sdmx-im Documentation

Release 0.0.1

sdmx-twg

Nov 05, 2020

TECHNICAL SPECIFICATION

CHAPTER

ONE

INTRODUCTION

We are in the process of cleaning up, unifying and simplifying the repositories associated to the SDMX standard's formats, documentation, technical references, guidelines and examples. This is done in the context of the forthcomming version 3.0.0 of the SDMX standard …

During this process we intend to regroup and simplify the access to the documentation of the standard.

Please bare with us whilst this is being executed.

Progress will be visible in the `develop` and other branches of this repository.

1.1 Framework for SDMX Technical Standards

1.1.1 Introduction

The Statistical Data and Metadata Exchange (SDMX) initiative (http://www.sdmx.org) sets standards that can facilitate the exchange of statistical data and metadata using modern information technology, with an emphasis on aggregated data.

There are several sections to the SDMX Technical Specification:

1. SDMX Framework Document – this document. The purpose of this document is to introduce SDMX and its scope. This document will be revised in due course to include the conformance requirements.
2. The SDMX Information Model - the information model on which syntax-specific implementations described in the other sections are based. This is intended for technicians wishing to understand the complete scope of the technical standards in a syntax-neutral form. It includes as an annex a tutorial on UML (Unified Modelling Language). This document is not normative.
3. SDMX-EDI - the UN/EDIFACT format for exchange of SDMX-structured data and metadata. This document contains normative sections describing the use of the UN/EDIFACT syntax in SDMX messages. This document has normative sections.
4. SDMX-ML - the XML format for the exchange of SDMX-structured data and metadata. This document has normative sections describing the use of the XML syntax in SDMX messages, and is accompanied by a set of normative XML schemas and non-normative sample XML document instances.
5. The SDMX Registry Specification provides for a central registry of information about available data and reference metadata, and for a repository containing structural metadata and provisioning information. This specification defines the basic services offered by the SDMX Registry: registration of data and metadata; querying for data and metadata; and subscription/notification regarding updates to the registry. This document has normative sections.
6. The SDMX Technical Notes – this is a guide to help those who wish to use the SDMX specifications. It includes notes on the expressive differences of the various messages and syntaxes; versioning; maintenance agencies; the SDMX Registry. This document is not normative.
7. Web Services Guidelines – this is a guide for those who wish to implement SDMX using web-services technologies. It places an emphasis on those aspects of web-services technologies (including, but not requiring,

an SDMX-conformant registry) which will work regardless of the development environment or platform used to create the web services. This document contains normative sections.

1.1.2 Changes from Previous Version

The 2.0 version of this standard represented a significant increase in scope, and also provided more complete support in those areas covered in the version 1.0 specification. Version 2.0 of this standard is backward-compatible with version 1.0, so that existing implementations can be easily migrated to conformance with version 2.0.

The 2.1 version of this standard represents a set of changes resulting from several years of implementation experience with the 2.0 standard. The changes do not represent a major increase in scope or functionality, but do correct some bugs, and add functionalities in some cases. Major changes in SDMX-ML include a much stronger alignment of the XML Schemas with the Information Model, to emphasize inheritance and object-oriented features, and increased precision and flexibility in the attachment of metadata reports to specific objects in the SDMX Information Model.

> Note that the idea of backward-compatibility in the standards is based on the information model. In both releases, some non-backward-compatible changes have been made to the SDMX-ML formats. The same set of information required to use version 1.0 of the specification will permit the use of the same features in the version 2.0 specifications, however. Thus, a Data Structure Definition is easily translated from version 1.0 to version 2.0, without requiring any new information regarding structures, etc. There have been no changes to the SDMX-EDI format.

The major changes from 1.0 to 2.0 can be briefly summarized:

- **Reference Metadata**: In addition to describing and specifying data structures and formats (along with related structural metadata), the version 2.0 specification also provides for the exchange of metadata which is distinct from the structural metadata in the 1.0 version. This category includes "reference" metadata (regarding data quality, methodology, and similar types – it can be configured by the user to include whatever concepts require reporting); metadata related to data provisioning (release calendar information, description of the data and metadata provided, etc.); and metadata relevant to the exchange of categorization schemes.
- **SDMX Registry**: Provision is made in the 2.0 standard for standard communication with registry services, to support a data-sharing model of statistical exchange. These services include registration of data and metadata, querying of registered data and metadata, and subscription/notification.
- **Structural Metadata**: The support for exchange of statistical data and related structural metadata has been expanded. Some support is provided for qualitative data; data cube structures are described; hierarchical code lists are supported; relationships between data structures can be expressed, providing support for extensibility of data structures; and the description of functional dependencies within cubes are supported.

The major changes from 2.0 to 2.1 can be briefly summarized:

- **Web-Services-Oriented Changes:** Several organizations have been implementing web services applications using SDMX, and these implementations have resulted in several changes to the specifications. Because the nature of SDMX web services could not be anticipated at the time of the original drafting of the specifications, the web services guidelines have been completely re-developed.
- **Presentational Changes:** Much work has gone into using various technologies for the visualization of SDMX data and metadata, and some changes have been proposed as a result, to better leverage this graphical visualization. These changes are largely to leverage the Cross-domain Concepts of the Content Oriented Guidelines.
- **Consistency Issues:** There have been some areas where the draft specifications were inconsistent in minor ways, and these have been addressed.
- **Clarifications in Documentation:** In some cases it has been identified that the documentation of specific fields within the standard needed clarification and elaboration, and these issues have been addressed.
- **Optimization for XML Technologies:** Implementation has shown that it is possible to better organize the XML schemas for use within common technology development tools which work with XML. These changes are primarily focused on leveraging the object-oriented features of W3C XML Schema to allow for easier processing of SDMX data and metadata.

- **Consistency between the SDMX-ML and the SDMX Information Model:** Certain aspects of the XML schemas and UML model have been more closely aligned, to allow for easier comprehension of the SDMX model.
- **Technical Bugs:** Some minor technical bugs have been identified in the registry interfaces and elsewhere. These bugs have been addressed.
- **Support for Non-Time-Series Data in the Generic Format:** One area which has been extended is the ability to express non-time-series data as part of the generic data message.
- **Simplification of the data structure definition - specific message types:** Both time series (version 2.0 Compact) and non-time series data sets (version 2.0 Cross Sectional) use the same underlying structure for a structure-specific formatted message, which is specific to the Data Structure Definition of the data set.
- **Simplification and better support for the metadata structure:** New use cases have been reported and these are now supported by a re-modelled metadata structure definition.
- **Support for partial item schemes such as a code list:** The concept of a partial (sub-set) item scheme such as a partial code list for use in exchange scenarios has been introduced**.**

1.1.3 Processes and Business Scope

Process Patterns

SDMX identifies three basic process patterns regarding the exchange of statistical data and metadata. These can be described as follows:

1. *Bilateral exchange:* All aspects of the exchange process are agreed between counterparties, including the mechanism for exchange of data and metadata, the formats, the frequency or schedule, and the mode used for communications regarding the exchange. This is perhaps the most common process pattern.
2. *Gateway exchange:* Gateway exchanges are an organized set of bilateral exchanges, in which several data and metadata collecting organizations or individuals agree to exchange the collected information with each other in a single, known format, and according to a single, known process. This pattern has the effect of reducing the burden of managing multiple bilateral exchanges (in data and metadata collection) across the sharing organizations/individuals. This is also a very common process pattern in the statistical area, where communities of institutions agree on ways to gain efficiencies within the scope of their collective responsibilities.
3. *Data-sharing exchange:* Open, freely available data formats and process patterns are known and standard. Thus, any organization or individual can use any counterparty's data and metadata (assuming they are permitted access to it). This model requires no bilateral agreement, but only requires that data and metadata providers and consumers adhere to the standards.

