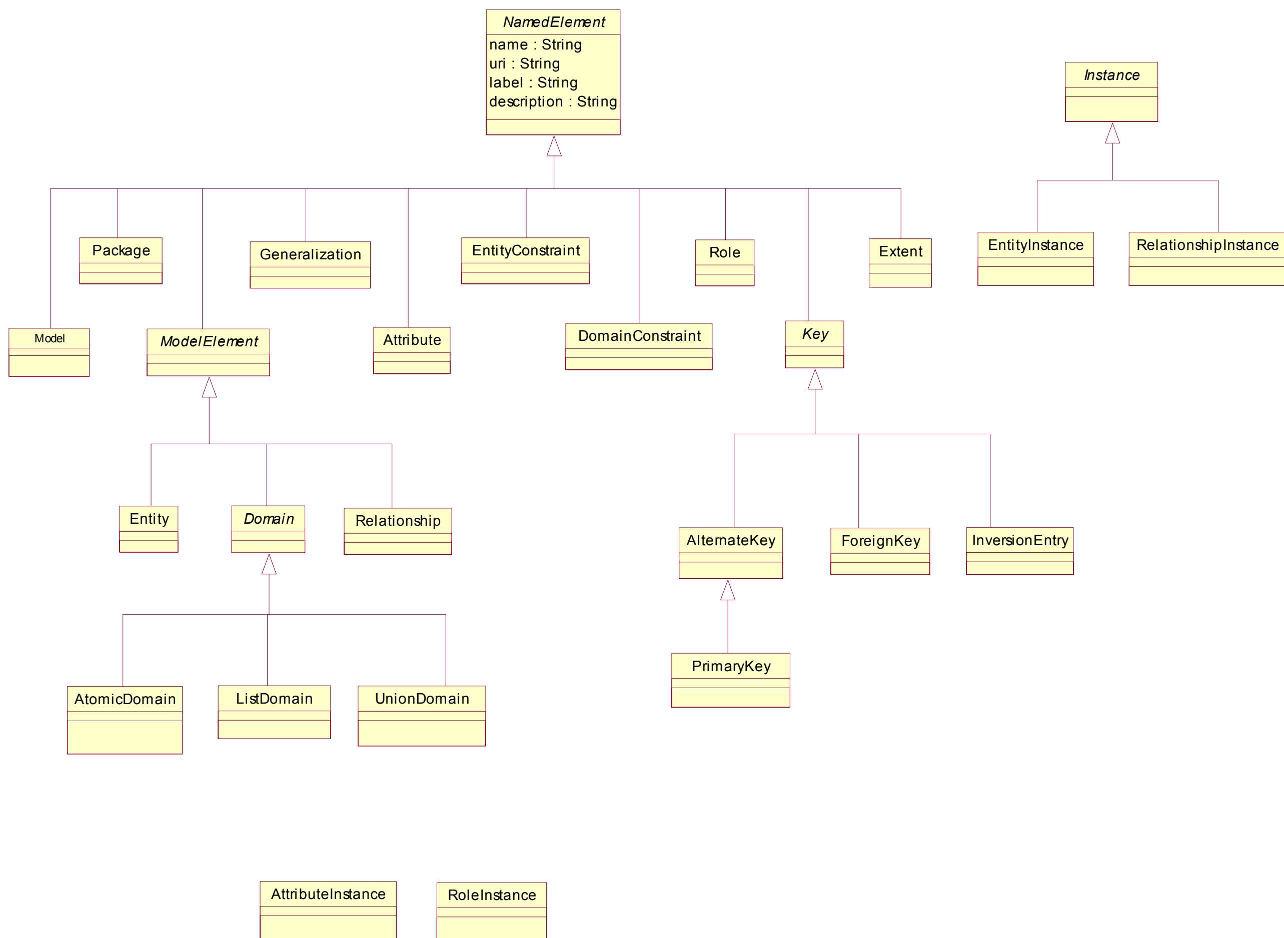


Figure 32 The Inheritance Diagram of the ER Metamodel

14.8.1 NamedElement

This represents a named model element.

Description

This represents a named model element.

Attributes

- name: String [0..1]
  The name of this model element.
- label: String [0..1]
  This may be used to provide a human-readable version of this model element's name.
- description: String [0..1]

This may be used to provide a human-readable description of this model element.

Associations

No associations.

Constraints

No constraints.

Semantics

See Description.

14.9 Examples

To illustrate the usage of the ER metamodel, two example object diagrams are provided below. The Model Example diagram is shown in Figure 33. The Instance Example diagram is shown in Figure 34.

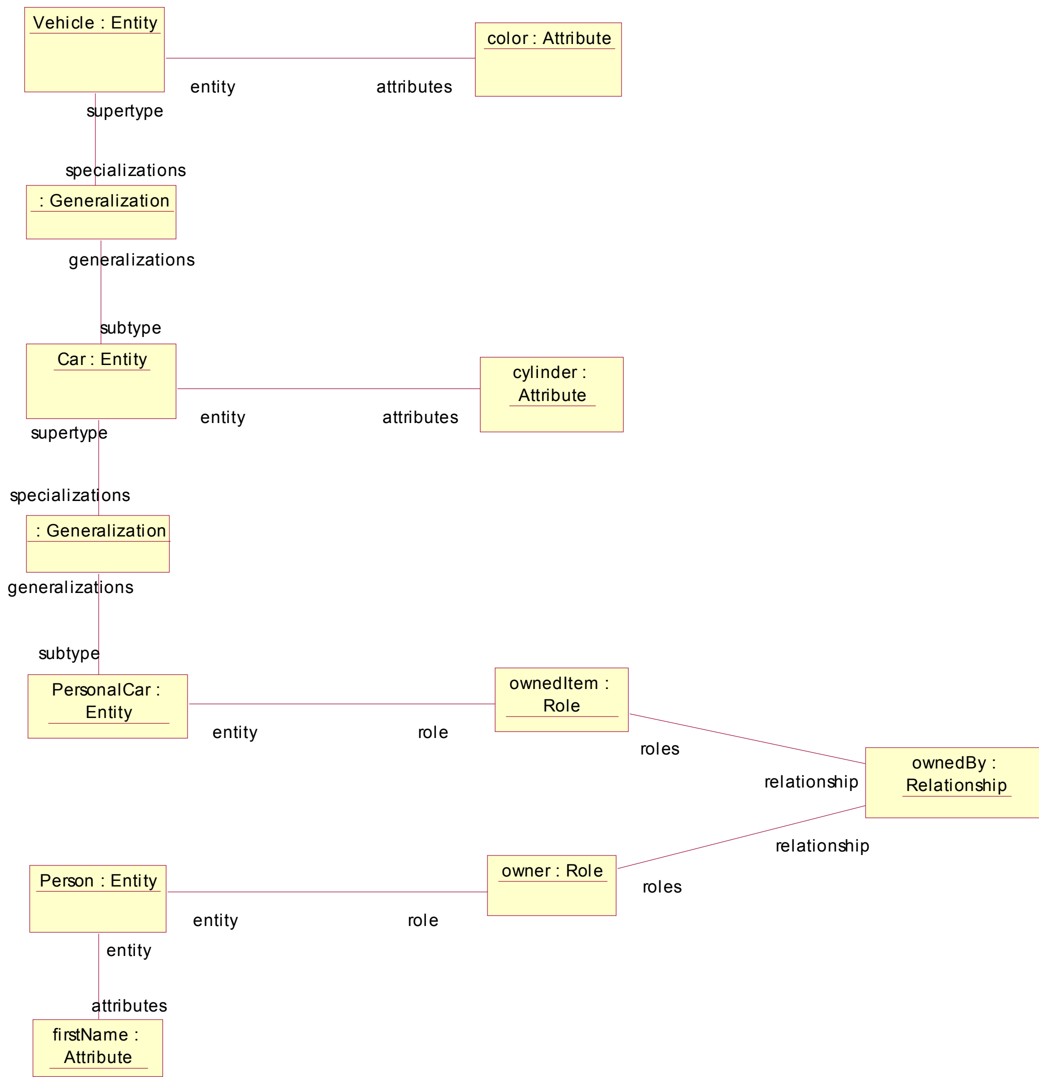


Figure 33 The Model Example Diagram of the ER Metamodel

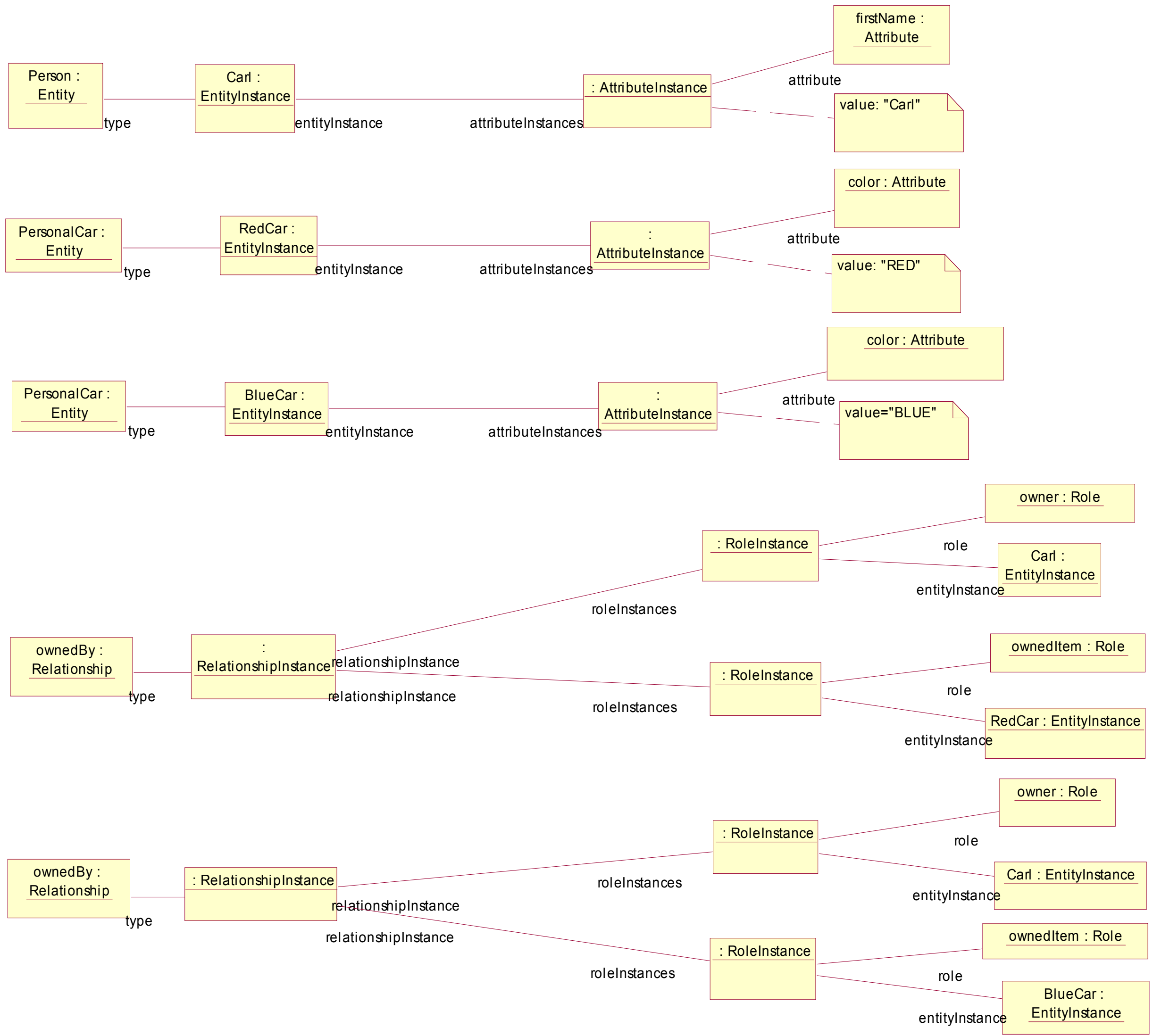


Figure 34 The Instance Example Diagram of the ER Metamodel

15 The Topic Map Metamodel

The Topic Maps Meta-Model is defined based primarily upon ISO 13250-2 Data Model [TMDM] and to a lesser degree ISO 13250-3 XML Syntax. The TMDM provides the most authoritative definition of the abstract syntax for Topic Maps. The following discussion assumes a basic understanding of Topic Maps.

15.1 Topic Map Constructs

Some of the primary elements in the TM meta-model are shown in Figure 35. Topic Maps are composed of a set of Topics and a set of Associations defining multi-way relations among those Topics.

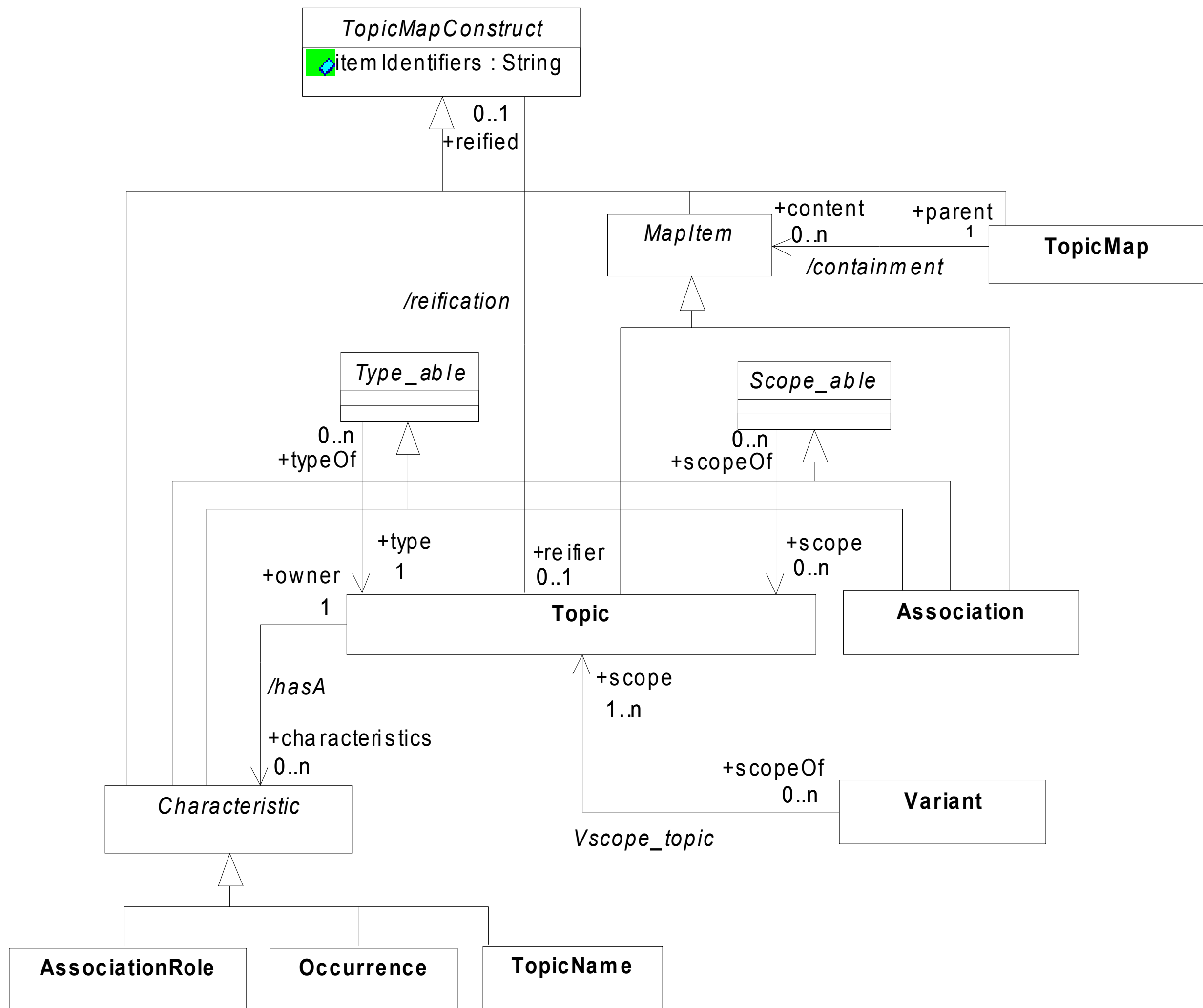


Figure 35 Primary Elements in the Topic Map Metamodel

Each Topic is about a single Subject. Subjects in TM may be anything physical or conceptual. A machine addressable Subject will have a locator (e.g. a URL) while non-machine addressable subjects will have an identifier (e.g. the URL of a page about the subject or a URN). Topics are roughly equivalent to RDF Resources, describing elements in a world of discourse. Note that this similarity does not include RDF Literals that in TMs are not normally considered Topics.

15.1.1 TopicMapConstruct

Description

TopicMapConstructs are the abstract collection of elements that are part of any Topic Map. All first class elements are a sub-type of Topic Map Construct and may optionally have a Source Locator.

Attributes

- itemIdentifiers [0..n] : string. Each instance is identifying.

Associations

- reifier[0..1]: Topic – An optional Topic that reifies a TopicMapConstruct, by having the construct as its subject. Derived by a Topic subjectIndicator referencing a TopicMapConstruct.

Constraints

- It is an error for two different Topic Map Constructs to have source locators that are equal, expressed as the following OCL

```
context TopicMapConstruct inv:
    TopicMapConstruct.allInstances()->
     forAll(v_tmc1, v_tmc2 | v_tmc1.itemIdentifiers->
          forAll(v_sl1 | not(v_tmc2.itemIdentifiers-> includes(v_sl1)))
    )
```

Semantics

The itemIdentifiers assigned to a TopicMapConstruct allows references to it. ItemIdentifiers may be freely assigned to TopicMapConstructs based upon source syntax or other implementation defined methods.

15.1.2 TopicMap

Description

A Topic Map represents a particular view of a set of subjects. It is a collection of MapItems.

Similar Terms

RDF Graph, Ontology

Attributes

None.

Associations

- content[0..n]: MapItem – the set of instances of MapItem that are part of this TopicMap, derived from the union of topics and associations.
- topics [0..n]: Topic – the set of Topics that are contained in this topic map.
- associations [0..n]: Association – the set of Associations that are contained in this topic map.

Constraints

None.

Semantics

A TopicMap itself does not represent anything, and in particular has no subject associated with it. It has no significance beyond its use as a container for Topics and Associations and the information about subjects they represent.

15.1.3 MapItem

Description

MapItems are those TopicMapConstructs that make up the contents of a Topic Map; they are those constructs that a topic map can directly contain.

Attributes

None.

Associations

None.

Constraints

None.

Semantics

MapItems are an abstract class of items that may be part of a topic map.

15.1.4 Topic

Description

Topic is the fundamental MapItem in a Topic Map. The class diagram for Topic is shown in Figure 36. Each Topic represents a Subject in the domain of discourse.

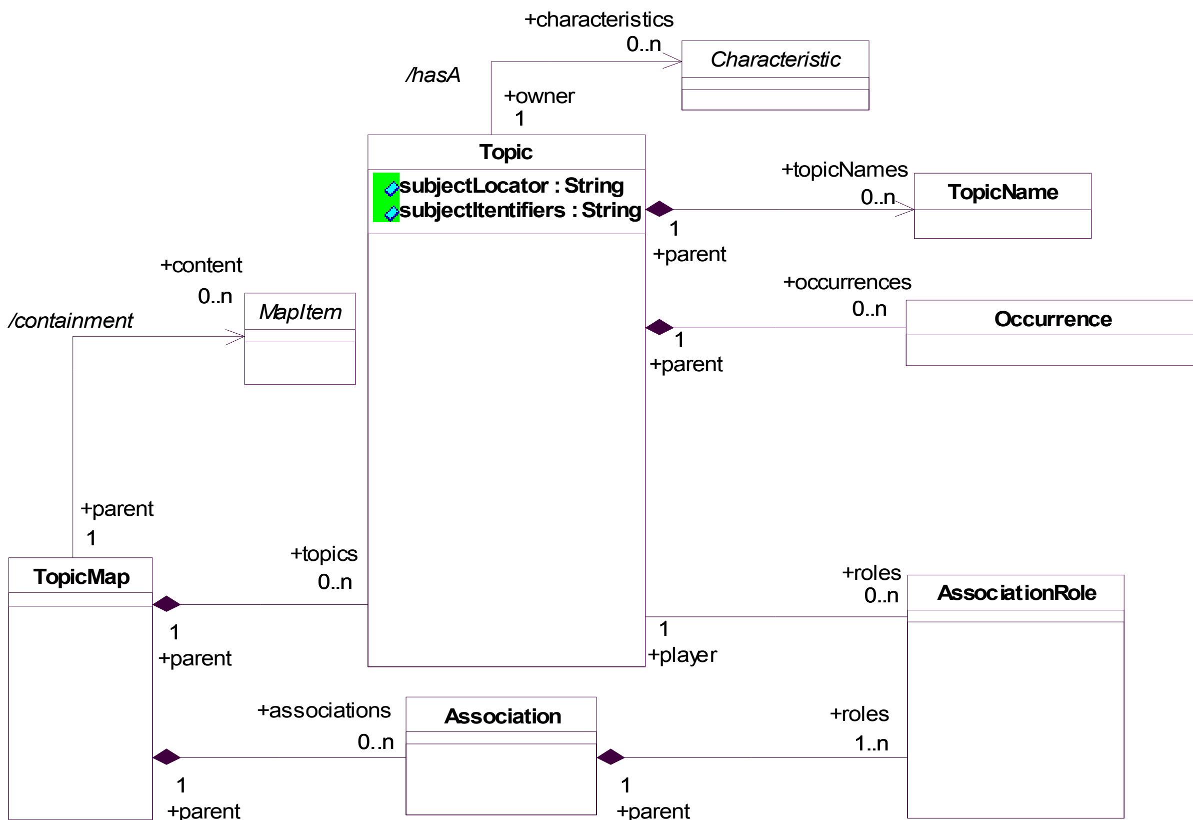


Figure 36 The Topic Class

Similar Terms

Node, Resource, Entity

Attributes

- subjectLocator[0..1]: string – an optional resource reference that locates a machine addressable subject.
- subjectIdentifiers[0..n]: string – the set of 0 or more resource references that identify a machine addressable indicator of a non-machine addressable subject.

Associations

- parent[1]:TopicMap – the required TopicMap that this Topic is part of.
- roles[0..n]: AssociationRole – the collection of AssociationRoles that are the roles that this Topic plays in Associations.
- occurrences[0..n]: Occurrence – the set of 0 or more occurrences for this Topic.
- topicNames[0..n]: TopicName – the set of 0 or more topic names for this Topic.
- characteristics[0..n]: Characteristic – the derived set of characteristics for this topic; it is the union of topicNames, occurrences, and roles
- reified[0..1]: TopicMapConstruct – A TopicMapConstruct may optionally be reified, becoming the subject of a Topic. A TopicMapConstruct is reified if it is another Topic's subjectIdentifier.

Constraints

- All topics must have a value for at least one of subject identifiers or subject locator that is neither the empty set nor null, expressed in the following OCL.

```
context Topic inv:
    self.subjectIdentifiers->notEmpty() or
    self.subjectLocator->notEmpty()
```

Semantics

Each instance of Topic is associated with exactly one Subject. A subject indicator, subject identifier or a subject locator identifies that subject. The Topic Map Data Model defines these terms in part as:

- A **subject indicator** is an information resource that is referred to from a topic map in an attempt to unambiguously identify the subject of a topic to a human being.
- A **subject identifier** is a locator that refers to a subject indicator.
- A **subject locator** is a locator that refers to the information resource that is the subject of a topic.

Topic maps contain only subject identifiers and subject locators, both of which refer to a subject indicator.

15.1.5 Association

Description

An Association is a multi-way relationship between one or more Topics. Associations must have a type and may be defined for a specified scope.

Similar Terms

Relation, Property

Attributes

None.

Associations

- parent[1]:TopicMap – the required TopicMap that this Association is part of.
- roles [1..n]:AssociationRole – An instance of Association is required to be linked to at least one instance of AssociationRole

Constraints

None

Semantics

The relationship defined by an Association is a relationship among the included Topic's subjects, rather that the Topics themselves.

15.2 Scope and Type

These '_able' abstract classes are intended as a concise mechanism to give a specific set of meta-classes in the TM meta-model the capability to be typed and scoped; those meta-classes are shown in Figure 35.

15.2.1 Scope_able

Description

Scope_able defines an abstract class that provides the TM scoping mechanism. Subclasses of Scope_able may have a defined scope of applicability.

Similar Terms

Context, Provenance, Qualification

Attributes

None.

Associations

- scope[0..n]: Topic – the topics which define the scope.

Constraints

None.

Semantics

If the scope association is empty, then the Scope_able items have the default scope.

15.2.2 Type_able

Description

Type_able defines an abstract class that provides the typing mechanism. Subclasses of Type_able must define types. Elements in TM are singly typed. A typed construct is an instance of its type. Type describes the nature of the represented construct.

Similar Terms

Type, isA, kindOf

Attributes

None.

Associations

- type [1..1]: Topic – the required topic which defines at most a single type.

Constraints

None.

Semantics

Typing is not transitive.

See also: Section 15.4 discussing published subjects.

15.3 Characteristics

Characteristics model the attributes of a Topic. They include the topic's names, topic occurrences and the roles that a topic plays in associations.

Each Topic has a set of Characteristics. Characteristics, as shown in Figure 35, include Association Roles, Occurrences and Topic Names.

Topic Names and Variant Names are human understandable labels for the Topic. While the primary Topic Name, termed a Base Name, is required to be a UNICODE string, variant names may include many data types not normally considered as 'names' such as icons, images or audio.

15.3.1 Characteristic

Description

Characteristic is the abstract base class of all Topic characteristics. It is a TopicMapConstruct, and must have a type and may be limited to a defined scope.

Similar Terms

Property, Attribute, Slot

Attributes

None.

Associations

None.

Constraints

None.

Semantics

Characteristic is an abstract class defining those items which may be characteristics of a Topic. It has no additional semantics.

15.3.2 AssociationRole

Description

An Association is composed of a collection of roles, which are played by Topics. The AssociationRole captures this relation. A Topic in an Association plays a particular part or role in the Association. This is specified in an Association Role. The Association and Association Role construct is similar to a UML Association or to an RDF Property.

Similar Terms

Role, UML Association End, UML Property

Attributes

None.

Associations

- parent[1]: Association – the required Association which the AssociatioRole is part of.
- player[1]: Topic – the required Topic that plays this role in the parent Association.

Constraints

None.

Semantics

An AssociationRole is the representation of the participation of subjects in an association. The association role has a topic playing the role and a type that defines the nature of the participation of the player in the association. The roles and associations are representing the relationships between the participating Topic's subject, rather than the topics themselves.

15.3.3 Occurrence

Description

An Occurrence is a Characteristic that is very similar to an attribute. Occurrences are Scope_able and Type_able. The value of the occurrence is specified by the locator role in the association with the abstract meta-class Resource, as shown in Figure 37 . The interpretation of the Resource is defined by its concrete specializations.

Similar Terms

Attribute, Slot

Attributes

- value[1]:string – If the datatype is IRI, a locator referring to the information resource the occurrence connects with the subject, otherwise the string is the information resource.
- datatype[1]:string – A locator identifying the datatype of the occurrence value.

Associations

None.

Constraints

None.

Semantics

It may be mistakenly inferred by the name 'Occurrence' that this Characteristic refers only to instances of a Topic. This is not the case. An Occurrence may be any descriptive information about a Topic, including instances, and may represent any characteristic of a Topic, including an 'occurrence' or instance of the subject. Occurrences are semantically similar to UML Attributes.

.

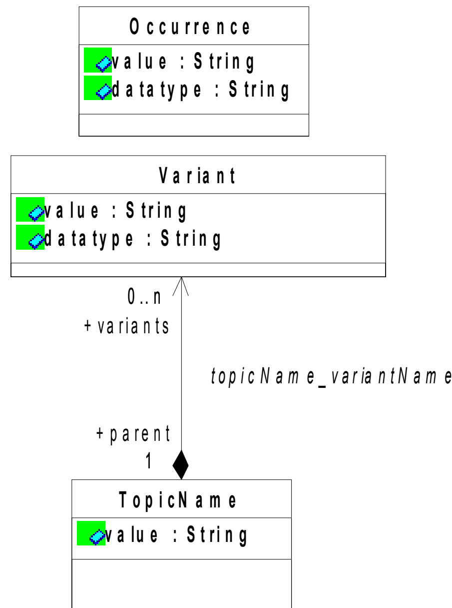


Figure 37 Topic Name Class

15.3.4 TopicName

Description

A TopicName is intended to provide a human readable text name for a topic.

Similar Terms

Label, Comment, Description (Brief)

Attributes

- value: String – The Base Name for this Topic; the string is UNICODE.

Associations

- variants[0..n]: VariantName – Zero or more variations of the TopicName.

Constraints

None.

Semantics

The term ‘name’ should not be misconstrued to imply uniqueness. Neither the topic name, nor it variants, are identifying; they serve only as human readable labels.

15.3.5 VariantName

Description

VariantName allows alternative names for a Topic to be specified. These names may be any format, including text, documents, images or icons. VariantNames must have scope.

Similar Terms

Label, Comment, Description (Brief), Icon

Attributes

- value[1]:string – If the datatype is IRI, a locator referring to the information resource the occurrence connects with the subject, otherwise the string is the information resource.
- datatype[1]:string – A locator identifying the datatype of the occurrence value.

Associations

- scope[1..n] : Topic – The topics which define the scope .

Constraints

- A VariantName is restricted to being a composite part of a TopicName. It cannot exist as a standalone construct as constrained by the topicName_variantName association multiplicity.

Semantics

Like TopicName variant names are not identifying.

15.4 Published Subjects

A Core set of Topic instances, termed Published Subjects, has been defined as part of the TM standard. These topics represent special instances of the TM meta-model and any implementation of the TM meta-model should handle these items as special, reserved topics with meanings as defined in Section 7 of the Topic Map Data Model [TMDM].

In summary, they represent five key areas:

- Types and Instance –Types and their instances are related by three subjects representing the type-instance association and, the type and instance association roles. A type is an abstraction that captures characteristics common to a set of instances. A type may itself be an instance of another type, and the type-instance relationship is not transitive.
- Super and Sub Types – Types may be arranged into a type hierarchy using the supertype-subtype association and, supertype and subtype association roles. The supertype-subtype relationship is the relationship between a more general type (the supertype) and a specialization of that type (the subtype). The supertype-subtype relationship is transitive.
- Special Variant Names – Display and Sort are two special types of variant names appropriate for human display and sorting.
- Uniqueness – A unique topic characteristic can be used to definitively identify a topic.
- Topic Map Constructs –Subjects that represent the reification of topic map constructs, such as association, associations-role or occurrence.
- These published subjects are identified by uri with base http://psi.topicmaps.com/iso13250/, called in the ODM by the Qname prefix ‘tmcore:’.

15.5 Example

Figure 38 depicts a simple instance model of the TM meta-model. The model depicted represents the following statements:

- A Personal Car is a Car (which may be owned by a Person).
- A Car is a Vehicle (which may have a Color).
- Carl is a person that owns a Personal Car that is red.

The parenthetical statements are not directly represented in Topic Maps.

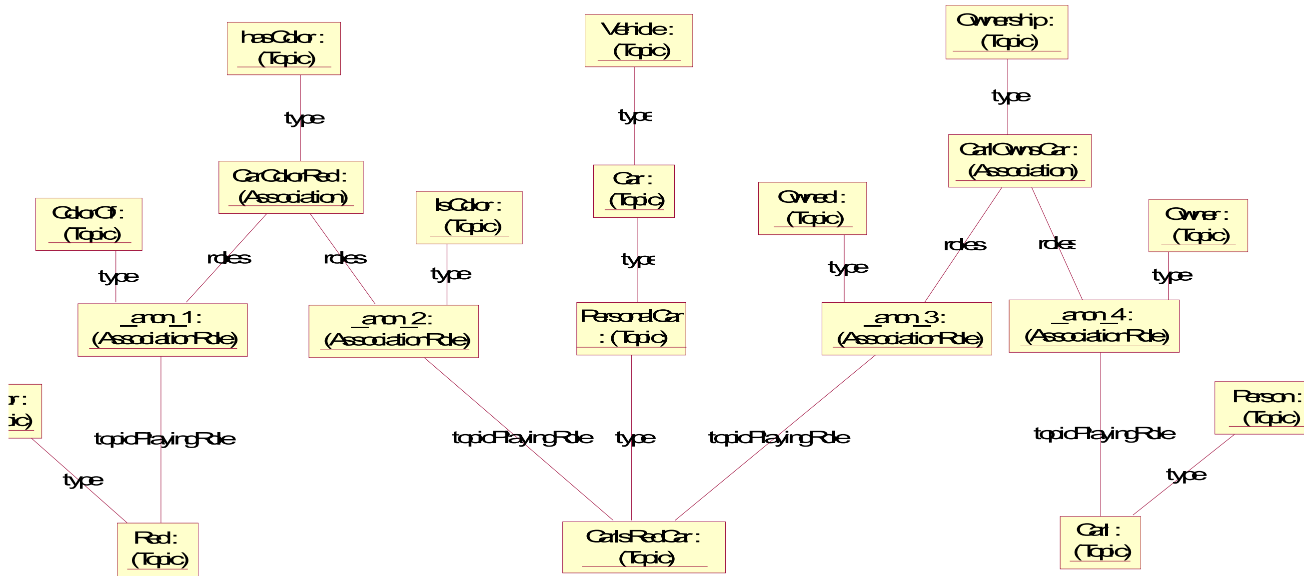


Figure 38 Instance of Topic Map Metamodel

16 UML Profiles for RDF Schema and OWL

This profile is based on the UML Kernel package defined in "Unified Modeling Language: Superstructure", version 2 [UML2] as well as on the Profiles section of the same document. It is designed to support modelers developing vocabularies or taxonomies in RDF Schema, or richer ontologies in the Web Ontology Language, through reuse of UML notation using tools that support UML2 extension mechanisms. It

- Reuses UML constructs when they have the same semantics as OWL, or, when this is not possible, stereotypes UML constructs that are consistent and as close as possible to OWL semantics.
- Uses standard UML 2 notation, or, in the few cases where this is not possible (stereotype property notation), follows the clarifications and elaborations of stereotype notation being proposed by the UML extension for Systems Engineering[6].
- Leverages the model library provided in Appendix A, Foundation Ontology (M1) for RDFS and OWL.

The profile has been partitioned to support users who wish to restrict their vocabularies to RDF Schema, as well as to reflect the structure of the RDFS and OWL metamodels (and the languages themselves). It leverages stereotypes extensively and also uses a few tagged values in traditional fashion. It depends on the metamodels defined in Chapter 11, The RDF Schema Metamodel, and in Chapter 12, The OWL Metamodel, respectively, for overall structure, semantics and language mapping. It also depends on the model libraries included in Appendix A, Foundation Ontology (M1) for RDFS and OWL, for certain basic definitions, such as the M1 level elements discussed in Chapter 8, Design Rationale.

16.1 UML Profile for RDF Schema

The UML profile for RDF Schema is organized similarly to the structure of the RDFS Metamodel, but with additional initial sections describing optional extensions to the basic metamodel to support the structure of RDF documents and the RDF graph model, respectively.

16.1.1 RDF Document Syntax (Optional)

RDF is the place in the Semantic Web "layer cake" where the languages (including RDF and OWL) are fitted to the Web. As a result, a few elements are included that are really part of the web architecture, including namespaces, for example, defined in the RDF syntax specification. This may appear to introduce unnecessary overhead or complexity, but in fact, these elements are necessary for a complete metamodel designed to support interoperability across modeling paradigms.

Concepts including RDF document, namespaces, the definitions that map namespaces to namespace prefixes, and the associations between a set of statements and the document that contains them facilitate the systematic exchange of these definitions across modeling environments, and can be mapped to similar features in a Common Logic ontology, Topic Map, UML model, or ER conceptual model.

Figure 39 specifies several concepts that link an RDF document to the names and statements it contains. While both documents and graphs may have sets of statements associated with them, namespace definitions and the mappings between namespace prefixes and URIs are associated with RDF documents (in this simplified view of XML Schema - in actuality, they are associated with XML elements), not with RDF graphs.

