

## SDMX STANDARDS: SECTION 2

# INFORMATION MODEL: UML CONCEPTUAL DESIGN

VERSION 2.1

July 2011



## Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Related Documents	1
1.2	Modelling Technique and Diagrammatic Notes	1
1.3	Overall Functionality	2
1.3.1	Information Model Packages .....	2
1.3.2	Version 1.0.....	3
1.3.3	Version 2.0/2.1 .....	3
<b>2</b>	<b>Actors and Use Cases .....</b>	<b>5</b>
2.1	Introduction	5
2.2	Use Case Diagrams	6
2.2.1	Maintenance of Structural and Provisioning Definitions .....	6
2.2.2	Publishing and Using Data and Reference Metadata.....	10
<b>3</b>	<b>SDMX Base Package.....</b>	<b>13</b>
3.1	Introduction	13
3.2	Base Structures - Identification, Versioning, and Maintenance	14
3.2.1	Class Diagram .....	14
3.2.2	Explanation of the Diagram.....	14
3.3	Basic Inheritance	18
3.3.1	Class Diagram– Basic Inheritance from the Base Inheritance Classes .....	18
3.3.2	Explanation of the Diagram.....	19
3.4	Data Types	19
3.4.1	Class Diagram .....	19
3.4.2	Explanation of the Diagram.....	20
3.5	The Item Scheme Pattern	21
3.5.1	Context.....	21
3.5.2	Class Diagram .....	21
3.5.3	Explanation of the Diagram.....	22

3.6	The Structure Pattern	23
3.6.1	Context.....	23
3.6.2	Class Diagrams.....	24
3.6.3	Explanation of the Diagrams.....	26
<b>4</b>	<b>Specific Item Schemes .....</b>	<b>31</b>
4.1	Introduction	31
4.2	Inheritance View	32
4.3	Codelist	33
4.3.1	Class Diagram .....	33
4.3.2	Explanation of the Diagram.....	34
4.4	Concept Scheme and Concepts	36
4.4.1	Class Diagram - Inheritance .....	36
4.4.2	Explanation of the Diagram.....	37
4.4.3	Class Diagram - Relationship .....	38
4.4.4	Explanation of the diagram .....	38
4.5	Category Scheme	40
4.5.1	Context.....	40
4.5.2	Class diagram - Inheritance .....	40
4.5.3	Explanation of the Diagram.....	41
4.5.4	Class diagram - Relationship .....	42
4.6	Organisation Scheme	44
4.6.1	Class Diagram .....	44
4.6.2	Explanation of the Diagram.....	44
4.7	Reporting Taxonomy	48
4.7.1	Class Diagram .....	48
4.7.2	Explanation of the Diagram.....	48
<b>5</b>	<b>Data Structure Definition and Dataset .....</b>	<b>51</b>
5.1	Introduction	51

5.2	Inheritance View	52
5.2.1	Class Diagram .....	52
5.2.2	Explanation of the Diagram.....	53
5.3	Data Structure Definition – Relationship View	55
5.3.1	Class Diagram .....	55
5.3.2	Explanation of the Diagram.....	55
5.4	Data Set – Relationship View	65
5.4.1	Context.....	65
5.4.2	Class Diagram .....	65
5.4.3	Explanation of the Diagram.....	66
<b>6</b>	<b>Cube.....</b>	<b>74</b>
6.1	Context	74
6.2	Support for the Cube in the Information Model	74
<b>7</b>	<b>Metadata Structure Definition and Metadata Set .....</b>	<b>75</b>
7.1	Context	75
7.2	Inheritance	75
7.2.1	Introduction .....	75
7.2.2	Class Diagram - Inheritance .....	76
7.2.3	Explanation of the Diagram.....	77
7.3	Metadata Structure Definition	77
7.3.1	Introduction .....	77
7.3.2	Structures Already Described .....	77
7.3.3	Class Diagram – Relationship .....	78
7.3.4	Explanation of the Diagram.....	78
7.4	Metadata Set	84
7.4.1	Class Diagram .....	84
7.4.2	Explanation of the Diagram.....	85
<b>8</b>	<b>Hierarchical Code List .....</b>	<b>92</b>

8.1	Scope	92
8.2	Inheritance	93
8.2.1	Class Diagram .....	93
8.2.2	Explanation of the Diagram.....	93
8.3	Relationship	94
8.3.1	Class Diagram .....	94
8.3.2	Explanation of the Diagram.....	94
<b>9</b>	<b>Structure Set and Mappings.....</b>	<b>98</b>
9.1	Scope	98
9.2	Structure Set	99
9.2.1	Class Diagram – Inheritance .....	99
9.2.2	Class Diagram – Relationship .....	100
9.2.3	Explanation of the Diagram.....	100
9.3	Structure Map	102
9.3.1	Class Diagram .....	102
9.3.2	Explanation of the Diagram.....	102
9.4	Item Scheme Map	104
9.4.1	Context.....	104
9.4.2	Class Diagram .....	105
9.4.3	Explanation of the Diagram.....	105
9.5	Hybrid Codelist Map	108
9.5.1	Class Diagram .....	108
9.5.2	Explanation of the Diagram.....	108
<b>10</b>	<b>Constraints.....</b>	<b>111</b>
10.1	Scope	111
10.2	Inheritance	111
10.2.1	Class Diagram of Constraining Artefacts - Inheritance.....	111
10.2.2	Explanation of the Diagram.....	111

10.3	Constraints	112
10.3.1	Relationship Class Diagram – high level view .....	112
10.3.2	Explanation of the Diagram.....	113
10.3.3	Relationship Class Diagram – Detail .....	114
<b>11</b>	<b>Data Provisioning.....</b>	<b>124</b>
11.1	Class Diagram	124
11.2	Explanation of the Diagram	125
11.2.1	Narrative.....	125
11.2.2	Definitions .....	126
<b>12</b>	<b>Process.....</b>	<b>128</b>
12.1	Introduction	128
12.2	Model – Inheritance and Relationship view	129
12.2.1	Class Diagram .....	129
12.2.2	Explanation of the Diagram.....	129
<b>13</b>	<b>Transformations and Expressions .....</b>	<b>132</b>
13.1	Scope	132
13.2	Model - Inheritance View	133
13.2.1	Class Diagram .....	133
13.2.2	Explanation of the Diagram.....	133
<b>14</b>	<b>Appendix 1: A Short Guide To UML in the SDMX Information Model.....</b>	<b>137</b>
14.1	Scope	137
14.2	Use Cases	137
14.3	Classes and Attributes	138
14.3.1	General .....	138
14.3.2	Abstract Class.....	139
14.4	Associations	139
14.4.1	General .....	139
14.4.2	Simple Association.....	139

14.4.3	Aggregation.....	140
14.4.4	Association Names and Association-end (role) Names .....	141
14.4.5	Navigability.....	141
14.4.6	Inheritance .....	142
14.4.7	Derived association.....	142



## Corrigendum

The following problems with the specification dated April 2011 have been rectified as described below.

### 1. Problem

Figure 35 - Class diagram of the Item Scheme Map – shows the ItemSchemeMap with an alias attribute. This attribute is not supported in the schemas.

#### Rectification

The attribute alias is removed from the ItemSchemeMap class and also from the table in section 9.4.3.2.

### 2. Problem

The Time Dimension and Measure Dimension in the Figure 40 - Constraints - Cube Region and Metadata Target Region Constraints – are shown as inheriting from Dimension, but in Figure 23 - Relationship class diagram of the Data Structure Definition excluding representation – they, and Dimension itself, inherit from DimensionComponent

#### Rectification

Dimension, TimeDimension, and MeasureDimension all inherit from DimensionComponent and Figure 40 is changed to reflect this.

### 3. Problem

The class SelectionValue is shown as a class in Figure 40 - Constraints - Cube Region and Metadata Target Region Constraints – but it is not described in the table at 10.3.3.2.

#### Rectification

The class SelectionValue is added to the table at 10.3.3.2.

## Change History

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36

Version 1.0 – initial release September 2004.

Version 2.0 – release November 2005

Major functional enhancements by addition of new packages:

- Metadata Structure Definition
- Metadata Set
- Hierarchical Code Scheme
- Data and Metadata Provisioning
- Structure Set and Mappings
- Transformations and Expressions
- Process and Transitions

Re-engineering of some SDMX Base structures to give more functionality:

- Item Scheme and Item can have properties – this gives support for complex hierarchical code schemes (where the property can be used to sequence codes in scheme), and Item Scheme mapping tables (where the property can give additional information about the map between the two schemes and the between two Items)
- revised Organisation pattern to support maintained schemes of organisations, such as a data provider
- modified Component Structure pattern to support identification of roles played by components and the attachment of attributes
- change to inheritance to enable more artefacts to be identifiable and versionable

Introduction of new types of Item Scheme:

- Object Type Scheme to specify object types in support of the Metadata Structure Definition (principally the object types (classes) in this Information Model)
- Type Scheme to specify types other than object type
- A generic Item Scheme Association to specify the association between Items in two or more Item Schemes, where such associations cannot be described in the Structure Set and Transformation.

The Data Structure Definition is introduced as a synonym for Key Family though the term Key Family is retained and used in this specification.

37 Modification to Data Structure Definition (DSD) to

38

39 • align the cross sectional structures with the functionality of the schema

40 • support Data Structure Definition extension (i.e. to derive and extend a Data Structure  
41 Definition from another Data Structure Definition), thus supporting the definition of a  
42 related “set” of key families

43 • distinguish between data attributes (which are described in a Data Structure Definition)  
44 from metadata attributes (which are described in a metadata structure definition)

45 • attach data attributes to specific identifiable artefacts (formally this was supported by  
46 attachable artefact)

47 Domain Category Scheme re-named Category Scheme to better reflect the multiple usage of  
48 this type of scheme (e.g. subject matter domain, reporting taxonomy).

49

50 Concept Scheme enhanced to allow specification of the representation of the Concept. This  
51 specification is the default (or core) representation and can be overridden by a construct that  
52 uses it (such as a Dimension in a Data Structure Definition).

53

54 Revision of cross sectional data set to reflect the functionality of the version 1.0 schema.

55

56 Revision of Actors and Use Cases to reflect better the functionality supported.

57

58 Version 2.1 – release April 2011

59

60 The purpose of this revision is threefold:

61

- 62 • To introduce requested changes to functionality
- 63 • To align the model and syntax implementations more closely (note, however, that the  
64 model remains syntax neutral)
- 65 • To correct errors in version 2.0

66

67 *SDMX Base*

68 *Basic inheritance and patterns*

69

70 1. The following attributes are added to Maintainable:

71

72 i) isExternalReference

73 ii) structure URL

74 iii) serviceURL

75

76 2. Added Nameable Artefact and moved the Name and Description associations from  
77 Identifiable Artefact to Nameable Artefact. This allows an artefact to be identified (with  
78 id and urn) without the need to specify a Name.

79

80 3. Removed any inheritance from Versionable Artefact with the exception of Maintainable  
81 Artefact – this means that only Maintainable objects can be versioned, and objects  
82 contained in a maintainable object cannot be independently versioned.

83

- 84 4. Renamed MaintenanceAgency to Agency 0 this is its name in the schema and the  
85 URN.  
86  
87 5. Removed abstract class Association as a subclass of Item (as these association types  
88 are not maintained in Item Schemes). Specific associations are modelled explicitly  
89 (e.g. Categorisation, ItemScheme, Item).  
90  
91 6. Added ActionType to data types.  
92  
93 7. Removed Coded Artefact and Uncoded Artefact and all subclasses (e.g. Coded Data  
94 Attribute and Uncoded Data Attribute) as the “Representation” is more complex than  
95 just a distinction between coded and uncoded.  
96  
97 8. Added Representation to the Component. Removed association to Type.  
98  
99 9. Removed concept role association (to Item) as roles are identified by a relationship to  
100 a Concept.  
101  
102 10. Removed abstract class Attribute as both Data Attribute and Metadata Attribute have  
103 different properties. Data Attribute and Metadata Attribute inherit directly from  
104 Component.  
105  
106 11. isPartial attribute added to Item Scheme to support partial Item Schemes (e.g. partial  
107 Code list).  
108

#### 109 *Representation*

- 110  
111 1. Removed interval and enumeration from Facet.  
112 2. added facetValueType to Facet.  
113 3. Re-named DataType to facetValueType.  
114 4. Added observationalTimePeriod, inclusiveValueRange and exclusiveValueRange to  
115 facetValueType.  
116 5. Added ExtendedFacetType as a sub class of FacetType. This includes Xhtml as a  
117 facet type to support this as an allowed representation for a Metadata Attribute  
118

#### 119 *Organisations*

- 120 1. Organisation Role is removed and replaced with specific Organisation Schemes of  
121 Agency, Data Provider, Data Consumer, Organisation Unit.  
122

#### 123 *Mapping (Structure Maps)*

124  
125 Updated Item Scheme Association as follows:  
126

- 127 1. Renamed to Item Scheme Map to reflect better the sub classes and relate better to the  
128 naming in the schema.  
129  
130 2. Removed inheritance of Item Scheme Map from Item Scheme, and inherited directly  
131 from Nameable Artefact.  
132  
133 3. Item Association inherits from Identifiable Artefact.  
134  
135 4. Removed Property from the model as this is not supported in the schema.

- 136  
137 5. Removed association type between Item Scheme Map and Item, and Association and  
138 Item.  
139  
140 6. Removed Association from the model.  
141  
142 7. Made Item Association a sub class of Identifiable, was a sub class Item.  
143  
144 8. Removed association to Property from both Item Scheme Map and Item.  
145  
146 9. Added attribute alias to both Item Scheme Association and Item Association.  
147  
148 10. Made Item Scheme Map and Item Association abstract.  
149  
150 11. Added sub-classes to Item Scheme Map – there is a subclass for each type of Item  
151 Scheme Association (e.g. Code list Map).  
152  
153 12. Added mapping between Reporting Taxonomy as this is an Item Scheme and can be  
154 mapped in the same way as other Item Schemes.  
155  
156 13. Added Hybrid Code list Map and Hybrid Code Map to support code mappings between  
157 a Code list and a Hierarchical Code list.  
158

159 Mapping: Structure Map

- 160  
161 1. This is a new diagram. Essentially removed inherited /hierarchy association between  
162 the various maps, as these no longer inherit from Item, and replaced the associations  
163 to the abstract Maintainable and Versionable Artefact classes with the actual concrete  
164 classes.  
165  
166 2. Removed associations between Code list Map, Category Scheme Map, and Concept  
167 Scheme Map and made this association to Item Scheme Map.  
168  
169 3. Removed hierarchy of Structure Map.  
170

171 Concept

- 172  
173 1. Added association to Representation.  
174

175 Data Structure Definition

- 176  
177 1. Added Measure Dimension to support structure-specific renderings of the DSD. The  
178 Measure Dimension is associated to a Concept Scheme that specifies the individual  
179 measures that are valid.  
180  
181 2. The three types of “Dimension”, - Dimension, Measure Dimension, Time Dimension –  
182 have a super class – Dimension Component  
183  
184 3. Added association to a Concept that defines the role that the component (Dimension,  
185 Data Attribute, Measure Dimension) plays in the DSD. This replaces the Boolean  
186 attributes on the components.  
187

- 188 4. Added Primary Measure and removed this as role of Measure.
- 189
- 190 5. Deleted the derived Data Structure Definition association from Data Structure
- 191 Definition to itself as this is not supported directly in DSD.
- 192
- 193 6. Deleted attribute GroupKeyDescriptor.isAttachmentConstraint and replaced with an
- 194 association to an Attachment Constraint.
- 195
- 196 7. Replaced association from Data Attribute to Attachable Artefact with association to
- 197 Attribute Relationship.
- 198
- 199 8. Added a set of classes to support Attribute Relationship.
- 200
- 201 9. Renamed KeyDescriptor to DimensionDescriptor to better reflect its purpose.
- 202
- 203 10. Renamed GroupKeyDescriptor to GroupDimensionDescriptor to better reflect its
- 204 purpose.
- 205

#### 206 Code list

- 207
- 208 1. CodeList classname changed to Codelist.
- 209
- 210 2. Removed codevalueLength from Codelist as this is supported by Facet.
- 211
- 212 3. Removed hierarchyView association between Code and Hierarchy as this association
- 213 is not implemented.
- 214

#### 215 Metadata Structure Definition(MSD)

- 216
- 217 1. Full Target Identifier, Partial Target Identifier, and Identifier Component are replaced by
- 218 Metadata Target and Target Object. Essentially this eliminates one level of
- 219 specification and reference in the MSD, and so makes the MSD more intuitive and
- 220 easier to specify and to understand.
- 221
- 222 2. Re-named Identifiable Object Type to Identifiable Object Target and moved to the MSD
- 223 package.
- 224
- 225 3. Added sub classes to Target Object as these are the actual types of object to which
- 226 metadata can be attached. These are Identifiable Object Target (allows reporting of
- 227 metadata to any identifiable object), Key Descriptor Values Target (allows reporting of
- 228 metadata for a data series key, Data Set Target (allows reporting of metadata to a
- 229 data set), and Reporting Period Target (allows the metadata set to specify a reporting
- 230 period).
- 231
- 232 4. Allowed Target Object can have any type of Representation, this was restricted in
- 233 version 2.0 to an enumerated representation in the model (but not in the schemas).
- 234
- 235 5. Removed Object Type Scheme (as users cannot maintain their own list of object
- 236 types), and replaced with an enumeration of Identifiable Objects.
- 237
- 238 6. Removed association between Metadata Attribute and Identifiable Artefact and
- 239 replaced this with an association between Report Structure and Metadata Target, and

240 allowed one Report Structure to reference more than one Metadata Target. This  
241 allowing a single Report Structure to be defined for many object types.

242

243 7. Added the ability to specify that a Metadata Attribute can be repeated in a Metadata  
244 Set and that a Metadata Attribute can be specified as “presentational” meaning that it  
245 is present for structural and presentational purposes, and will not have content in a  
246 Metadata Set.

247

248 8. The Representation of a Metadata Attribute uses Extended Facet (to support Xhtml).

249

#### 250 *Metadata Set*

251

252 1. Added link to Data Provider - 0..1 but note that for metadata set registration this will be  
253 1.

254

255 2. Removed Attribute Property as the underlying Property class has been removed.

256

257 3. One Metadata Set is restricted to reporting metadata for a single Report Structure.

258

259 4. The Metadata Report classes are re-structured and re-named to be consistent with the  
260 renaming and restructuring of the MSD.

261

262 5. Metadata Attribute Value is renamed Reported Attribute to be consistent with the  
263 schemas.

264

265 6. Deleted XML attribute and Contact Details from the inheritance diagram.

266

#### 267 *Category Scheme*

268 1. Added Categorisation. Category no longer has a direct association to Dataflow and  
269 Metadataflow.

270

271 2. Changed Reporting Taxonomy inheritance from Category Scheme to Maintainable  
272 Artefact.

273

274 3. Added Reporting Category and associated this to Structure Usage.

275

#### 276 *Data Set*

277

278 1. Removed the association to Provision Agreement from the diagram.

279

280 2. Added association to Data Structure Definition. This association was implied via the  
281 dataflow but this is optional in the implementation whereas the association to the Data  
282 Structure Definition is mandatory.

283

284 3. Added attributes to Data Set.

285

286 4. There is a single, unified, model of the Data Set which supports four types of data set:

287

288 • Generic Data Set – for reporting any type of data series, including time series  
289 and what is sometimes known as “cross sectional data”. In this data set, the  
290 value of any one dimension (including the Time Dimension) can be reported

- 291 with the observation (this must be for the same dimension for the entire data  
292 set)  
293  
294 • Structure-specific Data Set – for reporting a data series that is specific to a  
295 DSD  
296  
297 • Generic Time Series Data Set – this is identical to the Generic Data Set except  
298 it must contain only time series, which means that a value for the Time  
299 Dimension is reported with the Observation  
300  
301 • Structure-specific Time Series Data Set - this is identical to the Structure-  
302 specific Data Set except it must contain only time series, which means that a  
303 value for the Time Dimension is reported with the Observation.  
304
- 305 5. Removed Data Set as a sub class of Identifiable – but note that Data Set has a “setId”  
306 attribute.  
307
- 308 6. Added coded and uncoded variants of Key Value, Observation, and Attribute Value in  
309 order to show the relationship between the coded values in the data set and the  
310 Codelist in the Data Structure Definition.  
311
- 312 7. Made Key Value abstract with sub classes for coded, uncoded, measure  
313 (MeasureKeyValue) and time(TimeKeyValue) The Measure Key Value is associated to  
314 a Concept as it must take its identify from a Concept.  
315

#### 316 *XSDDataSet*

- 317 1. This is removed and replaced with the single, unified data set model.  
318

#### 319 *Constraint*

- 320  
321 1. Constraint is made Maintainable (was Identifiable).  
322  
323 2. Added artefacts that better support and distinguish (from data) the constraints for  
324 metadata.  
325  
326 3. Added Constraint Role to specify the purpose of the Constraint. The values are  
327 allowable content (for validation of sub set code code lists), and actual content (to  
328 specify the content of a data or metadata source).  
329

#### 330 *Process*

- 331 1. Removed inheritance from Item Scheme and Item: Process inherits directly from  
332 Maintainable and Process Step from Identifiable.  
333  
334 2. Removed specialisation association between Transition and Association.  
335  
336 3. Removed Transition Scheme - transitions are explicitly specified and not maintained as  
337 Items in a Item Scheme.  
338  
339 4. Removed Expression and replaced with Computation.  
340  
341 5. Transition is associated to Process Step and not Process itself. Therefore the source  
342 association to Process Step is removed.



- 343  
344 6. Removed Expressions as these are not implemented in the schemas. But note that the  
345 Transformations and Expressions model is retained, though it is not implemented in  
346 the schemas.

347

348 *Hierarchical Codelist*

349

- 350 1. Renamed HierarchicalCodeList to HierarchicalCodelist.  
351 2. This is re-modelled to reflect more accurately the way this is implemented: this is as an  
352 actual hierarchy rather than a set of relational associations from which the hierarchy  
353 can be derived.  
354  
355 3. Code Association is re-named Hierarchical Code and the association type association  
356 to Code is removed (as these association types are not maintained in an Item  
357 Scheme).  
358  
359 4. Hierarchical Code is made an aggregate of Hierarchy, and not of Hierarchical Codelist.  
360  
361 5. Removed root node in the Hierarchy – there can be many top-level codes in  
362 Hierarchical Code.  
363  
364 6. Added reference association between Hierarchical Code and Level to indicate the  
365 Level if the Hierarchy is a level based hierarchy.

366

367 *Provisioning and Registration*

- 368 1. Data Provider and Provision Agreement have an association to Datasource (was  
369 Query Datasource), as the association is to any of Query Datasource and Simple  
370 Datasource.  
371  
372 2. Provision Agreement is made Maintainable and indexing attributes moved to  
373 Registration  
374  
375 3. Registration has a registry assigned Id and indexing attributes.

## 376 1 Introduction

377 This document is not normative, but provides a detailed view of the information model on  
378 which the normative SDMX specifications are based. Those new to the UML notation or to the  
379 concept of Data Structure Definitions may wish to read the appendixes in this document as an  
380 introductory exercise.

### 381 1.1 Related Documents

382 This document is one of two documents concerned with the SDMX Information Model. The  
383 complete set of documents is:

384  
385 SDMX SECTION 02 INFORMATION MODEL: UML CONCEPTUAL DESIGN (this document)

386  
387 This document comprises the complete definition of the information model, with the exception  
388 of the registry interfaces. It is intended for technicians wishing to understand the complete  
389 scope of the SDMX technical standards in a syntax neutral form.

390

391 SDMX SECTION 05 REGISTRY SPECIFICATION: LOGICAL INTERFACES

392

393 This document provides the logical specification for the registry interfaces, including  
394 subscription/notification, registration/submission of data and metadata, and querying.

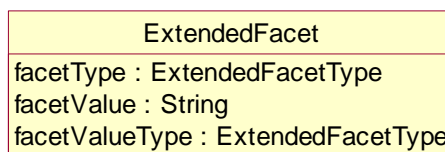
### 395 1.2 Modelling Technique and Diagrammatic Notes

396 The modelling technique used for the SDMX Information Model (SDMX-IM) is the Unified  
397 Modelling Language (UML). An overview of the constructs of UML that are used in the SDMX-  
398 IM can be found in the Appendix "A Short Guide to UML in the SDMX Information Model"

399

400 UML diagramming allows a class to be shown with or without the compartments for one or  
401 both of attributes and operations (sometimes called methods). In this document the operations  
402 compartment is not shown as there are no operations.

403



**Figure 1 Class with operations suppressed**

404

405 In some diagrams for some classes the attribute compartment is suppressed even though  
406 there may be some attributes. This is deliberate and is done to aid clarity of the diagram. The  
407 method used is:

408

409 • The attributes will always be present on the class diagram where the class is defined  
410 and its attributes and associations are defined.

411 • On other diagrams, such as inheritance diagrams, the attributes may be suppressed  
412 from the class for clarity.

413

ExtendedFacet

**Figure 2 Class with attributes also suppressed**

414  
415  
416  
417  
418  
419

Note that, in any case, attributes inherited from a super class are not shown in the sub class.

The following table structure is used in the definition of the classes, attributes, and associations.

Class	Feature	Description
ClassName		
	attributeName	.
	associationName	
	+roleName	

420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430

The content in the “Feature” column comprises or explains one of the following structural features of the class:

- Whether it is an abstract class. Abstract classes are shown in *italic Courier* font
- The superclass this class inherits from, if any
- The sub classes of this class, if any
- Attribute – the `attributeName` is shown in *Courier* font
- Association – the `associationName` is shown in *Courier* font. If the association is derived from the association between super classes then the format is `/associationName`
- Role – the `+roleName` is shown in *Courier* font

432  
433  
434  
435

The Description column provides a short definition or explanation of the Class or Feature. UML class names may be used in the description and if so, they are presented in normal font with spaces between words. For example the class `ConceptScheme` will be written as Concept Scheme.

### 436 **1.3 Overall Functionality**

#### 437 **1.3.1 Information Model Packages**

438 The SDMX Information Model (SDMX-IM) is a conceptual metamodel from which syntax  
439 specific implementations are developed. The model is constructed as a set of functional  
440 packages which assist in the understanding, re-use and maintenance of the model.  
441

442 In addition to this, in order to aid understanding each package can be considered to be in one  
443 of three conceptual layers:

- 444
- 445 • the SDMX Base layer comprises fundamental building blocks which are used by the  
446 Structural Definitions layer and the Reporting and Dissemination layer
  - 447 • the Structural Definitions layer comprises the definition of the structural artefacts  
448 needed to support data and metadata reporting and dissemination
  - 449 • the Reporting and Dissemination layer comprises the definition of the data and  
450 metadata containers used for reporting and dissemination

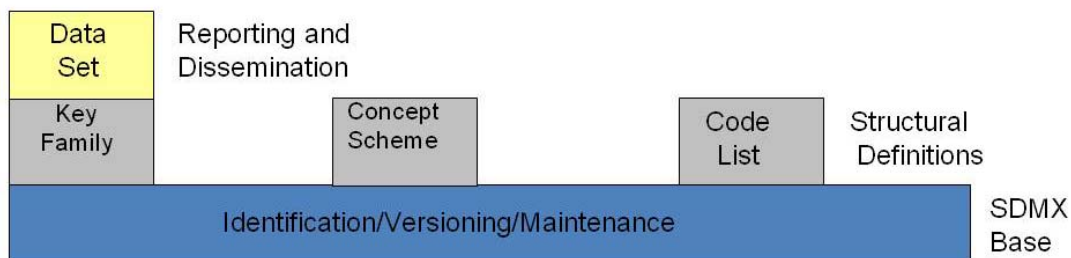
451 In reality the layers have no implicit or explicit structural function as any package can make  
452 use of any construct in another package.

### 453 1.3.2 Version 1.0

454 In version 1.0 the metamodel supported the requirements for:

- 455
- 456 • Data Structure Definition definition including (domain) category scheme, (metadata)  
457 concept scheme, and code list
  - 458
  - 459 • Data and related metadata reporting and dissemination

460 The SDMX-IM comprises a number of packages. These packages act as convenient  
461 compartments for the various sub models in the SDMX-IM. The diagram below shows the sub  
462 models of the SDMX-IM that were included in the version 1.0 specification.



463  
464 **Figure 3: SDMX Information Model Version 1.0 package structure**

### 465 1.3.3 Version 2.0/2.1

466 The version 2.0/2.1 model extends the functionality of version 1.0. principally in the area of  
467 metadata, but also in various ways to define structures to support data analysis by systems  
468 with knowledge of cube type structures such as OLAP<sup>1</sup> systems. The following major  
469 constructs have been added at version 2.0/2.1

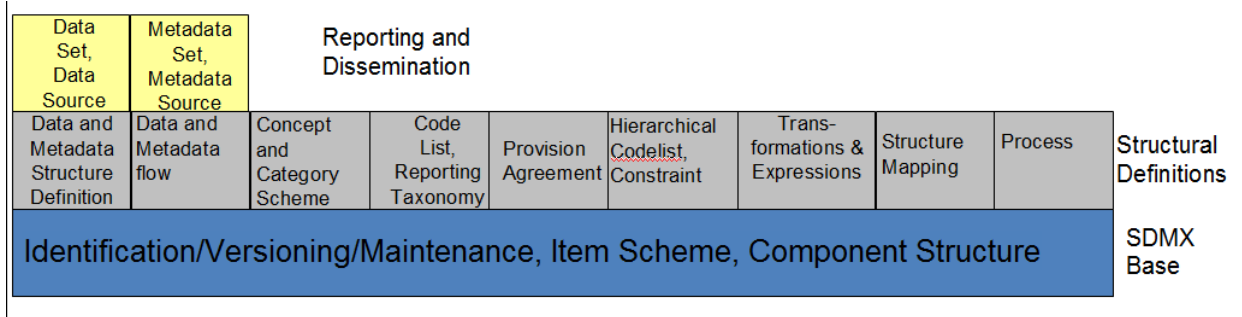
- 470
- 471 • Metadata structure definition
  - 472 • Metadata set

---

<sup>1</sup> OLAP: On line analytical processing

- 473 • Hierarchical Codelist
- 474 • Data and Metadata Provisioning
- 475 • Process
- 476 • Mapping
- 477 • Constraints
- 478 • Constructs supporting the Registry

479 Furthermore, the term Data Structure Definition replaces the term Key Family: as both of these  
 480 terms are used in various communities they are synonymous. The term Data Structure  
 481 Definition is used in the model and this document.

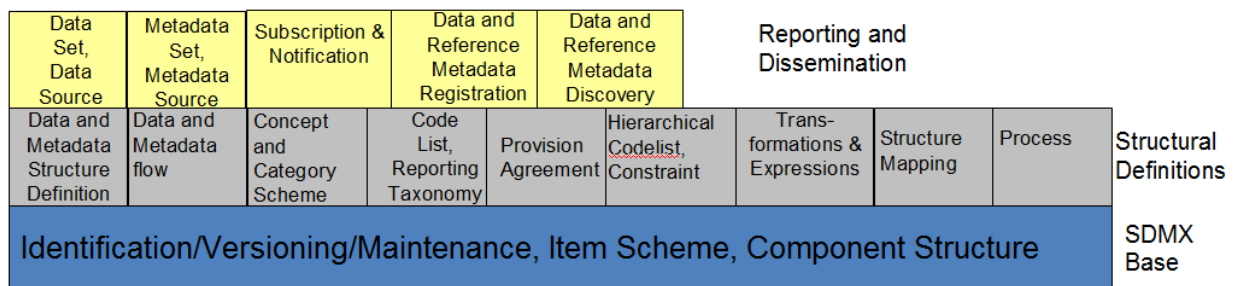


**Figure 4 SDMX Information Model Version 2.0/2.1 package structure**

482 Additional constructs that are specific to a registry based scenario can be found in the  
 483 Specification of Registry Interfaces. For information these are shown on the diagram below  
 484 and comprise:

- 485
- 486 • Subscription and Notification
- 487 • Registration
- 488 • Discovery

489 Note that the data and metadata required for registry functions are not confined to the registry,  
 490 and the registry also makes use of the other packages in the Information Model.



491  
 492 **Figure 5: SDMX Information Model Version 2.0/2.1 package structure including the registry**

## 493 **2 Actors and Use Cases**

### 494 **2.1 Introduction**

495 In order to develop the data models it is necessary to understand the functions to be  
496 supported resulting from the requirements definition. These are defined in a use case model.  
497 The use case model comprises actors and use cases and these are defined below.

498

#### 499 **Actor**

500 *“An actor defines a coherent set of roles that users of the system can play when interacting*  
501 *with it. An actor instance can be played by either an individual or an external system”*

502

#### 503 **Use case**

504 *“A use case defines a set of use-case instances, where each instance is a sequence of*  
505 *actions a system performs that yields an observable result of value to a particular actor”*

506

507 The overall intent of the model is to support data and metadata reporting, dissemination, and  
508 exchange in the field of aggregated statistical data and related metadata. In order to achieve  
509 this, the model needs to support three fundamental aspects of this process:

510

- 511 • Maintenance of structural and provisioning definitions
- 512 • Data and reference metadata publishing (reporting), and consuming (using)
- 513 • Access to data, reference metadata, and structural and provisioning definitions

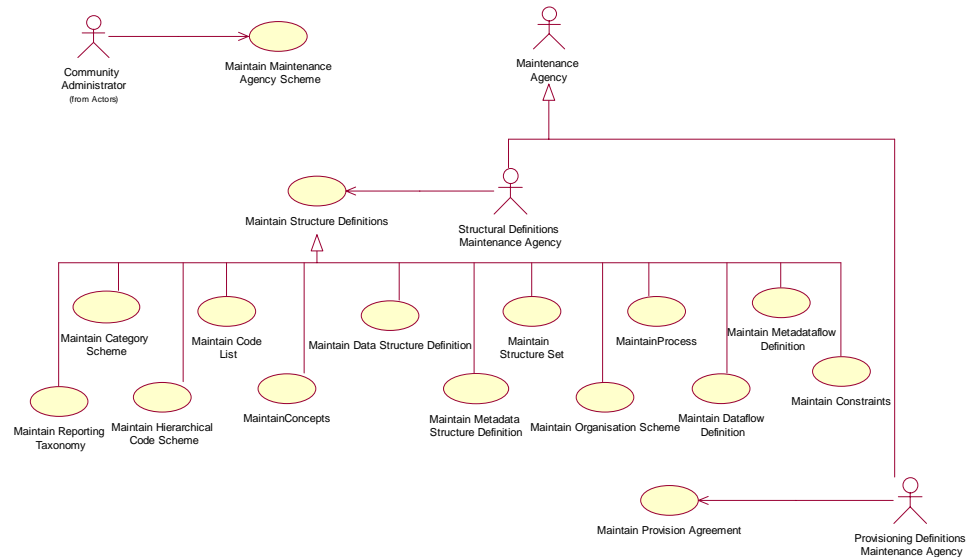
514 This document covers the first two aspects, whilst the document on the Registry logical model  
515 covers the last aspect.

516 **2.2 Use Case Diagrams**

517 **2.2.1 Maintenance of Structural and Provisioning Definitions**

518 **2.2.1.1 Use cases**

519



**Figure 6 Use cases for maintaining data and metadata structural and provisioning definitions**

520 **2.2.1.2 Explanation of the Diagram**

521 In order for applications to publish and consume data and reference metadata it is necessary  
 522 for the structure and permitted content of the data and reference metadata to be defined and  
 523 made available to the applications, as well as definitions that support the actual process of  
 524 publishing and consuming. This is the responsibility of a Maintenance Agency.




525  
 526 All maintained artefacts are maintained by a Maintenance Agency. For convenience the  
 527 Maintenance Agency actor is sub divided into two actor roles:

- 528  
 529
  - maintaining structural definitions  
 530
  - maintaining provisioning definitions







531 Whilst both these functions may be carried out by the same person, or at least by the same  
 532 maintaining organization, the purpose of the definitions is different and so the roles have been  
 533 differentiated: structural definitions define the format and permitted content of data and  
 534 reference metadata when reported or disseminated, whilst provisioning definitions support the  
 535 process of reporting and dissemination (who reports what to whom, and when).









536  
 537 In a community based scenario where at least the structural definitions may be shared, it is  
 538 important that the scheme of maintenance agencies is maintained by a responsible  
 539 organization (called here the Community Administrator), as it is important that the Id of the  
 540 Maintenance Agency is unique.


