SDMX

INFORMATION MODEL:

UML CONCEPTUAL DESIGN

(VERSION 1.0)

Initial Release September 2004

http://www.sdmx.org/

I. INTRODUCTION 7

A. Modelling technique and diagrammatic notes 7

B. Overall functionality 8

II. ACTORS AND USE CASES 10

A. Actors and use cases 10

B. Use case diagram 10

C. Explanation of the diagram 12

III. SDMX BASE PACKAGE 16

A. Introduction 16

B. SDMX Base 17

Class diagram 17

Explanation of the diagram 17

C. Data Types 21

Class diagram 21

Explanation of the diagram 21

D. The Item Scheme Pattern 22

Context 22

Class Diagram 22

Explanation of the diagram 23

E. The Component Structure Pattern 24

Context 24

Class Diagram 24

Explanation of the diagram 24

F. Organisation Pattern 29

Class Diagram 29
Explanation of the diagram 29

G. Inheritance View 32
Class Diagram 32
Explanation of the diagram 32

H. Relationship View 35
Class Diagram 35
Explanation of the diagram 35

IV. STRUCTURE DEFINITION METADATA 38

A. Introduction 38

B. CodeList 38
Context 38
Class diagram 39
Explanation of the diagram 39

C. ConceptScheme 41
Context 41
Class diagram 41
Explanation of the diagram 42

D. DomainCategory 43
Context 43
Class diagram 44
Explanation of the diagram 44

E. Key Family 46
Context 46
Class diagram 47
Explanation of the diagrams 48

F. Data Set - timeseries 55

Context 55
Class diagram 56
Explanation of the diagram 56

G. Data Set – cross sectional 60
Class diagram 60
Explanation of the diagram 61

V. EXAMPLES 63

A. Introduction 63

B. Example metadata and data 63
Domain scheme 63
Metadata concept scheme 64
Key family 64
Data set 67

C. Collaboration diagrams 68
Domain scheme 68
Metadata concept scheme 69
Key family 70
Data set 71

VI. APPENDIX I: A SHORT GUIDE TO UML IN THE SDMX INFORMATION MODEL 72

A. Scope 72

B. Use cases 72

C. Classes and attributes 73
General 73
Abstract class 74

D. Associations 74

General 74
Simple association 75
Aggregation 75
Association names and association-end (role) names 76
Navigability 77
Inheritance 77
Derived association 78

E. Collaboration diagram 79

VII. APPENDIX II: KEY FAMILIES - A TUTORIAL 81

A. Introduction 81

B. What is a Key Family? 81

C. Grouping Data 82

D. Attachment Levels 83

E. Keys 84

F. Code Lists and Other Representations 85

G. Cross-Sectional Data Structures 86

I. INTRODUCTION

This document is not normative, but provides a detailed view of the information model on which the normative SDMX specifications are based. Format implementors may wish to review the overview of the model presented in *Implementor's Guide for SDMX Format Standards.* Those new to the UML notation or to the concept of key families may wish to read the appendixes in this document as an introductory exercise.

A. Modelling technique and diagrammatic notes

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

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

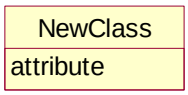


Figure 1: Class with operations suppressed

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

- The attributes will always be present on the class diagram where the class is defined and its attributes and associations are defined.
- On other diagrams, such as inheritance diagrams, the attributes may be suppressed from the class for clarity.

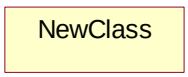


Figure 2: Class with attributes also suppressed

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

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

Class **Nearest MCV[1] Term**	**Feature**	**Description**
`ClassName` Nearest MCV term		
	`attributeName`	.
	associationName	
	+roleName	

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

161 • Whether it is an abstract class
162 • The superclass this class inherits from, if any
163 • The sub classes of this class, if any
164 • Attribute – the `attributeName` is shown in `Courier font`
165 • Association – the *associationName* is shown in the standard font in
166 *italics*
167 • Role – the *+roleName* is shown in the standard font in *italics*

168 **B. Overall functionality**

169 The SDMX Information Model (SDMX-IM) is a conceptual metamodel from which
170 syntax specific implementations are developed. In version 1.0 the metamodel covers
171 the requirements for:

172 • Key family definition including domain category scheme, metadata concept
173 scheme, and code list

174 • Data and related metadata reporting and dissemination

175 The metamodel also supports the specific data requirements of the SDMX registry
176 version 1.0 where these requirements have an impact on the key family definition
177 and data reporting. The functions of the SDMX registry version 1.0, its model, and
178 the map to the SDMX-IM are documented separately.

[1] MCV – Metadata Common Vocabulary, which is under development by SDMX

The SDMX-IM will be developed in the near future to support other aspects of statistical data and metadata reporting and dissemination. For this reason, the model has been structured into a number of packages to aid both concurrent development and ease of understanding. Furthermore, some advanced work has been done on the features in the next version, and this has led to the development of the SDMXBase package where common constructs have been abstracted into "patterns". This is to facilitate re-use of common structures, which can be augmented by re-naming (sub classing) and the addition of specific associations and attributes as required. Knowledge of these patterns will assist software development.

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

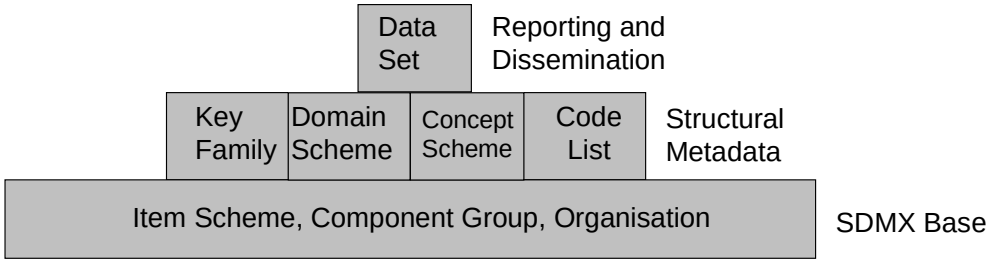


Figure 3: SDMX Information Model Version 1.0 package structure

Although any package can make use of any construct in another package, in conceptual terms the packages are shown in three layers:

- the SDMX Base layer comprises fundamental building blocks which are used by the Structural Metadata layer and the Reporting and Dissemination layer
- the Structural Metadata layer comprises the definition of the structural artefacts needed to support data and metadata reporting and dissemination
- the Reporting and Dissemination layer comprises the definition of the data and metadata containers used for reporting and dissemination

II. ACTORS AND USE CASES

A. Actors and use cases

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

Actor

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

Use case

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

B. Use case diagram

The SDMX-IM model supports the following use cases.

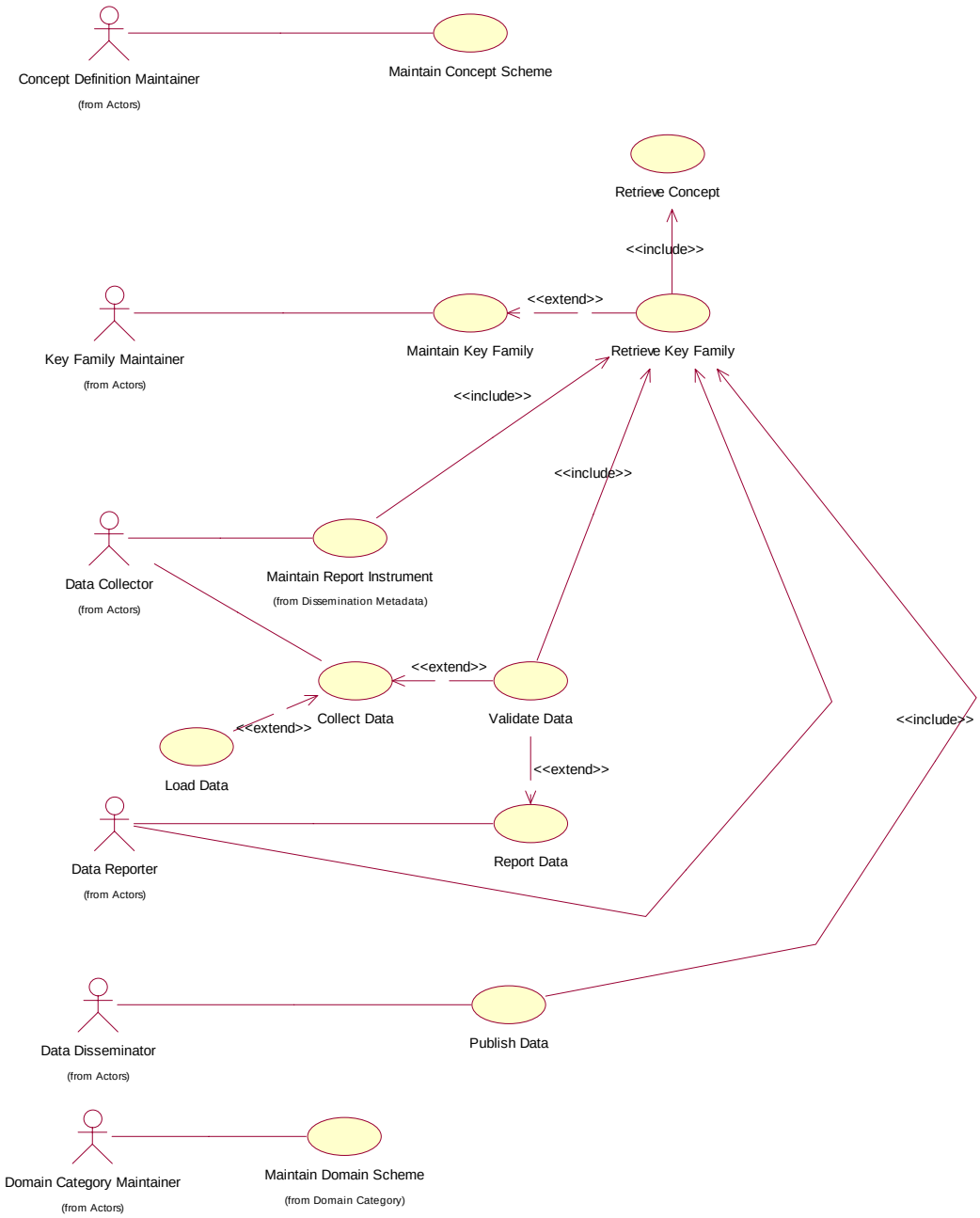


Figure 4: Use cases for data reporting and publishing

220 ## C. Explanation of the diagram

Actor	Use Case	Description
 Concept Definition Maintainer		Responsible for maintaining the classification scheme that defines the metadata concepts used in key family definitions.
	 Maintain Concept Scheme	Creation and maintenance of the concept scheme.
	 Retrieve Concept	Retrieve a metadata concept for use in a process or for viewing.
 Key Family Maintainer		Responsible for maintaining key families.
	 Maintain Key Family	Creation and maintenance of the key family in terms of metadata concepts that comprise the dimensions and attributes of the key family structure, and the code lists used. This use case is extended by the use case Retrieve Key Family.
	 Retrieve Key Family	Retrieve a key family for use in a process or for viewing. This use case includes the

Actor	Use Case	Description
		use case RetrieveConcept.
 Data Collector		Responsible for the collection of statistical data from reporting organizations. The actor is responsible for the collection process, including validation, and possibly, loading into a database. This actor may define report forms based on data flow definition and associated key family structure.
	 Maintain Report Instrument	Maintenance of reporting instruments such as XML, HTML, and spreadsheet forms.
	 Collect Data	The data collection process. This use case is optionally extended by Validate Data and Load Data.
	 Validate Data	Validation of the reported data according to the key family definition i.e. check that metadata concepts used for dimensions and attributes are consistent with the key family definition, and the values reported for concepts are consistent with the key family definition (e.g. they are contained in the

Actor	Use Case	Description
		code list linked to the concept). This use case includes the use case Retrieve Key Family.
	 Load Data	Loading of the data into a database.
 Data Reporter		Responsible for reporting the data. Note that these data are in the form of a data set and in this model this actor does not report raw data. The reporter may validate the data set according to the key family definition.
	 Report Data	Reporting of the data. This use case is optionally extended by Validate Data.
 Data Disseminator		Responsible for disseminating data, including its publication.
	 Publish Data	Publish the data, for instance on a web site. This use case includes the use case Retrieve Key Family

Actor	Use Case	Description
 Domain Category Maintainer		Responsible for maintaining a domain category scheme.
	 Maintain Domain Scheme	Maintenance of a domain scheme. The actor will define the domain scheme in terms of domain categories that comprise the scheme.

221

III. SDMX BASE PACKAGE

A. Introduction

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

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

- It is accepted "Best Practise" to identify fundamental archetypes occurring in a model
- identification of commonly found structures or "patterns" leads to easier understanding
- identification of patterns encourages re-use

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

B. SDMX Base

Class diagram

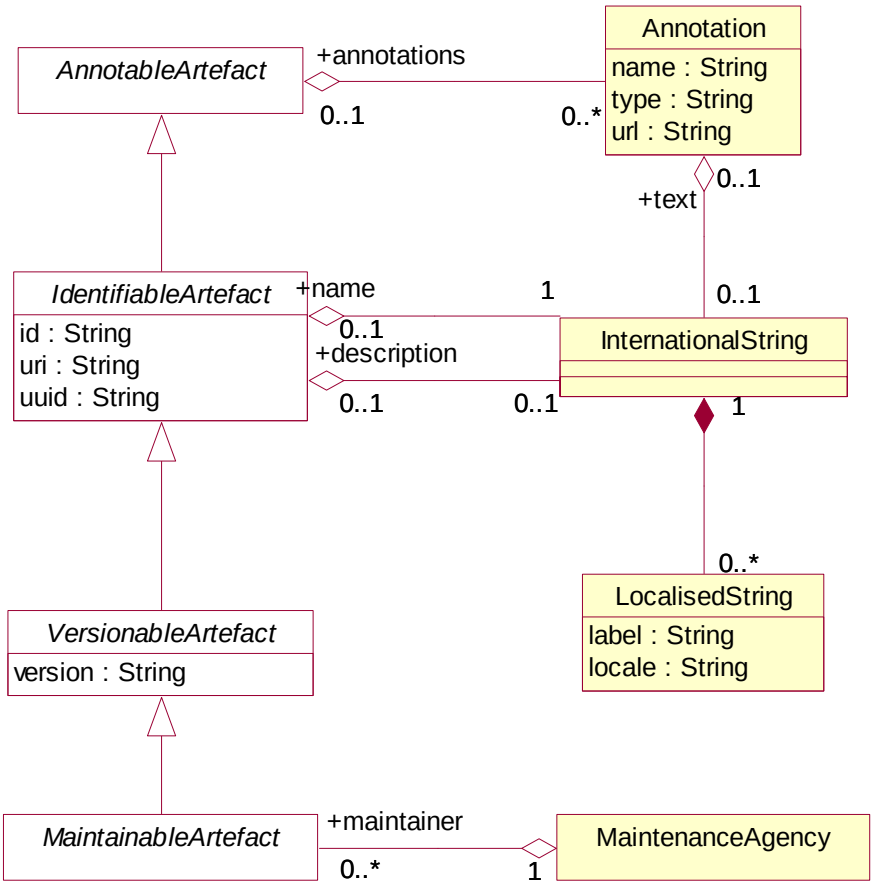


Figure 5: SDMX base classes

Explanation of the diagram

a) *Narrative*

This group of classes forms the nucleus of the SDMX Information Model. It provides features which are generally reusable by derived classes to support horizontal functionality such as identity, versioning etc. None of these classes are specific to statistical models, and could be used for any domain.

All classes derived from the abstract class *AnnotableArtefact* may have Annotations (or notes); this is to comply with the ability to add notes to all SDMX-ML

elements. The Annotation is used to convey extra information to describe any SDMX construct. This information may be in the form of a URL reference and / or a multilingual text (represented by the association to InternationalString)

The *IdentifiableArtefact* is an abstract class that comprises the basic attributes needed by many of the other classes in the metamodel. Concrete classes based on *IdentifiableArtefact* all inherit the ability to be uniquely identified. They also inherit the ability to carry annotations. In addition, the *+description* and *+name* roles supports multilingual descriptions and names for all objects based on *IdentifiableArtefact*. The InternationalString supports the representation of a description in multiple locales (locale is similar to language but includes geographic variations such as Canadian French, US English etc.). The LocalisedString supports the representation of a description in one locale.