This document specifies the SDMX standards designed to facilitate exchanges based on any of these process patterns, and shows how SDMX offers advantages in all cases. It is possible to agree bilaterally to use a standard format (such as SDMX-EDI or SDMX-ML); it is possible for data senders in a gateway process to use a standard format for data exchange with each other, or with any data providers who agree to do so; it is possible to agree to use the full set of SDMX standards to support a common data-sharing process of exchange, whether based on an SDMX-conformant registry or some other architecture.

The standards specified here specifically support a data-sharing process based on the use of central registry services. Registry services provide visibility into the data and metadata existing within the community, and support the access and use of this data and metadata by providing a set of triggers for automated processing. The data or metadata itself is not stored in a central registry – these services merely provide a useful set of metadata about the data (and additional metadata) in a known location, so that users/applications can easily locate and obtain whatever data and/or metadata is registered. The use of standards for all data, metadata, and the registry services themselves is ubiquitous, permitting a high level of automation within a data-sharing community.

It should be pointed out that these different process models are not mutually exclusive – a single system capable of expressing data and metadata in SDMX-conformant formats could support all three scenarios. Different standards may be applicable to different processes (for example, many registry services interfaces are used only in a data-sharing scenario) but all have a common basis in a shared information model.

In addition to looking at collection and reporting, it is also important to consider the dissemination of data. Data and metadata – no matter how they are exchanged between counterparties in the process of their development and creation – are all eventually supplied to an end user of some type. Often, this is through specific applications inside of institutions. But more and more frequently, data and metadata are also published on websites in various formats. The dissemination of data and its accompanying metadata on the web is a focus of the SDMX standards. Standards for statistical data and metadata allow improvements in the publication of data – it becomes more easily possible to process a standard format once the data is obtained, and the data and metadata are linked together, making the comprehension and further processing of the data easier.

In discussions of statistical data, there are many aspects of its dissemination which impact data quality: data discovery, ease of use, and timeliness. SDMX standards provide support for all of these aspects of data dissemination. Standard data formats promote ease of use, and provide links to relevant metadata. The concept of registry services means that data and metadata can more easily be discovered. Timeliness is improved throughout the data lifecycle by increases in efficiency, promoted through the availability of metadata and ease of use.

It is important to note that SDMX is primarily focused on the *exchange* and *dissemination* of statistical data and metadata. There may also be many uses for the standard model and formats specified here in the context of internal processing of data that are not concerned with the exchange between organizations and users, however. It is felt that a clear, standard formatting of data and metadata for the purposes of exchange and dissemination can also facilitate internal processing by organizations and users, but this is not the focus of the specification.

SDMX and Process Automation

Statistical data and metadata exchanges employ many different automated processes, but some are of more general interest than others. There are some common information technologies that are nearly ubiquitous within information systems today. SDMX aims to provide standards that are most useful for these automated processes and technologies.

Briefly, these can be described as:

1. *Batch Exchange of Data and Metadata:* The transmission of whole or partial databases between counterparties, including incremental updating.
2. *Provision of Data and Metadata on the Internet:* Internet technology - including its use in private or semi-private TCP/IP networks - is extremely common. This technology includes XML and web services as primary mechanisms for automating data and metadata provision, as well as the more traditional static HTML and database-driven publishing.
3. *Generic Processes:* While many applications and processes are specific to some set of data and metadata, other types of automated services and processes are designed to handle any type of statistical data and metadata whatsoever. This is particularly true in cases where portal sites and data feeds are made available on the Internet.
4. *Presentation and Transformation of Data:* In order to make data and metadata useful to consumers, they must support automated processes that transform them into application-specific processing formats, other standard formats, and presentational formats. Although not strictly an aspect of exchange, this type of automated processing represents a set of requirements that must be supported if the information exchange between counterparties is itself to be supported.

The SDMX standards specified here are designed to support the requirements of all of these automation processes and technologies.

Statistical Data and Metadata

To avoid confusion about which "data" and "metadata" are the intended content of the SDMX formats specified here, a statement of scope is offered. Statistical "data" are sets of often numeric observations which typically have time associated with them. They are associated with a set of metadata values, representing specific concepts, which act as identifiers and descriptors of the data. These metadata values and concepts can be understood as the named dimensions of a multi-dimensional co-ordinate system, describing what is often called a "cube" of data.

SDMX identifies a standard technique for modelling, expressing, and understanding the structure of this multi-dimensional "cube", allowing automated processing of data from a variety of sources. This approach is widely applicable across types of data and attempts to provide the simplest and most easily comprehensible technique that will support the exchange of this broad set of data and related metadata.

The term "metadata" is very broad indeed. A distinction can be made between "structural" metadata – those concepts used in the description and identification of statistical data and metadata – and "reference" metadata – the larger set of concepts that describe and qualify statistical data sets and processing more generally, and which are often associated not with specific observations or series of data, but with entire collections of data or even the institutions which provide that data.

The SDMX Information Model provides for the structuring not only of data, but also of "reference" metadata. While these reference metadata structures exist independent of the data and its structural metadata, they are often linked. The SDMX Information Model provides for the attachment of reference metadata to any part of the data or structural metadata, as well as for the reporting and exchange of the reference metadata and its structural descriptions. This function of the SDMX standards supports many aspects of data quality initiatives, allowing as it does for the exchange of metadata in its broadest sense, of which quality-related metadata is a major part.

Metadata are associated not only with data, but also with the process of providing and managing the flow of data. The SDMX Information Model provides for a set of metadata concerned with "data provisioning" – metadata which are useful to those who need to understand the content and form of a data provider's output. Each data provider can describe in standard fashion the content of and dependencies within the data and metadata sets which they produce, and supply information about the scheduling and mechanism by which their data and metadata are provided. This allows for automation of some validation and control functions, as well as supporting management of data reporting.

SDMX also recognizes the importance of classification schemes in organizing and managing the exchange and dissemination of data and metadata. It is possible to express information about classification schemes and domain categories in SDMX, along with their relationships to data and metadata sets, as well as to categorize other objects in the model.

The SDMX standards offer a common model, a choice of syntax and, for XML, a choice of data formats which support the exchange of any type of statistical data meeting the definition above; several optimized formats are specified based on the specific requirements of each implementation, as described below in the SDMX-ML section.

The formal objects in the information model are presented briefly below, but are also discussed in more detail elsewhere in this specification.

The SDMX View of Statistical Exchange

Version 1.0 of ISO/TS 17369 SDMX covered statistical data sets and the metadata related to the structure of these data sets. This scope was useful in supporting the different models of statistical exchange (bilateral exchange, gateway exchange, and data-sharing) but was not by itself sufficient to support them completely. Versions 2.0 and 2.1 provide a much more complete view of statistical exchange, so that an open data-sharing model can be fully supported, and other models of exchange can be more completely automated. In order to produce technical standards that will support this increased scope, the SDMX Information Model provides a broader set of formal objects which describe the actors, processes, and resources within statistical exchanges.