541 **2.2.1.3 Definitions**

Actor	Use Case	Description
 Community Administrator		Responsible organisation that administers structural definitions common to the community as a whole.
	 Maintain Maintenance Agency Scheme	Creation and maintenance of the top-level scheme of maintenance agencies for the Community.
 Maintenance Agency		Responsible agency for maintaining structural artefacts such as code lists, concept schemes, Data Structure Definition structural definitions, metadata structure definitions, data and metadata provisioning artefacts such as provision



Actor	Use Case	Description
		<p>agreement, and sub-maintenance agencies.</p> <p>sub roles are:</p> <p>Structural Definitions Maintenance Agency</p> <p>Provisioning Definitions Maintenance Agency</p>
 <p>Structural Definitions Maintenance Agency</p>		<p>Responsible for maintaining structural definitions.</p>
	 <p>Maintain Structure Definitions</p>	<p>The maintenance of structural definitions. This use case has sub class use cases for each of the structural artefacts that are maintained.</p>
	 <p>Maintain Code List</p>  <p>MaintainConcepts</p>  <p>Maintain Category Scheme</p>  <p>Maintain Data Structure Definition</p>	<p>Creation and maintenance of the Data Structure Definition, Metadata Structure Definition, and the supporting artefacts that they use, such as code list and concepts</p>

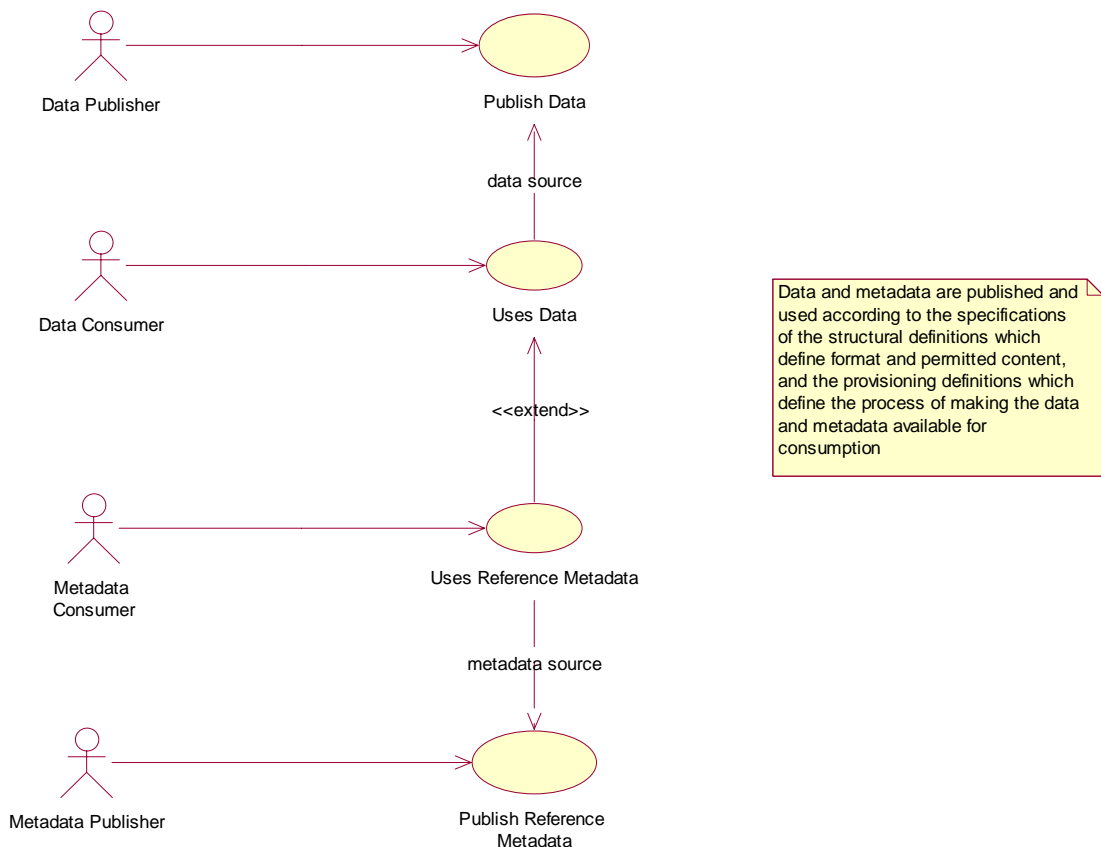
Actor	Use Case	Description
	<p style="text-align: center;"> Maintain Metadata Structure Definition</p> <p style="text-align: center;"> Maintain Hierarchical Code Scheme</p> <p style="text-align: center;"> Maintain Reporting Taxonomy</p> <p style="text-align: center;"> Maintain Organisation Scheme</p> <p style="text-align: center;"> MaintainProcess</p> <p style="text-align: center;"> Maintain Dataflow Definition</p> <p style="text-align: center;"> Maintain Metadataflow Definition</p>	<p>This includes Agency, Data Provider, Data Consumer, and Organisation Unit Scheme</p>
<p style="text-align: center;"> Provisioning Definitions Maintenance Agency</p>		<p>Responsible for maintaining data and metadata provisioning definitions.</p>

Actor	Use Case	Description
	 Maintain Provision Agreement	The maintenance of provisioning definitions.

542 **Figure 7: Table of Actors and Use Cases for Maintenance of Structural and Provisioning Definitions**

543 **2.2.2 Publishing and Using Data and Reference Metadata**

544 **2.2.2.1 Use Cases**



545 **Figure 8: Actors and use cases for data and metadata publishing and consuming**  
 546

547 **2.2.2.2 Explanation of the Diagram**



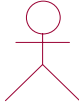

548 Note that in this diagram “publishing” data and reference metadata is deemed to be the same  
 549 as “reporting” data and reference metadata. In some cases the act of making the data  
 550 available fulfils both functions. Aggregated data is published and in order for the Data  
 551 Publisher to do this and in order for consuming applications to process the data and reference  
 552 metadata its structure must be known. Furthermore, consuming applications may also require  
 553 access to reference metadata in order to present this to the Data Consumer so that the data is  
 554 better understood. As with the data, the reference metadata also needs to be formatted in  
 555 accordance with a maintained structure. The Data Consumer and Metadata Consumer cannot

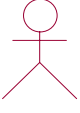



556 use the data or reference metadata unless it is “published” and so there is a “data source” or  
 557 “metadata source” dependency between the “uses” and “publish” use cases.

558

559 In any data and reference metadata publishing and consuming scenario both the publishing  
 560 and the consuming applications will need access to maintained Provisioning Definitions.  
 561 These definitions may be as simple as who provides what data and reference metadata to  
 562 whom, and when, or it can be more complex with constraints on the data and metadata that  
 563 can be provided by a particular publisher, and, in a data sharing scenario where data and  
 564 metadata are “pulled” from data sources, details of the source.

565 **2.2.2.3 Definitions**

Actor	Use Case	Description
 Data Publisher		Responsible for publishing data according to a specified Data Structure Definition (data structure) definition, and relevant provisioning definitions.
	 Publish Data	Publish a data set. This could mean a physical data set or it could mean to make the data available for access at a data source such as a database that can process a query.
 Data Consumer		The user of the data. It may be a human consumer accessing via a user interface, or it could be an application such as a statistical production system.
	 Uses Data	Use data that is formatted according to the structural definitions and made available according to the provisioning definitions. Data are often linked to metadata that may reside in a different location and be published and maintained independently.

Actor	Use Case	Description
 Metadata Publisher		Responsible for publishing reference metadata according to a specified metadata structure definition, and relevant provisioning definitions.
	 Publish Reference Metadata	Publish a reference metadata set. This could mean a physical metadata set or it could mean to make the reference metadata available for access at a metadata source such as a metadata repository that can process a query.
 Metadata Consumer		The user of the reference metadata. It may be a human consumer accessing via a user interface, or it could be an application such as a statistical production or dissemination system.
	 Uses Reference Metadata	Use reference metadata that is formatted according to the structural definitions and made available according to the provisioning definitions.

## 567 **3 SDMX Base Package**

### 568 **3.1 Introduction**

569 The constructs in the SDMX Base package comprise the fundamental building blocks that  
570 support many of the other structures in the model. For this reason, many of the classes in this  
571 package are abstract (i.e. only derived sub-classes can exist in an implementation).

572  
573 The motivation for establishing the SDMX Base package is as follows:

- 574 • it is accepted “Best Practise” to identify fundamental archetypes occurring in a model
- 575 • identification of commonly found structures or “patterns” leads to easier understanding
- 576 • identification of patterns encourages re-use
- 577 • identification of patterns encourages re-use

578 Each of the class diagrams in this section views classes from the SDMX Base package from a  
579 different perspective. There are detailed views of specific patterns, plus overviews showing  
580 inheritance between classes, and relationships amongst classes.

581

582 **3.2 Base Structures - Identification, Versioning, and Maintenance**

583 **3.2.1 Class Diagram**

584

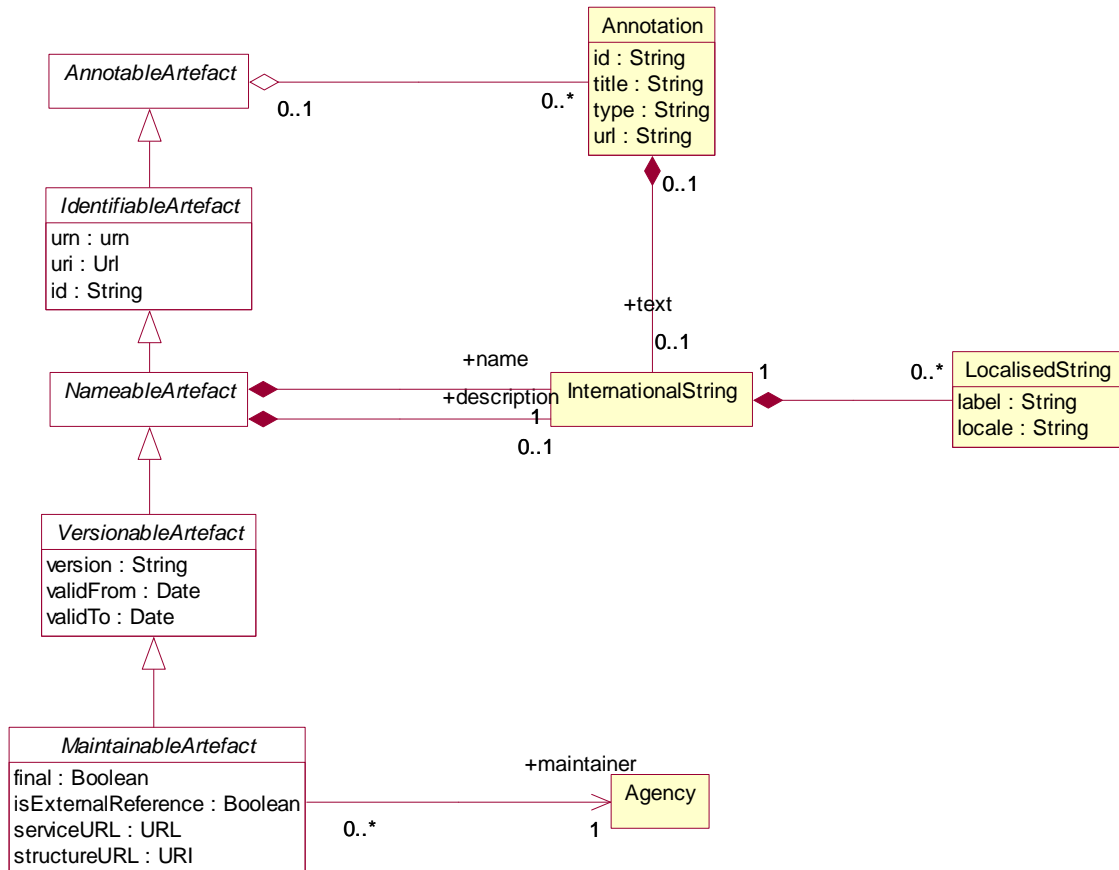


Figure 9: SDMX Identification, Maintenance and Versioning

585 **3.2.2 Explanation of the Diagram**

586 **3.2.2.1 Narrative**

587 This group of classes forms the nucleus of the administration facets of SDMX objects. They  
 588 provide features which are reusable by derived classes to support horizontal functionality such  
 589 as identity, versioning etc.

590

591 All classes derived from the abstract class *AnnotableArtefact* may have Annotations (or  
 592 notes): this supports the need to add notes to all SDMX-ML elements. The Annotation is used  
 593 to convey extra information to describe any SDMX construct. This information may be in the  
 594 form of a URL reference and/or a multilingual text (represented by the association to  
 595 *InternationalString*).

596

597 The *IdentifiableArtefact* is an abstract class that comprises the basic attributes  
598 needed for identification. Concrete classes based on *IdentifiableArtefact* all inherit the  
599 ability to be uniquely identified.

600

601 The *NamableArtefact* is an abstract class that inherits from *IdentifiableArtefact*  
602 and in addition the +description and +name roles support multilingual descriptions and  
603 names for all objects based on *NameableArtefact*. The *InternationalString* supports  
604 the representation of a description in multiple locales (locale is similar to language but includes  
605 geographic variations such as Canadian French, US English etc.). The *LocalisedString*  
606 supports the representation of a description in one locale.

607

608 *VersionableArtefact* is an abstract class which inherits from *NameableArtefact* and  
609 adds versioning ability to all classes derived from it.

610

611 *MaintainableArtefact* further adds the ability for derived classes to be maintained via its  
612 association to *Agency*, and adds locational information (i.e. from where the object can be  
613 retrieved). It is possible to define whether the artefact is draft or final with the *final* attribute.

614

615 The inheritance chain from *AnnotableArtefact* through to *MaintainableArtefact*  
616 allows SDMX classes to inherit the features they need, from simple annotation, through  
617 identity, naming, to versioning and maintenance.

618

### 619 3.2.2.2 Definitions

Class	Feature	Description
<i>AnnotableArtefact</i>	Base inheritance sub classes are: <i>IdentifiableArtefact</i>	Objects of classes derived from this can have attached annotations.
Annotation		Additional descriptive information attached to an object.
	id	Identifier for the Annotation. It can be used to disambiguate one Annotation from another where there are several Annotations for the same annotated object.
	title	A title used to identify an annotation.
	type	Specifies how the annotation is to be processed.
	url	A link to external descriptive text.
	+text	An <i>InternationalString</i> provides the multilingual text content of the annotation via this role.

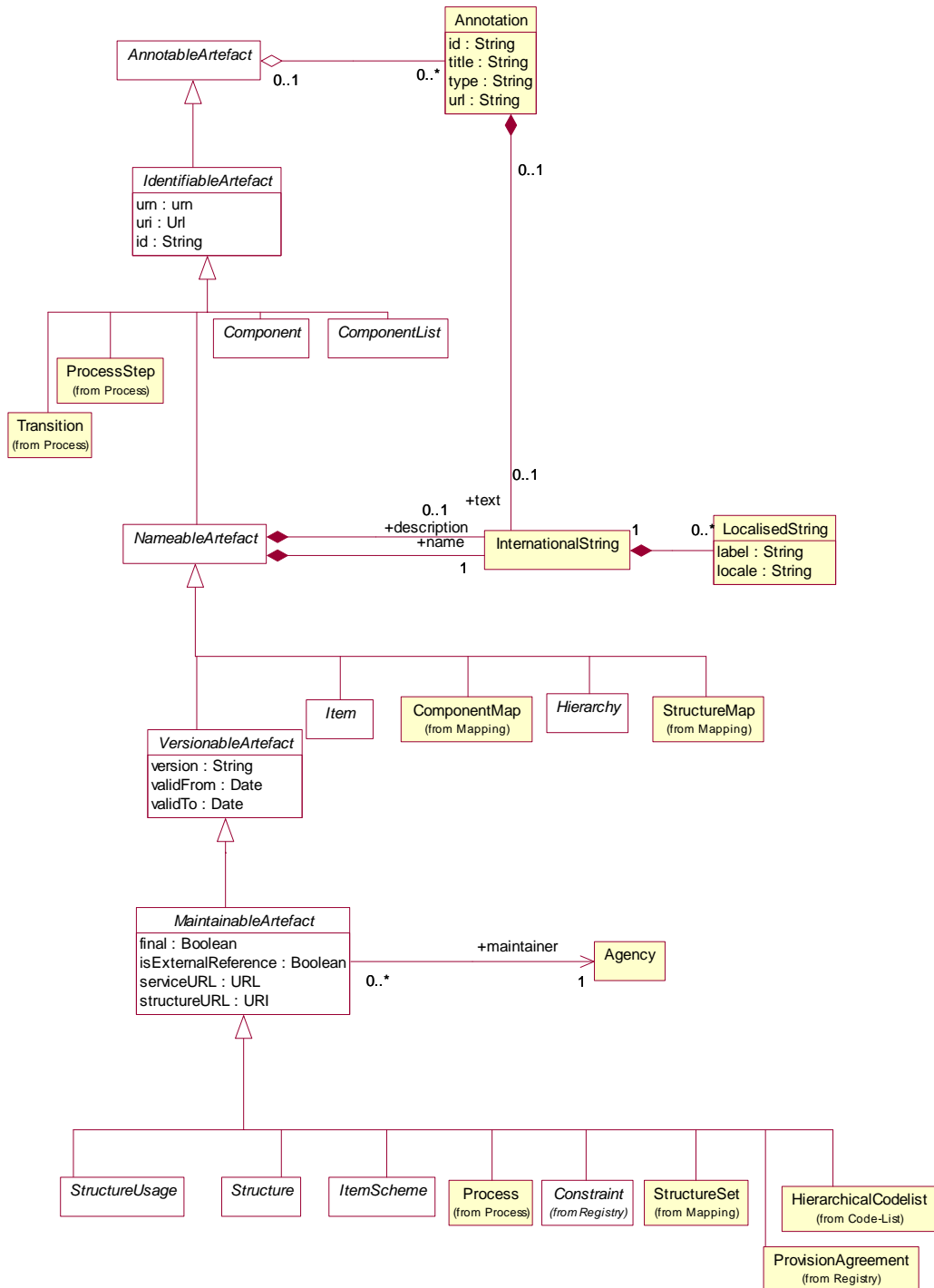


Class	Feature	Description
<i>IdentifiableArtefact</i>	<p>Superclass is <i>AnnotableArtefact</i></p> <p>Base inheritance sub classes are: <i>NameableArtefact</i></p>	Provides identity to all derived classes. It also provides annotations to derived classes because it is a subclass of Annotable Artefact.
	id	The unique identifier of the object.
	uri	Universal resource identifier that may or may not be resolvable.
	urn	Universal resource name – this is for use in registries: all registered objects have a urn.
<i>NameableArtefact</i>	<p>Superclass is <i>IdentifiableArtefact</i></p> <p>Base inheritance sub classes are: <i>VersionableArtefact</i></p>	Provides a Name and Description to all derived classes in addition to identification and annotations.
	+description	A multi-lingual description is provided by this role via the International String class.
	+name	A multi-lingual name is provided by this role via the International String class
InternationalString		The International String is a collection of Localised Strings and supports the representation of text in multiple locales.
LocalisedString		The Localised String supports the representation of text in one locale (locale is similar to language but includes geographic variations such as Canadian French, US English etc.).
	label	Label of the string.
	locale	The geographic locale of the string e.g French, Canadian French.

Class	Feature	Description
<i>VersionableArtefact</i>	Superclass is <i>NameableArtefact</i> Base inheritance sub classes are: <i>MaintainableArtefact</i>	Provides versioning information for all derived objects.
	version	A version string following an agreed convention
	validFrom	Date from which the version is valid
	validTo	Date from which version is superceded
<i>MaintainableArtefact</i>	Inherits from <i>VersionableArtefact</i>	An abstract class to group together primary structural metadata artefacts that are maintained by an Agency.
	final	Defines whether a maintained artefact is draft or final.
	isExternalReference	If set to "true" it indicates that the content of the object is held externally.
	structureURL	The URL of an SDMX-ML document containing the external object.
	serviceURL	The URL of an SDMX-compliant web service from which the external object can be retrieved.
	+maintainer	Association to the Maintenance Agency responsible for maintaining the artefact.
Agency		See section on "Organisations"

621 **3.3 Basic Inheritance**

622 **3.3.1 Class Diagram– Basic Inheritance from the Base Inheritance Classes**



623

624

**Figure 10: Basic Inheritance from the Base Structures**

625 **3.3.2 Explanation of the Diagram**

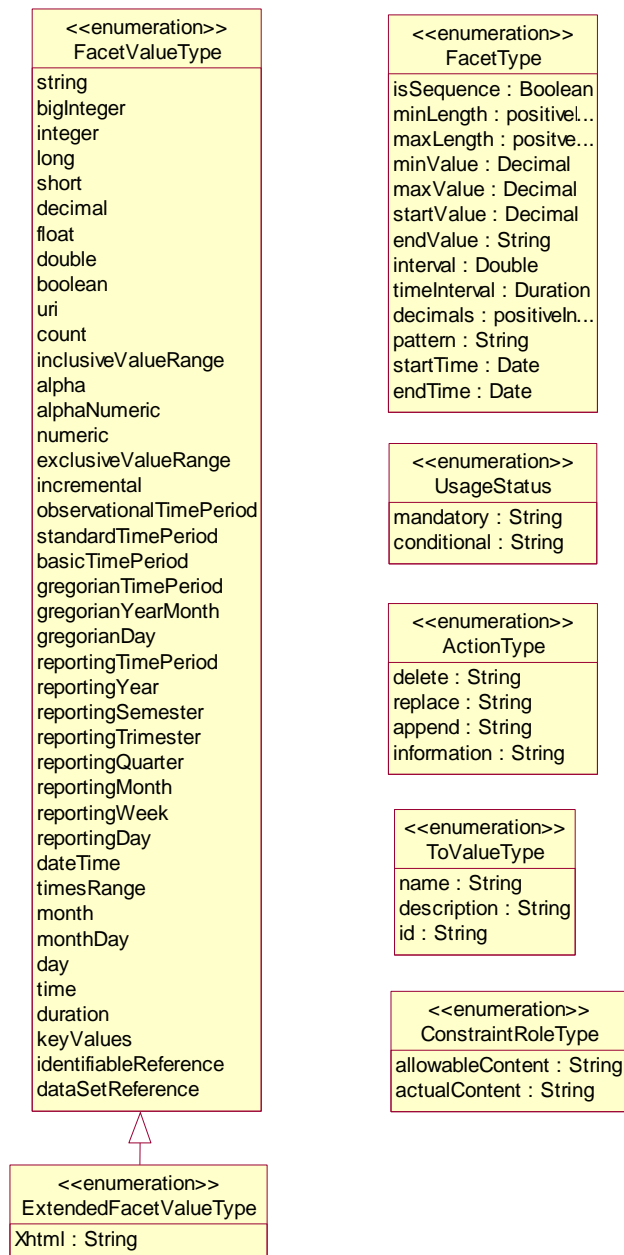
626 **3.3.2.1 Narrative**

627 The diagram above shows the inheritance within the base structures. The concrete classes  
628 are introduced and defined in the specific package to which they relate.

629 **3.4 Data Types**

630 **3.4.1 Class Diagram**

631



632

**Figure 11: Class Diagram of Basic Data Types**

633 **3.4.2 Explanation of the Diagram**

634 **3.4.2.1 Narrative**

635 The `UsageStatus` enumeration is used as a data type on a `DataAttribute` where the  
636 value of the attribute in an instance of the class must take one of the values in the  
637 `UsageStatus` (i.e. mandatory, conditional).

638  
639 The `FacetType` and `FacetValueType` enumerations are used to specify the valid format of  
640 the content of a non enumerated `Concept` or the usage of a `Concept` when specified for use  
641 on a `Component` on a `Structure` (such as a `Dimension` in a  
642 `DataStructureDefinition`). The description of the various types can be found in the  
643 section on `ConceptScheme` (section 4.4).

644

645 The `ActionType` enumeration is used to specify the action that a receiving system should  
646 take when processing the content that is the object of the action. It is enumerated as follows:

647

- Append

648

649  
650 Data or metadata is an incremental update for an existing data/metadata set or the  
651 provision of new data or documentation (attribute values) formerly absent. If any of the  
652 supplied data or metadata is already present, it will not replace that data or metadata. This  
653 corresponds to the "Update" value found in version 1.0 of the SDMX Technical Standards

654

- Replace

655

656  
657 Data/metadata is to be replaced, and may also include additional data/metadata to be  
658 appended.

659

- Delete

660

661  
662 Data/Metadata is to be deleted.

663

- Information

664

665  
666 Data and metadata are for information purposes.

667

668 The `IdentifiableObjectType` enumeration is used to specify an object type whose class  
669 is a sub class of `IdentifiableArtefact` either directly or via `NameableArtefact`,  
670 `VersionableArtefact` or `MaintainableArtefact`.

671

672 The `ToValueType` data type contains the attributes to support transformations defined in the  
673 `StructureMap` (see Section 9).

674

675 The `ConstraintRoleType` data type contains the attributes that identify the purpose of a  
676 `Constraint` (`allowableContent`, `actualContent`).

677 **3.5 The Item Scheme Pattern**

678 **3.5.1 Context**

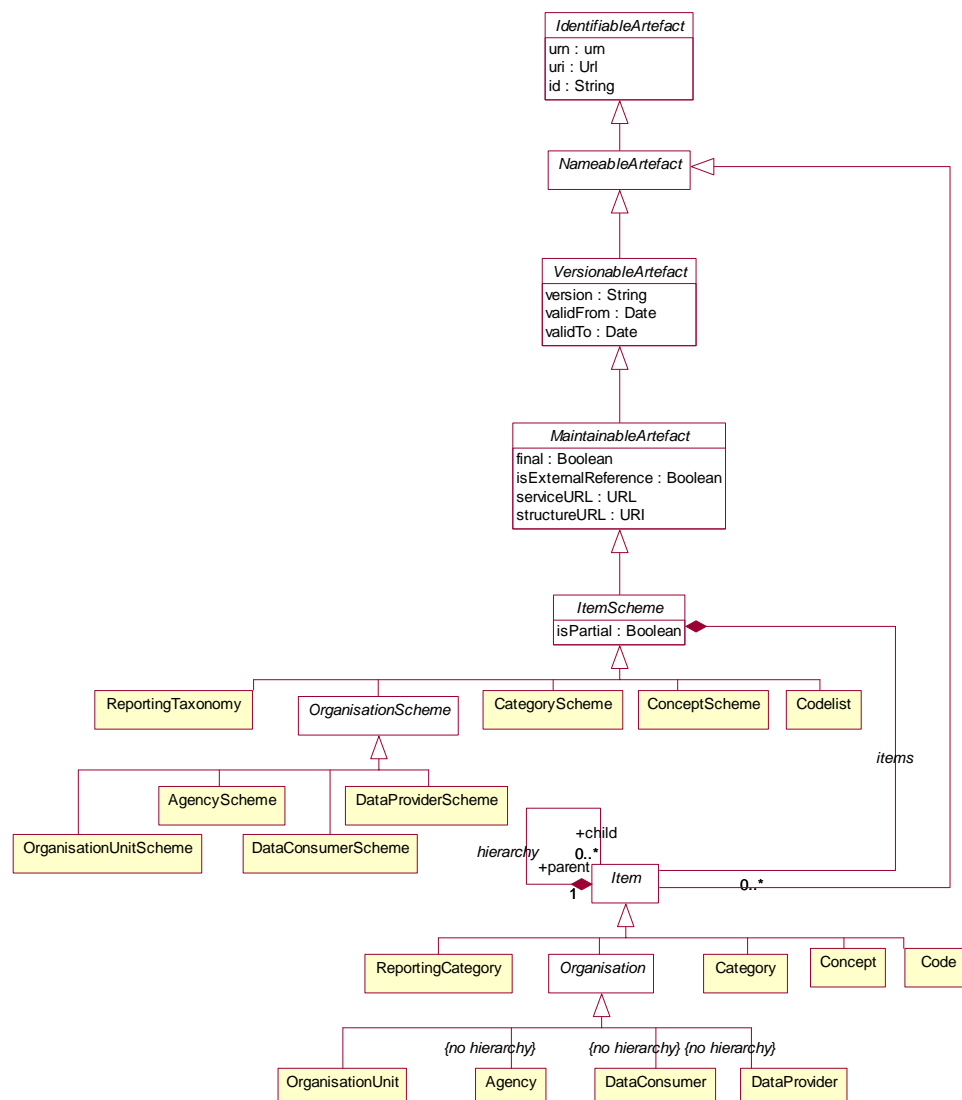
679 The Item Scheme is a basic architectural pattern that allows the creation of list schemes for  
 680 use in simple taxonomies, for example.

681

682 The `ItemScheme` is the basis for `CategoryScheme`, `Codelist`, `ConceptScheme`,  
 683 `ReportingTaxonomy`, and `OrganisationScheme`.

684

685 **3.5.2 Class Diagram**



**Figure 12 The Item Scheme pattern**

686 **3.5.3 Explanation of the Diagram**

687 **3.5.3.1 Narratve**

688 The *ItemScheme* is an abstract class which defines a set of *Item* (this class is also abstract).  
 689 Its main purpose is to define a mechanism which can be used to create taxonomies which can  
 690 classify other parts of the SDMX Information Model. It is derived from  
 691 *MaintainableArtefact* which gives it the ability to be annotated, have identity, naming,  
 692 versioning and be associated with an *Agency*. An example of a concrete class is a  
 693 *CategoryScheme*. The associated *Category* are *Items*.

694  
 695 In an exchange environment an *ItemScheme* is allowed to contain a sub-set of the *Items* in  
 696 the maintained *ItemScheme*. If such an *ItemScheme* is disseminated with a sub-set of the  
 697 *Items* then the fact that this is a sub-set is denoted by setting the *isPartial* attribute to  
 698 "true".

699  
 700 A "partial" *ItemScheme* cannot be maintained independently in its partial form i.e. it cannot  
 701 contain *Items* that are not present in the full *ItemScheme* and the content of any one *Item*  
 702 (e.g. names and descriptions) cannot deviate from the content in the full *ItemScheme*.  
 703 Furthermore, the *Id* of the *ItemScheme* where *isPartial* is set to "true" is the same as the  
 704 *Id* of the full *ItemScheme* (maintenance agency, id, version). This is important as this is the *Id*  
 705 that that is referenced in other structures (e.g. a *Codelist* referenced in a *DSD*) and this *Id* is  
 706 always the same, regardless of whether the disseminated *ItemScheme* is the full  
 707 *ItemScheme* or a partial *ItemScheme*.

708  
 709 The purpose of a partial *ItemScheme* is to support the exchange and dissemination of a sub-  
 710 set *ItemScheme* without the need to maintain multiple *ItemSchemes* which contain the same  
 711 *Items*. For instance when a *Codelist* is used in a *DataStructureDefinition* it is  
 712 sometimes the case that only a sub-set of the *Codes* in a *Codelist* are relevant. In this case  
 713 a partial *Codelist* can be constructed using the *Constraint* mechanism explained later in this  
 714 document.

715  
 716 *Item* inherits from *NameableArtefact* which gives it the ability to be annotated and have  
 717 identity, and therefore has *id*, *uri* and *urn* attributes, a name and a description in the form of an  
 718 *InternationalString*. Unlike the parent *ItemScheme*, the *Item* itself is not a  
 719 *MaintainableArtefact* and therefore cannot have an independent *Agency* (i.e. it implicitly  
 720 has the same agency as the *ItemScheme*).

721  
 722 The *Item* can be hierarchic and so one *Item* can have child *Items*. The restriction of the  
 723 hierarchic association is that a child *Item* can have only parent *Item*.

724  
 725 **3.5.3.2 Definitions**

Class	Feature	Description
<i>ItemScheme</i>	Inherits from: <i>MaintainableArtefact</i>  Direct sub classes are: <i>CategoryScheme</i> <i>ConceptScheme</i>	The descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common.

Class	Feature	Description
	CodeList ReportingTaxonomy OrganisationScheme	
	isPartial	Denotes whether the Item Scheme contains a sub set of the full set of Items in the maintained scheme.
	items	Association to the Items in the scheme.
<i>Item</i>	Inherits from: <i>NameableArtefact</i> Direct sub classes are Category Concept Code ReportingCategory <i>Organisation</i>	The Item is an item of content in an Item Scheme. This may be a node in a taxonomy or ontology, a code in a code list etc. Node that at the conceptual level the Organisation is not hierarchic
	hierarchy	This allows an Item optionally to have one or more child Items.

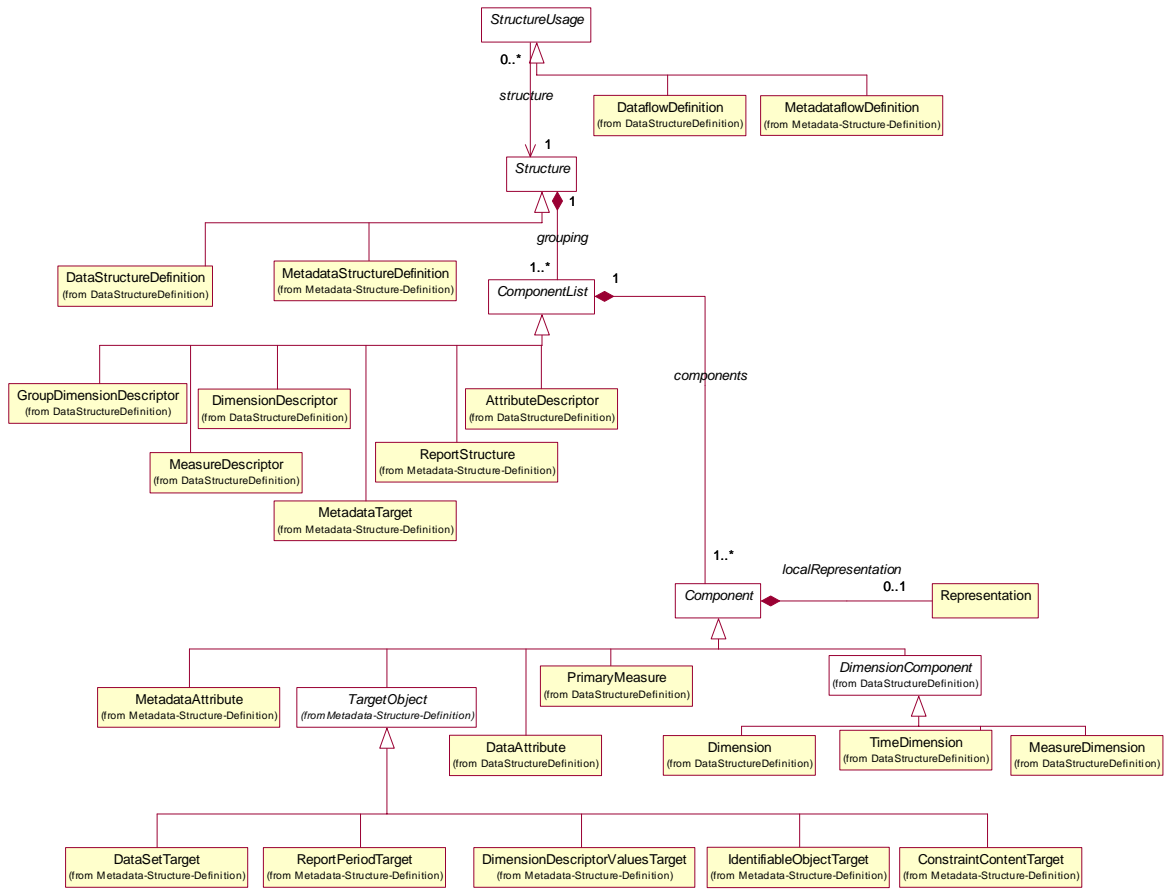
726 **3.6 The Structure Pattern**

727 **3.6.1 Context**

728 The Structure Pattern is a basic architectural pattern which allows the specification of complex  
 729 tabular structures which are often found in statistical data (such as Data Structure Definition,  
 730 and Metadata Structure Definition). A Structure is a set of ordered lists. A pattern to underpin  
 731 this tabular structure has been developed, so that commonalities between these structure  
 732 definitions can be supported by common software and common syntax structures.

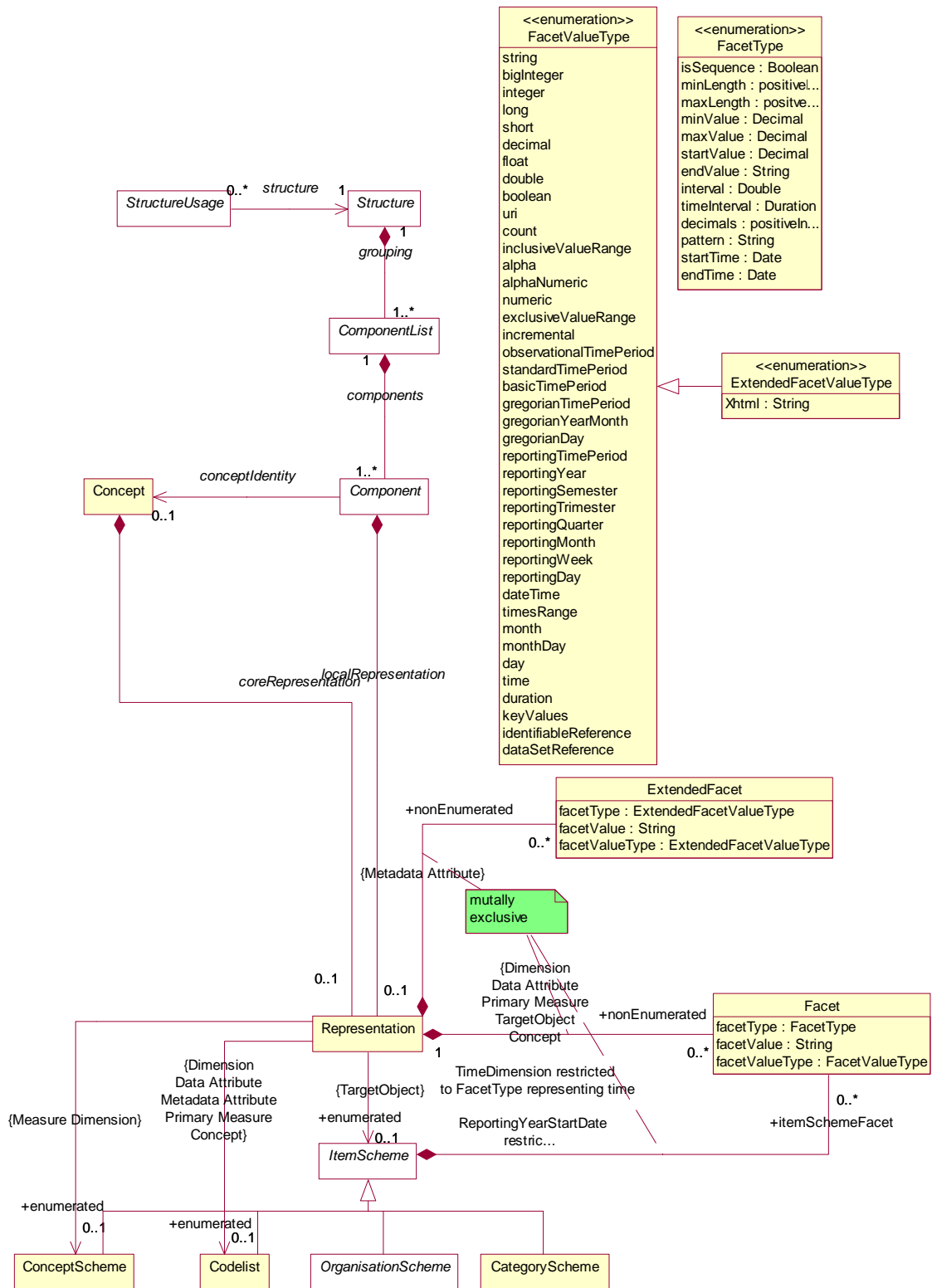


733 3.6.2 Class Diagrams



734  
735

Figure 13: The Structure Pattern



**Figure 14: Representation within the Structure Pattern**

737

738 **3.6.3 Explanation of the Diagrams**

739 **3.6.3.1 Narrative**

740 The *Structure* is an abstract class which contains a set of one or more *ComponentList(s)*  
 741 (this class is also abstract). An example of a concrete *Structure* is  
 742 *DataStructureDefinition*.

743

744 The *ComponentList* is a list of one or more *Component(s)*. The *ComponentList* has  
 745 several concrete descriptor classes based on it: *DimensionDescriptor*,  
 746 *GroupDimensionDescriptor*, *MeasureDescriptor*, and *AttributeDescriptor* of  
 747 the *DataStructureDefinition* and *MetadataTarget*, and *ReportStructure* of the  
 748 *MetaDataStructureDefinition*.

749

750 The *Component* is contained in a *ComponentList*. The type of *Component* in a  
 751 *ComponentList* is dependent on the concrete class of the *ComponentList* as follows:

752

- 753 *DimensionDescriptor*: *Dimension*, *Measure Dimension*, *Time Dimension*
- 754 *GroupDimensionDescriptor*: *Dimension*, *Measure Dimension*, *Time*  
 755 *Dimension*
- 756 *MeasureDescriptor*: *PrimaryMeasure*
- 757 *AttributeDescriptor*: *Data Attribute*
- 758 *MetadataTarget*: *TargetObject* and its sub classes
- 759 *ReportStructure*: *MetadataAttribute*

760

761 Each *Component* takes its semantic (and possibly also its representation) from a *Concept* in  
 762 a *ConceptScheme*. This is represented by the *conceptIdentity* association to *Concept*.