---

6. Systems Modelling Language (SysML) Specification, Addendum to SysML v0.9, Profiles and Model Libraries Chapter, http://doc.omg.org/ad/05-06-01.pdf.

Note that the model supports multiple graphs within a document, and the notion that a particular graph may cover multiple documents. While in common practice there can be a one to one correspondence between a document and a graph, examples of both kinds of exceptions are included in the set of RDF specifications defining the language and in related W3C documents.

Single graph covering multiple documents

The ability to refer to definitions that are external to a particular document (*e.g.*, XML Schema Datatypes), and in OWL, the ability to directly import such definitions, naturally extends a graph beyond the boundaries of a single document. Additionally, in [RDF Primer], there is a discussion of the use of XML Base, such that relative URIs may be defined based on a base URI other than that of the document in which they occur. This may be appropriate, for example, when there are mirror sites that share common definitions and extend them at the mirror site, but where it is not necessary to duplicate all definitions at every such site. In such cases, a graph can span multiple documents, and the URI of the mirror site document is distinct from that of its base. As a result, the metamodel provides for the optional definition of an `xml:base` distinct from the URI of the document.

Multiple graphs in the same document

It is common practice in ontology development to have multiple "main nodes" in the same document - for example, multiple concepts whose parent class is simply `owl:Thing`, or classes without a defined "parent class" in RDF. Some explicit examples are provided in the discussion of Named Graphs (see http://www.w3.org/2004/03/trix/, particularly those given on the TriG Homepage, at http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/). One can imagine others such as when defining SKOS-based concept schemes, or thesauri, and managing multiple versions of such schemes (see the SKOS Core Guide, http://www.w3.org/TR/swbp-skos-core-guide, and http://www.w3.org/TR/swbp-thesaurus-pubguide, for more information). The ability to name a graph provides a means by which multiple component graphs defined in the same document can be referenced externally as a unit, enabling graph mapping and alignment, for example. Thus, the optional name attribute on the Graph class supports naming graphs for those applications that require this feature. While the notion of a named graph is not yet part of the formal RDF W3C recommendations, emerging work on SKOS vocabularies and SPARQL confirms that use of named graphs is becoming increasingly important to applications, and is considered mainstream.

Bounding an RDF vocabulary

The notion of scope is somewhat opaque in the current set of recommendations that together define RDF and its vocabulary language, RDF Schema. This is, in part, due to the fact that URIs have global scope in RDF. Yet, we need a way of talking about and modeling the set of resources that describe a particular vocabulary. Each document is associated with a resource whose URI reference is the primary URL where the document is published. It is good practice to include this URL in the serialized form of an RDF XML document, as the value of an `xml:base` on its root element. The bounds of a particular RDF vocabulary is the collection of statements (triples) sharing a base URI, or, in the absence of such a URI, a graph, whose base URI is, by default, that of the document that contains it.

Qualified Names and Transformations

Instructions regarding how QNames and rdf:ID attribute values can be transformed into RDF URI references are defined in [RDF Syntax]. Additionally, RDF/XML allows further abbreviating RDF URI references through the use of the XML Infoset mechanism for setting a base URI that is used to resolve relative RDF URI references (`xml:base`), or by considering the base URI to be that of the document. The base URI applies to all RDF/XML attributes that deal with RDF URI references, including `rdf:about`, `rdf:resource`, `rdf:ID` and `rdf:datatype`. (See http://www.w3.org/TR/xmlbase/ for more on XML Base.)

Secondly, the `rdf:ID` attribute on a node element (not property element) can be used instead of `rdf:about` and gives a relative RDF URI reference equivalent to '#' concatenated with the `rdf:ID` attribute value. So for example if `rdf:ID="name"`, that would be equivalent to `rdf:about="#name"`. `rdf:ID` provides an additional check since the same name can only appear once in the scope of an `xml:base` value (or document, if none is given), so is useful for defining a set of distinct, related terms relative to the same RDF URI reference.

Both forms require a base URI to be known, either from an in-scope `xml:base`, or, in the case of a reference to a definition outside of the current document, from the URI of the RDF/XML document in which the target definition is specified.

Figure 39 RDF Document Definitions

16.1.1.1 Document

Description

RDF's conceptual model is a graph. RDF also provides an XML syntax for writing down and exchanging RDF graphs, called RDF/XML. An RDF document is a serialization of an RDF graph into a concrete syntax, as specified in [RDF Syntax], which provides the container for the graph, and conventionally also contains declarations of the XML namespaces referenced by the statements in the document.

RDF refers to a set of URI references as a vocabulary. Often, the URI references in such vocabularies are organized so that they can be represented as sets of QNames using common prefixes. URI references that are contained in the vocabulary are formed by appending individual local names to the relevant prefix. This practice is also commonly used in OWL ontology development for improved readability. While the metamodel does not explicitly support QNames, the elements required to enable such support in vendor implementations are provided.

Attributes

None.

Associations

- localName: LocalName [0..*] in association DocumentContainsLocalName - links a document to the set of local names it contains
- namespaceDefinition: NamespaceDefinition [0..*] in association NamespaceDefinitionForDocument - links a document to zero or more namespace definitions that may be used in any RDF (or OWL) assertions contained within the document
- statement: RDFStatement [1..*] in association StatementForDocument - links a document to the set of triples (statements) it contains
- xmlBase: Namespace [0..*] in association DefaultNamespaceForDocument - links a document to one or more default namespaces (xml:base namespaces) associated with the statements in the document
- Specialize Class RDFSResource - an RDF/S document is a resource

Constraints

[1] A document must have a URI.

[2] Local names with URIs that match the URI of the document are contained by (local to) the document.

Semantics

An RDF/XML document is only required to be well-formed XML; it is not intended to be validated against an XML DTD (or an XML Schema).

16.1.1.2 LocalName

Description

RDF uses an RDF URI Reference, which may include a fragment identifier, as a context free identifier for a resource. The meaning of a fragment identifier depends on the MIME content-type of a document, i.e. is context dependent.

These apparently conflicting views are reconciled by considering that a URI reference in an RDF graph is treated with respect to the MIME type `application/rdf+xml`. Given an RDF URI reference consisting of an absolute URI and a fragment identifier, the fragment identifier identifies the same thing that it does in an `application/rdf+xml` representation of the resource identified by the absolute URI component.

The typical practice is to split a URI reference into two parts such that the right is maximal being an NCName as specified by XML Namespaces, which might best be implemented by vendors as a method on the model. Atypical (but formally permitted) practice includes allowing multiple LocalNames for each URIReference, *i.e.* any split as above, without the right part being maximal. Also note that some URIrefs (specifically those suggested for user defined datatypes in XML Schema) cannot be split in this way, since they have no rightmost NCName.

The definitions provided in this extension to the RDFS metamodel are also sufficient to generate QNames: split each URI reference as above (or using LocalName), look the first half up as a namespace, and then form a qname.

Attributes

- name: String [1] - the string representing the local name or fragment identifier

Associations

- document: Document [0..*] in association DocumentContainsLocalName - links local names to the document that contains them
- uriRef: URIReference [1..*] in association FragmentIdentifierForURIRef - links the fragment identifier to one or more URIs that reference it

Constraints

None.

Semantics

None.

16.1.1.3 Namespace

Description

An XML namespace is a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

Attributes

None.

Associations

- document: Document [1..*] in association DefaultNamespaceForDocument - the document(s) for which it is the default namespace (or `xml:base`)
- namespaceDefinition: NamespaceDefinition [0..*] in association NamespaceForNamespaceDefinition - links a namespace definition to the namespace it describes (resolves to)
- namespaceURIRef: URIReference [1] in association URIReferenceForNamespace - links a namespace to the corresponding URI reference

Constraints

Namespaces should conform to the specification given in [XMLNS]. While it may not be possible to define constraints on character strings in OCL to enforce this (and while the namespace recommendation may not explicitly require enforcement), tools that implement this metamodel will be expected to support the W3C standards and related RFCs to the extent possible.

Semantics

None.

16.1.1.4 NamespaceDefinition

Description

A namespace is declared using a family of reserved attributes. These attributes, like any other XML attributes, may be provided directly or by default. Some names in XML documents (constructs corresponding to the nonterminal Name) may be given as qualified names. The prefix provides the namespace prefix part of the qualified name, and must be associated with a namespace URI in a namespace declaration.

Namespace definitions are used in RDF and OWL for referencing and/or importing externally specified terms, vocabularies or ontologies.

Attributes

- namespacePrefix: String [1] - the string representing the namespace prefix

Associations

- document: Document [1..*] in association NamespaceDefinitionForDocument - the document(s) using the namespace definition
- resolvesTo: Namespace [1] in association NamespaceDefinitionForNamespace - indicates that a namespace definition, if it exists, resolves to exactly one namespace

Constraints

[1] Namespace definitions should conform to the specification given in [XMLNS].

Semantics

None.

16.1.1.5 RDFStatement (Modified Definition)

Description

An RDF triple contains three components:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference, and represents a relationship
- the object, which is an RDF URI reference, a literal or a blank node

An RDF triple is conventionally written in the order subject, predicate, object. The relationship represented by the predicate is also known as the property of the triple. The direction of the arc is significant: it always points toward the object.

Attributes

- isReifiedOnly: Boolean [0..1] - indicates that a particular statement (triple) is reified but not asserted

Associations

- document: Document [1..*] in association StatementForDocument - the document(s) containing the statement
- graph: Graph [1..*] in association StatementForGraph - the graph(s) containing the statement
- isReifiedBy: URIReference [0..*] in association ReificationForStatement - the URI reference that reifies the statement
- RDFsubject: RDFSResource [0..1] in association SubjectForStatement - links a statement (triple) to the resource (node) that is the subject of the triple
- RDFpredicate: RDFProperty [0..1] in association PredicateForStatement - links a statement (triple) to the property that is the predicate of the triple
- RDFobject: RDFSResource [0..1] in association ObjectForStatement - links a statement (triple) to the resource (node) that is the object of the triple

Constraints

[1] The resource representing an RDFsubject can be an URI reference or a blank node but not a literal.

```
context RDFStatement SubjectNotALiteral inv:
   not self.RDFsubject.oclIsKindOf(RDFSLiteral)
```

[2] An RDFpredicate must be a URI reference (*i.e.*, must not be a literal or blank node).

```
context RDFStatement PredicateNotALiteral inv:
   not self.RDFpredicate.oclIsKindOf(RDFSLiteral)
context RDFStatement PredicateNotABlankNode inv:
   not self.RDFpredicate.oclIsKindOf(BlankNode)
```

Note: both of these constraints are subject to change (may be relaxed) based on user experience in the Semantic Web community. However, in any case, the constraint that a predicate must not be a literal is likely to remain.

Semantics

Each triple represents a statement of a relationship between the things denoted by the nodes that it links. The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. The subject, predicate, and object of a triple are optional to support reified but unasserted triples.

16.1.1.6 UniformResourceIdentifier

Description

The RDF abstract syntax is concerned primarily with URI references. The definition of a URI, distinct from URI reference, is included for mapping purposes. See [RDF Syntax] for definition details.

Note: this class is included primarily for syntactic/mapping/interoperability with other ODM and external metamodels, and may be considered optional depending on vendor requirements.

Attributes

- name: String [1] – the string representing the URI

Associations

- uriReference: URIReference [0..*] in association URIForURIReference - zero or more URI references associated with the URI
- Specialize Class URIReference - A UniformResourceIdentifier is a URIReference

Constraints

URIs must conform to the character encoding (including escape sequences and so forth) defined in [RDF Syntax] and are globally defined. This is in contrast to naming and namespace conventions in UML2, which can be limited to the package level or to a set of nested namespaces. While it may not be possible to define constraints on character strings in OCL to enforce this, tools that implement this metamodel will be expected to support the W3C standards and related RFCs in this regard.

Semantics

None.

16.1.1.7 URIReference

Description

RDF uses URI references to identify resources and properties. A URI reference within an RDF graph (an RDF URI reference) is a Unicode string conforming to the characteristics defined in [RDF Concepts] and [RDF Syntax].

RDF URI references ([RDF Concepts] Section 3.1) can be either:

- given as XML attribute values interpreted as relative URI references that are resolved against the in-scope base URI as described in section 5.3 to give absolute RDF URI references
- transformed from XML namespace-qualified element and attribute names (QNames)
- transformed from rdf:ID attribute values.

More on URI references and transformations from QNames is given in the discussion in [RDF Syntax].

Attributes

None.

Associations

- fragmentIdentifier: LocalName [0..1] in association FragmentIdentifierForURIRef - links URIReference to an optional fragment identifier
- namedGraph: Graph [0..1] in association NameForGraph - links a URI reference to the graph it names
- namespace: Namespace [0..1] in association URIReferenceForNamespace - links a URI reference to a namespace
- reifies: RDFStatement [0..*] in association ReificationForStatement - links URIReference to zero or more statements it reifies
- resource: RDFSResource [0..*] in association URIRefForResource - links a URI reference to a resource
- uri: UniformResourceIdentifier [1] in association URIForURIReference - links URIReference to an in-scope xml:base or to the URI of the RDF/ XML document that contains the definition referenced

Constraints

[1] URI references must conform to the specifications given under Description, above. While it may not be possible to define constraints on character strings in OCL to enforce this, tools that implement this metamodel will be expected to support the W3C standards and related RFCs in this regard.

[2] A non-empty fragmentIdentifier associated with an empty uri implies that the uri is the `xml:base` (default namespace) of the document.

Semantics

Two RDF URI references are equal if and only if they compare as equal, character by character, as Unicode strings.

16.1.2 RDF Graph Model (Optional)

For those applications that require use of the RDF graph model, as specified in [RDF Concepts], support for explicit manipulation of blank nodes may be needed. The definitions provided herein facilitate resolution of blank node semantics and finer granularity in manipulation of nodes in an RDF graph.

Statements in RDF can be reified and/or asserted. Reification enables us to make statements about statements, and in fact, we can make multiple statements about a given triple, which is supported through the relationship with URI reference. If a statement is asserted, its subject, predicate and object roles must be filled. Reification is a difficult concept to grasp and is not required by all applications, thus is considered optional.

URIReferenceNode, BlankNode and RDFSLiteral form a complete covering of RDFSResource and are pairwise disjoint.

These definitions require/depend on the extensions given in 16.1.1 ("RDF Document Syntax (Optional)"), above.

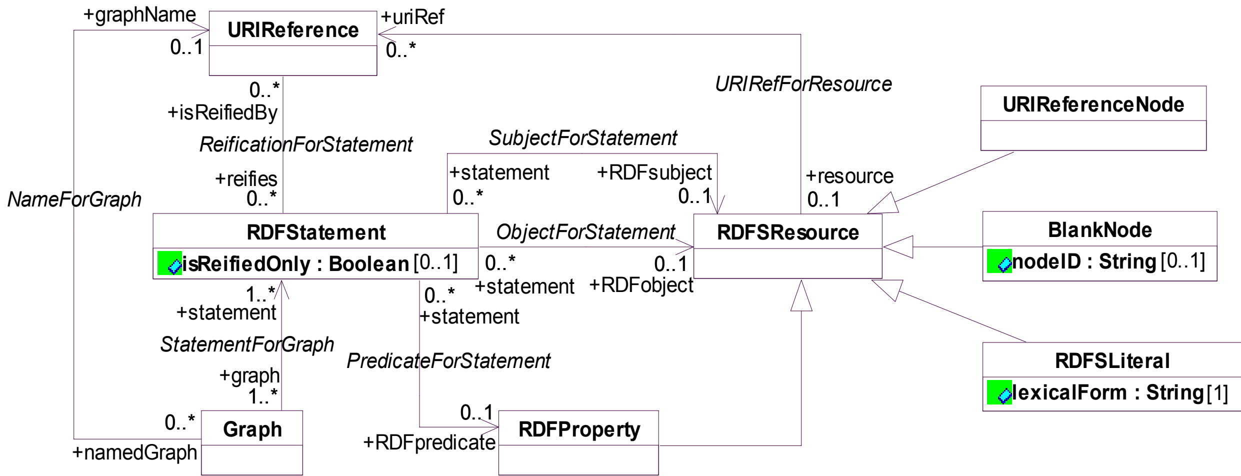


Figure 40 RDF Graph, Node & Statement Definitions

16.1.2.1 BlankNode

Description

A blank node is a node that is not a URI reference or a literal. In the RDF abstract syntax, a blank node is simply a unique node that can be used in one or more RDF statements, but has no intrinsic name.

A convention used to refer to blank nodes by some linear representations of an RDF graph is to use a blank node identifier, which is a local identifier that can be distinguished from URIs and literals. When graphs are merged, their blank nodes must be kept distinct if meaning is to be preserved. Blank node identifiers are not part of the RDF abstract syntax, and the representation of triples containing blank nodes is dependent on the particular concrete syntax used, thus no constraints are provided here on blank node identifiers. They are included strictly as a placeholder for tool vendors whose applications require them, and in particular, for interoperability among such tools.

Attributes

- nodeID: String [0..1] - is a placeholder for an optional blank node identifier

Associations

- Specialize Class RDFSResource

Constraints

[1] BlankNode is pairwise disjoint from URIReferenceNode and RDFSLiteral.

[2] BlankNode, RDFSLiteral, and URIReferenceNode form a complete covering of RDFSResource.

[3] BlankNode must not inherit a URI from RDFSResource.

Semantics

RDF makes no reference to the internal structure of blank nodes. However, given two blank nodes, it should be possible to determine whether or not they are the same. The methodology for making such a determination is left to the applications that use them, for example, through reasoning about them.

Blank nodes are treated as simply indicating the existence of a thing, without using, or saying anything about, the name of that thing. (This is not the same as assuming that the blank node indicates an 'unknown' URI reference; for example, it does not assume that there is any URI reference which refers to the thing.) Thus, they are essentially treated as existentially quantified variables in the graph in which they occur, and have the scope of the entire graph. More on the semantics of blank nodes is given in [RDF Semantics].

16.1.2.2 Graph

Description

An RDF graph is a set of RDF triples. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph.

Attributes

None.

Associations

- graphName: URIReference [0..1] in association NameForGraph - the optional name of a named graph, which must be a URI reference
- statement: RDFStatement [1..*] in association StatementForGraph - links a graph to the set of triples it contains

Constraints

None.

Semantics

As described in [RDF Semantics], RDF is an assertional language, intended for use in defining formal vocabularies and using them to state facts and axioms about some domain.

An RDF graph is defined as a set of RDF triples. A subgraph of an RDF graph is a subset of the triples in the graph. A triple is identified with the singleton set containing it, so that each triple in a graph is considered to be a subgraph. A proper subgraph is a proper subset of the triples in the graph. A ground RDF graph is one with no blank nodes.

The assertion of an RDF triple says that some relationship, indicated by the predicate, holds between the things denoted by subject and object of the triple. The assertion of an RDF graph amounts to asserting all the triples in it, so the meaning of an RDF graph is the conjunction (logical AND) of the statements corresponding to all the triples it contains.

16.1.2.3 RDFProperty (Modified Definition)

Description

The RDF Concepts and Abstract Syntax specification [RDF Concepts] describes the concept of an RDF property as a relation between subject resources and object resources.

Every property is associated with a set of instances, called the property extension. Instances of properties are pairs of RDF resources.

Associations

- RDFSdomain: RDFSClass [0..*] in association DomainForProperty - links a property to zero or more classes representing the domain of that property. A triple of the form: `P rdfs:domain C` states that `P` is an instance of the class `rdf:Property`, that `C` is a instance of the class `rdfs:Class` and that the resources denoted by the subjects of triples whose predicate is `P` are instances of the class `C`. Where a property `P` has more than one `rdfs:domain` property, then the resources denoted by subjects of triples with predicate `P` are instances of all the classes stated by the `rdfs:domain` properties.
- RDFSrange: RDFSClass [0..*] in association RangeForProperty - links a property to zero or more classes representing the range of that property. A triple of the form: `P rdfs:range C` states that `P` is an instance of the class `rdf:Property`, that `C` is a instance of the class `rdfs:Class` and that the resources denoted by the objects of triples whose predicate is `P` are instances of the class `C`. Where `P` has more than one `rdfs:range` property, then the resources denoted by the objects of triples with predicate `P` are instances of all the classes stated by the `rdfs:range` properties.
- RDFSsubPropertyOf: RDFProperty [0..*] in association PropertyGeneralization - links a property to another property that generalizes it. The property `rdfs:subPropertyOf` is used to state that all resources related by one property are also related by another. A triple of the form: `P1 rdfs:subPropertyOf P2` states that `P1` is an instance of `rdf:Property`, `P2` is an instance of `rdf:Property` and `P1` is a subproperty of `P2`. The `rdfs:subPropertyOf` property is transitive.
- statement: RDFStatement [0..*] in association PredicateForStatement - links a statement (triple) to the predicate of that triple
- superPropertyOf: RDFProperty [0..*] in association PropertyGeneralization - links a property to another property that specializes it (note that superPropertyOf is not an RDF concept).
- Specialize Class RDFSResource - the class `rdf:Property` is a subclass of `rdfs:Resource`

Constraints

[1] The predicate of an RDF triple is a URI Reference (thus, a resource that is an RDF property used as the predicate of a statement must have a URI reference).

Semantics

A property relates resources to resources or literals. A property can be declared with or without specifying its domain (*i.e.*, classes which the property can apply to) or range (*i.e.*, classes or datatypes that supply its values). Properties may be specialized (subPropertyOf). The existence of an instance of a specializing property implies the existence of an instance of the specialized property, relating the same set of resources.

16.1.2.4 RDFSLiteral (Modified Definition)

Description

Literals are used to identify values such as numbers and dates by means of a lexical representation. Anything represented by a literal could also be represented by a URI, but it is often more convenient or intuitive to use literals.

A literal may be the object of an RDF statement, but not the subject or the predicate.

Literals may be plain or typed:

- A plain literal is a string combined with an optional language tag. This may be used for plain text in a natural language.

- A typed literal is a string combined with a datatype URI.

Attributes

- lexicalForm: String [1] - represents a Unicode string in Normal Form C.

Associations

- Specialize Class RDFSResource - All literals are resources

Constraints

[1] RDFSLiteral is pairwise disjoint from URIReferenceNode and BlankNode.

[2] BlankNode, RDFSLiteral, and URIReferenceNode form a complete covering of RDFSResource.

[3] RDFSLiteral must not inherit a URI from RDFSResource.

[4] A literal may be the object of an RDF statement, but not the subject or predicate.

[5] PlainLiteral and TypedLiteral are disjoint.

Semantics

Plain literals are self-denoting. Typed literals denote the member of the identified datatype's value space obtained by applying the lexical-to-value mapping to the literal string.

16.1.2.5 RDFSResource (Modified Definition)

Description

All things described by RDF are called resources. This is the class of everything. All other RDF classes are subclasses of this class.

Attributes

None.

Associations

- list: RDFList [0..*] in association FirstElementInList - relates a particular resource to the list(s) for which it is the initial element
- RDFScomment: PlainLiteral [0..*] in association CommentForResource - links a resource to a comment, or human-readable description, about that resource
- RDFSisDefinedBy: RDFSResource [0..*] in association IsDefinedByResource - relates a resource to another resource that defines it; rdfs:isDefinedBy is a subPropertyOf rdfs:seeAlso.
- resource: RDFSResource [0..*] in association IsDefinedByResource - relates a particular resource to other resources that it defines
- RDFSlabel: PlainLiteral [0..*] in association LabelForResource - links a resource to a human-readable name for that resource.
- RDFSmember: RDFSResource [0..*] in association MemberOfResource - relates a resource to another resource of which it is a member (*i.e.* a resource that contains it).
- resource: RDFSResource [0..*] in association MemberOfResource - relates a particular resource to other resources that are its members

- RDFSseeAlso: RDFSResource [0..*] in association SeeAlsoForResource - relates a resource to another resource that may provide additional information about it.
- resource: RDFSResource [0..*] in association SeeAlsoForResource - relates a particular resource to other resources that it may assist in defining
- RDFtype: RDFSClass [1..*] in association TypeForResource - relates a resource to its type (i.e., states that the resource is an instance of the class that is its type).
- RDFvalue: RDFSResource [0..*] in association ValueForResource - relates a resource to its value (this is an idiomatic expression in RDF that may be used in describing structured values).
- resource: RDFSResource [0..*] in association ValueForResource - relates a value to a resource
- statement: RDFStatement [0..*] in association SubjectForStatement - a resource represents zero or more subjects of RDF statements or triples
- statement: RDFStatement [0..*] in association ObjectForStatement - a resource represents zero or more objects of RDF statements
- uriRef: URIReference [0..1] in association URIRefForResource - the optional URI reference associated with every resource

Constraints

[1] RDFSseeAlso and RDFSisDefinedBy must have non-empty URI references.

[2] RDFSisDefinedBy is a subPropertyOf RDFSseeAlso

[3] The set of blank nodes, the set of all RDF URI references (i.e., URIReferenceNodes) and the set of all literals are pairwise disjoint.

```
context RDFSResource inv DisjointPartition:
  (self.oclIsKindOf(URIReferenceNode) xor self.oclIsKindOf(BlankNode)) and
  (self.oclIsKindOf(URIReferenceNode) xor self.oclIsKindOf(RDFSLiteral)) and
  (self.oclIsKindOf(BlankNode) xor self.oclIsKindOf(RDFSLiteral))
```

[4] URIReferenceNode, BlankNode and RDFSLiteral form a complete covering of RDFSResource.

Semantics

The uriRef attribute is used to uniquely identify an RDF resource globally. Note that this attribute has a multiplicity of [0..1] which provides for the possibility of the absence of an identifier, as in the case of blank nodes and literals.

16.1.2.6 URIReferenceNode

Description

A URI reference or literal used as a node identifies what that node represents. URIReferenceNode is included in order to more precisely model the intended semantics in UML (*i.e.*, not all URI references are nodes). A URI reference used as a predicate identifies a relationship between the things represented by the nodes it connects. A predicate URI reference may also be a node in the graph.

Attributes

None.

Associations

- Specialize Class RDFSResource

Constraints

[1] URIReferenceNode is pairwise disjoint from RDFSLiteral and BlankNode.

[2] BlankNode, RDFSLiteral, and URIReferenceNode form a complete covering of RDFSResource.

[3] URIReferenceNode must inherit a URI from RDFSResource.

Semantics

No additional semantics.

16.1.3 RDF Schema Profile Package

The following sections specify the set of stereotypes and tagged values that comprise the UML2 profile for using UML notation to represent RDF Schema vocabularies, and extends the RDFS metamodel. There are essentially three compliance modes for the RDF Schema profile: Basic, Document Model, and Graph Model. The Basic model corresponds to the metamodel given in Chapter 11, The RDF Schema Metamodel, without extensions. The Document Model assumes that the extensions described in 16.1.1 ("RDF Document Syntax (Optional)") are important to the application and requires all of the Basic stereotypes as well as those defined in 16.1.5 ("RDF Document (optional)"), below. The Graph Model assumes that the extensions described in 16.1.2 ("RDF Graph Model (Optional)") are important and requires a combination of the Basic and Document Model stereotypes, as described in 16.1.10 ("RDF Graphs and Nodes (optional)").

The package shown in Figure 41, below provides the overall container for the profile itself. The RDFSchema profile depends on and extends the RDFS metamodel.

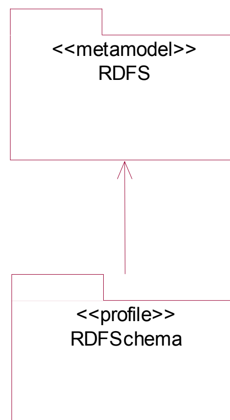


Figure 41 RDF Schema Profile Package

Constraints

- All classes in an RDFSchema package must be stereotyped by «rdfsClass».

16.1.4 RDFS Ontology

An ontology is the primary scoping mechanism for a set of graphs in the basic RDFS metamodel. This profile element maps directly to the RDFS metamodel elements given in 11.7 ("The Ontology Diagram"), as shown in Figure 42.

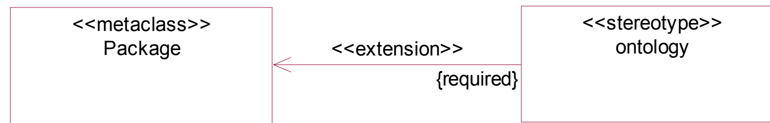


Figure 42 RDFS Ontology Package

Applying the «ontology» stereotype to a package requires that the UML constructs contained within the package be interpreted according to this profile definition.

16.1.5 RDF Document (optional)

As shown in Figure 43, if the optional metamodel and profile extensions to support RDF documents are implemented, an RDF document represents the primary scoping mechanism / container for an RDF/S vocabulary or OWL ontology and replaces the RDF ontology package defined in 16.1.4 ("RDFS Ontology"), above. This profile element maps directly to the RDFS metamodel extension representing such a document as discussed in 16.1.1.1 ("Document").

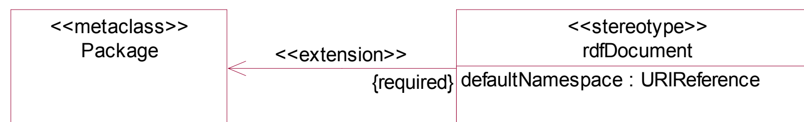


Figure 43 RDFDocument Provides an Alternate Container For RDFS Vocabularies and OWL Ontologies

Applying the «rdfDocument» stereotype to a package requires that the UML constructs contained within the package be interpreted according to this profile definition. If this approach is taken, the Ontology package specified in 16.1.4 ("RDFS Ontology") is not required.

Note that in order to annotate the RDF document, it may be necessary to create a corresponding UML class with the same stereotype, such that annotation properties can be attached to the class. This mechanism would also be useful in cases where UML tools do not support properties on packages.

16.1.5.1 Tagged Values

- defaultNamespace: URIReference [0..1] – provides the default namespace, or base for the document, if available, and maps directly to the default namespace for the document defined in the metamodel, as shown in Figure 39.

16.1.5.2 RDF Document Stereotypes

RDFS and OWL namespace declarations are associated with the *RDF document* that acts as the container for the set of RDF graphs that make up the vocabulary or ontology component, rather than with the optional ontology header definition (in the case of an OWL ontology) or other statements, as given in Table 33.

Table 33 RDF Documents

Stereotype	Base Class	Parent	Tags	Constraints	Description
Document «document»	Class	«resource»	n/a	n/a	Extends 16.1.1.1 ("Document"). The document class is optional, but is a convenient container for namespaces and namespace definitions, statements, etc.
Namespace «namespace»	Class	n/a	n/a	Must conform to [XMLNS]	Extends 16.1.1.3 ("Namespace")
XMLBase «xmlBase»	Property	n/a	n/a	n/a	XMLBase is a property of Document and links the document to zero or more base namespaces used therein.
NamespaceDefinition «namespaceDefinition»	Class	n/a	n/a	Must conform to [XMLNS]	Extends 16.1.1.4 ("NamespaceDefinition")
NamespacePrefix «namespacePrefix»	Property	n/a	n/a	n/a	Indicates the string declared as the local shorthand notation, or prefix, for use in referring to the namespace, a property of the NamespaceDefinition class
NamespaceDefinitionFor-Namespace «resolvesTo»	Property	n/a	n/a	n/a	Property of the NamespaceDefinition class linking it to the namespace that the definition resolves to
NamespaceDefinitionFor-Document «namespaceDefForDocument»	Property	n/a	n/a	n/a	DefinitionForDocument is a property of Document and links the document to zero or more namespace definitions used therein.
URIReference «uriReference»	Class	n/a	n/a	Must conform to [RDF Syntax]	Extends 16.1.1.7 ("URIReference")

Table 33 RDF Documents

Stereotype	Base Class	Parent	Tags	Constraints	Description
LocalName «localName»	Property	n/a	n/a	n/a	Extends 16.1.1.2 (“LocalName”); localName is a property of the URIReference class, and represents the fragment identifier for the URI reference
NamespaceURIReference «namespaceURIRef»	Property	n/a	n/a	n/a	Indicates the URI reference for a given namespace (property of namespace)
UniformResourceIdentifier «URI»	Class	«uriReference»	n/a	Must conform to [RDF Syntax]	Optional class used primarily for mapping purposes, Extends 16.1.1.6 (“UniformResourceIdentifier”)
URIForURIReference «uriForURIRef»	Property	n/a	n/a	n/a	Links a URI reference to the URI if the optional URI class is supported
RDFStatement «rdfStatement»	Class	n/a	n/a	n/a	Extends 16.1.1.5 (“RDFStatement (Modified Definition)”)
IsReifiedOnly «isReifiedOnly»	Property	n/a	n/a	n/a	If present, indicates that a statement is reified but not asserted (*i.e.*, has no subject, predicate, or object associated with it). Reflects the case where statements are made within a vocabulary or ontology about this statement, but nothing more is known.
StatementForDocument «statementForDocument»	Property	n/a	n/a	Statements are ordered with respect to the document(s) they occur in.	Links a document to the statements it contains.

16.1.6 Classes and Utilities

The stereotypes associated with the definitions given in 11.2 ("The Classes and Utilities Diagrams") of the RDFS metamodel are given in Table 34. Note that instances of RDFSDatatype are *classes* corresponding to the datatypes defined in [XML Schema Datatypes]. OWL restricts this set further, as discussed in 12 ("The OWL Metamodel"), and as defined in the model library given in Appendix A ("Foundation Ontology (M1) for RDFS and OWL").

Table 34 Classes and Utilities

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFSResource «resource»	Class	n/a	n/a	n/a	Extends 11.2.5 ("RDFSResource")
RDFScomment «comment»	Property	n/a	n/a	A comment is a plain literal, contained by the resource it describes.	Indicates a comment on a resource
RDFSisDefinedBy «isDefinedBy»	Constraint	n/a	n/a	rdfs:isDefinedBy is a subproperty of rdfs:seeAlso.	Indicates that the resource isDefinedBy the target resource; Relates the resource to a defining resource in this or another RDFSchema package, or in an external RDF or RDFS document
RDFSlabel «label»	Property	n/a	n/a	A label is a plain literal, contained by the resource it describes.	Indicates a human-readable label for a resource
RDFSmember «member»	Association	n/a	n/a	n/a	Indicates that the resource is a member of the target resource
RDFSseeAlso «seeAlso»	Constraint	n/a	n/a	n/a	Indicates that more information about the resource can be found at the target resource; Relates the resource to another resource in this or another RDFSchema package, or in an external RDF or RDFS document
RDFtype «rdfType»	Association	n/a	n/a	n/a	Indicates that the resource has the type of the target resource
RDFvalue «value»	Property	n/a	n/a	n/a	Is idiomatic, used to represent structured values

Table 34 Classes and Utilities

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFSClass «rdfsClass»	Class	«resource»	n/a	[1] All instances conforming to an RDFS class are instances of the class; [2] The RDFS class must have a URI reference.	Extends 11.2.2 (“RDFS-Class”)
RDFSsubClassOf «rdfsSubClassOf»	Generalization	n/a	n/a	Generally has the semantics of UML Generalization, but classes on both ends of the generalization must be stereotyped «rdfsClass» (or «owlClass», if used with the profile for OWL, and mixing the two is permitted, as long as proper subclassing structure between the two is maintained).	Indicates that the resource is a subclass of the target resource
RDFSDatatype «rdfsDatatype»	Class	«rdfsClass»	n/a	Members of RDFS-Datatype must have URI references.	Extends 11.2.3 (“RDFS-Datatype”)
RDFSLiteral «literal»	Class	«resource»	n/a	n/a	Extends 11.2.4 (“RDFSLiteral”)
PlainLiteral «plainLiteral»	Class	«literal»	n/a	n/a	Extends 11.2.1 (“PlainLiteral”)
TypedLiteral «typedLiteral»	Class	«literal»	n/a	The datatype property is required.	Extends 11.2.7 (“TypedLiteral”)
DatatypeFor-TypedLiteral «datatype»	Property	n/a	n/a	n/a	Links a typed literal to its type (must be an RDFS-Datatype)
RDFXMLLiteral «xmlLiteral»	Class	«typedLiteral»	n/a	n/a	Extends 11.2.6 (“RDFXML-Literal”)

These stereotypes may be used with or without the stereotypes defined in 16.1.5 (“RDF Document (optional)”), though the RDF document model is recommended for applications where namespace interoperability is desired. If the RDF graph model specified in 16.1.10 (“RDF Graphs and Nodes (optional)”) is adopted, however, the stereotypes specified in 16.1.5 (“RDF Document (optional)”) must also be used.