It is important to understand the set of formal objects not only in a technical sense, but also in terms of what they represent in the real-world exchange of statistical data and metadata.

The first version of SDMX provided for data sets - specific statistical data reported according to a specific structure, for a specific time range - and for data structure definitions - the metadata which describes the structure of statistical data sets. These are important objects in statistical exchanges, and are retained and enhanced in the second version

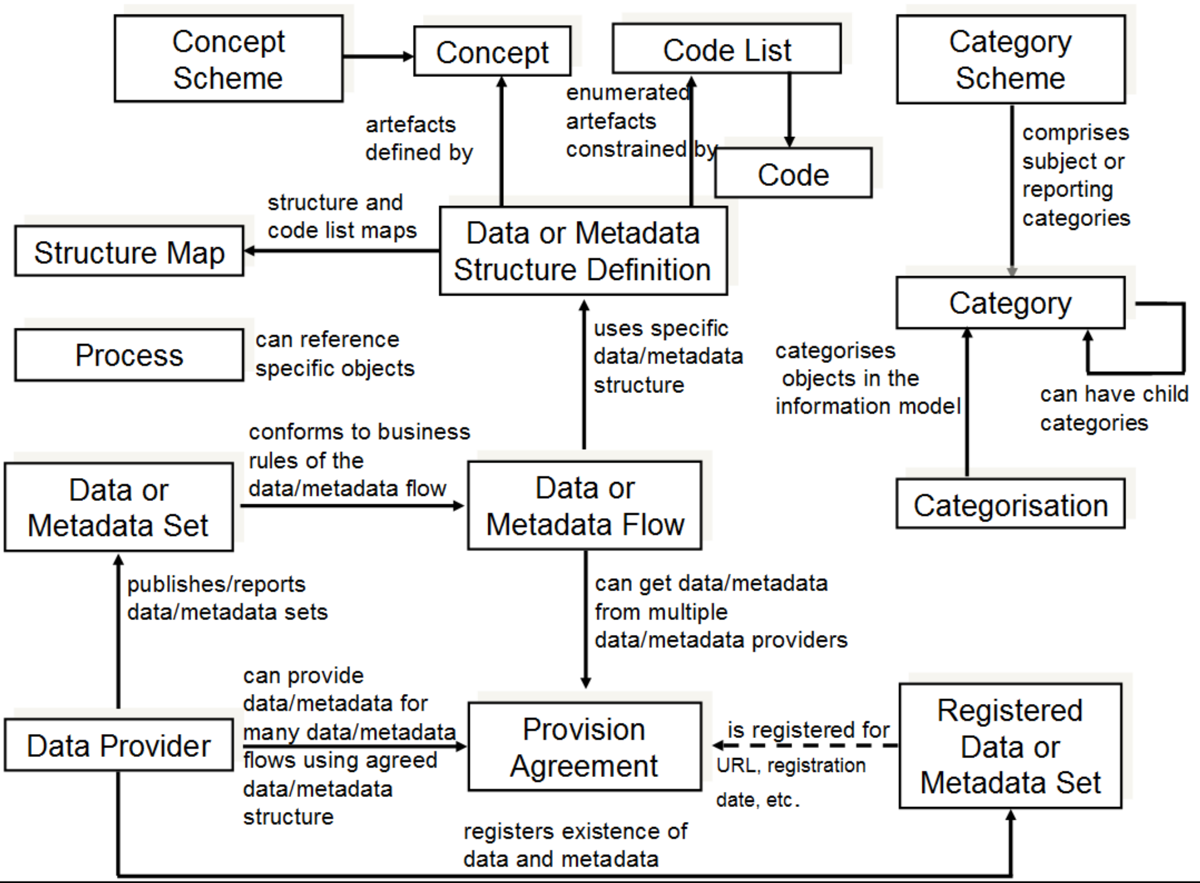

Fig. 1.1: High Level Schematic of Major Artefacts in the SDMX Information Model

of the standards in a backward-compatible form. A related object in statistical exchanges is the "data flow" - this supports the concept of data reporting or dissemination on an ongoing basis. "Data flows" can be understood as data sets which are not bounded by time. Data structures are owned and maintained by agencies - in a similar fashion, data flows are owned by maintenance agencies.

Versions 2.0 and 2.1 – like version 1.0 – allow for the publication of statistical data (and the related structural metadata) but also provide for the standard, systematic representation of reference metadata. Reference metadata are reported not as an integral part of a data set, but independent of the statistical data. SDMX provides for reference "metadata sets", "metadata structure definitions", and "metadata flows". These objects are very similar to data sets, data structure definitions, and data flows, but they concern reference metadata rather than statistical observations. In the same way that data providers may publish statistical data, they may also publish reference metadata. Metadata structural definitions are maintained by agencies in a fashion similar to the way that agencies maintain data structure definitions, the structural definitions of data sets.

The structural definitions of both data and reference metadata associate specific statistical concepts with their representations, whether textual, coded, etc. In SDMX version 2.0/2.1, these concepts are taken from a "concept scheme" which is maintained by a specific agency. Concept schemes group a set of concepts, provide their definitions and names, and allow for semantic relationships to be expressed, when some concepts are specializations of others. It is possible for a single concept scheme to be used both for data structures - key families - and for reference metadata structures.

Inherent in any statistical exchange – and in many dissemination activities - is a concept of "service level agreement", even if this is not formalized or made explicit. SDMX incorporates this idea in objects termed "provision agreements". Data providers may provide data to many different data flows. Data flows may incorporate data coming from more than one data provider. Provision agreements are the objects which tell you which data providers are supplying what data to which data flows. The same is true for metadata flows.

Provision agreements allow for a variety of information to be made available: the schedule by which statistical data or metadata is reported or published, the specific topics about which data or metadata is reported within the theoretically possible set of data (as described by a data structure definition or reference metadata structure definition), and the time period covered by the statistical data and metadata. This set of information is termed "constraint" in the SDMX Information Model.

A brief summary of the objects described in the information model includes:

- **Data Set:** Data is organized into discrete sets, which include particular observations for a specific period of time. A data set can be understood as a collection of similar data, sharing a structure, which covers a fixed period of time.
- **Data Structure Definition (DSD, also known as Key Family in Version 2.0):** Each data set has a set of structural metadata. These descriptions are referred to in SDMX as Data Structure Definitions, which include information about how concepts are associated with the measures, dimensions, and attributes of a data "cube," along with information about the representation of data and related identifying and descriptive (structural) metadata. In Version 2.1, the term "Key Family" is replaced by "Data Structure Definition" (DSD) both in XML Schemas and the Information Model.
- **Code list:** Code lists enumerate a set of values to be used in the representation of dimensions, attributes, and other structural parts of SDMX. They can be supplemented by other structural metadata which indicates how codes are organized into hierarchies.
- **Organisation Scheme:** Organisations and organisation structure can be defined in an Organisation Scheme. Specific Organisation Schemes exist for Maintenance Agency, Data Provider, Data Consumer, and Organisation Unit.
- **Category Scheme and Categorisation:** Category schemes are made up of a hierarchy of categories, which in SDMX may include any type of useful classification for the organization of data and metadata. A Categorisation links a category to an identifiable object. In this way sets of objects can be categorised. A statistical subject-matter domain scheme is implemented in SDMX as a Category Scheme.
- **Concept Scheme:** A concept scheme is a maintained list of concepts that are used in data structure definitions and metadata structure definitions. There can be many such concept schemes. A "core" representation of the concept can be specified (e.g. a core code list, or other representation such as "date"). Note that this core representation can be overridden in the data structure definition or metadata structure definition that

uses the concept. Indeed, organisations wishing to remain with version 1.0 key family schema specifications will continue to declare the representation in the key family definition.