763

764 The *Component* may also have a *localRepresentation*. This allows a concrete class,  
 765 such as *Dimension*, to specify its representation which is local to the *Structure* in which it  
 766 is contained (for *Dimension* this will be *DataStructureDefinition*), and thus overrides  
 767 any *coreRepresentation* specified for the *Concept*.

768

769 The *Representation* can be enumerated or non-enumerated. The valid content of an  
 770 enumerated representation is specified either in an *ItemScheme* which can be one of  
 771 *ConceptScheme*, *Codelist*, *OrganisationScheme*, *CategoryScheme*, and  
 772 *ReportingTaxonomy*. The valid content of a non-enumerated representation is specified as  
 773 one or more *Facet* (for example these may specify minimum and maximum values). For a  
 774 *MetadataAttribute* this is achieved by one of more *Extended Facet* which allows the  
 775 additional representation of XHTML.

776

777 The types of representation that are valid for specific components is expressed in the model  
 778 as a constraint on the association viz:

779

- 780 • The *MeasureDimension* must be enumerated and use a *ConceptScheme*
- 781 • The *Dimension* (but not *MeasureDimension*), *DataAttribute*,  
 782 *PrimaryMeasure*, *MetadataAttribute* may be enumerated and, if so, use a  
 783 *Codelist*

- 784 • The *TargetObject* may be enumerated and, if so, can use any *ItemScheme*  
 785 (*Codelist*, *ConceptScheme*, *OrganisationScheme*, *CategoryScheme*,  
 786 *ReportingTaxonomy*)
- 787 • The *Dimension* (but not *MeasureDimension*), *Data Attribute*,  
 788 *PrimaryMeasure*, *TargetObject* may be non-enumerated and, if so, use one of  
 789 more *Facet*, note that the *FacetValueType* applicable to the *TimeDimension*  
 790 is restricted to those that represent time
- 791 • The *MetadataAttribute* may be non-enumerated and, if so, uses one or more  
 792 *ExtendedFacet*

793

794 The *Structure* may be used by one or more *StructureUsage*. An example of this in terms  
 795 of concrete classes is that a *DataflowDefinition* (sub class of *StructureUsage*) may  
 796 use a particular *DataStructureDefinition* (sub class of *Structure*), and similar  
 797 constructs apply for the *MetadataflowDefinition* (link to  
 798 *MetadataStructureDefinition*).

799 **3.6.3.2 Definitions**

Class	Feature	Description
<i>StructureUsage</i>	Inherits from: <i>MaintainableArtefact</i>  Sub classes are: <i>DataflowDefinition</i> <i>MetadataflowDefinition</i>	An artefact whose components are described by a <i>Structure</i> . In concrete terms (sub-classes) an example would be a <i>Dataflow Definition</i> which is linked to a given structure – in this case the <i>Data Structure Definition</i> .
	structure	An association to a <i>Structure</i> specifying the structure of the artefact.
<i>Structure</i>	Inherits from: <i>MaintainableArtefact</i>  Sub classes are: <i>DataStructure Definition</i> <i>MetadataStructure Definition</i>	Abstract specification of a list of lists to define a complex tabular structure. A concrete example of this would be statistical concepts, code lists, and their organisation in a data or metadata structure definition, defined by a centre institution, usually for the exchange of statistical information with its partners.
	grouping	A composite association to one or more component lists.

Class	Feature	Description
<i>ComponentList</i>	<p>Inherits from: <i>IdentifiableArtefact</i></p> <p>Sub classes are: DimensionDescriptor GroupDimension Descriptor MeasureDescriptor AttributeDescriptor MetadataTarget ReportStructure</p>	An abstract definition of a list of components. A concrete example is a Dimension Descriptor which defines the list of Dimensions in a Data Structure Definition.
	components	An aggregate association to one or more components which make up the list.
<i>Component</i>	<p>Inherits from: <i>IdentifiableArtefact</i></p> <p>Sub classes are: PrimaryMeasure DataAttribute <i>DimensionComponent</i> <i>TargetObject</i> MetadataAttribute</p>	A component is an abstract super class used to define qualitative and quantitative data and metadata items that belong to a Component List and hence a Structure. Component is refined through its sub-classes.
	conceptIdentity	Association to a Concept in a Concept Scheme that identifies and defines the semantic of the Component
	localRepresentation	Association to the Representation of the Component if this is different from the coreRepresentation of the Concept which the Component uses (ConceptUsage)
Representation		The allowable value or format for Component or Concept

Class	Feature	Description
	+enumerated	Association to an enumerated list that contains the allowable content for the Component when reported in a data or metadata set. The type of enumerated list that is allowed for any concrete Component is shown in the constraints on the association (e.g. Identifier Component can have any of the sub classes of Item Scheme, whereas Measure Dimension must have a Concept Scheme).
	+nonEnumerated	Association to a set of Facets that define the allowable format for the content of the Component when reported in a data or metadata set.
Facet		Defines the format for the content of the Component when reported in a data or metadata set.
	facetType	A specific content type which is constrained by the FacetType enumeration
	facetValueType	The format of the value of a Component when reported in a data or metadata set. This is constrained by the FacetValueType enumeration.
	+itemSchemeFacet	Defines the format of the identifiers in an Item Scheme used by a Component. Typically this would define the number of characters (length) of the identifier.
ExtendedFacet		This has the same function as Facet but allows additionally an XHTML representation. This is constrained for use with a Metadata Attribute

801 The specification of the content and use of the sub classes to ComponentList and  
 802 Component can be found in the section in which they are used  
 803 (DataStructureDefinition and MetadataStructureDefinition)

### 804 3.6.3.3 Representation Constructs

805 The majority of SDMX FacetValueTypes are compatible with those found in XML Schema,  
 806 and have equivalents in most current implementation platforms:  
 807

SDMX Facet Value Type	XML Schema Data Type	.NET Framework Type	Java Data Type
String	xsd:string	System.String	java.lang.String
Big Integer	xsd:integer	System.Decimal	java.math.BigInteger
Integer	xsd:int	System.Int32	int
Long	xsd:long	System.Int64	long
Short	xsd:short	System.Int16	short
Decimal	xsd:decimal	System.Decimal	java.math.BigDecimal
Float	xsd:float	System.Single	float
Double	xsd:double	System.Double	double
Boolean	xsd:boolean	System.Boolean	boolean
URI	xsd:anyURI	System.Uri	Java.net.URI or java.lang.String
DateTime	xsd:dateTime	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
Time	xsd:time	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
GregorianYear	xsd:gYear	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
GregorianMonth	xsd:gYearMonth	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
GregorianDay	xsd:date	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
Day, MonthDay, Month	xsd:g*	System.DateTime	javax.xml.datatype.XMLGregorianCalendar
Duration	xsd:duration	System.TimeSpan	javax.xml.datatype.Duration

808  
 809 There are also a number of SDMX data types which do not have these direct  
 810 correspondences, often because they are composite representations or restrictions of a  
 811 broader data type. These are detailed in Section 6 of the standards.

812  
 813 The Representation is composed of Facets, each of which conveys characteristic  
 814 information related to the definition of a value domain. Often a set of Facets are needed to  
 815 convey the required semantic. For example, a sequence is defined by a minimum of two  
 816 Facets: one to define the start value, and one to define the interval.

817

Facet Type	Explanation
isSequence	The isSequence facet indicates whether the values are intended to be ordered, and it may work in combination with the interval, startValue, and endValue facet or the timeInterval, startTime, and endTime, facets. If this attribute holds a value of true, a start value or time and a numeric or time interval must be supplied. If an end value is not given, then the sequence continues indefinitely.
interval	The interval attribute specifies the permitted interval (increment) in a

	sequence. In order for this to be used, the isSequence attribute must have a value of true.
startValue	The startValue facet is used in conjunction with the isSequence and interval facets (which must be set in order to use this facet). This facet is used for a numeric sequence, and indicates the starting point of the sequence. This value is mandatory for a numeric sequence to be expressed.
endValue	The endValue facet is used in conjunction with the isSequence and interval facets (which must be set in order to use this facet). This facet is used for a numeric sequence, and indicates that ending point (if any) of the sequence.
timeInterval	The timeInterval facet indicates the permitted duration in a time sequence. In order for this to be used, the isSequence facet must have a value of true.
startTime	The startTime facet is used in conjunction with the isSequence and timeInterval facets (which must be set in order to use this facet). This attribute is used for a time sequence, and indicates the start time of the sequence. This value is mandatory for a time sequence to be expressed.
endTime	The endTime facet is used in conjunction with the isSequence and timeInterval facets (which must be set in order to use this facet). This facet is used for a time sequence, and indicates that ending point (if any) of the sequence.
minLength	The minLength facet specifies the minimum and length of the value in characters.
maxLength	The maxLength facet specifies the maximum length of the value in characters.
minValue	The minValue facet is used for inclusive and exclusive ranges, indicating what the lower bound of the range is. If this is used with an inclusive range, a valid value will be greater than or equal to the value specified here. If the inclusive and exclusive data type is not specified (e.g. this facet is used with an integer data type), the value is assumed to be inclusive.
maxValue	The maxValue facet is used for inclusive and exclusive ranges, indicating what the upper bound of the range is. If this is used with an inclusive range, a valid value will be less than or equal to the value specified here. If the inclusive and exclusive data type is not specified (e.g. this facet is used with an integer data type), the value is assumed to be inclusive.
decimals	The decimals facet indicates the number of characters allowed after the decimal separator.
pattern	The pattern attribute holds any regular expression permitted in the implementation syntax (e.g. W3C XML Schema).

## 818 4 Specific Item Schemes

### 819 4.1 Introduction

820 The structures that are an arrangement of objects into hierarchies or lists based on  
821 characteristics, and which are maintained as a group inherit from *ItemScheme*. These  
822 concrete classes are:

- 823
- 824 • `Codelist`



- 825      • ConceptScheme
- 826      • CategoryScheme
- 827      • AgencyScheme,                  DataProviderScheme,                  DataConsumerScheme,  
828                  OrganisationUnitScheme    which all inherit from the abstract class  
829                  *OrganisationScheme*
- 830      • Reporting Taxonomy

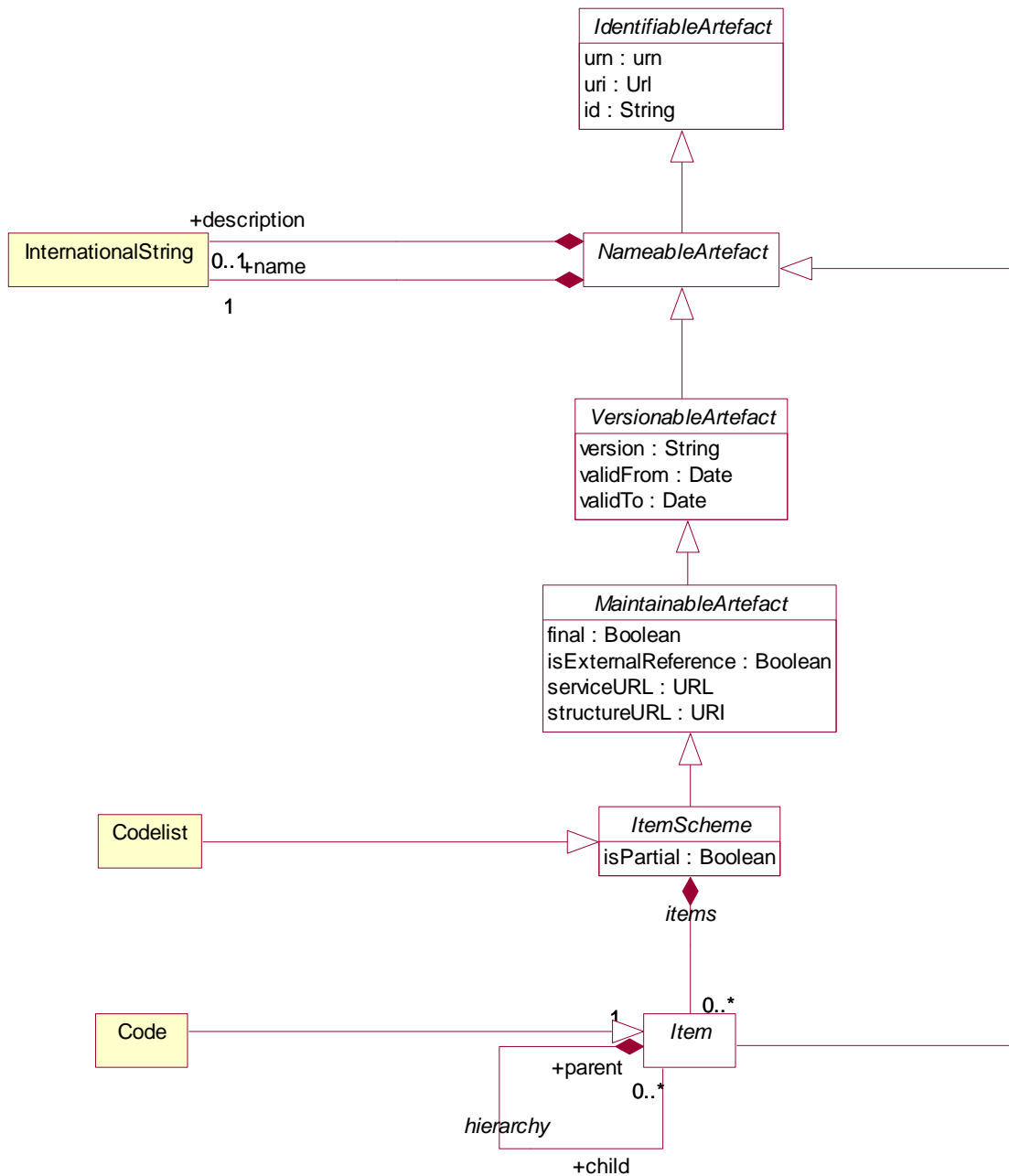
## 831      **4.2 Inheritance View**

832      The inheritance and relationship views are shown together in each of the diagrams in the  
833      specific sections below.

834 **4.3 Codelist**

835 **4.3.1 Class Diagram**

836



**Figure 15 Class diagram of the Codelist**

837

838 **4.3.2 Explanation of the Diagram**

839 **4.3.2.1 Narrative**

840 The `Codelist` inherits from the `ItemScheme` and therefore has the following attributes:

841

842     • `id`

843     • `uri`

844     • `urn`

845     • `version`

846     • `validFrom`

847     • `validTo`

848     • `isExternalReference`

849     • `serviceURL`

850     • `structureURL`

851     • `final`

852     • `isPartial`

853 The `Code` inherits from `Item` and has the following attributes:

854

855     • `id`

856     • `uri`

857     • `urn`

858 Both `Codelist` and `Code` have the association to `InternationalString` to support a  
859 multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to  
860 support notes (not shown).

861

862 Through the inheritance the `Codelist` comprise one or more `Codes`, and the `Code` itself can  
863 have one or more child `Codes` in the (inherited) hierarchy association. Note that a child  
864 `Code` can have only one parent `Code` in this association. A more complex  
865 `HierarchicalCodelist` which allow multiple parents and multiple hierarchies is described  
866 later.

867

868 A partial `Codelist` (where `isPartial` is set to “true”) is identical to a `Codelist` and  
869 contains the `Code` and associated names and descriptions, just as in a normal code list.  
870 However, its content is a sub set of the full `Codelist`. The way this works is described in  
871 section 3.5.3.1 on `ItemScheme`.

872

873 **4.3.2.2 Definitions**

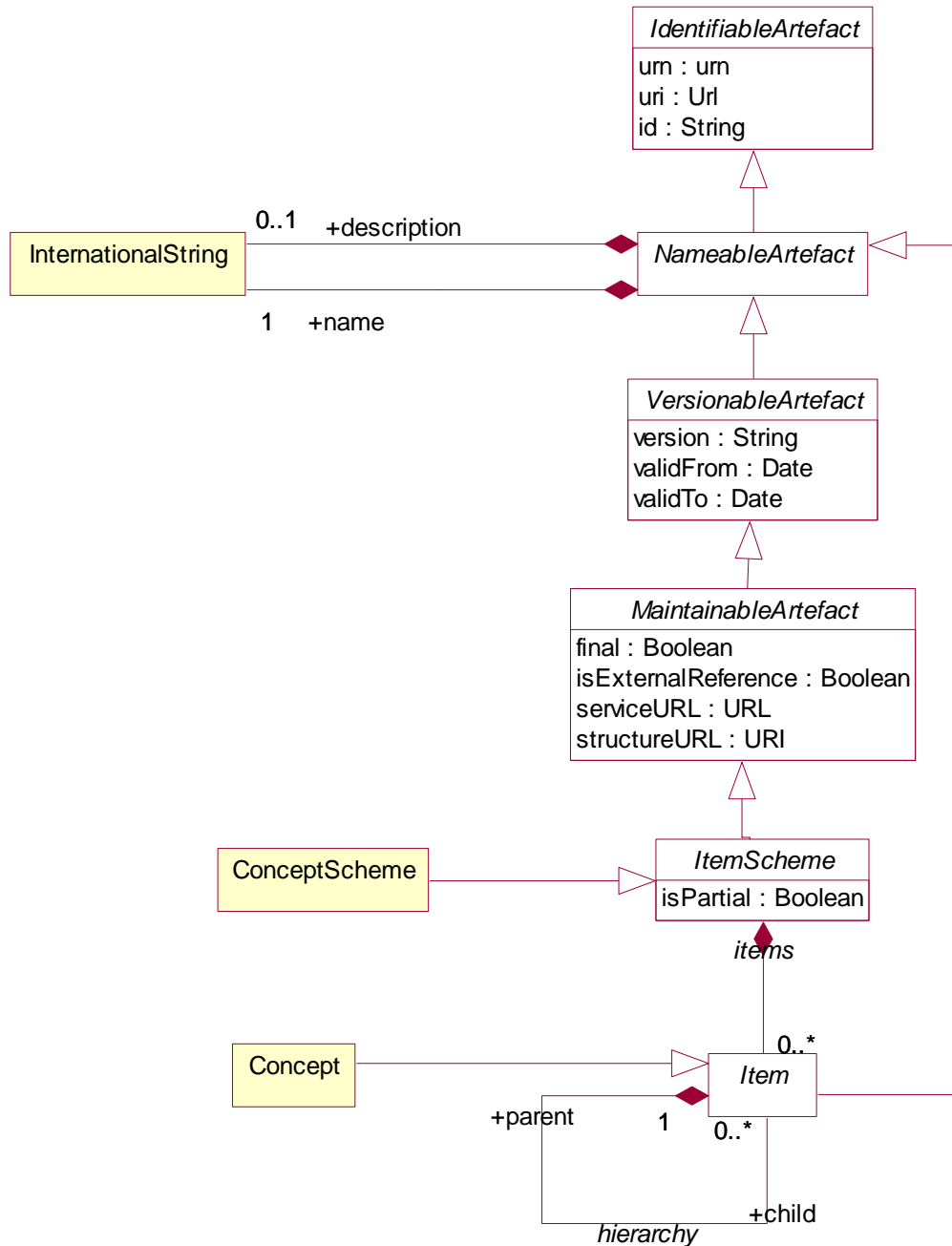
<b>Class</b>	<b>Feature</b>	<b>Description</b>
Codelist	Inherits from <i>ItemScheme</i>	A list from which some statistical concepts (coded concepts) take their values.
Code	Inherits from <i>Item</i>	A language independent set of letters, numbers or symbols that represent a concept whose meaning is described in a natural language.
	/hierarchy	Associates the parent and the child codes.

874

875 **4.4 Concept Scheme and Concepts**

876 **4.4.1 Class Diagram - Inheritance**

877



**Figure 16 Class diagram of the Concept Scheme**

878 **4.4.2 Explanation of the Diagram**

879 The `ConceptScheme` inherits from the `ItemScheme` and therefore has the following  
880 attributes:

- 881
- 882 • `id`
- 883
- 884 • `uri`
- 885
- 886 • `urn`
- 887
- 888 • `version`
- 889
- 890 • `validFrom`
- 891
- 892 • `validTo`
- 893
- 894 • `isExternalReference`
- 895
- 896 • `registryURL`
- 897
- 898 • `structureURL`
- 899
- 900 • `repositoryURL`
- 901
- 902 • `final`
- 903
- 904 • `isPartial`

894 `Concept` inherits from `Item` and has the following attributes:

- 895
- 896 • `id`
- 897
- 898 • `uri`
- 899
- 900 • `urn`

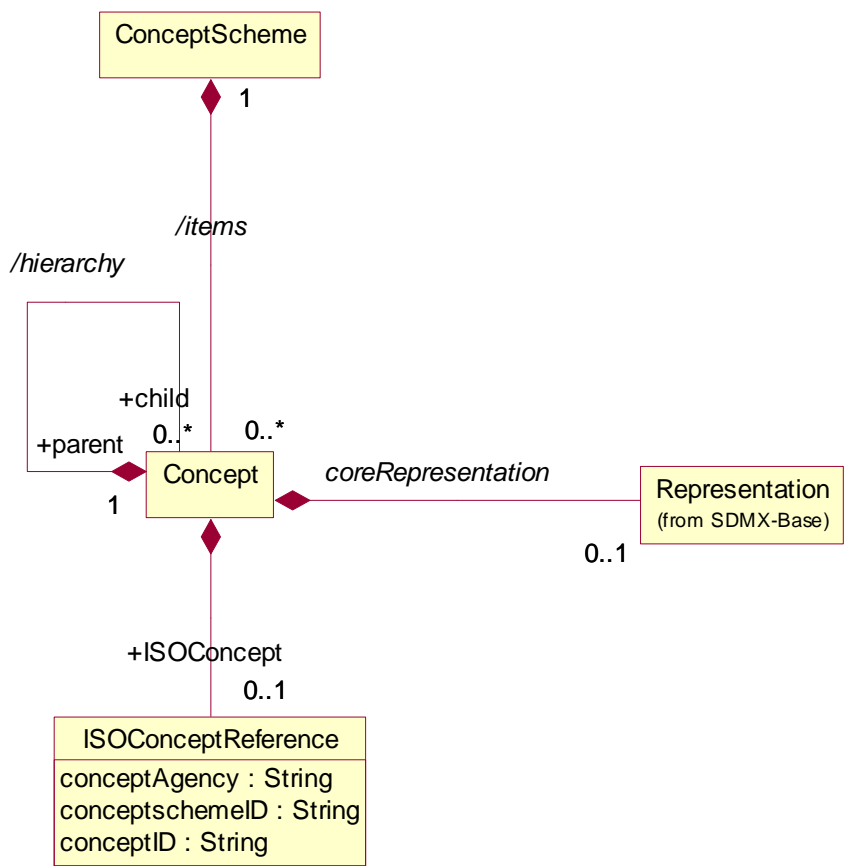
899 Through the inheritance from `NameableArtefact` both `ConceptScheme` and `Concept`  
900 have the association to `InternationalString` to support a multi-lingual name, an optional  
901 multi-lingual description, and an association to `Annotation` to support notes (not shown).  
902

903 Through the inheritance from `ItemScheme` the `ConceptScheme` comprise one or more  
904 `Concepts`, and the `Concept` itself can have one or more child `Concepts` in the (inherited)  
905 hierarchy association. Note that a child `Concept` can have only one parent `Concept` in  
906 this association.  
907

908 A partial `ConceptScheme` (where `isPartial` is set to “true”) is identical to a  
909 `ConceptScheme` and contains the `Concept` and associated names and descriptions, just as  
910 in a normal `ConceptScheme`. However, its content is a sub set of the full `ConceptScheme`.  
911 The way this works is described in section 3.5.3.1 on `ItemScheme`.

912

913 **4.4.3 Class Diagram - Relationship**



914

915

**Figure 17: Relationship class diagram of the Concept Scheme**

916 **4.4.4 Explanation of the diagram**

917 **4.4.4.1 Narrative**

918 The `ConceptScheme` can have one or more `Concepts`. A `Concept` can have zero or more  
 919 child `Concepts`, thus supporting a hierarchy of `Concepts`. Note that a child `Concept` can  
 920 have only one parent `Concept` in this association. The purpose of the hierarchy is to relate  
 921 concepts that have a semantic relationship: for example a `Reporting_Country` and  
 922 `Vis_a_Vis_Country` may both have `Country` as a parent concept, or a `CONTACT` may have a  
 923 `PRIMARY_CONTACT` as a child concept. It is not the purpose of such schemes to define  
 924 reporting structures: these reporting structures are defined in the  
 925 `MetadataStructureDefinition`.

926

927 The `Concept` can be associated with a `coreRepresentation`. The  
 928 `coreRepresentation` is the specification of the format and value domain of the `Concept`  
 929 when used on a structure like a `DataStructureDefinition` or a  
 930 `MetadataStructureDefinition`, unless the specification of the `Representation` is  
 931 overridden in the relevant structure definition. In a hierarchical `ConceptScheme` the

932 Representation is inherited from the parent `Concept` unless overridden at the level of the  
933 child `Concept`.

934

935 Note that the `ConceptScheme` is used as the `Representation` of the `MeasureDimension`  
936 in a `DataStructureDefinition` (see 5.3.2). Each `Concept` in this `ConceptScheme` is a  
937 specific measure, each of which can be given a `coreRepresentation`. Thus the valid  
938 format of the observation for each measure when reported in a data set for the  
939 `MeasureDimension` is specified in the `Concept`. This allows a different format for each  
940 measure. This is covered in more detail in 5.3.

941

942 The `Representation` is documented in more detail in the section on the SDMX Base.

943

944 The `Concept` may be related to a concept described in terms of the ISO/IEC 11179 standard.  
945 The `ISOConceptReference` identifies this concept and concept scheme in which it is  
946 contained.

947 **4.4.4.2 Definitions**

Class	Feature	Description
<code>ConceptScheme</code>	Inherits from <i>ItemScheme</i>	The descriptive information for an arrangement or division of concepts into groups based on characteristics, which the objects have in common.
<code>Concept</code>	Inherits from <i>Item</i>	A concept is a unit of knowledge created by a unique combination of characteristics.
	<code>/hierarchy</code>	Associates the parent and the child concept.
	<code>coreRepresentation</code>	Associates a <code>Representation</code> .
	<code>+ISOConcept</code>	Association to an ISO concept reference.
<code>ISOConceptReference</code>		The identity of an ISO concept definition.
	<code>conceptAgency</code>	The maintenance agency of the concept scheme containing the concept.
	<code>conceptSchemeID</code>	The identifier of the concept scheme.
	<code>conceptID</code>	The identifier of the concept.

948

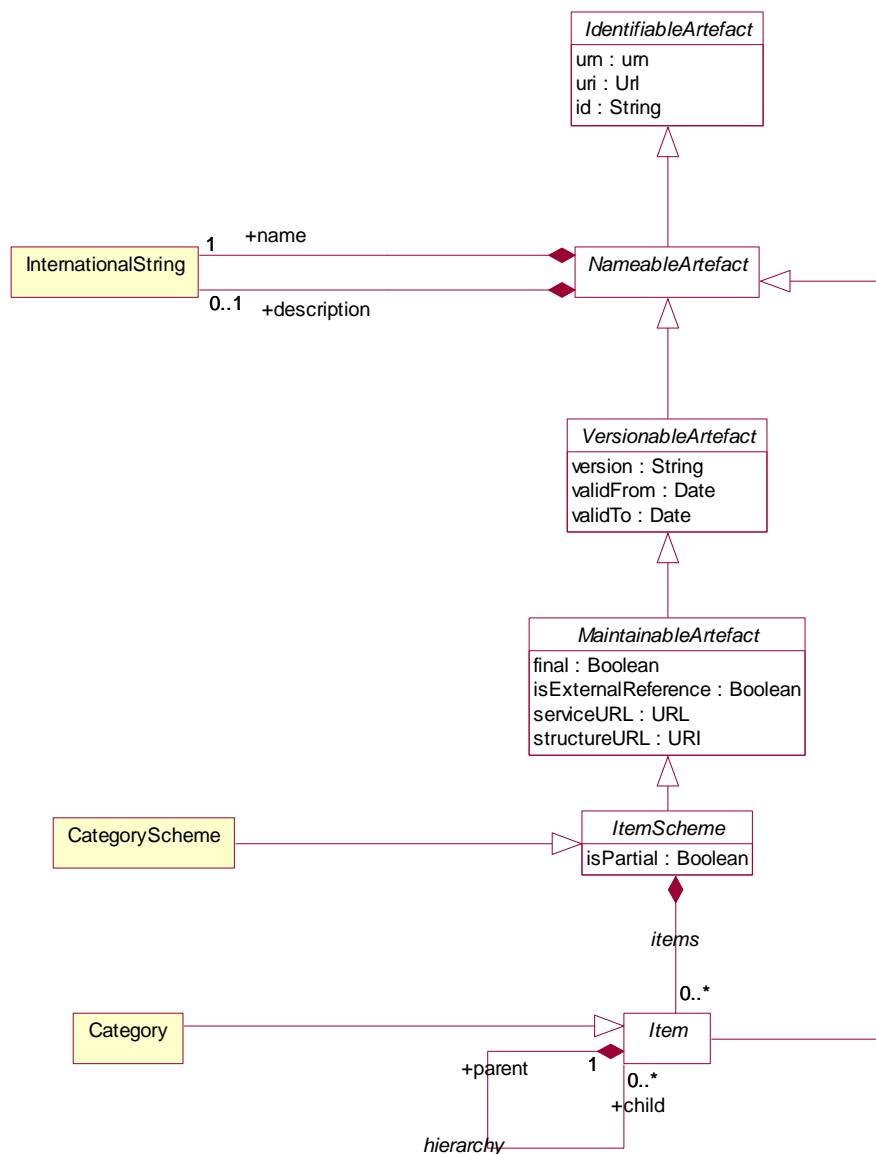


949 **4.5 Category Scheme**

950 **4.5.1 Context**

951 This package defines the structure that supports the definition of and relationships between  
 952 categories in a category scheme. It is similar to the package for concept scheme. An example  
 953 of a category scheme is one which categorises data – sometimes known as a subject matter  
 954 domain scheme or a data category scheme. Importantly, as will be seen later, the individual  
 955 nodes in the scheme (the “categories”) can be associated to any set of  
 956 `IdentifiableArtefacts` in a `Categorisation`.

957 **4.5.2 Class diagram - Inheritance**



**Figure 18 Inheritance Class diagram of the Category Scheme**

958

### 959 4.5.3 Explanation of the Diagram

#### 960 4.5.3.1 Narrative

961 The categories are modelled as a hierarchical *ItemScheme*. The *CategoryScheme* inherits  
962 from the *ItemScheme* and has the following attributes:

963

964

- id

965

- uri

966

- urn

967

- version

968

- validFrom

969

- validTo

970

- isExternalReference

971

- structureURL

972

- serviceURL

973

- final

974

- isPartial

975 *Category* inherits from *Item* and has the following attributes:

976

977

- id

978

- uri

979

- urn

980 Both *CategoryScheme* and *Category* have the association to *InternationalString* to  
981 support a multi-lingual name, an optional multi-lingual description, and an association to  
982 *Annotation* to support notes (not shown on the model).

983

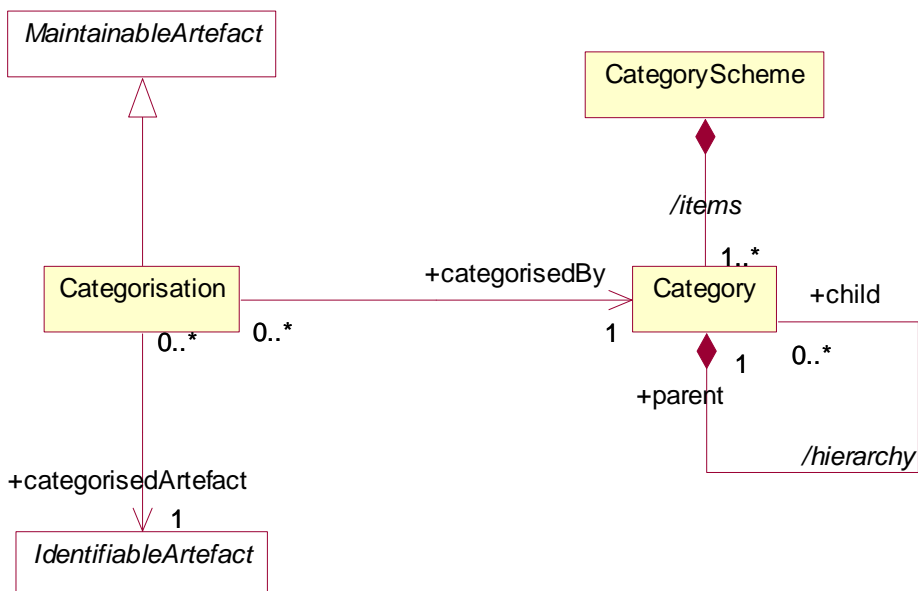
984 Through the inheritance the *CategoryScheme* comprise one or more *Category*s, and the  
985 *Category* itself can have one or more child *Category* in the (inherited) hierarchy  
986 association. Note that a child *Category* can have only one parent *Category* in this  
987 association.

988

989 A partial *CategoryScheme* (where *isPartial* is set to “true”) is identical to a  
990 *CategoryScheme* and contains the *Category* and associated names and descriptions, just

991 as in a normal `CategoryScheme`. However, its content is a sub set of the full  
 992 `CategoryScheme`. The way this works is described in section 3.5.3.1 on `ItemScheme`.  
 993

994 **4.5.4 Class diagram - Relationship**



995  
 996

**Figure 19: Relationship Class diagram of the Category Scheme**

997 The `CategoryScheme` can have one or more `Category`s. The `Category` is `Identifiable` and  
 998 has identity information. A `Category` can have zero or more child `Category`s, thus  
 999 supporting a hierarchy of `Category`s. Any `IdentifiableArtefact` can be  
 1000 `+categorisedBy` a `Category`. This is achieved by means of a `Categorisation`. Each  
 1001 `Categorisation` can associate one `IdentifiableArtefact` with one `Category`.  
 1002 Multiple `Categorisations` can be used to build a set of `IdentifiableArtefacts` that  
 1003 are `+categorisedBy` the same `Category`. Note that there is no navigation (i.e. no  
 1004 embedded reference) to the `Categorisation` from the `Category`. From an implementation  
 1005 perspective this is necessary as `Categorisation` has no affect on the versioning of either  
 1006 the `Category` or the `IdentifiableArtefact`.

1007 **4.5.4.1 Definitions**

Class	Feature	Description
<code>CategoryScheme</code>	Inherits from <i>ItemScheme</i>	The descriptive information for an arrangement or division of categories into groups based on characteristics, which the objects have in common.
	<code>/items</code>	Associates the categories.

<b>Class</b>	<b>Feature</b>	<b>Description</b>
Category	Inherits from <i>Item</i>	An item at any level within a classification, typically tabulation categories, sections, subsections, divisions, subdivisions, groups, subgroups, classes and subclasses.
	/hierarchy	Associates the parent and the child Category.
Categorisation	Inherits from MaintainableArtefact	Associates an IdentifiableArtefact with a Category.
	+categorisedArtefact	Associates the IdentifiableArtefact.
	+categorisedBy	Associates the Category.

1008 **4.6 Organisation Scheme**

1009 **4.6.1 Class Diagram**

1010

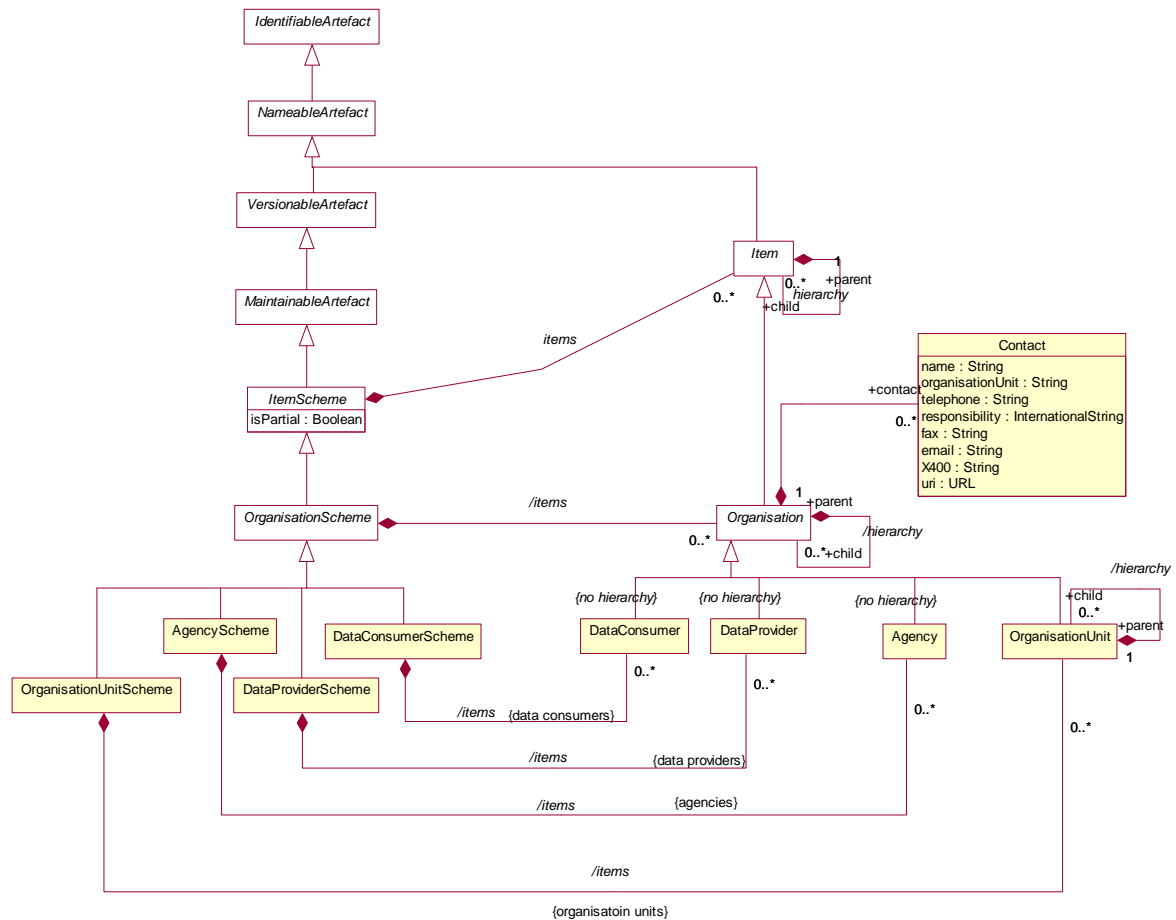


Figure 20 The Organisation Scheme class diagram

1011 **4.6.2 Explanation of the Diagram**

1012 **4.6.2.1 Narrative**

1013 The *OrganisationScheme* is abstract. It contains *Organisation* which is also abstract.  
1014 The *Organisation* can have child *Organisation*.