16.1.7 Properties in RDF/S

16.1.7.1 RDFProperty

Description

The RDFProperty profile construct extends the definition given in 11.3.1 ("RDFProperty"). The association class represents a binary relation with unidirectional navigation, from the class that defines the domain of the property to the class that defines its range, with association end names "`domain`" and "`range`", respectively.

Properties in RDF, RDF Schema, and OWL are defined globally, that is, that they apply to all ontologies in the universe of discourse on which they are defined – not only to the ontology that they are defined in, but to those that are imported or that import the ontology that defines them. For RDF properties that are defined without a specifying a domain or range (which is legal in RDF, unlike UML), the profile uses a global `Thing` class (`Thing` for RDF/S, `owl:Thing` in OWL ontologies) as default for the "missing" association end. Definitions for both are provided in the model library given in Appendix A ("Foundation Ontology (M1) for RDFS and OWL"). Properties that are defined with such a default domain or range may not have multiplicities (other than [0..*]) or other constraints, such as any OWL cardinality or value restrictions, defined on them.

Stereotype and Base Class

«rdfProperty» with base class of UML::AssociationClass

Parent

«resource»

Tags

None.

Constraints

[1] All instances are instances of `Thing`, at least indirectly.

[2] Properties on `Thing` may not have multiplicity (other than [0..*]) or other OWL restrictions.

[3] The RDF property must have a URI reference.

[4] Association classes with `rdfProperty` applied are binary, and have unidirectional navigation.

[5] Properties cannot have the same value twice (*i.e.*, in UML, isUnique=true).

[6] Property values are not ordered (*i.e.*, in UML, isOrdered=false).

[7] The navigable end of an `rdfProperty` association is an `rdfGlobal` property.

Notation

A. Properties without a specified domain are considered to be defined on the UML class `Thing`, (or `owl:Thing` in the case of an OWL ontology), for example, as shown in Figure 44 . `Thing` (for use in RDF/S vocabularies) and `owl:Thing` are defined in the model library provided in Appendix A, Foundation Ontology (M1) for RDFS and OWL.

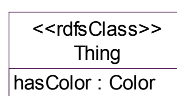


Figure 44 Property `hasColor` Without Specified Domain

From a UML perspective, RDF properties are semantically equivalent to binary associations with unidirectional navigation ("one-way" associations).

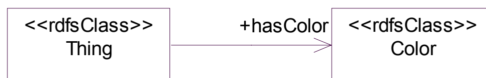


Figure 45 Property `hasColor` Without Specified Domain, Alternate Notation

Figure 45 shows that there is efficient navigation from an instance of `Thing` to an instance of `Color` through the `hasColor` end, just like a UML property. The only difference is the underlying repository will have an association with the `hasColor` property as one of its ends. The other end will be owned by the association itself, and be marked as nonnavigable.

Unidirectional associations can be classes, as shown in Figure 46 :

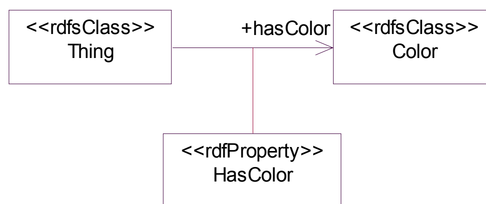


Figure 46 Property `hasColor` - Association Class Representation

An association class can have properties, associations, and participate in generalization as any other class. Notice that the association has a (slightly) different name than the property, by capitalizing the first letter, to distinguish the association class (of links, tuples) from the mapping (across those links, tuples). A stereotype "`rdfProperty`" may be introduced to highlight such binary, unidirectional association classes, as shown in Figure 46. In the examples given in the remainder of the profile, the notation showing properties in class rectangles is sometimes used, but unidirectional associations and association classes could be used instead (with the exception of the approach taken for `owl:inverseOf`, see section 16.2.6.2 ("owl:inverseOf Relation")).

B. Properties *with* a domain are defined on a UML class for the domain, where the property is not inherited from a supertype.

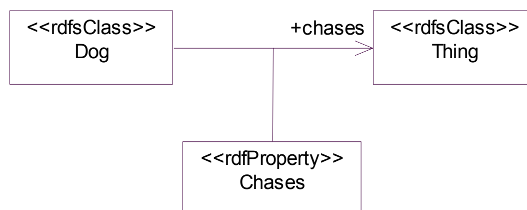


Figure 47 Properties With Defined Domain, Undefined Range

Normally UML models introduce properties and restrict them with multiplicities in the same class. This translates in RDF/OWL as global properties (along with restrictions in specific classes in OWL). If multiplicities or other restrictions are used on a property of `Thing`, the translation to OWL will be a global property, and a class called "`Thing`" with the restrictions. A stereotype "`rdfGlobal`" may be introduced to highlight properties that are introduced at that class. Properties that are inherited are distinguished in UML by subsetting or redefinition, as discussed below.

C. Properties *with* a defined range have the range class as their type in UML. Properties with no range have `Thing` as their type in UML, as shown in Figure 47. Property types are shown to the right of the colon after the property name, as shown in Figure 44.

D. Properties with a range have the range class as their type in UML, as shown in Figure 44.

E. The most natural representation for RDF/S and OWL property subtyping (*i.e.*, rdfs:subPropertyOf) in UML is to use UML property/unidirectional association subsetting or association class subtyping. The UML semantics for both is that all links (instances, tuples) of the subtype properties or associations are links (instances, tuples) of all the supertypes properties or associations.

The most natural notation for property subsetting in UML is to use "{subsets <super-property-name>}" at the end of the property entry in a class, as shown in Figure 48.

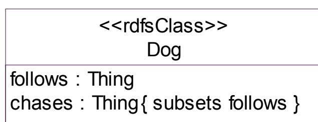


Figure 48 Property Subsetting, Notation on Property Entry for Class

Alternatively, the notation given in Figure 49 may be used for unidirectional association subsetting.

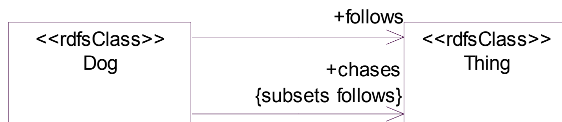


Figure 49 Property Subsetting, Unidirectional Association Notation

Finally, for use with association classes, the notation shown in Figure 50, which uses a UML Generalization with the stereotype «rdfsSubPropertyOf», as described in Table 35, is preferred.

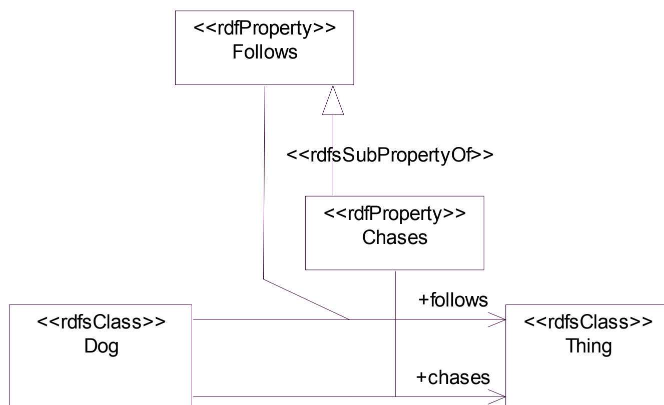


Figure 50 Property Subsetting, Association Class Notation

Additional stereotypes associated with the definitions given in 11.3 (“The Properties Diagram”) of the RDFS metamodel are given in Table 35.

Table 35 Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFGlobalProperty «rdfGlobal»	Property	n/a	n/a	n/a	An optional property on the RDFProperty association class indicating that a property is defined globally, *i.e.* that its domain and/or range is owl:Thing
RDFSsubPropertyOf «rdfsSubPropertyOf»	Generalization	n/a	n/a	Properties on both ends of the generalization must be stereotyped «rdfProperty» (or «owlObjectProperty» or «owlDatatypeProperty», if used with the profile for OWL, and limited mixing of RDF and OWL is permitted).	Indicates that the resource is a sub-property of the specified resource

16.1.8 Containers and Collections

The stereotypes associated with the definitions given in 11.4 (“The Containers Diagram”) and 11.5 (“The Collections Diagram”) of the RDFS metamodel are given in Table 36.

Table 36 Containers and Collections

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFSContainer «container»	Class	«resource»	n/a	n/a	Extends 11.4.3 (“RDFSContainer”)
RDFSContainerMembershipProperty «containerMembershipProperty»	Association Class	«rdfProperty»	n/a	n/a	Extends 11.4.4 (“RDFSContainerMembershipProperty”)
RDFAlt «alt»	Class	«container»	n/a	n/a	Extends 11.4.1 (“RDFAlt”)
RDFBag «bag»	Class	«container»	n/a	n/a	Extends 11.4.2 (“RDFBag”)
RDFSeq «seq»	Class	«container»	n/a	n/a	Extends 11.4.5 (“RDFSeq”)

Table 36 Containers and Collections

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFList «list»	Class	«resource»	n/a	n/a	Extends 11.5.1 (“RDFList”)
RDFfirst «first»	Property	n/a	n/a	n/a	Indicates that the target resource is the first member of the list
RDFrest «rest»	Association	n/a	n/a	n/a	Indicates that the target resource is the rest of the list

Note: need discussion of container membership properties...rdf:_1, rdf:_2, rdf:_3,... and their inclusion in the model library. Same for rdf:nil.

16.1.9 Reification

The stereotypes associated with the definitions given in 11.6 (“The Reification Diagram”) of the RDFS metamodel are given in Table 37.

Table 37 Reification (Basic Model)

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFStatement «statement»	Class	«resource»	n/a	n/a	Extends 11.6.1 (“RDFStatement”)
RDFsubject «subject»	Property	n/a	n/a	The target resource must be a URI or blank node.	Indicates that the target resource is the subject of the triple
RDFpredicate «predicate»	Property	n/a	n/a	The target resource must have a URI.	Indicates that the target resource is the predicate of the triple
RDFobject «object»	Property	n/a	n/a	The target resource must be a URI, blank node, or literal.	Indicates that the target resource is the object of the triple

16.1.10 RDF Graphs and Nodes (optional)

The stereotypes associated with the RDF Graph model given in 16.1.2 (“RDF Graph Model (Optional)”) are defined in Table 38. These definitions supersede those given in Table 37 when the RDF graph model is adopted., augment the definitions given in Table 35, and presume that the RDF Document stereotypes, defined in 16.1.5 (“RDF Document (optional)”), have also been adopted.

Table 38 RDF Graphs and Nodes

Stereotype	Base Class	Parent	Tags	Constraints	Description
BlankNode «blankNode»	Class	«resource»	n/a	n/a	Extends 16.1.2.1 (“BlankNode”)
NodeID «nodeID»	Property	n/a	n/a	n/a	Indicates an optional node identifier for the blank node
URIReferenceNode «uriReferenceNode»	Class	«resource»	n/a	n/a	Extends 16.1.2.6 (“URIReferenceNode”)
RDFProperty «rdfProperty»	Association Class	«resource»	n/a	See Table 35	Extends 16.1.2.3 (“RDFProperty (Modified Definition)”)
RDFSResource «resource»	Class	n/a	n/a	URIReferenceNode, BlankNode, and Literal are pairwise disjoint and covering.	Extends 16.1.2.5 (“RDFSResource (Modified Definition)”)
RDFStatement «statement»	Class	n/a	n/a	n/a	Extends 16.1.1.5 (“RDFStatement (Modified Definition)”)
RDFsubject «subject»	Property	n/a	n/a	The target node cannot be a literal.	Indicates that the target node is the subject of the triple
RDFpredicate «predicate»	Property	n/a	n/a	n/a	Indicates that the target property is the predicate of the triple
RDFobject «object»	Property	n/a	n/a	n/a	Indicates that the target node is the object of the triple
Graph	Class	n/a	n/a	n/a	Extends 16.1.2.2 (“Graph”)

16.2 UML Profile for OWL

This section specifies the UML profile for OWL. It is loosely organized based on the structure of the OWL metamodel, with sections reordered to facilitate understanding and utility.

16.2.1 OWL Profile Package

The following sections specify the set of stereotypes and tagged values that comprise the UML2 profile for using UML notation to represent OWL ontologies, and extends the OWL metamodel. Support for the optional RDF document and graph model and profile extensions, given in section 16.1 ("UML Profile for RDF Schema"), is also discussed.

As shown in Figure 51, the OWL profile package provides the container for the profile and extends the RDFSchema profile package.

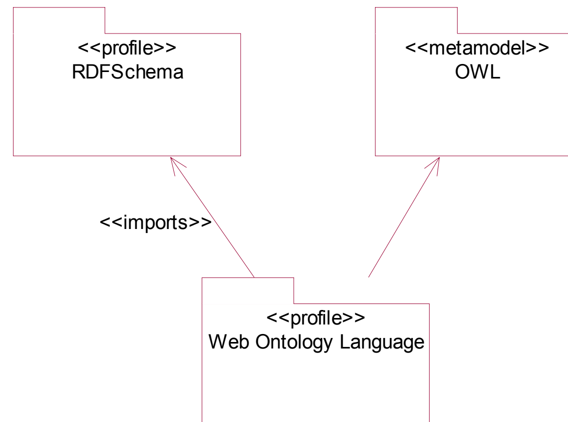


Figure 51 Web Ontology Language (OWL) Profile Package

Applying the Web Ontology Language stereotype to a package requires that the UML constructs contained within the package be interpreted according to this profile definition.

Constraints

[1] All classes in a Web Ontology Language package must be stereotyped by «rdfsClass» or by «owlClass».

[2] For applications intending to support OWL DL, all classes in a Web Ontology Language package must be stereotyped by «owlClass».

16.2.2 OWL Ontology Document

An OWL document consists of optional ontology headers (generally at most one) plus any number of class axioms, property axioms, and facts about individuals. Please note that "axiom" is the formal term used in [OWL S&AS].

As with most RDF documents, the OWL statements that comprise an OWL document should be subelements of a rdf:RDF element. This enclosing element generally holds XML namespace and base declarations. Also, an OWL ontology document often starts with several entity declarations. For these reasons, we recommend compliance with / implementation of the optional RDF document extensions defined in 16.1.1 ("RDF Document Syntax (Optional)") and 16.1.5 ("RDF Document (optional)").

16.2.3 OWL Annotation Properties

OWL annotation properties correspond, for the most part, to properties on other stereotypes defined herein. OWL provides several built-in annotation properties, which do have special meaning and implementation, as defined in Table 39 and 16.2.3.1 (“owl:versionInfo”), and also allows users to define such properties, as needed.

Table 39 Annotation Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
OWLAnnotationProperty «annotationProperty»	Association Class	«rdfProperty»	n/a	An instance of an annotation property is a single valued UML property, which is an instance of this class.	Extends 12.6.1 (“OWLAnnotationProperty”) to support user defined annotation properties. This definition represents the class of annotation properties rather than instances of annotation properties.
AnnotationPropertyInstance «annotation»	Property	n/a	n/a	[1] The range of an annotation property is either a URI reference, an RDFLiteral, or an Individual. [2] Annotations are not inherited by subclasses, subproperties, or individuals of the classes or properties on which they are defined.	User-defined annotations can be applied to any ontology element: ontologies themselves, classes, properties, or individuals.
RDFScomment «comment»	Property	n/a	n/a	[1] A comment is a plain literal, contained by the resource it describes. [2] Comments are not inherited by subclasses, subproperties, or individuals of the classes or properties on which they are defined.	Indicates a comment on a resource; rdfs:comment is an instance of owl:AnnotationProperty

Table 39 Annotation Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
RDFSisDefinedBy «isDefinedBy»	Constraint	n/a	n/a	rdfs:isDefinedBy is a subproperty of rdfs:seeAlso.	Indicates that the resource isDefinedBy the target resource; rdfs:isDefinedBy is an instance of owl:AnnotationProperty; Relates the resource to a defining resource in this or another RDFSchema package, or in an external RDF or RDFS document
RDFSlabel «label»	Property	n/a	n/a	[1] A label is a plain literal, contained by the resource it describes. [2] Labels are not inherited by subclasses, subproperties, or individuals of the classes or properties on which they are defined.	Indicates a human-readable label for a resource; rdfs:label is an instance of owl:AnnotationProperty
RDFSseeAlso «seeAlso»	Constraint	n/a	n/a	n/a	Indicates that more information about the resource can be found at the target resource; rdfs:seeAlso is an instance of owl:AnnotationProperty; Relates the resource to another resource in this or another RDFSchema package, or in an external RDF or RDFS document

Note: need discussion of annotations, comments, and labels as static properties in UML.

16.2.3.1 owl:versionInfo

Description

An `owl:versionInfo` statement generally has as its object a string giving information about this version, for example RCS/CVS keywords. This statement does not contribute to the logical meaning of the ontology other than that given by the RDF(S) model theory.

Although this property is typically used to make statements about ontologies, it may be applied to any OWL construct. For example, one could attach a `owl:versionInfo` statement to an OWL class.

Stereotype and Base Class

No stereotype; implemented as a UML Property of the stereotype it describes.

Parent

None.

Tags

None.

Constraints

[1] `owl:versionInfo` is an instance of `owl:AnnotationProperty` in the OWL language

Notation

In the case of an ontology, with a stereotyped package of «ontology» or «rdfDocument», the normal stereotype notation be used, with property values specified in braces under the stereotype label[7], as shown in Figure 52 ("Stereotype Notation for owl:versionInfo Applied to an Ontology or RDF Document"). In cases where the UML tools do not support stereotype property notation, a corresponding UML class, with the same stereotype, may be used, and annotation properties for the ontology or RDF document would be applied to that class. For OWL classes and properties, `owl:versionInfo` should be represented as a property on «owlClass» or «owlProperty» stereotype, as appropriate.

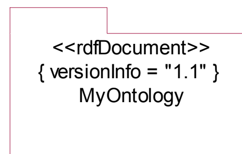


Figure 52 Stereotype Notation for owl:versionInfo Applied to an Ontology or RDF Document

16.2.4 OWL Ontology Properties

While an OWL ontology may seem to be analogous to the notion of an RDFS vocabulary on the surface, in fact, the `owl:ontology` *syntactic feature* of OWL essentially enables association of a set of annotations with an RDF vocabulary or OWL ontology. The target element for (or domain of) these annotations, called *ontology properties* (and *annotation properties*, for annotations on ontologies -- annotation properties are not limited to annotations on ontologies, however) in OWL, is the "ontology ID" (*i.e.*, the URI reference that is the identifier for the ontology, which it has by virtue of being a resource), not a subordinate OWL document.

7. The stereotype property notation follows the clarifications and elaborations of stereotype notation being proposed by the UML extension for Systems Engineering. See Systems Modelling Language (SysML) Specification, Addendum to SysML v0.9, Profiles and Model Libraries Chapter, http://doc.omg.org/ad/05-06-01.pdf.

OWL provides several built-in ontology properties, and also allows users to define such properties, as needed. Although there is not a great analogy in UML to cover all cases, ontology properties, particularly the set of built-in ontology properties, correspond to UML constraints between packages, and should be implemented as given in Table 40 (“Ontology Properties”) and the subsections that follow.

Table 40 Ontology Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
OWLOntologyProperty «ontologyProperty»	Association Class	«rdfProperty»	n/a	[1] Applies only between packages stereotyped by «ontology» or «rdfDocument»; [2] ontology properties must have a defined domain and range, which is an *ontology ID*, *i.e.*, the URI reference for the ontology or RDF vocabulary; [3] ontology properties are single valued.	Extends 12.7.2 (“OWLOntologyPropert y”) to allow for user defined ontology properties; In general, ontology properties are constraints between ontologies.
OntologyPropertyInsta nce «ontologyAnnotation»	Property	n/a	n/a	n/a	Indicates some kind of relationship between two ontologies, such as `owl:PriorVersion`, but in this case, is user defined.

User-defined ontology properties are essentially properties defined on the «ontology» or «rdfDocument» stereotypes, that can apply only between packages having these stereotypes. They should be defined individuals of the stereotyped OWLOntologyProperty class, in order to distinguish them from annotation properties or other RDF or OWL properties, however, in order to support downstream DL reasoning requirements.

16.2.4.1 owl:backwardCompatibleWith

Description

An `owl:backwardCompatibleWith` statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology, and further indicates that it is backward compatible with it.

Stereotype and Base Class

«backwardCompatibleWith» stereotype of *UML::Constraint*

Parent

None. It is an instance of `owl:OntologyProperty` in the OWL language.

Tags

None.

Constraints

[1] Applies only between packages stereotyped by «ontology» or «rdfDocument».
[2] Classes and properties in the new version that have the same name as classes and properties in the earlier version must either be equivalent to or extend those in the earlier versions.
[3] The later version must be logically consistent with the earlier version.
[4] (semantic constraint) Identifiers in the later version have the same interpretation in the earlier version.

Notation

Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 53 (“Stereotype Notation for owl:backwardCompatibleWith”).

Figure 53 Stereotype Notation for owl:backwardCompatibleWith

16.2.4.2 owl:imports

Description

An `owl:imports` statement references another OWL ontology containing definitions, whose meaning is considered to be part of the meaning of the importing ontology. Each reference consists of a URI specifying from where the ontology is to be imported. Syntactically, `owl:imports` is a property with the class `owl:Ontology` as its domain and range.

Stereotype and Base Class

«owlImports» stereotype of *UML::PackageImports*

Parent

None. It is an instance of `owl:OntologyProperty` in the OWL language.

Tags

None.

Constraints

[1] Applies only between packages stereotyped by «ontology» or «rdfDocument».

Notation

Dashed line between two instances with stereotype label, arrowhead towards the imported ontology, as shown in Figure 54 (“Stereotype Notation for owl:imports”).

Figure 54 Stereotype Notation for owl:imports

16.2.4.3 owl:incompatibleWith

Description

An `owl:incompatibleWith` statement contains a reference to another ontology. This indicates that the containing ontology is a later version of the referenced ontology, but is not backward compatible with it. Essentially, this is for use by ontology authors who want to be explicit that documents cannot upgrade to use the new version without checking whether changes are required.

Stereotype and Base Class

«incompatibleWith» stereotype of *UML::Constraint*

Parent

None. It is an instance of owl:OntologyProperty in the OWL language.

Tags

None.

Constraints

[1] Applies only between packages stereotyped by «ontology» or «rdfDocument».
[2] The later version must be logically inconsistent with the earlier version.

Notation

Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 55 (“Stereotype Notation for owl:incompatibleWith”).

Figure 55 Stereotype Notation for owl:incompatibleWith

Note: While it might seem reasonable to eliminate the arrowhead in this case, and make the relationship bi-directional, all RDF graphs and thus such relationships are unidirectional in RDF, RDF Schema and OWL. Applications that leverage this notation may optionally allow the user to indicate that they want a particular instance of `owl:incompatibleWith` to be bidirectional, and in this case, as a shorthand notation, eliminate the arrowhead and use a single dashed line; the interpretation of such notation should be two instances of `owl:incompatibleWith`, however.

16.2.4.4 owl:priorVersion

Description

An `owl:priorVersion` statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology. This has no meaning in the model-theoretic semantics other than that given by the RDF(S) model theory. However, it may be used by software to organize ontologies by versions.

Because of the lack of semantics, there is no obvious UML element to reuse or stereotype for this particular OWL property. However, assuming that the spirit of this property is similar to though not quite as strong as that of `owl:backwardCompatibleWith`, we suggest the following.

Stereotype and Base Class

«priorVersion» stereotype of *UML::Constraint*

Parent

None. It is an instance of `owl:OntologyProperty` in the OWL language.

Tags

None.

Constraints

[1] Applies only between packages stereotyped by «ontology» or «rdfDocument».

Notation

Dashed line between two instances with stereotype label, arrowhead towards the earlier version, as shown in Figure 56 (“Stereotype Notation for owl:priorVersion”).

Figure 56 Stereotype Notation for owl:priorVersion

16.2.5 OWL Classes, Restrictions, and Class Axioms

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Like RDF classes, every OWL class is associated with a set of individuals, called the class extension. The individuals in the class extension are called the instances of the class. A class has an intensional meaning (the underlying concept) which is related but not equal to its class extension. Thus, two classes may have the same class extension, but still be different classes.

A class description is the term used in [OWL S&AS] for the basic building blocks of class axioms. A class description describes an OWL class, either by a class name or by specifying the class extension of an unnamed anonymous class.

As described in 12.2.9 (“OWLClass”), OWL distinguishes six types of class descriptions:

1. a class identifier (a URI reference)

2. an exhaustive enumeration of individuals that together form the instances of a class
3. a property restriction
4. the intersection of two or more class descriptions
5. the union of two or more class descriptions
6. the complement of a class description

The first type is special in the sense that it describes a class through a class name (syntactically represented as a URI reference). The other five types of class descriptions describe an anonymous class by placing constraints on the class extension. They consist of a set of RDF triples in which a blank node represents the class being described. This blank node has an `rdf:type` property whose value is `owl:Class`.

Class descriptions of type 2-6 describe, respectively, a class that contains exactly the enumerated individuals (2nd type), a class of all individuals which satisfy a particular property restriction (3rd type), or a class that satisfies boolean combinations of class descriptions (4th, 5th and 6th type). Intersection, union and complement can be respectively seen as the logical AND, OR and NOT operators. The four latter types of class descriptions lead to nested class descriptions and can thus in theory lead to arbitrarily complex class descriptions. In practice, the level of nesting is usually limited. Stereotypes for OWL class descriptions are given in Table 41and the subsections that follow.

> *Note*: `owl:Class` is defined as a subclass of `rdfs:Class`. The rationale for having a separate OWL class construct lies in the restrictions on OWL DL (and thus also on OWL Lite), which imply that not all RDFS classes are legal OWL DL classes. In OWL Full these restrictions do not exist and therefore `owl:Class` and `rdfs:Class` are equivalent in OWL Full.

The set of stereotypes defined for use in constructing class descriptions are given in Table 41 and the subsections that follow.

Table 41 Class Descriptions

Stereotype	Base Class	Parent	Tags	Constraints	Description
OWLClass «owlClass»	Class	«rdfsClass»	isDeprecated: Boolean	All instances conforming to an OWL class are instances of the class.	A type 1 class description is syntactically represented as an named instance of owl:Class, a subclass of rdfs:Class. Extends 12.2.9 ("OWLClass").

Table 41 Class Descriptions

Stereotype	Base Class	Parent	Tags	Constraints	Description
EnumeratedClass «enumeratedClass»	Class	«owlClass»	n/a	n/a	Extends 12.2.4 ("EnumeratedClass"). The enumerated individuals are the UML instance specifications of the class.
EnumeratedDataValues «dataRange»	Enumeration	n/a	n/a	n/a	Represents enumerated datatype values. The enumerated values are the enumeration literals (a kind of instance specification) of the enumeration.
RestrictionClass «owlRestriction»	Class	«owlClass»	n/a	n/a	Extends 12.2.10 ("OWLRestriction"). This construct reifies a restriction class for use in complex class axioms. The restriction class is a subtype of the domain of the property (which might be `owl:Thing`), and a supertype of the class that it is restricting the property on. Note: although restriction classes are typically anonymous, they are not required to be and can be named (via a class ID URI reference/name).

A property restriction is a special kind of class description. When used in another class, the restriction class is effectively a supertype of the containing class, applying the restriction to all individuals of the containing class. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints. Property restrictions can be applied both to datatype properties (properties for which the value is a data literal) and object properties (properties for which the value is an individual).

16.2.5.1 Cardinality Constraints

Description

In OWL, like in RDF, it is assumed that any instance of a class may have an arbitrary number (zero or more) of values for a particular property. To make a property required (at least one), to allow only a specific number of values for that property, or to insist that a property must not occur, cardinality constraints can be used. OWL provides three

constructs for restricting the cardinality of properties locally within a class context: owl:maxCardinality, owl:minCardinality, and owl:Cardinality. These constructs are analogous to multiplicity in UML, thus the approach taken is

- for properties whose initial definition includes the cardinality constraint, simply apply multiplicities as appropriate.
- for inherited properties, redefine the property with new multiplicity.

Stereotype and Base Class

None. UML multiplicities are presented using the standard presentation options defined in section 7.4.1, "Unified Modeling Language: Superstructure", version 2 [UML2].

Parent

None.

Tags

None.

Constraints

[1] ValueSpecifications for multiplicities in OWL must be non-negative integer literals.
[2] isOrdered = false in OWL.
[3] isUnique = true in OWL, meaning that the values are a set, not a a bag.

Notation

For inherited properties, show the property with restricted multiplicity in subtype, and using "{redefines <restricted-property-name> }" at the end of the property entry in a class (can be elided), as shown in Figure 57.

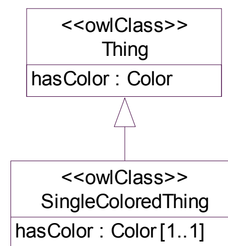


Figure 57 owl:Cardinality - Restricted Multiplicity in Subtype

Alternatively, when unidirectional associations are desirable, cardinality constraints can be represented as shown in Figure 58.

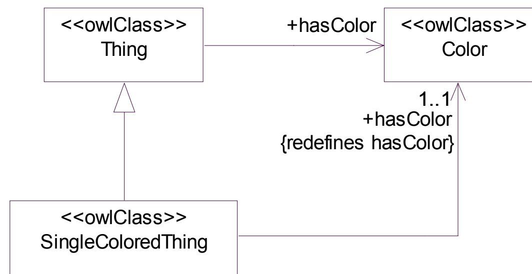


Figure 58 owl:Cardinality - Restricted Multiplicity in Subtype

16.2.5.2 owl:allValuesFrom Constraint

Description

The value constraint `owl:allValuesFrom` is a built-in OWL property that links a restriction class to either a class description or a data range. A restriction containing `owl:allValuesFrom` specifies a class or data range for which all values of the property under consideration are either members of the described class, or are data values within the specified data range.

Essentially, `owl:allValuesFrom` is used to *redefine* the type of a particular property (in effect define a subproperty), similar to UML property redefinition.

Stereotype and Base Class

None. Uses UML Generalization and property redefinition, as shown under Notation, below.

Note that the domain and/or target (for `owl:allValuesFrom`) for the subproperty will not always be a direct descendent of the superclass that the property is defined on, as it happens to be in the examples.

If the attribute form of notation is used, then "{redefines <parent-class>::<property-name>}" should be given at the end (*i.e.*, to the right) of the property entry. The parent class is optional if the property inherits from only one parent.

Parent

None.

Tags

None.

Constraints

[1] Property name is not changed in redefinition.

Notation

Several notation approaches are provided here, in keeping with the notation used for properties in the profile for RDF/S. First, we can show the property with restricted type in subtype, by adding "{redefines <restricted-property-name> }" at the end of the property entry (can be elided), as in Figure 59.

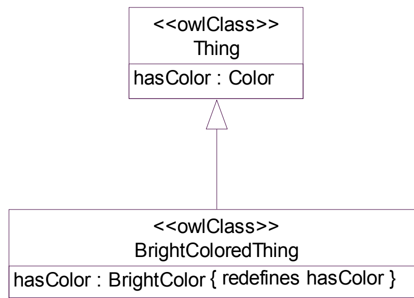


Figure 59 Simple Property Redefinition Example For owl:allValuesFrom

Secondly, we can show the same thing using unidirectional association style properties, as shown in Figure 60.

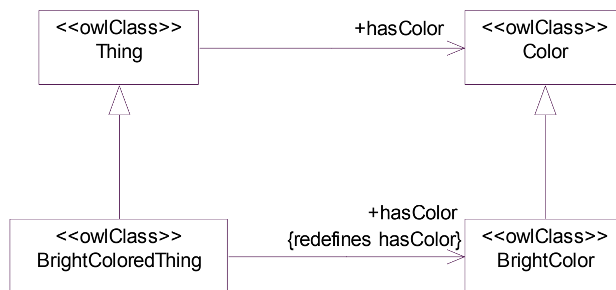


Figure 60 Property Redefinition For owl:allValuesFrom With Unidirectional Associations

An alternative using association classes is shown below. This might be confusing to OWL users, because the subproperty statement cannot place restrictions on the `HasBrightColor` subproperty. The UML generalization used this way would need to be interpreted differently in the mapping to OWL, perhaps by applying an "`owlRedefines`" stereotype to it.

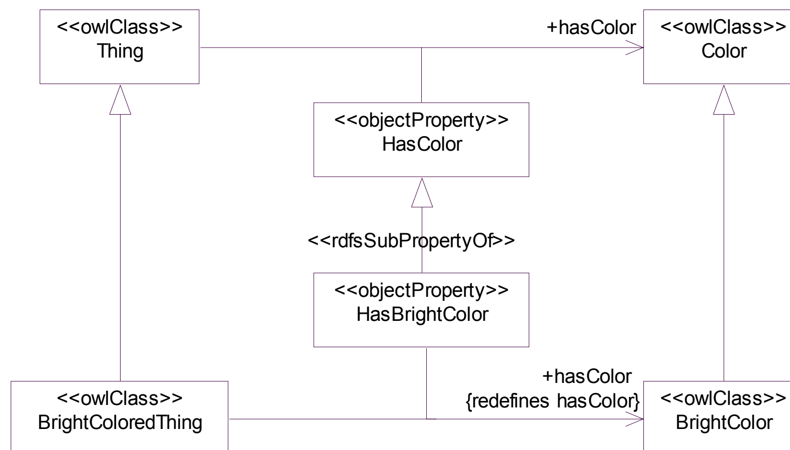


Figure 61 Property Redefinition For owl:allValuesFrom With Association Classes

16.2.5.3 owl:someValuesFrom and owl:hasValue

Description

Similar to `owl:allValuesFrom`, `owl:someValuesFrom` is a built-in OWL property that links a restriction class to either a class description or a data range. A restriction containing an `owl:someValuesFrom` constraint is used to describe a class or data range for which at least one value of the property concerned is either a member of the class extension of the class description or a data value within the specified data range. In other words, it defines a class of individuals x for which there is at least one y (either an instance of the class description or value of the data range) such that the pair (x,y) is an instance of P. This does not exclude that there are other instances (x,y') of P for which y' does not belong to the class description or data range.

The value constraint `owl:hasValue` is a built-in OWL property that links a restriction class to a value V, which can be either an individual or a data value. A restriction containing an `owl:hasValue` constraint describes a class of all individuals for which the property concerned has at least one value semantically equal to V (it may have other values as well).

Again, like `owl:allValuesFrom`, `owl:someValuesFrom` and `owl:hasValue` are used to redefine the type of a particular property, similar to UML property redefinition.

Stereotype and Base Class

A stereotype «owlValue» of *UML::Constraint* is applied to redefining properties. The stereotype has these properties.

- `hasValue` – of type *UML::InstanceSpecification*
- `someValuesFrom` – of type *UML::Class*

Parent

None.

Tags

None.

Constraints

[1] Can be applied to RDF and OWL properties, but only to properties that redefine other properties.

[2] In the case of owl:someValuesFrom class, the class must be stereotyped «owlClass».

Notation

Put before the property name: "«owlValue» {hasValue = <instance-name>, <instance-name>; someValuesFrom = <class-name>, <class-name>}", for example, as shown in Figure 62, where volume is an individual of type physical dimension[8].

---

8. This notation follows the clarifications and elaboration of stereotype notation being proposed by the UML extension for Systems Engineering. The properties of the stereotypes must be strings, because they cannot be typed by UML metamodel elements. Requires namespace notation to resolve name conflicts. See Systems Modelling Language (SysML) Specification, Addendum to SysML v0.9, Profiles and Model Libraries Chapter, http://doc.omg.org/ad/05-06-01.pdf.