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

MaintainableArtefact further adds the ability for derived classes to be maintained via its association to *MaintenanceAgency*.

The inheritance chain from *AnnotableArtefact* through to *MaintainableArtefact* allows SDMX classes to inherit the features they need, whether it be simple annotation, identity, versioning or maintenance.

b) *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
AnnotableArtefact	Direct sub classes are: IdentifiableArtefact	Allows additional supporting documentation for SDMX objects to be specified.
	+annotations	This role allows some annotations to be linked to an SDMX object derived from AnnotableArtefact.
Annotation		Supplies additional supporting documentation for SDMX objects.

Class **Nearest MCV Term**	**Feature**	**Description**
	name	A name used to identify an annotation
	type	Specifies how the annotation is to be processed
	url	A link to external descriptive text
	+text	An InternationalString provides the multilingual text content of the annotation via this role.
IdentifiableArtefact	Superclass is *AnnotableArtefact*. Derived classes: *VersionableArtefact*	Provides identity to all derived classes. It also provides annotations to derived classes because it is a subclass of AnnotableArtefact.
	id	The unique identifier of the object.
	uri	Universal resource identifier that may or may not be resolved.
	uuid	Universally unique identifier – this is for use in registries: all registered objects have a uuid.
	+description	A multi-lingual description is provided by this role via the `InternationalString` class.

Class **Nearest MCV Term**	**Feature**	**Description**
	+name	A multi-lingual name is provided by this role via the `InternationalString` class
`VersionableArtefact`	Superclass is *`IdentifiableArtefact`* Derived classes: *`MaintainableArtefact`*	Provides versioning information for all derived objects.
	version	A version string following an agreed convention
`InternationalString`		The `InternationalString` is a collection of `LocalisedStrings` and supports the representation of a description in multiple locales.
`LocalisedString`		The `LocalisedString` supports the representation of a description 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 Nearest MCV Term	Feature	Description
MaintainableArtefact	Inherits from VersionableArtefact Derived classes: *ComponentStructure* *StructureClassifier* *ItemScheme*	An abstract class to group together primary structural metadata artefacts. These artefacts need to be maintained by a Contact acting on behalf of a MaintenanceAgency.
	+maintainer	Derived classes will be maintained by the MaintenanceAgency specified by this role.

C. Data Types

Class diagram

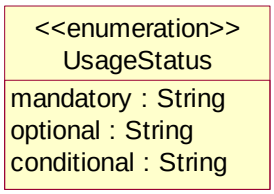


Figure 6: Enumerations

Explanation of the diagram

a) *Narrative*

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

b) *Definitions*

Class Nearest MCV Term	Feature	Description

Class Nearest MCV Term	Feature	Description
`UsageStatus`		Lists the possible values that an attribute can take when it is assigned the data type of `UsageStatus`.
	mandatory	The usage is mandatory.
	optional	The usage is optional.
	conditional	The usage is mandatory when certain conditions are satisfied.

D. The Item Scheme Pattern

Context

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

The Item Scheme is the basis for Domain Category Schemes, Code Lists and Concept Schemes.

Class Diagram

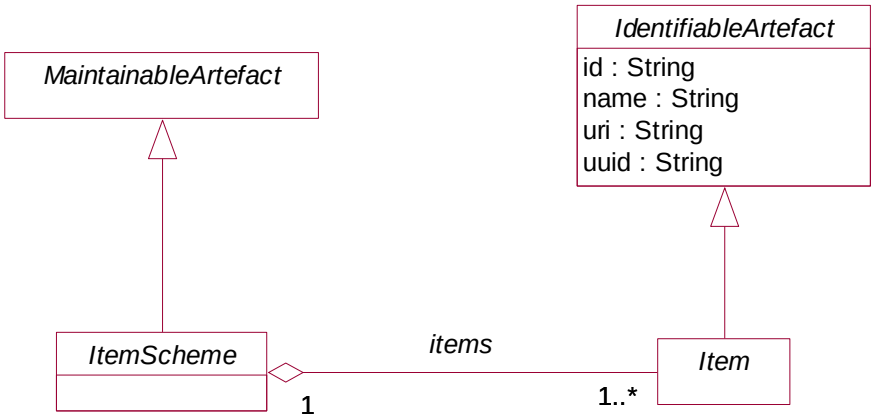


Figure 7: The Item Scheme pattern

Explanation of the diagram

a) *Narrative*

The *ItemScheme* is an abstract class which defines a set of *Item* (this class is also abstract). Its main purpose is to define a mechanism which can be used to create taxonomies which can classify other parts of the SDMX Information Model. It is derived from *MaintainableArtefact* which imbues the ability to be annotated, have identity, versioning and be associated with a MaintenanceAgency. Example of concrete classes are DataCategoryScheme and DataCategory – these are shown later in the Inheritance class diagram.

Item inherits from *IdentifiableArtefact* and therefore has id, uri and uuid attributes, a name in the form of an InternationalString, and may have a description in the form of an InternationalString.

b) *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
ItemScheme	Inherits from: *MaintainableArtefact* Direct sub classes are: DomainCategoryScheme ConceptScheme CodeList	The descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common.
Item	Inherits from: *IdentifiableArtefact* Direct sub classes are DomainCategory Concept Code	The Item is an item of content in an ItemScheme. This may be a node in a taxonomy or ontology, a code in a code list etc.

E. The Component Structure Pattern

Context

The Component Structure is a basic architectural pattern which allows the specification of complex tabular structures which are often found in statistical data (such as key family definitions). A Component Structure is a set of ordered lists. A pattern to underpin this tabular structure has been elicited, so that it can be re-used in future versions of the SDMX-IM to describe other metadata structures.

Class Diagram

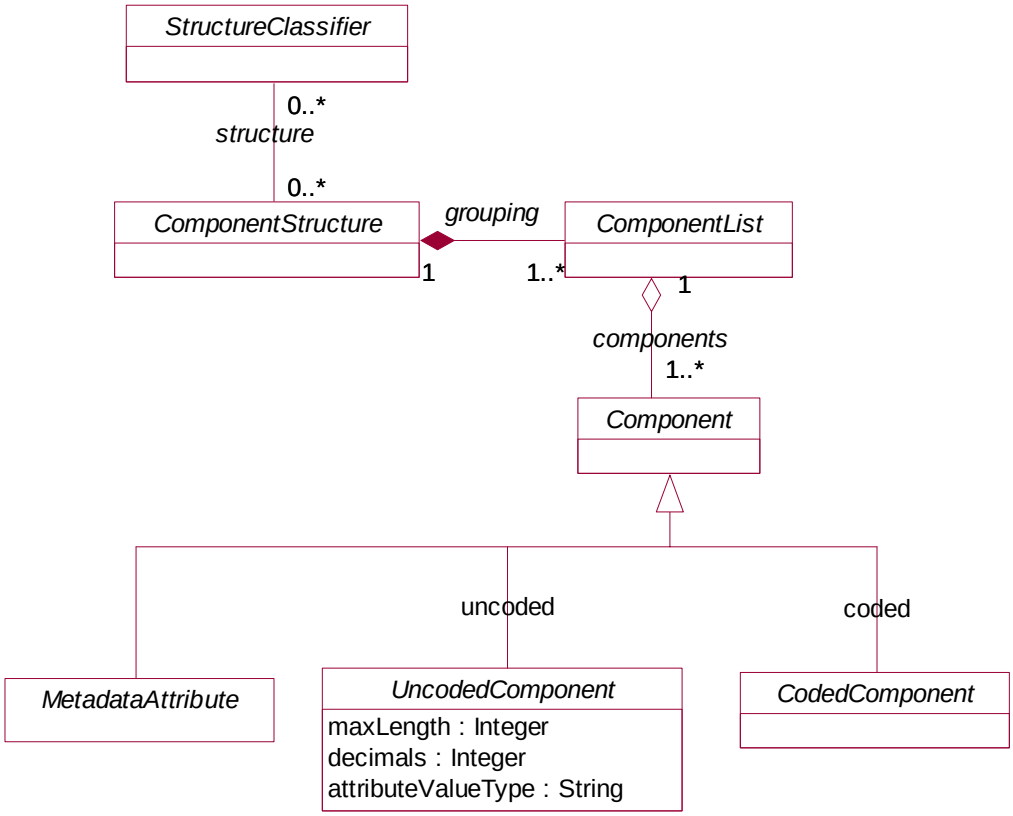


Figure 8: The Component Structure pattern

Explanation of the diagram

a) *Narrative*

The `ComponentStructure` is an abstract class which contains a set of one or more `ComponentList`(s) (this class is also abstract). An example of a concrete

321 *ComponentStructure* is KeyFamily – this is shown later in the Inheritance class
322 diagram. The *ComponentList*(s) are embedded within the
323 *ComponentStructure*, and this is indicated by the solid diamond on the *grouping*
324 association.
325 The *ComponentList* is a list of one or more *Component*(s). The *ComponentList*
326 has several concrete descriptor classes based on it: KeyDescriptor,
327 GroupKeyDescriptor are two examples. Again, based on the example of the
328 KeyFamily as a concrete class, this would normally have three
329 *ComponentList*(s): one for the Dimensions, one for Measures and one for
330 MetadataAttributes (this is explained later in the key family section). In the case
331 of a KeyDescriptor acting as a *ComponentList*, its *Component*(s) would be
332 Dimension(s).
333 The *Component* itself is divided into two abstract classes: *CodedComponent*(s) are
334 qualitative in nature and take their values from a pre-defined code list, and
335 *UncodedComponent*(s) which are usually either free text or quantitative values.
336 Finally, the *StructureClassifier* class provides a way to classify a
337 *ComponentStructure* via an *DomainCategory* node of an
338 *DomainCategoryScheme*. This is described in more detail in the Relationship View
339 section below.
340

341 *Definitions*

342

Class Nearest MCV Term	Feature	Description
StructureClassifier	Inherits from: *MaintainableArtefact* (see figure 10) Direct sub classes are: DataflowDefinition	A specific sub class of Component Structure may be classified against an Item in an Item Scheme. When this happens, a StructureClassifier is created. In concrete terms (sub-classes) an example would be a dataflow definition which allows data to be shared for a particular domain category with a given structure (key family).
	structure	An association to a ComponentStructure specifying the structure of data that has been classified.
ComponentStructure KeyFamily	Inherits from: *MaintainableArtefact* (see figure 10) Direct sub classes are: KeyFamily	Abstract specification of a list of lists to define a complex tabular structure. In concrete terms (sub-classes) an example would be a key family definition.
	grouping	A composite association to one or more component lists.

Class **Nearest MCV Term**	**Feature**	**Description**
ComponentList	Inherits from: *AttachableArtefact* Direct sub classes are: KeyDescriptor GroupKeyDescriptor MeasureDescriptor MeasureTypeDescriptot AttributeDescriptor	An abstract definition of a list of components. In more concrete terms, a key descriptor defines the list of dimensions that make up a key for a key family. The inheritance from attachable artefact allows metadata attributes to attach to keys and group keys.
	components	An aggregate association to one or more components which make up the list.
Component Dimension	Inherits from: *AttachableArtefact* (See figure 10). Direct sub classes are: *UncodedComponent* *CodedComponent* *MetadataAttribute*	A component is an abstract super class used to define qualitative and quantitative data and metadata items that belong to a ComponentList and hence a component structure. Component is refined through its sub-classes.
UncodedComponent Measure	Inherits from: *Component* Direct sub classes are: Measure UncodedAttribute	An uncoded component is an abstract class used to define quantitative values and free-text data
	maxLength	The maximum length of the component.

Class Nearest MCV Term	Feature	Description
	`decimals`	The decimal precision of the component if the component is numeric.
	`attributeValueType`	identifies the type of value expected in the component (values are alpha, alphanumeric, numeric)

`CodedComponent`
Dimension
	Inherits from:
	`Component`
	Direct sub classes are:
	`Dimension`
	`CodedAttribute`
		A coded component is an abstract class used to define qualitative values which are drawn from a defined value set.

`MetadataAttribute`
Attribute
	Inherits from:
	`Component`
	Direct sub classes are:
	`UncodedAttribute`
	`CodedAttribute`
		Used to define metadata concepts via its sub-classes which are either free-text or coded values.

F. Organisation Pattern

Class Diagram

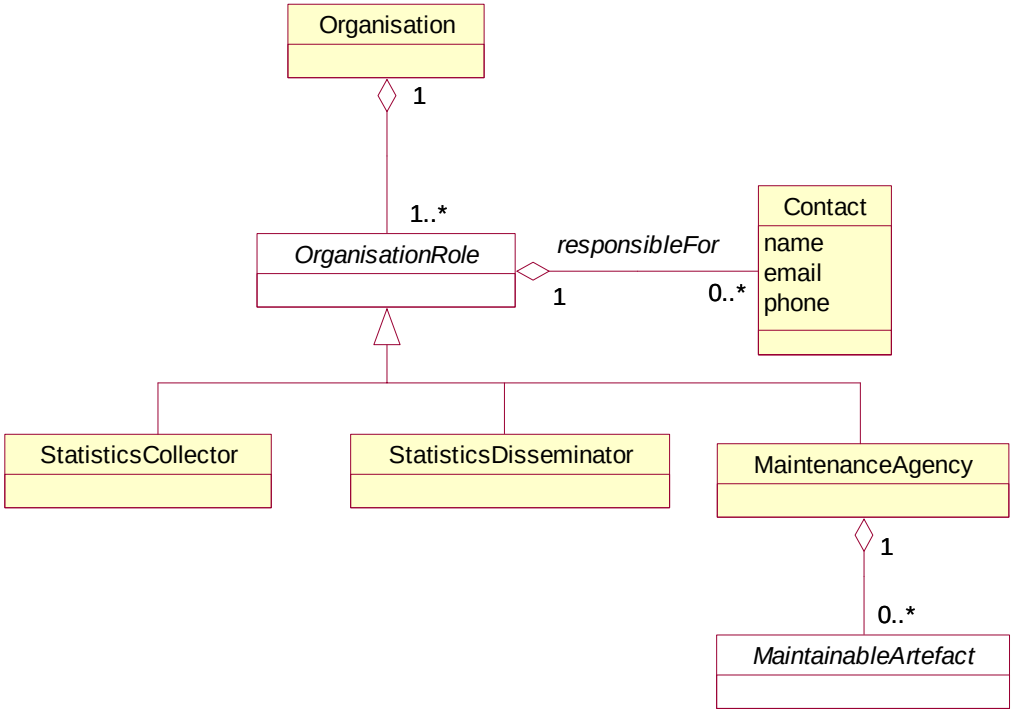


Figure 9: The Organisation class diagram

Explanation of the diagram

a) *Narrative*

An `Organisation` can play a number of *`OrganisationRole`*. Four roles are identified at present: `StatisticsCollector`; `StatisticsDisseminator`; `DataReporter`; `MaintenanceAgency`.. The classes that are associated with the `DataReporter`, `StatisticsCollector`, and `StatisticsDisseminator`, are defined in the package(s) where they are relevant. Note that the `DataReporter` reports a data set and in this model this `OrganisationRole` does not include raw data collection.

Each *`OrganisationRole`* has a set of Contacts, who are individuals responsible for an organisation for each role that it plays. So, for example, one set of contacts would be responsible for an organisation in its role as maintenance agency, and a different set of individuals would be responsible for statistics dissemination.

360 The `MaintenanceAgency` can maintain a variety of *`MaintainableArtefact`*.
361 This is an abstract class and the concrete classes are shown in the table below and
362 described in the section “Inheritance”.

363 *b) Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
`Organisation`	Inherits from *`IdentifiableArtefact`*	Source: MCV An organization is a unique framework of authority within which a person or persons act, or are designated to act, towards some purpose.
`OrganisationRole`	Inherits from *`IdentifiableArtefact`*	The role of an organisation in the processes of statistics collection, dissemination, and metadata maintenance.
`MaintenanceAgency` RegistrationAuthority	Inherits from *`OrganisationRole`*	Responsible agency for maintaining artefacts such as statistical classifications, glossaries, key family structural definitions.
`StatisticsCollector`	Inherits from *`OrganisationRole`*	A collector of statistical data or metadata.