- **Metadata Set:** A reference metadata set is a set of information pertaining to an object within the formal SDMX view of statistical exchange: they may describe the maintainers of data or structural definitions; they may describe the schedule on which data is released; they may describe the flow of a single type of data over time; they may describe the quality of data, etc. In SDMX, the creators of reference metadata may take whatever concepts they are concerned with, or obliged to report, and provide a reference metadata set containing that information.
- **Metadata Structure Definition:** A reference metadata set also has a set of structural metadata which describes how it is organized. This metadata set identifies what reference metadata concepts are being reported, how these concepts relate to each other (typically as hierarchies), what their presentational structure is, how they may be represented (as free text, as coded values, etc.), and with which formal SDMX object types they are associated.
- **Dataflow Definition:** In SDMX, data sets are reported or disseminated according to a data flow definition. The data flow definition identifies the data structure definition and may be associated with one or more subject matter domains via a Categorisation (this facilitates the search for data according to organised category schemes). Constraints, in terms of reporting periodicity or sub set of possible keys that are allowed in a data set, may be attached to the data flow definition.
- **Metadataflow Definition:** A metadata flow definition is very similar to a data flow definition, but describes, categorises, and constrains metadata sets.
- **Data Provider:** An organization which produces data or reference metadata is termed a data provider.
- **Provision Agreement:** The set of information which describes the way in which data sets and metadata sets are provided by a data provider. A provision agreement can be constrained in much the same way as a data or metadata flow definition. Thus, a data provider can express the fact that it provides a particular data flow covering a specific set of countries and topics, Importantly, the actual source of registered data or metadata is attached to the provision agreement (in terms of a URL). The term "agreement" is used because this information can be understood as the basis of a "service-level agreement". In SDMX, however, this is informational metadata to support the technical systems, as opposed to any sort of contractual information (which is outside the scope of a technical specification).
- **Constraint:** Constraints describe a subset of a data source or metadata source, and may also provide information about scheduled releases of data. They are associated with data providers, provision agreements, data flows, metadataflows, data structure definitions and metadata structure definitions.
- **Structure Set:** Structure sets provide a mechanism for grouping structural metadata together to form a complete description of the relationships between specific, related sets of data and metadata. They can be used to map dimensions and attributes to one another, to map concepts, to map code lists, and to map category schemes. They can be used to describe "cubes" of data, even when the data within the cube does not share a single dimensionality.
- **Reporting Taxonomy:** A reporting taxonomy allows an organisation to link (possibly in a hierarchical way) a number of cube or data flow definitions which together form a complete "report" of data or metadata. This supports primary reporting which often comprises multiple cubes of heterogeneous data, but may also support other collection and reporting functions. It also supports the specification of publications such as a yearbook, in terms of the data or metadata contained in the publication.
- **Process:** The process class provides a way to model statistical processes as a set of interconnected *process steps*. Although not central to the exchange and dissemination of statistical data and metadata, having a shared description of processing allows for the interoperable exchange and dissemination of reference metadata sets which describe processes-related concepts.
- **Hierarchical Code List:** This supports the specification of code hierarchies. The codes themselves are referenced from the code lists in which they are maintained. The Hierarchical Code List thus specifies the organisation of the codes in one or more hierarchies, but does not define the codes themselves.

Notes on Data Structuring

A "cube" is a rich, multi-dimensional construct, which can be viewed along any of its axes (or "dimensions"). Whilst the full structure of cube data can be described in SDMX, the actual "data" specification of SDMX takes a slightly narrower view of these requirements in its version 2.0/2.1 specifications for the purposes of formatting the data for transmission. The view of data in many SDMX formats is primarily as time series – that is, as a set of observations which are organized around the time dimension, so that each observation occurs progressively through time.

There are, however, many types of statistical data which are not typically organized for exchange as time series where data are organized around some other, non-time dimension of the cube – what is often called "cross-sectional" data. SDMX supports a unified format that represents in the data set an organisation of the data along any single dimension. In this context, time series is a particular case of the unified format.

Another type of structure commonly found in statistical "cubes" of data is the hierarchical classification, used to describe the points along any of its dimensions (or axes). In the 1.0 version, SDMX standards did not provide full support for this functionality. The introduction of these hierarchical classifications is present in the current version of the standard.

Further, there is support for the expression of functional dependencies between the various dimensions of a cube, giving support for better processing of "sparse cubes". This is an aspect of "constraints", which allow for the framing of a cube region, or for the provision of a set of valid keys within the total set of keys described by the data structure definition.

Notes on Reference Metadata Structuring

Metadata structures are based on the idea that concepts can be organised into semantic and presentational hierarchies, and that these hierarchies can form the basis for the structuring of XML reporting formats. There are three message types in SDMX-ML which serve this purpose: the Structure message (providing the metadata structure definition), the Generic Metadata message (providing a single format for any metadata structure definition), and the Structure-specific Metadata message (providing a metadata structure definition-specific format). Typically, this mechanism is suited to supporting reference metadata reporting and dissemination.

The Metadata Structure Definition takes *any* concept from concept schemes, and describes how they can be formed into a reporting or dissemination structure as metadata attributes – either as a flat list, or as a hierarchy. The metadata attributes are assigned representations (coded, textual, etc.) and the number of occurrences. The "target" of the metadata – that is, the class of process, information, organisation, exchange, etc. – which is the subject of the metadata is described. Because the SDMX Information Model gives a formalization of statistical exchange and dissemination, the model can be used as a typology of the different actors and resources within statistical activities. Thus, the "targets" (subjects) of reference metadata sets and metadata flows can be described as corresponding to some standard class by reference to this model.

As with data structures, the generic format for metadata sets provides a known document structure, whilst the structure specific format is derived specifically from a metadata structure definition and can perform a higher degree of schema validation.

SDMX Registry Services

In order to provide visibility into the large amount of data and metadata which exists within the SDMX model of statistical exchange, it is felt that an architecture based on a set of registry services is potentially useful. A "registry" – as understood in web-services terminology – is an application which maintains and stores metadata for querying, and which can be used by any other application in the network with sufficient access privileges (though note that the mechanism of access control is outside of the scope of the SDMX standard). It can be understood as the index of a distributed database or metadata repository which is made up of all the data provider's data sets and reference metadata sets within a statistical community, located across the Internet or similar network.

Note that the SDMX registry services are not concerned with the storage of data or reference metadata. The assumption is that data and reference metadata lives on the sites of its data providers. The SDMX registry services concern themselves with providing visibility of the data and reference metadata, and information needed to access the data and reference metadata. Thus, a registered data set will have its URL available in the registry, but not the data itself. An application which wishes to access that data would query the registry, perhaps by drilling down via a Category Scheme and Dataflow, for the URL of a registered data source, and then retrieve the data directly from the data provider (using an SDMX-ML query message or other mechanism).