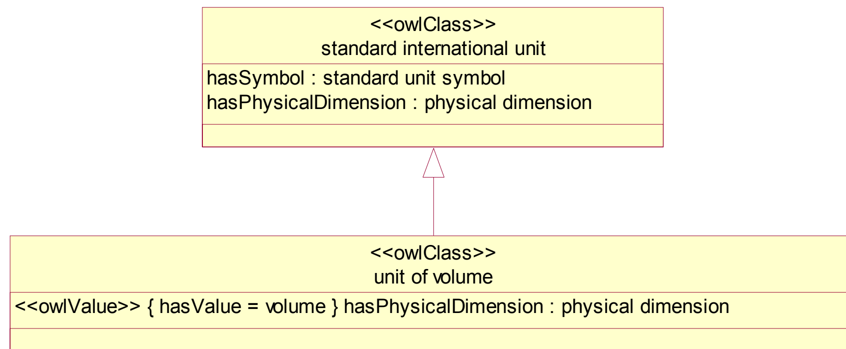


Figure 62 Example Using owl:hasValue Constraint

16.2.5.4 owl:intersectionOf Property

Description

The `owl:intersectionOf` property links a class to a list of class descriptions. An `owl:intersectionOf` statement describes a class for which the class extension contains precisely those individuals that are members of the class extension of all class descriptions in the list. `owl:intersectionOf` can be viewed as being analogous to logical conjunction.

Stereotype and Base Class

A stereotype «intersectionOf» of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies to generalizations with a common subtype.

[2] All instances of super types along intersection generalizations are instances of the subtype.

[3] (UML generalization semantics) All instances of the subtype are instances of the super types.

Notation

Dashed line between generalization lines with stereotype label, as shown in Figure 63.

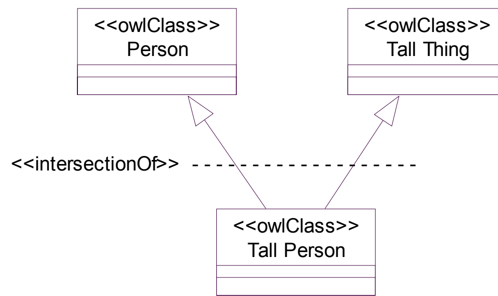


Figure 63 Example Using owl:intersectionOf

The stereotype is based on UML::Generalization rather than UML::Class, so there can be other supertypes not required by the intersection. Use of UML::GeneralizationSet was prohibited in this case, because it requires one supertype – its semantics refers to the instances of the subtypes, not the supertypes.

16.2.5.5 owl:unionOf Property

Description

The `owl:unionOf` property links a class to a list of class descriptions. An `owl:unionOf` statement describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list. `owl:unionOf` is analogous to logical disjunction.

Stereotype and Base Class

UML::GeneralizationSet with isCovering = true, as shown in Figure 64. For consistency with the other class descriptions, vendors can also optionally define a «unionOf» stereotype of *UML::Constraint*, applied to *UML::Generalization* (similar to intersection, above).

Parent

None.

Tags

None.

Constraints

[1] (UML semantics) All instances of the supertype are instances of at least one of the subtypes.

Notation

Dashed line between generalization lines labeled with “{complete}”.

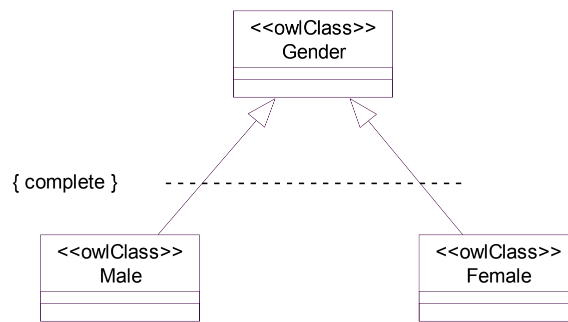


Figure 64 Example Using owl:unionOf

16.2.5.6 owl:complementOf Property

Description

An `owl:complementOf` property links a class to precisely one class description. An `owl:complementOf` statement describes a class for which the class extension contains exactly those individuals that do not belong to the class extension of the class description that is the object of the statement. `owl:complementOf` is analogous to logical negation: the class extension consists of those individuals that are NOT members of the class extension of the complement class.

Stereotype and Base Class

«complementOf» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies between exactly two classes.

[2] All instances (of owl:Thing) are instances of exactly one of the two classes.

Notation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will have owl:complementOf in XML syntax (since all RDF, RDF Schema, and OWL graphs are unidirectional, by definition). Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production in this case should result in two instances of `owl:complementOf` – one for each "side" of the bidirectional constraint.

Figure 65 Example Using owl:complementOf

16.2.5.7 owl:disjointWith Class Axiom

Description

`owl:disjointWith` is a built-in OWL property with a class description as domain and range. Each `owl:disjointWith` statement asserts that the class extensions of the two class descriptions involved have no individuals in common. A class axiom may also contain (multiple) `owl:disjointWith` statements. Like axioms with `rdfs:subClassOf`, declaring two classes to be disjoint is a partial definition: it imposes a necessary but not sufficient condition on the class.

Stereotype and Base Class

«disjointWith» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies only between classes.

[2] No shared instances.

Notation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will have `owl:disjointWith` in XML syntax (since all RDF, RDF Schema, and OWL graphs are unidirectional, by definition). Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production in this case should result in two instances of `owl:disjointWith` – one for each "side" of the bidirectional constraint.

Figure 66 Example Using owl:disjointWith

In cases where there are multiple participants in the same `owl:disjointWith` class axiom, a constraint note with stereotype label and dashed lines to more than one class should be used, as shown in Figure 67.

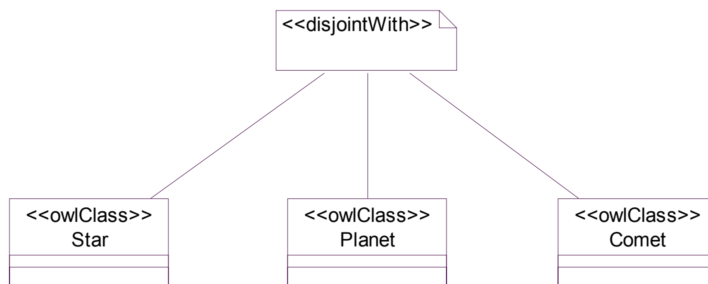


Figure 67 Example Using owl:disjointWith With Multiple Participants

Alternatively, if the classes have a common supertype, use UML::GeneralizationSet with isDisjoint = true. Notation is dashed line between generalization lines labeled with "{disjoint}".

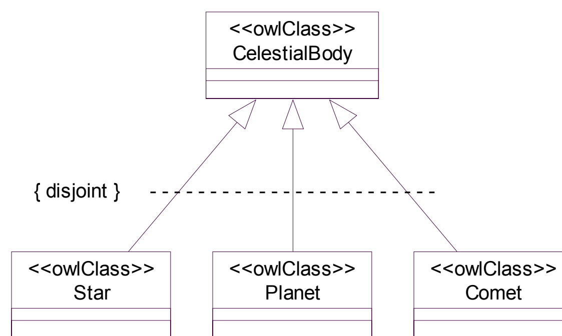


Figure 68 Example Using owl:disjointWith With Common Supertype

16.2.5.8 owl:equivalentClass Class Axiom

Description

`owl:equivalentClass` is a built-in property that links a class description to another class description. The meaning of such a class axiom is that the two class descriptions involved have the same class extension (*i.e.*, both class extensions contain exactly the same set of individuals). A class axiom may contain (multiple) `owl:equivalentClass` statements.

Stereotype and Base Class

«equivalentClass» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies only between classes.

[2] Instances are the same.

Notation

Dashed line between two classes with stereotype label. An arrowhead should be used opposite from the class that will have owl:equivalentClass in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production in this case should result in two instances of `owl:equivalentClass` – one for each “side” of the bidirectional constraint. .

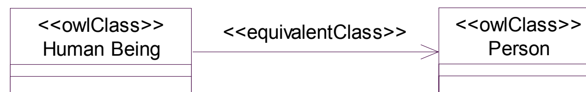


Figure 69 Example Using owl:equivalentClass

Alternatively two UML::Generalizations may be used, again within a given ontology, if such circular definitions are supported by the tool (*i.e.*, class a generalizes class b and vice versa).

In cases where there are multiple participants in the same `owl:equivalentClass` class axiom, a constraint note with stereotype label and dashed lines to more than one class should be used, similarly to the example used for `owl:disjointWith`.

16.2.6 Properties

OWL distinguishes between two main categories of properties that an ontology builder may want to define:

- Object properties link individuals to individuals.
- Datatype properties link individuals to data values.

Note: OWL also has the notion of annotation properties (`owl:AnnotationProperty`) and ontology properties (`owl:OntologyProperty`). These are needed in OWL DL for semantic reasons, and are defined above.

An object property is defined as an instance of the built-in OWL class `owl:ObjectProperty`. A datatype property is defined as an instance of the built-in OWL class `owl:DatatypeProperty`. Both `owl:ObjectProperty` and `owl:DatatypeProperty` are subclasses of the RDF class `rdf:Property`.

Note: In OWL Full, object properties and datatype properties are not disjoint. Because data values can be treated as individuals, datatype properties are effectively subclasses of object properties. In OWL Full `owl:ObjectProperty` is equivalent to `rdf:Property` In practice, this mainly has consequences for the use of `owl:InverseFunctionalProperty`. We have introduced the stereotyped «owlProperty» association class as a means of generalizing these notions, as shown in Table 42 (“Properties”).

A property axiom defines characteristics of a property. In its simplest form, a property axiom just defines the existence of a property. Often, property axioms define additional characteristics of properties. OWL supports the following constructs for property axioms:

- RDF Schema constructs: `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`
- relations to other properties: `owl:equivalentProperty` and `owl:inverseOf`
- global cardinality constraints: `owl:FunctionalProperty` and `owl:InverseFunctionalProperty`
- logical property characteristics: `owl:SymmetricProperty` and `owl:TransitiveProperty`

The relevant RDF Schema concepts are defined in 16.1.7 (“Properties in RDF/S”); global cardinality constraints and logical property characteristics are represented as UML properties on either «owlProperty» or «objectProperty», as given in Table 42. The relations between properties are defined in the sections that follow.

Table 42 Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
OWLProperty «owlProperty»	Association Class	«rdfProperty»	isDeprecated: Boolean	n/a	Extends 12.3.3 (“Property”)
OWLFunctionalProperty «isFunctional»	Association Class, Property	«owlProperty»	n/a	n/a	Indicates that an object or datatype property is functional; is defined as a subclass of `rdf:Property`. A functional property can have only one (unique) value y for each instance x, *i.e.* there cannot be two distinct values y1 and y2 such that the pairs (x,y1) and (x,y2) are both instances of this property.
OWLObjectProperty «objectProperty»	Association Class	«owlProperty»	n/a	n/a	Extends 12.3.2 (“OWLObjectProperty”)
OWLInverseFunctionalProperty «isInverseFunctional»	Association Class, Property	«objectProperty»	n/a	n/a	Indicates that an object property is inverse Functional; is defined as a subclass of `owl:ObjectProperty`; Because in OWL Full datatype properties are a subclass of object properties, an inverse-functional property can be defined for datatype properties, which is not possible in OWL DL. If P is an `owl:InverseFunctionalProperty`, then this asserts that a value y can only be the value of P for a single instance x, *i.e.* there cannot be two distinct instances x1 and x2 such that both pairs (x1,y) and (x2,y) are instances of P.

Table 42 Properties

Stereotype	Base Class	Parent	Tags	Constraints	Description
OWLSymmetricProperty «isSymmetric»	Property	«objectProperty»	n/a		Indicates that an object property is symmetric; is defined as a subclass of `owl:ObjectProperty`. A symmetric property is a property for which holds that if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P. The domain and range of a symmetric property are the same.
OWLTransitiveProperty «isTransitive»	Property	«objectProperty»	n/a		Indicates that an object property is transitive; is defined as a subclass of `owl:ObjectProperty`. If a property P is transitive, this means that if a pair (x,y) is an instance of P, and the pair (y,z) is also instance of P, then we can infer the pair (x,z) is also an instance of P.
OWLDatatypeProperty «datatypeProperty»	Association Class	«owlProperty»	n/a	n/a	Extends 12.3.1 ("OWL-DatatypeProperty")

16.2.6.1 owl:equivalentProperty Relation

Description

The `owl:equivalentProperty` construct can be used to state that two properties have the same property extension. Syntactically, `owl:equivalentProperty` is a built-in OWL property with `rdf:Property` as both its domain and range.

> *Note*: Property equivalence is not the same as property equality. Equivalent properties have the same "values" (*i.e.*, the same property extension), but may have different intensional meaning *(i.e.*, denote different concepts). Property equality should be expressed with the `owl:sameAs` construct. As this requires that properties are treated as individuals, such axioms are only allowed in OWL Full.

Stereotype and Base Class

«equivalentProperty» stereotype of *UML::Constraint* between classes stereotyped as «rdfProperty», «owlProperty», «objectProperty», or «datatypeProperty».

Parent

None.

Tags

None.

Constraints

[1] Applies only between global properties.

[2] Instances (property extensions, or sets of tuples) are the same.

Notation

Dashed line between two association classes with stereotype label. An arrowhead should be used opposite from the association class that will have owl:equivalentProperty in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production should result in two instances of owl:equivalentProperty – one for each "side" of the bidirectional constraint.

In cases where there are multiple participants in the same owl:equivalentProperty relation, a constraint note with stereotype label and dashed lines to more than one association class representing the property should be used, similarly to the example for owl:disjointWith.

16.2.6.2 owl:inverseOf Relation

Description

Properties have a direction, from domain to range. In practice, people often find it useful to define relations in both directions: persons own cars, cars are owned by persons. The `owl:inverseOf` construct can be used to define such an inverse relation between properties.

Syntactically, `owl:inverseOf` is a built-in OWL property with `owl:ObjectProperty` as its domain and range. An axiom of the form `P1 owl:inverseOf P2` asserts that for every pair (x,y) in the property extension of P1, there is a pair (y,x) in the property extension of P2, and vice versa. Thus, `owl:inverseOf` is a symmetric property.

Stereotype and Base Class

«inverseOf» stereotype of *UML::Association*.

Parent

None.

Tags

None.

Constraints

[1] Applies only to global properties.

[2] `owl:inverseOf` is binary -- between exactly two object properties.

Notation

A. Several approaches are possible from a notation perspective for representing `owl:inverseOf`. The most natural choice from a UML perspective would be to use a simple association with properties as ends, *i.e.*, a line between classes with properties on the ends closest to their ranges, for example, as shown in Figure 70.

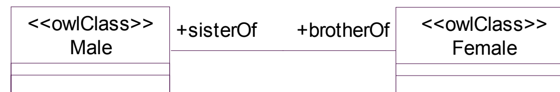


Figure 70 Using owl:inverseOf With Bidirectional Shorthand Notation

Additional constraints if this approach is taken:

[1] (UML) A property cannot be an inverse of itself (use "symmetric")

[2] (UML) A property can have at most one inverse.

B. Alternatively, one could use an «inverseOf» stereotype of *UML::Constraint* between association classes for binary, unidirectional associations, as shown in Figure 71. An arrowhead should be used opposite from the association class that will have `owl:inverseOf` in XML syntax. Shorthand notation that eliminates the arrowhead may be used within an ontology, but XML production should result in two instances of `owl:inverseOf` – one for each "side" of the bidirectional constraint.

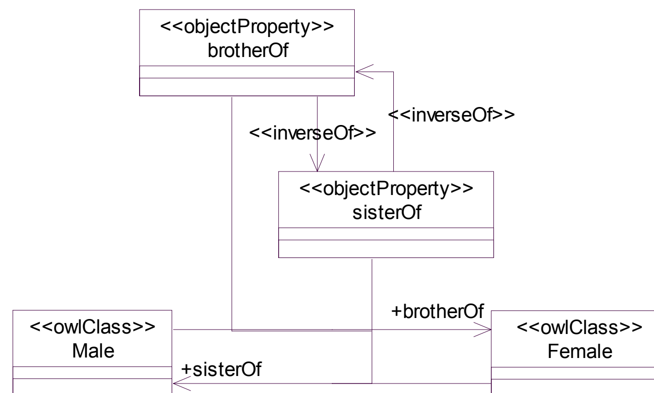


Figure 71 Using owl:inverseOf Between Association Classes

Additional constraints if this approach is taken:

[1] (UML) A property cannot be an inverse of itself (use "symmetric")

C. A third notational option would be to use a stereotype «inverse» of a *UML::Property* with a property:

OF of type *UML:String*

Values for the OF property must be strings, because they cannot be typed by UML metamodel elements. This approach requires namespace notation to resolve name conflicts.

Using a similar notation to the approach taken in 16.2.5.3 ("owl:someValuesFrom and owl:hasValue"), put before the property name: ""inverse" { of = <property-name>, <property-name> }".

Additional constraints if this approach is taken:

[1] Value of OF property must refer to a global property.

[2] (UML) A property cannot be an inverse of itself (use "symmetric")

16.2.7 Individuals

Individuals are defined with individual axioms (also called “facts”). These include:

- Facts about class membership and property values of individuals
- Facts about individual identity

Many languages have a so-called “unique names” assumption: different names refer to different things in the world. On the web, such an assumption is not possible. For example, the same person could be referred to in many different ways (i.e. with different URI references). For this reason OWL does not make this assumption. Unless an explicit statement is being made that two URI references refer to the same or to different individuals, OWL tools should in principle assume either situation is possible.

OWL provides three constructs for stating facts about the identity of individuals:

- owl:sameAs is used to state that two URI references refer to the same individual.
- owl:differentFrom is used to state that two URI references refer to different individuals
- owl:AllDifferent provides an idiom for stating that a list of individuals are all different.

16.2.7.1 Class Membership and Property Values of Individuals

Description

Many facts typically are statements indicating class membership of individuals and property values of individuals. Individual axioms need not necessarily be about named individuals: they can also refer to anonymous individuals.

Stereotype and Base Class

UML::InstanceSpecification typed by a class having the properties desired for the individual. The class may stereotyped by «singleton» to indicate it is for a specific individual[9]. Singleton classes are not translated to OWL, and their properties appear in OWL as properties of the individual.

Parent

None.

Tags

None.

Constraints

[1] Classes stereotyped by «singleton» have exactly one instance each.
[2] Instances of «singleton» classes have exactly one classifier.

Notation

Instance specifications use the same symbol as classes, but their names are underlined, and have a colon separating the instance name from the class name. Singleton classes can be anonymous, omitted from the notation, and generated by tools. Instances of anonymous classes show nothing after the colon.

9. UML supports individuals without classes and properties on such individuals, for tools that choose to support it.

16.2.7.2 owl:sameAs Relation

Description

The built-in OWL property owl:sameAs links an individual to an individual. Such an owl:sameAs statement indicates that two URI references actually refer to the same thing: the individuals have the same "identity".

owl:sameAs statements are often used in defining mappings between ontologies. It is unrealistic to assume everybody will use the same name to refer to individuals, particularly in a web based environment.

Additionally, in OWL Full, where a class can be treated as instances of (meta)classes, the owl:sameAs construct can be used to define class equality, thus indicating that two concepts have the same intensional meaning.

Stereotype and Base Class

«sameAs» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies only between instance specifications, or in OWL Full, between instances or between classes.

Notation

Dashed line between two instances (or classes) with stereotype label. An arrowhead can be used opposite from the instance (or class) that will have owl:sameAs in XML syntax.

Note: need example diagram.

Constraint note with stereotype label and dashed lines to more than one instance (or class - translates to multiple owl:sameAs statements).

Note: need example diagram.

16.2.7.3 owl:differentFrom Relation

Description

The built-in OWL owl:differentFrom property links an individual to an individual. An owl:differentFrom statement indicates that two URI references refer to different individuals.

Stereotype and Base Class

«differentFrom» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies only between instance specifications.

Notation

Dashed line between two instances with stereotype label. An arrowhead can be used opposite from the instance that will have owl:differentFrom in XML syntax.

Note: need example diagram.

Constraint note with stereotype label and dashed lines to more than one instance (translates to multiple owl:differentFrom statements).

Note: need example diagram.

16.2.7.4 owl:AllDifferent Construct

Description

For ontologies in which the unique-names assumption holds, the use of owl:differentFrom is likely to lead to a large number of statements, as all individuals have to be declared pairwise disjoint. For such situations OWL provides a special idiom in the form of the construct owl:AllDifferent. owl:AllDifferent is a special built-in OWL class, for which the property owl:distinctMembers is defined, which links an instance of owl:AllDifferent to a list of individuals. The intended meaning of such a statement is that all individuals in the list are all different from each other.

Stereotype and Base Class

«allDifferent» stereotype of *UML::Constraint*.

Parent

None.

Tags

None.

Constraints

[1] Applies only between instance specifications.

Notation

Constraint note with stereotype label and dashed lines to more than one instance.

Note: need example diagram.

16.2.7.5 Individual Property Values

Description

In RDF, RDF Schema, and OWL, properties of individuals are accessed essentially through the *triples* (or statements), where the individual is the subject of the triple. In this profile, while we have optionally provided explicit access to the elements of the triple in a way that identifies the subject for this purpose, we also provide a more intuitive representation from a UML perspective.

Stereotype and Base Class

No stereotype, use *UML::Slot* to represent properties on individuals.

Parent

None.

Tags

None.

Constraints

[1] Values must conform to constraints on the property, such as type and multiplicity.

Notation

Put values after equal sign at end of property entry in instance.

Note: need example diagram.

16.2.8 Datatypes

OWL allows three types of data range specifications:

- An RDF datatype specification.
- The RDFS class rdfs:Literal.
- An enumerated datatype, using the owl:oneOf construct.

OWL makes use of the RDF datatyping scheme, which provides a mechanism for referring to XML Schema datatypes. Data values are instances of the RDF Schema class rdfs:Literal. Literals can be either plain (no datatype) or typed. Datatypes are instances of the class rdfs:Datatype. In RDF/XML, the type of a literal is specified by an rdf:datatype attribute of which the value is recommended to be one of the following:

- A canonical URI reference to an XML Schema datatype of the form:

    http://www.w3.org/2001/XMLSchema#NAME

  where "NAME" should be the name of a simple XML Schema built-in datatype, with the provisos specified below.

- The URI reference of the datatype rdf:XMLLiteral. This datatype is used to include XML content into an RDF/OWL document. The URI reference of this datatype is:

    http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral

The RDF Semantics document recommends use of the following simple built-in XML Schema datatypes.

- The primitive datatype xsd:string, plus the following datatypes derived from xsd:string: xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, and xsd:NCName.
- The primitive datatype xsd:boolean.
- The primitive numerical datatypes xsd:decimal, xsd:float, and xsd:double, plus all derived types of xsd:decimal (xsd:integer, xsd:positiveInteger. xsd:nonPositiveInteger, xsd:negativeInteger, xsd:nonNegativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte)
- The primitive time-related datatypes: xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonth-

Day, xsd:gDay, and xsd:gMonth.

- The primitive datatypes xsd:hexBinary, xsd:base64Binary, and xsd:anyURI.

Note: It is not illegal, although not recommended, for applications to define their own datatypes by defining an instance of rdfs:Datatype. Such datatypes are “unrecognized”, but are treated in a similar fashion as “unsupported datatypes”.

The set of XML Schema datatypes that are allowable for use in OWL DL are given in the model library provided in Appendix A, Foundation Ontology (M1) for RDFS and OWL.

In addition to the RDF datatypes, OWL provides one additional construct for defining a range of data values, namely an enumerated datatype. In the RDF/XML and OWL syntax, this datatype format makes use of the owl:oneOf construct, that is also used for describing an enumerated class. In the case of an enumerated datatype, the subject of owl:oneOf is a blank node of class owl:DataRange and the object is a list of literals. The stereotype for enumerated datatypes is given in Table 41 (“Class Descriptions”).

17 The Topic Map Profile

This chapter defines a UML2 profile to support the usage of UML notation for the modeling of Topic Maps.

Note that the structure of Topic Maps differs considerably from UML, making the profiles somewhat misleading. UML specifies a class structure, with instances specified by a generic instances model. Topic Map constructs are largely at the individual level, in some cases gathered into classes via a type association.

In particular, the stereotype Topic extends UML Class. An instance of Class is itself a class, while an instance of Topic in the TM metamodel is generally not, although some instances of Topic serve as types for others. The profile only models instances of Topic which are types.

Further, the stereotype Association extends UML Association. An instance of UML Association specifies a set of tuples. An instance of TM Association specifies a particular link among particular individual instances of Topic. However, a TM Association is linked to an instance of Topic which serves as its type. So the stereotype models the instance of Topic which is the type of the TM Association.

17.1 Stereotypes

17.1.1 Topic Map

The Topic Map stereotype is defined as an extension of the UML Package base meta-class, as shown in Figure 72.

Figure 72 Topic Map Stereotype

Applying this stereotype to a package requires that the UML constructs contained within the package be interpreted according to this profile definition.

Tagged Values

- baseLocator - used to specify the storage location of the Topic Map, it must be a URI String.

17.1.2 Topic

The Topic stereotype extends the UML Class meta-class, as shown in Figure 73. Its application indicates that the UML Class is interpreted as a Topic.

Figure 73 Topic Stereotype

Tagged Values

- subjectIdentifier - used to reference the Topic's subjectIdentifier, it must be a URI string.

- subjectLocator - used to reference the Topic's subjectLocators, it must be a URI string.

17.1.3 Association

The association stereotype extends the UML Association meta-class, as shown in Figure 74. Its application indicates the UML Association is interpreted as a Topic Map Association. Both binary and n-ary associations are supported.

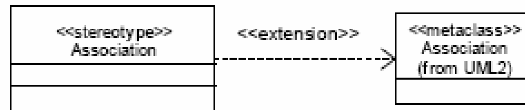


Figure 74 Association Stereotype

17.1.4 Characteristics

The characteristics of Topics use a shared set of stereotypes that extend the UML Properties meta-class, as shown in Figure 75. These four stereotypes are used to model the association roles played by topics and the attributes of topics

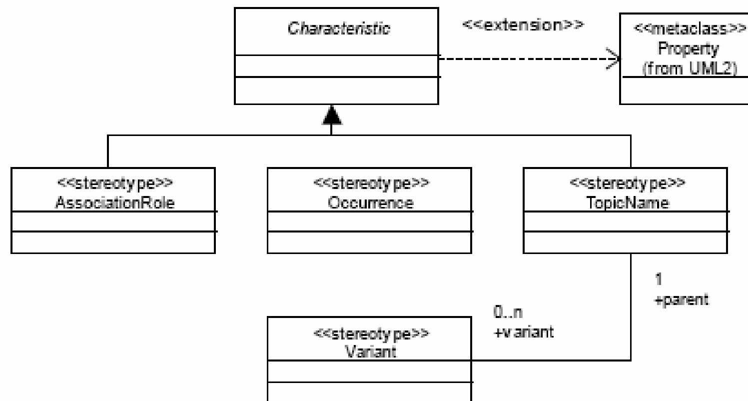


Figure 75 Characteristic Stereotype

AssociationRole

The AssociationRole stereotype is used to indicate an UML Association owned end Property is a Topic Map AssociationRole. The owning UML Association must be stereotyped using the Association stereotype.

Occurrence

The Occurrence stereotype may be applied to either a UML Attribute or a UML Association owned-end Property. Values of this property are interpreted as Topic Map Occurrence values.

Names

The TopicName stereotype may be applied to char array or string typed attributes to indicate that values of the attribute represent Topic Names. The TopicName stereotyped Property may have multiple values for the tagged value 'variant' indicating a set of Variant stereotype Properties that are associated with this TopicName.

The Variant stereotype may be applied to any UML Property, including attributes and association ends, to indicate these values represent Variants. The Variant stereotype Property is required to have a tagged value ‘parent’ linking the variant to a parent TopicName.

17.2 Abstract Bases

Several abstract base meta-classes are defined in the profile. They purpose is to define shared tagged values for sets of stereotype meta-classes.

17.2.1 TopicMapElement

All stereotypes in the profile are specializations of the TopicMapELement abstract base class, as shown in Figure 76. This class provides all profile stereotypes with the ‘itemIdentifiers’ tagged value.

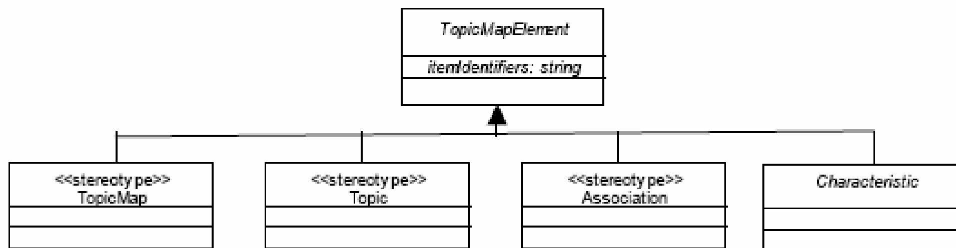


Figure 76 TopicMapElement Stereotypes

Tagged Values

- sourceLocator - used to optionally provide an application specific unique identifier to the stereotyped elements; it must be a URI String.

17.2.2 Scoped Element

Some stereotyped elements in a profiled model, as shown in Figure 77, may have ‘scope’ tagged values that define when this scoped element is applicable.

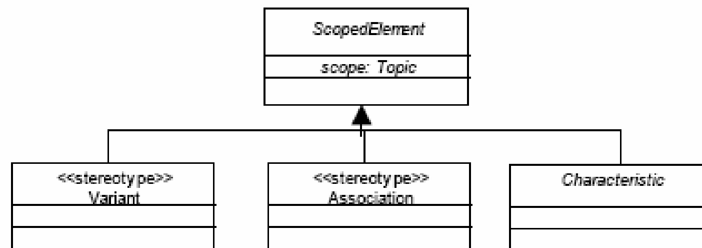


Figure 77 ScopedElement Stereotypes

Tagged Values

- scope – a set of references to Topic Stereotyped elements that define the scope of the element.

17.2.3 TypedElement

Some stereotyped elements in the profiled model, as shown in Figure 78, may have a ‘type’ tagged value. This value references a Topic stereotyped class that defines the general nature of the owning element.

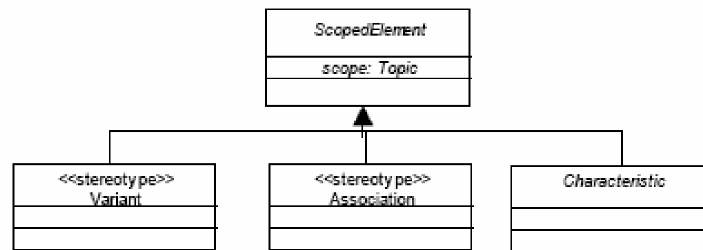


Figure 78 TypedElement Stereotypes

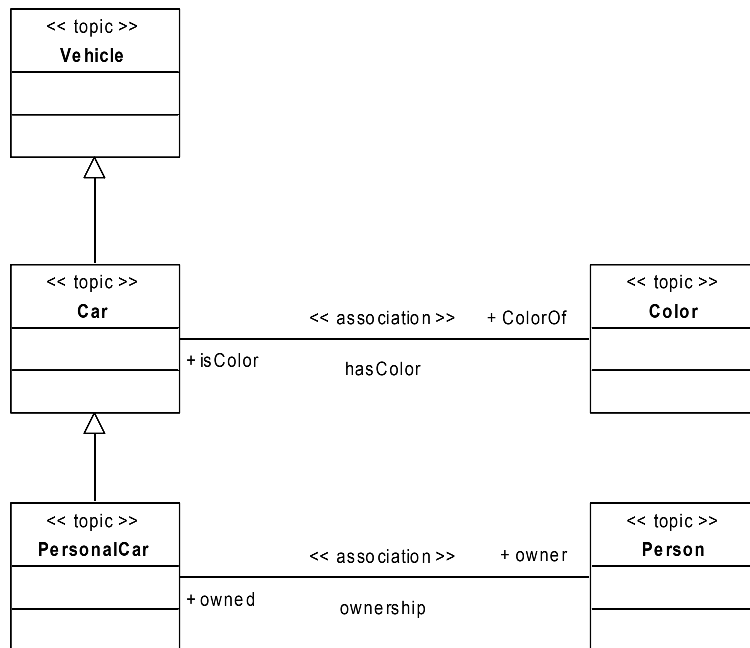
Tagged Values

- type – a reference to a Topic stereotyped element that is the type of this element.

17.3 Example

The figure below, Figure 79, show an example profile applied to a simple UML model of:

- A Personal Car is a Car, which may be owned by a Person.
- A Car is a Vehicle, which may have a Color.
- Carl is a person that owns one Personal Car that is red and another that is blue.



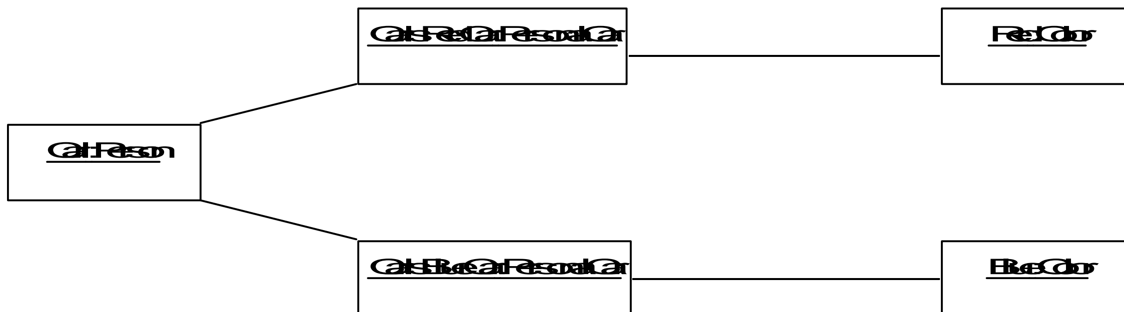


Figure 79 Example Profile

18 Mapping UML to OWL

18.1 Overview

This chapter describes the mapping or transformation between instances of the UML metamodel and those of the OWL metamodel. The mappings are defined in one direction: from UML to OWL. This chapter uses the abstract syntax that is defined in section 19.1.

18.2 UML to OWL Mapping

This section describes mappings from [UML2] models to ODM OWL models. The UML2 metamodel is based on ptc/04-10-02. The mapping is limited to OWL DL, which means only OWL-DL constructs will be used in mapping definitions. There are many abstract metaclasses in UML2 kernel package, so only important concrete classes are mapped to OWL constructs.

18.2.1 Package

A package is a namespace for its members, and may contain other packages. Only packageable elements can be owned members of a package. By virtue of being a namespace, a package can import either individual members of other packages, or all the members of other packages. A package has,

- a name,
- a set of members that are owned by this package,
- a set of PackageImports that are owned by this package.

Therefore, the following URIs can be used to index all elements appearing in this description:

- m: the m-th Package of a model
- m.o: the o-th PackageImport of Package (m)
- m.o.p: the p-th Package whose members are imported by PackageImport (m.o)
- m.c: the c-th Class of Package (m)
- m.a: the a-th Association of Package (m)
- m.i: the i-th InstanceSpecification of Package (m)

UML Abstract Syntax

The above description of a Package can be represented in the UML Abstract Syntax as follows:

```
Package (m
    name (String)
    {importedPackage(m.o:Package)}o
    {ownedMember(m.c:Class)}c
    {ownedMember(m.a:Association)}a
    {ownedMember(m.i:InstanceSpecification)}i)
```

ODM OWL Abstract Syntax

A Package is mapped to a OWLOntology, which has

- a localName,
- a set of imported OWLOntologies,
- a set of contained OWLClasses, OWLRestrictions, OWLDatatypeProperties, OWLObjectProperties and Individuals.

Then the result target model can be represented as follows using the ODM OWL abstract syntax (defined in Section 19.5):

```
UML2OWL(Package (m)) :=
OWLOntology(m
        // Map name to localName
            localName(m.name:String)
        // Map PackageImport.importedPackage to OWLimports
            {OWLimports(m.o.p:OWLOntology)}o
        // Map Class to OWLClass
            {contains(m.c:OWLClass)}c
        // Map Association to OWLObjectProperty
            {contains(m.a:OWLObjectProperty)}a
        // Map InstanceSpecification to Individual
            {contains(m.i:Individual)}i)
```

18.2.2 Class

Class is a kind of classifier whose features are attributes and operations. Attributes of a class are represented by instances of Property that are owned by the class. Some of these attributes may represent the navigable ends of binary associations.