Class **Nearest MCV Term**	**Feature**	**Description**
`StatisticsDisseminator`	Inherits from *`OrganisationRole`*	A disseminator of statistical data or metadata Dissemination is an activity in the survey life cycle to distribute or transmit statistical data and metadata to its users.
`DataReporter`	Inherits from *`OrganisationRole`*	A reporter of data.
`Contact`		A person who acts for an Organisation and who is responsible for performing duties with respect to an *`OrganisationRole.`*
`MaintainableArtefact`	Inherits from *`IdentifiableArtefact`* Sub classes *`ComponentStructure`* *`StructureClassifier`* *`ItemScheme`*	An abstract class to group together structural metadata artefacts. These artefacts need to be maintained by a Contact acting on behalf of a MaintenanceAgency.

G. Inheritance View

Class Diagram

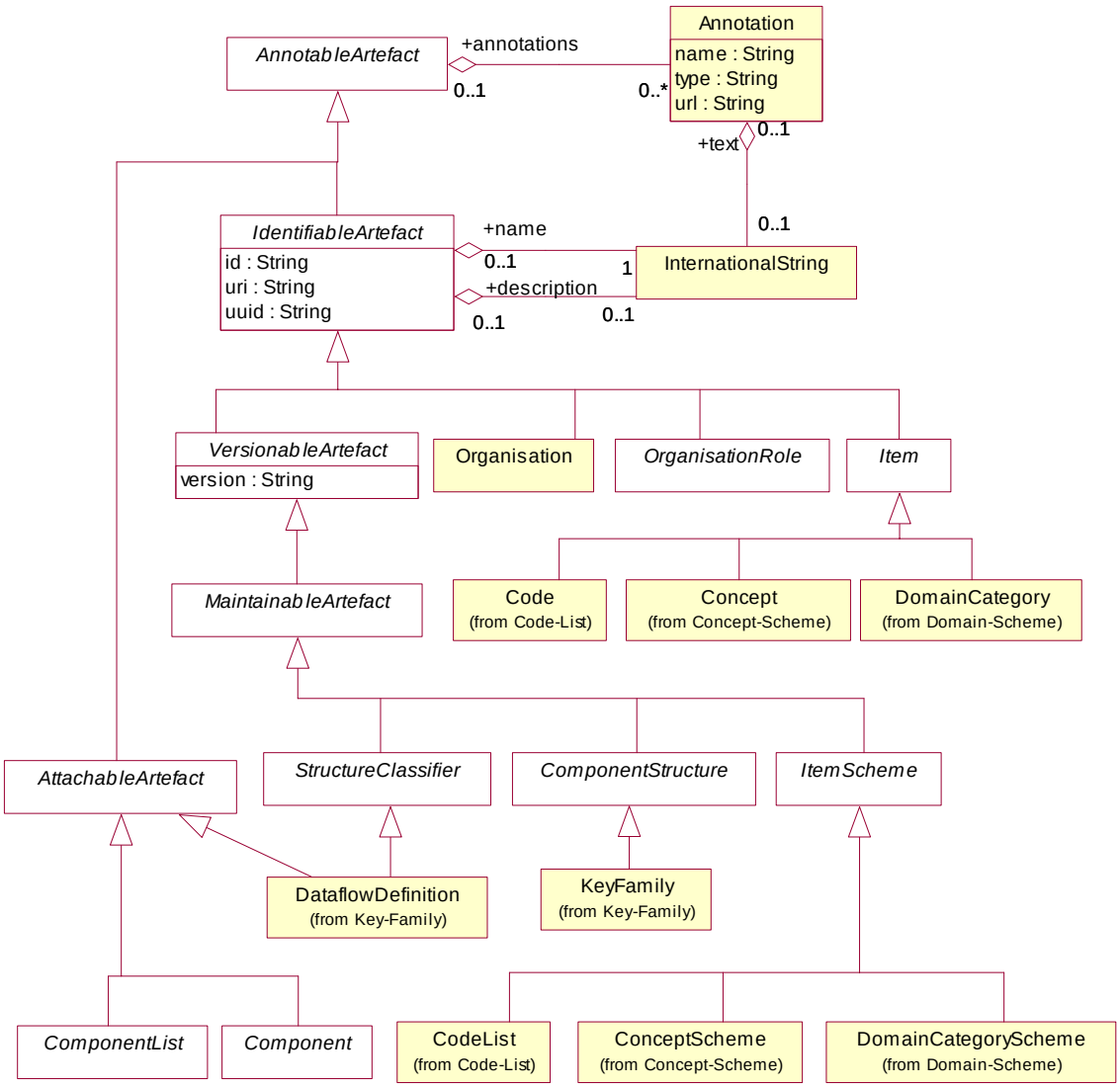


Figure 10: Class inheritance in the SDMX Base package

Explanation of the diagram

a) *Narrative*

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

- *IdentifiableArtefact*
- *AttachableArtefact*

Those classes in the SDMX metamodel which require annotations, global identity, multilingual name, and an optional multilingual description are derived from *IdentifiableArtefact* . These are:

- *VersionableArtefact*
- *OrganisationRole*
- *Organisation*
- *Item*

The class in the SDMX metamodel which requires annotations, global identity, multilingual name, an optional multilingual description, and versioning is derived from *VersionableArtefact* . This is:

- *MaintainableArtefact*

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

- *StructureClassifier*
- *ComponentStructure*
- *ItemScheme*

Those concrete classes in the SDMX metamodel which require to be maintained by Maintenance Agencies all inherit from `MaintainableArtefact,` these are:

- `DataflowDefinition`
- `KeyFamily`
- `CodeList`
- `DomainCategoryScheme`
- `ConceptScheme`

The structures that are an arrangement of objects into hierarchies or lists based on characteristics, and which are maintained as a group inherit from *ItemScheme*, which is a `MaintainableArtefact`. These are:

- `CodeList`
- `ConceptScheme`
- `DomainCategoryScheme`

The component structures that are lists of lists, inherit directly from *`ComponentStructure`*. A *`ComponentStructure`* contains several lists of components (e.g. a KeyFamily contains a list of Dimensions, a list of Measures and a list of Attributes). At version 1.0 this inheritance is restricted to:

- `KeyFamily`

H. Relationship View

Class Diagram

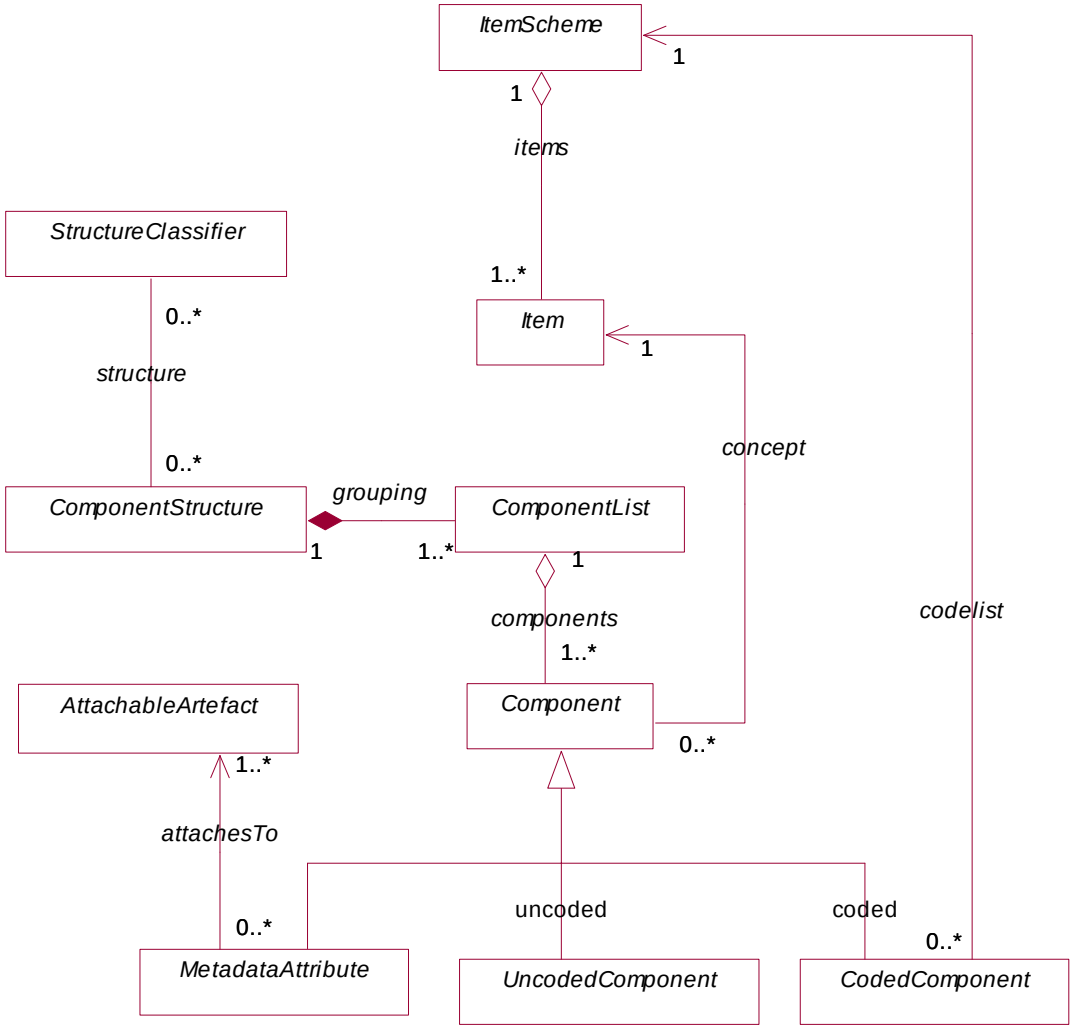


Figure 11 Relationships between classes in the SDMX Base package

Explanation of the diagram

a) *Narrative*

This diagram shows how the two fundamental patterns in the SDMX Base package relate. The *ItemScheme*, and *Item* provide the basic pattern for creating coding schemes which can be used for classification, as previously described. The *ComponentStructure*, *Component* and *ComponentList* provide the pattern to

419 define multiple lists of artefacts, also previously described. There are two points of
420 contact between the two patterns.

421 1. The association *concept* between ***Component*** and ***Item***. In concrete terms,
422 this could be the reference from a `Dimension` to its `Concept`.

423 2. The association from ***CodedComponent*** to ***ItemScheme***. In concrete terms
424 this could be the reference from a `Dimension` to its `CodeList`.

425 Finally, Component can be sub-classified into ***UncodedComponent*** and
426 ***CodedComponent***. A `Measure` is always uncoded, whereas a `Dimension` is always
427 coded. This is described earlier in this section. A MetadataAttribute can be either
428 coded (as in unit of measure) or uncoded (as in a text footnote).
429 The information model allows metadata attributes (see Key Family section) to qualify
430 certain classes. Those classes which can be attribute-qualified inherit from the
431 abstract class ***AttachableArtefact*** and are:

432 - `DataflowDefinition`
433 - ***ComponentList***
434 - ***Component***

435
436 The reason to identify the two patterns (“Component Structure” and “Item Scheme”)
437 is that they will be re-used in later work relating to the SDMX Information Model.

438 *b) Definitions*
439

Class **Nearest MCV Term**	**Feature**	**Description**
	concept	An association from a component to an item which allows sub-classes to define what statistical concept this component represents.
	codelist	An association from a coded component to an

Class Nearest MCV Term	Feature	Description
		item scheme which allows sub-classes to define what code list this component takes its values from.
`AttachableArtefact`	*Direct sub classes are:* *DataflowDefinition* *ComponentList* *Component*	Sub classes identify which artefacts can be specified as allowing attribute qualification or allowing footnotes to be attached.

IV. STRUCTURE DEFINITION METADATA

A. Introduction

Many of the constructs in this layer of the model inherit from the SDMX Base layer. Therefore, it is necessary to study both the inheritance and the relationship diagrams to understand the functionality of individual packages. In simple models these are shown in the same diagram. In more complex models these are shown on separate diagrams.

B. CodeList

Context

The model supports the following use cases

Maintain Key Family

Class diagram

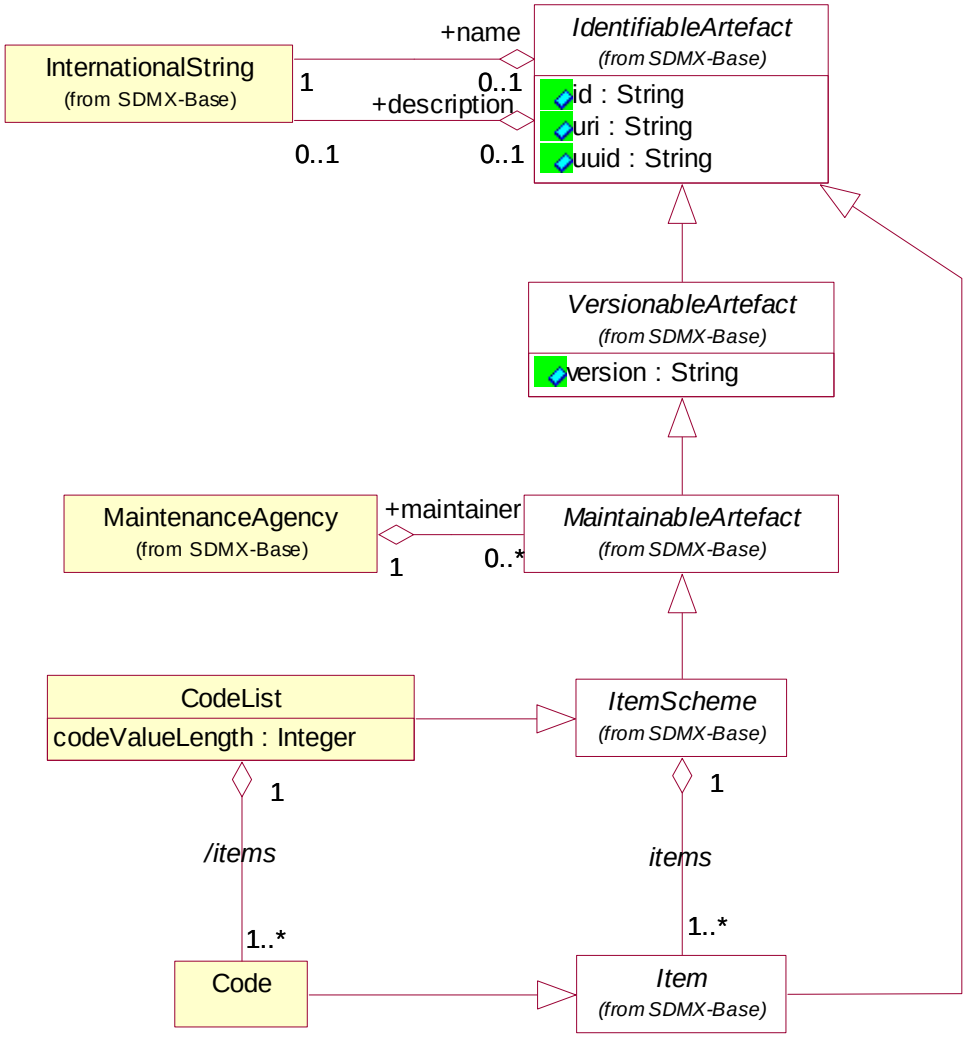


Figure 12: Class diagram of the Code List

Explanation of the diagram

a) *Narrative*

The `CodeList` inherits from the *`ItemScheme`* and therefore has the following attributes:

- `id`
- `uri`

- `uuid`
- `version`

It also has the association to `InternationalString` to support a multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to support notes.

The Code inherits from the *Item* and therefore has the following attributes:

- `id`
- `uri`
- `uuid`

It also has the association to `InternationalString` to support a multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to support notes (not shown).

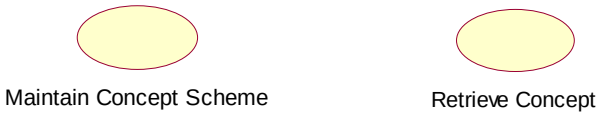
b) *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
`CodeList` Code list	Inherits from *`ItemScheme`*	Source: MCV A list from which some statistical concepts (coded concepts) take their values.
	`codeValueLength`	The length of a code (i.e. identifier) in the code list.
	items	Associates the codes.
`Code`	Inherits from *`Item`*	A value in a code list.