1015

1016 The *OrganisationScheme* can be one of four types:

1017

- 1018 1. *AgencyScheme* – contains *Agency* which is restricted to a flat list of agencies (i.e.  
1019 there is no hierarchy). Note that the SDMX system of (Maintenance) *Agency* can be  
1020 hierarchic and this is explained in more detail in the separate document “Technical  
1021 Notes”.
- 1022 2. *DataProviderScheme* – contains *DataProvider* which is restricted to a flat list of  
1023 agencies (i.e. there is no hierarchy).

- 1024 3. `DataConsumerScheme` – contains `DataConsumer` which is restricted to a flat list of  
 1025 agencies (i.e. there is no hierarchy).  
 1026 4. `OrganisationUnitScheme` – contains `OrganisationUnit` which does inherit the  
 1027 `/hierarchy` association from `Organisation`.  
 1028

1029 Reference metadata can be attached to the *Organisation* by means of the metadata  
 1030 attachment mechanism. This mechanism is explained in the Reference Metadata section of  
 1031 this document (see section 7). This means that the model does not specify the specific  
 1032 reference metadata that can be attached to a `DataProvider`,  
 1033 `DataConsumer`, `OrganisationUnit` or `Agency`, except for limited `Contact` information.  
 1034

1035 A partial *OrganisationScheme* (where `isPartial` is set to “true”) is identical to a  
 1036 *OrganisationScheme* and contains the `Organisation` and associated names and  
 1037 descriptions, just as in a normal *OrganisationScheme*. However, its content is a sub set of  
 1038 the full *OrganisationScheme*. The way this works is described in section 3.5.3.1 on  
 1039 *ItemScheme*.  
 1040

1041 **4.6.2.2 Definitions**

Class	Feature	Description
<i>OrganisationScheme</i>	Abstract Class Inherits from <i>ItemScheme</i>  Sub classes are: AgencyScheme DataProviderScheme DataConsumerScheme OrganisationUnitScheme	A maintained collection of Organisations.
	<code>/items</code>	Association to the Organisations in the scheme.
<i>Organisation</i>	Inherits from <i>Item</i>  Sub classes are: Agency DataProvider DataConsumer OrganisationUnit	An organisation is a unique framework of authority within which a person or persons act, or are designated to act, towards some purpose.
	<code>+contact</code>	Association to the Contact information.
	<code>/hierarchy</code>	Association to child Organisations.

Class	Feature	Description
Contact		An instance of a role of an individual or an organization (or organization part or organization person) to whom an information item(s), a material object(s) and/or person(s) can be sent to or from in a specified context.
	name	The designation of the Contact person by a linguistic expression.
	organisationUnit	The designation of the organisational structure by a linguistic expression, within which Contact person works.
	responsibility	The function of the contact person with respect to the organisation role for which this person is the Contact.
	telephone	The telephone number of the Contact.
	fax	The fax number of the Contact.
	email	The Internet e-mail address of the Contact.
	X400	The X400 address of the Contact.
	uri	The URL address of the Contact.
AgencyScheme		A maintained collection of Maintenance Agencies.
	/items	Association to the Maintenance Agency in the scheme.
DataProviderScheme		A maintained collection of Data Providers.
	/items	Association to the Data Providers in the scheme.
DataConsumerScheme		A maintained collection of Data Consumers.

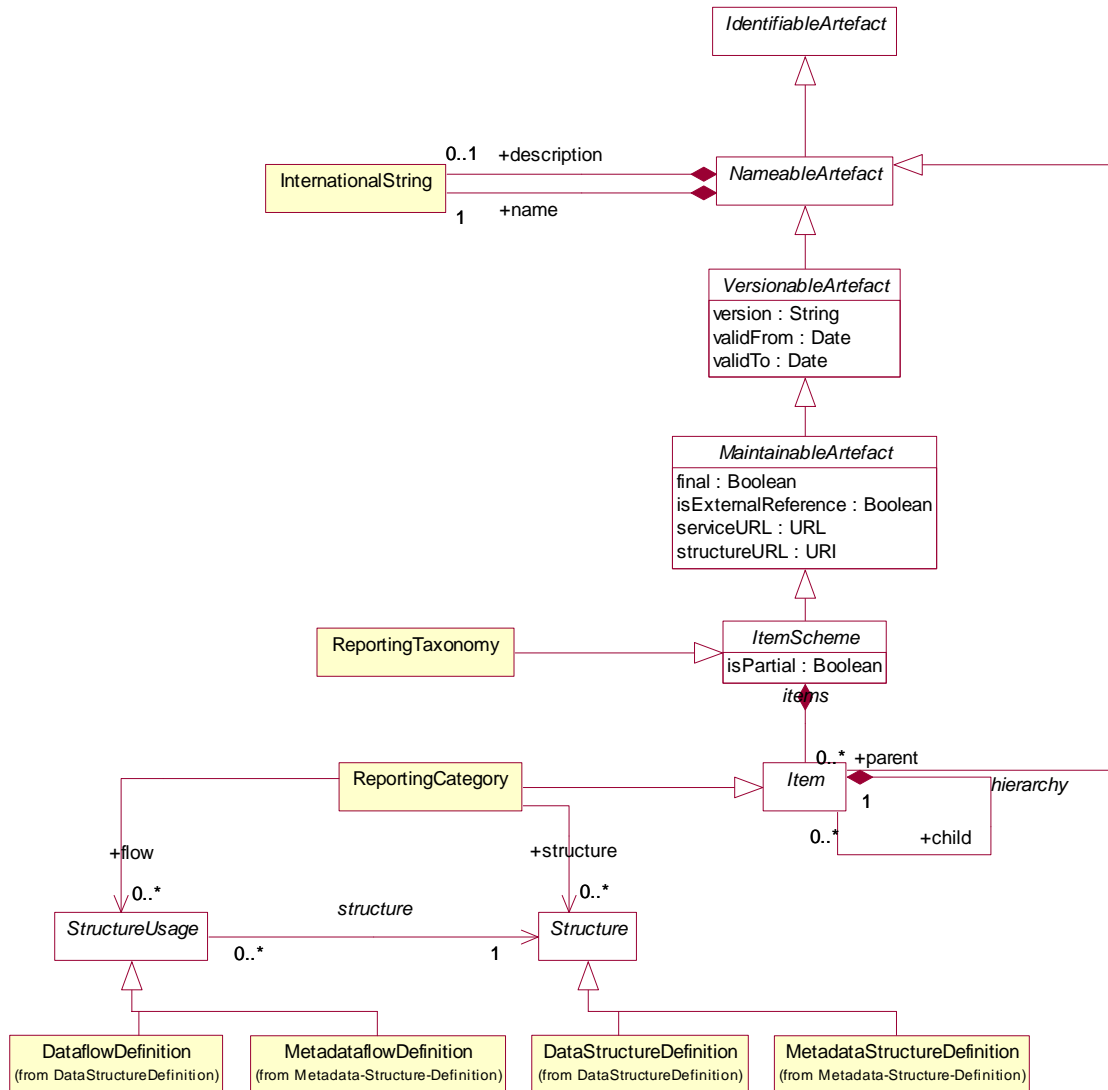
Class	Feature	Description
	/items	Association to the Data Consumers in the scheme.
OrganisationUnitScheme		A maintained collection of Organisation Units.
	/items	Association to the Organisation Units in the scheme.
Agency	Inherits from <i>Organisation</i>	Responsible agency for maintaining artefacts such as statistical classifications, glossaries, structural metadata such as Data and Metadata Structure Definitions, Concepts and Code lists.
DataProvider	Inherits from <i>Organisation</i>	An organisation that produces data or reference metadata.
DataConsumer	Inherits from <i>Organisation</i>	An organisation using data as input for further processing.
OrganisationUnit	Inherits from <i>Organisation</i>	A designation in the organisational structure.
	/hierarchy	Association to child Organisation Units

1042



1043 **4.7 Reporting Taxonomy**

1044 **4.7.1 Class Diagram**



1045  
1046

**Figure 21: Class diagram of the Reporting Taxonomy**

1047 **4.7.2 Explanation of the Diagram**

1048 **4.7.2.1 Narrative**

1049 In some data reporting environments, and in particular those in primary reporting, a report may  
1050 comprise a variety of heterogeneous data, each described by a different *Structure*. Equally,  
1051 a specific disseminated or published report may also comprise a variety of heterogeneous  
1052 data. The definition of the set of linked sub reports is supported by the *ReportingTaxonomy*.

1053

1054 The *ReportingTaxonomy* is a specialised form of *ItemScheme*. Each  
1055 *ReportingCategory* of the *ReportingTaxonomy* can link to one or more

1056 *StructureUsage* which itself can be one of *DataflowDefinition*, or  
 1057 *MetadataflowDefinition*, and one or more *Structure*, which itself can be one of  
 1058 *DataStructureDefinition* or *MetadataStructureDefinition*. It is expected that  
 1059 within a specific *ReportingTaxonomy* each *Category* that is linked in this way will be linked  
 1060 to the same class (e.g. all *Category* in the scheme will link to a *DataflowDefinition*).  
 1061 Note that a *ReportingCategory* can have child *ReportingCategory* and in this way it is  
 1062 possible to define a hierarchical *ReportingTaxonomy*. It is possible in this taxonomy that  
 1063 some *ReportingCategory* are defined just to give a reporting structure. For instance:

- 1064  
 1065 Section 1  
 1066     1. linked to *DataflowDefinition\_1*  
 1067     2 linked to *DataflowDefinition\_2*  
 1068 Section 2  
 1069     1 linked to *DataflowDefinition\_3*  
 1070     2 linked to *DataflowDefinition\_4*  
 1071

1072 Here, the nodes of Section 1 and Section 2 would not be linked to *DataflowDefinition* but  
 1073 the other would be linked to a *DataflowDefinition* (and hence the  
 1074 *DataStructureDefinition*).

1075  
 1076 A partial *ReportingTaxonomy* (where *isPartial* is set to “true”) is identical to a  
 1077 *ReportingTaxonomy* and contains the *ReportingCategory* and associated names and  
 1078 descriptions, just as in a normal *ReportingTaxonomy* However, its content is a sub set of  
 1079 the full *ReportingTaxonomy* The way this works is described in section 3.5.3.1 on  
 1080 *ItemScheme*.  
 1081

1082 **4.7.2.2 Definitions**

Class	Feature	Description
<i>ReportingTaxonomy</i>	Inherits from <i>ItemScheme</i>	A scheme which defines the composition structure of a data report where each component can be described by an independent <i>DataflowDefinition</i> or <i>MetadataflowDefinition</i> .
	items	Associates the <i>ReportingCategory</i>
<i>ReportingCategory</i>	Inherits from <i>Item</i>	A component that gives structure to the report and links to data and metadata.
	hierarchy	Associates child <i>ReportingCategory</i> .

Class	Feature	Description
	+flow	Association to the data and metadata flows that link to metadata about the provisioning and related data and metadata sets, and the structures that define them.
	+structure	Association to the Data Structure Definition and Metadata Structure Definitions which define the structural metadata describing the data and metadata that are contained at this part of the report.

1083

## 1084 **5 Data Structure Definition and Dataset**

### 1085 **5.1 Introduction**

1086 The `DataStructureDefiniton` is the class name for a structure definition for data. Some  
1087 organisations know this type of definition as a “Key Family” and so the two names are  
1088 synonymous. The term Data Structure Definition (also referred to as DSD) is used in this  
1089 specification.

1090  
1091 Many of the constructs in this layer of the model inherit from the SDMX Base Layer. Therefore,  
1092 it is necessary to study both the inheritance and the relationship diagrams to understand the  
1093 functionality of individual packages. In simple sub models these are shown in the same  
1094 diagram, but are omitted from the more complex sub models for the sake of clarity. In these  
1095 cases, the inheritance diagram below shows the full inheritance tree for the classes concerned  
1096 with data structure definitions.

1097  
1098 There are very few additional classes in this sub model other than those shown in the  
1099 inheritance diagram below. In other words, the SDMX Base gives most of the structure of this  
1100 sub model both in terms of associations and in terms of attributes. The relationship diagrams  
1101 shown in this section show clearly when these associations are inherited from the SDMX Base  
1102 (see the Appendix “A Short Guide to UML in the SDMX Information Model” to see the  
1103 diagrammatic notation used to depict this).

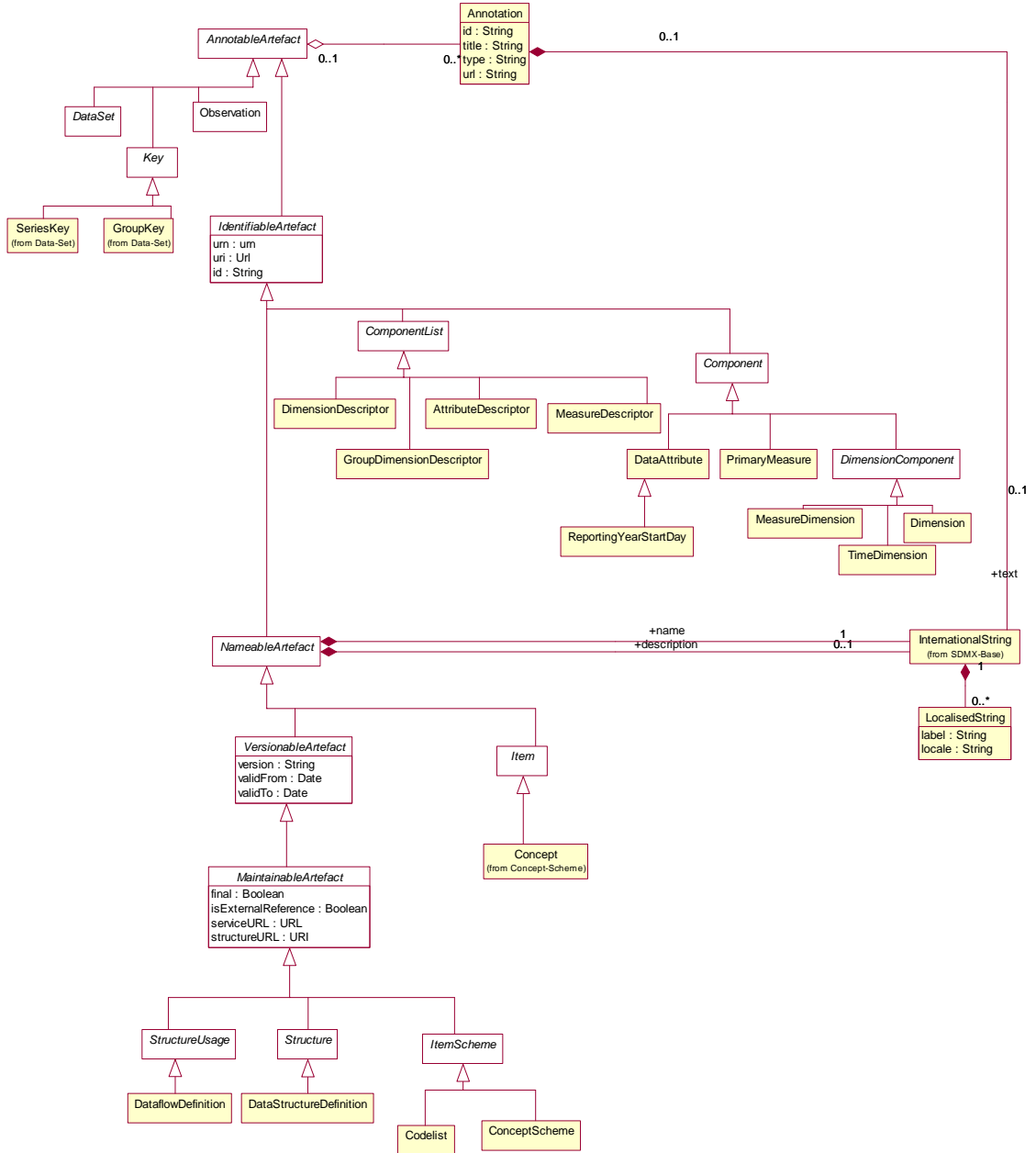
1104  
1105 The actual SDMX Base construct from which the concrete classes inherit depends upon the  
1106 requirements of the class for:

- 1107
- 1108
  - Annotation - *AnnotableArtefact*
- 1109
  - Identification - *IdentifiableArtefact*
- 1110
  - Naming - *NameableArtefact*
- 1111
  - Versioning – *VersionableArtefact*
- 1112
  - Maintenance - *MaintainableArtefact*

1113 **5.2 Inheritance View**

1114 **5.2.1 Class Diagram**

1115



**Figure 22 Class inheritance in the Data Structure Definition and Data Set Packages**

1116 **5.2.2 Explanation of the Diagram**

1117 **5.2.2.1 Narrative**

1118 Those classes in the SDMX metamodel which require annotations inherit from  
1119 *AnnotableArtefact* . These are:

1120

1121 • *IdentifiableArtefact*

1122 • *DataSet* (and therefore *StructureSpecificDataSet*, *GenericDataSet*,  
1123 *GenericTimeSeriesDataSet* *StructureSpecificTimeSeriesDataSet*)

1124 • *Key* (and therefore *SeriesKey* and *GroupKey*)

1125 Those classes in the SDMX metamodel which require annotations and global identity are  
1126 derived from *IdentifiableArtefact* . These are:

1127

1128 • *NameableArtefact*

1129 • *ComponentList*

1130 • *Component*

1131 Those classes in the SDMX metamodel which require annotations, global identity, multilingual  
1132 name and multilingual description are derived from *NameableArtefact* . These are:

1133

1134 • *VersionableArtefact*

1135 • *Item*

1136 The classes in the SDMX metamodel which require annotations, global identity, multilingual  
1137 name and multilingual description, and versioning are derived from *VersionableArtefact* .  
1138 These are:

1139

1140 • *MaintainableArtefact*

1141 Abstract classes which represent information that is maintained by Maintenance Agencies all  
1142 inherit from *MaintainableArtefact*, they also inherit all the features of a  
1143 *VersionableArtefact*, and are:

1144

1145 • *StructureUsage*

1146 • *Structure*

1147 • *ItemScheme*

1148 All the above classes are abstract. The key to understanding the class diagrams presented in  
1149 this section are the concrete classes that inherit from these abstract classes.

1150

1151 Those concrete classes in the SDMX Data Structure Definition and Dataset packages of the  
1152 metamodel which require to be maintained by Agencies all inherit (via other abstract classes)  
1153 from *MaintainableArtefact*, these are:

1154

1155     • *DataflowDefinition*

1156     • *DataStructureDefinition*

1157 The component structures that are lists of lists, inherit directly from *Structure*. A  
1158 *Structure* contains several lists of components. The concrete class that inherits from  
1159 *Structure* is:

1160     • *DataStructureDefinition*

1161 A *DataStructureDefinition* contains a list of dimensions, a list of measures and a list of  
1162 attributes.

1163

1164 The concrete classes which inherit from *ComponentList* and are sub components of the  
1165 *DataStructureDefinition* are:

1166

1167     • *DimensionDescriptor* - content is *Dimension*, *MeasureDimension* and  
1168       *Time Dimension*

1169     • *DimensionGroupDescriptor* - content is an association to *Dimension*,  
1170       *MeasureDimension*, *TimeDimension*

1171     • *MeasureDescriptor* - content is *PrimaryMeasure*

1172     • *AttributeDescriptor* - content is *DataAttribute*

1173 The classes that inherit from *Component* are:

1174

1175     • *PrimaryMeasure*

1176     • *DimensionComponent* and thereby its sub classes of *Dimension*,  
1177       *MeasureDimension*, and *TimeDimension*

1178

1179     • *DataAttribute*

1180 The class that inherit from *DataAttribute* is:

1181

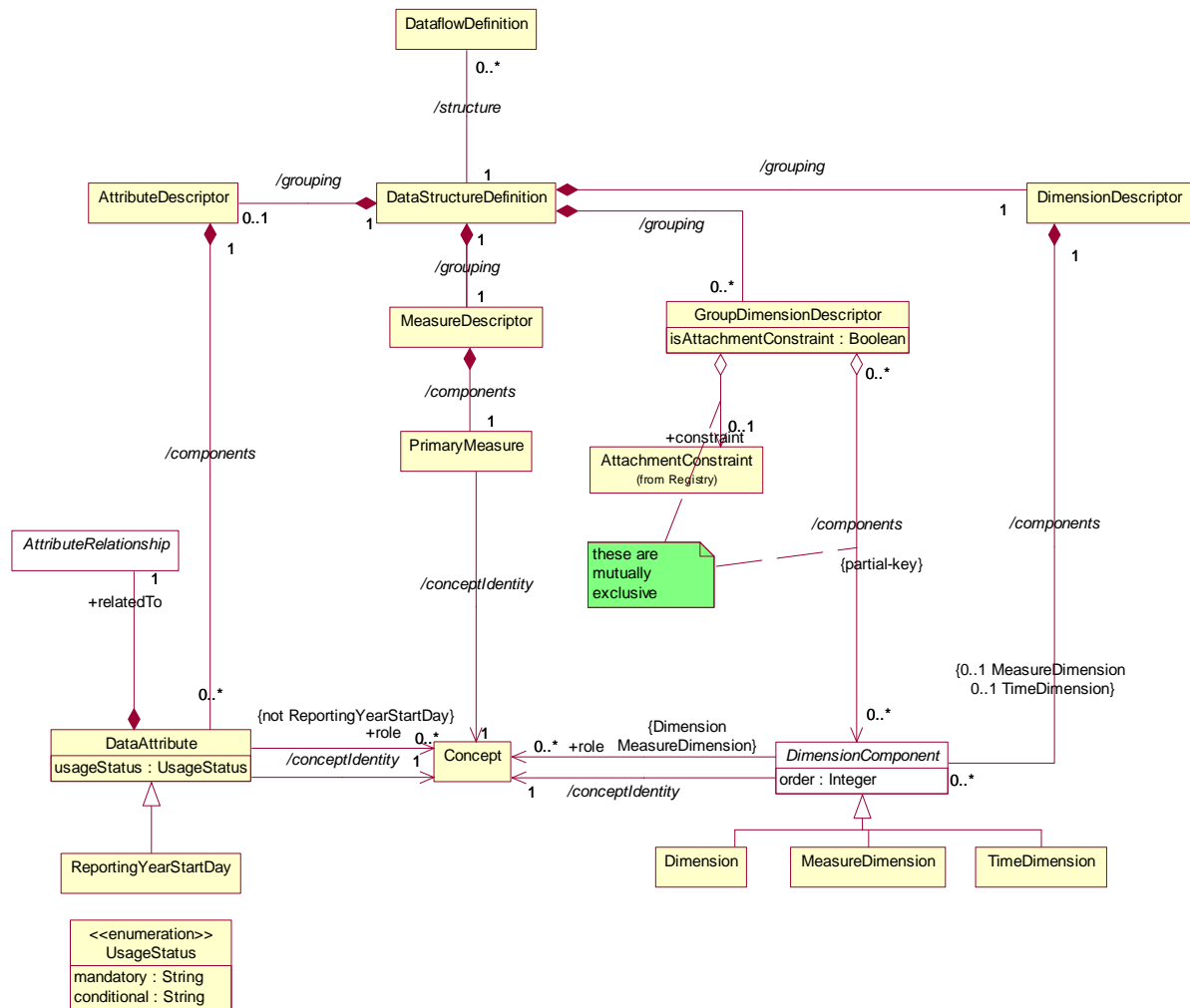
1182     • *ReportingYearStartDay*

1183

1184 The concrete classes identified above are the majority of the classes required to define the  
1185 metamodel for the *DataStructureDefinition*. The diagrams and explanations in the rest  
1186 of this section show how these concrete classes are related in order to support the  
1187 functionality required.

1188 **5.3 Data Structure Definition – Relationship View**

1189 **5.3.1 Class Diagram**



1190  
1191

**Figure 23 Relationship class diagram of the Data Structure Definition excluding representation**

1192 **5.3.2 Explanation of the Diagrams**

1193 **5.3.2.1 Narrative**

1194 A DataStructureDefinition defines the Dimensions, MeasureDimension, TimeDimension, DataAttributes, and PrimaryMeasure, and associated  
 1195 Representation that comprise the valid structure of data and related attributes that are  
 1196 contained in a DataSet, which is defined by a DataflowDefinition.  
 1197

1198 The DataflowDefinition may also have additional metadata attached that defines  
 1199 qualitative information and Constraints on the use of the DataStructureDefinition  
 1200 such as the sub set of Codes used in a Dimension (this is covered later in this document –  
 1201



1202 see “Data Constraints and Provisioning” section 9). Each `DataflowDefinition` has a  
1203 maximum of one `DataStructureDefinition` specified which defines the structure of any  
1204 `DataSets` to be reported/disseminated.

1205

1206 There are three types of dimension each having a common association to `Concept`:

1207

- 1208 • `Dimension`
- 1209 • `MeasureDimension`
- 1210 • `TimeDimension`

1211

1212 Note that In the description here `DimensionComponent` can be oany or all of its sub classes  
1213 i.e. `Dimension`, `MeasureDimension`, `TimeDimension`., and the term “`DataAttribute`”  
1214 refers to both `DataAttribute` and its sub class `ReportingYearStartDate`.

1215

1216 The `DimensionComponent`, `DataAttribute`, and `PrimaryMeasure` link to the `Concept`  
1217 that defines its name and semantic (`/conceptIdentity` association to `Concept`). The  
1218 `DataAttribute`, `Dimension`, and `MeasureDimension` (but not `TimeDimension`) can  
1219 optionally have a `+conceptRole` association with a `Concept` that identifies its role in the  
1220 `DataStructureDefinition`. Therefore, the allowable roles of a `Concept` are maintained  
1221 in a `ConceptScheme`. Examples of roles are: geography, entity, count, unit of measure. The  
1222 use of these roles is to enable applications to process the data in a meaningful way (e.g.  
1223 relating a dimension value to a mapping vector). It is expected that communities (such as the  
1224 official statistics community) will harmonise these roles with their community so that data can  
1225 be exchanged and shared in a meaningful way in the community.

1226

1227 The valid values for a `DimensionComponent`, `PrimaryMeasure`, or `DataAttribute`,  
1228 when used in this `DataStructureDefinition`, are defined by the `Representation`. This  
1229 `Representation` is taken from the `Concept` definition (`coreRepresentation`) unless it is  
1230 overridden in this `DataStructureDefinition` (`localRepresentation`) – see Figure 23.  
1231 Note that for the `MeasureDimension` the `Representation` must be a `ConceptScheme`  
1232 and this must always be referenced from the `MeasureDimension` and cannot therefore be  
1233 defaulted to the `Representation` of the `Concept` associated by the `/conceptIdentity`.  
1234 Note also that `TimeDimension` and `ReportingYearStartDate` are constrained to specific  
1235 `FacetValueTypes`

1236

1237 There will always be a `DimensionDescriptor` grouping that identifies all of the `Dimension`  
1238 comprising the full key. Together the `Dimensions` specify the key of an `Observation`.

1239

1240 The `DimensionComponent` can optionally be grouped by multiple  
1241 `GroupDimensionDescriptors` each of which identifies the group of `Dimensions` that can  
1242 form a partial key. The `GroupDimensionDescriptor` must be identified  
1243 (`GroupDimensionDescriptor.id`) and this is used in the `GroupKey` of the `DataSet` to  
1244 declare which `DataAttributes` are reported at this group level in the `DataSet`.

1245

1246 There may be a maximum of one `MeasureDimension` specified in the  
1247 `DimensionDescriptor`. The purpose of a `MeasureDimension` is to specify formally the  
1248 meaning of the measures (because the `PrimaryMeasure` typically has a generic meaning  
1249 e.g. observation value) and to enable multiple measures to be defined and reported in a  
1250 `StructureSpecificDataSet`. Note that the `MeasureDimension` references a

1251 ConceptScheme as its Representation (see later) whereas a Dimension can have either  
1252 an enumerated (Codelist) or non-enumerated (Facet) representation. For a  
1253 MeasureDimension the Concepts in the ConceptScheme comprise the list of allowable  
1254 measures. This enables the representation for each individual measure (Concept) to be  
1255 declared as the coreRepresentation of the Concept, thus overriding the  
1256 Representation specified for the PrimaryMeasure for the observation value of this  
1257 MeasureDimension Concept.

1258

1259 There can be a maximum of one TimeDimension specified in the DimensionDescriptor.  
1260 The TimeDimension is used to specify the Concept used to convey the time period of the  
1261 observation in a data set. The TimeDimension must contain a valid representation of time  
1262 and cannot be coded

1263

1264 The PrimaryMeasure is the observable phenomenon, and, although there can be only one  
1265 PrimaryMeasure, for consistency with the ComponentList/Component pattern it is  
1266 grouped by a MeasureDescriptor.

1267

1268 The DataAttribute defines a characteristic of data that are collected or disseminated and is  
1269 grouped in the DataStructureDefinition by a single AttributeDescriptor. The  
1270 DataAttribute can be specified as being mandatory, or conditional, as defined in  
1271 usageStatus. The DataAttribute may play a specific role in the structure and this is  
1272 specified by the +role association to the Concept that identifies its role.

1273

1274 A DataAttribute is specified as being +relatedTo an AttributeRelationship which  
1275 defines the constructs to which the DataAttribute is to be reported present in a DataSet.  
1276 The DataAttribute can be specified as being related to one of the following artefacts:

1277

1278 • DataSet (NoSpecifiedRelationship)

1279

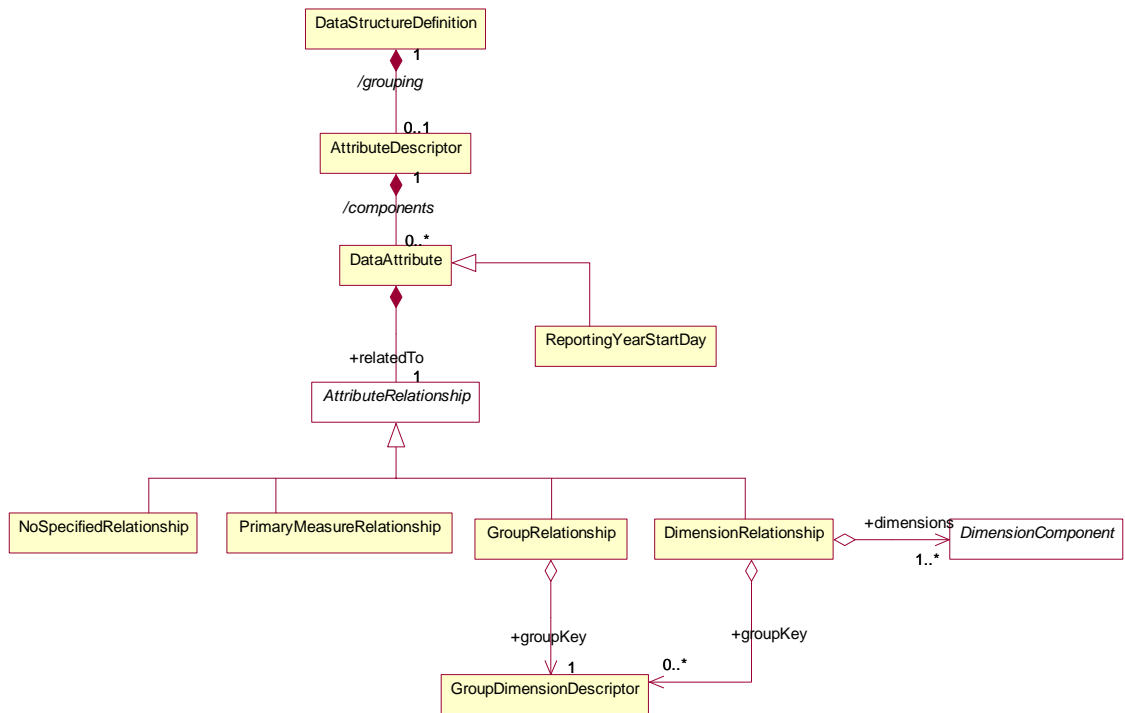
• Dimension or set of Dimensions (DimensionRelationship)

1280

• Set of Dimensions specified by a GroupKey (GroupRelationship – this is retained  
1281 for compatibility reasons – or +groupKey of the DimensionRelationship)

1282

• Observation (PrimaryMeasureRelationship)



1283  
1284

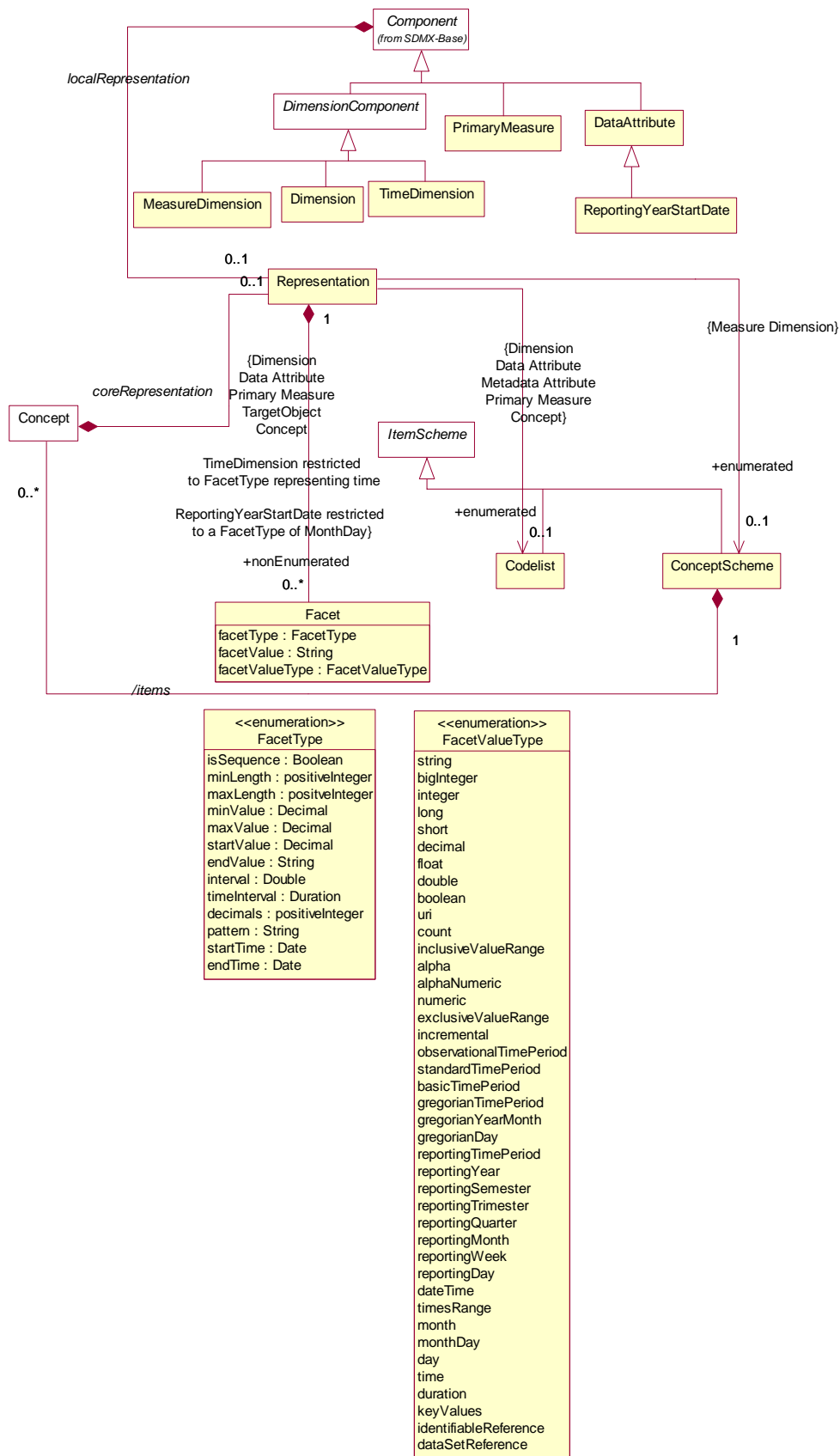
**Figure 24: Attribute Attachment Defined in the Data Structure Definition**

1285 The following table details the possible relationships a `DataAttribute` may specify. Note  
1286 that these relationships are mutually exclusive, and therefore only one of the following is  
1287 possible.

Relationship	Meaning	Location in Data Set at which the Attribute is reported
None	The value of the attribute does not vary with the values of any other Component.	The attribute is reported at the level of the Dataset Attribute.
Dimension (1..n)	The value of the attribute will vary with the value(s) of the referenced Dimension(s). In this case, Group(s) to which the attribute should be attached may optionally be specified.	The attribute is reported at the lowest level of the Dimension to which the Attribute is related, otherwise at the level of the Group if Attachment Group(s) is specified.

Relationship	Meaning	Location in Data Set at which the Attribute is reported
Group	The value of the Attribute varies with combination of values for all of the Dimensions contained in the Group. This is added as a convenience to listing all Dimensions and the attachment Group, but should only be used when the Attribute value varies based on <u>all</u> Group Dimension values.	The attribute is reported at the level of Group.
Primary Measure	The value of the Attribute varies with the observed value.	The attribute is reported at the level of Observation.

1288  
1289



1290  
1291

**Figure 25: Representation of DSD Components**

1292 Each of Dimension, MeasureDimension, TimeDimension, PrimaryMeasure, and  
 1293 DataAttribute can have a Representation specified (using the  
 1294 localRepresentation association). If this is not specified in the  
 1295 DataStructureDefinition then the representation specified for Concept  
 1296 (coreRepresentation) is used. For the MeasureDimension the representation for the  
 1297 individual measures is specified for the Concept in the ConceptScheme referenced by the  
 1298 MeasureDimension.

1299

1300 A DataStructureDefinition can be extended to form a derived  
 1301 DataStructureDefinition. This is supported in the StructureMap.