The mapping from Class to OWLClass mainly contains the transformation of generalization relationships between Classes and the Properties that are owned by Class. A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. It has the same semantics of RDFSsubClassOf in RDF Schema, and the two ends of the generalization relationships can be accessed by the source and target that are defined in DirectedRelationship.

The ownedAttribute defines the attributes owned by the class. It is an ordered set of Properties, which can be mapping to either OWLDatatypeProperty or OWLObjectProperty. If a property is part of the memberEnds of an Association, the mapping of it will be discussed in 18.2.3 (“Association”).

If the type of the property is a PrimitiveType, the property is mapped to the OWLDatatypeProperty. If the type of the property is an Enumeration, and the ownedLiteral of the Enumeration has specification as ValueSpecification, then the property is OWLDatatypeProperty. If the type of the property is Class, or the ownedLiteral of an Enumeration type has at least one classifier, the property can be mapped to OWLObjectProperty.

Multiplicity is another issue in mapping. In UML, property is a MultiplicityElement, which defines upperValue and lowerValue to express cardinality. However, OWL uses Restrictions to represent Cardinality. So in addition to map Class to OWLClass, some OWLRestrictions will be generated based on multiplicity definitions of the ownedProperties and corresponding RDFSsubClassOf relationships beween OWLClass and OWLRestriction will also be created.

The following URIs are defined from the source UML model,

- m.c: the c-th Class of Package (m)
- m.c.g: the g-th Generalization of Class (m.c)
- m.c.g.g: the general of Generalization (m.g)
- m..c.p: the p-th ownedAttribute of Class (m.c)
- m.c.p.a: the association of Property (m.c.p)
- m.c.p.u: the upperValue of Property (m.c.p)
- m.c.p.l: the lowerValue of Property (m.c.p)
- m.c.p.t: the type of Property (m.c.p)
- m.c.p.t.ol: the ownedLiteral of the typed Enumeration (m.c.p.t)
- m.c.p.t.ol.s: the specification of the EnumerationLiteral
- m.c.p.t.ol.c: the classifier of the EnumerationLiteral

If m.c.p.a is not NULL, see definition in 18.2.3 ("Association"). In order to simplify the mapping definitions, four different UML abstract syntax and the corresponding target OWL models are defined.

18.2.2.1 Case I - m.c.p.t is PrimitiveType

UML Abstract Syntax

```
Class (m.c
    name (String)
    {generalization(m.c.g:Generalization)}g
    {ownedAttribute(m.c.p:Property)}p)

Generalization(c.g
    {general(c.g.g:Class)}g)

Property (c.p
    name (String)
    {association(c.p.a:Association)}a
    {upperValue(c.p.u:ValueSpecification)}u
    {lowerValue(c.p.l:ValueSpecification)}l
    {type(c.p.t:PrimitiveType)}t)
```

ODM OWL Abstract Syntax

```
UML2OWL(Class (m.c)) :=

// Map Class to OWLClass
   OWLClass(m.c

        // Map name to localName
            localName(m.c.name:String)

        // Map Generalization to RDFSsubClassOf
            {RDFSsubClassOf(m.c.g.g:OWLClass)}g
```

{RDFSsubClassOf(m.c.p.R:OWLRestriction)}p)

// Map Property to OWLDatatypeProperty
OWLDatatypeProperty(m.c.p

// Map name to localName
localName(m.c.p.name:String)
{RDFSdomain(m.c:OWLClass)}
{RDFSrange (m.c.p.t:dataTypeID)})

OWLRestriction(m.c.p.R
{OWLonProperty(m.c.p:OWLDatatypeProperty)}
{OWLmincardinality (m.c.p.l:Integer))
{OWLmaxcardinality (m.c.p.u:Integer)})

18.2.2.2 Case II - m.c.p.t is Enumeration and m.c.p.t.ol is instance of ValueSpecification

UML Abstract Syntax

Class (m.c

name (String)
{generalization(m.c.g:Generalization)}g
{ownedAttribute(m.c.p:Property)}p)

Generalization(c.g

{general(c.g.g:Class)}g)

Property (c.p

name (String)
{association(c.p.a:Association)}a
{upperValue(c.p.u:ValueSpecification)}u
{lowerValue(c.p.l:ValueSpecification)}l
{type(c.p.t:Enumeration)}t)

Enumeration (c.p.t

name (String)
{ownedLiteral(c.p.ol:EnumerationLiteral)}t)

EnumerationLiteral(c.p.t.ol

{specification(c.p.t.ol.s:ValueSpecification)}s)

ODM OWL Abstract Syntax

UML2OWL(Class (m.c)) :=

// Map Class to OWLClass
OWLClass(m.c

// Map name to localName
localName(m.c.name:String)

// Map Generalization to RDFSsubClassOf
{RDFSsubClassOf(m.c.g.g:OWLClass)}g
{RDFSsubClassOf(m.c.p.R:OWLRestriction)}p)

```
// Map Enumeration to OWLDataRange
   OWLDataRange(m.c.p.t.D
               localName(m.c.p.t.name:String)
               {OWLoneOf(m.c.p.t.ol:dataLiteral)}t)

// Map Property to OWLDatatypeProperty
   OWLDatatypeProperty(m.c.p

        // Map name to localName
               localName(m.c.p.name:String)
               {RDFSdomain(m.c:OWLClass)}
               {RDFSrange (m.c.p.t.D:OWLDataRange)})

OWLRestriction(m.c.p.R
               {OWLonProperty(m.c.p:OWLDatatypeProperty)}
               {OWLmincardinality (m.c.p.l:Integer))
               {OWLmaxcardinality (m.c.p.u:Integer)})
```

18.2.2.3 Case III - m.c.p.t is Class

UML Abstract Syntax

```
Class (m.c

   name (String)
   {generalization(m.c.g:Generalization)}g
   {ownedAttribute(m.c.p:Property)}p)

Generalization(c.g

   {general(c.g.g:Class)}g)

Property (c.p

   name (String)
   {association(c.p.a:Association)}a
   {upperValue(c.p.u:ValueSpecification)}u
   {lowerValue(c.p.l:ValueSpecification)}l
   {type(c.p.t:Class)}t)
```

ODM OWL Abstract Syntax

```
UML2OWL(Class (m.c)) :=

// Map Class to OWLClass
   OWLClass(m.c

        // Map name to localName
               localName(m.c.name:String)

        // Map Generalization to RDFSsubClassOf
               {RDFSsubClassOf(m.c.g.g:OWLClass)}g
               {RDFSsubClassOf(m.c.p.R:OWLRestriction)}p)

// Map Property to OWLObjectProperty
   OWLObjectProperty(m.c.p

        // Map name to localName
               localName(m.c.p.name:String)
```

{RDFSdomain(m.c:OWLClass)}
{RDFSrange (m.c.p.t:OWLClass)})

OWLRestriction(m.c.p.R
{OWLonProperty(m.c.p:OWLObjectProperty)}
{OWLmincardinality (m.c.p.l:Integer))
{OWLmaxcardinality (m.c.p.u:Integer)})

18.2.2.4 Case IV - m.c.p.t is Enumeration and m.c.p.t.ol is instance of Class

UML Abstract Syntax

Class (m.c

name (String)
{generalization(m.c.g:Generalization)}g
{ownedAttribute(m.c.p:Property)}p)

Generalization(c.g

{general(c.g.g:Class)}g)

Property (c.p

name (String)
{association(c.p.a:Association)}a
{upperValue(c.p.u:ValueSpecification)}u
{lowerValue(c.p.l:ValueSpecification)}l
{type(c.p.t:Enumeration)}t)

Enumeration (c.p.t

name (String)
{ownedLiteral(c.p.ol:EnumerationLiteral)}t)

EnumerationLiteral(c.p.t.ol

{classifier(c.p.t.ol.c:Class)}c)

ODM OWL Abstract Syntax

UML2OWL(Class (m.c)) :=

// Map Class to OWLClass
OWLClass(m.c

// Map name to localName
localName(m.c.name:String)

// Map Generalization to RDFSsubClassOf
{RDFSsubClassOf(m.c.g.g:OWLClass)}g
{RDFSsubClassOf(m.c.p.R:OWLRestriction)}p)

// Map Enumeration to EnumeratedClass
EnumeratedClass(m.c.p.t.C
localName(m.c.p.t.name:String)
{OWLoneOf(m.c.p.t.ol:Individual)}t)

// Map Property to OWLObjectProperty
OWLObjectProperty(m.c.p

```
        // Map name to localName
            localName(m.c.p.name:String)
            {RDFSdomain(m.c:OWLClass)}
            {RDFSrange (m.c.p.t.C:EnumeratedClass)})

OWLRestriction(m.c.p.R
            {OWLonProperty(m.c.p:OWLObjectProperty)}
            {OWLmincardinality (m.c.p.l:Integer))
            {OWLmaxcardinality (m.c.p.u:Integer)})
```

18.2.3 Association

An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end. More than one end of the association may have the same type. In this specification, only binary association is discussed. The following URIs are defined from the source UML model:

- m.a: the a-th Association of Package (m)
- m.a.p0: the 0-th member end of the association (m.a)
- m.a.p0.t: the type of the 0-the member end (m.a.p0)
- m.a.p0.c: the class that owns the 0-the member end (m.a.p0)
- m.a.p0.u: the upper value of the 0-the member end (m.a.p0)
- m.a.p0.l: the lower value of the 0-the member end (m.a.p0)
- m.a.p1: the 1-th member end of the association (m.a)
- m.a.p1.c: the class that owns the 1-the member end (m.a.p0)
- m.a.p1.t: the type of the 1-the member end (m.a.p1)
- m.a.p1.u: the upper value of the 1-the member end (m.a.p1)
- m.a.p1.l: the lower value of the 1-the member end (m.a.p1)

UML Abstract Syntax

The above description of an Association can be represented in the UML Abstract Syntax as follows:

```
Association (m.a

    {memberEnd(m.a.p0:Property)}
    {memberEnd (m.a.p1:Class)}p)

Property (m.a.p0

    name (String)
    {upperValue(m.a.p0.u:ValueSpecification)}u
    {lowerValue(m.a.p0.l:ValueSpecification)}l
    {type(m.a.p0.t:Class)}t)

Property (m.a.p1

    name (String)
    {upperValue(m.a.p1.u:ValueSpecification)}u
    {lowerValue(m.a.p1.l:ValueSpecification)}l
```

```
{type(c.p1.t:Class)}t)
```

ODM OWL Abstract Syntax

The association can be mapped to two OWLObjectProperties in OWL, as follows:

```
UML2OWL(Association (m.a)) :=

// Map Association to one OWLObjectProperty
   OWLObjectProperty(m.a.p0

        // Map name to localName
            localName(m.a.p0.name:String)
           {RDFSdomain(m.a.p0.c:OWLClass)}
           {RDFSrange (m.a.p0.t:OWLClass)))

        OWLRestriction(m.a.p0.R
           {OWLonProperty(m.a.p0:OWLObjectProperty)}
           {OWLmincardinality (m.a.p0.l:Integer)}
           {OWLmaxcardinality (m.a.p0.u:Integer)})

        OWLClass(m.a.p0.c
            {RDFSsubClassOf(m.a.p0.R)})
// Map Association to another OWLObjectProperty
   OWLObjectProperty(m.a.p1

        // Map name to localName
            localName(m.a.p1.name:String)
           {RDFSdomain(m.a.p1.c:OWLClass)}
           {RDFSrange (m.a.p1.t:OWLClass)))

        OWLRestriction(m.a.p1.R
           {OWLonProperty(m.a.p1:OWLObjectProperty)}
           {OWLmincardinality (m.a.p1.l:Integer)}
           {OWLmaxcardinality (m.a.p1.u:Integer)})

        OWLClass(m.a.p1.c
           {RDFSsubClassOf(m.a.p1.R)})
```

18.2.4 InstanceSpecification

An instance specification may specify the existence of an entity in a modeled system. The entity conforms to the specification of each classifier of the instance specification, and has features with values indicated by each slot of the instance specification. Having no slot in an instance specification for some feature does not mean that the represented entity does not have the feature, but merely that the feature is not of interest in the model. An InstanceSpecification has,

- a name,
- a set of classifiers of the represented instance,
- a set of slots giving the value or values of a structure feature of the instance.

Therefore, the following URIs can be used to defined from the source UML model:

- m.i: the i-th InstanceSpecification of Package (m)
- m.i.c: the c-th classifier of the InstanceSpecification (m.i)

- m.i.s: the s-th slot of the InstanceSpecification (m.i)
- m.i.s.v: the v-th value of the slot (m.i.s)
- m.i.s.d: the defining feature of the slot (m.i.s)

UML Abstract Syntax

The above description of an InstanceSpecification can be represented in the UML Abstract Syntax as follows:

```
InstanceSpecification (m.i

    name (String)
    {classisfier(m.i.c:Classifier)}c
    {slot(m.i.s:Slot)}s)

Slot (m.i.s

    {value(m.i.s.v:ValueSpecification)}v
    {definingFeature(m.i.s.d:StructureFeature)}d)
```

ODM OWL Abstract Syntax

An InstanceSpecification is mapped to an Individual in OWL, which contains:

- a name,
- a set of RDFtypes that specify the rdfs:type of the Individual,
- a set of DatatypeSlots that define the properties and related datatype values
- a set of ObjectSlots that define the properties and related individual values

So the Slot in UML can be mapped to either DatatypeSlot or ObjectSlot according to the value of the Slot. If the value is an InstanceValue, the Slot should be mapped to ObjectSlot; If the value is an Expression, OpaqueExpression or LiteralSpecification, the Slot should be mapped to DatatypeSlot. The resultant OWL model can be represented using OWL abstract syntax as follows:

If value of slot is not InstanceValue

```
UML2OWL(InstanceSpecificaiton (m.i)) :=

// Map InstanceSpecification to Individual
   Individual(m.i

         // Map name to localName
            localName(m.i. name:String)

         // Map classifier to RDFtype
            {RDFtype(m.i.c:OWLClass)}c

         // Map slot to DatatypeSlot
            {DatatypeSlot(m.i.c.s:DatatypeSlot)}s)

// Map Slot to DatatypeSlot
   DatatypeSlot(m.i.s

         // Map value to content
            {content(m.i.s.v:dataLiteral)}v

         // Map definingFeature to OWLDatatypeProperty
```

```
            {property(m.i.s.d:OWLDatatypeProperty)}d)
```

If value of slot is InstanceValue

```
  UML2OWL(InstanceSpecificaiton (m.i)) :=
  // Map InstanceSpecification to Individual
    Individual(m.i
         // Map name to localName
            localName(m.i. name:String)
         // Map classifier to RDFtype
            {RDFtype(m.i.c:OWLClass)}c
         // Map slot to ObjectSlot
            {ObjectSlot(m.i.c.s:ObjectSlot)}s)
  // Map Slot to ObjectSlot
    DatatypeSlot(m.i.s
         // Map value to content
            {content(m.i.s.v:Individual)}v
         // Map definingFeature to OWLObjectProperty
            {property(m.i.s.d:OWLObjectProperty)}d)
```

Note: The remainder of this section will be completed for the next revision of the document.

18.3 OWL to UML Mapping

The elements of the OWL metamodel are mapped into elements of the UML metamodel as specified by the UML Profile for OWL.

Note: This section will be completed for the next revision of the document.

19 ER to OWL Mapping

19.1 Overview

This chapter describes the mappings or transformations between instances of the ER metamodel and those of the OWL metamodel. The mappings are defined in both directions: from ER to OWL, and from OWL to ER.

Two informal frame-like abstract syntaxes are used for representing instances (i.e., models) of the ER metamodel and those of the OWL metamodel, respectively. Mappings are defined using these syntaxes, which provide a conceptual overview of the overall mapping. Nevertheless, both representations are machine processable, therefore the mapping specifications can be used to automatically produce target models from source models.

In the future, when QVT [QVT] is finalized, the mappings will also be defined using QVT, thus providing formal and concrete mapping specifications.

19.1.1 Representation of Source and Target Models

Informal frame-like abstract syntaxes are defined based on the instance model of the ER and OWL metamodels, respectively. They are used to represent both the source and the target models of the mappings. Section 19.4 defines the ER abstract syntax used in this chapter. Section 19.5 defines the ODM OWL abstract syntax. Using these abstract syntaxes, Section 19.2 defines ER-to-OWL mapping and Section 19.3 defines OWL-to-ER mapping.

Each M1 instance (e.g., a Entity) of a M2 model element (e.g., Entity) is given an ID for the convenience of mapping representation. IDs are grouped using the names corresponding to the M2 model elements. For example, all IDs of the instances of Entity are called entityID. That is, each instance of Entity is given an entityID.

ID variables are used to index each model element in a M1 model. For example, an entityID m.e denotes the e-th Entity of the m-th Model in the source ER model. When it is obvious from the context, some ID variables are not shown. The ID variables are annotated with their types to be more informative. That is, a property p having an ID i of type T is written as p(i:T).

Braces ({}) are used to denote inclusion of each element that can be indexed by the ID variables appearing in the content of the braces. For example,

```
{ p( m.n.i:T ) }i
```

denotes that p(m.n.i:T) is added for each m.n.i given m and n fixed but varying i from 1 to the maximum value of i. That is, { p(m.n.i:T) }i is equivalent to p(m.n.1:T), p(m.n.2:T),..., p(m.n.N:T) where N is the maximum number of i.

Alternatives are separated by vertical bars (|). Source contents that are not mapped to target contents are italicized.

19.1.2 Representation of Mapping Specifications

EROWL mappings are defined by two mapping functions: ER2OWL(e) and OWL2ER(e) where e is a model element typed according to the source meta-model. The mapping functions map each typed element from a source model to a set of elements typed according to the target meta-model.

A mapping specification is given for each type the mapping functions take. For example, a mapping of a model element Entity(m.e) to OWL is defined as:

```
ER2OWL( Entity( m.e ) ) :=
    OWLClass( m.e
            RDFSsubClassOf( m.e.s:OWLClass ) }s )
```

Here, ER2OWL(Entity(m.e)) creates OWLClass(m.e) which has zero or more RDFSsubClassOf associations to OWLClass(m.e.s).

19.2 ER to OWL Mapping

This section describes mappings from ODM ER models to ODM OWL models. Each mapping starts with element IDs identifying each element in the source model. Then, a representation of the source model element is given using the IDs. Then, the result of the mapping is followed.

Section Section describes how source models and target models are represented in the abstract syntaxes referring to the ODM metamodels and how IDs are used to identify elements participating in the mappings. Since the approach is basically the same throughout this chapter, it is verbosely described only in Section .

The followings are the types of IDs defined for ER model elements:

- modelID
- entityID
- attributeID
- relationshipID
- roleID
- keyID

Each ID in a source ER model defines a URI in the target OWL model and given properties according to the semantics of the source model.

19.2.1 ER to OWL Mapping Summary

Table 43 shows a summary of the ER to OWL mapping.

Table 43 ER to OWL Mapping Summary

Source: ER Model	Target: OWL Model
NamedElement	RDFSResource
Model	OWLOntology
Entity	OWLClass
Generalization	OWLsubClassOf (See Entity)
Attribute	OWLDatatypeProperty OWLRestriction
Relationship	OWLClass
Role	OWLObjectProperty OWLRestriction

19.2.2 NamedElement

In the ER metamodel, nearly all elements are NamedElement; in the OWL metamodel, nearly all elements are RDFSResouce. In general a NamedElement in the ER metamodel is mapped to a RDFSResource in the OWL metamodel. For convenience, the mapping that applies to all NamedElement in the ER metamodel is specified here rather than repeated in each of the following sections.

ER Abstract Syntax

In the following "n" represents the ID of a NamedElement, e.g., "n" is "m" for a Model, "m.e" for an Entity, and "m.e.a" for an Attribute.

```
NamedElement( n
    name( String )
     uri( String)
    label( String )
    description( String ) )
```

ODM OWL Abstract Syntax

```
ER2OWL( NamedElement( n ) ) :=
RDFSResource( n
    localName( n.name:String )                // use the source model name as localName
    uri( n.uri:String )
    RDFSlabel( n.lable:plainLiteral )
    RDFScomment( n.description:plainLiteral ) )
```

19.2.3 Model

The ODM ER metamodel says that a Model m, among other things, has

- a namespace,
- a set of imported models,
- a set of contents each of which is either an Entity or a Relationship,

Therefore, the following IDs can be used to index all elements appearing in this description:

- m: the m-th modelID of a model.
- m.e: the e-th entityID of Model(m).
- m.r: the r-th relationshipID of Model(m).

ER Abstract Syntax

The above description of a Model can be represented naturally in the ER abstract syntax (defined in Section 19.4) as follows:

```
Model( m
    namespace( String )
    { import( n:Model ) }n
    { contents( m.e:Entity ) }e
    { contents( m.r:Relationship ) }r )
```

ODM OWL Abstract Syntax

A Model is mapped to a OWLOntology which contains all resources identified by the source model IDs. The ODM OWL metamodel says that among other things, an OWLOntology has

- a namespace
- a set of imported OWLOnologies.
- a set of contained RDFResources,

For each entityID or relationshipID, we creates a OWLClass. That is, the IDs in the source model is used to identify the model constructs in the target model. The resulting target model can be represented naturally as follows using the ODM OWL abstract syntax (defined in Section 19.5):

```
ER2OWL( Model( m ) ) :=
OWLOntology( m
     namespace( m.namespace:String )
    // Map import to OWLimports
    { OWLimports( m.n:OWLOntology ) }n
    // Map Entity to OWLClass
    { contains( m.e:OWLClass )
            // Map Attribute to OWLDatatypeProperty and OWLRestriction on the classes
            { contains( m.e.a:OWLDatatypeProperty }e
    // Map Relationship to OWLClass
    { contains( m.r:OWLClass )
            // Map Role to OWLObjectProperty and OWLRestriction on the classes
            { contains( m.r.o:OWLObjectProperty )
              contains( m.r.o.R:OWLRestriction ) }o }r )
```

The source ID variables appear in the target model are used to identify which source model element causes the creation of each target model element. For example, the ID variables used in the above target model assert that each Entity(m.e) is mapped to a OWLClass(m.e) and each Attribute(m.e.a) is mapped to a OWLDataTypeProperty(m.e.a). This tractability can be used to update the target models when the source models are changed.

19.2.4 Entity

An Entity is a set of instances. The following IDs are defined from the source ER model:

- m.e: the e-th entityID of Model(m).
- m.e.s: the s-th supertype entityID of Entity(m.e). // indirectly through Entity.generalizations and // Generalization.supertpe
- m.e.a: the a-th attributeID of Entity(m.e).
- m.e.k: the n-th keyID of Entity(m.e).

ER Abstract Syntax

```
Entity( m.e
    { supertype( m.e.s:Entity ) }s
    { attribute( m.e.a:Attribute ) }a
    { key( m.e.k:Key ) }n )
```

ODM OWL Abstract Syntax

Attributes and Keys are mapped to properties. Restrictions for cardinality etc are added in Attributes and Key mappings.

```
ER2OWL( Entity( m.e ) ) :=
OWLClass( m.e
    { RDFSsubClassOf( m.e.s:OWLClass ) }s )
```

19.2.5 Attribute

An Attribute is mapped as a datatype property of an Entity. Its domain is the Entity, and its range is the datatype.

The following URIs are defined from the source ER model:

- e.a: the a-th attribute of Entity (e).
- e.a.dt: the datatype of Attribute(e.a).

ER Abstract Syntax

```
Attribute( e.a
    datatype( t.a.dt:String ) )
```

ODM OWL Abstract Syntax

```
ER2OWL( Attribute(e.a) ) :=
OWLDatatypeProperty( e.a
    RDFSdomain( e:OWLClass)
    RDFSrange( e.a.dt:dataTypeID ) ) )
```

19.2.6 Relationship and Role

A relationship involves two entities. The cardinalities of the entities involved in a relationship can be classified as optional (0..*) and mandatory (n..*) where n is an integer greater than 0. The following URIs are defined for the source ER model:

- m.r: the n-th relationshipID of Model(m).
- m.r.o1: the first roleID of Relationship(m.r).
- m.r.o2: the second roleID of Relationship(m.r).

The roles of Relationship(m.r):

- m.r.o.car: the cardinality value of Role(m.r.o)
- m.r.o.max: the maximum cardinality value of Role(m.r.o)
- m.r.o.min: the minimum cardinality value of Role(m.r.o)
- m.r.o.e: the entityID of Role(m.r.o)
- m.r.o.k: the keyID of Role(m.r.o).

ER Abstract Syntax

```
Relationship( m.r
    [ roles( m.r.o1:Role ) ]
```

```
        [ roles(m.r.o2:Role) ] )

    Role( r.o1
        entity( r.o1.e:Entity )
        cardinality( r.o1.card:Integer )
        minCardinality( r.o1.max:Integer )
        maxCardinality( r.o1.min:Integer )
        [ key( r.o1.k:Key ) ] )

    Role( r.o2
        entity( r.o2.e:Entity )
        cardinality( r.o2.card:Integer )
        minCardinality( r.o2.max:Integer )
        maxCardinality( r.o2.min:Integer )
        [ key( r.o2.k:Key ) ] )
```

ODM OWL Abstract Syntax

```
    ER2OWL( Relationship( m.r ) ) : =
    OWLClass( m.r                                   // Relationship class
        { RDFSsubClassOf( m.r.o.R:OWLRestriction ) }o )
    OWLObjectProperty( m.r.o1
        OWLdomain( m.r:OWLClass )
        OWLrange( m.r.o1.entity:OWLClass ) )
    OWLObjectProperty( m.r.o2
        OWLdomain( m.r:OWLClass )
        OWLrange( m.r.o2.entity:OWLClass ) )
    OWLRestriction( m.r.o1.R
        OWLonProperty( m.r.o1:OWLObjectProperty )
        OWLcardinality( m.r.o1.card )
        OWLmincardinality( m.r.o1.min )
        OWLmaxcardinality( m.r.o1.max ) )
    OWLRestriction( m.r.o2.R
        OWLonProperty( m.r.o2:OWLObjectProperty )
        OWLcardinality( m.r.o2.card )
        OWLmincardinality( m.r.o2.min )
        OWLmaxcardinality( m.r.o2.max ) ) )
```

19.3 OWL to ER Mapping

This section describes mappings from ODM OWL models to ODM ER models. The followings are the types of IDs defined for OWL model elements:

- ontologyID
- classID
- restrictionID
- dataTypePropertyID

- objectPropertyID
- dataRangeID

Each ID in a source OWL model defines a URI in the target ER model and given properties according to the semantics of the source model.

19.3.1 OWL to ER Mapping Summary

Table 44shows a summary of the OWL to ER mapping.

Table 44 OWL to ER Mapping Summary

Source: OWL Model	Target: ER Model
RDFSResource	NamedElement
OWLOntology	Model
OWLClass	Entity
OWLRestriction	See OWLDatatypeProperty and OWLObjectProperty
OWLDatatypeProperty	Attribute
OWLObjectProperty	Relationship
OWLDataRange	AtomicDomain

19.3.2 RDFSResource

In the OWL metamodel, nearly all elements are RDFSResouce; in the ER metamodel, nearly all elements are NamedElement. In general a RDFSResource in the OWL metamodel is mapped to a NamedElement in the ER metamodel. For convenience, the mapping that applies to all RDFSResource in the OWL metamodel is specified here rather than repeated in each of the following sections.

ODM OWL Abstract Syntax

In the following “r” represents the ID of a RDFSResource, e.g., “r” is “o” for a OWLOntology, “o.c” for an OWLClass, and “m.d” for an OWLDataTypeProperty.

```
RDFSResource( r
    localName( String )
    uri( String )
    { RDFSlabel( r.label:plainLiteral ) }label
    { RDFScomment( r.comment:plainLiteral )  }comment )
```

ER Abstract Syntax

```
OWL2ER(RDFSResource(r) ) :=
NamedElement( r
    name( r.localName:String )
     uri( r.uri:String)
    label( r.label:String )              // one label only
    description( r.comment:String ) )    // one comment only
```

19.3.3 OWLOntology

- o: the o-th ontologyID of OWLOntology(o)
- o.m: the m-th imported ontologyID of OWLOntology(o)
- o.c: the c-th classID of OWLOntology(o)
- o.r: the r-th restrictionID of OWLOntology(o)
- o.d: the d-th dataTypePropertyID of OWLOntology(o)
- o.o: the i-th objectPropertyID of OWLOntology(o)
- o.t: the t-th dataRangeID of OWLOntology(o)

The ontologyID of imported elements are o.m. Therefore, we have:

- o.m.c: the c-th classID of OWLOntology(o.m) and so on.

ODM OWL Abstract Syntax

```
OWLOntology( o
      { OWLimports( o.p:OWLOntology ) }p
      { contains( o.c:OWLClass ) }c
      { contains( o.d:OWLDatatypeProperty ) }d
      { contains( o.o:OWLObjectProperty ) }i
      { contains( o.t:OWLDataRange ) }t )
```

ER Abstract Syntax

```
OWL2ER( OWLOntology( o ) ) :=
Model( o
      // Map imports
      { import( o.p:Model ) }p
      // For each contained OWLClass(o.c),
      { contents( o.c:Entity ) }c
      // For each contained OWLDatatypeProperty(o.d),
      { contents(
                  { Entity( o.d.Domain )                   // create a Domain entity
                    Entity( o.d.Range ) )}d                // create a Range entity
      // For each contained OWLObjectProperty(o.i),
      { contents( o.o:Relationship              // objectProperty to relationship
                  Entity( o.o.Domain )          // create a Domain entity
                  Entity( o.o.Range ) )}i                  // create a Range entity
      // For each OWLDataRange( o.t ),
      { contents( AtomicDomain( o.t ) )}t
```

19.3.4 OWLClass

A OWLClass is mapped to an Entity.

ODM OWL Abstract Syntax

- o.c: the c-th classID of OWLOntology(o)
- c.s: the s-th superclass classID of OWLClass(c)

- c.r: the r-th superclass restrictionID of OWLClass(c)

```
OWLClass( c
    { RDFSsubClassOf( c.s:OWLClass ) }s
    { RDFSsubClassOf( c.r:OWLClass ) }r )
```

ER Abstract Syntax

```
OWL2ER( OWLClass( c ) ) :=
Entity( c
    { supertype( c.s:EREntity ) }s )
```

19.3.5 OWLRestriction

OWLRestrictions are mapped by the mappings for OWLDataTypeProperty and OWLObjectProperty.

19.3.6 OWLDatatypeProperty

Instances of OWLDatatypeProperty can be mapped to instances of Attribute. A OWLDatatypeProperty can have multiple domains and ranges. Multiple domains and ranges mean intersection of those domains and ranges. To represent this, classes representing domains and ranges need to be created.

- d: the d-th dataTypePropertyID of OWLOntology(o)
- d.domain: the domain classID of OWLDatatypeProperty(d)
- d.range: the range dataTypeID of OWLDatatypeProperty(d)

ODM OWL Abstract Syntax

If only one domain is specified:

```
OWLDatatypeProperty( d
    RDFSdomain( d.domain:OWLClass )
    RDFSrange( d.range:RDFSDatatype ) )
```

ER Abstract Syntax

```
OWL2ER( OWLDatatypeProperty( d ) ) :=
Entity( d.domain
    attribute( d:Attribute ) )
Attribute( d
    datatype( d.range:String ) )
```

19.3.7 OWLObjectProperty

OWLObjectProperty can have multiple domains and ranges. Multiple domains and ranges mean intersection of those domains and ranges. To represent this, classes representing domains and ranges need to be created.

- i: the i-th objectPropertyID of OWLOntology(o).
- i.domain: the domain classID of OWLObjectProperty(i).

- i.range: the range classID of OWLObjectProperty(i).

ODM OWL Abstract Syntax

If only one domain is specified:

```
OWLObjectProperty( i
    RDFSdomain( i.domain:OWLClass )
    RDFSrange( i.range:OWLClass ) )
```

ER Abstract Syntax

- i.domain.Role: a new roleID of the domain side of OWLObjectProperty(d).
- i.range.Role: a new roleID of the range side of OWLObjectProperty(d).

```
OWL2ER( OWLObjectProperty( i ) ) :=
Entity( i.domain )
Entity( i.range )
Relationship( i
    roles( i.domain.Role:Role )
    roles( i.range.Role:Role ) )
Role( i.domain.Role
    entity( i.domain:Entity ) )
Role( i.range.Role
    entity( i.range:Entity ) )
```

19.3.8 OWLDataRange

- r: the r-th dataTypeID of OWLOnoloty(o).
- r.s: the s-th subClassOf classID of OWLDataRange(r)

ODM OWL Abstract Syntax

```
OWLDataRange( r
    { OWLoneOf( rdfsLiteralID ) }
    { RDFSsubClassOf( r.s:OWLDataRange ) }s )
```

ER Abstract Syntax

```
OWL2ER( OWLDataRange( r ) ) :=
AtomicDomain( r                        // to add: DomainConstraint
    baseType( r.s:AtomicDomain ) )
```

19.4 ER Abstract Syntax

The syntax notations are based on Section 2 of [W3OWL-ABS:04]. Therefore, its description is exactly borrowed from it:

The abstract syntax is specified here by means of a version of Extended BNF, very similar to the EBNF notation used for XML. Terminals are quoted; non-terminals are bold and not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([]); components that can occur any number of times (including zero) are enclosed in braces ({}). Whitespace is ignored in the productions here.

```
namedElementContent :=
      'name(' String ')'
       ‘uri(‘ String ‘)’
      'label(' String ')'
      'description(' String ')' ‘)’

model ::= 'Model(' modelID
       ‘namespace(‘ String ‘)’
       { ‘prefixes(‘ namespacePrefixID ‘)’ }
       { 'contents(' domainID | entityID | relationshipID ')' }
       { ‘import(‘ modelID ‘)’ }
       [ ‘parent(‘ modelID ‘)’ ]                    // inverse: children
       namedElementContent )'

entity ::= 'Entity(' entityID                       // owned by Model
      { 'supertype(' entityID ')' }                 // indirectly through generalizations and Generalization.supertype
                                                    // inverse: subtype
      { 'attributes(' attributeID ')' }
      { 'keys(' keyID ')' }
      { 'constraints(' constraintID ')' }
      { 'relationships(' relationshipID ')' }       // inverse: Relationship.owningEntity
        abbreviation(‘ String “)’
        namedElementContent ')'

attribute ::= 'Attribute(' attributeID              // owned by Entity
      'datatype(' String ')'
       ‘defaultValue(‘ String ‘)’
       ‘required(‘ boolean ‘)’
      'derived(' boolean ')'
        ‘surrogateKey(‘ boolean ‘)’
        namedElementContent ')'

relationship ::= 'Relationship(' relationshipID// owned by Model
      roles( roleID )
       roles( roleID )
      [ 'relationshipType(' integer ')' ]
        abbreviation(‘ String “)’
        namedElementContent ')'

role ::= 'Role(' roleID                             // owned by Relationship
       ‘verbPhrase(‘ String ‘)’
       ‘navigable(‘ boolean ‘)’
       ‘cardinality(‘ integer ‘)’
```

```
 ‘maxCardinality(‘ integer ‘)’
 ‘minCardinality(‘ integer ‘)’
'entity(' entityID ')'             // inverse: Entity.role
 ‘key(‘ keyID ‘)’                  // inverse: Key.role (Key: AlternateKey, ForeignKey, InversionEntry,
                                   //          PrimaryKey)
namedElementContent ')'
```

19.5 ODM OWL Abstract Syntax

The following abstract syntax for OWL is based on the ODM OWL metamodel and differs in many areas from W3 OWL abstract syntax.

```
rdfsResourceContent :: =
    'namespace(' String ')'
    'localName(' String ')'
    'uri(' String ')'
    { 'RDFtype(' classID | restrictionID ')' }
    { 'RDFSseeAlso(' resourceID ')' }
    { 'RDFSisDefinedBy(' resourceID ')' }
    { 'RDFvalue(' resourceID ')' }
    { 'RDFSmember(' resourceID ')' }
    { 'RDFScomment(' plainLiteral ')' }
    { 'RDFSlabel(' plainLiteral ')' }

owlOntology ::= 'OWLOntology(' ontologyID
    { ‘OWLversionInfo(‘ rdfsLiteralID ‘)’ }
     { 'OWLbackwardCompatibleWith(' ontologyID ')' }
    { 'OWLincompatableWith(' ontologyID ')' }
    { 'OWLpriorVersion(' ontologyID ')' }
    { 'OWLimports(' ontologyID ')' }
    { 'contains('
      classID
      | resrictionID
      | dataTypePropertyID | objectPropertyID
      | ontologyPropertyID | annotationPropertyID
      | dataTypeID
      | individualID ')' }
    rdfsResourceContent ')'

ontologyID ::= URIreference
classID ::= URIreference
restrictionID ::= URIreference
dataTypePropertyID ::= URIreference
objectPropertyID ::= URIreference
ontologyPropertyID ::= URIreference
annotationPropertyID ::= URIreference
datatypeID ::= URIreference
```

```
individualID ::= URIreference

dataLiteral ::= typedLiteral | plainLiteral
typedLiteral ::= lexicalForm^^URIreference
plainLiteral ::= lexicalForm | lexicalForm@languageTag
lexicalForm ::= a Unicode string in normal form C
languageTag ::= an XML language tag

owlClass ::= 'OWLClass(' classID
       'deprecated(' Boolean(false) ')'
       { 'RDFSsubClassOf(' classID | restrictionID ')' }
       { 'OWLdisjointWith(' classID ')' }
       { 'OWLequivalentClass(' classID ')' }
        rdfsResourceContent
        | 'OWLClass(' classID
             { 'OWLoneOf(' individualID ')' }
             | [ 'OWLcomplementOf(' classID ')' ]
             | { 'OWLintersectionOf(' classID ')' }
             | { 'OWLunionOf(' classID ')' } ')' ')'

owlRestriction ::= 'OWLRestriction(' restrictionID
       'OWLonProperty(' objectPropertyID
       | dataTypePropertyID
       | ontologyPropertyID
       | annotationPropertyID ')'
       [ 'OWLhasValue(' classID
           | dataTypeID
           | objectPropertyID
           | dataTypePropertyID
           | ontologyPropertyID
           | annotationPropertyID ')' ]
       [ 'OWLallValuesFrom(' classID | dataTypeID ')' ]
       [ 'OWLsomeValuesFrom(' classID | dataTypeID ')' ]
       [ 'OWLminCardinality(' nonNegativeInteger ')' ]
       [ 'OWLmaxCardinality(' nonNegativeInteger ')']
        [ 'OWLcardinality(' nonNegativeInteger') '] ')'

owlDataTypeProperty ::= 'OWLDatatypeProperty(' dataTypePropertyID
       'deprecated(' Boolean(false) ')'
       'functional(' Boolean(false) ')'
       { 'OWLequivalentProperties(' dataTypePropertyID | objectPropertyID ') '}
       { 'RDFSsubPropertyOf(' dataTypePropertyID ')' }
       { 'RDFSdomain(' classID | anonClassID | restrictionID ')' }
       { 'RDFSrange(' dataTypeID ')' }
        rdfsResourceContent ')'

owlObjectProperty ::= 'OWLObjectProperty(' objectPropertyID
       'deprecated(' Boolean(false) ')'
       'functional(' Boolean(false) ')'
       'inverseFunctional(' Boolean(false) ')'
```

```
        'symmetric(' Boolean(false) ')'
        'transitive(' Boolean(false) ')'
        { 'OWLequivalentProperties(' dataTypePropertyID | objectPropertyID ')' }
        [ 'OWLinverseOf(' objectPropertyID ')' ]
        { 'RDFSsubPropertyOf(' objectPropertyID ')' }
        { 'RDFSdomain(' classID | anonClassID | restrictionID ')' }
        { 'RDFSrange(' classID | anonClassID | restrictionID ')'}
         rdfsResourceContent ')'

owlDataRange ::= 'OWLDataRange(' dataTypeID
        { 'OWLoneOf(' rdfsLiteralID ')' }
         { ‘RDFSsubClassOf(‘ OWLDataRange ‘)’ }
         rdfsResourceContent ')'
```

20 Mapping Topic Maps to OWL

20.1 Overview

The mapping defined below uses a Tefkat like syntax. Tefkat is the concrete syntax proposed for one of the original MOF Q/V/T proposals by DSTC.