C. ConceptScheme

Context

The model supports the following use cases



Class diagram

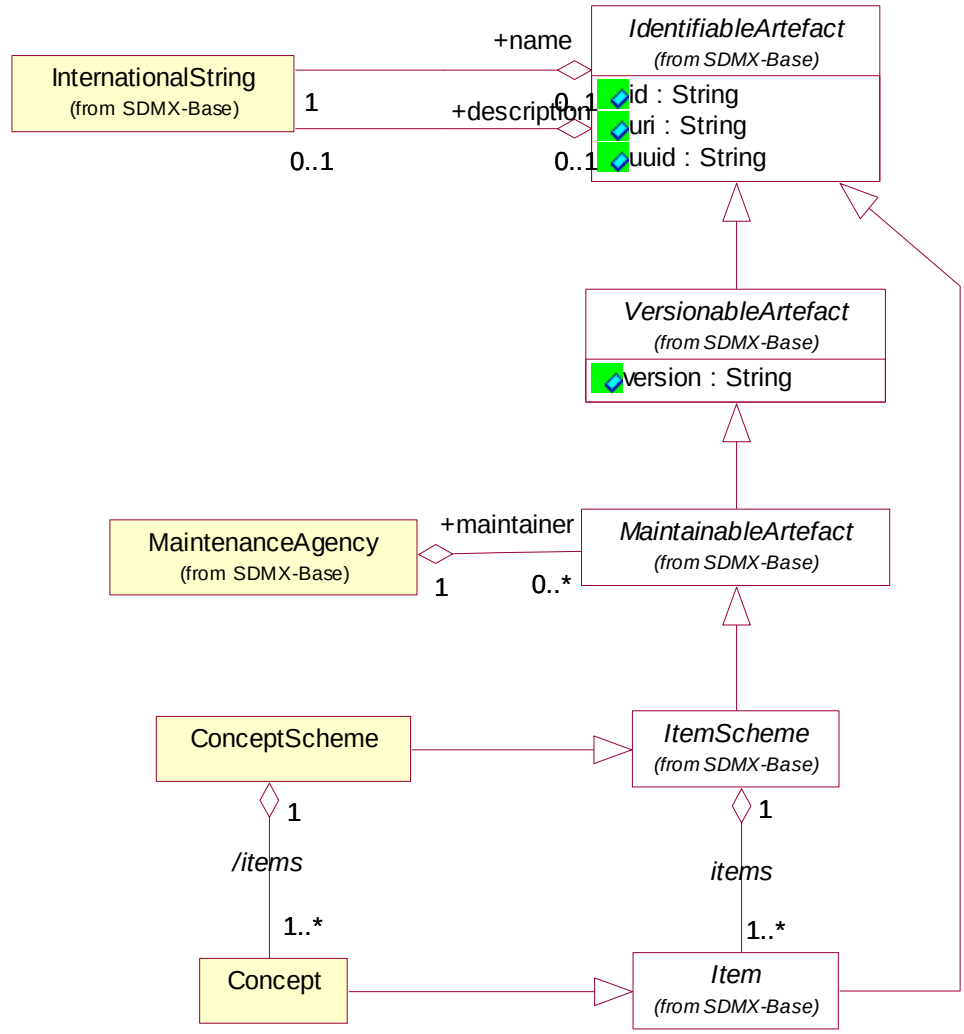


Figure 13: Class diagram of the Concept Scheme

Explanation of the diagram

a) *Narrative*

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

- `id`
- `uri`
- `uuid`
- `version`

It also has the association to `InternationalString` to support a multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to support notes.

The `Concept` inherits from the *Item* and therefore has the following attributes:

- `id`
- `uri`
- `uuid`

It also has the association to `InternationalString` to support a multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to support notes (not shown).

b) *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
`ConceptScheme`	Inherits from *`ItemScheme`*	The descriptive information for an arrangement or division of concepts into groups based on characteristics, which the objects have in common.
	items	Associates the metadata

Class Nearest MCV Term	Feature	Description
		concept.
Concept Concept	Inherits from *Item*	A concept is a unit of knowledge created by a unique combination of characteristics.

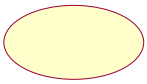
500

501 ### D. DomainCategory

502 **Context**

503 This package defines the structure that supports the definition of and relationships
504 between domain categories. It is similar to the package for metadata concept
505 scheme.

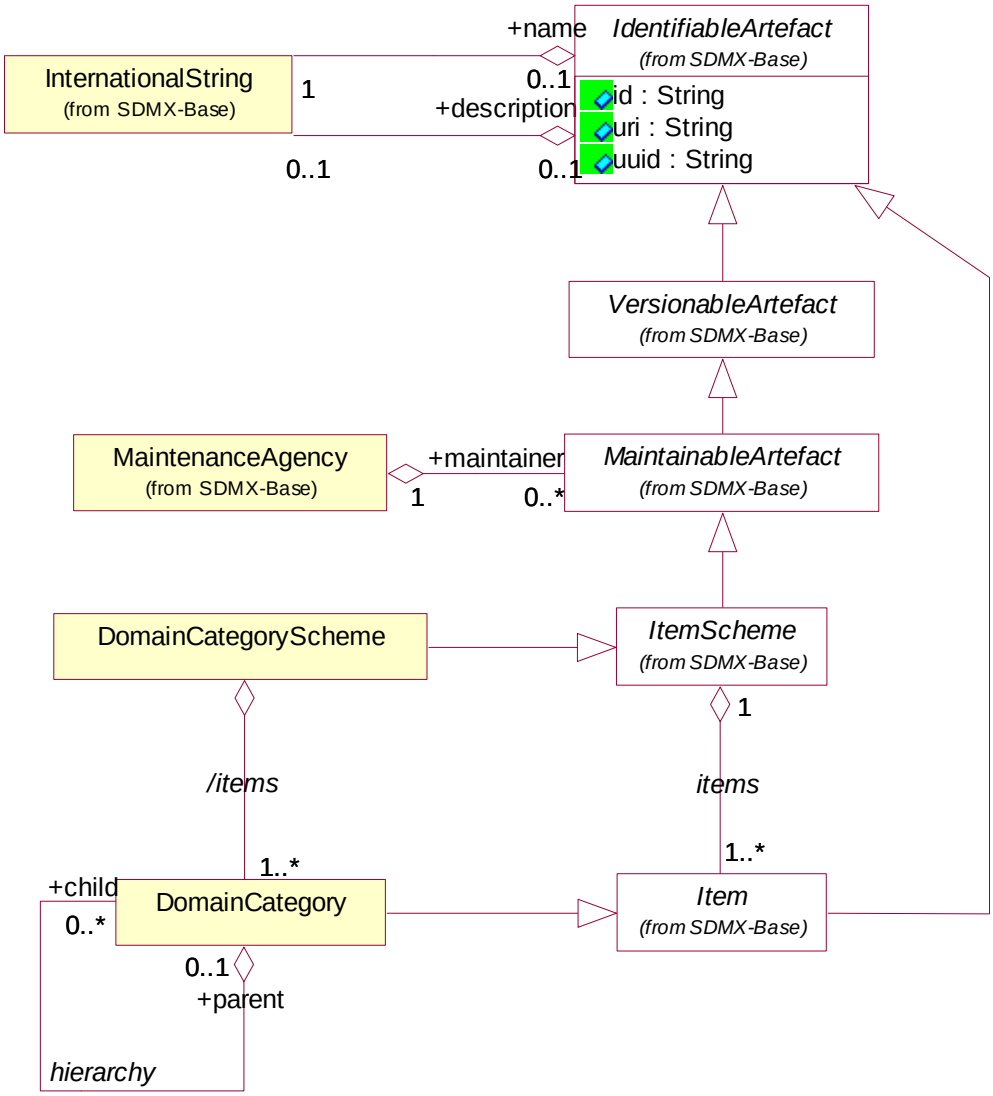
506 The model supports the following use cases.



Maintain Domain Scheme

507

508 **Class diagram**



509

510 **Figure 14: Class diagram of the Domain Category Scheme**

511 **Explanation of the diagram**

512 *a)* *Narrative*

513 The domain categories are modelled as a hierarchical item scheme. The
514 `DomainCategoryScheme` inherits from the *`ItemScheme`* and therefore has the
515 following attributes:

516 - `id`

- `version`
- `uri`
- `uuid`

It also has the association to `InternationalString` to support a multi-lingual name, an optional multi-lingual description, and an association to `Annotation` to support notes.

The `DomainCategory` inherits from the *`Item`* and therefore has the following attributes:

- `id`
- `uri`
- `uuid`

It also has the association to `InternationalString` to support a multi-lingual name, and optional multi-lingual description, and an association to `Annotation` to support notes (not shown).

The `DomainCategoryScheme` can have one or more `DomainCategory`. A `DomainCategory`.can have zero or more child `DomainCategory`, thus creating a hierarchy.

b) *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
`DomainCategoryScheme`	Inherits from *`ItemScheme`*	The descriptive information for an arrangement or division of domain categories into groups based on characteristics, which the objects have in common.
	Items	Associates the domain category.
`DomainCategory`	Inherits from	A domain of interest for

Class Nearest MCV Term	Feature	Description
	Item	which statistics are collected or disseminated (e.g. balance of payments, external debt).
	hierarchy	Associates the parent and the child domain category.

535

536 ### E. Key Family

537 **Context**

538 The key family defines the group of concepts that comprise the key, measures, and
539 associated attributes for which data are collected or disseminated.

540 It supports the following use cases:

541

Class diagram

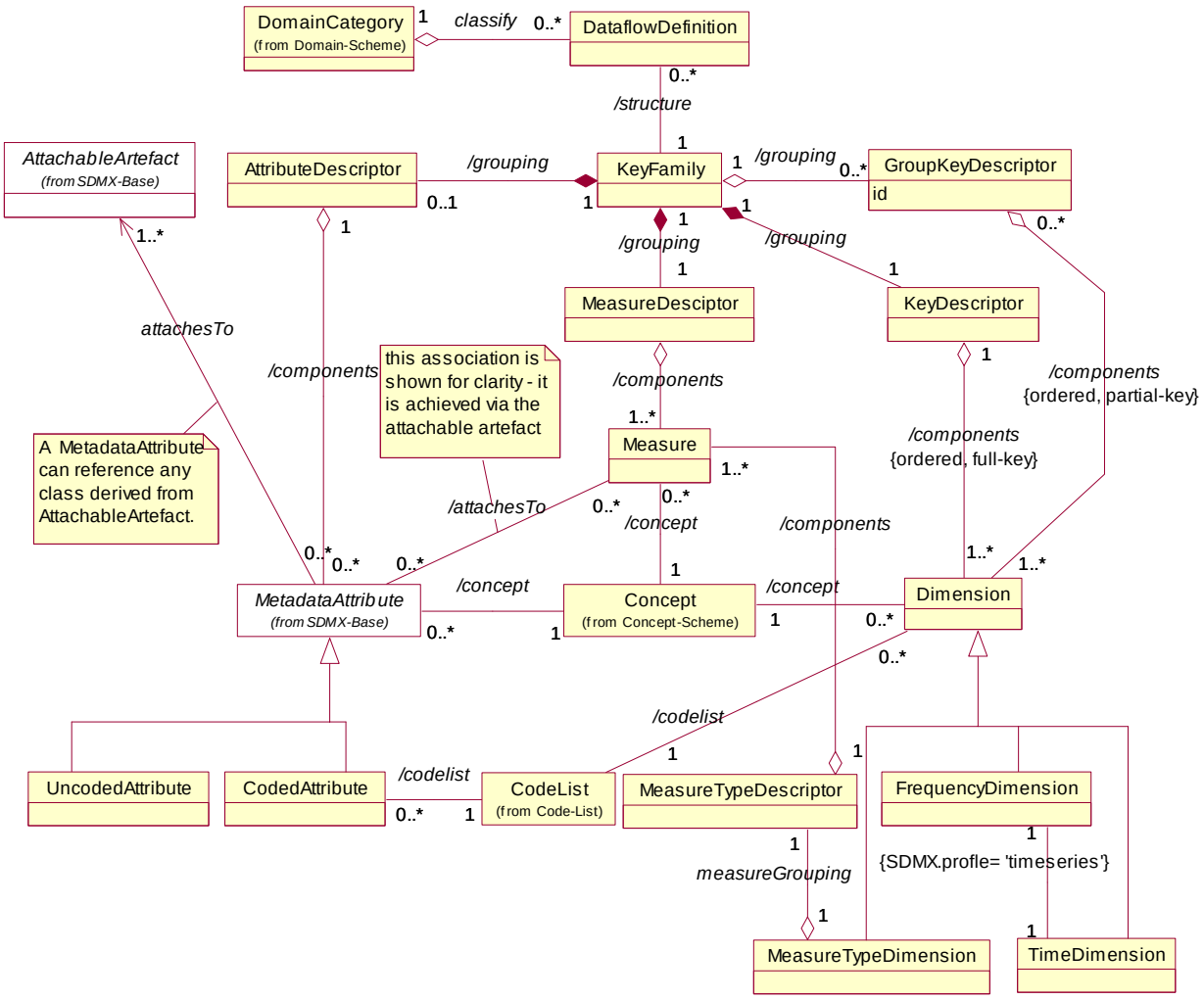


Figure 15: Relationship class diagram of the Key Family

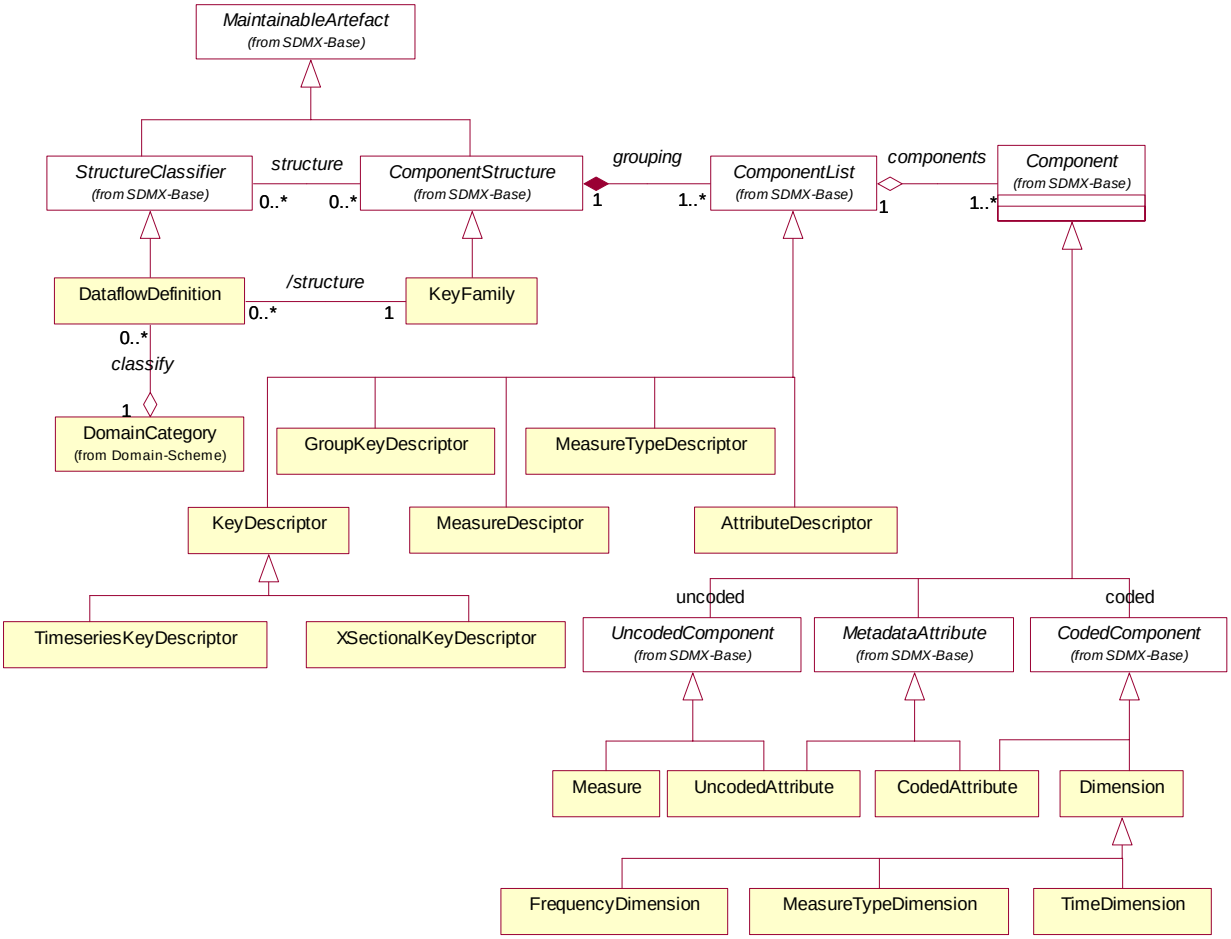


Figure 16: Inheritance class diagram of the Key Family

Explanation of the diagrams

a) *Narrative*

A `KeyFamily` defines the `Dimensions`, *`MetadataAttributes`*, and `Measures`, and associated `CodeLists`, that comprise the valid structure of data and metadata that are contained in a `DataSet` which is defined by a `DataflowDefinition`.