SDMX does not require a particular technology implementation of the registry – instead, it specifies the standard interfaces which may be supported by a registry. Thus, users may implement an SDMX-conformant registry in any fashion they choose, so long as the interfaces are supported as specified here. These interfaces are expressed as XML documents, and form a new part of the SDMX-ML language.

The registry services discussed here can be briefly summarized:

- **Maintenance of Structural Metadata:** This registry service allows users with maintenance agency access privileges to submit and modify structural metadata. In this aspect the registry is acting as a structural metadata repository. However, it is permissible in an SDMX structure to submit just the "stub" of the structural object, such as a code list, and for this stub to reference the actual location from where the metadata can be retrieved, either from a file or a structural metadata resource, such as another registry.
- **Registration of Data and Metadata Sources:** This registry service allows users with maintenance agency access privileges to inform the registry of the existence and location (for retrieval) of data sets and reference metadata sets. The registry stores metadata about these objects, and links it to the structural metadata that give sufficient structural information for an application to process it, or for an application to discover its existence. Objects in the registry are organized and categorized according to one or more category schemes.
- **Querying:** The registry services have interfaces for querying the metadata contained in a registry, so that applications and users can discover the existence of data sets and reference metadata sets, structural metadata, the providers/agencies associated with those objects, and the provider agreements which describe how the data and metadata are made available, and how they are categorized.
- **Subscription/Notification:** It is possible to "subscribe" to specific objects in a registry, so that a notification will be sent to all subscribers whenever the registry objects are updated.

Web services

Web services allow computer applications to exchange data directly over the Internet, essentially allowing modular or distributed computing in a more flexible fashion than ever before. In order to allow web services to function, however, many standards are required: for requesting and supplying data; for expressing the enveloping data which is used to package exchanged data; for describing web services to one another, to allow for easy integration into applications that use other web services as data resources.

SDMX provides guidelines for using these standards in a fashion which will promote interoperability among SDMX web services, and allow for the creation of generic client applications which will be able to communicate meaningfully with any SDMX web service which implements these guidelines.

More specifically, the SDMX web services guidelines offer:

- A normative interface (WSDL) for SOAP-based web services: The 2.0 Web-Services Guidelines contained a set of web-services functions, but these have been found through implementation to be insufficient for the types of SDMX-based web services now being developed. Furthermore, the operations and their payload have now become normative (WSDL).
- A normative interface (WADL) for RESTful web services: The RESTful API focuses on simplicity. The aim is not to replicate the full semantic richness of the SDMX-ML Query message but to make it simple to perform a limited set of standard queries. Also, in contrast to other parts of the SDMX specification, the RESTful API focuses solely on data retrieval (via HTTP GET).

A normative list of common error codes: When web services are used, it is necessary to have error codes which can help to explain the situation when problems are encountered. Prior to version 2.1 of the SDMX standard, there was no set of agreed error codes for use with SDMX web services. Version 2.1 of the SDMX standard fills that gap.

1.1.4 The SDMX Information Model

SDMX provides a way of modelling statistical data, and defines the set of metadata constructs used for this purpose. Because SDMX specifies formats in two syntaxes for expressing data and structural metadata, the model is used as a mechanism for guaranteeing that transformation between the different formats are lossless. All of the formats are syntax-bound expressions of the common information model. SDMX version 1.0 has based itself on GESMES/TS as an input to the model and formats, both to build on the proven success of this model for time series data exchange, and to ensure backward compatibility with existing GESMES/TS-based systems. Version 2.0/2.1 expands upon the version 1.0 basis to provide a more comprehensive model.

SDMX recognizes that statistical data is structured; in SDMX this structure is termed a Data Structure Definition. "Data sets" are made up of one or more lower-level "groups", based on their degrees of similarity. Each group is in turn comprised of one or more "series" of data. Each series or section has a "key" - values for each of a cluster of concepts, also called "dimensions" - which identifies it, and one or more "observations", which typically combine the time of the observation, and the value of the observation (e.g., measurement). Additionally, metadata may be attached at any level of this structure as descriptive "attributes". Code lists (enumerations) and other patterns for representation of data and metadata are also modelled.

There is some similarity between "cube" structures commonly used to process statistical data, and the Data Structure Definition idea in the SDMX Information Model. It is important to note that the data as structured according to the SDMX Information Model is optimized for exchange, potentially with partners who may have no ability to process a "cube" of data coming from complex statistical systems. SDMX time series can be understood as "slices" of the cube. Such a slice is identified by its key. A "series" key consists of the values for all dimensions specified by the key family except time. It is certainly possible to reconstruct and describe data cubes from SDMX-structured data, and to exchange such databases according to the proposed standards. In version 2.0, it becomes possible to more fully describe the structure of cubes, with hierarchical code lists, constraints, and relationships between data structure definitions.

In version 2.0/2.1, the SDMX standards also provide a view of reference metadata: a mechanism for referencing the meaningful "objects" within the SDMX view of statistical exchange processes (data providers, structures, provisioning agreements, dataflows, metadata flows, etc.) to which metadata is attached; a mechanism for describing a set of meaningful concepts, of organizing them into a presentational structure, and of indicating how their values are represented. This is based on a simple, hierarchical view of reference metadata which is common to many metadata systems and classification/categorization schemes. SDMX provides a model (and XML formats) for both describing reference metadata structures, and of reporting reference metadata according to those structures.

Version 2.0/2.1 also introduces support for metadata related to the process aspects of statistical exchange. A step-by-step process can be modelled; information about who is providing data and reference metadata and how they are providing it can be expressed; and the technical aspects of service-level agreements (and similar types of provisioning agreements) can be represented.

The SDMX Information Model formally describes all of the objects listed above, so as to present a standard view of the statistical exchange process.

The SDMX Information Model is presented using UML, and is also described in prose. While the information model is not normative, it is a valuable tool for understanding and using the normative format specifications.

1.1.5 SDMX-EDI

The SDMX-EDI format is drawn from the GESMES/TS version 3.0 implementation guide, as published as a standard of the SDMX initiative.

1. *Statistical Definitions:* An expression of the structural metadata covered by the SDMX information model in a UN/EDIFACT format.

2. *Statistical Data:* Optimized for the batch exchange of large amounts of time series data between counterparties, it allows for extremely compact expression of large whole or partial data sets. Non time series data, such as cross-sectional, can be supported if represented as repackaged time series, but there is no direct support for cross-sectional data in this format.

3. *Data Set List:* a list of data sets and their structural metadata.

The SDMX Information Model provides the constructs which are found in the EDIFACT syntax used for SDMX-EDI, and those found in the XML syntax of SDMX-ML. Since both syntactic implementations reflect the same logical constructs, SDMX-EDI data and structural metadata messages can be transformed into corresponding SDMX-ML formats, and vice-versa. Thus, these standards provide for interoperability between the UN/EDIFACT-based and XML-based systems processing and exchanging statistical data and metadata.

1.1.6 SDMX-ML

While the SDMX-EDI format is primarily designed to support batch exchange, SDMX-ML supports a wider range of requirements. XML formats are used for many different types of automated processing, and thus must support more varied processing scenarios. That is why there are several types of messages available as SDMX-ML formats. Each is suited to support a specific set of processing requirements.