1302 **5.3.2.2 Definitions**

Class	Feature	Description
StructureUsage		See "SDMX Base".
DataflowDefinition	Inherits from <i>StructureUsage</i>	Abstract concept (i.e. the structure without any data) of a flow of data that providers will provide for different reference periods.
	/structure	Associates a Dataflow Definition to the Data Structure Definition.
DataStructureDefinition		A collection of metadata concepts, their structure and usage when used to collect or disseminate data.
	/grouping	An association to a set of metadata concepts that have an identified structural role in a Data Structure Definition.
Group DimensionDescriptor	Inherits from <i>ComponentList</i>	A set metadata concepts that define a partial key derived from the Dimension Descriptor in a Data Structure Definition.
	+constraint	Identifies an Attachment Constraint that specifies the sub set of Dimension, Measure, or Attribute values to which an Attribute can be attached.
	/components	An association to the Dimension and Measure

Class	Feature	Description
		Dimension components that comprise the group.
DimensionDescriptor	Inherits from <i>ComponentList</i>	An ordered set of metadata concepts that, combined, classify a statistical series, and whose values, when combined (the key) in an instance such as a data set, uniquely identify a specific observation.
	/components	An association to the Dimension, Measure Dimension, and Time Dimension comprising the Key Descriptor.
AttributeDescriptor	Inherits from <i>ComponentList</i>	A set metadata concepts that define the attributes of a Data Structure Definition.
	/components	An association to a Data Attribute component.
MeasureDescriptor	Inherits from <i>ComponentList</i>	A metadata concept that defines the measure of a Data Structure Definition.
	/components	An association to a measure component.
Dimension	Inherits from Component	A metadata concept used (most probably together with other metadata concepts) to classify a statistical series, e.g. a statistical concept indicating a certain economic activity or a geographical reference area.
	/role	Association to the Concept that specifies the role that the Dimension plays in the Data Structure Definition.
	/conceptIdentity	An association to the metadata concept which defines the semantic of the Dimension.
MeasureDimension	Inherits from Dimension	A statistical concept that identifies the component in the key structure that

Class	Feature	Description
		has an enumerated list of measures. This dimension has, as its representation the Concept Scheme that enumerates the measure concepts.
TimeDimension	Inherits from Dimension	A metadata concept that identifies the component in the key structure that has the role of "time".
DataAttribute	Inherits from Component  Sub class  ReportingYear StartDay	A characteristic of an object or entity.
	/role	Association to the Concept that specifies the role that the Data Attribute plays in the Data Structure Definition.
	usageStatus	Defines the usage status which is constrained by the data type Usage Status.
	+relatedTo	Association to a Attribute Relationship.
	/conceptIdentity	An association to the Concept which defines the semantic of the component.
ReportingYearStartDay	Inherits from DataAttribute	A specialised Data Attribute whose value is used in conjunction with the predefined reporting periods in the Time Dimension. If this is not present, then by default all reporting period values for the Time Dimension will be assumed to be based on a reporting year start day of January 1.



Class	Feature	Description
PrimaryMeasure	Inherits from <i>Component</i>	The metadata concept that is the phenomenon to be measured in a data set. In a data set the instance of the measure is often called the observation.
	/conceptIdentity	An association to the Concept which carries the values of the measures.
<i>AttributeRelationship</i>	Abstract Class  Sub classes NoSpecifiedRelationship PrimaryMeasureRelationship GroupRelationship DimensionRelationship	Specifies the type of artefact to which a Data Attribute can be attached in a Data Set.
NoSpecifiedRelationship		The Data Attribute is not related to any specific construct.
PrimaryMeasureRelationship		The Data Attribute is related to the Primary Measure construct.
GroupRelationship		The Data Attribute is related to a Group Dimension Descriptor construct.
	+groupKey	An association to the Group Dimension Descriptor
DimensionRelationship		The Data Attribute is related to a set of Dimensions.
	+dimensions	Association to the set of Dimensions to which the Data Attribute is related.
	+groupKey	Association to the Group Dimension Descriptor which specifies the set of Dimensions to which the Data Attribute is attached.

1304 The explanation of the classes, attributes, and associations comprising the Representation is  
 1305 described in the section on the SDMX Base.

1306 **5.4 Data Set – Relationship View**

1307 **5.4.1 Context**

1308 A data set comprises the collection of data values and associated metadata that are collected  
 1309 or disseminated according to a known DataStructureDefinition.

1310 **5.4.2 Class Diagram**

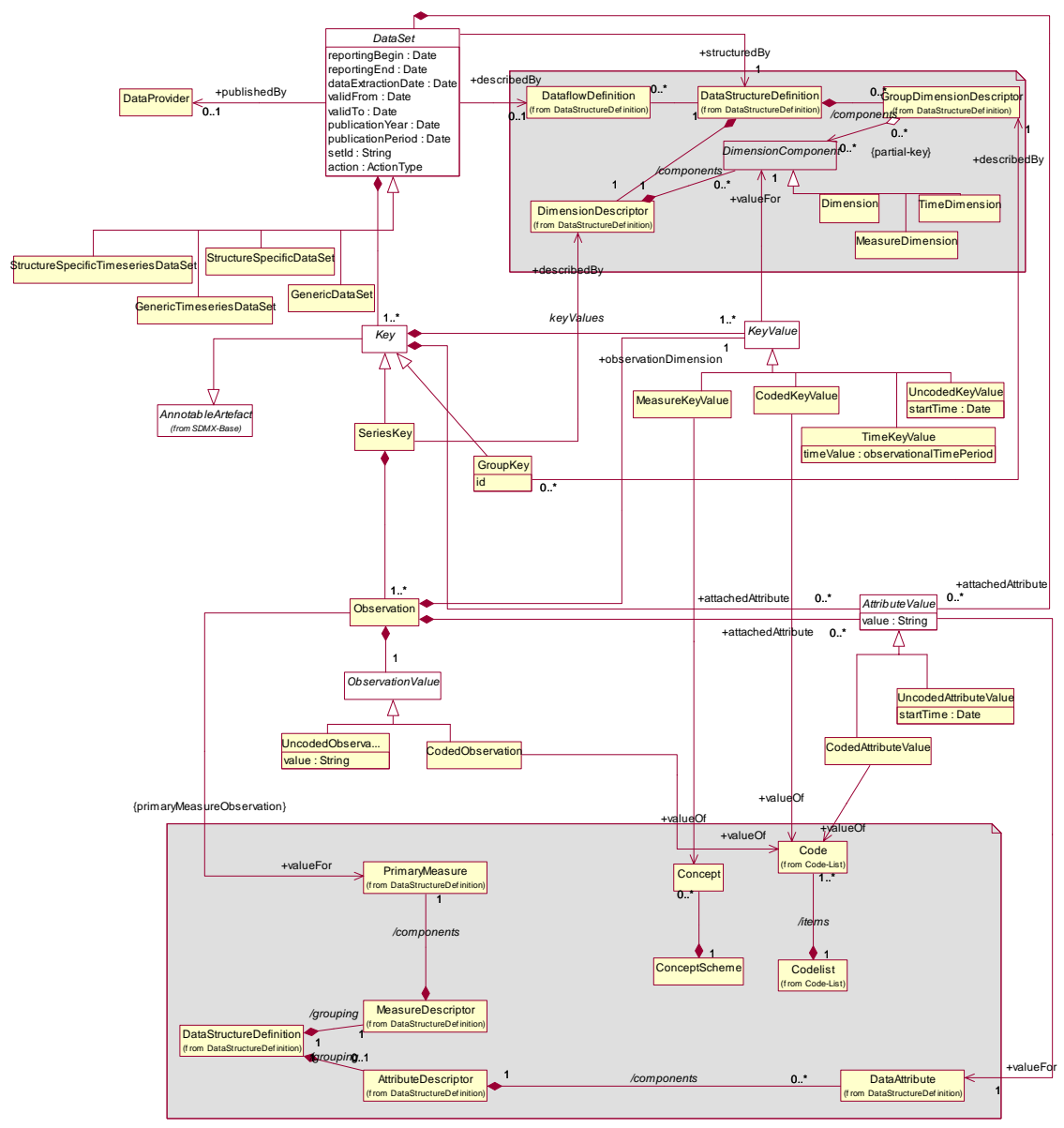


Figure 26 Class Diagram of the Data Set

1311 **5.4.3 Explanation of the Diagram**

1312 **5.4.3.1 Narrative – Data Set**

1313 Note that the *DataSet* must conform to the *DataStructureDefinition* associated to the  
1314 *DataflowDefinition* for which this *DataSet* is an “instance of data”. Whilst the model  
1315 shows the association to the classes of the *DataStructureDefinition*, this is for  
1316 conceptual purposes to show the link to the *DataStructureDefinition*. In the actual  
1317 *DataSet* as exchanged there must, of course, be a reference to the  
1318 *DataStructureDefinition* and optionally a *DataflowDefinition*, but the  
1319 *DataStructureDefinition* is not necessarily exchanged with the data. Therefore, the  
1320 *DataStructureDefinition* classes are shown in the grey areas, as these are not a part of  
1321 the *DataSet* when the *DataSet* is exchanged. However, the structural metadata in the  
1322 *DataStructureDefinition* can be used by an application to validate the contents of the  
1323 *DataSet* in terms of the valid content of a *KeyValue* as defined by the *Representation* in  
1324 the *DataStructureDefinition*.

1325

1326 An organisation playing the role of *DataProvider* can be responsible for one or more  
1327 *DataSet*.

1328

1329 A *DataSet* can be formatted either as a generic data set (*GenericDataSet*,  
1330 *GenericTimeseriesDataSet*) or a *DataStructureDefinition* specific data set  
1331 (*StructureSpecificDataSet*, *StructureSpecificTimeseriesDataSet*). The  
1332 generic data set is structured in exactly the same way no matter which  
1333 *DataStructureDefinition* the *DataSet* expresses. The structured data set is structured  
1334 according to one specific *DataStructureDefinition*. Depending on the syntax chosen for  
1335 the implementation the structured data set should support better validation at the syntax level.

1336

1337 A *DataSet* is a collection of a set of *Observations* that share the same dimensionality,  
1338 which is specified by a set of unique components (*Dimension*, *MeasureDimension*,  
1339 *TimeDimension*) defined in the *DimensionDescriptor* of the  
1340 *DataStructureDefinition*, together with associated *AttributeValues* that define  
1341 specific characteristics about the artefact to which it is attached. - *DataSet*, *Observation*,  
1342 set of *Dimensions*. It is structured in terms of a *SeriesKey* to which *Observations* are  
1343 reported.

1344

1345 The *Observation* can be the value of the variable being measured for the *Concept*  
1346 associated to the *PrimaryMeasure* in the *MeasureDescriptor* of the  
1347 *DataStructureDefinition*. This is true when there is no *MeasureDimension* that  
1348 specifies the precise meaning of each *Observation*. Each *Observation* associates an  
1349 *ObservationValue* with a *KeyValue* (+*observationDimension*) which is the value for  
1350 the “Dimension at the Observation Level”. Any dimension can be specified as being the  
1351 “Dimension at the Observation Level”, and this specification is made at the level of the  
1352 *DataSet* (i.e. it must be the same dimension for the entire *DataSet*).

1353

1354 If the “Dimension at the Observation Level” is the *MeasureDimension* it is possible (but not  
1355 mandatory) that an *Observation* can be reported with an explicit identification of one or  
1356 more *Concept* in the *ConceptScheme* referenced by the *MeasureDimension* as its  
1357 *Representation*. In other words, the actual *Concepts* are explicitly stated in the  
1358 *Observation*.

1359  
 1360 If it is required to specify explicitly that the DataSet is time series then one of  
 1361 GenericTimeSeriesDataSet or StructureSpecificTimeSeriesDataSet is used and  
 1362 the *KeyValue* for the +observationDimension must be a TimeKeyValue. In a  
 1363 GenericDataSet and a StructureSpecificDataSet it is permissible to have any  
 1364 dimension as the +observationDimension including the TimeDimension.

1365  
 1366 The *KeyValue* is a value for one of MeasureDimension, TimeDimension, or  
 1367 Dimension specified in the DataStructureDefinition. If it is a Dimension it can be  
 1368 coded (CodedKeyValue) or uncoded (UncodedKeyValue). If it is a MeasureDimension  
 1369 then it is MeasureKeyValue. If it is TimeDimension then it is a TimeKeyValue. The actual  
 1370 value that the CodedDimensionValue can take must be one of the Codes in the Codelist  
 1371 specified as the Representation of the Dimension in the DataStructureDefinition.  
 1372 The actual value that the MeasureDimensionValue can take must be a valid representation  
 1373 specified for the Concept in the ConceptScheme to which this MeasureDimensionValue  
 1374 is related (+valueFor).

1375  
 1376 The ObservationValue can be coded - this is the CodedObservation - or it can be  
 1377 uncoded - this is the UncodedObservation.

1378  
 1379 The GroupKey is a sub unit of the Key that has the same dimensionality as the SeriesKey,  
 1380 but defines a subset of the KeyValues of the SeriesKey. Its sub dimension structure is  
 1381 defined in the GroupDimensionDescriptor of the DataStructureDefinition identified  
 1382 by the same id as the GroupKey. The id identifies a "type" of group and the purpose of the  
 1383 GroupKey is to report one or more AttributeValue that are contained at this group level.  
 1384 The GroupKey is present when the GroupDimensionDescriptor is related to the  
 1385 GroupRelationship in the DataStructureDefinition. There can be many types of  
 1386 groups in a DataSet. If the Group is related to the DimensionRelationship in the  
 1387 DataStructureDefinition then the AttributeValue will be reported with the  
 1388 appropriate dimension in the SeriesKey or Observation.

1389  
 1390 In this way each of DataSet, SeriesKey, GroupKey, and Observation can have zero or  
 1391 more AttributeValue that defines some metadata about the object to which it is  
 1392 associated. The allowable Concepts and the objects to which these metadata can be  
 1393 associated (attached) are defined in the DataStructureDefinition.

1394  
 1395 The *AttributeValue* links to the object type (DataSet, SeriesKey, GroupKey,  
 1396 Observation, ) to which it is associated.

1397

1398 **5.4.3.2 Definitions**

Class	Feature	Description
-------	---------	-------------

Class	Feature	Description
<i>DataSet</i>	<p>Abstract Class</p> <p>Sub classes</p> <p>GenericDataSet StructureSpecificDataSet GenericTime SeriesDataSet StructureSpecificTime SeriesDataSet</p>	An organised collection of data.
	reportingBegin	A specific time period in a known system of time periods that identifies the start period of a report.
	reportingEnd	A specific time period in a known system of time periods that identifies the end period of a report.
	dataExtractionDate	A specific time period that identifies the date and time that the data are extracted from a data source.
	validFrom	Indicates the inclusive start time indicating the validity of the information in the data set.
	validTo	Indicates the inclusive end time indicating the validity of the information in the data set.
	publicationYear	Specifies the year of publication of the data or metadata in terms of whatever provisioning agreements might be in force.
	publicationPeriod	Specifies the period of publication of the data or metadata in terms of whatever provisioning agreements might be in force.
	setId	Provides an identification of the data set.
	action	Defines the action to be taken by the recipient system (update, append, delete)

Class	Feature	Description
	describedBy	Associates a data flow definition and thereby a Data Structure Definition to the data set.
	+structuredBy	Associates the Data Structure Definition that defines the structure of the Data Set. Note that the Data Structure Definition is the same as that associated (non-mandatory) to the Dataflow Definition.
	+publishedBy	Associates the Data Provider that reports/publishes the data.
	+attachedAttribute	Association to the Attribute Values relating to the Data Set
GenericDataSet		A data format structure that is able to contain data corresponding to any Data Structure Definition .
StructureSpecificDataSet		A data format structure that contains data corresponding to one specific Data Structure Definition .
GenericTimeseriesDataSet		A data format structure that is able to contain timeseries data corresponding to any Data Structure Definition .
StructureSpecificTimeseriesDataSet		A data format structure that contains timeseries data corresponding to one specific Data Structure Definition .
Key	Abstract class Sub classes SeriesKey GroupKey	Comprises the cross product of values of dimensions that identify uniquely an Observation.
	keyValues	Association to the individual Key Values that comprise the Key.

Class	Feature	Description
	+attachedAttribute	Association to the Attribute Values relating to the Series Key or Group Key.
<i>KeyValue</i>	Abstract class Sub classes <i>MeasureKeyValue</i> <i>TimeKeyValue</i> <i>CodedKeyValue</i> <i>UncodedKeyValue</i>	The value of a component of a key such as the value of the instance a Dimension in a Dimension Descriptor of a Data Structure Definition.
	+valueFor	Association to the key component in the Data Structure Definition for which this Key Value is a valid representation.  Note that this is conceptual association as the key component is identified explicitly in the data set.
<i>MeasureKeyValue</i>	Inherits from <i>KeyValue</i>	The value of the Measure Dimension component of the key. The value is the Concept to which this class is associated.
	+value	Association to the Concept.  Note that this is a conceptual association showing that the Concept must exist in the Concept Scheme associated with the Measure Dimension in the Data Structure Definition. In the actual Data Set the value of the Concept is placed in the Key Value.
<i>TimeKeyValue</i>	Inherits from <i>KeyValue</i>	The value of the Time Dimension component of the key.
<i>CodedKeyValue</i>	Inherits from <i>KeyValue</i>	The value of a coded component of the key. The value is the Code to which this class is associated.

Class	Feature	Description
	+value	Association to the Code.  Note that this is a conceptual association showing that the Code must exist in the Code list associated with the Dimension in the Data Structure Definition. In the actual Data Set the value of the Code is placed in the Key Value.
UnCodedKeyValue	Inherits from <i>KeyValue</i>	The value of an uncoded component of the key.
	value	The value of the key component.
	startTime	This attribute is only used if the textFormat of the attribute is of the Timespan type in the Data Structure Definition (in which case the value field takes a duration).
	+valueFor	Associates Dimension, Measure Dimension, or Time Dimension to the Key Value, and thereby to the Concept that is the semantic of the Dimension, or Time Dimension.
GroupKey	Inherits from <i>Key</i>	A set of Key Values that comprise a partial key, of the same dimensionality as the Time Series Key for the purpose of attaching Data Attributes.
	+describedBy	Associates the Group Dimension Descriptor defined in the Data Structure Definition.
SeriesKey	Inherits from <i>Key</i>	Comprises the cross product of values of all the Key Values that, together with the Key Value of the +observation Dimension identify uniquely an Observation.



Class	Feature	Description
	+describedBy	Associates the Dimension Descriptor defined in the Data Structure Definition.
Observation		The value of the observed phenomenon in the context of the Key Values comprising the key.
	+valueFor	Associates the Primary Measure defined in the Data Structure Definition.
	+attachedAttribute	Association to the Attribute Values relating to the Observation.
	+observationDimension	Association to the Key Value that holds the value of the "Dimension at the Observation Level".
<i>ObservationValue</i>	Abstract class Sub classes UncodedObservation CodedObservation	
UncodedObservation	Inherits from ObservationValue	An observation that has a text value.
	value	The value of the Uncoded Observation.
CodedObservation	Inherits from ObservationValue	An Observation that takes its value from a code in a Code list.
	+value	Association to the Code that is the value of the Observation.  Note that this is a conceptual association showing that the Code must exist in the Code list associated with the Primary Measure or the Concept of the Measure Dimension in the Data Structure Definition. In the actual Data Set the value of the Code is placed in the Observation.

Class	Feature	Description
<i>AttributeValue</i>	<p>Abstract class</p> <p>Sub classes <i>UncodedAttributeValue</i> <i>CodedAttributeValue</i></p>	The value of an attribute, such as the instance of a Coded Attribute or of an Uncoded Attribute in a structure such as a Data Structure Definition.
	value	The value of the attribute.
	+valueFor	Association to the Data Attribute defined in the Data Structure Definition. Note that this is conceptual association as the Concept is identified explicitly in the data set.
<i>UncodedAttribute Value</i>	Inherits from <i>AttributeValue</i>	An attribute value that has a text value.
	startTime	This attribute is only used if the textFormat of the attribute is of the Timespan type in the Data Structure Definition (in which case the value field takes a duration).
<i>CodedAttribute Value</i>	Inherits from <i>AttributeValue</i>	An attribute that takes its value from a Code in Code list.
	+value	<p>Association to the Code that is the value of the Attribute Value.</p> <p>Note that this is a conceptual association showing that the Code must exist in the Code list associated with the Data Attribute in the Data Structure Definition. In the actual Data Set the value of the Code is placed in the Attribute Value.</p>

## 1400 **6 Cube**

### 1401 **6.1 Context**

1402 Some statistical systems create views of data based on a “cube” structure. In essence, a cube  
1403 is an n-dimensional object where the value of each dimension can be derived from a  
1404 hierarchical code list. The utility of such cube systems is that it is possible to “roll up” or “drill  
1405 down” each of the hierarchy levels for each of the dimensions to specify the level of granularity  
1406 required to give a “view” of the data – some dimensions may be rolled up, others may be  
1407 drilled down. Such systems give a dynamic view of the data, with aggregated values for rolled  
1408 up dimension positions. For example, the individual countries may be rolled up into an  
1409 economic region such as the EU, or a geographical region such as Europe, whilst another  
1410 dimension, such as “type of road” may be drilled down to its lower level. The resulting  
1411 measure (such as “number of accidents”) would then be an aggregation of the value for each  
1412 individual country for the specific type of road.

1413

1414 Such cube systems rely, not on simple code lists, but on hierarchical code sets (see section  
1415 8).

### 1416 **6.2 Support for the Cube in the Information Model**

1417 Data reported using a Data Structure Definition structure (where each dimension value, if  
1418 coded, is taken from a flat code list) can be described by a cube definition and can be  
1419 processed by cube aware systems. The SDMX-IM supports the definition of such cubes in the  
1420 following way:

1421

1422 • The `HierarchicalCodeList` defines the (often complex) hierarchies of codes

1423 • If required, the `StructureSet` can

1424     ○ group `DataStructureDefinition` that describe the cube

1425     ○ provide a mapping mechanism between the codes in the flat code lists used by  
1426     the `DataStructureDefinition` and a `HierarchicalCodeList` where  
1427     the `HierarchicalCodeList` uses code lists that are not used in the  
1428     `DataStructureDefinition`

1429

## 1430 **7 Metadata Structure Definition and Metadata Set**

### 1431 **7.1 Context**

1432 The SDMX metamodel allows metadata:

1433

1434 1. To be exchanged without the need to embed it within the object that it is describing.

1435

1436 2. To be stored separately from the object that it describes, yet be linked to it (for  
1437 example, an organisation has a metadata repository which supports the dissemination  
1438 of metadata resulting from metadata requests generated by systems or services that  
1439 have access to the object for which the metadata pertains. This is common in web  
1440 dissemination where additional metadata is available for viewing (and eventually  
1441 downloading) by clicking on an “information” icon next to the object to which the  
1442 metadata is attached).

1443

1444 3. To be indexed to aid searching (example: a registry service can process a metadata  
1445 report and extract structural information that allows it to catalogue the metadata in a  
1446 way that will enable users to query for it).

1447

1448 4. To be reported according to a defined structure.

1449

1450 In order to achieve this, the following structures are modelled:

1451

1452 • metadata structure definition which has the following components:

1453 ○ the object types to which the metadata are to be associated (attached)

1454 ○ the components that, together, comprise a unique key of the object type to  
1455 which the metadata are to be associated

1456 ○ the reporting structure comprising the metadata attributes that can be attached  
1457 to the various object types (these attributes can be structured in a hierarchy),  
1458 together with any constraints that may apply (e.g. association to a code list that  
1459 contains valid values for the attribute when reported in a metadata set)

1460 • the metadata set, which contains reported metadata

### 1461 **7.2 Inheritance**

#### 1462 **7.2.1 Introduction**

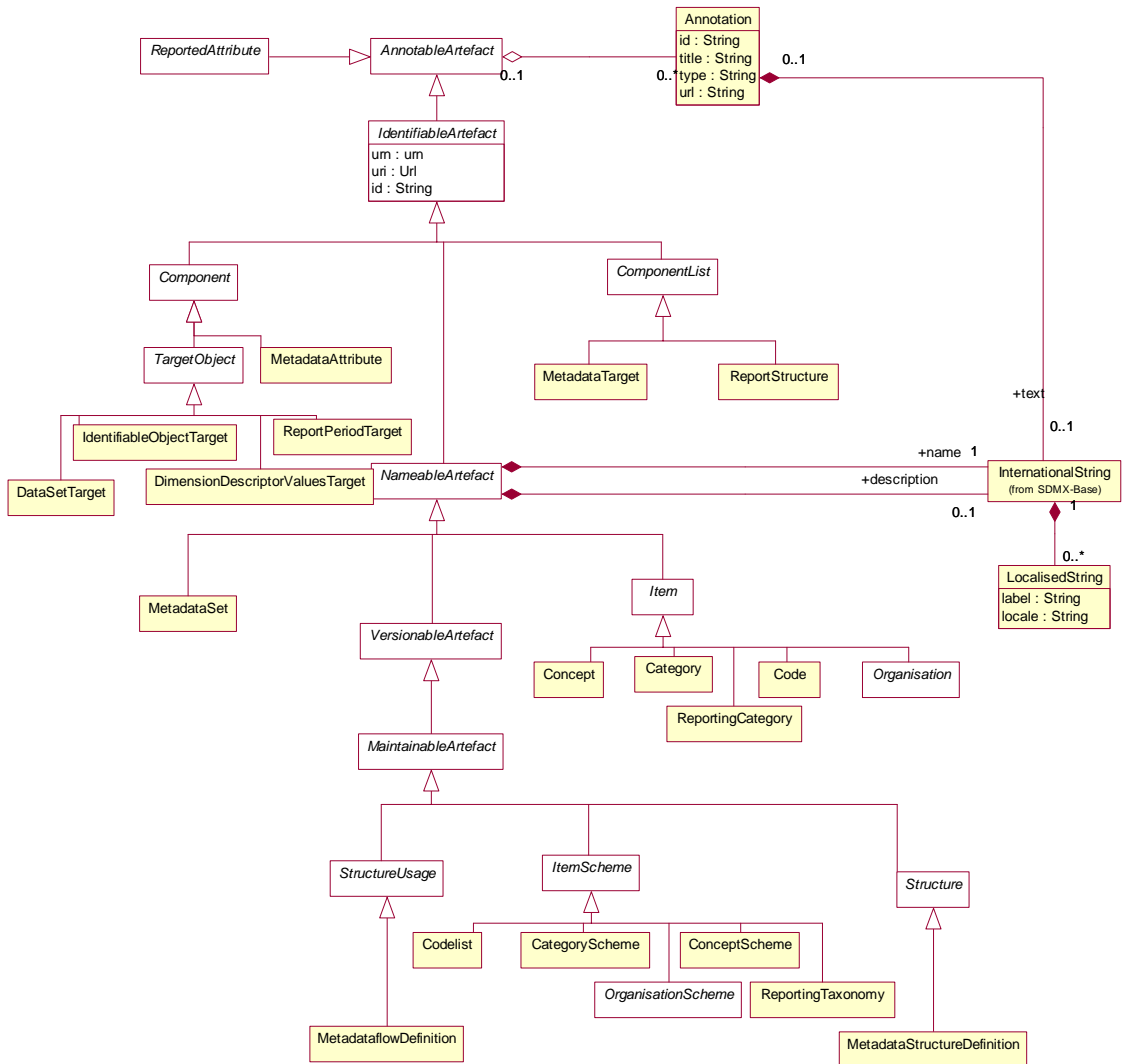
1463 As with the Data Structure Definition Structure, many of the constructs in this layer of the  
1464 model inherit from the SDMX Base layer. Therefore, it is necessary to study both the  
1465 inheritance and the relationship diagrams to understand the functionality of individual  
1466 packages. The diagram below shows the full inheritance tree for the classes concerned with  
1467 the `MetadataStructureDefinition` and the `MetadataSet`.

1468

1469 There are very few additional classes in the `MetadataStructureDefinition` package that  
1470 do not themselves inherit from classes in the SDMX Base. In other words, the SDMX Base  
1471 gives most of the structure of this sub model both in terms of associations and in terms of

1472 attributes. The relationship diagrams shown in this section show clearly when these  
 1473 associations are inherited from the SDMX Base (see the Appendix “A Short Guide to UML in  
 1474 the SDMX Information Model” to see the diagrammatic notation used to depict this). It is  
 1475 important to note that SDMX base structures used for the MetadataStructureDefinition  
 1476 are the same as those used for the DataStructureDefinition and so, even though the  
 1477 usage is slightly different, the underlying way of defining a  
 1478 MetadataStructureDefinition is similar to that used for defining a  
 1479 DataStructureDefinition.

1480 **7.2.2 Class Diagram - Inheritance**



1481

1482

**Figure 27: Inheritance class diagram of the Metadata Structure Definition**

1483 **7.2.3 Explanation of the Diagram**

1484 **7.2.3.1 Narrative**

1485 It is important to the understanding of the relationship class diagrams presented in this section  
1486 to identify the concrete classes that inherit from the abstract classes.

1487

1488 The concrete classes in this part of the SDMX metamodel which require to be maintained by  
1489 Maintenance Agencies all inherit from `MaintainableArtefact`. These are:

1490

1491 • `StructureUsage` (concrete class is `MetadataflowDefinition`)

1492

• `Structure` (concrete class is `MetadataStructureDefinition`)

1493

These classes also inherit the identity and versioning facets of `IdentifiableArtefact`,  
1494 `NameableArtefact`, and `VersionableArtefact`.

1495

1496 A `Structure` contains several lists of components. The concrete classes which inherit from  
1497 `ComponentList` and in themselves are sub components of the  
1498 `MetadataStructureDefinition` are:

1499

1500 • `MetadataTarget`

1501

• `ReportStructure`

1502

`ComponentList` contains `Components`. The classes that inherit from `Component` are:

1503

1504 • Sub Classes of `TargetObject`

1505

• `MetadataAttribute`

1506 **7.3 Metadata Structure Definition**

1507 **7.3.1 Introduction**

1508 The diagrams and explanations in the rest of this section show how these concrete classes  
1509 are related so as to support the functionality required.

1510 **7.3.2 Structures Already Described**

1511 The `MetadataStructureDefinition` makes use of the following `ItemScheme` structures  
1512 either as explicit concrete classes in the model, or as possible lists which comprise the value  
1513 domain of a `TargetObject`.

1514

1515 • `CategoryScheme`

1516

• `ConceptScheme`

1517

• `Codelist`

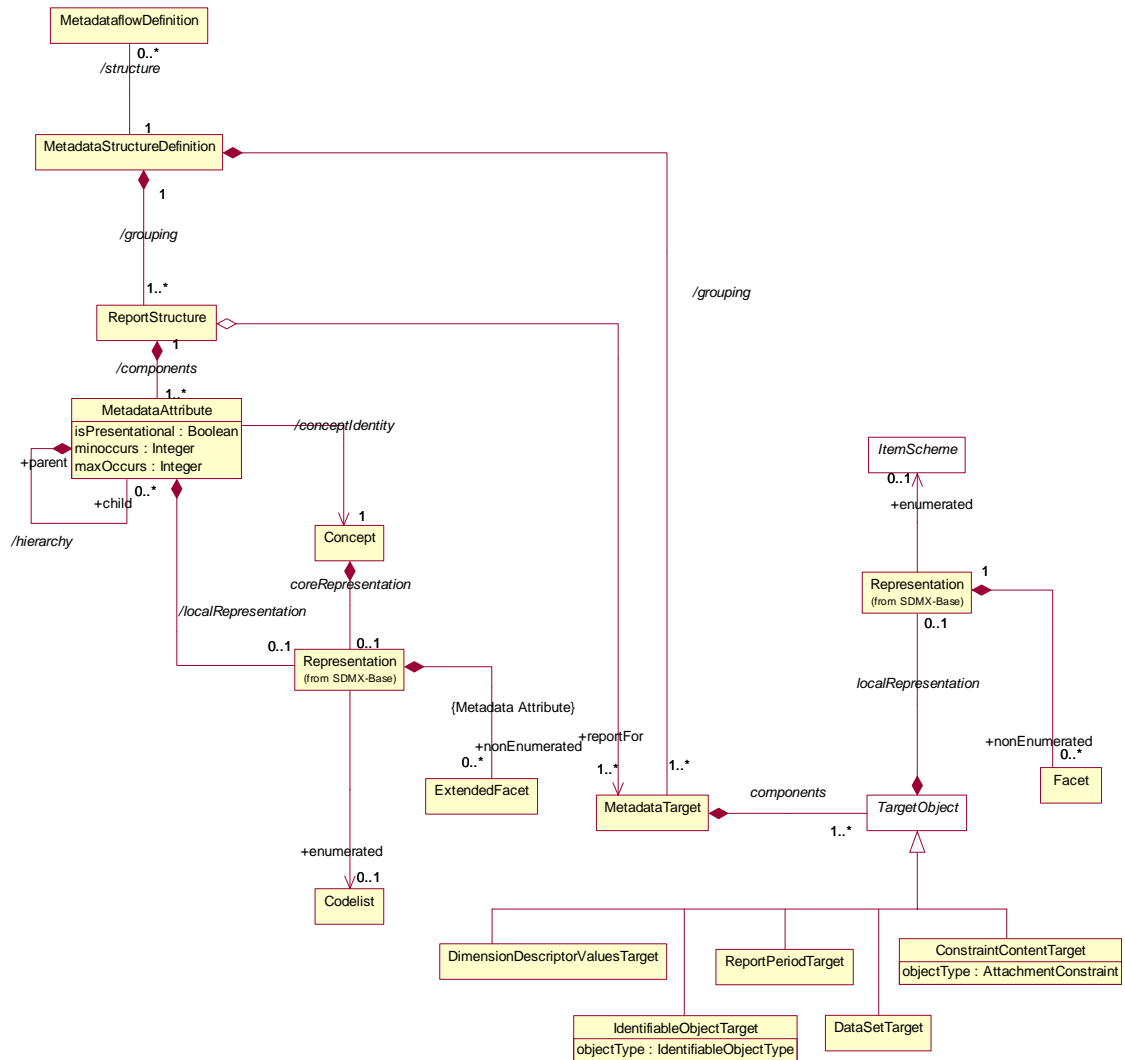
1518

• `OrganisationScheme`

1519

• `Reporting Taxonomy`

1520 **7.3.3 Class Diagram – Relationship**



1521

1522

**Figure 28: Relationship class diagram of the Metadata Structure Definition**

1523 **7.3.4 Explanation of the Diagram**

1524 **7.3.4.1 Narrative**

1525 In brief a MetadataStructureDefinition (MSD) defines:

1526

1527

1528

1529

- The MetadataTarget which defines the components (*TargetObject*) and their Representation which are valid for this MetadataStructureDefinition, and which are the metadata target object of one or more ReportStructure
- The ReportStructures comprising the MetadataAttributes that can be associated with the object type identified in the referenced MetadataTargets, and hierarchical structure of the attributes

1530

1531

1532

1533 The `MetadataTarget` comprises one or more `TargetObjects`. The combination of  
1534 `TargetObjects` identifies a specific object type to which metadata can be attached in a  
1535 `MetadataSet`.

1536

1537 The `TargetObject` is one of the following:

1538

1539 • `DimensionDescriptorValuesTarget` - this allows the specification of a full or  
1540 partial key (as used in a dataset) to be specified in a `MetadataSet` as the target  
1541 object

1542 • `IdentifiableObjectTarget` – this defines a specific object type, which can be any  
1543 `IdentifiableArtefact`

1544 • `DataSetTarget` – this specifies that the target object is a `DataSet`

1545 • `ReportPeriodTarget` - this specifies that the report period must be present in the  
1546 `MetadataSet`

1547 • `ConstraintContentTarget` – this specifies that target object is the content of an  
1548 `AttachmentConstraint` i.e. the part of the data set or metadata set identified by the  
1549 content of an `AttachmentConstraint`

1550 The valid content of a `TargetObject` when reported in a `MetadataSet` is defined in the  
1551 `Representation`. This can be an enumerated representation (i.e. a reference to one of the  
1552 sub classes of `ItemScheme` – these are `Codelist`, `ConceptScheme`,  
1553 `OrganisationScheme`, `CategoryScheme`, or `ReportingTaxonomy`) or non-  
1554 enumerated.

1555

1556 Thus a single `MetadataStructureDefinition` can be defined for a discrete set of related  
1557 object types. For example, a single definition can be constructed to define the metadata that  
1558 can be attached to any part of a `Data Structure Definition`, or that can be attached to  
1559 any artefact concerned with the reporting of quality metadata (such as data provider and  
1560 (data) category). The `MetadataTarget` specifies the identification properties of a specific  
1561 object type to which metadata can be attached in a `MetadataSet`. For example, in a  
1562 `DataStructureDefinition` the `MetadataTarget` might be a `Dimension`, and therefore  
1563 the `TargetObjects` are those that uniquely identify a `Dimension`. This will include both the  
1564 `DataStructureDefinition` and the `Dimension` (both of these are an  
1565 `IdentifiableArtefact` and will use the `IdentifiableObjectTarget`) as both  
1566 `TargetObjects` are required in order to identify uniquely a `Dimension`).

1567

1568 The `ReportStructure` comprises a set of `MetadataAttributes` - these can be defined  
1569 as a hierarchy. Each `MetadataAttribute` identifies a `Concept` that is reported or  
1570 disseminated in a `MetadataSet` (`/conceptIdentity`) that uses this  
1571 `MetadataStructureDefinition`. Different `MetadataAttributes` in the same  
1572 `ReportStructure` can use `Concepts` from different `ConceptSchemes`. Note that a  
1573 `MetadataAttribute` does not link to a `Concept` that defines its role in this  
1574 `MetadataStructureDefinition` (i.e. the `MetadataAttribute` does not play a role).  
1575



1576 The `MetadataAttribute` can be specified as having multiple occurrences and/or specified  
 1577 as being mandatory (`minOccurs=1` or more) or conditional (`minOccurs=0`). A hierarchical  
 1578 `ReportStructure` can be defined by specifying a hierarchy for a `MetadataAttribute`.

1579  
 1580 The `ReportStructure` is associated to one or more of the `MetadataTargets` which  
 1581 specify to which object the `MetadataAttributes` specified in the `ReportStructure` are  
 1582 attached when reported in a `MetadataSet`.

1583  
 1584 It can be seen from this that the specification of the object types to which a  
 1585 `MetadataAttribute` can be attached is indirect: the `MetadataAttributes` are defined in  
 1586 a `ReportStructure` which itself is attached to one or more `MetadataTarget` and the  
 1587 actual object is identified by the `TargetObjects` comprising the `MetadataTarget`. This  
 1588 gives a flexible mechanism by which the actual object types need not be defined in concrete  
 1589 terms in the model, but are defined dynamically in the `MetadataStructureDefinition`,  
 1590 in much the same way as the keys to which data observation are “attached” in a  
 1591 `DataStructureDefinition`. In this way the `MetadataStructureDefinition` can be  
 1592 used to define any set of `MetadataAttributes` and any set of object types to which they  
 1593 can be attached.

1594  
 1595 Each `MetadataAttribute` can have a `Representation` specified (using the  
 1596 `/localRepresentation` association). If this is not specified in the  
 1597 `MetadataStructureDefinition` then the `Representation` is taken from that defined  
 1598 for the `Concept` (the `coreRepresentation` association).

1599  
 1600 The definition of the various types of `Representation` can be found in the specification of  
 1601 the `Base` constructs. Note that if the `Representation` is non-enumerated then the  
 1602 association is to the `ExtendedFacet` (which allows for `xhtml` as a `FacetValueType`). If the  
 1603 `Representation` is enumerated then it must use a `Codelist`.

1604  
 1605 The `MetadataStructureDefinition` is linked to a `MetadataflowDefinition`. The  
 1606 `MetadataflowDefinition` does not have any attributes in addition to those inherited from  
 1607 the `Base` classes.