Each `DomainCategory` may have many `DataflowDefinitions` specified. The `DataflowDefinition` associates a `KeyFamily` with one or more `DomainCategory` (possibly from different `DomainCategorySchemes`). This gives the model the ability to state which `DataSets` are to be reported/disseminated for a

given domain, and which DataSets can be reported using the KeyFamily definition. The DataflowDefinition may also contain additional information such as its identifier, and the status of the data reported in a DataSet linked to the DataflowDefinition. Each DataflowDefinition must have one KeyFamily specified which defines the structure of any DataSets to be reported/disseminated. Additionally, the DataflowDefinition holds the metadata that relates to the DataSets that are reported or disseminated using this definition. Such metadata comprises information that does not change with each DataSet but is constant across the DataSets.

Dimension, *MetadataAttribute*, and Measure each link to the Concept that defines its name and semantic. The valid values for a Dimension or CodedAttribute, when used in this KeyFamily, are defined in a CodeList. For the timeseries profile it is necessary to identify the FrequencyDimension and the TimeDimension, and these specialised Dimensions are tightly bound in a one to one association.

The Dimension can be grouped in two ways:

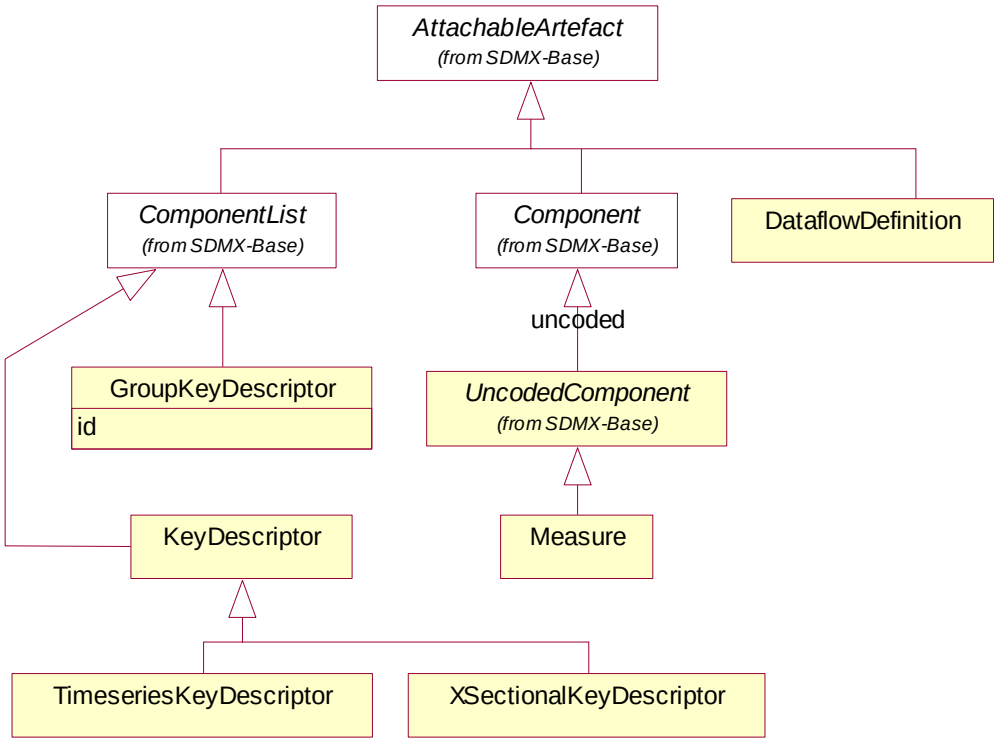
1. There will always be a KeyDescriptor grouping that identifies all of the Dimension comprising the full key. Two specialised classes of the KeyDescriptor are identified: TimeSeriesKeyDescriptor for grouping Dimensions comprising the key components of a time series and XSectionalKeyDescriptor for grouping Dimensions comprising the key components of a cross sectional series

2. Optionally there may be multiple GroupKeyDescriptors each of which identifies the group of Dimensions that can form a partial key. The GroupKeyDescriptor must be identified (GroupKeyDescriptor.id) and is used in the GroupKey of the DataSet to group sets of full keys to which a *MetadataAttribute* can be attached.

The Measure is the observable phenomenon and the set of Measures in the KeyFamily is grouped by a single MeasureDescriptor.

The *MetadataAttribute* defines a characteristic of data that are collected or disseminated and is grouped in the KeyFamily by a single AttributeDescriptor. The *MetadataAttribute* can be specified as being mandatory or optional (Attribute.usageStatus – inherited from MetadataAttribute.usageStatus)

591 The *MetadataAttribute* is an abstract class and is either a CodedAttribute or
592 an UncodedAttribute. With the exception of the Measures defined in a cross
593 sectional KeyFamily, each *MetadataAttribute* can be specified as being
594 attachable to a single *AttachableArtefact*.
595 The MeasureTypeDimension identifies the Measures in a cross sectional key
596 family, and supports the transformation of a cross sectional data set to a time series
597 data set and also vice versa: the Concepts that are the Measures in a cross
598 sectional key family are the codes in the CodeList attached to the
599 MeasureDimension. Furthermore, the CodeList attached to each of
600 CodedAttribute that define the measurement characteristics (such as unit of
601 measure) of each of the Measures in a cross sectional data set are concatenated
602 into a single CodeList that define the measurement characteristics of the single
603 Measure in the time series.
604 The diagram below shows the classes that inherit from *AttachableArtefact*.



605
606 **Figure 17: Class diagram of attachable artefacts in the key family definition**
607

608 The valid `AttachableArtefacts` are:

609 • `DataflowDefinition`

610 • `TimeSeriesKeyescriptor`

611 • `XSectionalKeyDescriptor`

612 • `GroupKeyDescriptor` (identified by the `GroupKeyDescriptor.id`)

613 • `Measure` (identified by the `Concept` that is the `Measure`)

614 *b) Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
`KeyFamily` Key Family		A collection of metadata concepts, their structure and usage when used to collect or disseminate data.
	families	Associates a data set definition to the keyfamily.
	grouping	An association to a set of metadata concepts that have an identified structural role in a key family.
`DataflowDefinition`		A definition of a data set in terms of its identifier, version, periodicity of reporting, confidentiality etc.
	category	Associates the domain category.
`GroupKeyDescriptor`	Inherits from	A set metadata concepts

Class **Nearest MCV Term**	**Feature**	**Description**
	ComponentList	that define a partial key of a key family.
	Id	A unique identifier.
	components	An association to a component in a set of components.
keyDescriptor	Inherits from ComponentList Direct sub classes TimeSeriesKeyDescriptor XSectionalKeyDescriptor	A set metadata concepts that define the full key of a key family.
TimeSeriesKeyDescriptor	Inherits from KeyDescriptor	A set metadata concepts that define the full key of a time series key family.
XSectionalKeyDescriptor	Inherits from KeyDescriptor	A set metadata concepts that define the full key of a cross sectional series key family.
	components	An association to a component in a set of components.
AttributeDescriptor	Inherits from ComponentList	A set metadata concepts that define the attributes of a key family
	components	An association to a component in a set of components.
MeasureDescriptor	Inherits from ComponentList	A set metadata concepts that define the measures of a key family.
	components	An association to a

Class Nearest MCV Term	Feature	Description
		component in a set of components.
	uniqueId	A unique identifier.
	description	A natural language description.
Dimension Dimension	Inherits from CodedComponent Sub classes FrequencyDimension TimeDimension MeasureTypeDimension	A metadata concept used to refer to and identify a part of the key in a key scheme, such as a time series scheme.
	concept	An association to the metadata concept which defines the semantic of the component.
FrequencyDimension	Inherits from Dimension	A metadata concept used to refer to and identify the dimension in a time series that is the frequency dimension.
TimeDimension	Inherits from Dimension	A metadata concept used to refer to and identify the dimension in a series which, in its instance in the data set, will contain a discrete TimePeriod to which the instance of the related Measure or Measures (the Observation)

Class Nearest MCV Term	Feature	Description
		pertains.
`MeasureTypeDimension`	Inherits from `Dimension`	A metadata concept used to refer to and identify the dimension in a time series that defines the concepts for the Measure when cross sectional data is represented in a time series.
	measureGrouping	Associates the `MeasureTypeDescriptor`
`MeasureTypeDescriptor`		Identifies the `Measures` that are associated with the `MeasureTypeDimension.`

`MetadataAttribute`
Attribute
	Abstract class
	Sub classes:
	`CodedAttribute`
	`UncodedAttribute`
		MCV
		A characteristic of an object or entity.

`UnCodedAttribute` Attribute	Abstract class Inherits from *`MetadataAttribute`*	A characteristic of an object or entity that has a free text representation.
`CodedAttribute` Attribute	Abstract class Inherits from *`MetadataAttribute`*	A characteristic of an object or entity that takes its values from a code list.
	`assignmentStatus`	Defines whether the component is mandatory

Class Nearest MCV Term	Feature	Description
		in the key family.
	attachment	Associates the attachable artefacts to which the attribute can be attached when data or metadata are reported or disseminated.
`Measure`	Inherits from `UncodedComponent`	A measure is the concept that is the phenomenon to be measured. In a data set the instance of the measure is called the observation.

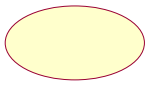
615

616 ### F. Data Set - timeseries

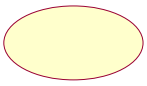
617 **Context**

618 A data set comprises the collection of data values and associated metadata that are
619 collected or disseminated according to a known key family definition.

620 It supports the following use cases:



Report Data



Publish Data

621

Class diagram

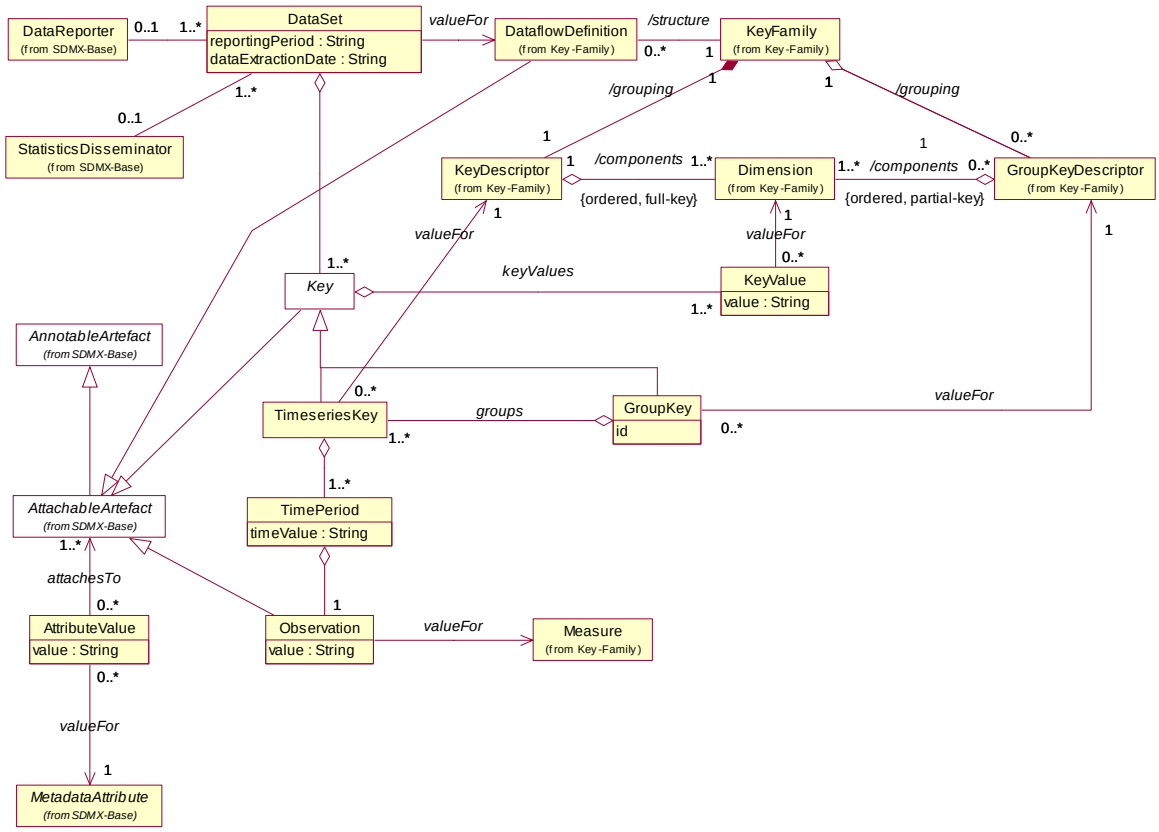


Figure 18: Class diagram of the time series Data Set

Explanation of the diagram

a) *Narrative*

An organisation in the role of `StatisticsDisseminator` or `DataReporter` can be responsible for one or more `DataSet`.

A timeseries `DataSet` is a collection of a set of `Observations` that share the same dimensionality, which is specified by a set of unique `Dimension` defined in the `KeyDescriptor` of the `KeyFamily`, together with associated `AttributeValues` that define specific characteristics about the `Observation`, `Key`, or `DataSet`.

For timeseries each unique combination of `KeyValue` (`TimeseriesKey`) combined with a `TimePeriod`, identifies precisely one `Observation`.

The `Observation` is the value of the variable being measured for the `Concept` associated to the `Measure` in the `MeasureDescriptor` of the `KeyFamily`.

637 The GroupKey is a sub unit of the SeriesKey that has the same dimensionality as
638 the SeriesKey, but defines a subset of the KeyValues of the SeriesKey. Its sub
639 dimension structure is defined in the GroupKeyDescriptor of the KeyFamily
640 identified by the same id as the GroupKey. The id identifies a "type" of group and
641 the purpose of the GroupKey is to identify a set of individual SeriesKey so that one
642 or more MetadataAttribute can be attached at this group level. There can be
643 many types of groups in a DataSet.
644 Each of DataSet, Key, and Observation can have zero or more
645 AttributeValue that defines some metadata about the object to which it is
646 associated.

647 *b) Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**
DataSet Data Set		Any organised collection of data.
	reportingPeriod	A specific time period in a known system of time periods that identifies the period of a report.
	dataExtractionDate	A specific time period that identifies the date and time that the data are extracted from a data source.
	valueFor	Associates a data set definition and thereby a key family to the data set.
Key	Abstract class Sub classes TimeSeriesKey GroupKey	

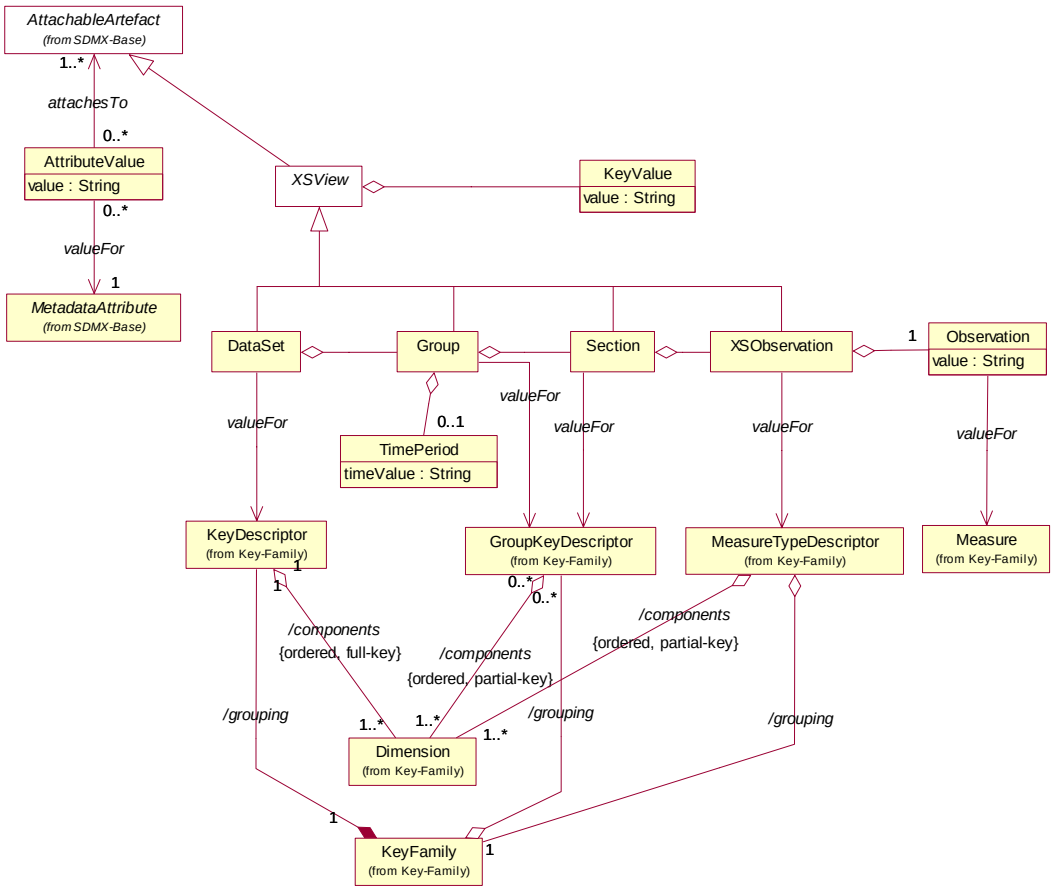
Class Nearest MCV Term	Feature	Description
	valueFor	Associates the individual Key Values that comprise the Key.
`KeyValue`		The value of a component of a Key such as the value of the instance a Dimension in a multidimensional structure, like the key descriptor of a key family.
	`value`	The value of the key component.
	valueFor	Associates a dimension to the key value, and thereby to the metadata concept that is the semantic of the dimension.
`GroupKey`	Inherits from `Key`	A set of key values that comprise a partial key, of the same dimensionality as the series key, and which group together a set of series keys (ie, the scope of the `SeriesKeys` identified by the `GroupKey` is defined using the same `Dimensions` as the `SeriesKey`).