1. *Structure Definition:* All SDMX-ML message types share a common XML expression of the metadata needed to understand and process a data set or metadata set, and additional metadata about category schemes and organisations is included. Also, the structural aspects of data and metadata provision – dataflows and metadataflows – are described using this format.

2. *Generic Data:* All statistical data expressible in SDMX-ML can be marked up according to this data format, in agreement with the contents of a Structure Definition message. It is designed for any scenario where applications receiving the data need to process it according to a single format. Such applications may need independent access to the data set's structure before they process it. Data marked up in this format are not particularly compact, but they make easily available all aspects of the data set. This format does not provide strict validation between the data set and its structural definition using a generic XML parser. It supports the transmission of partial data sets (incremental updates) as well as whole data sets. It supports both the time-series and the cross-sectional use cases.

3. *Structure-specific Data:* This format is specific to the Data Structure Definition of the data set (in other terms, it is DSD-specific) and is created by following mappings between the metadata constructs defined in the Structure Definition message and the technical specification of the format. It supports the exchange of large data sets in XML format (typically the size of the data set is 50% of the same data expressed as Generic Data), provides strict validation of conformance with the DSD using a generic XML parser, and supports the transmission of partial data sets (incremental updates) as well as whole data sets. The Structure-specific Data format specified in SDMX 2.1 supports both the time-series and the cross-sectional use cases which were covered by two distinct formats in SDMX 2.0.

   Many XML tools and technologies have expectations about the functions performed by an XML schema, one of which is a very direct relationship between the XML constructs described in the XML schema and the tagged data in the XML instance. Strong data typing is also considered normal, supporting full validation of the tagged data. These message types are designed to support validation and other expected XML schema functions.

4. *Generic Metadata:* All reference metadata expressible in SDMX-ML format can be marked up according to this schema. It performs only a minimum of validation, and is somewhat verbose, but it does support the creation of generic software tools and services for processing reference metadata.

5. *Structure-specific Metadata:* For each metadata structure definition, an XML schema specific to that structure can be created, to perform validation on sets of reported metadata. This structure is less verbose than the Generic Metadata format, and, because the XML mark-up relates directly to the reported concepts, it is appropriate for applications that are designed to process a specific type of metadata report. It is analogous to the Structure-specific Data format for data in its approach to the use of XML.

6. *Query:* Data and metadata are often published in databases which are available on the web. Thus, it is necessary to have a standard query document which allows the databases to be queried, and return an SDMX-ML data, reference metadata, or structure message. The Query document is an implementation of the SDMX Information Model for use in web services and database-driven applications, allowing for a standard request to be sent to data providers using these technologies.

7. *Registry:* All of the possible interactions with the SDMX registry services are supported using SDMX-ML interfaces. All but one of these documents are based on a synchronous exchange of documents – a "request" message answered by a "response" message. There are two basic types of request – a "Submit", which

writes metadata to the registry services, and a "Query", which is used to discover that metadata. Registry interactions provide formats for all types of provisioning metadata, as well as for subscription/notification, structural metadata, and data and metadata registration. The exception is the (Registry) notification message which is asynchronous.

Because all of the SDMX-ML formats are implementations of the same information model, and all the data and metadata messages are derivable from the Structure message which describes a data set or metadata set, it is possible to have standard mappings between each of the similar formats. These mappings can be implemented in generic transformation tools, useful to all SDMX-ML users, and not specific to a particular data set's key family or metadata set's structure definition (even though some of the formats they deal with may be). Part of the SDMX-ML package is the set of mappings between the structure-specific data and metadata formats and the Structure Definition format from which all are derivable.

1.1.7 Conformance

This section will contain a normative statement of what applications must do to be considered conformant with the SDMX version 2.1 specifications. This will address both the application functionality that must be supported, and the contents of an Implementer's Conformance Statement regarding SDMX conformance.

1.1.8 Dependencies on SDMX content-oriented guidelines

The technical standards proposed here are designed so that they can be used in conjunction with other SDMX guidelines which are more closely tied to the content and semantics of statistical data exchange. The SDMX Information Model works equally well with any statistical concept, but to encourage interoperability, it is also necessary to standardize and harmonize the use of specific concepts and terminology. To achieve this goal, SDMX creates and maintains guidelines for cross-domain concepts, terminology, and structural definitions. There are three major parts to this effort.

Cross-Domain Concepts

The SDMX Cross-Domain Concepts is a content guideline concerning concepts which are used across statistical domains. This list is expected to grow and to be subject to revision as SDMX is used in a growing number of domains. The use of the SDMX Cross-Domain Concepts, where appropriate, provides a framework to further promote interoperability among organisations using the technical standards presented here. The harmonization of statistical concepts includes not only the definitions of the concepts, and their names, but also, where appropriate, their representation with standard code lists, and the role they play within data structure definitions and metadata structure definitions.

The intent of this guideline is two-fold: to provide a core set of concepts which can be used to structure statistical data and metadata, to promote interoperability between systems ("structural metadata", as described above); and to promote the exchange of metadata more widely, with a set of harmonized concept names and definitions for other types of metadata ("reference metadata", as defined above.)

Metadata Common Vocabulary

The Metadata Common Vocabulary is an SDMX guideline which provides definition of terms to be used for the comparison and mapping of terminology found in data structure definitions and in other aspects of statistical metadata management. Essentially, it provides ISO-compliant definitions for a wide range of statistical terms, which may be used directly, or against which other terminology systems may be mapped. This set of terms is inclusive of the terminology used within the SDMX Technical Standards.

The MCV provides definitions for terms on which the SDMX Cross-Domain Metadata Concepts work is built.

Statistical Subject-Matter Domains

The Statistical Subject-Matter Domains is a listing of the breadth of statistical information for the purposes of organizing widespread statistical exchange and categorization. It acts as a standard scheme against which the categorization schemes of various counterparties can be mapped, to facilitate interoperable data and metadata exchange. It serves another useful purpose, however, which is to allow an organization of corresponding “domain groups”, each of which could define standard data structure definitions, concepts, etc. within their domains. Such groups already exist within the international community. SDMX would use the Statistical Subject-Matter Domains list to facilitate the efforts of these groups to develop the kinds of content standards which could support the interoperation of SDMX-conformant technical systems within and across statistical domains. The organisation of the content of such schemes is supported in SDMX as a Category Scheme.

SDMX Statistical Subject-Matter Domains will be listed and maintained by the SDMX Initiative and will be subject to adjustment.

1.1.9 Looking Forward

The SDMX initiative sees this set of data and metadata formats and registry services interfaces standards as useful in creating more efficient and open systems for statistical exchange. It is anticipated that SDMX will refine these standards further as they are implemented, so as to build on the interoperability enabled by having a set of standard formats and exchanges based on a common information model.

The review process for version 2.0 and 2.1 has suggested that future work should take advantage of a wider participation of the SDMX user community (statistical offices, central banks and other national and international organisations dealing with statistics) in further enhancing the Technical Standards and improving its use.

1.2 Information Model

1.2.1 Change History

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.