The following functions and operators have been defined that are beyond the current Tefkat syntax:

- `genid()` – Creates a new unique identifier.
- `new()` – Creates a new instances of the specified meta-class.
- `<element>.identifier()` – Returns the unique identifier for a Topic Map element, nominally sourceLocator.value.
- `<collection>.contains(<element>)` – returns true if and only if <element> is contained in the <collection>
- `<element1>.instanceOf(<element2>)` – returns true if and only if <element1> is of type <element2>
- `<collection>[i]` – index to the i-th element in the collection, using 1-based indexing.

Where `<collection>` represents a multi-valued MOF association end, and `<element>` represents a single MOF element.

In the processing of the mapping rules, it is assumed that:

- Empty element and null values will not be processed, and
- Equivalent elements will be merged, rather than producing duplicates.

20.2 Topic Maps to OWL Full Mapping

20.2.1 Overview

The elements of the Topic Maps MOF meta-model are mapped into the OWL Full MOF model as shown in the overview table below. In many cases a Topic Map construct can map into several different OWL Full constructs, depending on the usage pattern in the TM meta-model.

There is no equivalent in OWL for the Topic Map concept of Scope, which is not mapped.

20.2.2 Basic Constructs

20.2.2.1 TopicMap

The TopicMap element is mapped into an OWL Ontology. The ID of the ontology is determined by what identifiers are available for the Topic Map.

TopicMap Rules

```
RULE TMap_Onto_Source (tmap, onto)
    FORALL
        TopicMap tmap
    WHERE
        tmap.identifier() != NULL
```

```
    MAKE
        OWLOntology onto
        onto.type = OWLOntology
        onto.ID = tmap.identifier();

RULE TMap_Onto_Base (tmap, onto)
    FORALL
        TopicMap tmap
    WHERE
        tmap.identifier() == NULL
    MAKE
        OWLOntology onto
        onto.type = OWLOntology
        onto.ID = tmap.baseLocator.value;
```

20.2.2.2 Topic

Topics may be mapped to OWL Classes or OWL Individuals or both depending on their usage. The Topic_Class mapping rule checks for three patterns that imply the Topic is being used like a Class.

1. If the topic is the value of topicPlayingRole for any instance AssociationRole in an Association with the type-subject[10] of tmcore:supertype-subtype
2. If the topic has as its type another Topic with the type-subject of tmcore:type, from a tmcore:type-instance Association.
3. If the topic is the value of another topics type property.

Throughout this section the namespace prefix 'tmcore:' is used for the URI http://psi.topicmaps.org/sam/1.0/# This namespace and its included published subjects are defined in the Topic Map Data Model [TMDM] in Section 15.4.

The Topic_Individual mapping rule checks for the inverse of conditions 2 and 3 above.

Topic Rules

```
RULE Topic_Class (top, oc, asr_ti, asr_ss)
    FORALL
        Topic top
        Topic inst
        AssociationRole asr_ti
        AssociationRole asr_ss
    WHERE
        inst.type == top OR
        (asr_ti.parent.type == 'tmcore:type-instance' AND
         asr_ti.topicPlaingRole == top AND
         asr_it.type.subjectIdentifier == 'tmcore:type'
         ) OR
        (asr_ss.parent.type == 'tmcore:supertype-subtype' AND
         asr_ss.topicPlayingRole == top
         )
    MAKE
        OWLClass oc
        oc.type = OWLClass
        oc.ID = top.identifier();
```

10. Note that the phrase 'with the type-subject is' is shorthand for 'with a type that is a topic whose subjectIdentifier is'.

```
RULE Topic_Individual (top, indv, asr_ti, asr_ss)
    FORALL
        Topic top
        Topic inst
        AssociationRole asr_ti
        AssociationRole asr_ss
    WHERE
        top.type != NULL OR
        (asr_ti.parent.type == 'tmcore:type-instance' AND
         asr_ti.topicPlayingRole == top AND
         asr_ti.type.subjectIdentifier == 'tmcore:instance'
         )
    MAKE
        OWLIndividual indv
        indv.type = OWLIndividual
        indv.ID = top.identifier();
```

20.2.2.3 Associations and Association Roles

Topic Map Associations and Association Roles are mapped to OWL Object Properties. If no identifier() is available for the TM construct, then a generated identifier is used for the Object Property.

Association Mapping Rules

```
RULE Assoc_ObjProp (as, oop)
    FORALL
        Association as
    WHERE
        assoc.identifier() != NULL
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
        oop.ID = as.identifier()

RULE Assoc_ObjProp_Anon (as, oop)
    FORALL
        Association as
    WHERE
        assoc.identifier() == NULL
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
        oop.ID = genid()

RULE Role_ObjProp (ar, oop)
    FORALL
        AssociationRole ar
    WHERE
        ar.identifier() != NULL
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
        oop.ID = ar.identifier()

RULE Role_ObjProp_Anon (as, oop)
    FORALL
        AssociationRole ar
    WHERE
```

```
        ar.identifier() == NULL
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
        oop.ID = genid()
```

20.2.2.4 Occurrences

Occurrences may be mapped to either OWL Object Properties or to OWL DatatypeProperties. Occurrences with a resource that is an instance of Locator become Object Properties, those with a resource that is an instance of Data become Datatype Properties.

If no identifier() is available for the TM construct, then a generated identifier is used for the OWL property construct.

Occurrence Rules

```
RULE Ocrr_DTyProp (ar, dtp)
    FORALL
        Occurrence ocr
    WHERE
        ocr.identifier() != NULL AND
        ocr.resource.instanceOf(Data)
    MAKE
        OWLDatatypeProperty dtp
        dtp.type = OWLDatatypeProperty
        dtp.ID = ocr.identifier();

RULE Ocrr_DTyProp_Anon (as, dtp)
    FORALL
        Occurrence ocr
    WHERE
        ocr.identifier() == NULL AND
        ocr.resource.instanceOf(Data)
    MAKE
        OWLDatatypeProperty dtp
        dtp.type = OWLDatatypeProperty
        dtp.ID = genid();

RULE Ocrr_ObjProp (ar, oop)
    FORALL
        Occurrence ocr
    WHERE
        ocr.identifier() != NULL AND
        ocr.resource.instanceOf(Locator)
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
        oop.ID = ocr.identifier();

RULE Ocrr_OBjProp_Anon (as, oop)
    FORALL
        Occurrence ocr
    WHERE
        ocr.identifier() == NULL AND
        ocr.resource.instanceOf(Locator)
    MAKE
        OWLObjectProperty oop
        oop.type = OWLObjectProperty
```

```
oop.ID = genid();
```

20.2.3 Property Restriction Patterns

Each of the OWL property constructs created by a basic pattern rule may also map to class property restrictions in OWL.

Property Restriction Rules

```
RULE Role_PropRestiction (as, asr_s, asr_o, ocls, oop_s, rdfs_o, res)
    FORALL
        Association as
        AssociationRole asr_s
        AssociationRole asr_o
        OWLClass ocls
        OWLObjectProperty oop_s
        RDFSResource rdfs_o
    WHERE
        as.roles.contains(asr_s) AND
        as.roles.contains(asr_o) AND
        as.type.identifier() == ocls.ID AND
        asr_s.type.identifier() == oop_s.ID AND
        asr_o.type.identifier() == rdfs_o.ID
    MAKE
        OWLRestriction res
        res.type = OWLRestriction
        ocls.subClassOf = res
        res.onProperty = oop_s
        res.someValuesFrom = rdfs_o

RULE Ocr_ObjPropRestiction (top, top_ocr, ocls, oop, rdfsr, res)
    FORALL
        Topic top
        Occurrence top_ocr
        OWLClass ocls
        OWLObjectProperty oop
        RDFSResource rdfsr
    WHERE
        top.occurrences.contains(top_ocr) AND
        top_ocr.resource.instanceOf(Locator) AND
        top.identifier() == ocls.ID AND
        top_ocr.identifier() == oop.ID AND
        top_ocr.type.identifier() == rdfsr.ID
    MAKE
        OWLRestriction res
        res.type = OWLRestriction
        ocls.subClassOf = res
        res.onProperty = oop
        res.someValuesFrom = rdfsr

RULE Ocr_DTPropRestiction (top, top_ocr, ocls, dtp, rdfsr, res)
    FORALL
        Topic top
        Occurrence top_ocr
        OWLClass ocls
        OWLDatatypeProperty dtp
```

```
        RDFSResource rdfsr
    WHERE
        top.occurrences.contains(top_ocr) AND
        top_ocr.resource.instanceOf(Data) AND
        top.identifier() == ocls.ID AND
        top_ocr.identifier() == dtp.ID AND
        top_ocr.type.identifier() == rdfsr.ID
    MAKE
        OWLRestriction res
        res.type = OWLRestriction
        ocls.subClassOf = res
        res.onProperty = dtp
        res.someValuesFrom = rdfsr
```

20.2.4 Type Hierarchy Pattern

Associations having the type of `tmcore:supertype-subtype` are mapped into RDFS subClassOf relations.

Type Hierarchy Rules

```
RULE ClassHeirarchy (as, asr_sup, asr_sub, sup_ocls, sub_ocls)
    FORALL
        Association as
        AssociationRole asr_sup
        AssociationRole asr_sub
        OWLClass sup_ocls
        OWLClass sub_ocls
    WHERE
        as.subjectIdentifier == 'tmcore:superType-subType' AND
        as.roles.contains(asr_sup) AND
        asr_sup.subjectIdentifier == 'tmcore:superType' AND
        as.roles.contains(asr_sub) AND
        asr_sub.subjectIdentifier == 'tmcore:subType' AND
        asr_sup.identifier() == sup_ocls.ID AND
        asr_sub.identifier() == sub_ocls.ID
    MAKE
        asr_sub.subClassOf = asr_sup;
```

20.2.5 Naming Patterns

Topic Names and Variant Names are converted into OWL Labels. For Variant Names, two possibilities exist. Those with resources that are instances of Data are mapped to labels having literal values, however those with resources that are instances of Locator are mapped to OWL labels having resource references.

Naming Rules

```
RULE TopicName_Label (top, ocls, ch_tn)
    FORALL
        Topic top,
        OWLClass ocls,
        TopicName ch_tn
    WHERE
        top.identifier() == ocls.ID AND
        top.characteristics.contains(ch_tn)
    MAKE
```

```
        ocls.label = new RDFSLiteral(ch_tn.value) // M1

RULE VariantName_Object (top, oidv, ch_tn, tn_vn, oobp, obpv)
    FORALL
        Topic top,
        OWLIndividual oidv,
        TopicName ch_tn,
        VariantName tn_vn,
        OWLObjectProperty oobp
    WHERE
        top.identifier() == oidv.ID AND
        top.characteristics.contains(ch_tn) AND
        ch_tn.variants.contains(tn_vn) AND
        tn_vn.resource.instanceOf(Locator) AND
        tn_vn.resource.notation == 'URI' AND
        tn_vn.type == oobp.ID
    MAKE
        ObjectPropertyValue obpv
        oidv.objectValue = obpv
        obpv.property = OWLLabel
        obpv.content = new(RDFSResource(tn_vn.resource.value)) //M1

RULE VariantName_Data (top, ocls, ch_tn, tn_vn)
    FORALL
        Topic top,
        OWLClass ocls,
        TopicName ch_tn,
        VariantName tn_vn
    WHERE
        top.identifier() == ocls.ID AND
        top.characteristics.contains(ch_tn) AND
        ch_tn.variants.contains(tn_vn) AND
        tn_vn.resource.instanceOf(Data)
    MAKE
        ocls.label = new RDFSLiteral(tn_vn.value) // M1
```

20.2.6 Instance Patterns

Topic, Associations and Occurrence instances in Topic Maps are mapped into Object and Datatype property values.

Instance Rules

```
RULE Topic_Indiv (top, rdfsr)
    FORALL
        Topic top
        RDFSResource rdfsr
    WHERE
        top.indentifier() == rdfsr AND
        top.type.size() > 0
    MAKE
        rdfsr.type = OWLIndividual;

RULE Topic_OccurInstn_Indiv (top, oidv, ch_oc, oobp, obpv, rdfsr)
    FORALL
        Topic top,
        OWLIndividual oidv,
        Occurrence ch_oc,
```

```
        OWLObjectProperty oobp
        RDFSResource rdfsr
    WHERE
        top.identifier() == oidv.ID AND
        top.occurrences.contains(ch_oc) AND
        ch_oc.resource.instanceOf(Locator) AND
        ch_oc.resource.notation == 'URI' AND
        ch_oc.resource.value == rdfsr.ID AND
        ch_oc.type == oobp.ID

    MAKE
        ObjectPropertyValue obpv
        oidv.objectValue = obpv
        obpv.property = oobp
        obpv.content = rdfsr;

RULE Topic_OccurInstn_DTy (top, oidv, ch_oc, odtp, dtpv)
    FORALL
        Topic top,
        OWLIndividual oidv,
        Occurrence ch_oc,
        OWLDatatypeProperty odtp
    WHERE
        top.identifier() == oidv.ID AND
        top.characteristics.contains(ch_oc) AND
        ch_oc.resource.instanceOf(Data) AND
        ch_oc.type == odtp.ID
    MAKE
        DatatypePropertyValue dtpv
        oidv.datatypeValue = dtpv
        dtpv.property = odtp
        dtpv.content = new(RDFSLiteral(ch_oc.resource.value)); //M1

RULE Topic_OccurInstn_Obj (as, asr_s, asr_o, rdfs_s, rdfs_o, obpv)
    FORALL
        Association as
        AssociationRole asr_s
        AssociationRole asr_o
        RDFSResource rdfs_s
        RDFSResource rdfs_o
    WHERE
        as.roles.contains(asr_s) AND
        as.roles.contains(asr_o) AND
        rdfs_s == asr_s.topicPlayingRole AND
        rdfs_o == asr_o.topicPlayingRole
    MAKE
        ObjectPropertyValue obpv
        rdfs_s.objectValue = obpv
        obpv.property = asr_s.type
        obpv.content = rdfs_o;
```

20.3 OWL to Topic Maps Mapping

20.3.1 Overview

The constructs in the OWL MOF Meta-model, which can be mapped into the Topic Maps MOF meta-model, are shown below in Table 15-2. Those constructs in OWL that are not included in the table have no equivalent Topic Map construct and are not included in this mapping. For example, there is no concept in Topic Maps for restricting the type or cardinality of a property.

OWL Resources that are multiple types are mapped into a single Topic with appropriately merged characteristics. For example consider and OWL Class and OWL Individual. The Class-ness of the Topic would result in it being used as the value of another Topic Map Constructs type relation; its individual-ness would result in it having a subject identifier relation.

20.3.2 Basic Constructs

20.3.2.1 Ontology

OWL Ontologies are mapped to Topic Maps. The URI of the Ontology becomes the source locator of the Topic Map.

Ontology Rule

```
RULE TMap_Onto_Source (tmap, onto)
    FORALL
        OWLOntology onto
    WHERE
        onto.ID != NULL
    MAKE
        TopicMap tmap
        tmap.sourceLocator.value = onto.ID
        tmap.sourceLocator.notation = 'URI';
```

20.3.2.2 Class

OWL Classes are mapped to Topics. The Class' URI becomes the Topic's subject Identifier.

Class Rules

```
RULE Class_Topic (top, oc)
    FORALL
        OWLClass oc
    WHERE
        oc.type.contains(OWLClass)
    MAKE
        Topic top
        top.sourceLocator.value = oc.ID
        top.sourceLocator.notation = 'URI';
```

20.3.2.3 Individual

OWL Individuals are mapped to Topics. The Individual's URI becomes the Topic's subject Identifier.

Mapping Rules

```
RULE Individual_Topic (top, indv)
    FORALL
```

```
        OWLIndividual indv
    WHERE
        indv.type.contains(OWLIndividual)

    MAKE
        Topic top
        top.subjectIdentifier.value = indv.ID
        top.subjectIdentifier.notation = 'URI';
```

20.3.2.4 Properties

OWL Object Properties are mapped to Topic Map Association Roles, while OWL Datatype Properties are mapped into Topic Map Occurrences. This results in a slight asymmetry in the round trip mapping. Occurrences that have Locators as their resource value will become Associations as the result of a round trip mapping through OWL.

Property Rules

```
RULE ObjProp_Role (ar, oop)
    FORALL
        OWLObjectProperty oop
    WHERE
        oop.type.contains(OWLObjectProperty)
    MAKE
        AssociationRole ar
        ar.sourceLocator.value = oop.ID
        ar.sourceLocator.notation = 'URI';

RULE DTyProp_Occr (ocr, dtp)
    FORALL
        OWLDatatypeProperty dtp
    WHERE
        oop.type.contains(OWLDatatypeProperty)
    MAKE
        Occurrence ocr
        ocr.sourceLocator.value = dtp.ID
        ocr.sourceLocator.notation = 'URI';
```

20.3.2.5 Property Restrictions

OWL Property restrictions are used to associate Association Roles and Occurrences with Topics.

Property Restriction Rules

```
RULE ObjPropRest_Role (res, ocls, oop_s, rdfs_o, ocls_top, as, asr_s, asr_o)
    FORALL
        OWLRestriction res
        OWLClass ocls
        OWLObjectProperty oop_s
        RDFSResource rdfs_o
        Topic ocls_top
    WHERE
        ocls.subClassOf.contains(res) AND
        ocls_top.identifier() == ocls.ID AND
        res.onProperty == oop_s AND
        res.somValuesFrom == rdfs_o
    MAKE
        Association as
        AssociationRole asr_s
```

```
        AssociationRole asr_o

        as.type = ocls_top
        asr_s.subjectIdentifier = oop_s.ID
        as.roles = asr_s
        asr_o.subjectIdentifier = rdfs_o.ID
        as.roles = asr_o;

RULE DTypPropRest_Occr (res, ocls, dtp_s, rdfs_o, ocls_top, ocr)
    FORALL
        OWLRestriction res
        OWLClass ocls
        OWLDatatypeProperty dtp_s
        RDFSResource rdfs_o
        Topic ocls_top
    WHERE
        ocls.subClassOf.contains(res) AND
        ocls_top.identifier() == ocls.ID AND
        res.onProperty == dtp_s AND
        res.somValuesFrom == rdfs_o
    MAKE
        Occurrence ocr
        ocr.type = rdfs_o
        ocr.subjectIdentifier = dtp_s.ID
        ocr.parent = ocls_top;
```

20.3.3 Class Hierarchy

OWL Class hierarchies expressed using OWL subClassOf properties are mapped into type hierarchies in Topic Maps. There is no equivalent mapping for OWL subPropertyOf.

Class Hierarchy Rule

```
RULE ClassHeirarchy (as, asr_sup, asr_sub, sup_ocls, sub_ocls)
    FORALL
        OWLClass sup_ocls
        OWLClass sub_ocls
        Topic sup_top
        Topic sub_top
    WHERE
        sub_ocls.subClassOf.contains(sup_ocls) AND
        sup_top.identifier() == sup_ocls.ID AND
        sub_top.identifier() == sub_ocls.ID

    MAKE
        Association as
        AssociationRole asr_sup
        AssociationRole asr_sub

        as.subjectIdentifier = 'tmcore:superType-subType'
        as.roles = asr_sup
        asr_sup.subjectIdentifier = 'tmcore:superType'
        asr_sup.topicPlayingRole = sup_top
        as.roles = asr_sub
        asr_sub.subjectIdentifier = 'tmcore:subType'
        asr_sub.topicPlayingRole = sub_top;
```

20.3.4 Labels

OWL Labels are mapped to Topic Map base names.

Label Rule

```
RULE Label_TopicName (top, ocls, ch_tn)
    FORALL
        Topic top,
        OWLClass ocls,
    WHERE
        top.identifier() == ocls.ID AND
        ocls.label != NULL
    MAKE
        TopicName ch_tn
        ch_tn.value = ocls.label
        top.baseName = ch_tn;
```

20.3.5 Instances

The datatype and object property values of OWL Individuals are mapped into Topic Map Occurrences and Associations respectively.

Instance Rules

```
RULE Topic_OccurInstn_DTy (oidv, top, dtpv, ch_oc, ch_oc_dt)
    FORALL
        OWLIndividual oidv,
        Topic top,
        DatatypePropertyValue dtpv
    WHERE
        oidv.ID == top.identifier() AND
        dtpv == oidv.datatypeValue
    MAKE
        Occurrence ch_oc
        Data ch_oc_dt
        top.occurrences = ch_oc
        ch_oc.resource = ch_oc_dt
        ch_oc.type = dtpv.property.ID
        ch_oc_dt.value = dtpv.content;

RULE ObjProp_AssocInst (oidv, top, obpv, as, asr_s, asr_o)
    FORALL
        OWLIndividual oidv,
        Topic top,
        ObjectPropertyValue obpv
        Association as
    WHERE
        oidv.ID == top.identifier() AND
        obpv == oidv.objectValue
    MAKE
        AssociationRole asr_s
        AssociationRole asr_o
        as.roles = asr_s
        as.roles = asr_o
        asr_s.type = obpv.property.ID
```

```
asr_s.topicPlayingRole = obpv.parent.ID
asr_o.topciPlayingRole = obpv.content.ID;
```

20.3.6 Example

Table 45 below summarizes some equivalent Topic Map and OWL constructs, serialized using their respective XML representations.

Table 45 Equivalent Topic Map and OWL Constructs

Topic Map XTM	OWL RDF/XML
<topic id="Car"> <baseName><baseNameString>Car</baseNameString> </baseName> </topic>	<Class rdf:ID="Car"> <label>Car</label> </Class>
<topic id="PersonalCar"> <baseName> <baseNameString>Personal Car</baseNameString> </baseName> <instanceOf> <topicRef xlink:href="#Car"/> </instanceOf> </topic>	<Class rdf:ID="PersonalCar"> <label>Personal Car</label> <subClassOf rdf:resource="#Car"/> </Class>
<topic id="CarlsRedCar"> <baseName> <baseNameString>Carl's Red Car</baseNameString> </baseName> <instanceOf> <topicRef xlink:href="#PersonalCar"/> </instanceOf> </topic>	<PersonalCar rdf:ID="CarlsRedCar"/>
<association id="CarlsRedCarIsRed"> <instanceOf> <topicRef xlink:href="#hasColor"/> </instanceOf> <member> <roleSpec> <topicRef xlink:href="#ColorOf"/> </roleSpec> <topicRef xlink:href="#CarlsRedCar"/> </member> <member> <roleSpec> <topicRef xlink:href="#IsColor"/> </roleSpec> <topicRef xlink:href="#Red"/> </member> </association>	<ObjectProperty rdf:ID="hasColor"/> <Description rdf:about="CarlsRedCar"> <subClassOf> <Restriction> <onProperty rdf:resource="hasColor"/> <hasValue> <Color rdf:ID="Red"/> </hasValue> </Restriction> </subClassOf> </Description>

21 Mapping RDFS and OWL to CL

21.1 Overview

Mapping from the W3C Semantic Web languages, the Resource Description Framework [RDF Primer] [RDF Concepts] and the Web Ontology Language [OWL S&AS] to Common Logic (CL) is relatively straightforward, as per the draft mapping under development by Pat Hayes [SCL Translation] for incorporation in ISO 24707 [ISO 24707]. The mapping supports translation of RDF vocabularies and OWL ontologies from the RDFS and OWL metamodels, respectively, to the CL metamodel, in the spirit of the language mapping. Users are encouraged to familiarize themselves with the original translation specification and to recognize that the overarching goal is to preserve not only the abstract syntax of the source languages but their underlying semantics, such that CL reasoners can accurately represent and reason about content represented in knowledge bases that reflect those models. The mapping, as it stands, is intended to take an RDFS/OWL ontology as input and map it directly to CL from the triple format. Additional work, including (1) a direct mapping from an RDFS/OWL ontology represented solely in a UML/MOF environment using the metamodels and profiles contained herein, (2) representation of the mappings using MOF QVT, (3) a lossy, reverse mapping from CL to RDFS/OWL using MOF QVT to preserve lossy information, and (4) bi-directional mappings from CL to and from UML 2, again using MOF QVT to preserve lossy information, are planned.

Note that we have not attempted to address the issues raised in [SCL Translation] regarding the distinction between an embedded or translation approach to determining how to map language constructs – such decisions are left to the vendor, depending on the target application(s).

21.2 RDFS to CL Mapping

The separation between RDF and RDF Schema given in [SCL Translation] is not maintained in the ODM, which supports RDF Schema by design. As discussed in the Design Rationale, maintaining that separation from a MOF/UML perspective did not make sense, since (1) it is difficult, at best, to separate the abstract syntax of RDF from that of RDF Schema, and (2) the goal of ODM is to support ontology definition in MOF and UML tools, which is most commonly done using RDF Schema, OWL, or another knowledge representation language. Basic RDF graphs can be translated to CL using the mapping described herein, however.

21.2.1 RDF Triples

Any simple RDF triple (expressed as *subject predicate object*), can be embedded in CL as `(rdf_triple subject predicate object)`[1] and/or translated to an CL atomic sentence directly `(predicate subject object)`[2]. These mappings can be expressed in terms of the metamodel elements shown inTable 46.

Table 46 RDF Triple to CL Mapping

RDFS Metamodel Element	RDFS Metamodel Property	CL Metamodel Element	CL Metamodel Property
RDFStatement		Relation[1]	`predicate: rdf_triple`
	RDFsubject		`arguments [1]`
	RDFpredicate		`arguments [2]`
	RDFobject		`arguments [3]`

Table 46 RDF Triple to CL Mapping

RDFStatement	RDFpredicate	Relation[2]	`predicate`
	RDFsubject		`arguments [1]`
	RDFobject		`arguments [2]`

These two approaches are completely compatible, and the relationship between them can be expressed through the axiom:

```
(forall (x y z)(iff (rdf_triple y x z)(x y z)))
```

The translation extends this notion further through to ensure that the predicate expressed by the triple is indeed a valid RDF property, the "cautious translation approach":

RDF Property Axiom

```
(forall (x y z)(iff (rdf_triple y x z)(and (rdf:Property x)(x y z))))
```

RDF Promiscuity Axiom

```
(forall (x)(rdf:Property x))
```

For the purposes of this specification, any RDF or RDFS predicate that is not explicitly mapped to CL can be translated directly using this method.

21.2.2 RDF Literals

Literals in RDF can be defined as either "plain literals" or "typed literals", corresponding to classes of the same names in the RDFS metamodel. Plain literals translate into CL quoted strings, possibly paired with a language tag, and in both RDF and CL they are understood to denote themselves. The function `stringInLang` is used to indicate the pair consisting of a plain literal and a language tag. Typed literals in RDFS and OWL have two parts: a character string representing the lexical form of the literal, and a datatype name that indicates a function from a lexical form to a value. In RDFS/OWL these two components are incorporated into a special literal syntax; in CL, the datatype is represented as a function name applied to the lexical form as an argument. Table 47 provides the corresponding metamodel mappings.

21.2.3 RDF URIs and Graphs

URIs and URI references can be used directly as CL names. Blank nodes in an RDF graph translate to existentially bound variables with a quantifier whose scope is the entire graph. A graph is the conjunction of the triples it contains. Basic translation for the corresponding metamodel elements is given in Table 47.

Table 47 Basic RDF to CL Mapping

RDFS Metamodel Element	RDFS Metamodel Property	CL Metamodel Element	CL Metamodel Property
RDFSResource	URI reference *"aaa"* –or– namespace and local name *"aaa"* –or– blank node ID *"_:aaa"*	LogicalName	`string:` *`aaa`*
PlainLiteral	lexicalForm: *"aaa"*	SpecialName	`specialNameKind: quotedString, string:` *`aaa`*

Table 47 Basic RDF to CL Mapping

PlainLiteral	lexicalForm: *"aaa"* languageTag: *"tag"*	Function	`operator: stringInLang, arguments [1]: aaa, arguments [2]: tag`
TypedLiteral	lexicalForm: *"aaa"* datatypeURI: *"ddd"*	Function	`operator: ddd arguments [1]: aaa`
RDFDescription	contains: RDFSResource	CLModule	`CLText: Phrase`
RDF graph (set of triples) $\{ttt_1,...,ttt_n\}$		Sentence	(exists(bbb_1...bbb_m) (and ttt_1 ... ttt_n) where _:bbb_1... _:bbb_m are all the blank node IDs in the graph.

For example, the RDF graph

```
_:x ex:firstName "Jack"^^xsd:string .
_:x rdf:type ex:Human .
_:x Married _:y .
_:y ex:firstName "Jill"^^xsd:string .
```

maps into the CL sentence:

```
(exists (x y)(and
    (ex:firstName x (xsd:string 'Jack'))
    (rdf:type x ex:Human)
    (Married x y)
    (ex:firstName y (xsd:string 'Jill'))
))
```

The RDF vocabularies for reification, containers and values have no special semantic conditions, so translate uniformly into CL using the above conversion methods.

21.2.4 RDF Lists

[SCL Translation] includes a discussion relevant to both RDFS and OWL ontologies regarding the mapping of lists that represent relations between multiple arguments to CL. Since RDF triple syntax can directly express only unary and binary relations, relations of higher arity must be encoded, and OWL in particular uses lists to do this encoding. Axioms for translating such lists, derived from [Fikes & McGuinness], are provided in [SCL Translation] and are incorporated herein by reference.

21.2.5 RDF Schema

As discussed in [SCL Translation], RDF Schema extends RDF through semantic constraints that impose additional meaning on the RDFS vocabulary. In particular, it gives a special interpretation to *rdf:type* as being a relationship between a 'thing' and a 'class', which approximates the set-membership relationship in set theory. This relationship is captured in several axioms, repeated here due to their importance with regard to streamlining the mapping.

RDFS Class Axiom

```
(forall (x y)(iff (rdf:type x y) (and (rdfs:Class y)(y x))))
```

RDFS Promiscuity Axiom

```
(forall (x) (rdfs:Class x))
```

RDFS Universal Resource Axiom

```
(forall (x) (rdfs:Resource x))
```

Taken together, these axioms justify the more efficient mapping of RDFS triples to CL given in Table 48, to be used in place of Table 46.

Table 48 RDFS Triple to CL Mapping

RDFS Metamodel Element	RDFS Metamodel Property	CL Metamodel Element	CL Metamodel Property
(1) RDFStatement	RDFsubject *(aaa)* RDFpredicate *(rdf:type)* RDFobject *(bbb)*	Relation	`predicate:` *bbb* `arguments [1]:` *aaa*
(2) RDFStatement, (any other triple)	RDFsubject *(aaa)* RDFpredicate *(ppp)* RDFobject *(bbb)*	Relation	`predicate:` *ppp* `arguments [1]:` *aaa* `arguments [2]:` *bbb*

The translations are ordered, with the second used only when the first does not apply.

The above example now translates into the more intuitive form

```
(exists (x y)
    (and
    (ex:firstName x (xsd:string 'Jack'))
    (ex:Human x)
    (Married x y)
    (ex:firstName y (xsd:string 'Jill'))
))
```

where `ex:Human` is a genuine predicate.

Similarly to the case for RDF, this assumes that *every* unary predicate corresponds to an RDFS class; to be more cautious, one would omit the promiscuity axiom and insert an extra assumption explicitly as part of the translation process: if (1), add axiom `(rdfs:Class bbb)`; otherwise, (2) add axiom: `(rdf:Property ppp)`.

21.2.6 RDFS Semantics

In [RDF Semantics], several of the constraints are expressed as RDFS assertions ("axiomatic triples"), but others are too complex to be represented in RDFS and so must be stated explicitly as external model-theoretic constraints on RDFS interpretations. All of these can be expressed directly as CL axioms, however. An CL encoding of RDFS is obtained by following the translation rules and adding a larger set of axioms. RDFS interpretations of a graph can be identified with CL interpretations of the translation of the graph with the RDF and RDFS axioms added.

A series of tables encoding numerous axioms is provided in [SCL Translation] which reflect the axiomatic triples, RDFS "semantic conditions", and extensional axioms, as well as axioms for interpreting datatypes, which are incorporated herein by reference. Some of these are summarized in an RDFS extensional logical form translation table, which may be more efficient than deriving the translation from the embedding and axioms. These are provided in Table 49, mapped to the appropriate metamodel elements.

Where possible clauses included in sentences, such as the antecedent and consequent of an implication, are expanded for further clarification. The translations are ordered, with the final one used only when the others do not apply.