Class Nearest MCV Term	Feature	Description
	valueFor	Associates the group key descriptor defined in the key family.
	groups	Associates a set of series key.
`TimeSeriesKey`	Inherits from `Key`	The `TimeSeriesKey` comprises the product of values of all the `Dimensions` comprising the full key of a series (often called the cartesian product.
`TimePeriod`		A specific time period in a known system of time periods.
	`timeValue`	The value of a time period.
`Observation` Observation		The value, at a particular period, of a particular variable.
	`value`	The value of the observation.
	valueFor	Associates the measure defined in the key family.
`AttributeValue` Attribute Instance		The value of an attribute, such as the instance of a coded attribute or of an uncoded attribute in a structure such as a key family.

Class Nearest MCV Term	Feature	Description
	`value`	The value of the attribute instance.
	valueFor	Associates an attribute defined in the key family.
	attachesTo	Associates the attribute to the object to which it is attached.

G. Data Set – cross sectional

Class diagram



652 **Figure 19: Class diagram of the cross sectional Data Set**

653 **Explanation of the diagram**

654 *a)* *Narrative*

655 A cross sectional `DataSet` is a collection of a set of `XSObservations` that share
656 the same dimensionality, which is specified by a set of unique `Dimension` defined in
657 the `KeyDescriptor` of the `KeyFamily`, together with associated
658 `AttributeValues` that define specific characteristics about the `XSObservation`,
659 `Section`, `Group`, or `DataSet`.

660 The *`Group`* is an artefact that allows compression of data in that it allows a partial
661 key to be defined (according to the `GroupKeyDescriptor` in the `KeyFamily`), and
662 to which attributes can be attached. Importantly, if there is a `TimePeriod` associated
663 with the cross sectional view, then it is always associated with the `Group`.

664 A *`Group`* can contain a number of `Sections`, each described by a partial key
665 according to a `GroupKeyDescriptor` in the `KeyFamily`. Each `Section` has one
666 or more `XSObservation`, each of which comprises an `Observation` and its
667 related *`Measure`* concept. The `MeasureTypeDescriptor` of the `KeyFamily`
668 defines the `Measures` for which `Observation` values are expected, together with
669 appropriate attached attributes.

670 *b)* *Definitions*

Class **Nearest MCV Term**	**Feature**	**Description**

XSView	Abstract class
	Sub classes
	`DataSet`
	`Group`
	`Section`	

Group	Inherits from `XSView`	A Section in a cross sectional view

Class Nearest MCV Term	Feature	Description
	valueFor	Associates the `GroupKeyDescriptor` that defines the partial key.
Section	Inherits from `XSView`	A Section in a cross sectional view
	valueFor	Associates the `GroupKeyDescriptor` that defines the partial key.
XSObservation	Inherits from `XSView`	An Observation in a cross view
	valueFor	Associates the `MeasureTypeDescriptor` that defines the Measures.

671

V. EXAMPLES

A. Introduction

UML collaboration diagrams are drawn for the example data and metadata shown below. These examples show a consistent set of metadata for reporting the example data set. These metadata are neither comprehensive nor complete, but are sufficiently complete for this example:

- Domain scheme
- Concept scheme
- Key family

B. Example metadata and data

Domain scheme

BIS Domain Scheme		
Level 1	**Level 2**	**Level 3**
Real Sector		
	National Accounts	
	Others	
Fiscal Sector		
	Central Government Debt	
	Others	
Financial Sector		
	Interest Rates	
	Others	
External Sector		
	External Debt	
	International Trade	
		Trade In Goods
		Trade In Services

683

684 **Metadata concept scheme**

685

BIS Concept Scheme	
Identifier	**Label**
TIME	Time
FREQ	Frequency
JD_TYPE	Data Type (amounts outstanding net disbursement or charges)
JD_CATEGORY	Debt category
VIS_CTY	Vis-à-vis country
AVAILABILITY	Availability
COLLECTION	Collection indicator
DECIMALS	Decimals
OBS_CONF	Observation confidentiality
OBS_STATUS	Observation status
OBS_PRE_BREAK	Pre-break observation value
UNIT	Unit
UNIT_MULTIPLIER	Unit multiplier
OBS_VALUE	Observation value

686

687 **Key family**

688 Name: BIS_JOINT_DEBT

689 Dataset Definition: CREDITOR_JOINT_DEBT

690 Domain Category: EXTERNAL_DEBT

691 *Dimensions*

Dimensions - Key		
Type	**Concept**	**Code list**

Time Dimension	TIME	
Frequency Dimension	FREQ	CL_FREQ
Dimension	JD_TYPE	CL_JD_TYPE
Dimension	JD-CATEGORY	CL_JD_CATEGORY
Dimension	VIS-CTY	CL_BIS_IF-REF_AREA
Dimensions- Group Key: Sibling		
Type	**Concept**	**Code list**
Dimension	JD_TYPE	CL_JD_TYPE
Dimension	JD-CATEGORY	CL_JD_CATEGORY
Dimension	VIS-CTY	CL_BIS_IF-REF_AREA

692 *Measures*

Concept
OBS_VALUE

693

694 *Attributes*

Cell			
OBS_VALUE			
Attributes			
Concept	**Assignment status**	**Attachment level**	**Code list**
AVAILABILITY	Mandatory	Key	CL_AVAILABILITY
COLLECTION	Mandatory	Sibling group	CL_COLLECTION
DECIMALS	Mandatory	Sibling group	CL_DECIMALS
OBS_CONF	Conditional	Measure	CL_BIS_OBS-CONF
OBS_STATUS	Mandatory	Measure	CL_OBS-STATUS
OBS_PRE_BREAK	Conditional	Measure	
UNIT	Mandatory	Sibling group	CL_BIS_UNIT
UNIT_MULTIPLIER	Mandatory	Sibling group	CL_UNIT_MULT

695

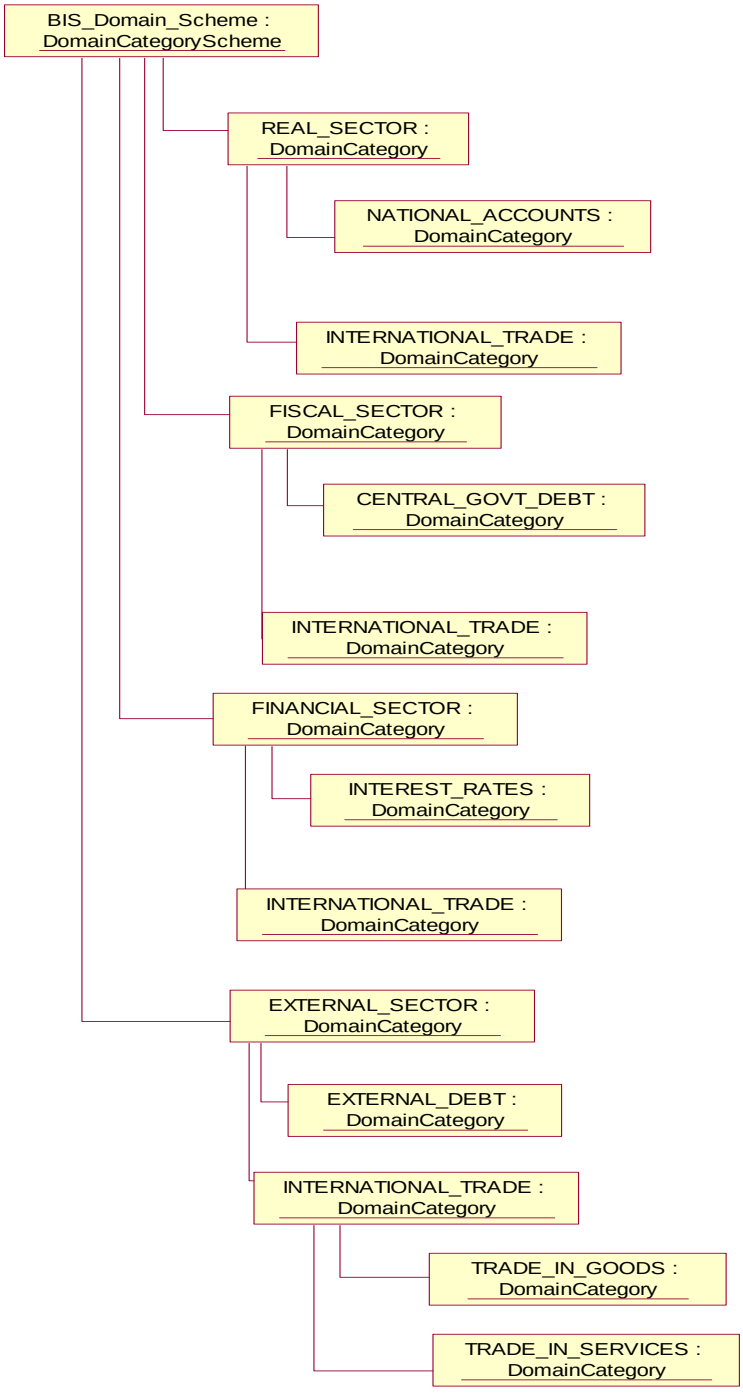
Data set

Dataset Definition: CREDITOR_JOINT_DEBT Dataset Id: JD014 Version: 1.0 Data extraction date = 11 March 2003 09:30		
Key: Dimensions: FREQ=M;JD_TYPE=P;JD_CATEGORY=A;VIS_CTY=MX Attributes: COLLECTION=B		
Date/Time	**Observation value**	**Attributes**
2001-01-01	3.14	OBS_STATUS=A
2000-02-01	3.14	OBS_STATUS=A
2000-03-01	4.29	OBS_STATUS=A
2000-04-01	6.04	OBS_STATUS=A

Group Key: Sibling
Dimensions: JD_TYPE=P;JD_CATEGORY=A;VIS_CTY=MX
Attributes:
AVAILABILITY=A
DECIMALS=2
UNIT=USD
UNIT_MULT=5