1608

#### 1609 7.3.4.2 Definitions

Class	Feature	Description
<code>StructureUsage</code>		See “SDMX Base”.
<code>Metadataflow Definition</code>	Inherits from: <i>StructureUsage</i>	Abstract concept (i.e. the structure without any metadata) of a flow of metadata that providers will provide for different reference periods.
	<code>/structure</code>	Associates a <code>Metadata Structure Definition</code> .

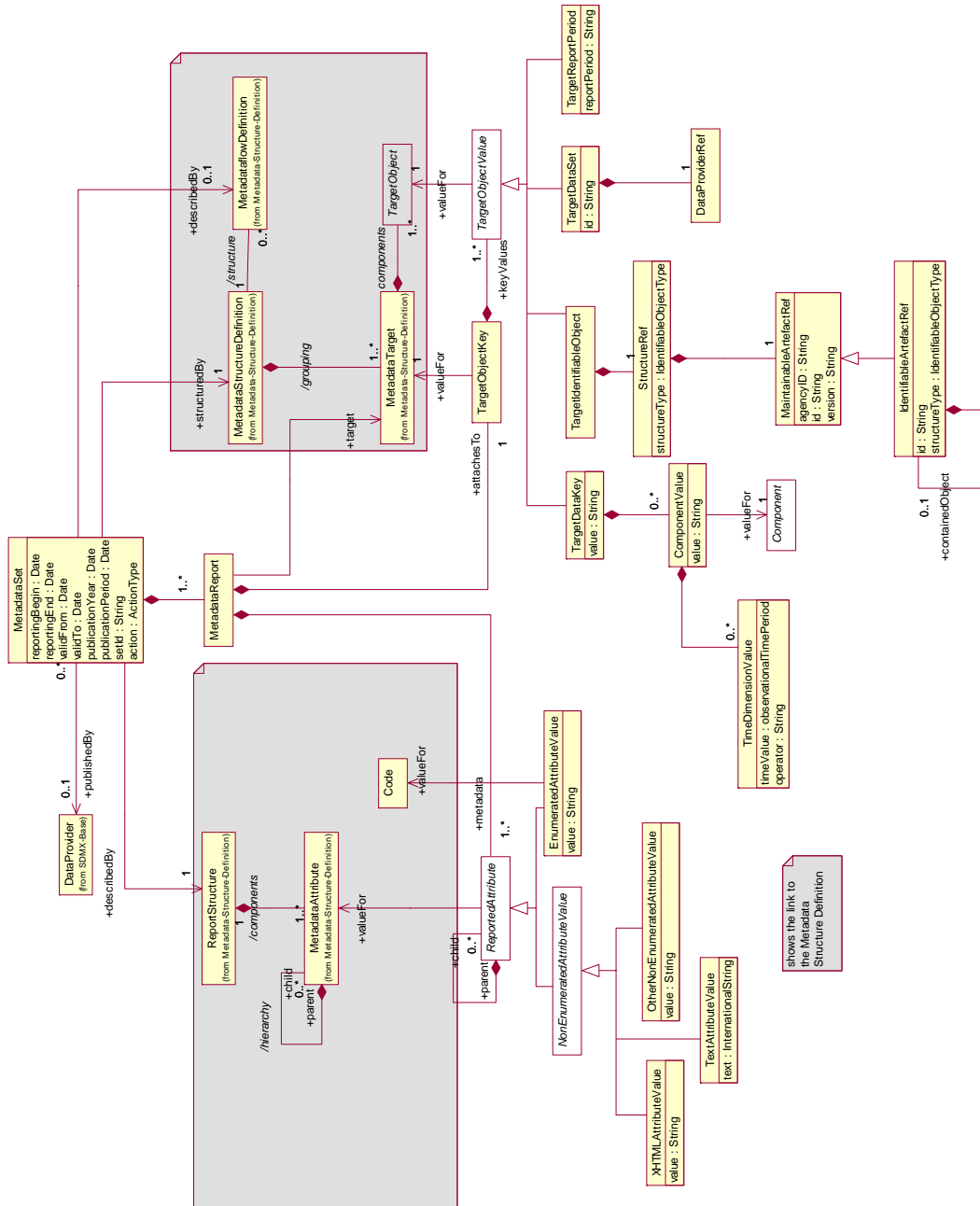
Class	Feature	Description
MetadataStructure Definition		A collection of metadata concepts, their structure and usage when used to collect or disseminate reference metadata.
	/grouping	An association to a Metadata Target or Report Structure.
MetadataTarget	Inherits from <i>ComponentList</i>	A set of components that define a key of an object type to which metadata may be attached.
	/components	Associates the Target Object components that define the key of the Metadata Target.
<i>TargetObject</i>	Abstract Class  Sub Classes DimensionDescriptorValues Target IdentifiableObjectTarget DataSetTarget ReportPeriodTarget	
	/localRepresentation	Associates a Representation to the Target Object that must be respected when the object is identified in a Metadata Set. This may be enumerated or non-enumerated.
DimensionDescriptor ValuesTarget	Inherits from <i>TargetObject</i>	The target object is the key of a data series.
IdentifiableObject Target	Inherits from <i>TargetObject</i>	The target object is a specified object type.
	objectType	Identifies the object type.
DataSetTarget	Inherits from <i>TargetObject</i>	The target object is a Data Set.

Class	Feature	Description
ReportPeriodTarget	Inherits from <i>TargetObject</i>	The target is a report period. Note that this does not describe the use of an object, but rather serves as a unique metadata key for metadata reports. Metadata reports attached to a particular object may vary over time, and this time identifier component can be used to disambiguate the reports, much like the time dimension disambiguates observations in a data series.
ConstraintTarget	Inherits from <i>TargetObject</i>	The target object is the data or reference metadata that is identified in the content of an Attachment Constraint.
ReportStructure	Inherits from: <i>ComponentList</i>	Defines a set of concepts that comprises the Metadata Attributes to be reported.
	/components	An association to the Metadata Attributes relevant to the Report Structure.
	+reportFor	Associates the Metadata Targets for which this Report Structure is used.
MetadataAttribute		Identifies a Concept for which a value may be reported in a Metadata Set.
	/hierarchy	Association to one or more child Metadata Attribute.
	/conceptIdentity	An association to the concept which defines the semantic of the attribute.

Class	Feature	Description
	isPresentational	Indication that the Metadata Attribute is present for structural purposes (i.e. it has child attributes) and that no value for this attribute is expected to be reported in a Metadata Set using this Report Structure.
	minOccurs maxOccurs	Specifies how many occurrences of the Metadata Attribute may be reported at this point in the Metadata Report.
ConceptUsage		The use of a Concept as Metadata Attribute.
	concept	Association to a Concept in a ConceptScheme.
	/localRepresentation	Associates a Representation that overrides any core representation specified for the Concept itself.
Representation		The representation of the Metadata Attribute.

1610 **7.4 Metadata Set**

1611 **7.4.1 Class Diagram**



1612

1613

**Figure 29: Relationship Class Diagram of the Metadata Set**

1614 **7.4.2 Explanation of the Diagram**

1615 **7.4.2.1 Narrative**

1616 Note that the `MetadataSet` must conform to the `MetadataStructureDefinition`  
 1617 associated to the `MetadataflowDefinition` for which this `MetadataSet` is an “instance  
 1618 of metadata”. Whilst the model shows the association to the classes of the  
 1619 `MetadataStructureDefinition`, this is for conceptual purposes to show the link to the  
 1620 `MetadataStructureDefinition`. In the actual `MetadataSet` as exchanged there must,  
 1621 of course, be a reference to the `MetadataStructureDefinition` and the  
 1622 `ReportStructure`, and optionally a `MetadataflowDefinition`, but the  
 1623 `MetadataStructureDefinition` is not necessarily exchanged with the metadata.  
 1624 Therefore, the `MetadataStructureDefinition` classes are shown in the grey areas, as  
 1625 these are not a part of the `MetadataSet` itself.

1626  
 1627 An organisation playing the role of `DataProvider` can be responsible for one or more  
 1628 `MetadataSet`.

1629  
 1630 A `MetadataSet` comprises one or more `MetadataReport`, each of which must be for the  
 1631 same `ReportStructure`. It references both a `MetadataTarget`, defined in the  
 1632 `MetadataStructureDefinition`, and contains a `TargetObjectKey` and  
 1633 `ReportedAttributes`.

1634  
 1635 The identified `ReportStructure` specifies which `MetadataAttributes` are expected as  
 1636 `ReportedAttributes`. The identified `MetadataTarget` specifies the expected content of  
 1637 the `TargetObjectKey` i.e. it specifies the information required to identify the object for  
 1638 which the `ReportedAttributes` are reported.

1639  
 1640 The `TargetObjectValue` can be one of:

- 1641
- 1642 • `TargetDataKey` – this can contain:
    - 1643 ○ a `SeriesKey` (set of dimension values)
    - 1644 ○ a `SeriesKey` plus a value or values (giving time range) for the  
 1645 `TimeDimension` (`TimeDimensionValue`)
    - 1646 ○ a value of values for the `TimeDimension`
  - 1647 • `TargetIdentifiableObject` –this identifies any identifiable object (which includes  
 1648 both `Maintainable` and `Identifiable` objects
  - 1649 • `TargetDataSet` – this identifies a `DataSet`
  - 1650 • `TargetReportPeriod` – this specifies the report period for the `Report`

1651  
 1652 A simple text value for the `ReportedAttribute` uses the  
 1653 `NonEnumeratedAttributeValue` sub class of `ReportedAttribute` whilst a coded value  
 1654 uses the `EnumeratedAttributeValue` sub class.

1655  
 1656 The `NonEnumeratedAttributeValue` can be one of:

- 1657
- 1658 • `XHTMLAttributeValue` – the content is XHTML
  - 1659 • `TextAttributeValue` – the content is textual and may contain the text in multiple  
 1660 languages

- 1661 • OtherNonEnumeratedAttributeValue – the content is a string value that must  
 1662 conform to the Representation specified for the MetadataAttribute in the  
 1663 MetadataStructureDefinition for the relevant ReportStructure  
 1664

1665 The EnumeratedAttributeValue contains a value for a Code specified as the  
 1666 Representation for the MetadataAttribute in the MetadataStructureDefinition  
 1667 for the relevant ReportStructure.

1668 **7.4.2.2 Definitions**

Class	Feature	Description
MetadataSet		Any organised collection of metadata.
	reportingBegin	A specific time period in a known system of time periods that identifies the start period of a report.
	reportingEnd	A specific time period in a known system of time periods that identifies the end period of a report.
	dataExtractionDate	A specific time period that identifies the date and time that the data are extracted from a data source.
	validFrom	Indicates the inclusive start time indicating the validity of the information in the data set.
	validTo	Indicates the inclusive end time indicating the validity of the information in the metadata set.
	publicationYear	Specifies the year of publication of the data or metadata in terms of whatever provisioning agreements might be in force.
	publicationPeriod	Specifies the period of publication of the data or metadata in terms of whatever provisioning agreements might be in force.
	setId	Provides an identification of the metadata set.

Class	Feature	Description
	action	Defines the action to be taken by the recipient system (update, replace, delete)
	+describedBy	Associates a Metadataflow Definition to the Metadata Set.
	+structuredBy	Associates the Metadata Structure Definition that defines the structure of the Metadata Set. Note that the Metadata Structure Definition is the same as that associated (non-mandatory) to the Metadataflow Definition.
	+publishedBy	Associates the Data Provider that reports/publishes the metadata.
	+describedBy	Reference to the Report Structure.
MetadataReport		A set of values for Metadata Attributes defined in a Report Structure of a Metadata Structure Definition.
	+attachesTo	Associates the object key to which metadata is to be attached.
	+target	Associates the Metadata Target that defines the target object to which the metadata are to be associated.
	+metadata	Associates the Reported Attribute values which are to be associated with the object or objects identified by the Target Object Key.
TargetObjectKey		Identifies the key of the object to which the metadata are to be attached.



Class	Feature	Description
	+valueFor	Associates the Metadata Target that identifies the object type and the component structure of the Target Object Key.  Note that this is a conceptual association showing the link to the MSD construct.
	+keyValues	Associates the Target Object Values of the Target Object Key.
<i>TargetObjectValue</i>	Abstract class Sub classes are  TargetDataKey TargetIdentifiableObject TargetDataSet TargetReportPeriod	The key of an individual object of the type specified in the Metadata Target of the Metadata Structure Definition.
	+valueFor	Associates the Target Object for which this value is provided.  Note that this is a conceptual association showing the link to the MSD construct.
TargetDataKey	Inherits from <i>TargetObjectValue</i>	The identification of the components and the values that form the data or metadata key.
ComponentValue		Collectively contain the identification of the components and the values that form the data key.
value		The key value.
	+valueFor	Associates the Component for which the value is declared.
TimeDimensionValue		Contains identification of the Time Dimension and the value.
TargetIdentifiableObject	Inherits from <i>TargetObjectValue</i>	Specifies the identification of an Identifiable object.

Class	Feature	Description
StructureRef		Contains the identification of an Identifiable object.
	structureType	The object type of the target object.
Maintainable ArtefactRef  Identifiable ArtefactRef		Identification of the target object by means of its identifier constructs i.e agency ID, id, version for Maintainable Object plus, for Identifiable Object, the id.
	+containedObject	Association to a contained object in a hierarchy of Identifiable Objects such as a Transition in a Process Step.
TargetDataSet	Inherits from <i>TargetObjectValue</i>	Contains the identification of a Data Set
TargetReportPeriod	Inherits from <i>TargetObjectValue</i>	Contains the period covered by the Metadata Report.
<i>ReportedAttribute</i>	Abstract class Sub classes are: <i>NonEnumeratedAttributeValue</i> <i>EnumeratedAttributeValue</i>	The value for a Metadata Attribute.
	+valueFor	Association to the Metadata Attribute in the Metadata Structure Definition that identifies the Concept and allowed Representation for the Reported Attribute.  Note that this is a conceptual association showing the link to the MSD construct. The syntax for the Reported Attribute will state, in some form, the id of the Metadata Attribute.
	+child	Association to a child Reported Attribute consistent with the hierarchy defined in the Report Structure for the Metadata Attribute for which this child is a Reported Attribute.

Class	Feature	Description
<i>NonEnumerated AttributeValue</i>	Inherits from <i>ReportedAttribute</i>  Sub class: <i>XHTMLAttributeValue</i> <i>TextAttributeValue</i> <i>OtherNonEnumerated AttributeValue</i>	The content of a Reported Attribute where this is textual.
XHTMLAttributeValue		This contains XHTML.
	value	The string value of the XHTML.
TextAttributeValue		This value of a Reported Attribute where the content is human-readable text.
	text	The string value is text. This can be present in multiple language versions.
OtherNonEnumerated AttributeValue		The value of a Reported Attribute where the content is not of human-readable text.
	value	A text string that is consistent in format to that defined in the Representation of the Metadata Attribute for which this is a Reported Attribute.
EnumeratedAttribute Value	Inherits from <i>MetadataAttributeValue</i>	The content of a Reported Attribute that is taken from a Code in a Code list.
	value	The Code value of the Reported Attribute.

Class	Feature	Description
	+value	<p>Association to a Code in the Code list specified in the Representation of the Metadata Attribute for which this Reported Attribute is the value</p> <p>Note that this shows the conceptual link to the Item that is the value. In reality, the value itself will be contained in the Enumerated Attribute Value.</p>

1669

## 1670 **8 Hierarchical Code List**

### 1671 **8.1 Scope**

1672 The `Codelist` described in the section on structural definitions supports a simple hierarchy of  
1673 `Codes`, and restricts any child `Code` to having just one parent `Code`. Whilst this structure is  
1674 useful for supporting the needs of the `DataStructureDefinition` and the  
1675 `MetadataStructureDefinition`, it may not be sufficient for supporting the more complex  
1676 associations between codes that are often found in coding schemes such as a classification  
1677 scheme. Often, the `Codelist` used in a `DataStructureDefinition` is derived from a  
1678 more complex coding scheme. Access to such a coding scheme can aid applications, such as  
1679 OLAP applications or data visualisation systems, to give more views of the data than would be  
1680 possible with the simple `Codelist` used in the `DataStructureDefinition`.

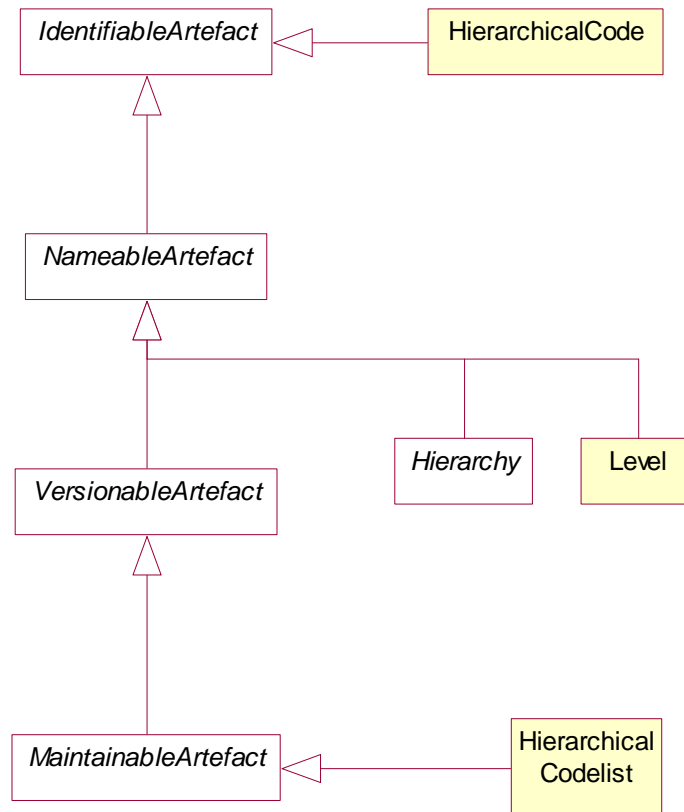
1681  
1682 Note that a hierarchical code list is not necessarily a balanced tree. A balanced tree is where  
1683 levels are pre-defined and fixed, (i.e. a level always has the same set of codes, and any code  
1684 has a fixed parent and child relationship to other codes). A statistical classification is an  
1685 example of a balanced tree, and the support for a balanced hierarchy is a sub set, and special  
1686 case, of the hierarchical code list.

1687  
1688 The principal features of the Hierarchical `Codelist` are:

- 1689 1. A child code can have more than one parent.
- 1690 2. There can be more than one code that has no parent (i.e. more than one “root node”).
- 1691 3. There may be many hierarchies (or “views”) defined, in terms of the associations  
1692 between the codes. Each hierarchy serves a particular purpose in the reporting,  
1693 analysis, or dissemination of data.
- 1694 4. The levels in a hierarchy can be explicitly defined or they can be implicit: (i.e. they  
1695 exist only as parent/child relationships in the coding structure).
- 1696
- 1697
- 1698
- 1699

1700 **8.2 Inheritance**

1701 **8.2.1 Class Diagram**



1702

1703 **Figure 30: Inheritance class diagram for the Hierarchical Codelist**

1704 **8.2.2 Explanation of the Diagram**

1705 **8.2.2.1 Narrative**

1706

1707 The `HierarchicalCodelist` inherits from `MaintainableArtefact` and thus has  
 1708 identification, naming, versioning and a maintenance agency. Both `Hierarchy` and `Level`  
 1709 are a `NameableArtefact` and therefore have an `Id`, multi-lingual name and multi-lingual  
 1710 description. A `HierachicalCode` is an `IdentifiableArtefact`.

1711

1712 It is important to understand that the `Codes` participating in a `HierarchicalCodelist` are  
 1713 not themselves contained in the list – they are referenced from the list and are maintained in  
 1714 one or more `Codelists`. This is explained in the narrative of the relationship class diagram  
 1715 below..

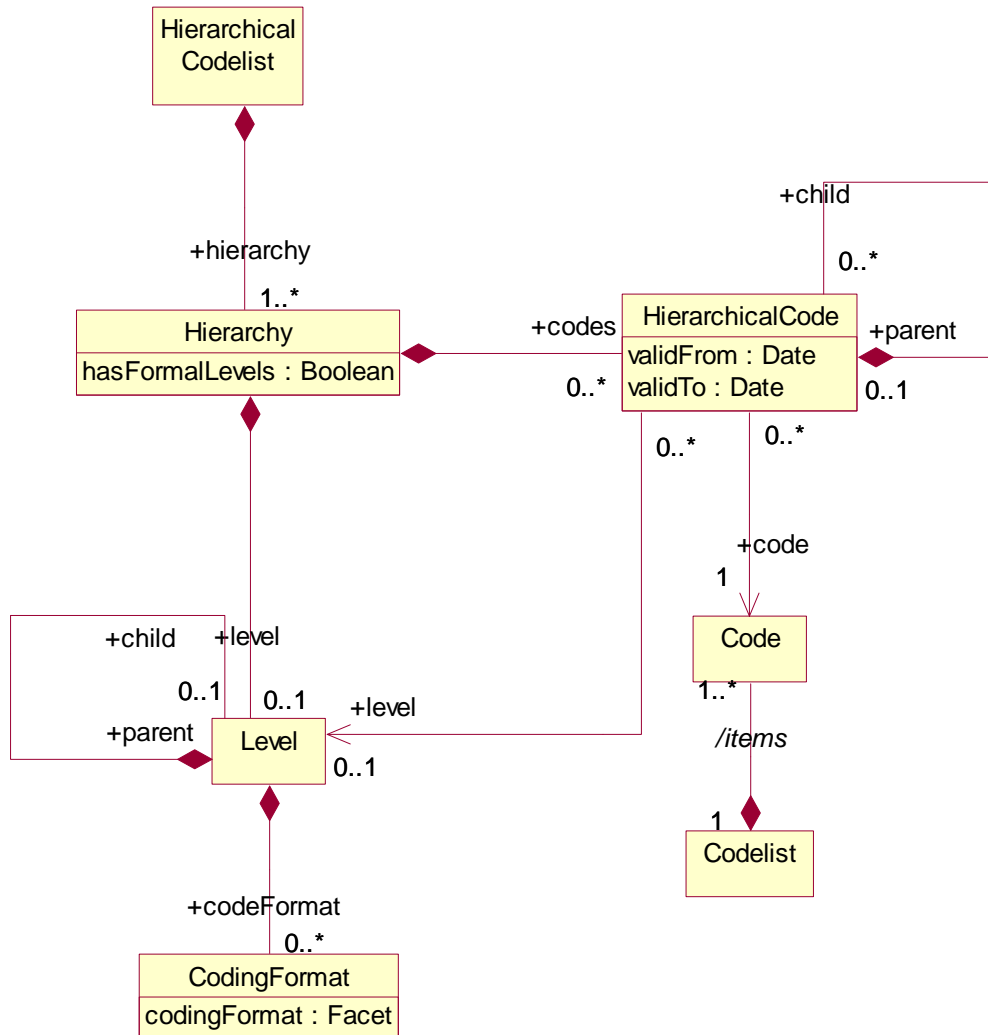
1716 **8.2.2.2 Definitions**

1717 The definitions of the various classes, attributes, and associations are shown in the  
 1718 relationship section below.

1719

1720 **8.3 Relationship**

1721 **8.3.1 Class Diagram**



1722

1723

**Figure 31: Relationship class diagram of the Hierarchical Code Scheme**

1724 **8.3.2 Explanation of the Diagram**

1725 **8.3.2.1 Narrative**

1726 The basic principles of the HierarchicalCodelist are:

1727

1728 1. The HierarchicalCodelist is a specification of the Codes comprising the scheme  
 1729 and the specification of the structure of the Codes in the scheme in terms of one or  
 1730 more Hierarchy.

1731

1732 2. The Codes in the HierarchicalCodelist are not themselves a part of the scheme,  
 1733 rather they are references to Codes in one or more external Codelists.

1734

- 1735 3. Any individual Code may participate in many Hierarchys, in order to give structure to  
 1736 the HierarchicalCodeList.  
 1737  
 1738 4. The Hierarchy of Codes is specified in HierarchicalCode. This references the  
 1739 Code and its immediate child HierarchicalCodes.  
 1740

1741 A Hierarchy can have formal levels (hasFormalLevels="true"). However, even if  
 1742 hasFormalLevels="false" the Hierarchy can still have one or more Levels associated  
 1743 in order to document information about the HierarchicalCodes.  
 1744

1745 If hasFormalLevels="false" the Hierarchy is "value based" comprising a hierarchy of  
 1746 codes with no formal Levels. If hasFormalLevels="true" then the hierarchy is "level  
 1747 based" where each Level is a formal Level in the HierarchicalCodeList, such as  
 1748 those present in statistical classifications. In a "level based" hierarchy each  
 1749 HierarchicalCode is linked to the Level in which it resides (which must be in the same  
 1750 Hierarchy as the HierarchicalCode). It is expected that all HierarchicalCodes at the  
 1751 same hierarchic level defined by the +parent/+child association will be linked to the same  
 1752 Level. Note that the +level association need only be specified if the HierarchicalCode is at a  
 1753 different hierarchical level ((implied by the HierarchicalCode parent/child association) than the  
 1754 actual Level in the level hierarchy (implied by the Level parent/child association).  
 1755

1756 [Note that organisations wishing to be compliant with accepted models for statistical  
 1757 classifications should ensure that the Id is the number associated with the Level, where  
 1758 Levels are numbered consecutively starting with level 1 at the highest Level].  
 1759

1760 The Level may have CodingFormat information defined (e.g. coding type at that level).  
 1761

1762 **8.3.2.2 Definitions**

1763

Class	Feature	Description
HierarchicalCodeList	Inherits from: <i>MaintainableArtefact</i>	An organised collection of codes that may participate in many parent/child relationships with other Codes in the scheme, as defined by one or more Hierarchy of the scheme.
	+hierarchy	Association to Hierarchies of Codes.
Hierarchy	Inherits from: <i>NameableArtefact</i>	A classification structure arranged in levels of detail from the broadest to the most detailed level.



Class	Feature	Description
	hasFormalLevels	<p>If “true” this indicates a hierarchy where the structure is arranged in levels of detail from the broadest to the most detailed level.</p> <p>If “false” this indicates a hierarchy structure where the items in the hierarchy have no formal level structure.</p>
	+codes	Association to the top-level Hierarchical Codes in the Hierarchy.
	+level	Association to the top Level in the Hierarchy.
Level	Inherits from <i>NameableArtefact</i>	<p>In a “level based” hierarchy this describes a group of Codes which are characterised by homogeneous coding, and where the parent of each Code in the group is at the same higher level of the Hierarchy.</p> <p>In a “value based” hierarchy this describes information about the HierarchicalCodes at the specified nesting level.</p>
	+codeFormat	Association to the Coding Format.
	+child	Association to a child Level of Level.
CodingFormat		Specifies format information for the codes at this level in the hierarchy such as whether the codes at the level are alphabetic, numeric or alphanumeric and the code length.
HierarchicalCode		A hierarchic structure of code references.
	validFrom	Date from which the construct is valid

Class	Feature	Description
	validTo	Date from which construct is superseded.
	+code	Association to the Code that is used at the specific point in the hierarchy.
	+child	Association to a child Code in the hierarchy.
	+level	Association to a Level where levels have been defined for the Hierarchy.
Code		The Code to be used at this point in the hierarchy.
	/items	Association to the Code list containing the Code.
CodeList		The Code list containing the Code.

1764

1765 **9 Structure Set and Mappings**

1766 **9.1 Scope**

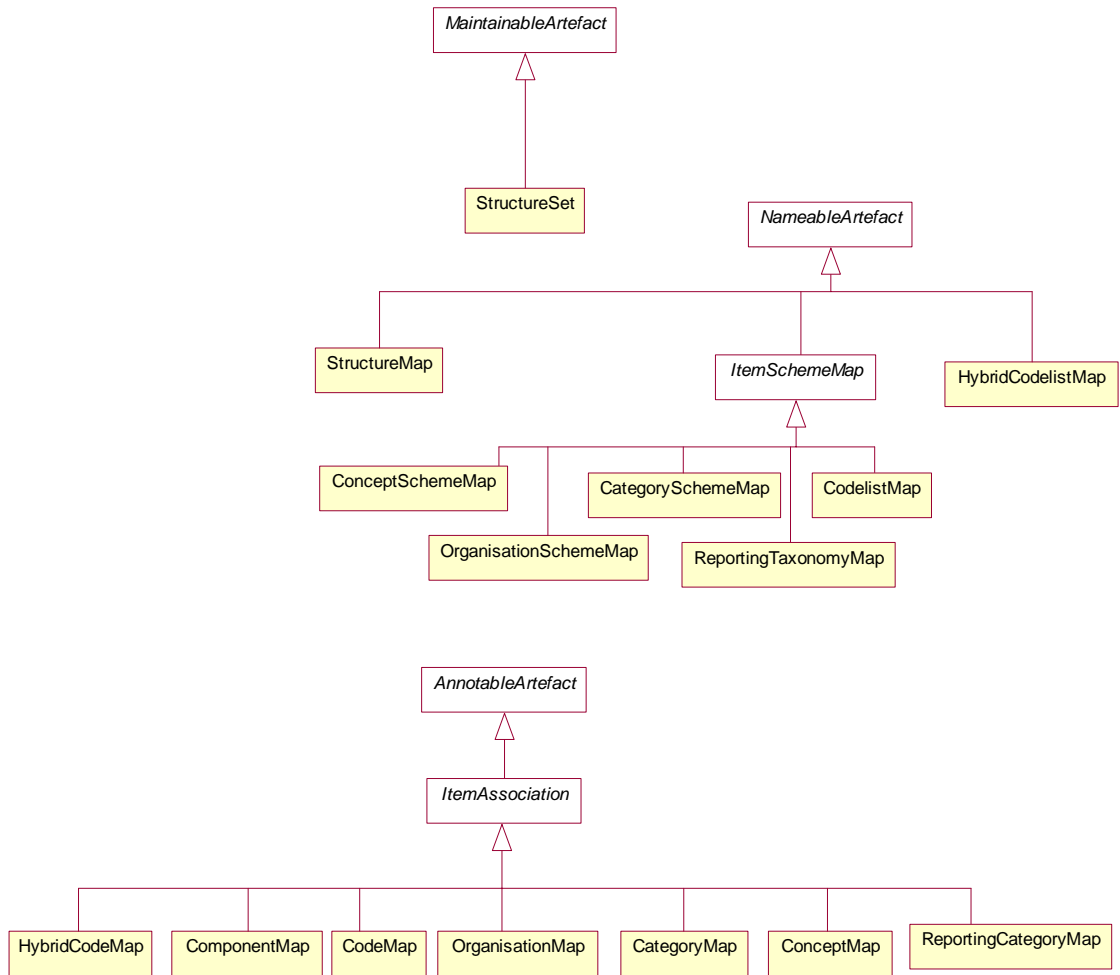
1767 A `StructureSet` allows components in one structure to be mapped to components in  
 1768 another structure of the same type. In this context the term “structure” is used loosely to  
 1769 include types of `ItemScheme`, types of `Structure`, and types of `StructureUsage`. The  
 1770 allowable structures that can be mapped, and the components that can be mapped within  
 1771 these structures are:  
 1772

Structure Type	Component type
Codelist	Code
Category Scheme	Category
Concept Scheme	Concept
Organisation Scheme	Organisation – this allows mapping any type of Organisation to any type of Organisation (e.g. a Data Provider to an Organisation Unit)
Hierarchical Codelist	Hierarchical Code to Code or vice-versa
Data Structure Definition	Dimension, Measure Dimension, Time Dimension. Data Attribute, Primary Measure
Metadata Structure Definition	Target Object, Metadata Attribute
Dataflow Definition	None
Metadataflow Definition	None

1773  
 1774 The `StructureSet` can contain one or more “maps” and can define related structures (via  
 1775 the association `+relatedStructure`) which group related `DataStructureDefinitions`,  
 1776 `MetadataStructureDefinitions`, `DataflowDefinitions`,  
 1777 `MetadataflowDefinitions`.

1778 **9.2 Structure Set**

1779 **9.2.1 Class Diagram – Inheritance**



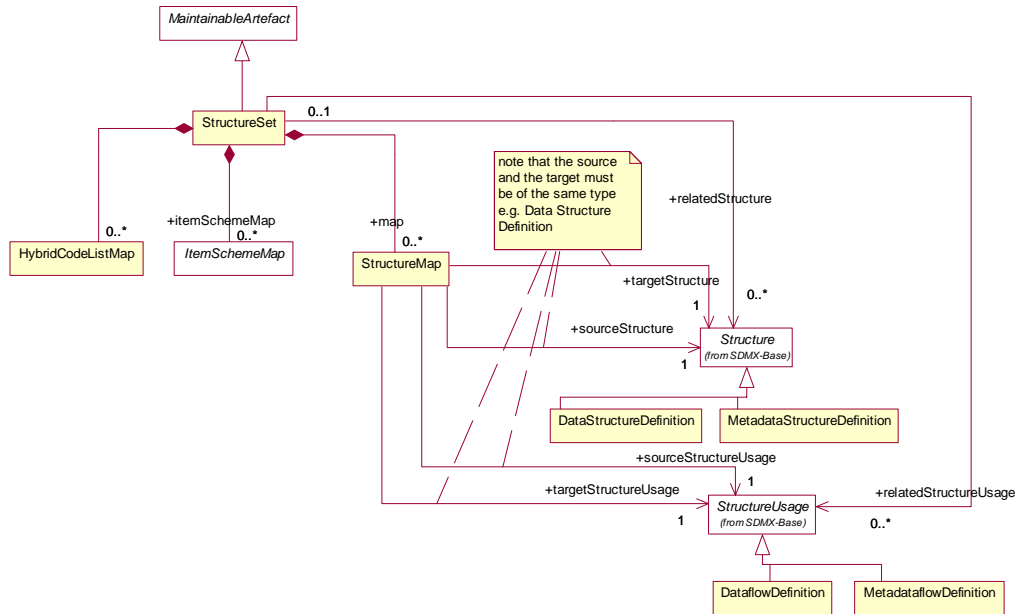
1780

1781

**Figure 32: Inheritance Class Diagram of the Structure Set**

1782 **9.2.2 Class Diagram – Relationship**

1783



1784

1785

**Figure 33: Relationship Class diagram of the Structure Set**

1786 **9.2.3 Explanation of the Diagram**

1787 **9.2.3.1 Narrative**

1788 The *StructureSet* is a *MaintainableArtefact*. It can contain:

1789

1790

1791

1792

1793

1794

1795

1796

1797

1798

1799

1800

1801

1802

1803

1. A set of references to concrete sub-classes of *Structure* and *StructureUsage* (*DataStructureDefinition*, *MetadataStructureDefinition*, *DataflowDefinition* or *MetadataflowDefinition*) to indicate that a relationship exists between them. For example there may be a group of *DataStructureDefinition* which, together, form the definition of a cube, each *DataStructureDefinition* defining a part of the cube.
2. A set of *StructureMaps* which define which components of one structure are equivalent to those in another in a *ComponentMap*.
3. A set of *ItemSchemeMaps* which define the mapping between two concrete classes of *ItemScheme*, and the mapping of the *Items* in these schemes, such as the mapping of Codes in two *Codelists*.
4. A set of *HybridCodelistMaps* which define the mapping between a *Codelist* and a *HierarchicalCodelist*.

1804

1805

1806

1807

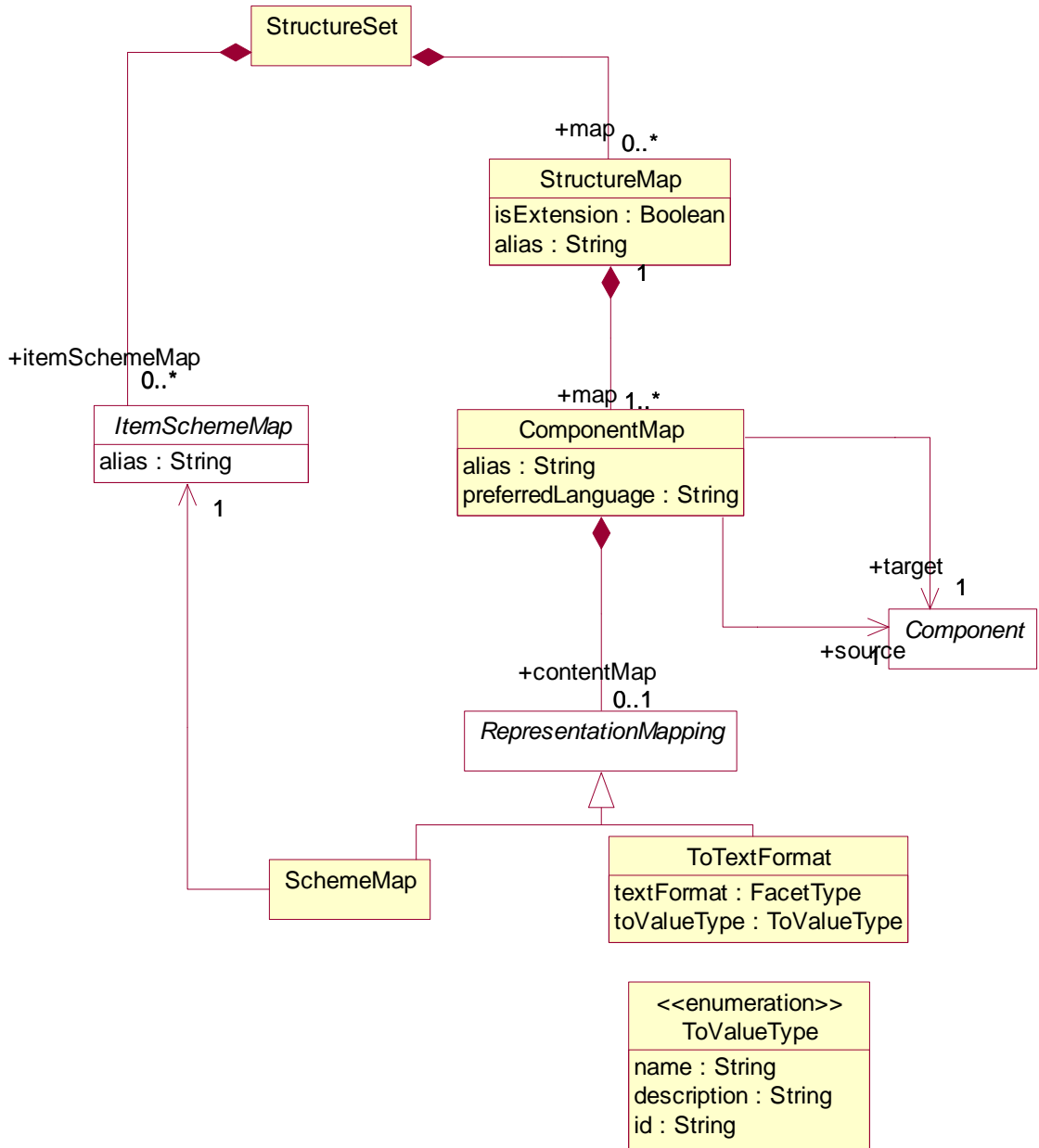
The *StructureMap* references two *Structures* or *StructureUsages*. In concrete terms these references will be to *DataStructureDefinitions*, *MetadataStructureDefinitions*, *DataflowDefinitions* or *MetadataflowDefinitions*.