Table 49 RDFS Extensional Logical Form Translation

RDFS Metamodel Element	RDFS Metamodel Property	CL Metamodel Element	CL Metamodel Property	'Cautious' Axiom(s)
RDFStatement	RDFsubject *(aaa)* RDFpredicate (*rdf:type)* RDFobject *(bbb)*	Relation	`predicate:` *bbb* `arguments [1]:` *aaa*	`(rdfs:Class bbb)`
RDFStatement	RDFsubject *(aaa)* RDFpredicate (*rdfs:domain)* RDFobject *(bbb)*	UniversalQuantification	`Binding: Term:` *u* `Term:` *y*	`(rdfs:Class bbb)` `(rdf:Property aaa)`
		Implication	`antecedent: (`*aaa u y*`)`[3] `consequent: (`*bbb u*`)`	
		[3]Relation	`predicate:` *aaa* `arguments [1]:` *u* `arguments [2]:` *y*	
		Relation	`predicate:` *bbb* `arguments [1]:` *u*	
RDFStatement	RDFsubject *(aaa)* RDFpredicate (*rdfs:range)* RDFobject *(bbb)*	UniversalQuantification	`Binding: Term:` *u* `Term:` *y*	`(rdfs:Class bbb)` `(rdf:Property aaa)`
		Implication	`antecedent: (`*aaa u y*`)` `consequent: (`*bbb y*`)`	
		Relation	`predicate:` *aaa* `arguments [1]:` *u* `arguments [2]:` *y*	
		Relation	`predicate:` *bbb* `arguments [1]:` *y*	
RDFStatement	RDFsubject *(aaa)* RDFpredicate (*rdfs:subClassOf)* RDFobject *(bbb)*	UniversalQuantification	`Binding: Term:` *u*	`(rdfs:Class bbb)` `(rdfs:Class aaa)`
		Implication	`antecedent: (`*aaa u*`)` `consequent: (`*bbb u*`)`	
		Relation	`predicate:`*aaa* `arguments [1]:` *u*	
		Relation	`predicate:` *bbb* `arguments [1]:` *u*	

Table 49 RDFS Extensional Logical Form Translation

RDFStatement	RDFsubject *(aaa)* RDFpredicate (*rdfs:subPropertyOf)* RDFobject *(bbb)*	UniversalQuantification	`Binding: Term:` *u* `Term:` *y*	`(rdf:Property bbb)` `(rdf:Property aaa)`
		Implication	`antecedent: (`*aaa u y*`)` `consequent: (`*bbb u y*`)`	
		Relation	`predicate:` *aaa* `arguments [1]:` *u* `arguments [2]:` *y*	
		Relation	`predicate:` *bbb* `arguments [1]:` *u* `arguments [2]:` *y*	
RDFStatement (any other triple)	RDFsubject *(aaa)* RDFpredicate *(ppp)* RDFobject *(bbb)*	Relation	`predicate:` *ppp* `arguments [1]:` *aaa* `arguments [2]:` *bbb*	`(rdf:Property ppp)`

21.3 OWL to CL Mapping

As described in the relevant specifications, the Web Ontology Language (OWL) is actually three closely related dialects rather than a single language, which share a common set of basic definitions but differ in scope and by the degree to which their syntactic forms are restricted. The OWL metamodel given in Chapter 12 of this specification is intended to represent the abstract syntax for OWL Full, but can also represent the abstract syntax for OWL DL, as long as restrictions to support the more constrained semantics of OWL DL are applied.

The discussion provided in [SCL Translation] provides additional insight into the variations among OWL dialects. It then provides an unrestricted translation from the OWL vocabulary to CL, and further refines it for each dialect given a common starting point. There are a number of important considerations provided in that discussion, including a series of axioms applicable to any CL reasoning environment designed to support OWL ontologies as input.

Table 50 provides a summary translation from RDFS/OWL triples, as represented in the metamodel triple constructs, mapped to the appropriate high-level CL metamodel sentence constructs. We've taken this approach in keeping with the translation, but also due to the fact that what is mapped in some cases is actually a subgraph consisting of multiple RDFS/OWL statements as well as for increased clarity. Further refinement of some of the CL sentences will be accomplished during the finalization phase of the specification, along with inclusion of examples. The translation assumes the axioms stated in Section 21.1 and Section 21.2, as well as the following identity axioms:

```
(forall ((x owl:Thing)(y owl:Thing))(iff (owl:differentFrom x y)(not (= x y)) ))

(forall ((x owl:Thing))(not (owl:Nothing x)))
```

Note that OWL assertions involving annotation and ontology properties are not covered explicitly, and should be simply transcribed as atomic assertions in CL, using the same mechanisms described for RDF triples.

To use the table below to translate an OWL/RDF graph, simply generate the corresponding CL for every subgraph that matches the pattern specified in the leftmost two columns. The notation `ALLDIFFERENT` is used as a shorthand for conjunction of n(n-1) "inequations" which assert that the terms are all distinct:

```
[ALLDIFFERENT x1 ... xn]
```

means:

```
(and
      (not (= x1 x2)) (not (=x1 x3)) ... (not (= x1 xn))
      (not (= x2 x3)) ... (not (= x2 xn))
      (not (= x3 xn)) ...
      ...
      (not (= xn-1 xn))
)
```

Note that the negation of this is a disjunction of equations. *owl_Property* should be read as shorthand for the union of *owl:DatatypeProperty* and *owl:ObjectProperty.*

Unlike the RDFS translation, this translates entire RDF subgraphs into logical sentences. To achieve a full translation, *all* matching subgraphs must be translated, and then any remaining triples rendered into logical atoms using the RDF translation. Note that a triple in the graph may occur in more than one subgraph; in particular, the *owl:onProperty* triples will often occur in several subgraph patterns when cardinality and value restrictions are used together.

Table 50 RDFS/OWL to CL Metamodel Translation

RDFS/OWL Metamodel Element	RDFS/OWL Metamodel Property	CL Metamodel Element	CL Metamodel Property	Assumption(s)
Subgraph: RDFStatement RDFStatement	 RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)* RDFsubject *(rrr)* RDFpredicate *(owl:minCardinality)* RDFobject *(n)*	UniversalQuantification Implication	Binding: (Term: *(x owl:Thing))* antecedent: *(rrr x)* consequent: *(exists ((*x_1 *owl:Thing) ... (*x_n *owl:Thing))* *(and* *[ALLDIFFERENT* x_1 *...* x_n*]* *(ppp x* x_1*) ...(ppp x* x_n*)* *))*	*(owl:Restriction rrr)* *(rdf:Property ppp)*
subgraph: RDFStatement RDFStatement	 RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)* RDFsubject *(rrr)* RDFpredicate *(owl:maxCardinality)* RDFobject *(n)*	UniversalQuantification Implication	Binding: *(* *Term:(x owl:Thing)* *Term:(*x_1 *owl:Thing)* *...* *Term: (*x_{n+1} *owl:Thing)* *)* antecedent: *(and (rrr x)* *(ppp x* x_1*) ...(ppp x* x_{n+1}*)* *)* consequent: *(not [ALLDIFFERENT* x_1 *...* x_{n+1}*])*	*(owl:Restriction rrr)* *(rdf:Property ppp)*
subgraph RDFStatement RDFStatement	 RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)* RDFsubject *(rrr)* RDFpredicate (*owl:cardinality)* RDFobject *(n)*	UniversalQuantification Implication	Binding: (Term: *(x owl:Thing))* antecedent: *(rrr x)* consequent: *(exists ((*x_1 *owl:Thing) ... (*x_n *owl:Thing))* *(and* *[ALLDIFFERENT* x_1 *...* x_n*]* *(ppp x* x_1*) ...(ppp x* x_n*)* *(forall ((z owl:Thing))(implies* *(ppp x z)* *(or (= z* x_1*) ... (= z* x_n*))* *))* *))*	*(owl:Restriction rrr)* *(rdf:Property ppp)*

Table 50 RDFS/OWL to CL Metamodel Translation

subgraph RDFStatement	RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)*	UniversalQuantification Equivalence	`Binding: (Term:` *`(x owl:Thing))`* `lvalue:` *`(rrr x)`* `rvalue:` *`(forall (y)(implies (ppp x y)(ccc y)))`*	*`(owl:Restriction rrr)`* *`(rdf:Property ppp)`*
RDFStatement	RDFsubject *(rrr)* RDFpredicate (*owl:allValuesFrom)* RDFobject *(ccc)*			
subgraph RDFStatement	RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)*	UniversalQuantification Equivalence	`Binding: (Term: (`*`x owl:Thing))`* `lvalue:` *`(rrr x)`* `rvalue:` *`(exists (y)(and (ppp x y)(ccc y)))`*	*`(owl:Restriction rrr)`* *`(rdf:Property ppp)`*
RDFStatement	RDFsubject *(rrr)* RDFpredicate *(owl:someValuesFrom)* RDFobject *(ccc)*			
subgraph RDFStatement	RDFsubject *(rrr)* RDFpredicate *(owl:onProperty)* RDFobject *(ppp)*	UniversalQuantification Equivalence	`Binding: (Term:` *`(x owl:Thing))`* `lvalue:` *`(rrr x)`* `rvalue:` *`(ppp x vvv)`*	*`(owl:Restriction rrr)`* *`(rdf:Property ppp)`*
RDFStatement	RDFsubject *(rrr)* RDFpredicate *(owl:hasValue)* RDFobject *(vvv)*			

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate *(rdf:type)*
	RDFobject *(owl:FunctionalProperty)*
		Conjunction	`UniversalQuantification:  Binding: (`
			`    Term:  (x owl:Thing)`
			`    Term:  (y owl:Thing)`
			`    Term:  (z owl:Thing)`
			`)`
			
			`Implication: (`
			`    antecedent:  (and (ppp x y)(ppp x z))`
			`    consequent:  (= y z)`
			`)`
				(owl_Property ppp)

	-or-	UniversalQuantification	`Binding: (`
			`    Term:  (x owl:Thing)`
			`    Term:  (y rdfs:Literal)`
			`    Term:  (z rdfs:Literal)`
			`)`	

		Implication	`antecedent:  (and (ppp x y)(ppp x z))` `consequent:  (= y z)`	

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate *(rdf:type)*
	RDFobject*(owl:InverseFunctionalProperty)*
		UniversalQuantification	`Binding: (`
			`    Term:  (x owl:Thing)`
			`    Term:  (y owl:Thing)`
			`    Term:  (z owl:Thing)`
			`)`
				(owl:ObjectProperty ppp)

		Implication	`antecedent:  (and (ppp y x)(ppp z x))` `consequent:  (= y z)`	

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate *(rdf:type)*
	RDFobject*(owl:SymmetricProperty)*
		UniversalQuantification	`Binding: (`
			`    Term:  (x owl:Thing)`
			`    Term: (y owl:Thing)`
			`)`
				(owl:ObjectProperty ppp)

		Implication	`antecedent:  (ppp x y)` `consequent:  (ppp y x)`	

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate *(rdf:type)*
	RDFobject *(owl:TransitiveProperty)*
		UniversalQuantification	`Binding: (`
			`    Term:  (x owl:Thing)`
			`    Term:  (y owl:Thing)`
			`    Term:   (z owl:Thing)`
			`)`
				(owl:ObjectProperty ppp)

		Implication	`antecedent:  (and (ppp x y)(ppp y z))` `consequent:  (ppp x z)`	

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate*(owl:equivalentProperty)*
	RDFobject *(qqq)*
		UniversalQuantification
		
		Implication
			Binding: (
			    Term: *(x owl:Thing)*
			    Term: *(y owl:Thing)*
			)
			
			antecedent: *(ppp x y)*
			consequent: *(qqq x y)*
				(owl_Property ppp)
				(owl_Property qqq)

RDFStatement	RDFsubject *(ppp)*
	RDFpredicate *(owl:inverseOf)*
	RDFobject *(qqq)*
		UniversalQuantification
		
		Implication
			Binding: (
			    Term: *(x owl:Thing)*
			    Term: *(y owl:Thing)*
			)
			
			antecedent: *(ppp x y)*
			consequent: *(qqq y x)*
				(owl_Property ppp)
				(owl_Property qqq)

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(owl:equivalentClass)* RDFobject *(ddd)*	UniversalQuantification Implication	Binding: (Term: *(x owl:Thing)*) antecedent: *(ccc x)* consequent: *(ddd x)*	*(owl:Class ccc)* *(owl:Class ddd)*

RDFStatement	RDFsubject *(ccc)*
	RDFpredicate *(owl:disjointWith)*
	RDFobject *(ddd)*
		UniversalQuantification
		
		Negation
			Binding: (Term: *(x owl:Thing)*)
			
			Sentence: (
			    Conjunction: (
			        Sentence: *(ccc x)*
			        Sentence: *(ddd x)*
			))
				(owl:Class ccc)
				(owl:Class ddd)

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(owl:complementOf)* RDFobject *(ddd)*	UniversalQuantification Implication	Binding: (Term: *(x owl:Thing)*) antecedent: *(ccc x)* consequent: *(not(ddd x))*	*(owl:Class ccc)* *(owl:Class ddd)*

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ccc)*
	RDFpredicate *(owl:intersectionOf)*
	RDFobject *(lll-1)*
		UniversalQuantification
		
		Implication
			`Binding: (Term: (x owl:Thing))`
			
			`antecedent: (ccc x)`
			`consequent: (and (aaa-1 x) ... (aaa-n x) )`
				`(owl:Class ccc)`

RDFStatement	RDFsubject *(lll-1)* RDFpredicate *(rdf:first)* RDFobject *(aaa-1)* RDFsubject *(lll-1)* RDFpredicate *(rdf:rest)* RDFobject *(lll-2)*			
…	…			
RDFStatement	RDFsubject *(lll-n)* RDFpredicate *(rdf:first)* RDFobject *(aaa-n)* RDFsubject *(lll-n)* RDFpredicate *(rdf:rest)* RDFobject *(rdf:nil)*			

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(owl:unionOf)* RDFobject *(lll-1)*	UniversalQuantification Implication	`Binding: (Term:` *`(x owl:Thing))`* `antecedent:` *`(ccc x)`* `consequent:` *`(or (aaa-1 x) ... (aaa-n x) )`*	*`(owl:Class ccc)`*
RDFStatement	RDFsubject *(lll-1)* RDFpredicate *(rdf:first)* RDFobject *(aaa-1)* RDFsubject *(lll-1)* RDFpredicate *(rdf:rest)* RDFobject *(lll-2)*			
…	…			
RDFStatement	RDFsubject *(lll-n)* RDFpredicate *(rdf:first)* RDFobject *(aaa-n)* RDFsubject *(lll-n)* RDFpredicate *(rdf:rest)* RDFobject *(rdf:nil)*			

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(owl:oneOf)* RDFobject *(lll-1)*	UniversalQuantification Implication	`Binding: (Term:` *`(x owl:Thing))`* `antecedent:` *`(ccc x)`* `consequent:` *`(or (= aaa-1 x) ... (= aaa-n x) )`*	*`(owl:Class ccc)`*
RDFStatement	RDFsubject *(ccc)* RDFpredicate *(rdf:type)* RDFobject *(owl:Class)*			
RDFStatement	RDFsubject *(lll-1)* RDFpredicate *(rdf:first)* RDFobject *(aaa-1)* RDFsubject *(lll-1)* RDFpredicate *(rdf:rest)* RDFobject *(lll-2)*			
…	…			
RDFStatement	RDFsubject *(lll-n)* RDFpredicate *(rdf:first)* RDFobject *(aaa-n)* RDFsubject *(lll-n)* RDFpredicate *(rdf:rest)* RDFobject *(rdf:nil)*			

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(owl:oneOf)* RDFobject *(lll-1)*	UniversalQuantification Implication	`Binding: (Term:` *`(x rdfs:Literal))`* `antecedent:` *`(ccc x)`* `consequent:` *`(or (= aaa-1 x) ... (= aaa-n x) )`*	*`(owl:DataRange ccc)`*
RDFStatement	RDFsubject *(ccc)* RDFpredicate *(rdf:type)* RDFobject *(owl:DataRange)*			
RDFStatement	RDFsubject *(lll-1)* RDFpredicate *(rdf:first)* RDFobject *(aaa-1)* RDFsubject *(lll-1)* RDFpredicate *(rdf:rest)* RDFobject *(lll-2)*			
…	…			
RDFStatement	RDFsubject *(lll-n)* RDFpredicate *(rdf:first)* RDFobject *(aaa-n)* RDFsubject *(lll-n)* RDFpredicate *(rdf:rest)* RDFobject *(rdf:nil)*			

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(ccc)* RDFpredicate *(rdf:type)* RDFobject *(owl:AllDifferent)*	UniversalQuantification Implication	`Binding: (Term: ` *(x owl:Thing))* `antecedent: ` *(ccc x)* `consequent: ` *(or (= aaa-1 x) ... (= aaa-n x))*	*(owl:Class ccc)*
RDFStatement	RDFsubject *(ccc)* RDFpredicate*(owl:distinctMembers)* RDFobject *(lll-1)*	Sentence	`[ALLDIFFERENT aaa-1 ... aaa-n]`	
RDFStatement	RDFsubject *(lll-1)* RDFpredicate *(rdf:first)* RDFobject *(aaa-1)* RDFsubject *(lll-1)* RDFpredicate *(rdf:rest)* RDFobject *(lll-2)*			
…	…			
RDFStatement	RDFsubject *(lll-n)* RDFpredicate *(rdf:first)* RDFobject *(aaa-n)* RDFsubject *(lll-n)* RDFpredicate *(rdf:rest)* RDFobject *(rdf:nil)*			
RDFStatement	RDFsubject *(aaa)* RDFpredicate *(rdf:type)* RDFobject *(bbb)*	Relation	`predicate: ` *bbb* `arguments [1]: ` *aaa*	*(owl:Class bbb)*

RDFStatement	RDFsubject *(aaa)*
	RDFpredicate *(rdfs:domain)*
	RDFobject *(bbb)*
		UniversalQuantification
		
		Implication
			`Binding: (`
			`Term: ` *(u rdfs:Resource)*
			`Term: ` *(y rdfs:Resource)*
			`)`
			
			`antecedent: ` *(aaa u y)*
			`consequent: ` *(bbb u)*
				(owl:Class bbb)
				(rdf:Property aaa)

Table 50 RDFS/OWL to CL Metamodel Translation

RDFStatement	RDFsubject *(aaa)*
	RDFpredicate *(rdfs:range)*
	RDFobject *(bbb)*
		UniversalQuantification	`Binding: (`
			`Term:` *(u rdfs:Resource)*
			`Term:` *(y rdfs:Resource)*
			`)`
				(owl:Class bbb)
				(rdf:Property aaa)

		Implication	`antecedent:` *(aaa u y)* `consequent:` *(bbb y)*	
RDFStatement	RDFsubject *(aaa)* RDFpredicate *(rdfs:subClassOf)* RDFobject *(bbb)*	UniversalQuantification	`Binding: Term:` *(u rdfs:Resource)*	*(owl:Class bbb)* *(owl:Class aaa)*
		Implication	`antecedent:` *(aaa u)* `consequent:` *(bbb u)*	

RDFStatement	RDFsubject *(aaa)*
	RDFpredicate*(rdfs:subPropertyOf)*
	RDFobject *(bbb)*
		UniversalQuantification	`Binding: (`
			`Term:` *(u rdfs:Resource)*
			`Term:` *(y rdfs:Resource)*
			`)`
				(rdf:Property bbb)
				(rdf:Property aaa)

		Implication	`antecedent:` *(aaa u y)* `consequent:` *(bbb u y)*	
any other triple RDFStatement	RDFsubject *(aaa)* RDFpredicate *(ppp)* RDFobject *(bbb)*	Relation	`predicate:` *ppp* `arguments [1]:` *aaa* `arguments [2]:` *bbb*	*(rdf:Property ppp)*

In addition, depending on the dialect of OWL (OWL DL or OWL Full) in question, certain hierarchical axioms are assumed, which enforce the distinction between owl:ObjectProperty and owl:DatatypeProperty, for example. For OWL DL, they also enforce the strict segregation between classes, properties, and individuals. These are summarized below for comparison purposes.

OWL Hierarchy Axioms

```
(forall ((x owl:Thing)(y owl:Thing))(iff (owl:sameAs x y)(= x y) ))
(forall (x) (implies (rdfs:Class x) (rdfs:Resource x))
(forall (x) (implies (rdf:Property x) (rdfs:Resource x))
(forall (x) (implies (rdfs:Datatype x) (rdfs:Class x))
(forall (x) (implies (owl:Thing x) (rdfs:Resource x))
(forall (x) (implies (owl_Property x) (rdf:Property x))
(forall (x) (implies (owl:Class x) (rdfs:Class x))
(forall (x) (implies (owl:DataRange x) (rdfs:Class x))
(forall (x) (implies (owl:Restriction x) (owl:Class x))
(forall (x) (implies (owl:ObjectProperty x) (owl_Property x))
(forall (x) (implies (owl:DatatypeProperty x) (owl_Property x))
(forall (x) (implies (owl:Thing x) (rdfs:Resource x))
(forall (x) (not (and (owl:Thing x)(rdfs:Literal x))))
(forall (x) (not (and (owl:Thing x)(owl:Ontology x))))
(forall (x) (not (and (owl:ObjectProperty x)(owl:DatatypeProperty x))))
```

OWL-DL Specific Hierarchy Axioms

```
(forall (x) (not (and (owl:Thing x)(owl_Property x))))
(forall (x) (not (and (owl:Thing x)(owl:Class x))))
(forall (x) (not (and (owl:Class x)(owl_Property x))))
(forall (x) (not (and (owl:OntologyProperty x)(owl_Property x))))
(forall (x) (not (and (owl:AnnotationProperty x)(owl_Property x))))
```

22 References (non-normative)

[BCMNP] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider editors;*The Description Logic Handbook: Theory, Implementation and Applications;* Cambridge University Press, January 2003

[CGS] Sowa, J.F., *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, MA, 1984.

[DOLCE] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider, Sweetening Ontologies with DOLCE, 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), 1-4 October 2002, Siguenza, Spain.

[Fikes & McGuinness] *An Axiomatic Semantics for RDF, RDF Schema and DAML+OIL* <http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics-august2001.html> Fikes, R., McGuinness, D. L., KSL Technical Report KSL-01-01, August 2001.

[GE] J. Paul, S. Withanachchi, R. Mockler, M. Gartenfeld, W. Bistline and D. Dologite, Enabling B2B Marketplaces: the case of GE Global Exchange Services, in *Annals Of Cases On Information Technology*, Hershey, PA : Idea Group, 2003

[GuarWel] N. Guarino and C. Welty, Identity, Unity and Individuality: Towards a Formal Toolkit for Ontological Analysis, in: W. Horn (ed) Proceedings of ECAI-2000: The European Conference on Artificial Intelligence IOS Press, Amsterdam, 2000.

[KIF] M. R. Genesereth & R. E. Fikes, Knowledge Interchange Format, Version 3.0 Reference Manual. KSL Report KSL-92-86, Knowledge Systems Laboratory, Stanford University, June 1992.

[MSDW] R. Colomb, A. Gerber and M. Lawley, Issues in Mapping Metamodels in the Ontology Definition Metamodel, 1st International Workshop on the Model-Driven Semantic Web (MSDW 2004) Monterey, California, USA. 20-24 September, 2004.

[ODM RFP] Ontology Definition Metamodel Request for Proposal, OMG Document ad/2003-03-40.

[OntoClean] N. Guarino and C. Welty. Evaluating Ontological Decisions with OntoClean, Communications of the ACM, 45(2) (2002) 61-65.

[OWL OV] OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004. Deborah L. McGuinness and Frank van Harmelen eds. Latest version is available at: http://www.w3.org/TR/owl-features/.

[OWL Reference] OWL Web Ontology Language Reference. W3C Recommendation 10 February 2004, Mike Dean, Guus Schreiber, eds. Latest version is available at http://www.w3.org/TR/owl-ref/.

[OWL XML Syntax] OWL Web Ontology Language XML Presentation Syntax. Masahiro Hori, Jérôme Euzenat, and Peter F. Patel-Schneider, Editors. W3C Note, 11 June 2003. Latest version is available at http://www.w3.org/TR/owl-xmlsyntax/.

[QVT] MOF 2.0 Query/View/Transformation. Second Revised Submission to MOF 2.0 QVT RFP, ad/2002-04-10. Latest version is available at http://www.omg.org/docs/ad/05-03-02.pdf.

[RDF/TM] W3C A survey of RDF/Topic Maps Interoperability Proposals W3C Working Draft 29 March, 2005. Latest version is available at http://www.w3.org/TR/2005/WD-rdftm-survey-20050329.

[Rose] IBM Rational Rose, http://www-130.ibm.com/developerworks/rational/products/rose.

[SCL Translation] *Translating Semantic Web Languages into SCL*, Patrick Hayes, IHMC, November 2004. Latest version available at http://www.ihmc.us/users/phayes/CL/SW2SCL.html.

[WinChafHerr] M. Winston, R. Chaffin, and D. Herrmann, (1987). A taxonomy of part-whole relations. Cognitive Science 11, 417-444.

A Foundation Ontology (M1) for RDFS and OWL

As noted in the Design Rationale, it is impossible to completely capture RDF, RDF Schema, and OWL semantics in an M2-level MOF metamodel. RDF, RDF Schema, and OWL do not make clear distinction between M3, M2 and M1 objects (in MOF terms). By design choice, all RDF, RDF Schema, and OWL constructs have been modeled as M2 objects.

An M1 instance of either the RDFS or OWL metamodels will generally include some built-in resources as M1 instances of some M2 classes. Table 51 gives a foundation ontology containing some of the more significant such resources. An M1 instance of an RDFS or OWL ontology will generally wish to include these resources..

Table 51 Foundation Ontology (M1) for RDFS and OWL

M1 Object	Properties
owl:Nothing	RDFSsubclassOf every instance of OWLClass
owl:Thing	Every instance of OWLClass is RDFSsubclassOf owl:Thing. Default domain and range of every instance of OWLObjectProperty. Default domain of every instance of OWLDatatypeProperty. Every MOF instance of Individual is directly or indirectly of rdf:type owl:Thing.
rdf:nil	Special instance of RDFList indicating termination of an RDFList.
XML Schema built-in datatypes (see Section 11)	xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger,xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, and xsd:positiveInteger

B A Description Logic Metamodel

This appendix provides an introduction to Description Logics through the elaboration of a exemplar Description Logic Meta-Model.

B.1 Introduction

The Description Logic (DL) meta-model defines a basic, minimally constrained DL. In use, DLs are typically found in the Knowledge-Base of a Knowledge Representation System, as shown in Figure 80 .

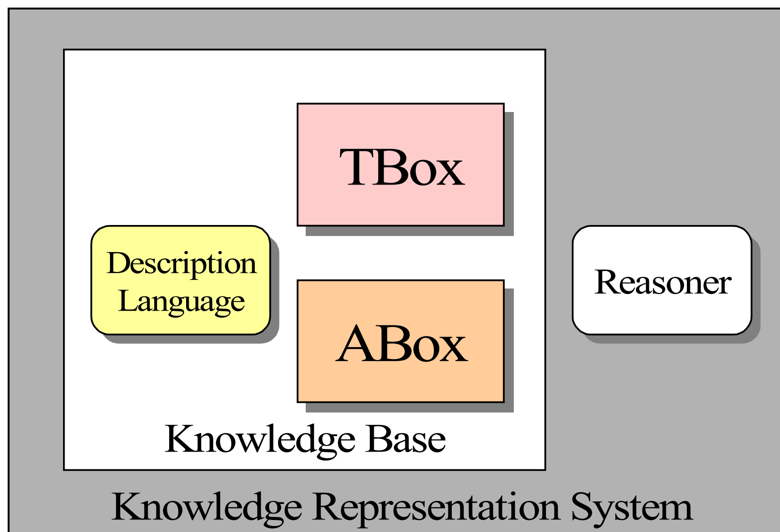


Figure 80 Knowledge Representation System

A DL Knowledge Base is traditionally divided into three principal parts:

- Terminology or schema, the vocabulary of application domain, called the “TBox”,
- Assertions, which are named individuals expressed in terms of the vocabulary, called the “ABox” and
- Description Language that define terms and operators for build expressions.

Note that the TBox and ABox elements represent two separate meta-levels in the application domain.

B.2 Containers

Basic containment constructs of this DL meta-model, as shown in Figure 81 , are provided through the TBox and ABox elements, which correspond directly to the TBox and ABox concepts from description logics.

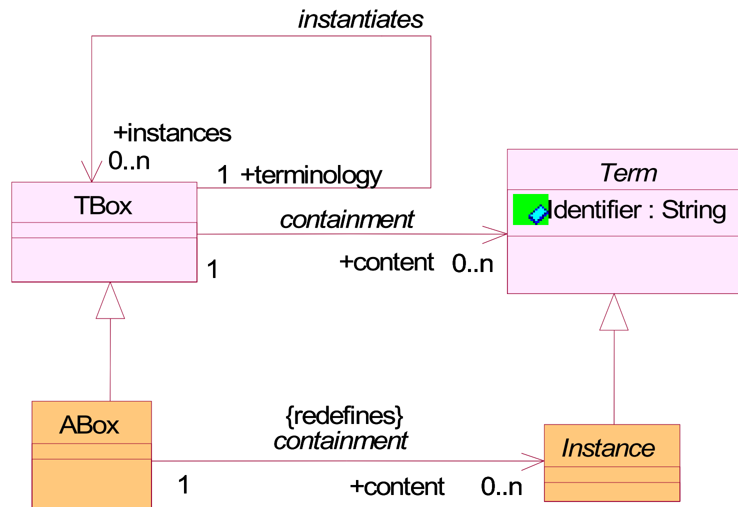


Figure 81 Basic Containment Constructs

B.2.1 TBox

Description

A TBox contains all of a DL model's terminology. The TBox may include Terms and any of the sub-classes of Term. Note that this includes Instances to allow supporting predefined, delineated instances as 'special terms' in the ABox. An example of this would be OWL Thing.

Associations

- Containment.content[0..n]: Term – the terminology contained in this TBox.
- instances[0..n]: ABox – the TBox that uses terms and instances from this TBox.
- terminology[1]: TBox – the TBox that contains the terminology used by this TBox (or ABox)

B.2.2 ABox

Description

An ABox contains all of a DL model's instances. The ABox extends TBox and restricts its content to be only the sub-classes Instance.

Associations

- Containment.content[0..n]: Instance – the instances contained in this ABox, redefining containment from TBox.

Semantics

All the instances in an ABox are expressed using the terminology from exactly one TBox.

B.3 Concepts and Roles

B.3.1 Element

Description

Element is an abstract base class of all atomic components in a DL as seen in Figure 82 . It defines the notion of unique identity so that references may be made to elements using that identifier.

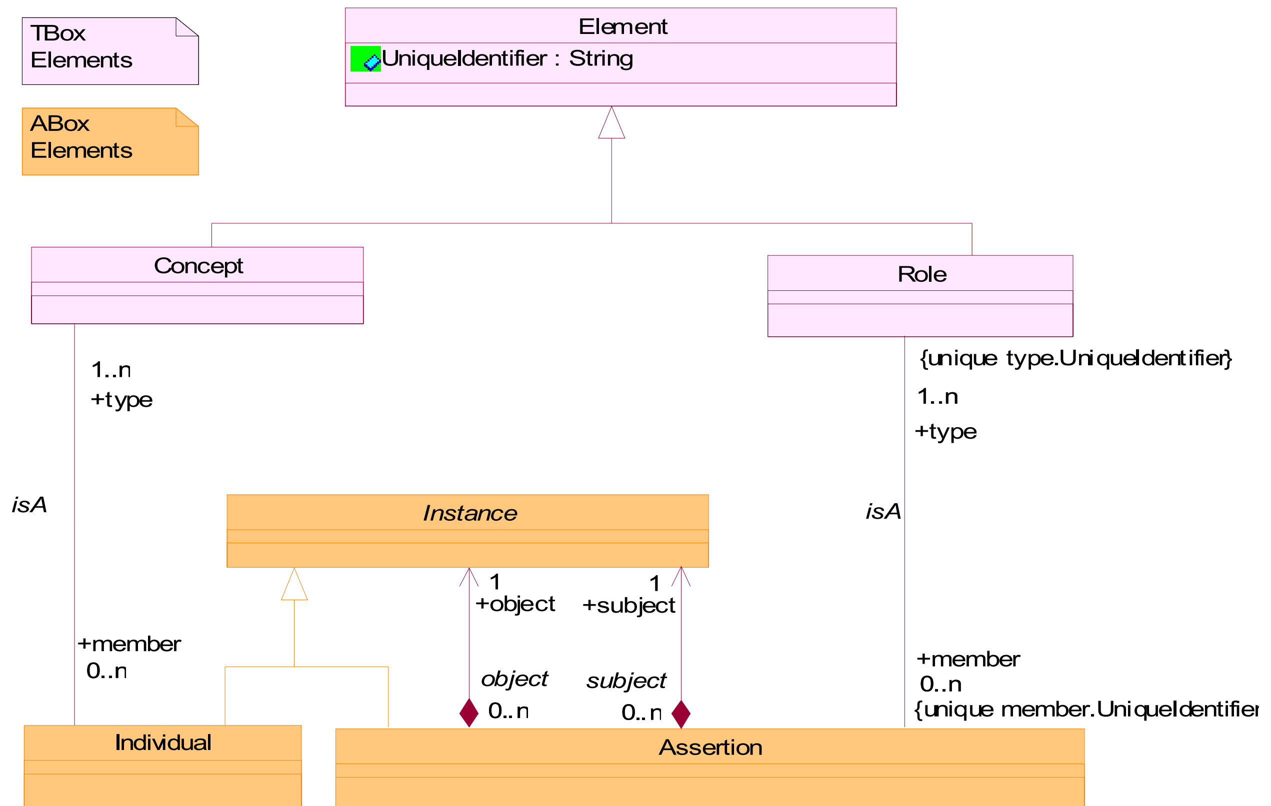


Figure 82 Element Model

Attributes

- UniqueIdentifier: String - Uniquely identifies a Element and all Elements are identified by a single value. That is if UniqueIdentifiers of two Elements are different, then the Elements are different. Note that this is different than URIs. UniqueIdentifier is required.

B.3.2 Concept

Description

Concept is a set of Instances which are define as having something in common.

Concept is a specialization of Element.

Similar Terms

Class, Entity, Topic, Type

Associations

- isA.member: Individual[0..n] -- The set of Individuals that are the extent of the concept.

B.3.3 Instance

Description

Instance provides an abstract base class for all ABox constructs. Instance is a specialization of Term.

Similar Terms

Object, Instantiation

B.3.4 Role

Description

A Role is a set of binary tuples, specifically (subject, object), that asset that this role for subject is satisfied by object. Role is a specialization of Element.

Similar Terms

Association, Attribute, Property, Slot

Associations

- isA.member: Assertion [0..n] -- The set of Assertions that are the extent of the concept.

B.3.5 Individual

Description

Individual is an instance of a Concept. An Individual is a specialization of Instance

Similar Terms

Object

Associations

- isA.type: Concept[1..n] – The set of concept sets that has this individual as a member.

B.3.6 Assertion

Description

Assertions are the specific binary tuples that are instances of Roles. An Assertion is a specialization of Instance.

Similar Terms

Link, Statement, Fact

Associations

- subject.subject: Instance[1] – The Instance that is the subject of the assertion.
- object.object: Instance[1] – The Instance that is the object of the assertion.

- sA.type: Concept[1..n] – The set of Roles that has this assertion as a member.
- predicate: Instance[1] – A derived reference to the Role which this assertion is an instance of. (Not shown in diagram.)

B.4 Datatypes

B.4.1 Datatype

Description

A Datatype is a specialization of Concept. Datatypes are those concepts whose members have no identity except their value, that is the members of a datatype are literals, as shown in Figure 83 . Datatype may represent primitive types, for example integer, string, or boolean; or user defined type, for example time-interval or length-in-meters.