Modification to Data Structure Definition (DSD) to

- align the cross sectional structures with the functionality of the schema
- support Data Structure Definition extension (i.e. to derive and extend a Data Structure Definition from another Data Structure Definition), thus supporting the definition of a related "set" of key families
- distinguish between data attributes (which are described in a Data Structure Definition) from metadata attributes (which are described in a metadata structure definition)
- attach data attributes to specific identifiable artefacts (formally this was supported by attachable artefact)

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

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

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

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

Version 2.1 – release April 2011

The purpose of this revision is threefold:

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

SDMX Base

Basic inheritance and patterns

1. The following attributes are added to Maintainable:

i) isExternalReference

ii) structure URL

iii) serviceURL

2. Added Nameable Artefact and moved the Name and Description associations from Identifiable Artefact to Nameable Artefact. This allows an artefact to be identified (with id and urn) without the need to specify a Name.
3. Removed any inheritance from Versionable Artefact with the exception of Maintainable Artefact – this means that only Maintainable objects can be versioned, and objects contained in a maintainable object cannot be independently versioned.
4. Renamed MaintenanceAgency to Agency 0 this is its name in the schema and the URN.
5. Removed abstract class Association as a subclass of Item (as these association types are not maintained in Item Schemes). Specific associations are modelled explicitly (e.g. Categorisation, ItemScheme, Item).
6. Added ActionType to data types.

7. Removed Coded Artefact and Uncoded Artefact and all subclasses (e.g. Coded Data Attribute and Uncoded Data Attribute) as the "Representation" is more complex than just a distinction between coded and uncoded.
8. Added Representation to the Component. Removed association to Type.
9. Removed concept role association (to Item) as roles are identified by a relationship to a Concept.
10. Removed abstract class Attribute as both Data Attribute and Metadata Attribute have different properties. Data Attribute and Metadata Attribute inherit directly from Component.
11. isPartial attribute added to Item Scheme to support partial Item Schemes (e.g. partial Code list).

Representation

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

Organisations

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

Mapping (Structure Maps)

Updated Item Scheme Association as follows:

1. Renamed to Item Scheme Map to reflect better the sub classes and relate better to the naming in the schema.
2. Removed inheritance of Item Scheme Map from Item Scheme, and inherited directly from Nameable Artefact.
3. Item Association inherits from Identifiable Artefact.
4. Removed Property from the model as this is not supported in the schema.
5. Removed association type between Item Scheme Map and Item, and Association and Item.
6. Removed Association from the model.
7. Made Item Association a sub class of Identifiable, was a sub class Item.
8. Removed association to Property from both Item Scheme Map and Item.
9. Added attribute alias to both Item Scheme Association and Item Association.
10. Made Item Scheme Map and Item Association abstract.
11. Added sub-classes to Item Scheme Map – there is a subclass for each type of Item Scheme Association (e.g. Code list Map).
12. Added mapping between Reporting Taxonomy as this is an Item Scheme and can be mapped in the same way as other Item Schemes.
13. Added Hybrid Code list Map and Hybrid Code Map to support code mappings between a Code list and a Hierarchical Code list.

Mapping: Structure Map

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

Concept

1. Added association to Representation.

Data Structure Definition

1. Added Measure Dimension to support structure-specific renderings of the DSD. The Measure Dimension is associated to a Concept Scheme that specifies the individual measures that are valid.
2. The three types of “Dimension”, - Dimension, Measure Dimension, Time Dimension – have a super class – Dimension Component
3. Added association to a Concept that defines the role that the component (Dimension, Data Attribute, Measure Dimension) plays in the DSD. This replaces the Boolean attributes on the components.
4. Added Primary Measure and removed this as role of Measure.
5. Deleted the derived Data Structure Definition association from Data Structure Definition to itself as this is not supported directly in DSD.
6. Deleted attribute GroupKeyDescriptor.isAttachmentConstraint and replaced with an association to an Attachment Constraint.
7. Replaced association from Data Attribute to Attachable Artefact with association to Attribute Relationship.
8. Added a set of classes to support Attribute Relationship.
9. Renamed KeyDescriptor to DimensionDescriptor to better reflect its purpose.
10. Renamed GroupKeyDescriptor to GroupDimensionDescriptor to better reflect its purpose.

Code list

1. CodeList classname changed to Codelist.
2. Removed codevalueLength from Codelist as this is supported by Facet.
3. Removed hierarchyView association between Code and Hierarchy as this association is not implemented.

Metadata Structure Definition(MSD)

1. Full Target Identifier, Partial Target Identifier, and Identifier Component are replaced by Metadata Target and Target Object. Essentially this eliminates one level of specification and reference in the MSD, and so makes the MSD more intuitive and easier to specify and to understand.
2. Re-named Identifiable Object Type to Identifiable Object Target and moved to the MSD package.
3. Added sub classes to Target Object as these are the actual types of object to which metadata can be attached. These are Identifiable Object Target (allows reporting of metadata to any identifiable object), Key Descriptor Values Target (allows reporting of metadata for a data series key, Data Set Target (allows reporting of metadata to a data set), and Reporting Period Target (allows the metadata set to specify a reporting period).
4. Allowed Target Object can have any type of Representation, this was restricted in version 2.0 to an enumerated representation in the model (but not in the schemas).
5. Removed Object Type Scheme (as users cannot maintain their own list of object types), and replaced with an enumeration of Identifiable Objects.
6. Removed association between Metadata Attribute and Identifiable Artefact and replaced this with an association between Report Structure and Metadata Target, and allowed one Report Structure to reference more than on Metadata Target. This allowing a single Report Structure to be defined for many object types.
7. Added the ability to specify that a Metadata Attribute can be repeated in a Metadata Set and that a Metadata Attribute can be specified as “presentational” meaning that it is present for structural and presentational purposes, and will not have content in a Metadata Set.
8. The Representation of a Metadata Attribute uses Extended Facet (to support Xhtml).

Metadata Set

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

2. Removed Attribute Property as the underlying Property class has been removed.
3. One Metadata Set is restricted to reporting metadata for a single Report Structure.
4. The Metadata Report classes are re-structured and re-named to be consistent with the renaming and restructuring of the MSD.
5. Metadata Attribute Value is renamed Reported Attribute to be consistent with the schemas.
6. Deleted XML attribute and Contact Details from the inheritance diagram.

Category Scheme

1. Added Categorisation. Category no longer has a direct association to Dataflow and Metadataflow.
2. Changed Reporting Taxonomy inheritance from Category Scheme to Maintainable Artefact.
3. Added Reporting Category and associated this to Structure Usage.

Data Set

1. Removed the association to Provision Agreement from the diagram.
2. Added association to Data Structure Definition. This association was implied via the dataflow but this is optional in the implementation whereas the association to the Data Structure Definition is mandatory.
3. Added attributes to Data Set.
4. There is a single, unified, model of the Data Set which supports four types of data set:
   - Generic Data Set – for reporting any type of data series, including time series and what is sometimes known as "cross sectional data". In this data set, the value of any one dimension (including the Time Dimension) can be reported with the observation (this must be for the same dimension for the entire data set)
   - Structure-specific Data Set – for reporting a data series that is specific to a DSD
   - Generic Time Series Data Set – this is identical to the Generic Data Set except it must contain only time series, which means that a value for the Time Dimension is reported with the Observation
   - Structure-specific Time Series Data Set - this is identical to the Structure-specific Data Set except it must contain only time series, which means that a value for the Time Dimension is reported with the Observation.
5. Removed Data Set as a sub class of Identifiable – but note that Data Set has a "setId" attribute.
6. Added coded and uncoded variants of Key Value, Observation, and Attribute Value in order to show the relationship between the coded values in the data set and the Codelist in the Data Structure Definition.
7. Made Key Value abstract with sub classes for coded, uncoded, measure (MeasureKeyValue) ads time(TimeKeyValue) The Measure Key Value is associated to a Concept as it must take its identify from a Concept.