1808 9.2.3.2 Definitions

Class	Feature	Description
StructureSet	Inherits from <i>MaintainableArtefact</i>	A maintainable collection of structural maps that link components together in a source/target relationship where there is a semantic equivalence between the source and the target components.
	+relatedStructure	Association to a set of Data Structure Definitions and Metadata Structure Definitions.
	+relatedStructureUsage	Association to a set of Dataflow Definition and Metadataflow Definition.
	+map	Association to Structure Map.
	+itemSchemeMap	Association to Item Scheme Map
StructureMap	Inherits from <i>NameableArtefact</i>	Links a source and target structure where there is a semantic equivalence between the source and the target structures.
	sourceStructure	Association to the source Structure.
	targetStructure	Association to the target Structure which must be of the same type as the source Structure.
	sourceStructureUsage	Association to the source Structure Usage.
	targetStructureUsage	Association to the target Structure Usage which must be of the same type as the source Structure Usage.

1809 **9.3 Structure Map**

1810 **9.3.1 Class Diagram**



1811  
1812

**Figure 34: Class diagram of the Structure Map**

1813 **9.3.2 Explanation of the Diagram**

1814 **9.3.2.1 Narrative**

1815 The StructureMap contains a set of ComponentMaps, each one indicating equivalence  
 1816 between Components of the referenced Structure. ComponentMap has a  
 1817 RepresentationMapping which can be one of the concrete classes of ItemSchemeMap

1818 (e.g. for a Dimension this would be a CodelistMap) or ToTextFormat which takes values:  
 1819 id, name, description. This instructs mapping tools to use the id, name or description of a  
 1820 coded component to determine equivalence with an uncoded component's value.

1821  
 1822 An example of a ComponentMap is linking the source *Component* that is a Dimension in the  
 1823 source *DataStructureDefinition* (identified in the *StructureMap*) to the equivalent  
 1824 target *Component* that is a Dimension in the target *DataStructureDefinition*).  
 1825

1826 **9.3.2.2 Definitions**

Class	Feature	Description
StructureMap	Inherits from <i>NameableArtefact</i>	Links a source and target structure where there is a semantic equivalence between the source and the target structures.
	alias	An alternate identification of the map, that allows the relation of multiple maps to be expressed by the sharing of this value.
	+map	Association to the Component Map.
ComponentMap	Inherits from <i>AnnotableArtefact</i>	Links a source and target Component where there is a semantic equivalence between the source and the target Components.
	alias	An alternate identification of the map, that allows the relation of multiple maps to be expressed by the sharing of this value.
	preferredLanguage	Specifies the language to use for the content of the To Text Format option of RepresentationMap
	+source	Association to the source Component.
	+target	Association to the target Component.
	+contentMap	Association to the constructs that map the content of the Components – this will be either one of sub classes of Item Scheme or a mapping to text.



Class	Feature	Description
<i>Representation Mapping</i>	AbstractClass  Sub classes:  SchemeMap ToTextFormat	Defines the mapping of the content of the source Component to the content of the target Component.
SchemeMap	Inherits from  <i>RepresentationMapping</i>	Associates an Item Scheme Map
ToTextFormat	Inherits from  <i>RepresentationMapping</i>	Defines the text format
	textFormat	Text format type.
	toValueType	Identifies the construct to be taken from the Item of the source Component when mapping the content of the source Component to the content of the target Component.
ToValueType		Enumeration of the construct in the Item.

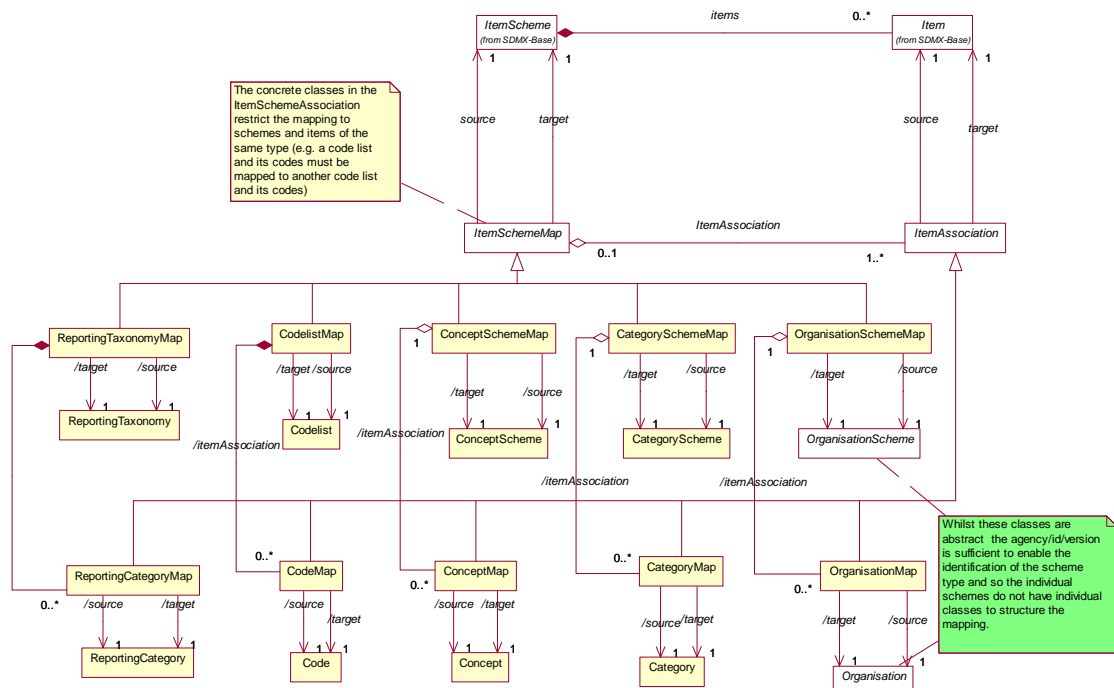
1827 **9.4 Item Scheme Map**

1828 **9.4.1 Context**

1829 The *ItemSchemeMap* is used to associate the *Items* in two different *ItemSchemes*. This is a  
 1830 generic mechanism that can be used to map *Items*. Specific models exist for mapping  
 1831 schemes where there is a semantic equivalence between *Items* in the *ItemScheme*. The  
 1832 model supports the mapping of any two *ItemSchemes* of the same type. These are:

- 1833
- 1834 • *ConceptScheme*
- 1835 • *CategoryScheme*
- 1836 • *OrganisationScheme*
- 1837 • *Codelist*
- 1838 • *ReportingTaxonomy*

1839 **9.4.2 Class Diagram**



1840  
1841

**Figure 35: Class diagram of the Item Scheme Map**

1842 **9.4.3 Explanation of the Diagram**

1843 **9.4.3.1 Narrative**

1844 Both the *ItemSchemeMap* and the *ItemAssociation* inherit from *NameableArtefact*.

1845

1846 Each of *ConceptSchemeMap*, *CategorySchemeMap*, *CodelistMap* and  
1847 *OrganisationSchemeMap*, *ReportingTaxonomyMap* provides a mechanism for  
1848 specifying semantic equivalence between the items (*Concept*, *Category*, *Code*,  
1849 *Organisation*, *ReportingCategory*) in the scheme. Note that any type of  
1850 *OrganisationScheme* and *Organisation* can be mapped (e.g. an Agency in an  
1851 AgencyScheme can be mapped to an OrganisationUnit in an  
1852 OrganisationUnitScheme).

1853

1854 Each scheme map identifies a +source and +target scheme whose content is to be  
1855 mapped. Note that many schemes can be joined together via a set of pair-wise mappings. The  
1856 *ConceptMap*, *CategoryMap*, *CodelistMap*, *OrganisationMap*, and  
1857 *ReportingTaxonomyMap* denotes which Concepts, Categories, Codes, Organisations,  
1858 and ReportingCategories are semantically equivalent and a shared alias can be specified  
1859 to refer to a set of mapped concepts to facilitate querying.

1860 **9.4.3.2 Definitions**

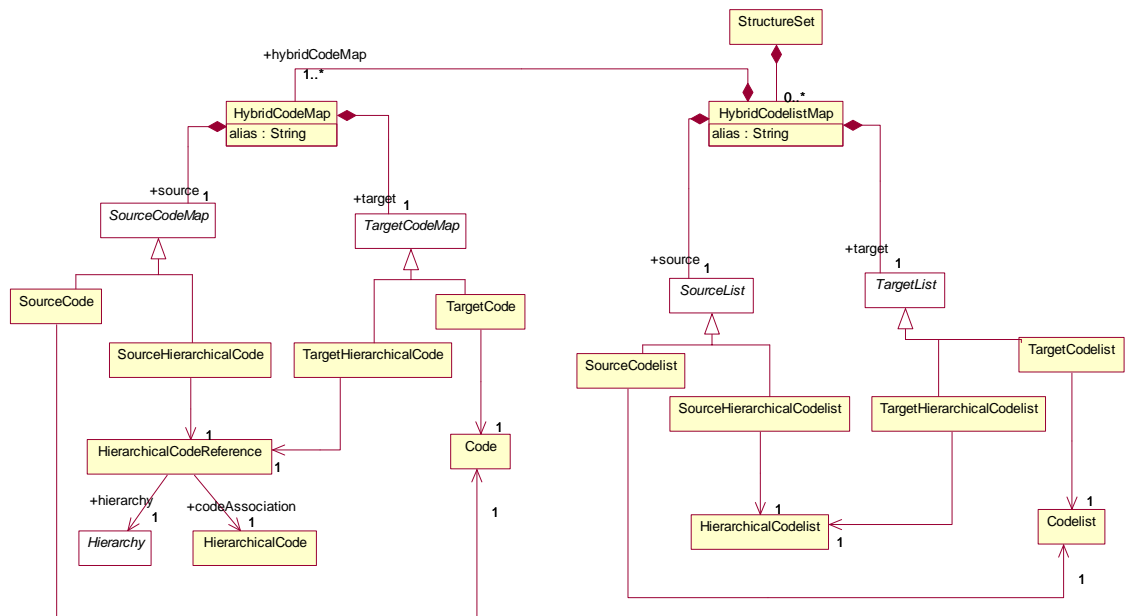
Class	Feature	Description
<i>ItemSchemeMap</i>	Inherits from	Associates two Item Schemes

Class	Feature	Description
	<i>NameableArtefact</i>  <b>Sub Classes</b>  ConceptSchemeMap CategorySchemeMap CodelistMap OrganisationSchemeMap ReportingTaxonomySchemeMap	
	source	Association to the source Item Scheme.
	target	Association to the target Item Scheme.
	ItemAssociation	Association to the Item Association.
<i>ItemAssociation</i>	<b>Inherits from</b> <i>AnnotableArtefact</i>  <b>Sub Classes</b>  ConceptMap CategoryMap CodeMap OrganisationMap ReportingCategoryMap	
	source	Association to the source Item.
	target	Association to the target Item.
ConceptSchemeMap	<b>Inherits from</b> <i>ItemSchemeMap</i>	Associates a source and target Concept Scheme
	/source	Association to the source Concept Scheme.
	/target	Association to the target Concept Scheme.
ConceptMap	<b>Inherits from</b> <i>ItemAssociation</i>	Associates a source and target Concept.
	/source	Association to the source Concept.
	/target	Association to the target Concept.
CodelistMap	<b>Inherits from</b> <i>ItemSchemeMap</i>	Associates a source and target Code list.
	/source	Association to the source Code list.
	/target	Association to the target Code list.

<b>Class</b>	<b>Feature</b>	<b>Description</b>
CodeMap	Inherits from <i>ItemAssociation</i>	Associates a source and target Code.
	/source	Association to the source Code.
	/target	Association to the target Code.
CategorySchemeMap	Inherits from <i>ItemSchemeMap</i>	Associates a source and target Category Scheme.
	/source	Association to the source Category Scheme.
	/target	Association to the target Category Scheme.
CategoryMap	Inherits from <i>ItemAssociation</i>	Associates a source and target Category.
	/source	Association to the source Category.
	/target	Association to the target Category.
OrganisationSchemeMap	Inherits from <i>ItemSchemeMap</i>	Associates a source and target Organisation Scheme.
	/source	Association to the source Organisation Scheme.
	/target	Association to the target Organisation Scheme.
OrganisationMap	Inherits from <i>ItemAssociation</i>	Associates a source and target Organisation.
	/source	Association to the source Organisation.
	/target	Association to the target Organisation.
ReportingTaxonomyMap	Inherits from <i>ItemSchemeMap</i>	Associates a source and target Reporting Taxonomy.
	/source	Association to the source Reporting Taxonomy.
	/target	Association to the target Reporting Taxonomy.
ReportingCategoryMap	Inherits from <i>ItemAssociation</i>	Associates a source and target Reporting Category.
	/source	Association to the source Reporting Category.
	/target	Association to the target Reporting Category.

1861 **9.5 Hybrid Codelist Map**

1862 **9.5.1 Class Diagram**



1863

1864

**Figure 36: Class diagram of the Hybrid Codelist Map**

1865 **9.5.2 Explanation of the Diagram**

1866 **9.5.2.1 Narrative**

1867 The HybridCodelistMap maps the content of a Codelist and a  
 1868 HierarchicalCodelist. It contains a mapping of the codes in the two schemes  
 1869 (HybridCodeMap). The HybridCodeMap maps either a Code or HierarchicalCode to a  
 1870 Code or HierarchicalCode. The HierarchicalCode is identified by a combination of the  
 1871 Hierarchy and the HierarchicalCode.

1872

1873 **9.5.2.2 Definitions**

Class	Feature	Description
HybridCodelist Map	Inherits from <i>NameableArtefact</i>	Associates a Codelist and a Hierarchical Codelist.
	alias	An alternate identification of the map, that allows the relation of multiple maps to be expressed by the sharing of this value.
	+source	Association to the source List.
	+target	Association to the target List.

Class	Feature	Description
	+hybridCodeMap	Association to the set of Hybrid Code Maps in the Hybrid Codelist Map.
<i>SourceList</i>	Abstract Class  Sub classes SourceCodelist SourceHierarchical Codelist	
<i>TargetList</i>	Abstract Class  Sub classes TargetCodelist TargetHierarchical Codelist	
SourceCodelist		Identifies the Codelist where this is the source of the map.
TargetCodelist		Identifies the Codelist where this is the target of the map.
SourceHierarchical Codelist		Identifies the Hierarchical Codelist where this is the source of the map.
TargetHierarchical Codelist		Identifies the Hierarchical Codelist where this is the target of the map.
HybridCodeMap	Inherits from <i>AnnotableArtefact</i>	Associates the source and target codes in Hybrid Codelist Map.
	+source	Associates the Source Code Map.
	+target	Associates the Target Code Map.
<i>SourceCodeMap</i>	Abstract Class  Sub classes SourceCode SourceHierarchical Code	
<i>TargetCodeMap</i>	Abstract Class  Sub classes TargetCode TargetHierarchical Code	
SourceCode		Identifies the Code where this is the source of the map.

Class	Feature	Description
TargetCode		Identifies the Code where this is the target of the map.
SourceHierarchical Code		Identifies the Hierarchical Code where this is the source of the map
TargetHierarchical Code		Identifies the Hierarchical Code where this is the target of the map.
HierarchicalCode Reference		References both the Hierarchy and the Hierarchical Code in a Hierarchical Codelist.
	+hierarchy +codeAssociation	Associates the Hierarchical Code in the Hierarchy of the Hierarchical Codelist.

1874

1875 **10 Constraints**

1876 **10.1 Scope**

1877 The scope of this section is to describe the support in the metamodel for specifying both the  
 1878 access to and the content of a data source. The information may be stored in a resource such  
 1879 as a registry for use by applications wishing to locate data and metadata which is available via  
 1880 the Internet. The Constraint is also used to specify a sub set of a Codelist which may used as  
 1881 a partial code list which is relevant in the context of the artefact to which the Constraint is  
 1882 attached e.g. Data Structure Definition, Dataflow, Provision Agreement.  
 1883

1884 Note that in this metamodel the term data source refers to both data and metadata sources,  
 1885 and data provider refers to both data and metadata providers.  
 1886

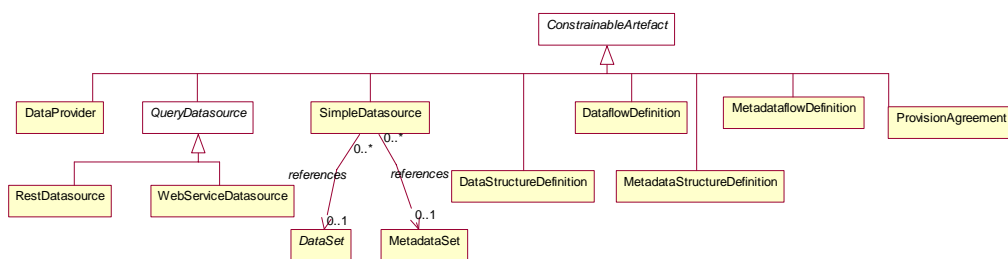
1887 A data source may be a simple file of data or metadata (in SDMX-ML format), or a database or  
 1888 metadata repository. A data source may contain data for many data or metadataflows (called  
 1889 DataflowDefinition, and MetadataflowDefinition in the model), and the  
 1890 mechanisms described in this section allow an organisation to specify precisely the scope of  
 1891 the content of the data source where this data source is registered (SimpleDataSource,  
 1892 QueryDataSource).

1894 The DataflowDefinition and MetadataflowDefinition, themselves may be  
 1895 specified as containing only a sub set of all the possible keys that could be derived from a  
 1896 DataStructureDefinition or MetadataStructureDefinition.  
 1897

1898 These specifications are called *Constraint* in this model.

1899 **10.2 Inheritance**

1900 **10.2.1 Class Diagram of Constraining Artefacts - Inheritance**



1901

1902 **Figure 37: Inheritance class diagram of constrainable and provisioning artefacts**

1903 **10.2.2 Explanation of the Diagram**

1904 **10.2.2.1 Narrative**

1905 Any artefact that is derived from *ConstrainingArtefact* can have constraints defined.  
 1906 The artefacts that can have constraint metadata attached are:

1907

- 1908 • DataflowDefinition

- 1909 • ProvisionAgreement

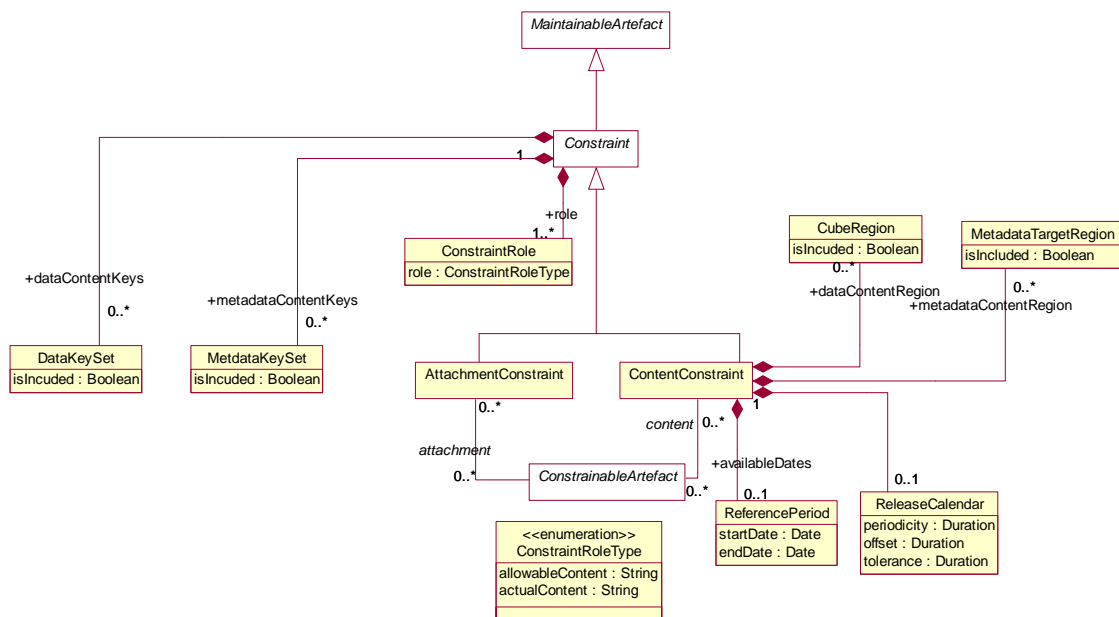


- 1910 • DataProvider – this is restricted to release calendar
- 1911 • MetadataflowDefinition
- 1912 • DataStructureDefinition
- 1913 • MetadataStructureDefinition
- 1914 • DataSet
- 1915 • SimpleDataSource – this is a registered data source where the
- 1916 registration references the actual DataSet or MetadataSet
- 1917 • QueryDataSource

1918 Note that, because the Constraint can specify a sub set of the component values implied  
 1919 by a specific *Structure* (such a specific DataStructureDefinition or specific  
 1920 MetadataStructureDefinition), the *ConstrainableArtefacts* must be associated  
 1921 with a specific *Structure*. Therefore, whilst the Constraint itself may not be linked directly  
 1922 to a DataStructureDefinition or MetadataStructureDefinition, the artefact that  
 1923 it is constraining will be linked to a DataStructureDefinition or  
 1924 MetadataStructureDefinition. As a Data Provider does not link to any one specific  
 1925 DSD or MSD the type of information that can be contained in a Constraint linked to a  
 1926 DataProvider is restricted to Release Calendar.

## 1927 10.3 Constraints

### 1928 10.3.1 Relationship Class Diagram – high level view



1929

1930

**Figure 38: Relationship class diagram showing constraint metadata**

1931 **10.3.2 Explanation of the Diagram**

1932 **10.3.2.1 Narrative**

1933 The constraint mechanism allows specific constraints to be attached to a  
1934 *ConstrainableArtefact*. With the exception of *ReferencePeriod*, and  
1935 *ReleaseCalendar* these constraints specify a sub set of the total set of values or keys that  
1936 may be present in any of the *ConstrainableArtefacts*.

1937

1938 For instance a *DataStructureDefinition* specifies, for each *Dimension*, the list of  
1939 allowable code values. However, a specific *DataflowDefinition* that uses the  
1940 *DataStructureDefinition* may contain only a sub set of the possible range of keys that  
1941 is theoretically possible from the *DataStructureDefinition* definition (the total range of  
1942 possibilities is sometimes called the Cartesian product of the dimension values). In addition to  
1943 this, a *DataProvider* that is capable of supplying data according to the  
1944 *DataflowDefinition* has a *ProvisionAgreement*, and the *DataProvider* may also  
1945 wish to supply constraint information which may further constrain the range of possibilities in  
1946 order to describe the data that the provider can supply. It may also be useful to describe the  
1947 content of a *datasource* in terms of the *KeySets* or *CubeRegions* contained within it.

1948

1949 A *ConstrainableArtefact* can have two types of *Constraint*:

1950

1951 1. *ContentConstraint* – is used solely as a mechanism to specify either the available  
1952 set of keys (*DataKeySet*, *MetadataKeySet*) or set of component values  
1953 (*CubeRegion*, *MetadataTargetRegion*) in a *DataSource* such as a *DataSet* or a  
1954 database (*QueryDataSource*), or the allowable keys that can be constructed from a  
1955 *DataStructureDefinition*. Multiple such constraints may be present for a  
1956 *ConstrainableArtefact*. For instance, there may be a *ContentConstraint*  
1957 that specifies the values allowed for the *ConstrainableArtefact* (*role* is  
1958 *allowableContent*) which can be used for validation or for constructing a partial  
1959 code list, whilst another constraint can specify the actual content of a data or  
1960 metadata source (*role* is *actualContent*).

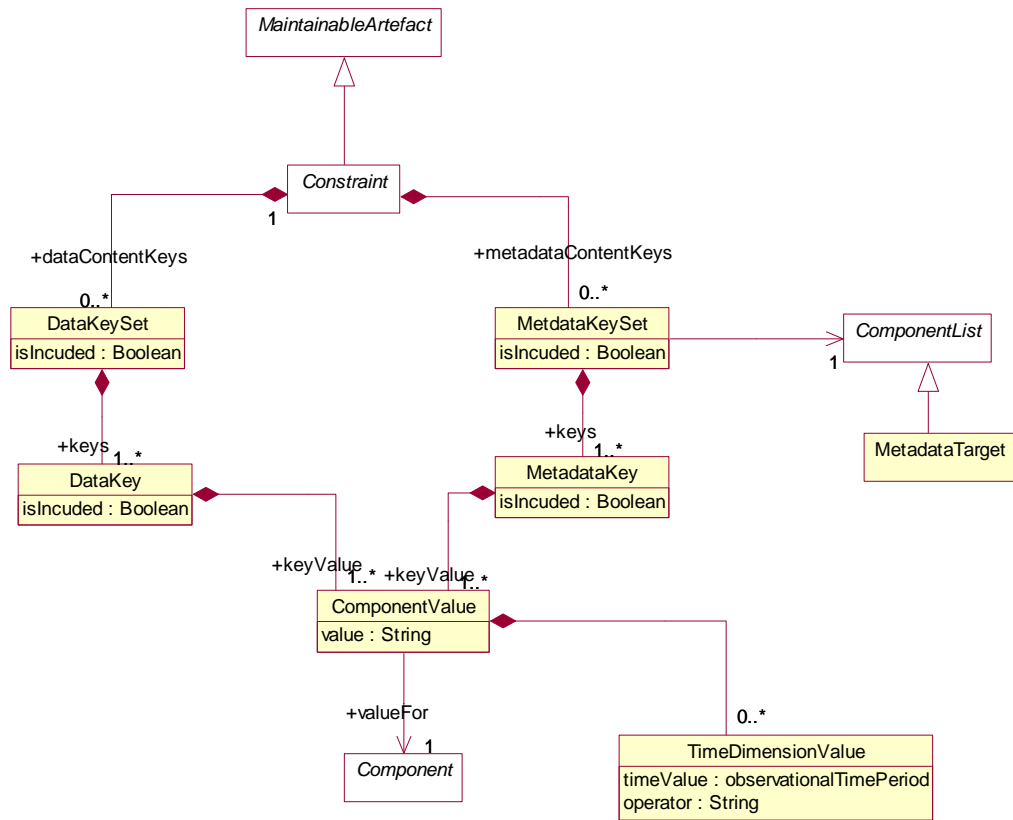
1961 2. *AttachmentConstraint* – is used as a mechanism to define slices of the full set of  
1962 data and to which metadata can be attached in a *Data Set* or *MetadataSet*. These  
1963 slices can be defined either as a set of keys (*KeySet*) or a set of component values  
1964 (*CubeRegion*). There can be many *AttachmentConstraints* specified for a  
1965 specific *AttachableArtefact*.

1966

1967 In addition to (*DataKeySet*, *MetadataKeySet*, *CubeRegion*,  
1968 *MetadataTargetRegion*, a *Constraint* can have a *ReferencePeriod* defining one of  
1969 more date ranges (*ValidityPeriod*) specifying the time period for which data or metadata  
1970 are available in the *ConstrainableArtefact* and a *ReleaseCalendar* specifying when  
1971 data are released for publication or reporting.

1972

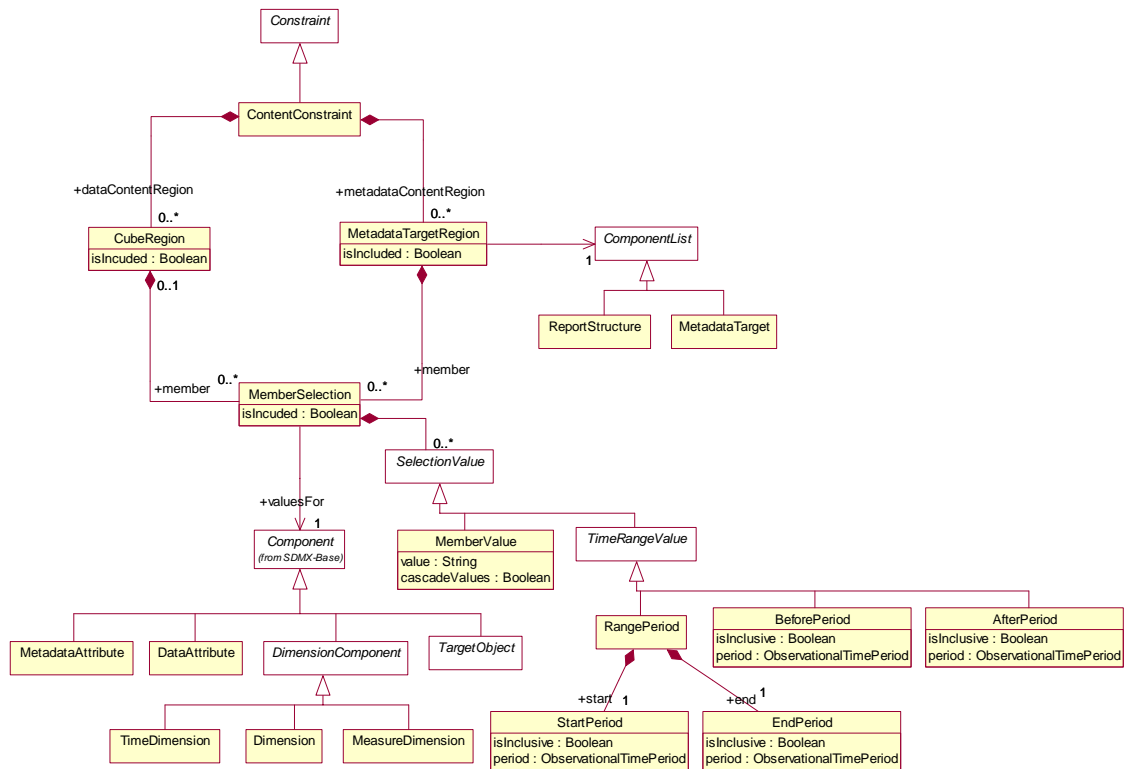
1973 **10.3.3 Relationship Class Diagram – Detail**



1974

1975

**Figure 39: Constraints - Key Set Constraints**



1976  
1977

**Figure 40: Constraints - Cube Region and Metadata Target Region Constraints**

1978 **10.3.3.1 Explanation of the Diagram**

1979 A *Constraint* is a *MaintainableArtefact*.

1980

1981 A *Constraint* has a choice of two ways of specifying value sub sets:

1982

1983 1. As a set of keys that can be present in the *DataSet* (*DataKeySet*) or *MetadataSet*  
1984 (*MetadataKeySet*). Each *DataKey* or *MetadataKey* specifies a number of  
1985 *ComponentValues* each of which reference a *Component* (e.g. *Dimension*,  
1986 *TargetObject*). Each *ComponentValue* is a value that may be present for a  
1987 *Component* of a structure when contained in a *DataSet* or *MetadataSet*. The  
1988 *MetadataKeySet* must also identify the *MetadataTarget* as there can be many of  
1989 each of these in a *MetadataStructureDefinition*. For the *DataKeySet* the  
1990 equivalent identification is not necessary as there is only one *DimensionDescriptor*  
1991 and one *AttributeDescriptor*.

1992 2. As a set of *CubeRegions* or *MetadataTaregetRegions* each of which defines a  
1993 “slice” of the total structure (*MemberSelection*) in terms of one or more  
1994 *MemberValues* that may be present for a *Component* of a structure when contained  
1995 in a *DataSet* or *MetadataSet*.

1996 The difference between (1) and (2) above is that in (1) a complete key is defined whereas in  
1997 (2) above the “slice” defines a list of possible values for each of the *Components* but does  
1998 not specify specific key combinations. In addition, in (1) the association between *Component*

1999 and DataKeyValue or MetadataKeyValue is constrained to the components that comprise  
 2000 the key or identifier, whereas in (2) it can contain other component types (such as attributes).  
 2001 The value in ComponentValue.value and MemberValue.value must be consistent with  
 2002 the *Representation* declared for the *Component* in the DataStructureDefinition or  
 2003 MetadataStructureDefinition. Note that in all cases the “operator” on the value is  
 2004 deemed to be “equals”. Furthermore, it is possible in a MemberValue to specify that child  
 2005 values (e.g. child codes) are included in the constraint by means of the cascadeValues  
 2006 attribute.

2007  
 2008 It is possible to define for the DataKeySet, DataKey, MetadataKeySet, MetadataKey,  
 2009 CubeRegion, MetadataTargetRegion, and MemberSelection whether the set is  
 2010 included (isIncluded = “true”) or excluded (isIncluded = “false”) from the constraint  
 2011 definition. This attribute is useful if, for example, only a small sub-set of the possible values  
 2012 are not included in the set, then this smaller sub-set can be defined and excluded from the  
 2013 constraint. Note that if the child construct is “included: and the parent construct is “excluded”  
 2014 then the child construct is included in the list of constructs that are “excluded”.

2015 **10.3.3.2 Definitions**

Class	Feature	Description
<i>Constrainable Artefact</i>	Abstract Class Sub classes are:  DataflowDefinition Metadataflow Definition ProvisionAgreement DataProvider QueryDatasource SimpleDatasource DataStructure Definition MetadataStructure Definition	An artefact that can have Constraints specified.
	content	Associates the metadata that constrains the content to be found in a data or metadata source linked to the Constrainable Artefact.
	attachment	Associates the metadata that constrains the valid content of a Constrainable Artefact to which metadata may be attached.

Class	Feature	Description
<i>Constraint</i>	Inherits from <i>MaintainableArtefact</i>  Abstract class. Sub classes are:  <i>AttachmentConstraint</i> <i>ContentConstraint</i>	Specifies a sub set of the definition of the allowable or actual content of a data or metadata source that can be derived from the Structure that defines code lists and other valid content.
	+availableDates	Association to the time period that identifies the time range for which data or metadata are available in the data source.
	+dataContentKeys	Association to a sub set of Data Key Sets (i.e. value combinations) that can be derived from the definition of the structure to which the Constraining Artefact is linked.
	+metadataContentKeys	Association to a sub set of Metadata Key Sets (i.e. value combinations) that can be derived from the definition of the Structure to which the Constraining Artefact is linked.
	+dataContentRegion	Association to a sub set of component values that can be derived from the Data Structure Definition to which the Constraining Artefact is linked.
	+metadataContentRegion	Association to a sub set of component values that can be derived from the Metadata Structure Definition to which the Constraining Artefact is linked.

Class	Feature	Description
ContentConstraint	Inherits from <i>Constraint</i>	Defines a Constraint in terms of the content that can be found in data or metadata sources linked to the Constraining Artefact to which this constraint is associated.
	+role	Association to the role that the Constraint plays
ConstraintRole		Specifies the way the type of content of a Constraint in terms of its purpose.
	allowableContent	The Constraint contains a specification of the valid sub set of the Component values or keys.
	actualContent	The Constraint contains a specification of the actual content of a data or metadata source in terms of the Component values or keys in the source.
Attachment Constraint	Inherits from <i>Constraint</i>	Defines a Constraint in terms of the combination of component values that may be found in a data source, and to which a Constraining Artefact may be associated in a structure definition.
DataKeySet		A set of data keys.
	isIncluded	Indicates whether the Data Key Set is included in the constraint definition or excluded from the constraint definition.
	+keys	Association to the Data Keys in the set.
MetadataKeySet		A set of metadata keys.
	isIncluded	Indicates whether the Metadata Key Set is included in the constraint definition or excluded from the constraint definition.
	+keys	Association to the Metadata Keys in the set.

Class	Feature	Description
DataKey		The values of a key in a data set.
	isIncluded	Indicates whether the Data Key is included in the constraint definition or excluded from the constraint definition.
	+keyValue	Associates the Component Values that comprise the key.
MetadataKey		The values of a key in a metadata set.
	isIncluded	Indicates whether the Metadata Key is included in the constraint definition or excluded from the constraint definition.
	+keyValue	Associates the Component Values that comprise the key.
ComponentValue		The identification of and value of a Component of the key (e.g. Dimension)
	value	The value of Component
	+valueFor	Association to the Component (e.g. Dimension) in the Structure to which the Constraining Artefact is linked.
TimeDimensionValue		The value of the Time Dimension component.
	timeValue	The value of the time period.



Class	Feature	Description
	operator	<p>Indicates whether the specified value represents and exact time or time period, or whether the value should be handled as a range.</p> <p>A value of greaterThan or greaterThanOrEqual indicates that the value is the beginning of a range (exclusive or inclusive, respectively).</p> <p>A value of lessThan or lessThanOrEqual indicates that the value is the end of a range (exclusive or inclusive, respectively).</p> <p>In the absence of the opposite bound being specified for the range, this bound is to be treated as infinite (e.g. any time period after the beginning of the provided time period for greaterThanOrEqual)</p>
CubeRegion		A set of Components and their values that defines a sub set or “slice” of the total range of possible content of a data structure to which the Constraining Artefact is linked.
	isIncluded	Indicates whether the Cube Region is included in the constraint definition or excluded from the constraint definition.
	+member	Associates the set of Components that define the sub set of values.