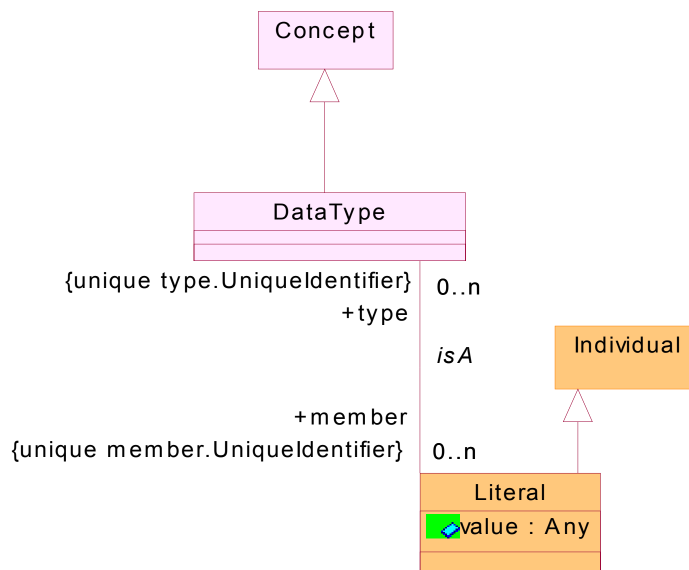


Figure 83 DataType Model

Associations

- isA.member: Literal[0..n] – the set of literals that are members of this datatype.

B.4.2 Literal

Description

Literals are the specification of instances of datatypes. The UniqueIdentifier inherited from Element is for a literal, uniquely defined by the literal's value itself.

Attributes

- value: Any – The implementation and Datatype dependent value of this literal.

Associations

- type: Datatype[0..n] – the possibly empty set of datatypes in which this literal is a member.

Constraints

Restricts range of Concept.type to a set of Datatypes.

Semantics

Element.UniqueIdentifier has a functional relation with Literal.value.

Literal.value has a functional relation with Element.UniqueIdentifier.

B.5 Collections

B.5.1 Collection

Description

A Collection is a specialization of Concept. Collection allows instances to be brought together as a group and referenced as a single collective. The class diagram for Collection is shown in Figure 84 .

Collection is conceptually a 'bag', that is un-order and allowing duplicate members..

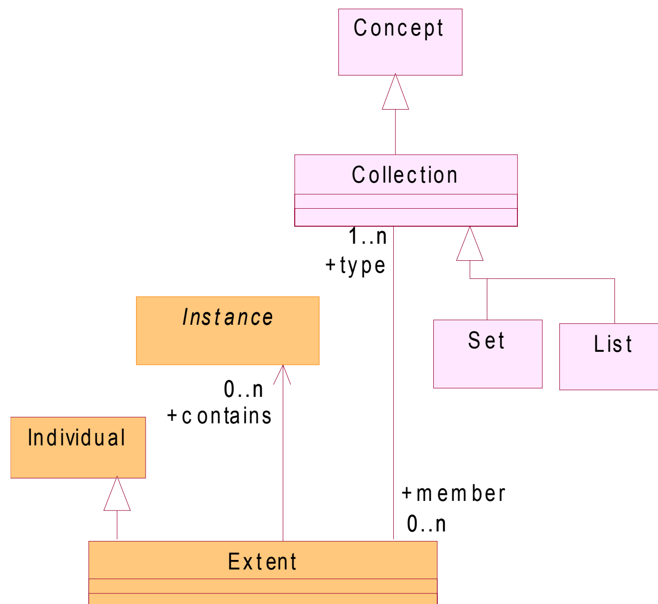


Figure 84 Collection Model

Similar Terms

Container; and Sequence, List, Bag, Set as specific types.

Associations

- isA.member: Extent[0..n] – The set of instances of a particular kind of collection.

B.5.2 List

Description

List is a specialization of Collection. List requires that the member instances that are in the collection are ordered in a user defined way.

Semantics

For all a_i , a_j members of the list, there is a comparator function C() such that $C(a_i) < C(a_j)$ if $i < j$

B.5.3 Set

Description

Set is a specialization of Collection. Set requires that the member instances in the collection are unique.

Semantics

For all a_i , a_j members of the list, there is a identity function I() such that $I(a_i)=I(a_j)$ iff $i = j$

B.5.4 Extent

Description

Extent is a specialization of Instance. Extent is the set of all instances of a collection of a particular type, for example the set of all Alphabetical-Lists.

Associations

- containment.contains: Instance[0..n] – Those instances that are in this instance of a collection.
- isA.type: Collection - The set of collection sets that has this extent as a member.

B.6 Expressions and Constructors

Expressions provide the mechanism for constructing class definitions and implications about TBox elements. They provide a hook for more expressive constraint and rule languages.

A number of common expression constructors, shown in Figure 85 , are provided as specializations of Constructor.

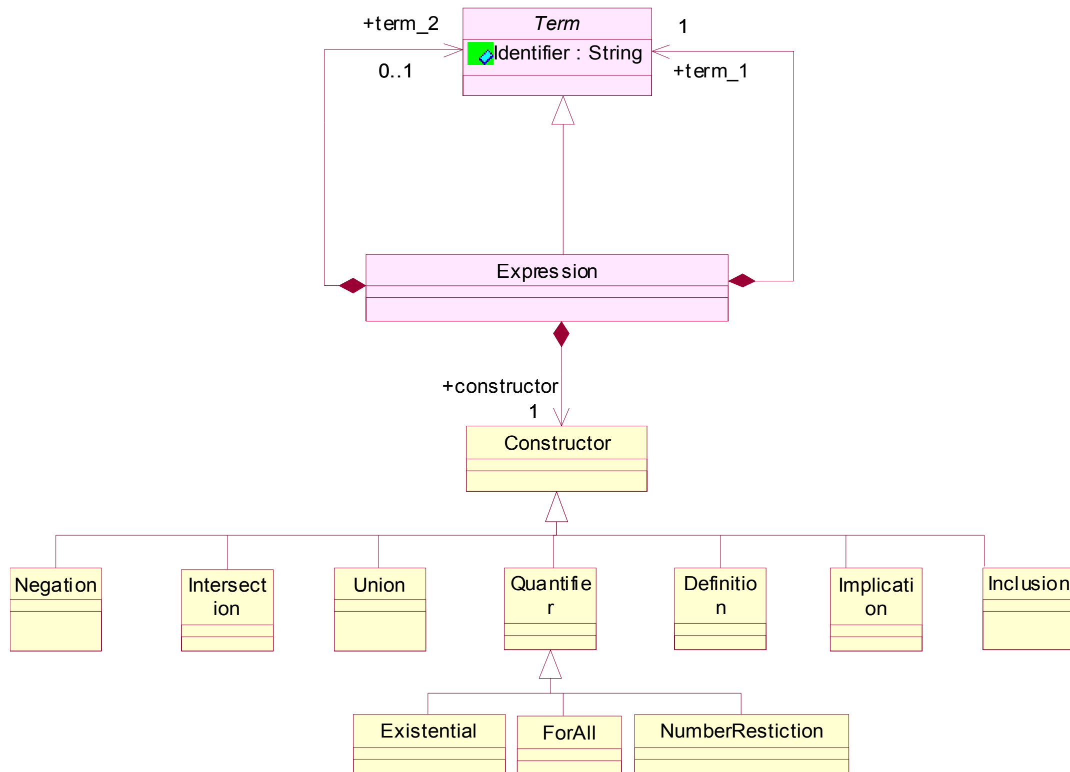


Figure 85 Specialisations of Constructor

B.6.1 Term

Description

Terms are the components used to build expressions. They are an abstract root class of most DL classes, excluding only ABox, TBox, and Constructors.

Similar Terms

Word, Component

Attributes

- Identifier: String [0..1] – An optional identifier for this term.

B.6.2 Expression

Description

Expressions are the representation of the DL Knowledge Base Description Language, shown in Figure 80 . Expressions are an extension of Term and are also constructed from Terms using Constructors. Thus allowing arbitrarily complex expressions to be created.

Similar Terms

Statement, Formula

Associations

- term_1: Term[1] – The required term for the constructor.
- term_2: Term[0..1] – The optional term for the constructor.
- constructor: Constructor[1] – a monadic or dyadic operator applied to the terms.

B.6.3 Constructor

Description

A Constructor is an operator that is used to build expressions. A Constructor may be either monadic or dyadic.

Note that individual specializations of constructor may have additional semantics and restrictions that are not elaborated here.

Similar Terms

Operator

Semantics

- Monadic constructors have term_2.multiplicity = 0
- Dyadic constructors have term_2.multiplicity = 1

B.6.4 Intersection

Description

The Intersection constructor is a dyadic constructor. It results in the set of instances that are members of both the left-hand term and the right-hand term.

B.6.5 Negation

Description

The Negation constructor is a monadic constructor. It results in the set containing all instances not contained in the right-hand term.

B.6.6 Union

Description

The Union constructor is a dyadic constructor. It results is the set containing any instance that is a member of either the left-hand or right-hand term.

B.6.7 Quantifier

Description

A Quantifier is a specialization of Constructor. It is a monadic constructor. They are operators that bind the number of a role's assertions by specifying their quantity in a logical formula.

B.6.8 ForAll

Description

ForAll is a specialization of Quantifier. ForAll specifies that all members of term_1 must have the binding value for the specified role.

B.6.9 Existential

Description

Existential is a specialization of Quantifier. Existential specifies that at least one member of term_1 has the binding value for the specified role.

B.6.10 NumberRestriction

Description

NumberRestriction is a specialization of Quantifier. NumberRestriction specifies that a specified number of members have a value for the specified role, similar to cardinality or multiplicity.

Further specializations of NumberRestriction may include upper bound, lower bound and exact number specifications.

B.6.11 Definition

Description

Definition is a specialization of Constructor. It is dyadic. Definition is used in axioms to define the left-hand term as exactly the right-hand term.

B.6.12 Implication

Description

Implication is a specialization of Constructor. It is dyadic. Implication is a logical relationship between the term_1 and term_2, that states term_2 is true if term_1 is true.

B.6.13 Inclusion

Description

Inclusion is a specialization of Constructor. It is dyadic. Inclusion is a relation between the term_1 and the term_2 that states all members of the first are also members of the second. Inclusion is similar to sub-types, in that all members of a sub-type are included in the super-type.

B.7 Examples

The following two examples, in Figures A1-7 and A1-8, illustrate the representation of simple statements as instance models of the DL meta-model.

B.7.1 Example One

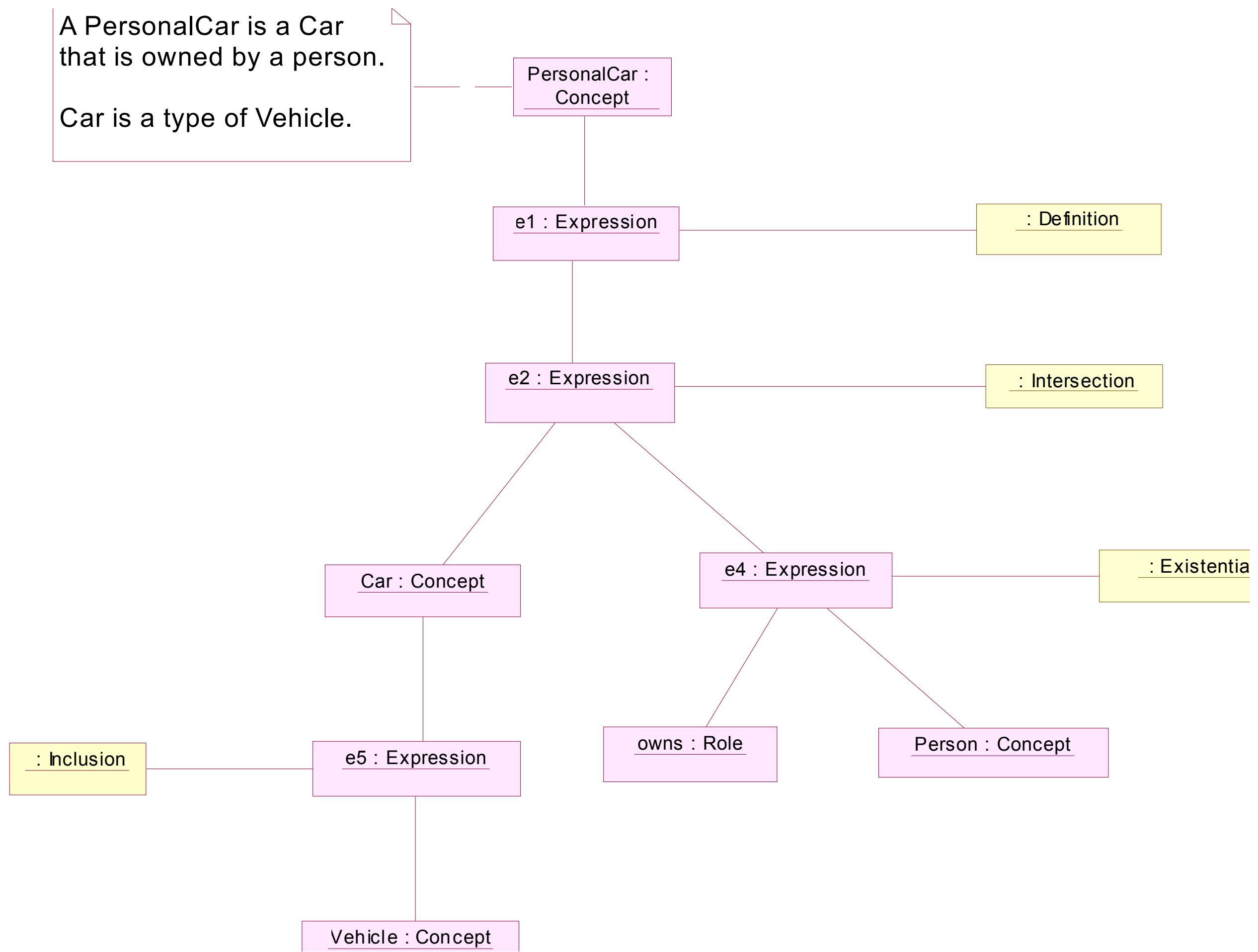


Figure 86 Example One

B.7.2 Example Two

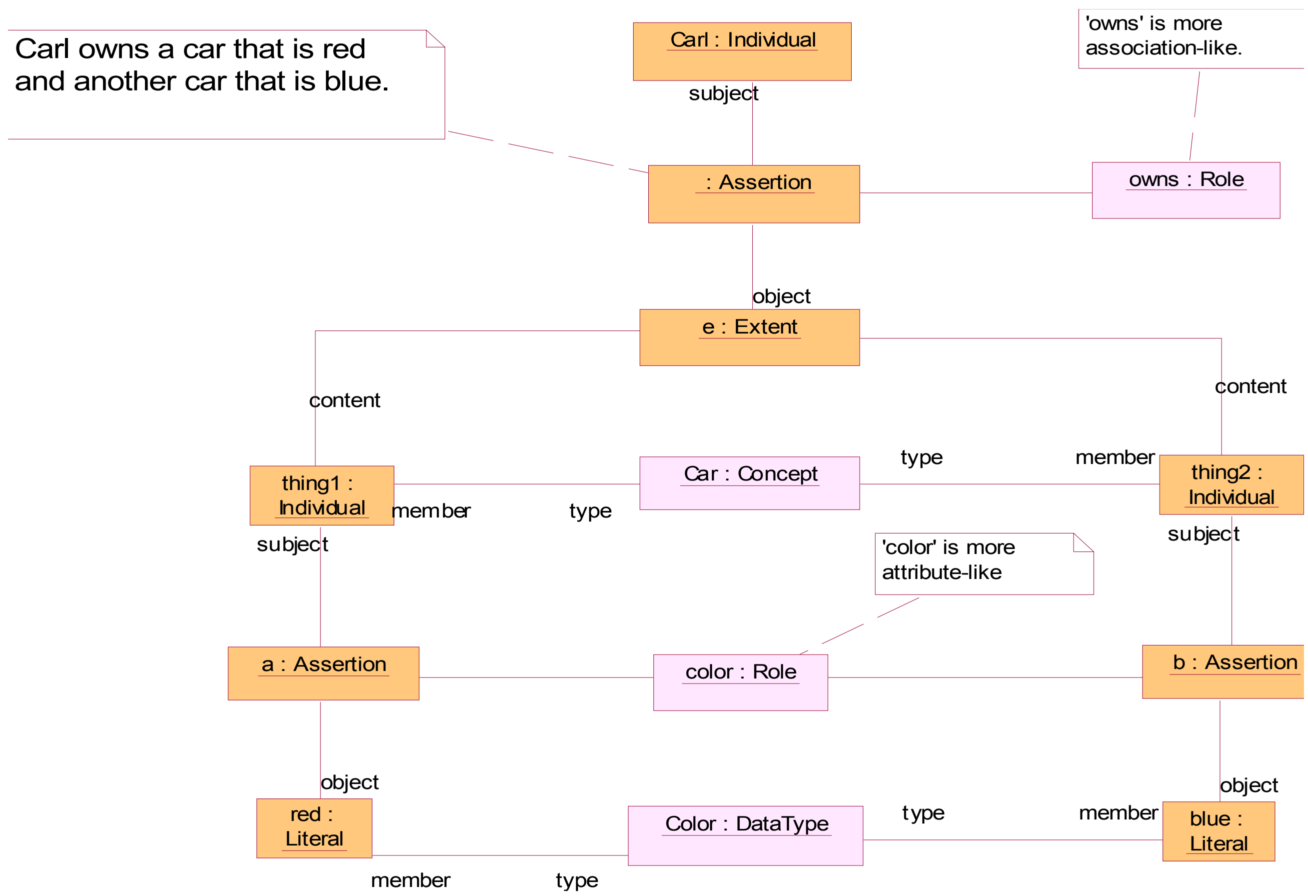


Figure 87 Example Two

B.8 Overview Diagram

Figure 88 provides a overview of the complete class hierarchy and key associations in the DL meta-model.

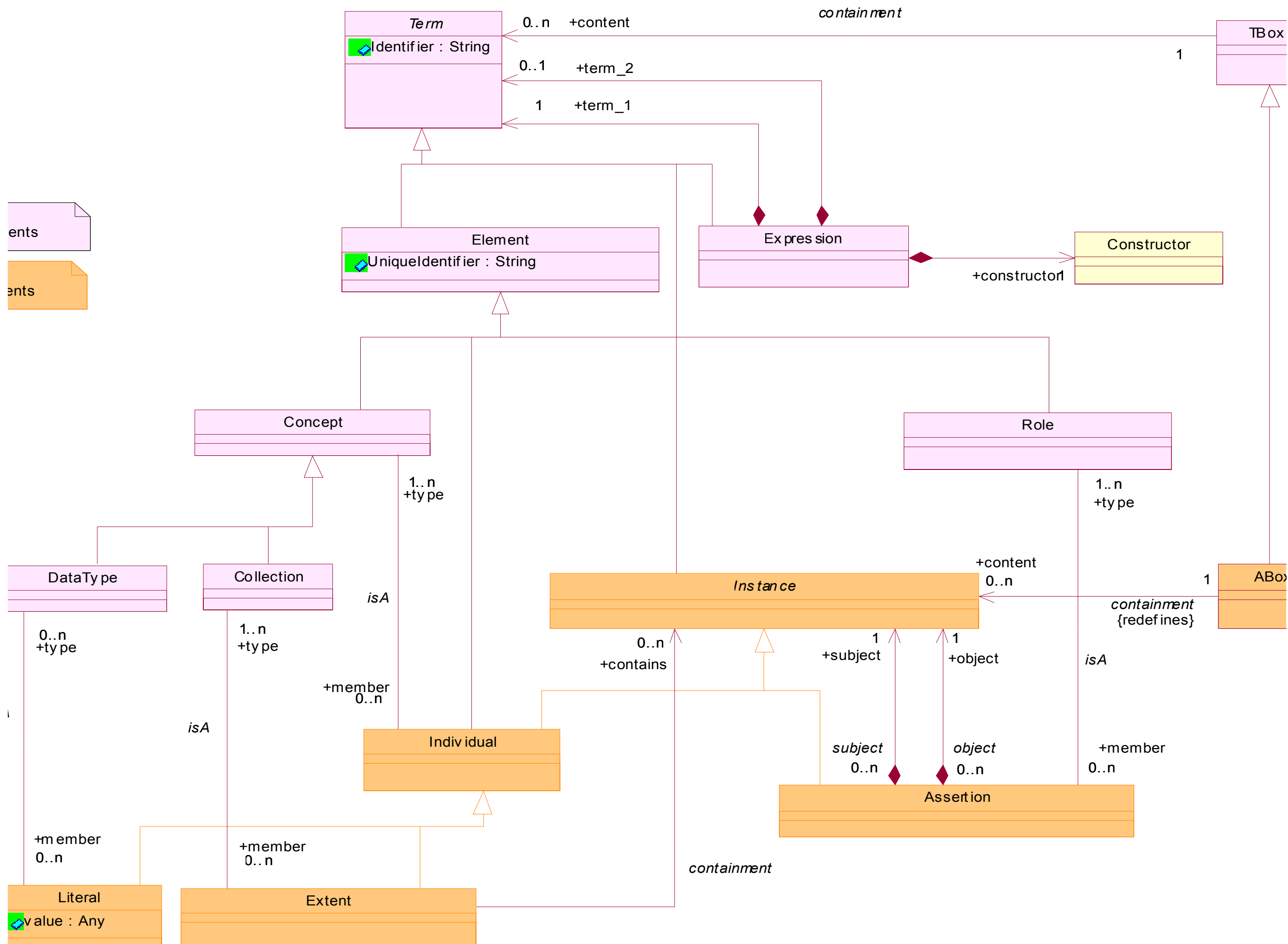


Figure 88 Complete DL Metamodel

C Extending the ODM

C.1 Extendibility

From the Usage Scenarios and Goals of Chapter 7, there is an enormous variety of kinds of application for ontologies. They can be used at design time only or at both design and run time. They can be schemas only or involve both schemas and instances. Their structure can be imposed from outside their domain or can emerge from the activities of interoperating parties. And so on.

Many of these kinds of application have special requirements which are common to many application instances but which are not at all universal. The ODM submission has limited its efforts to the most general structural issues.

However, in practice one can envisage particular extensions to the general structures which support significant numbers of application instances, which would be published by third parties outside the OMG ODM process but which would be consistent with the ODM, in much the same way as the Dublin Core metadata standard is published as an RDFS namespace. These extensions would use the MOF Package as a medium.

We will illustrate this facility with three examples, all of which use model elements from OWL packages so are seen as extending OWL. The examples are respectively of metaclass taxonomies, semantic domain instance models, and n-ary associations.

C.2 Metaclass Taxonomy

The first example, shown in Figure 89 , that of a metaclass taxonomy, extends OWLClass with the distinction between countable and bulk classes as advocated by Guarino and Welty [GuarWel]. A countable class has an extent consisting of identifiable individuals while a bulk class is a sort of amorphous mass like length measured in metres or value measured in Euros. In a model instance, classes would be instances of one of the specialized subclasses rather than of the more general OWLClass.

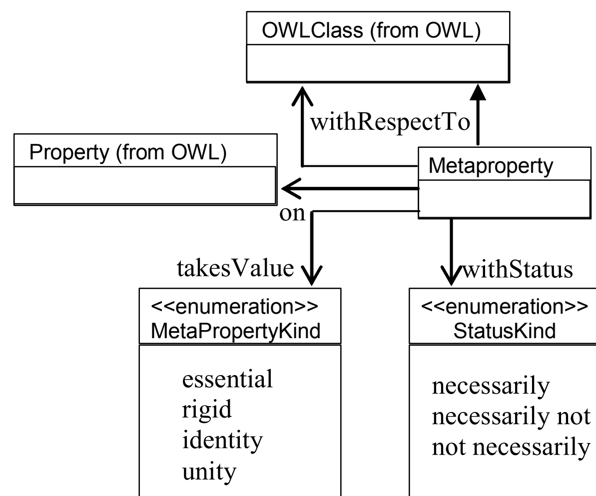


Figure 89 Countable/Bulk Package Extending OWL

This same approach can be used with other taxonomies of metaclasses, for example the taxonomy of endurants and perdurants proposed in the DOLCE system [DOLCE].

It is possible to develop these packages as extensions to one of the metamodels, in this case OWL, then use the ODM mapping facilities to migrate it to any of the other metamodels. Note that all of the metamodels supported by the ODM permit multiple inheritance, so that several such extensions can be used simultaneously.

C.3 Models of General Kinds of Application Domains

A feature of OWL is that properties are by default defined globally, with range and domain both Thing. This makes it possible to represent mereological relationships as instances of property. Instances of metaclasses can be modeled using semantic domain models, a facility of MOF 2.0. For example, Figure 90 defines a version of isPartOf which is transitive, every part belongs to at least one whole (and by transitivity to all the wholes up the chain), and a part cannot exist without its corresponding whole. This kind of part-of relation could be suitable for modeling say the Olympic family. An athlete is part of an event (if a competitor), an event is part of a sporting program, a sporting program is part of the Olympics of a given Olympiad, and anyone who competes in any event in any program in any Olympics is a part of the Olympic family. But an Olympics cannot exist without at least one program, a program must have at least one event, and an event at least one competitor.

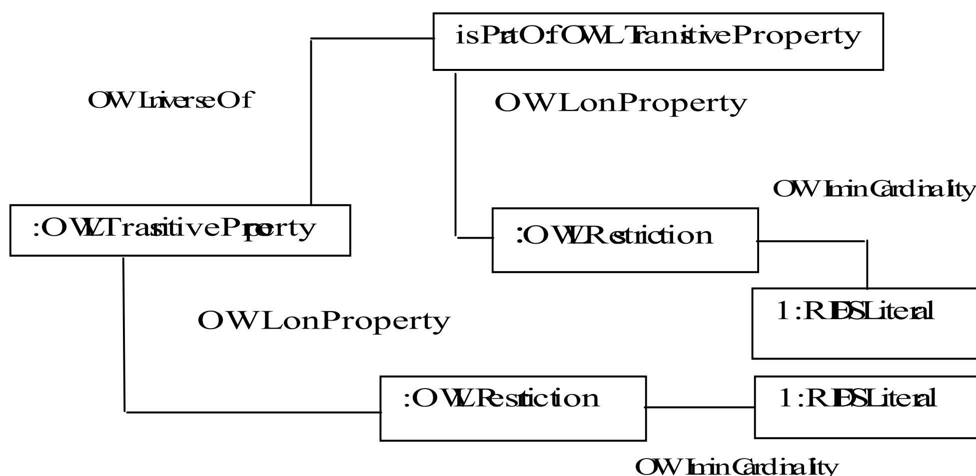


Figure 90 Semantic Domain Model for isPartOf Property

There are a large number of varieties of mereotopological relationships [WinChafHerr], including those specified in UML. They could be catalogued and published as a package, perhaps with specialized software.

C.4 N-ary Associations

A key aspect of the OntoClean methodology [OntoClean] is the concept of a metaproperty. For example, a property has the metaproperty essential with respect to a class if being an instance of that class determines the value of the property. Besides essential, the metaproperties include rigid, identity and unity. A property with respect to a class can necessarily, necessarily not or not necessarily have a given metaproperty. A natural way to model metaproperties is as quaternary associations.

Most of the metamodels in the ODM permit n-ary associations, except RDFS/OWL. But an n-ary association can be represented as a class with n binary properties. To be consistent with the previous examples, a possible package to model metaproperties in Figure 91 extends the OWL metamodel. Note that the metaproperty is modeled as a subclass of OWLClass. This can facilitate mapping from OWL to an n-ary association or equivalent in another metamodel. Note also the enumerations, which are instances of the MOF element type.

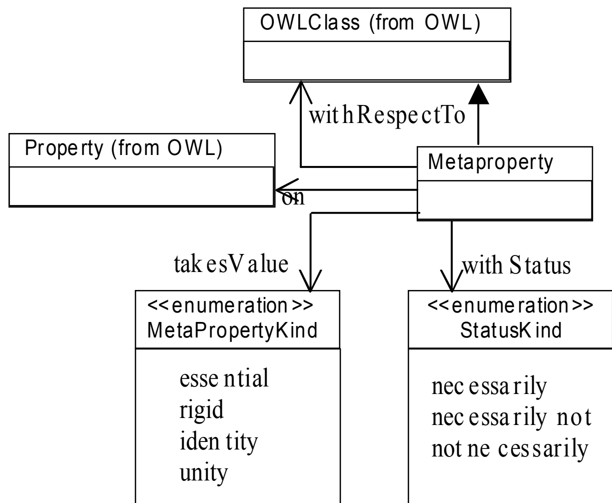


Figure 91 Metaproperty Package for OWL

D Open Issues

A small set of open issues exists primarily due to limitations on time and resource availability. It is the desire of the team that the issues be corrected before the finalization of the standard.

The primary area of concern at this point is with regard to the structure and packaging of the RDFS Metamodel. Several issues were raised by team members and other supporters regarding the need for support for certain RDF concepts in the RDFS (or a related, subordinate) metamodel. These elements facilitate interoperability among tools that exchange several concrete syntax representations of RDF, and have a direct downstream impact on the representation of individuals, in particular, in OWL. Numerous working meetings including RDF language experts have been held since these issues were first raised. Recently, additional expertise from within IBM has been focused on assisting us in finalizing an approach. This extended team has raised issues with the W3C regarding ambiguities in the RDF specifications, which have contributed to the lack of consensus within the core ODM team. We believe that with the help of this expanded ODM development team and RDF language authors, agreement and a finalized specification are within reach.

Thus, this draft revised submission reflects no change from the prior draft to the RDFS and OWL metamodels, pending resolution of these outstanding issues in the RDFS metamodel:

- Incorporation (and possibly repackaging / refactoring) of the RDF-related metamodel extensions currently specified in Chapter 16, UML Profiles for RDF Schema and OWL - specifically revisions to better support RDF statements, reification, graphs, and blank node semantics, as well as to provide a number of concrete syntax related elements that are common to several RDF serialization formats
- Determination of whether or not (and how) to separate RDF-specific constructs from those of RDF Schema, and related to that, whether or not the RDF Schema metamodel should be able to stand alone, without the RDF constructs
- Agreement on the appropriate representation for RDF statements, or triples, and whether or not they should be considered resources
- Agreement on the appropriate representation for blank nodes, and whether or not they should be considered resources

In addition to these issues, the other area of note is with regard to whether the language mappings should be normative and the nature of the representation. In the process of attempting to map the Topic Maps metamodel to the RDFS and OWL metamodels in particular, DSTC discovered issues in the metamodels as well as issues with the mapping representation, which have led us to the conclusion that the mappings should be non-normative. More on this topic is given in Chapter 8, Design Rationale, and in Appendix E, Mappings - Informative, Not Normative. Nonetheless, we believe the mappings are important and will provide guidelines for those attempting to use this specification, and thus are planning to complete them (Chapters 18-21), in the MOF QVT Relations language, over the course of the next revision cycle.

Finally, the following additional minor issues remain open:

- Open issues with respect to Chapter 12 OWL Metamodel include (1) aspects of the metamodel concerning representation of individuals, (2) issues in navigation from an ontology to the statements it contains, (3) minor issues with regard to missing value restrictions and class inheritance for a couple of OWL features, and (4) the need for a section at the end of the chapter concerning constraints required to differentiate OWL DL from OWL Full.
- An open issue exists related to the technical completeness of the RDFS, OWL and ER metamodels. Missing are some multiplicities, role names and OCL specifications for non-graphically conveyed constraints.
- A minor issue exists related to the completeness of Chapter 16 UML Profiles for RDFS and OWL -- primarily with regard to a few missing example diagrams in the OWL profile.

- An open issue exists with respect to Chapter 18 Mapping UML to OWL -- this chapter is incomplete (primarily with regard to the mapping from OWL to UML), and depends on chapters 11, 12, and 16 (thus modifications to those chapters must be synchronized with this mapping).
- A minor issue exists with respect to Chapter 21 Mapping RDFS and OWL to CL -- a few corrections to the mapping are needed due to recent modifications to the metamodel to align with the ISO CD revision.
- An open issue exists regarding the model library defined in Appendix A -- this needs to be updated to reflect recent metamodel and profile modifications.
- No MOF2-compliant tool available for graphically defining metamodels. This submission uses IBM Rational Software Modeler (recent modifications only), which is “close” and IBM Rational Rose for graphically defining metamodels. The submitters hope to migrate to UML2 compliant tools (or versions of tools) as they become more readily available.
- No UML2-compliant tool available for defining UML profiles. This submission uses text (primarily) for defining UML profiles.
- No MOF2-compliant tool available fo generating XMI artifacts from metamodels.
- No MOF2-compliant tool available for generating JMI artifacts from metamodels.

E Mappings - Informative, Not Normative

The RFP calls for mappings (6.2) which are two-way and bounded (6.5.3, 6.5.3.1). However, in developing the mappings for the various ODM languages, the team concluded that the mappings we specify can not in practice be normative.

In our discussion in 10.2.3, for example we see that there are two different ways to map n-ary associations from UML to OWL, depending on whether we take OWL Full or OWL DL as target. In 10.2.2, we note that OWL has a mandatory universal superclass (`owl:Thing`) which can map to a universal superclass in UML, but this is contrary to normal practice in UML modelling. A particular project might analyze the uses of universal properties in the OWL source model and choose to declare a number of more general but not universal superclasses in the UML target.

In the W3C Semantic Web Best Practices and Deployment task force's report on Topic Map mappings [RDF/TM], the point is made several times that there are different ways to map particular structures, and that each way has its advantages and disadvantages. In any particular project, design decisions will be taken in favor of advantages and against disadvantages so different projects will map in different ways.

There are several kinds of problems. One we can call structure conflation, where two constructs in one system map to a single construct in the other. In this case, a general-purpose mapping doesn't round trip. UML binary associations and class-valued attributes map to OWL properties, for example. In topic maps, three different kinds of identifiers map to one kind in OWL.

But there is nothing to stop a particular project from specifying naming conventions so there is a record in the target of what construct the source was, and from maintaining that convention in subsequent development.

A second kind of problem we will call structure loss. Here a complex construct is mapped to a collection of simpler constructs. There is insufficient information in the target metamodel for a general mapping to map collections of simple constructs to complex constructs in the source metamodel. Examples here are UML N-ary associations and association classes, which get mapped to a class and a collection of properties. In Topic Maps, the Association construct is typed itself and has N typed roles. The association maps to a class and the typed roles to properties. It is in general impossible to reliably map the reverse.

But again, there is nothing to stop a particular project from using naming conventions or annotations to retain a memory of the structure, and maintaining those conventions in subsequent maintenance so as to be able to reverse map.

Alternatively, a TM project could decide to limit itself to binary associations, making possible mapping associations directly to properties in that particular case.

The third kind of problem we will call trapdoor mappings, where a kind of construct in the source is mapped to a very specific arrangement of a general structure in the target. The analogy is with cryptography, where the encryption function takes any plaintext into an encrypted text, but almost no encrypted texts map back to plain text.

In topic maps, this occurs with the mapping of scope and variant names to specific properties in OWL identified with TM URIs. OWL properties map to TM associations with specific roles named with OWL URIs. Unless the source for a reverse mapping happened to maintain these conventions, it would be impossible to reverse in a sensible way.

A fourth kind of problem stems from what we will call feature lack, that the target metamodel lacks a feature present in the source. In this case there is no apparent general way to map the feature from the source. But in a particular project the feature may for example be used in a particular way leading to a mapping to target features particularized by naming conventions. OWL restriction classes relative to UML or Topic Map are of this kind.

The fifth kind of problem is what we will call incompatible structural principles. The different metamodels are organized very differently. UML is organized around classes, with instances as subordinate objects. OWL has both classes and individuals typed only by a universal superclass. In Topic Maps, a Topic instance can be either typed or not. But a particular project might use a particular discipline in its use of these structures leading to mappings not otherwise feasible.

In practice, the mappings provided in the ODM can be useful, though. First, they show feasibility of one set of design choices for the mappings, providing a baseline from which a particular project can vary. Second, they bring clearly to the fore the detailed relationships among the metamodels. These relationships can help those who understand one of the target languages to come to an understanding of the others. UML is similar to ER, but both are very different from RDFS/OWL, and all are quite different from TM. CL has far greater functionality than any of the others.

So although normative mappings are not feasible, we argue that the mappings presented have strong informative value.