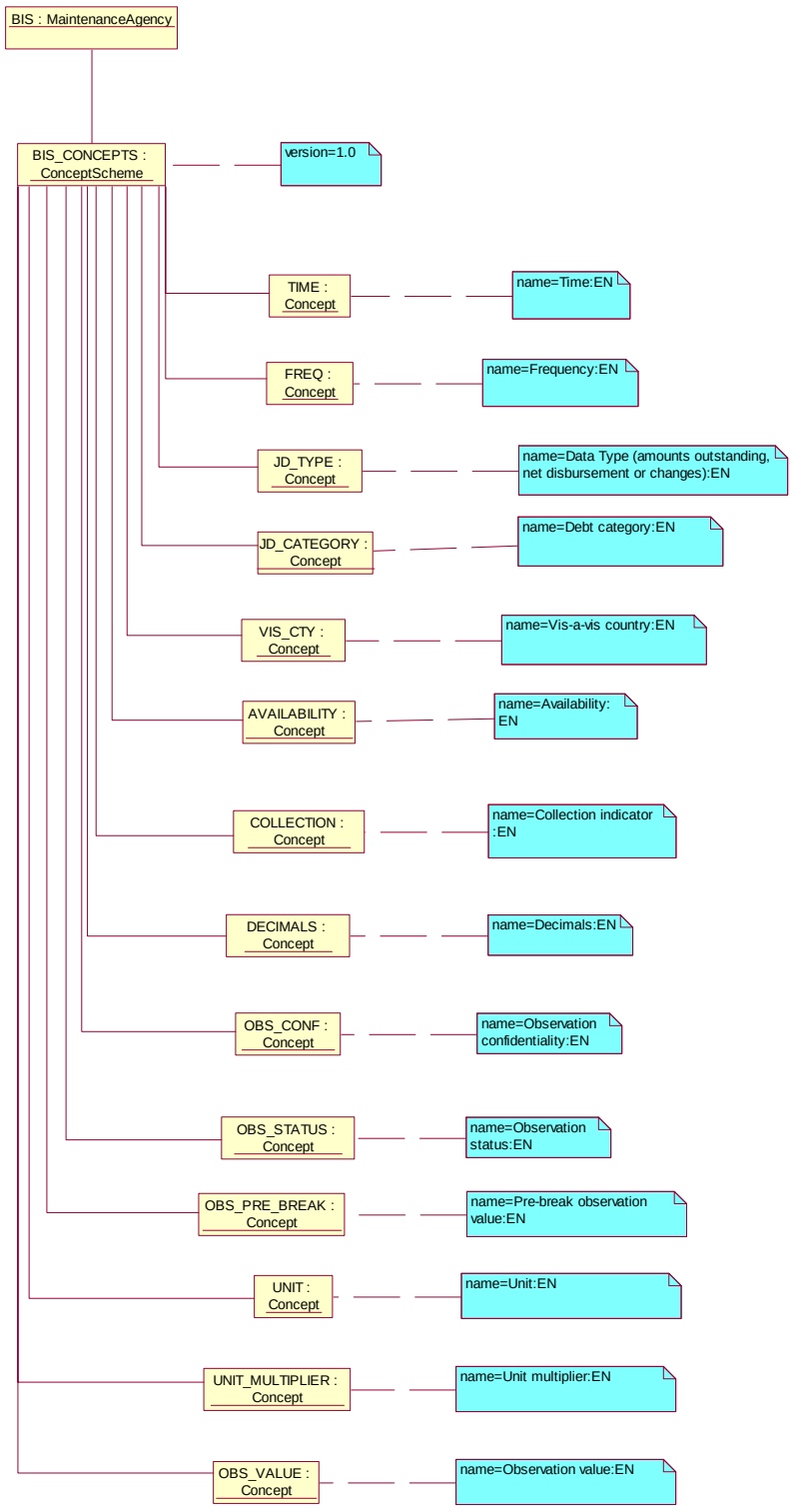
700 # C. Collaboration diagrams

701 ## Domain scheme



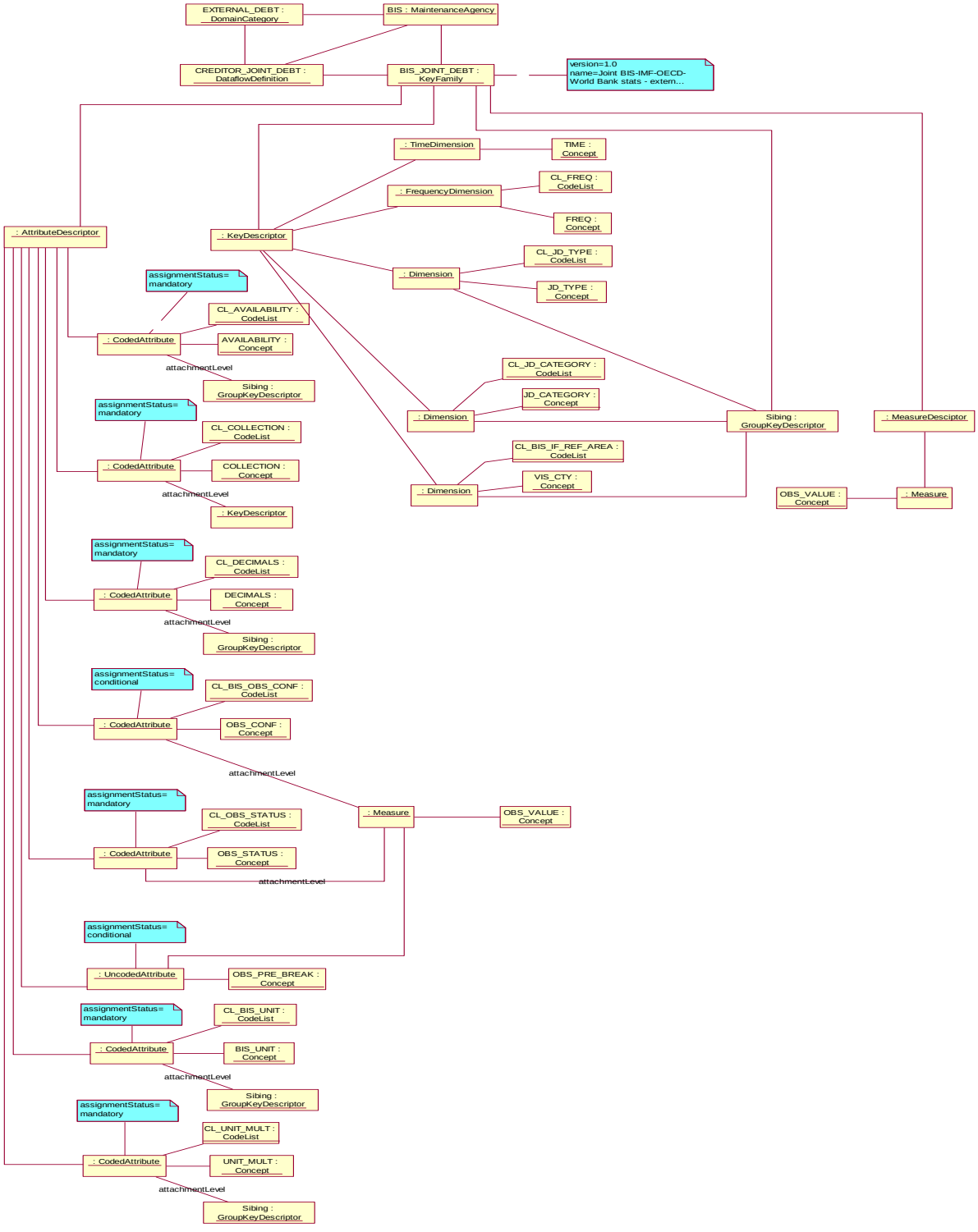
702

703 **Metadata concept scheme**



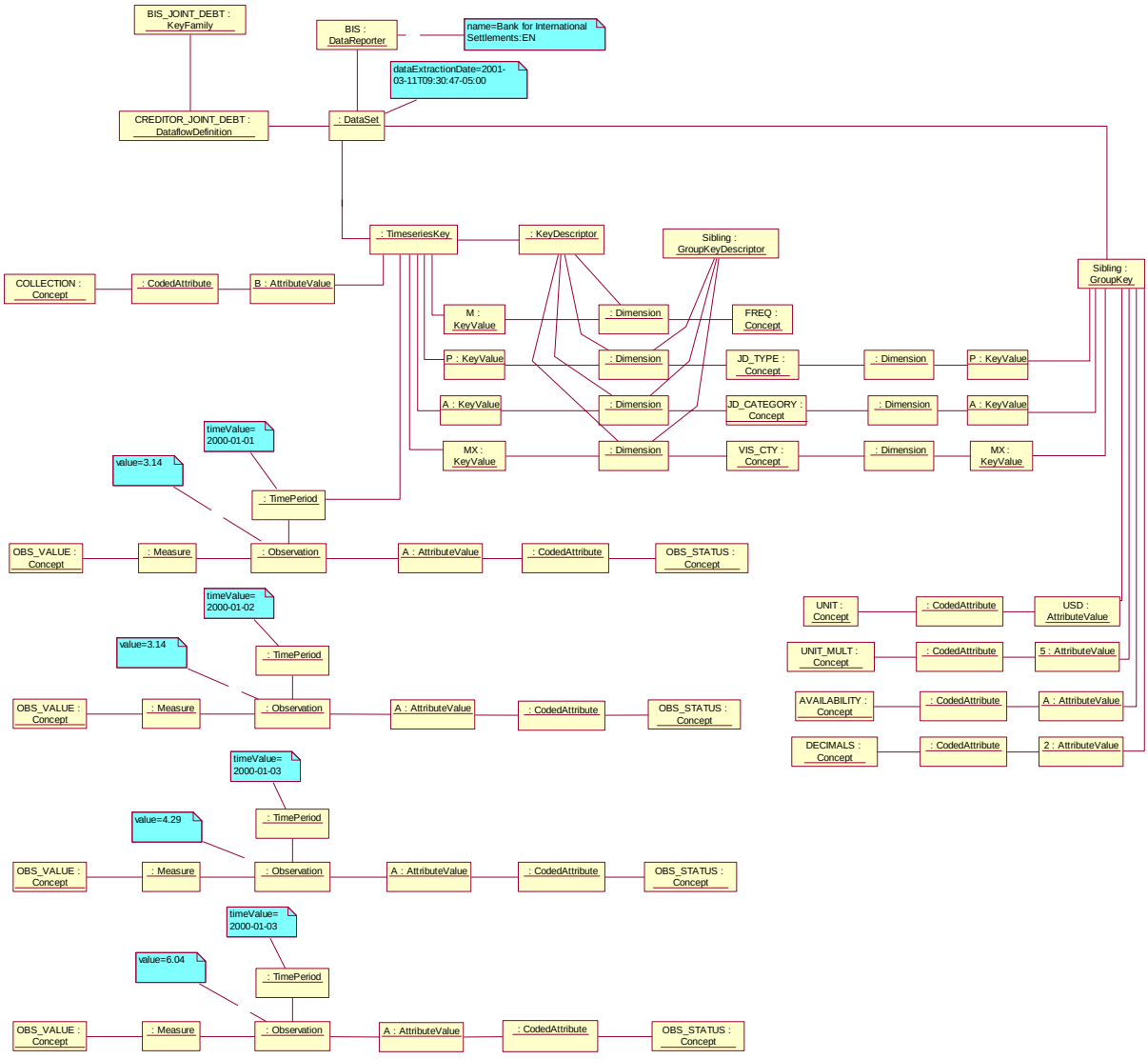
704

705 **Key family**



706

Data set


BIS_JOINT_DEBT : KeyFamily
BIS : DataReporter
name=Bank for International Settlements:EN
dataExtractionDate=2001-03-11T09:30:47-05:00
CREDITOR_JOINT_DEBT : DataflowDefinition
: DataSet
: TimeseriesKey
: KeyDescriptor
Sibling : GroupKeyDescriptor
Sibling : GroupKey
COLLECTION : Concept
: CodedAttribute
B : AttributeValue
M : KeyValue
: Dimension
FREQ : Concept
P : KeyValue
JD_TYPE : Concept
A : KeyValue
JD_CATEGORY : Concept
MX : KeyValue
VIS_CTY : Concept
timeValue= 2000-01-01
value=3.14
: TimePeriod
OBS_VALUE : Concept
: Measure
: Observation
A : AttributeValue
OBS_STATUS : Concept
timeValue= 2000-01-02
value=3.14
timeValue= 2000-01-03
value=4.29
timeValue= 2000-01-03
value=6.04
UNIT : Concept
USD : AttributeValue
UNIT_MULT : Concept
5 : AttributeValue
AVAILABILITY : Concept
DECIMALS : Concept
2 : AttributeValue

VI. APPENDIX I: A SHORT GUIDE TO UML IN THE SDMX INFORMATION MODEL

A. Scope

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

B. Use cases

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

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

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

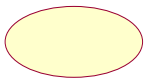
The actor is depicted as a stick man as shown below.

Data Reporter

Figure 20: Actor

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

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



Report Data

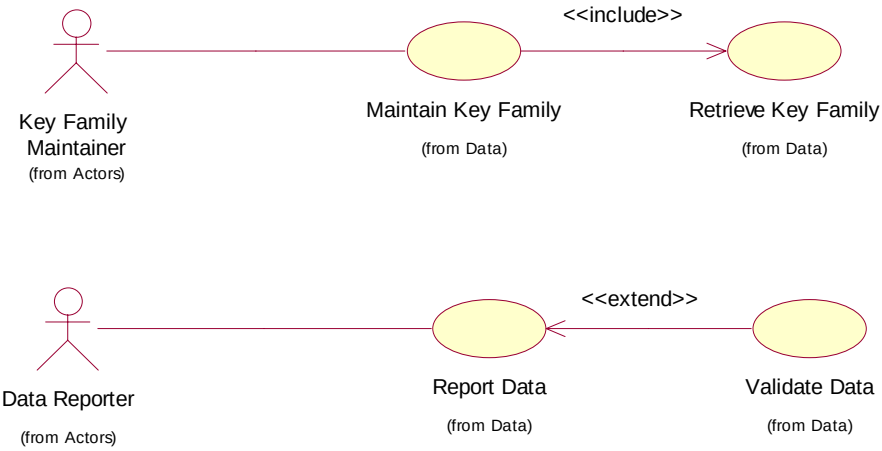
Figure 21: Use case

Actors and use cases are joined by an association.

738

739 **Figure 22: Actor and use case**

740 Some use cases can be invoked directly by other use cases, either as an extension
741 (optional), or as an inclusion (always occurs). These are documented as <<name>>
742 on the use case association.



743

744 **Figure 23: Extend and include use cases**

745 ## C. Classes and attributes

746 **General**

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

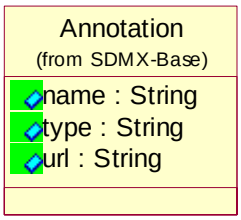


Figure 24: Class and its attributes

Figure 5 shows that a class is represented by a rectangle split into three compartments. The top compartment is for the class name, the second is for attributes and the last is for operations. Only the first compartment is mandatory. The name of the class is `Annotation`, and it belongs to the package SDMX-Base. It is common to group related artefacts (classes, use-cases, etc.) together in packages. . `Annotation` has three "String" attributes – `name`, `type`, and `url`. The full identity of the attribute includes its class e.g. the name attribute is `Annotation.name`.

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

Abstract class

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

Figure 25: Abstract and concrete classes

D. Associations

General

In an E-R model these are known as relationships. A UML model can give more meaning to the associations than can be given in an E-R relationship. Furthermore,

the UML notation is fixed (i.e. there is no variation in the way associations are drawn). In an E-R diagram, there are many diagramming techniques, and it is the relationship in an E-R diagram that has many forms, depending on the particular E-R notation used.

Simple association

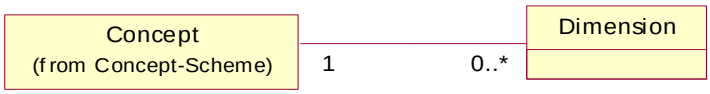


Figure 26: A simple association

Here the `Dimension` class has an association with the `Concept` class. The diagram shows that a `Dimension` can have an association with only one `Concept` (1) and that a `Concept` can be linked to many `Dimensions` (0..*). The association is sometimes named to give more semantics.

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

- Zero or one (0..1)
- Zero or many (0..*)
- One or many (1..*)
- Many (*)
- Unspecified (blank)

Aggregation

Simple Aggregation

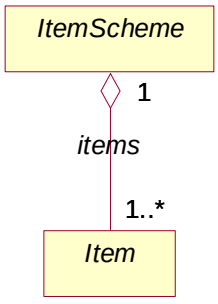


Figure 27: A simple aggregate association

An association with an aggregation relationship indicates that one class is a subordinate class (or a part) of another class. In an aggregation relationship, the

child class instance can outlive its parent class. To represent an aggregation relationship, draw a solid line from the parent class to the subordinate class, and draw an unfilled diamond shape on the parent class's association end. Figure 8 shows an example of an aggregation relationship between an `ItemScheme` and an `Item`.

Composition aggregation

The composition aggregation relationship is just another form of the aggregation relationship, but the child class's instance lifecycle is dependent on the parent class's instance lifecycle. In Figure 9, which shows a composition relationship between a `ComponentStructure` class and a `ComponentList` class, notice that the composition relationship is drawn like the aggregation relationship, but this time the diamond shape is filled.

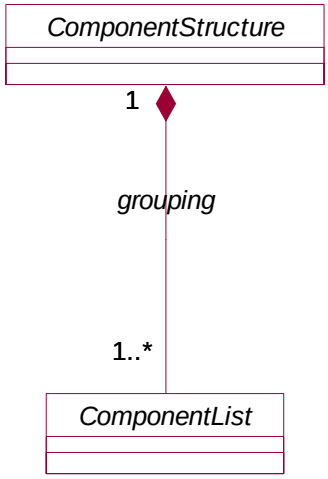
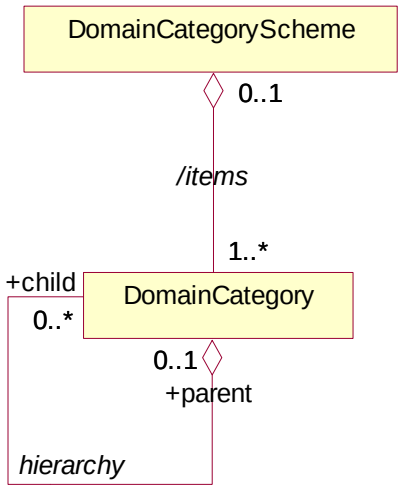


Figure 28: An aggregate association by composition

Association names and association-end (role) names

It can be useful to name associations as this gives some more semantic meaning to the model i.e. the purpose of the association. It is possible for two classes to be joined by two (or more) associations, and in this case it is extremely useful to name the purpose of the association. Figure 10 shows a simple aggregation between `DomainCategoryScheme` and `DomainCategory` called *items*, and another between `DomainCategory` called *hierarchy*.



822

823 **Figure 29: Association names and end names**

824 Furthermore, it is possible to give role names to the association-ends to give more
825 semantic meaning – such as parent and child in a tree structure association.

826 **Navigability**

827 Associations are navigable in both directions. For a data model it is not necessary to
828 give any more semantic than this. However, if there is an intent to implement the
829 model in a database or message structure, it can be useful to identify when the
830 association is not navigable (i.e. there is no intention or necessity to implement a
831 navigation in a particular direction).

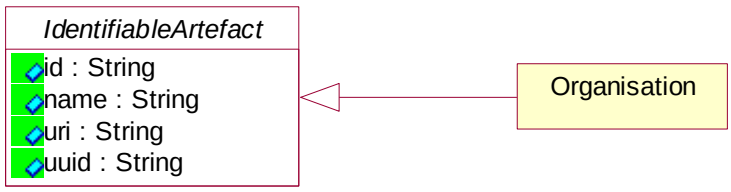
832

833 **Figure 30: One way association**

834 Here it is possible to navigate from A to B, but there is no need (e.g. no functional
835 need) to navigate from B to A using this association.