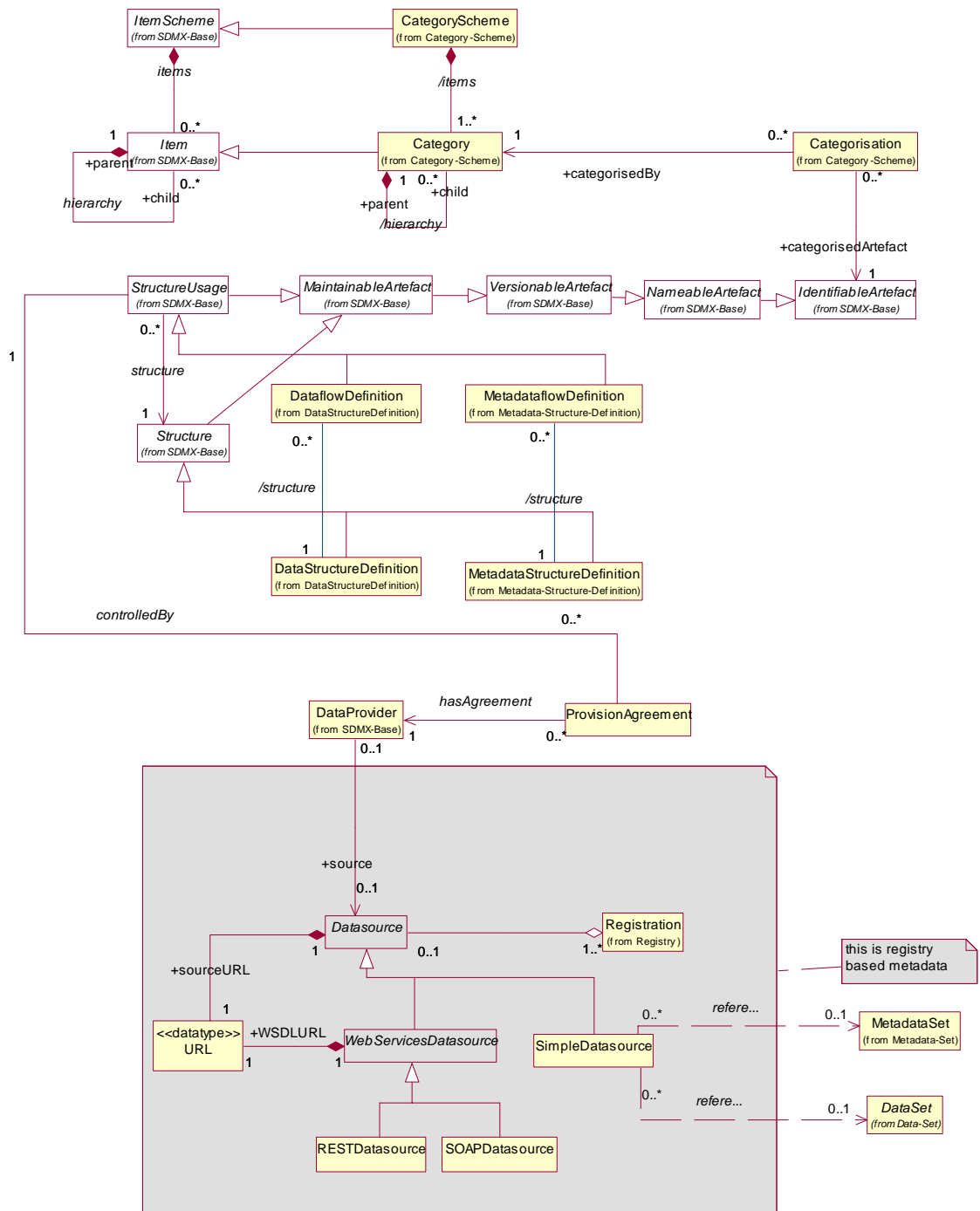
Class	Feature	Description
MetadataTargetRegion		A set of Components and their values that defines a sub set or “slice” of the total range of possible content of a metadata structure to which the Constraining Artefact is linked.
	isIncluded	Indicates whether the Metadata Target Region is included in the constraint definition or excluded from the constraint definition.
	+member	Associates the set of Components that define the sub set of values.
MemberSelection		A set of permissible values for one component of the axis.
	isIncluded	Indicates whether the Member Selection is included in the constraint definition or excluded from the constraint definition.
	+valuesFor	Association to the Component in the Structure to which the Constraining Artefact is linked, which defines the valid Representation for the Member Values.
SelectionValue	Abstract class. Sub classes are: MemberValue TimeRangeValue	A collection of values for the Member Selections that, combined with other Member Selections, comprise the value content of the Cube Region.
MemberValue	Inherits from SelectionValue	A single value of the set of values for the Member Selection.
	value	A value of the member.

Class	Feature	Description
	cascadeValues	Indicates that the child nodes of the member are included in the Member Selection (e.g. child codes)
<i>TimeRangeValue</i>	Inherits from SelectionValue  Abstract Class  Concrete Classes  BeforePeriod AfterPeriod RangePeriod	A time value or values that specifies the date or dates for which the constrained selection is valid.
BeforePeriod	Inherits from  <i>TimeRangeValue</i>	The period before which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
AfterPeriod	Inherits from  <i>TimeRangeValue</i>	The period after which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
RangePeriod		The start and end periods in a date range.
	+start	Association to the Start Period.
	+end	Association to the End Period.
StartPeriod	Inherits from  <i>TimeRangeValue</i>	The period from which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
EndPeriod	Inherits from  <i>TimeRangeValue</i>	The period to which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.

Class	Feature	Description
ReferencePeriod		A set of dates that constrain the content that may be found in a data or metadata set.
	startDate	The start date of the period.
	endDate	The end date of the period.
ReleaseCalendar		The schedule of publication or reporting of the data or metadata
	periodicity	The time period between the releases of the data or metadata
	offset	Interval between January 1 <sup>st</sup> and the first release of the data
	tolerance	Period after which the data or metadata may be deemed late.

2016 **11 Data Provisioning**

2017 **11.1 Class Diagram**



2018

2019

**Figure 41: Relationship and inheritance class diagram of data provisioning**

2020 **11.2 Explanation of the Diagram**

2021 **11.2.1 Narrative**

2022 This sub model links many artefacts in the SDMX-IM and is pivotal to an SDMX metadata  
2023 registry, as all of the artefacts in this sub model must be accessible to an application that is  
2024 responsible for data and metadata registration or for an application that requires access to the  
2025 data or metadata.

2026  
2027 Whilst a registry contains all of the metadata depicted on the diagram above, the classes in  
2028 the grey shaded area are specific to a registry based scenario where data sources (either  
2029 physical data and metadata sets or databases and metadata repositories) are registered.  
2030 More details on how these classes are used in a registry scenario can be found in the SDMX  
2031 Registry Interface document. (Section 5 of the SDMX Standards).

2032  
2033 A `ProvisionAgreement` links the artefact that defines how data and metadata are  
2034 structured and classified (`StructureUsage`) to the `DataProvider`, and, by means of a data  
2035 or metadata registration, it references the `Datasource` (this can be data or metadata),  
2036 whether this be an SDMX conformant file on a website (`SimpleDatasource`) or a database  
2037 service capable of supporting an SDMX query and responding with an SDMX conformant  
2038 document (`QueryDatasource`).

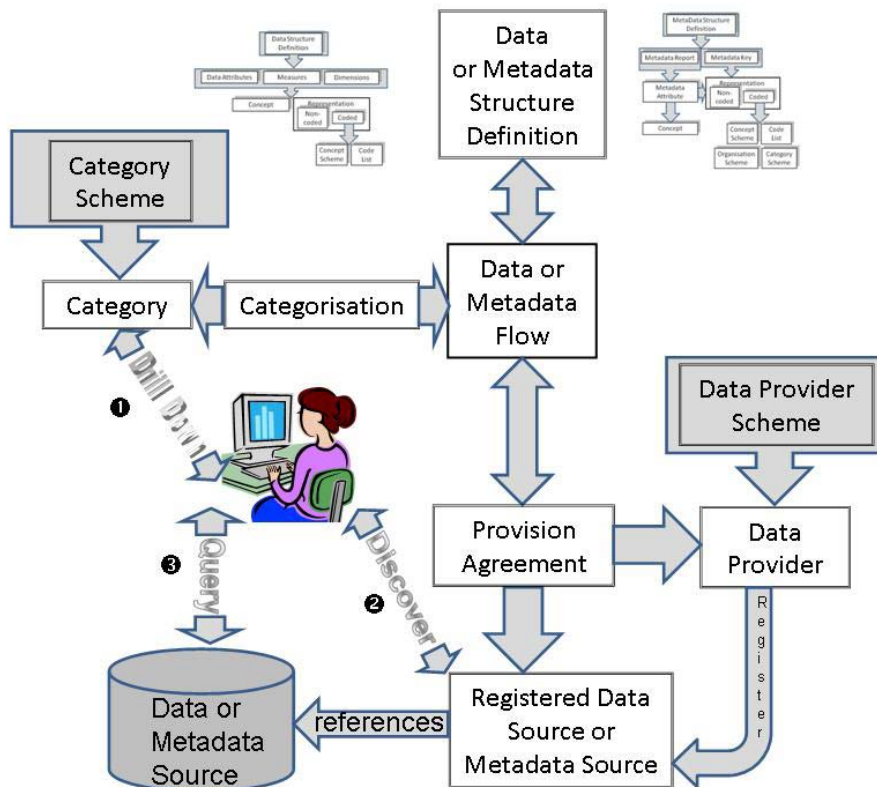
2039  
2040 The `StructureUsage`, which has concrete classes of `DataflowDefinition` and  
2041 `MetadataflowDefinition` identifies the corresponding `DataStructureDefinition` or  
2042 `MetadataStructureDefinition`, and, via `Categorisation`, can link to one or more  
2043 `Category` in a `CategoryScheme` such as a subject matter domain scheme, by which the  
2044 `StructureUsage` can be classified. This can assist in drilling down from subject matter  
2045 domains to find the data or metadata that may be relevant.

2046  
2047 The `SimpleDatasource` links to the actual `DataSet` or `MetadataSet` on a website (this is  
2048 shown on the diagram as a dependency called “references”). The `sourceURL` is obtained  
2049 during the registration process of the `DataSet` or the `MetadataSet`. Additional information  
2050 about the content of the `SimpleDatasource` is stored in the registry in terms of a  
2051 `ContentConstraint` (see 10.3) for the `Registration`.

2052  
2053 The `QueryDatasource` is an abstract class that represents a data source which can  
2054 understand an SDMX-ML query (`SOAPDatasource`) or RESTful query (`RESTDatasource`)  
2055 and respond appropriately. Each of these different `Datasources` inherit the `dataURL` from  
2056 `Datasource`, and the `QueryDatasource` has an additional URL to locate a WSDL or WADL  
2057 document to describe how to access it. All other supported protocols are assumed to use the  
2058 `SimpleDatasource` URL.

2059  
2060 The diagram below shows in schematic way the essential navigation through the SDMX  
2061 structural artefacts that eventually link to a data or metadata registration.

2062



2063  
2064 **Figure 42: Schematic of the linking of structural metadata to data and metadata registration**

2065 **11.2.2 Definitions**

2066

Class	Feature	Description
<i>StructureUsage</i>	Abstract class: Sub classes are:  DataflowDefinition MetadataflowDefinition	This is described in the Base.
	controlledBy	Association to the Provision Agreements that comprise the metadata related to the provision of data.
DataProvider		See Organisation Scheme.
	hasAgreement	Association to the Provision Agreements for which the provider supplies data or metadata.

Class	Feature	Description
	+source	Association to a data or metadata source which can process a data or metadata query.
ProvisionAgreement		Links the Data Provider to the relevant Structure Usage (e.g. Dataflow Definition or Metadataflow Definition) for which the provider supplies data or metadata The agreement may constrain the scope of the data or metadata that can be provided, by means of a Constraint.
	+source	Association to a data or reference metadata source which can process a data or metadata query.
<i>Datasource</i>	Abstract class:  Sub classes are:  <i>SimpleDatasource</i>  <i>WebServices Datasource</i>	Identification of the location or service from where data or reference metadata can be obtained.
	+sourceURL	The URL of the data or reference metadata source (a file or a web service).
SimpleDatasource		An SDMX-ML data set accessible as a file at a URL.
<i>WebServices Datasource</i>	Abstract class: Inherits from:  <i>Datasource</i>  Sub classes are:  <i>RESTDatasource</i>  <i>SOAPDatasource</i>	A data or reference metadata source which can process a data or metadata query.



Class	Feature	Description
RESTDataSource		A data or reference metadata source that is accessible via a RESTful web services interface.
SOAPDataSource		A data or reference metadata source that conforms to a SOAP web service interface.
	+WSDLURL	Association to the URL of the Web Service Definition Language (SOAP) or Web Service Application Language (REST) profile of the web service.
Registration		This is not detailed here but is shown as the link between the SDMX-IM and the Registry Service API. It denotes a data or metadata registration document.

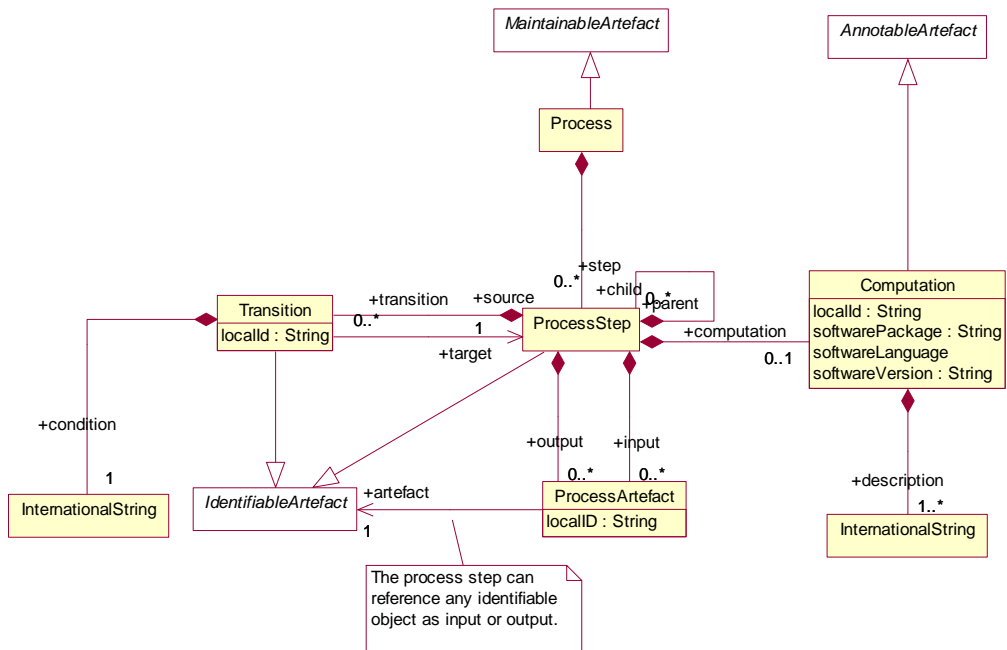
2067 **12 Process**

2068 **12.1 Introduction**

2069 In any system that processes data and reference metadata the system itself is a series of  
 2070 processes and in each of these processes the data or reference metadata may undergo a  
 2071 series of transitions. This is particularly true of its path from raw data to published data and  
 2072 reference metadata. The process model presented here is a generic model that can capture  
 2073 key information about these stages in both a textual way and also in a more formalised way by  
 2074 linking to specific identifiable objects, and by identifying software components that are used.

2075 **12.2 Model – Inheritance and Relationship view**

2076 **12.2.1 Class Diagram**



2077

2078

**Figure 43: Inheritance and Relationship class diagram of Process and Transitions**

2079 **12.2.2 Explanation of the Diagram**

2080 **12.2.2.1 Narrative**

2081 The *Process* is a set of hierarchical *ProcessSteps*. Each *ProcessStep* can take zero or  
 2082 more *IdentifiableArtefacts* as input and output. Each of the associations to the input  
 2083 and output *IdentifiableArtefacts* (*ProcessArtefact*) can be assigned a *localID*.

2084

2085 The computation performed by a *ProcessStep* is optionally described by a *Computation*,  
 2086 which can identify the software used by the *ProcessStep* and can also be described in  
 2087 textual form (*+description*) in multiple language variants. The *Transition* describes the  
 2088 execution of *ProcessSteps* from *+source* *ProcessStep* to *+target* *ProcessStep*  
 2089 based on the outcome of a *+condition* that can be described in multiple language variants.

2090

2091 **12.2.2.2 Definitions**

Class	Feature	Description
-------	---------	-------------

Class	Feature	Description
Process	Inherits from <i>Maintainable</i>	A scheme which defines or documents the operations performed on data or metadata in order to validate data or metadata to derive new information according to a given set of rules.
	+step	Associates the Process Steps.
ProcessStep	Inherits from <i>IdentifiableArtefact</i>	A specific operation, performed on data or metadata in order to validate or to derive new information according to a given set of rules.
	+input	Association to the Process Artefact that identifies the objects which are input to the Process Step.
	+output	Association to the Process Artefact that identifies the objects which are output from the Process Step.
	+child	Association to child Processes that combine to form a part of this Process.
	+computation	Association to one or more Computations.
	+transition	Association to one or more Transitions.
Computation		Describes in textual form the computations involved in the process.
	localId	Distinguishes between Computations in the same Process.
	softwarePackage softwareLanguage softwareVersion	Information about the software that is used to perform the computation.
	+description	Text describing or giving additional information about the computation. This can be in multiple language variants.

Class	Feature	Description
Transition	Inherits from <i>IdentifiableArtefact</i>	An expression in a textual or formalised way of the transformation of data between two specific operations (Processes) performed on the data.
	+target	Associates the Process Step that is the target of the Transition.
	+condition	Associates a textual description of the Transition.
ProcessArtefact		Identification of an object that is an input to or an output from a Process Step.
	+artefact	Association to an Identifiable Artefact that is the input to or the output from the Process Step.

2092

## 2093 **13 Transformations and Expressions**

### 2094 **13.1 Scope**

2095 The purpose of this package in the model is to be able to track the derivation of data. It is  
2096 similar in concept to lineage in data warehousing – i.e. how data are derived.

2097

2098 The functionality of this part of the model allows the identification and documentation of the  
2099 calculations performed (these will normally be automated, program calculations), as well as  
2100 defining structures that support a syntax neutral expression “grammar” that can specify the  
2101 operations at a granular level such that a program can “read” the metadata and compose the  
2102 expression required in whatever computer language is appropriate.

2103

2104 This part of the model also allows specifying and documenting the coherence rules among  
2105 different data, expressing them as calculations (for example, the coherence rule “ $a + b = c$ ”  
2106 can be written as “ $a + b - c = 0$ ” and checked through the calculation “if( $(a + b - c) = 0$ , then  
2107 ..., else ...)”).

2108

2109 It should be noted that the model represented below is similar in scope and content to the  
2110 Expression metamodel in the Common Warehouse Metamodel (CWM) developed by the  
2111 Object Management Group (OMG). This specification can be found at:

2112

2113 <http://www.omg.org/cwm>

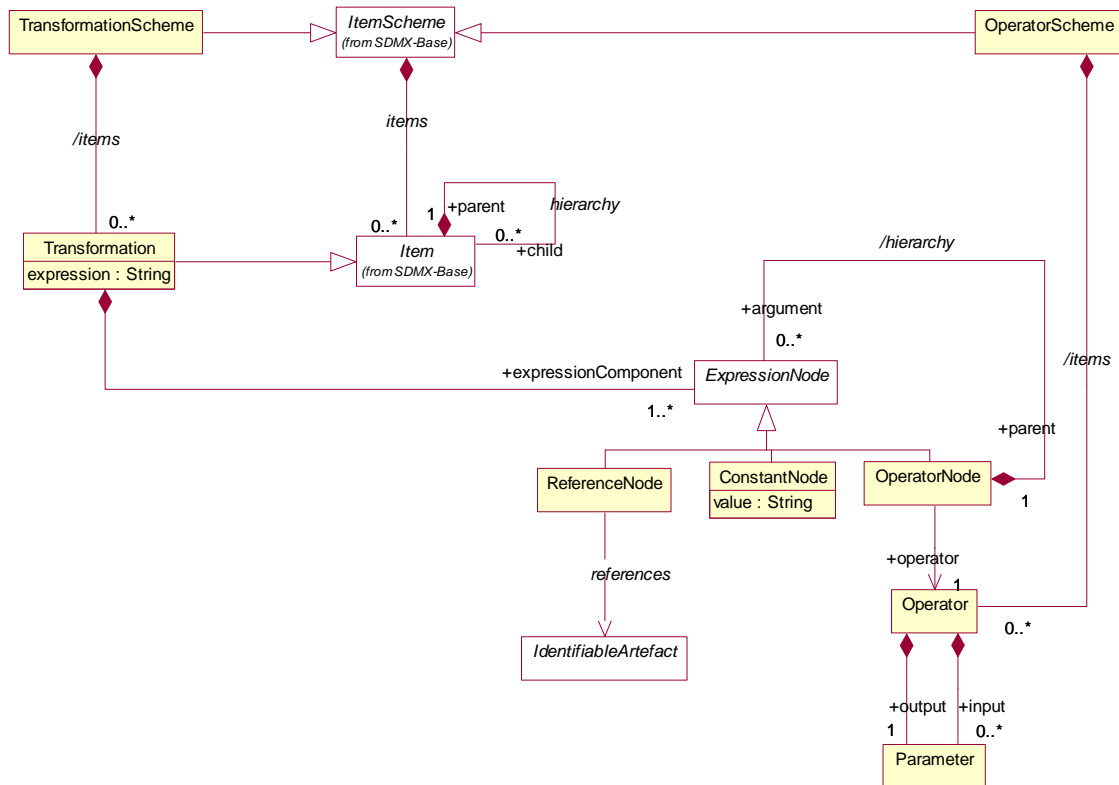
2114

2115 The Expression metamodel is described in Section 8.5 of Part 1 of the CWM specification. The  
2116 class diagram shown below is an interpretation of the CWM Expression metamodel expressed  
2117 in the base classes of the SDMX-IM.

2118

2119 **13.2 Model - Inheritance View**

2120 **13.2.1 Class Diagram**



2121  
2122

**Figure 44: Inheritance and relationship class diagram of transformation classes**

2123 **13.2.2 Explanation of the Diagram**

2124 **13.2.2.1 Narrative**

2125 There are three type of *ItemScheme* relevant to this model.

2126

2127

1. A *TransformationScheme* which comprises one or more *Transformations*.

2128

2. An *OperatorScheme* which comprises one or more *Operators*.

2129

3. An *ExpressionNodeScheme* scheme which contains one or more *ExpressionNodes*.

2130

2131

2132

The model presented here is a basic framework which can be used for expressions and transformations, but requires more work on elaborating its integration into the model and its actual use within the model. This elaboration will be in a future release of the standard.

2134

2135

2136

The expression concept in the SDMX-IM takes a functional view of expression trees, resulting in the ability of relatively few expression node types to represent a broad range of expressions. Every function or traditional mathematical operator that appears in an expression hierarchy is represented by the *+operator* role on the association to *Operator* which in turn comprises input and output *Parameter*. For example, the arithmetic plus operation “a + b” can be thought of as the function “sum(a, b).” The “sum” is the *Operator*, and “a” and “b” are

2141

2142 its `Parameters`. A parameter is a generic possible input and output of an operator (e.g. base  
2143 and exponent are the parameters of the power operator), while an argument is the specific  
2144 value that a parameter takes in a specific calculation (e.g. in the Einstein equation “ $E = MC^2$ ”  
2145 the arguments of the “power” operation are “C” (the base) and “2” (the exponent)). The actual  
2146 semantics of a particular function or operation are left to specific tool implementations and are  
2147 not captured by the SDMX-IM.

2148  
2149 The hierarchical nature of the SDMX-IM representation of expressions is achieved by the  
2150 recursive nature of the `OperatorNode` association. This association allows the sub-  
2151 hierarchies within an expression to be treated as actual arguments of their parent nodes.

2152  
2153 The model can be used equally to define data derivations and to define integrity checks (e.g.  
2154 the Sum of A+B must equal C).

2155  
2156 Although the model defines the data structures that are used to contain a syntax neutral  
2157 expression, the model itself does not specify a syntax neutral expression grammar.  
2158 Alternatively, the function can be described in a text form either as an unstructured  
2159 explanation of the function, or as a more formal language like BNF<sup>2</sup>.

2160  
2161 The data structures work as follows:

2162  
2163 The actual basic mathematical functions that need to be performed (e.g. sum, multiply, divide,  
2164 assign (=), <, > etc.) are defined as `Operators` an `OperatorScheme`. For each `Operator`  
2165 the input and output `Parameters`, are defined in the `Parameter` class.

2166  
2167 The calculations are defined as `Transformations` in a `TransformationScheme`. A  
2168 `Transformation` is a specific calculation and is specified by means of an expression, which  
2169 is obtained by applying one or more `Operators` in the desired order (for example, in the  
2170 textual form, using parenthesis) and specifying the actual arguments for the `Operators`'  
2171 `Parameters`; the result of the whole expression is assigned (=) to the model item that is the  
2172 result of the `Transformation` (that is “E” in the Einstein equation). A `Transformation`  
2173 operates on existing `IdentifiableArtefacts` and its result is another  
2174 `IdentifiableArtefact`. A calculated `IdentifiableArtefact` may be in its turn be an  
2175 operand of other `Transformations`.

2176  
2177 The expression of a `Transformation` (for example, for the Einstein equation calculus, “ $E =$   
2178  $M*(C**2)$ ”) may be decomposed in a hierarchy of `ExpressionNodes` (in the example, “M”,  
2179 “C”, “2”, \*, \*\*). The `ExpressionNode` can be a `ReferenceNode`, a `ConstantNode` or an  
2180 `OperatorNode`. The `ReferenceNode` references an identifiable model artefact  
2181 (in the example, “M” and “C”). The `ConstantNode` is by definition a constant value (in the  
2182 example “2”). The `OperatorNode` references an `Operator` in the `OperatorScheme` (in the  
2183 example \*, \*\*). The `Transformation` has an association to its component  
2184 `ExpressionNodes`.

2185  
2186 The hierarchy of the `ExpressionNodes` conveys the order in which the operators are applied  
2187 in the expression and is obtained by means of the `/hierarchy` association of the  
2188 `OperatorNode` class, in which the child `ExpressionNodes` are the arguments of the

---

<sup>2</sup> BNF: Backus Naur Form

2189 parent `OperatorNode`. The child `ExpressionNodes` must correspond to the formal  
 2190 parameters of the `Operator` referenced by the parent `OperatorNode` in the correct  
 2191 sequence. The (child) `ExpressionNode` can be the result of another operation (that is  
 2192 another `OperatorNode`) or can be a `Constant` or can be a reference to an  
 2193 *IdentifiableArtefact* (`ReferenceNode`). All *IdentifiableArtefacts* in the SDMX-  
 2194 IM have a unique urn comprising the values of the individual objects that identify it. The  
 2195 structure of this urn is defined in the Registry Specification. An example would be the urn of a  
 2196 code which comprises the agency:code-list-id.code-id – an actual example is  
 2197 "urn:sdmx:org.sdmx.infomodel.codelist.Code=TFFS:CL\_AREA(1.0).1A".  
 2198

### 2199 13.2.2.2 Definitions

Class	Feature	Description
Transformation Scheme	Inherits from <i>ItemScheme</i>	A scheme which defines or documents the transformations required in order to derive or validate data from other data.
Transformation	Inherits from <i>Item</i>	An individual Transformation.
	+expressionComponent	Association to an Expression Node.
<i>ExpressionNode</i>	Abstract class  Sub Classes <i>ReferenceNode</i> <i>ConstantNode</i> <i>OperatorNode</i>	A node in a possible hierarchy of nodes that together define or document an expression.
	/hierarchy	Association to child Expression Nodes
<i>ReferenceNode</i>	Inherits from <i>ExpressionNode</i>	A specific type of Expression Node that references a specific object.
	references	Association to the Identifiable Artefact that is the referenced object.
<i>ConstantNode</i>	Inherits from <i>ExpressionNode</i>	A specific type of Expression Node that contains a constant value.
	value	The value of the Constant
<i>OperatorNode</i>	Inherits from <i>ExpressionNode</i>	A specific type of Expression Node that references an Operator
	+operator	Association to an Operator that defines the mathematical operator of the Operator Node.



Class	Feature	Description
	+arguments	Association to mathematical arguments of an Operator Node.
OperatorScheme	Inherits from <i>ItemScheme</i>	A scheme which defines mathematical operators.
Operator	Inherits from <i>Item</i>	The mathematical operator in an Operator Scheme.
	+input	Association to the input Parameters of the Operator
	+output	Association to the output Parameter of the Operator.
Parameter		The input or output of an Operator.

2200

2201 **14 Appendix 1: A Short Guide To UML in the SDMX**  
2202 **Information Model**

2203 **14.1 Scope**

2204 The scope of this document is to give a brief overview of the diagram notation used in UML.  
2205 The examples used in this document have been taken from the SDMX UML model.

2206 **14.2 Use Cases**

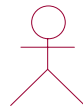
2207 In order to develop the data models it is necessary to understand the functions that require to  
2208 be supported. These are defined in a use case model. The use case model comprises actors  
2209 and use cases and these are defined below.

2210 The **actor** can be defined as follows:

2211 *“An actor defines a coherent set of roles that users of the system can play when*  
2212 *interacting with it. An actor instance can be played by either an individual or an*  
2213 *external system”*

2214 The actor is depicted as a stick man as shown below.

2215



Data Publisher

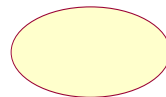
**Figure 45 Actor**

2218

2219 The **use case** can be defined as follows:

2220 *“A use case defines a set of use-case instances, where each instance is a sequence of*  
2221 *actions a system performs that yields an observable result of value to a particular*  
2222 *actor”*

2223



Publish Data

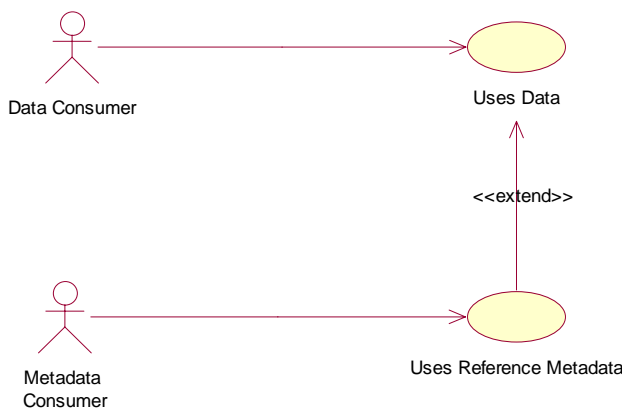
**Figure 46 Use case**

2224



**Figure 47 Actor and use case**

2225



**Figure 48 Extend use cases**

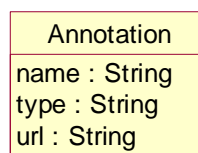
2226 An extend use case is where a use case may be optionally extended by a use case that is  
 2227 independent of the using use case. The arrow in the association points to the owning use case  
 2228 of the extension. In the example above the Uses Data use case is optionally extended by the  
 2229 Uses Metadata use case.

2230 **14.3 Classes and Attributes**

2231 **14.3.1 General**

2232 A class is something of interest to the user. The equivalent name in an entity-relationship  
 2233 model (E-R model) is the entity and the attribute. In fact, if the UML is used purely as a means  
 2234 of modelling data, then there is little difference between a class and an entity.

2235



**Figure 49 Class and its attributes**

2236

2237 Figure 49 shows that a class is represented by a rectangle split into three compartments. The  
 2238 top compartment is for the class name, the second is for attributes and the last is for  
 2239 operations. Only the first compartment is mandatory. The name of the class is Annotation,  
 2240 and it belongs to the package SDMX-Base. It is common to group related artefacts (classes,  
 2241 use-cases, etc.) together in packages. . Annotation has three “String” attributes – name,

2242 type, and url. The full identity of the attribute includes its class e.g. the name attribute is  
 2243 Annotation.name.

2244  
 2245 Note that by convention the class names use UpperCamelCase – the words are  
 2246 concatenated and the first letter of each word is capitalized. An attribute uses  
 2247 lowerCamelCase - the first letter of the first (or only) word is not capitalized, the remaining  
 2248 words have capitalized first letters.

2249 **14.3.2 Abstract Class**

2250 An abstract class is drawn because it is a useful way of grouping classes, and avoids drawing  
 2251 a complex diagram with lots of association lines, but where it is not foreseen that the class  
 2252 serves any other purpose (i.e. it is always implemented as one of its sub classes). In the  
 2253 diagram in this document an abstract class is depicted with its name in italics, and coloured  
 2254 white.  
 2255



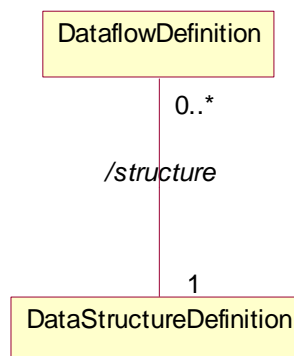
**Figure 50 Abstract and concrete classes**

2256 **14.4 Associations**

2257 **14.4.1 General**

2258 In an E-R model these are known as relationships. A UML model can give more meaning to  
 2259 the associations than can be given in an E-R relationship. Furthermore, the UML notation is  
 2260 fixed (i.e. there is no variation in the way associations are drawn). In an E-R diagram, there  
 2261 are many diagramming techniques, and it is the relationship in an E-R diagram that has many  
 2262 forms, depending on the particular E-R notation used.

2263 **14.4.2 Simple Association**



**Figure 51 A simple association**

2264  
 2265 Here the DataflowDefinition class has an association with the  
 2266 DataStructureDefinition class. The diagram shows that a DataflowDefinition can  
 2267 have an association with only one DataStructureDefinition (1) and that a

2268 DataStructureDefinition can be linked to many DataflowDefinitions (0..\*). The  
 2269 association is sometimes named to give more semantics.

2270

2271 In UML it is possible to specify a variety of “multiplicity” rules. The most common ones are:

2272

2273 • Zero or one (0..1)

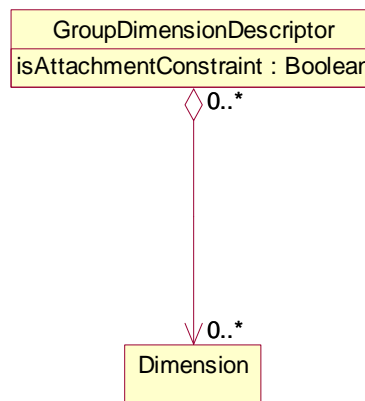
2274 • Zero or many (0..\*)

2275 • One or many (1..\*)

2276 • Many (\*)

2277 • Unspecified (blank)

2278 **14.4.3 Aggregation**

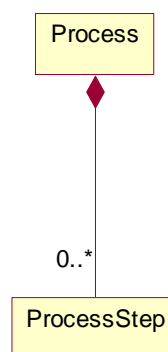


2279

2280

**Figure 52: A simple aggregate association**

2281



**Figure 53 A composition aggregate association**

2282

2283 An association with an aggregation relationship indicates that one class is a subordinate class

2284 (or a part) of another class. In an aggregation relationship. There are two types of aggregation,

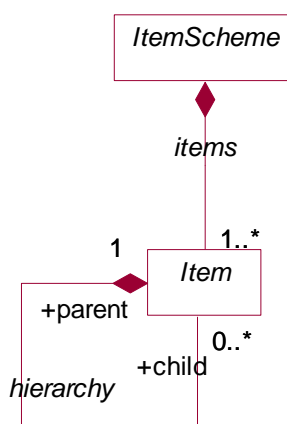
2285 a simple aggregation where the child class instance can outlive its parent class, and a

2286 composition aggregation where

2287 the child class's instance lifecycle is dependent on the parent class's instance lifecycle. In the  
 2288 simple aggregation it is usual, in the SDMX Information model, for this association to also be a  
 2289 reference to the associated class.

2290 **14.4.4 Association Names and Association-end (role) Names**

2291 It can be useful to name associations as this gives some more semantic meaning to the model  
 2292 i.e. the purpose of the association. It is possible for two classes to be joined by two (or more)  
 2293 associations, and in this case it is extremely useful to name the purpose of the association.  
 2294 Figure 54 shows a simple aggregation between *CategoryScheme* and *Category* called  
 2295 */items* (this means it is derived from the association between the super classes – in this case  
 2296 between the *ItemScheme* and the *Item*, and another between *Category* called */hierarchy*.  
 2297



**Figure 54 Association names and end names**

2298 Furthermore, it is possible to give role names to the association-ends to give more semantic  
 2299 meaning – such as parent and child in a tree structure association. The role is shown with “+”  
 2300 preceding the role name (e.g. in the diagram above the semantic of the association is that a  
 2301 *Item* can have zero or one parent *Items* and zero or many child *Item*).

2302  
 2303 In this model the preference has been to use role names for associations between concrete  
 2304 classes and association names for associations between abstract classes. The reason for  
 2305 using an association name is often useful to show a physical association between two sub  
 2306 classes that inherit the actual association between the super class from which they inherit.  
 2307 This is possible to show in the UML with association names, but not with role names. This is  
 2308 covered later in “Derived Association”.

2309  
 2310 Note that in general the role name is given at just one end of the association.

2311 **14.4.5 Navigability**

2312 Associations are, in general, navigable in both directions. For a conceptual data model it is not  
 2313 necessary to give any more semantic than this.

2314  
 2315 However, UML allows a notation to express navigability in one direction only. In this model this  
 2316 “navigability” feature has been used to represent referencing. In other words, the class at the  
 2317 navigable end of the association is referenced from the class at the non-navigable end. This is  
 2318 aligned, in general, with the way this is implemented in the XML schemas.

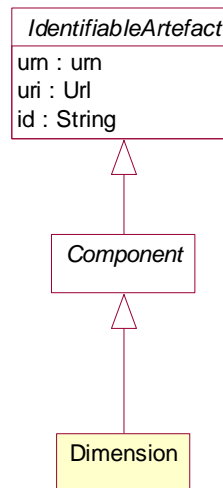


**Figure 55 One way association**

2319 Here it is possible to navigate from A to B, but there is no implementation support for  
 2320 navigation from B to A using this association.

2321 **14.4.6 Inheritance**

2322 Sometimes it is useful to group common attributes and associations together in a super class.  
 2323 This is useful if many classes share the same associations with other classes, and have many  
 2324 (but not necessarily all) attributes in common. Inheritance is shown as a triangle at the super  
 2325 class.  
 2326



**Figure 56 Inheritance**

2327 Here the Dimension is derived from Component which itself is derived from  
 2328 *IdentifiableArtefact*. Both Component and *IdentifiableArtefact* are abstract  
 2329 superclasses. The Dimension inherits the attributes and associations of all of the the super  
 2330 classes in the inheritance tree. Note that a super class can be a concrete class (i.e. it exists in  
 2331 its own right as well as in the context of one of its sub classes), or an abstract class.

2332 **14.4.7 Derived association**

2333 It is often useful in a relationship diagram to show associations between sub classes that are  
 2334 derived from the associations of the super classes from which the sub classes inherit. A  
 2335 derived association is shown by “/” preceding the association name e.g. */name*.  
 2336

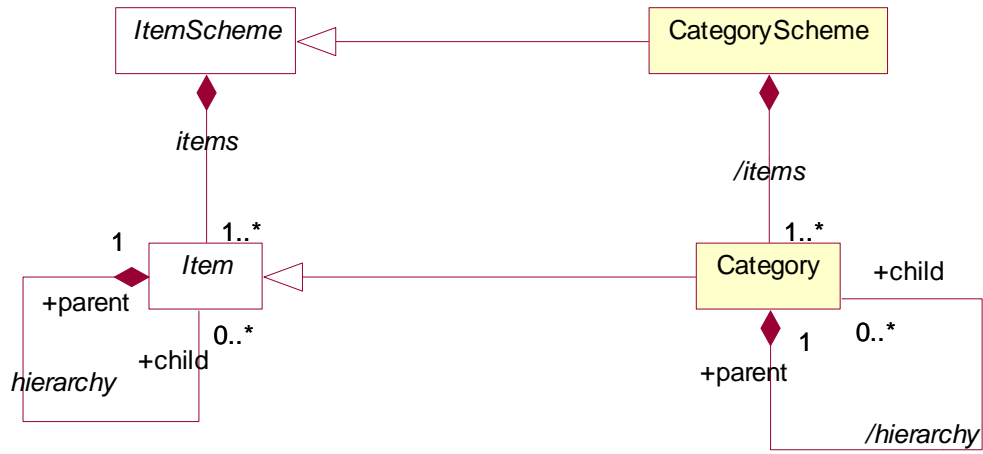


Figure 57 Derived associations

2337