836 **Inheritance**

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



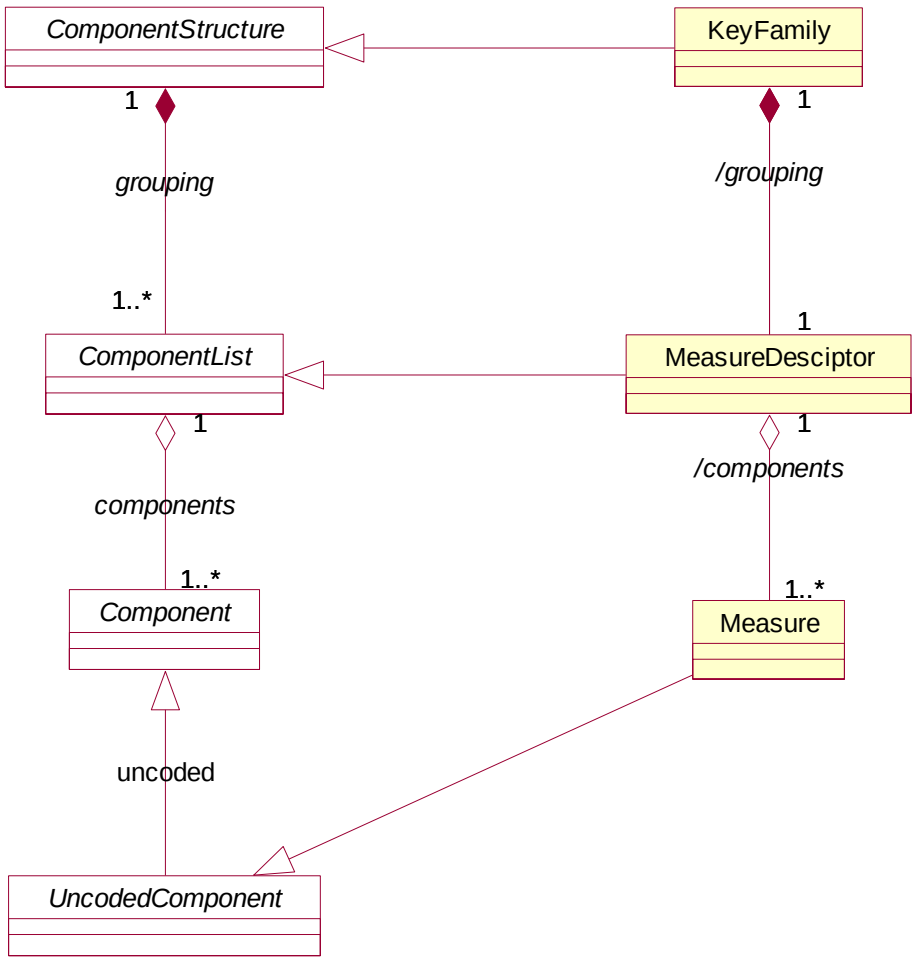
841

842 **Figure 31: Inheritance**

843 Here the Organisation is derived from *IdentifiableArtefact*, which is an
844 abstract superclass. This class inherits the attributes and associations of the super
845 class. Such a super class can be a concrete class (i.e. it actually exists), or an
846 abstract class.

847 **Derived association**

848 It is often useful in a relationship diagram to show associations between sub classes
849 that are derived from the associations of the super classes from which the sub
850 classes inherit. A derived association is shown by "/" preceding the association name
851 e.g. */name*.



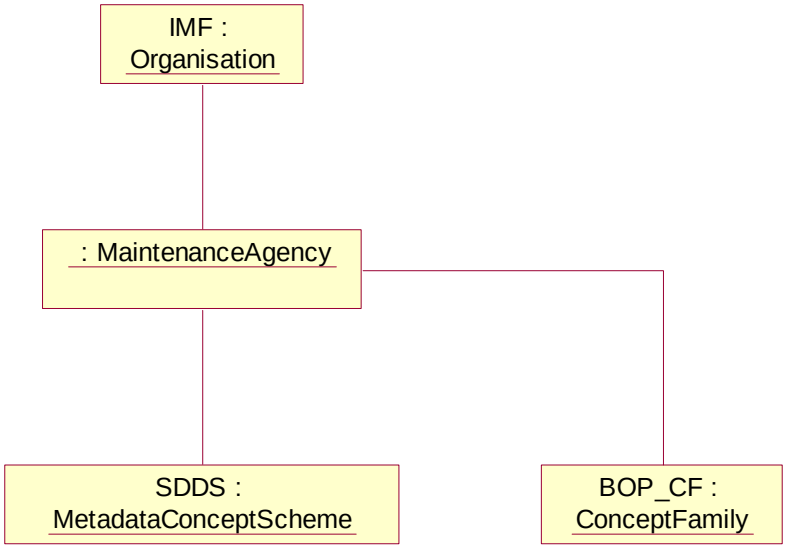
852

853 **Figure 32: Derived associations**

854 Note that the multiplicity at the association ends can be made more restrictive in the
855 derived association. In the example above the `grouping` association is 1..* whereas
856 the `/grouping` association is 1.

857 ### E. Collaboration diagram

858 A collaboration diagram shows an example of an instance of the classes (an instance
859 of a class is called an object). An instance of a class is class with a unique name.



860

861 **Figure 33: Collaboration diagram**

862 Here there is an object of the `Organisation` class called IMF. In its role as
863 `MaintenanceAgency` the IMF maintains a `MetadataConceptScheme` called
864 SDDS and `ConceptFamily` called BOP_CF.
865 Sometimes it is not useful to give a name to an object. Here the object is still an
866 instance of the class (e.g. `MaintenanceAgency`) but there is no name – so it
867 means “any” or “it does not matter which name”.
868 Objects are joined together using an object link.
869
870

871

VII. APPENDIX II: KEY FAMILIES - A TUTORIAL

A. Introduction

This document is intended to explain "key families" to those who are completely unfamiliar with the concept. Key families are an important part of the SDMX family of standards for exchanging statistical data, and they are modelled and explained in much greater detail in other documents. However, those documents are not written to explain the basics, and will make difficult reading for those new to the idea. This document provides a basic tutorial, to help provide the basic level of understanding needed to make sense of the SDMX standards.

B. What is a Key Family?

In order to answer this question, we need to look at statistical data. Statistical data is represented with numbers, such as:

17369

If you are presented with a number - as above - you will have no idea of what it actually represents. You know that it is a piece of statistical data, and therefore is a measurement of some phenomenon - also known as an "observation" - but you can't tell from the number alone what it is a measurement of. A number of questions come immediately to mind:

- What is the subject of the measurement?
- What units does it measure in?
- What country or geographical region, if any, does it apply to?
- When was the measurement made?

The list of questions is potentially endless. Behind each of these questions is a particular idea, or "concept", which is used to describe the data. In our questions above, these descriptor concepts are Subject, Unit of measure, Country, and Time. If I tell you the answers to these questions, the data will begin to make sense:

- the Subject is "total population"
- the Unit of measure is "thousands of people"
- the Country is "Country ABC"
- the Time is "1 January 2001"

This is a simplified and fictional example, but it does demonstrate how we can begin to make sense of statistical data with a set of descriptor concepts. We now know that our number represents the fact that the total population of Country ABC on 1 January, 2001, was 17,369,000.

The simplest explanation of a key family is that it is a set of descriptor concepts, associated with a set of data, which allow us to understand what that data means. There is more to it, however.

C. Grouping Data

Numbers are often grouped together in various ways, to serve as useful packages of information. One very common approach is to have a set of observations - known as a "series", or a "time series" - made over time. This allows us to see trends in the phenomenon being measured. Thus, if I measure the total population in Country ABC on 1 January of every year, I can see whether the population is growing or declining.

A time series always has a "frequency". This is a descriptor concept which describes the intervals of time between observations. Usually, this is a regular interval, so that the frequency can be expressed as "annual" or "monthly" or "weekly". Sometimes, the intervals are irregular. Notice that a single observation does not have a frequency - only series of observations have frequencies. Frequency is an example of a descriptor concept which only applies to series of data.

There are other, higher-level groupings of data as well. A number of series are often grouped together into a "Group". Traditionally, the Group was known as a "Sibling Group", and it contained a set of Series which were identical except that they were measured with different frequencies. Thus, a given phenomenon would be measured as daily, monthly, and annually, and these Series, taken together, would be a "Sibling Group".

It is possible to have Groups which have variable values for descriptor concepts other than frequency, however: if I want to express the US daily exchange rate for all of the world's currencies over the past year, I have a different kind of group. All of the "frequency" descriptors would be the same - "daily" - but the descriptor concept which gives the "foreign currency" would be different for each series.

There is also a higher level of package known as a "Data Set". This represents a set of data that may be made up of several Groups. Typically, it is maintained and published by an agency, so that it becomes a known source of statistical data.

A basic structure is emerging: We have Observations, grouped into Series, which are grouped into Groups, which are grouped into Data Sets.

Note: It should be mentioned that there is another way of packaging Observations, which we call "cross-sectional" data. In cross-sectional data, a large number of related Observations are presented for a single point or period in time. This organization of data is very similar to Time Series data in the way a set of descriptor concepts can be associated with it. A Key Family can be used to describe both cross-sectional and time series data. For the purposes of this part of the tutorial, however, we will focus on time series data. Once we have described the Key Family for time series data, we will go back and see how cross-sectional data are structured.

What is a key family? (Answer #1)

A key family is a way of associating a set of descriptor concepts with a specific set of statistical data, as well as a technique for packaging or structuring that set of data into groups and sub-groups. This is only one way of understanding the structure and meaning of statistical data, but it provides us with a solid, generic model.

D. Attachment Levels

Some descriptor concepts are not meaningful at the Observation level, but only at a higher level. The example we saw earlier was frequency, which means nothing for a single Observation, but has meaning when applied to a Series of Observations. This is because it represents the interval of time between Observations. Time, on the other hand, is meaningful at the Observation level - every Observation is associated with a specific point or period in Time. Key families provide information about the level at which a particular descriptor concept is attached: at the Observation level, the Series level, the Group level, or the Data Set level. This is known as the "attachment level" of the descriptor concept.

If we think about Groups, particularly, we can see how this works. Within a group, some descriptor concepts have values that are the same for all Series within the Group, while other descriptor concepts are changeable. For the Group described above, of all US exchange rates measured daily for all of the world's currencies, the descriptor concepts of Subject ("US exchange rate") and Frequency ("daily") will be

the same for all members of the Group. The descriptor concept "Foreign Currency", however, will change for each Series within the group: there will be a Series for "Swiss Francs," a Series for the "Euro," a Series for "New Zealand dollars," etc.

The rule is that descriptor concepts are “attached” to the grouping level where they become variable. Thus, if, within a single set of data, all the contents of a Series share a single value for a descriptor concept, then that descriptor concept should be attached at the Series level. This rule also assumes that the chosen level is the highest structural level where all sub-groups will share the same value. (While it is true that all Series in a Group where the country is “Switzerland” share a single value, if every Group in the Data Set would always also have the value “Switzerland” for country, then the attachment level should be the Data Set, not the Group.)

Attachment levels of descriptor concepts are always at least at the level where the concept is meaningful: thus, you cannot attach the descriptor concept frequency at the Observation level, because as a concept it only operates at the level of Series (that is, with multiple Observations made over time).

E. Keys

A "key family" is so called because of the term "key". "Key" refers to the values for the descriptor concepts which describe and *identify* a particular set of data. Let's take a simple example:

I have a set of statistical data which uses the following descriptor concepts:

- Time
- Frequency
- Topic
- Country

Time is always attached at the Observation level - the value for Time is the time at which the Observation was made. Time - because it is a concept connected to all statistical data - does not form part of the key. The other descriptor concepts - frequency, topic, and country - are all attached at the series level. For any given Series of Observations, they will all have a single value.

If we have a Series of data which is the monthly measurement of the total population of Country ABC, we will have a key made up of the following values for each descriptor concept:

Frequency = "monthly"

Topic = "total population"

Country = "Country ABC"

This set of values - "Monthly - total population - Country ABC" is the "key" for this data Series: it identifies what the data is.

Keys are most often associated with data at the Series level, but they also exist at other levels. For example, we could enlarge our example to be a Group including the monthly total population data for all of the countries in the world. At the Group level, Frequency would have a value of "monthly", and Topic would have a value of "total population", but we would not specify the Country descriptor concept, because it would change from Series to Series. The key for the Group is known as a "Group Key" - it identifies what the Group is, rather than identifying the Series. (In order to completely understand the Group, of course, we also need to know which descriptor concepts are changeable - in this case, Country.)

The key values are attached at the Series level, and are given in a fixed sequence. Frequency is the first descriptor concept, and the other concepts are assigned an order for that particular data set. This makes it much easier to share and understand statistical data.

If you look back to our initial use of this example, you will notice that we have not been discussing the "Unit of measure" descriptor concept. This is because the "key" only contains values for those descriptor concepts which identify the data. If we have the measurements made in thousands or in millions, the data are the same - they can be derived from one another by simply multiplying the numbers in the data by the appropriate conversion factor.

This points out a major distinction between the two types of descriptor concepts: the ones which both identify and describe the data are called "dimensions", and those which are purely descriptive are called "attributes". Only "dimensions" - that is, the descriptor concepts which also identify the data - are used in the "key", because the "key" is fundamentally a way of identifying a set of data.

F. Code Lists and Other Representations

In order to be able to exchange and understand data, a key family tells us what the possible values for each dimension are. This list of possible values is known as a "code list." Each value on that list is given a language-independent abbreviation - a "code" - and a language-specific description. This helps us avoid problems of

translation in describing our data: the code can be translated into descriptions in any language without having to change the code associated with the data itself. Wherever possible, the values for code lists are taken from international standards, such as those provided by ISO for countries and currencies.

As stated, dimensions are always represented with codes. Attributes are sometimes represented with codes, but sometimes represented by numeric or free-text values. This is allowed because the attributes do not serve an identification function, but merely describe the data.

What is a key family? (Answer #2)

We now have a more sophisticated understanding of a what a key family does: it specifies a set of concepts which describe and identify a set of data. It tells us which concepts are dimensions (identification and description), and which are attributes (just description), and it gives us an attachment level for each of these concepts, based on the packaging structure (Data Set, Group, Series, Observation). It also tells us which code lists provide possible values for the dimensions, and gives us the possible values for the attributes, either as code lists or as numeric or free text fields.

G. Cross-Sectional Data Structures

Given the explanation of Key Families thus far, we understand that a Key Family associates descriptor concepts with data, some of which also serve to identify the data – the “dimension” concepts which make up the Key.

Cross-sectional data structures do not apply a different set of concepts to the data: the same concepts still apply in describing and identifying the data. It attaches the concepts to the data differently, to create a different presentation of the data.

If we go back to our earlier example, we had the following concepts:

- Time
- Frequency
- Topic
- Country

If we want to take a set of data which is described and identified by this set of concepts, and present it in a cross-sectional fashion, we would not change these

concepts – we would merely change the way in which they are represented – that is, attached – to the data structure.

Take, as an example, the total population of each country in the world on January 1, 2001 as a set of data. In our earlier example, we measured the population of Country ABC over a period of years – that is, over time. *Time* was the concept we used to organize our data in a sequence of observations.

If we organize our data to reflect only a single point in time – in this case, January 1, 2001 – then organizing our data over time makes less sense. It is still a possible way to structure the data, but we may wish to view it as a cross-section.

Think about the term “cross-section” – it can be understood to mean a group of parallel series over time, from which a *section* is taken, *across* time. Thus, a cross-section is created.

In our example, it is easy to see how this applies: instead of organizing our data over time – that is, using the time concept - we are choosing to organize it over the Country concept. Thus, instead of having a single value for Frequency, Topic, and Country for all Observations in our series, with a Time value associated with each Observation, we will have a Country value associated with each Observation, and a single value for Frequency, Topic, and Time. Instead of calling the group of Observations a “Series”, we now use the term “Section”.

In our earlier example, we had a key which existed mostly at the Series level:

Frequency = "monthly"

Topic = "total population"

Country = "Country ABC"

Time – our remaining concept, was associated with the Observations, with a different value for each one. Thus, we could have a Series which looks like this:

January 1, 2001 – 17369

February 1, 2001 – 17370

March 1, 2001 – 17405

For our cross-sectional presentation, we would have most of our key at the Section level (or, potentially, at a higher level of grouping):

Frequency = "monthly"

Topic = "total population"

Time = "January 1, 2001"

With each Observation, we now have a Country value, instead of a Time value:

Country ABC = “17369”

Country XYZ = “24982”

Country HIJ = “37260”

In this cross-sectional presentation of our data set, we have chosen to present each Observation paired with a Country value, taken from our Codelist of values for the concept Country. Other dimensions could as easily produce a cross-sectional view, by attaching their values at The Observation level, instead of the values for Country, as in our example.

Because the concepts themselves do not change, but only the way in which they are attached to the data structure, a single key family can be used to describe both time-series and cross-sectional presentations.

In the version 1.0 SDMX standards, formats are capable of presenting cross-sectional data for any single dimension concept, as well as presenting the data as a time series. It is up to the key family creator to select which non-Time concept, used as a dimension, will serve to organize a cross-sectional presentation. In future versions, it is possible that more complete support for the possible cross-sectional views for a key family will be provided.