XSDataSet

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

Constraint

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

Process

1. Removed inheritance from Item Scheme and Item: Process inherits directly from Maintainable and Process Step from Identifiable.
2. Removed specialisation association between Transition and Association.

3. Removed Transition Scheme - transitions are explicitly specified and not maintained as Items in a Item Scheme.
4. Removed Expression and replaced with Computation.
5. Transition is associated to Process Step and not Process itself. Therefore the source association to Process Step is removed.
6. Removed Expressions as these are not implemented in the schemas. But note that the Transformations and Expressions model is retained, though it is not implemented in the schemas.

Hierarchical Codelist

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

Provisioning and Registration

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

1.2.2 Introduction

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

Related Documents

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

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

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

SDMX SECTION 05 REGISTRY SPECIFICATION: LOGICAL INTERFACES

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

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 Appendix "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.

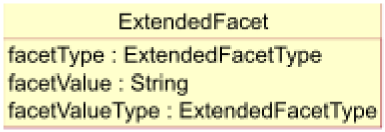

Fig. 1.2: Class with operations suppressed

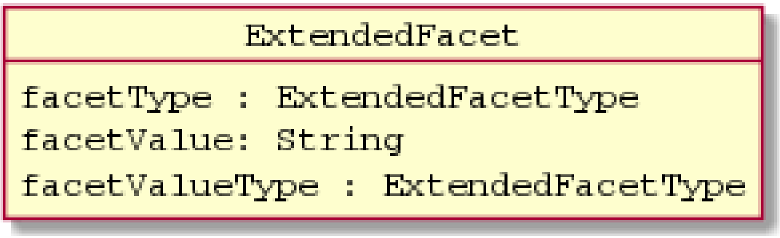

Fig. 1.3: Class with operations suppressed 2

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 method used is:

- 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.

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

Class	Feature	Description
ClassName		
	attributeName	.
	associationName	
	+roleName	

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

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.

Overall Functionality

Information Model Packages

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

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

- the SDMX Base layer comprises fundamental building blocks which are used by the Structural Definitions layer and the Reporting and Dissemination layer
- the Structural Definitions 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

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

Version 1.0

In version 1.0 the metamodel supported the requirements for:

- Data Structure Definition definition including (domain) category scheme, (metadata) concept scheme, and code list
- Data and related metadata reporting and dissemination

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 were included in the version 1.0 specification.

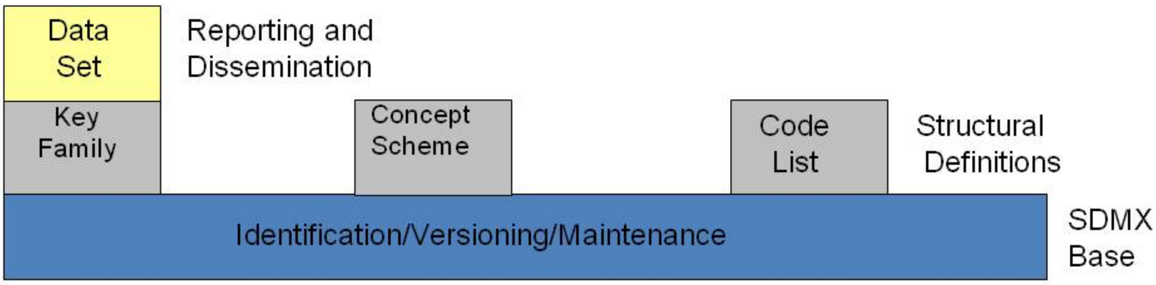

Figure 3: SDMX Information Model Version 1.0 package structure

Version 2.0/2.1

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

- Metadata structure definition
- Metadata set
- Hierarchical Codelist
- Data and Metadata Provisioning
- Process
- Mapping
- Constraints
- Constructs supporting the Registry

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

[1] OLAP: On line analytical processing

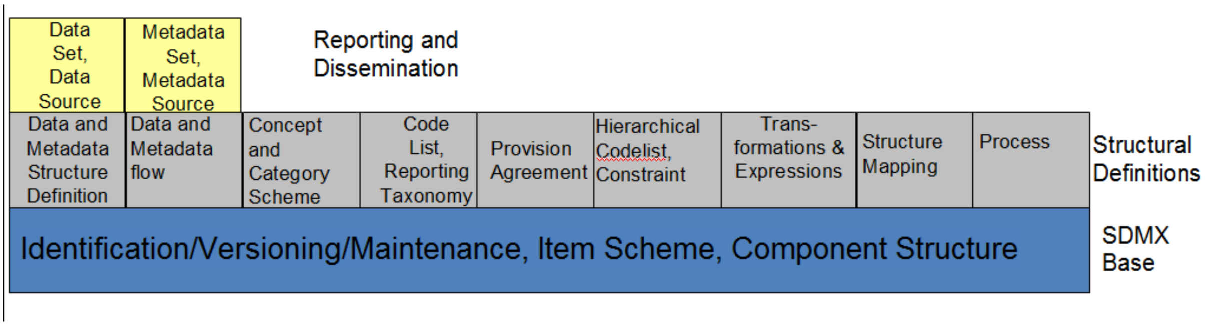

Figure 4 SDMX Information Model Version 2.0/2.1 package structure

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

- Subscription and Notification
- Registration
- Discovery

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

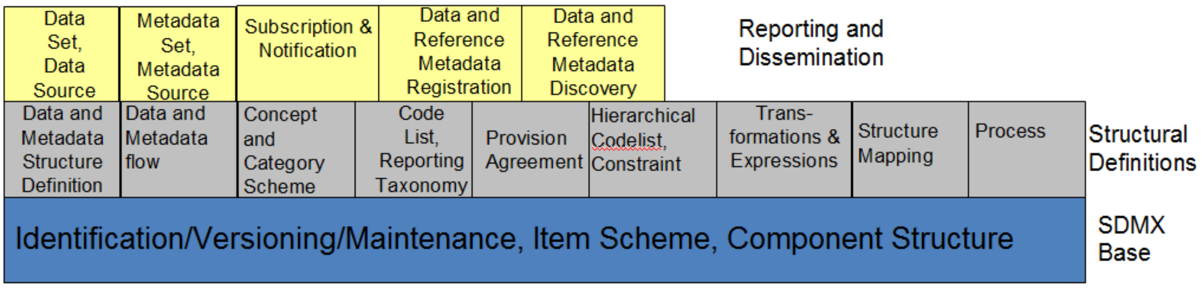

Figure 5: SDMX Information Model Version 2.0/2.1 package structure including the registry

1.2.3 Actors and Use Cases

Introduction

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"

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

- Maintenance of structural and provisioning definitions
- Data and reference metadata publishing (reporting), and consuming (using)

- Access to data, reference metadata, and structural and provisioning definitions

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

Use Case Diagrams

Maintenance of Structural and Provisioning Definitions

Use cases

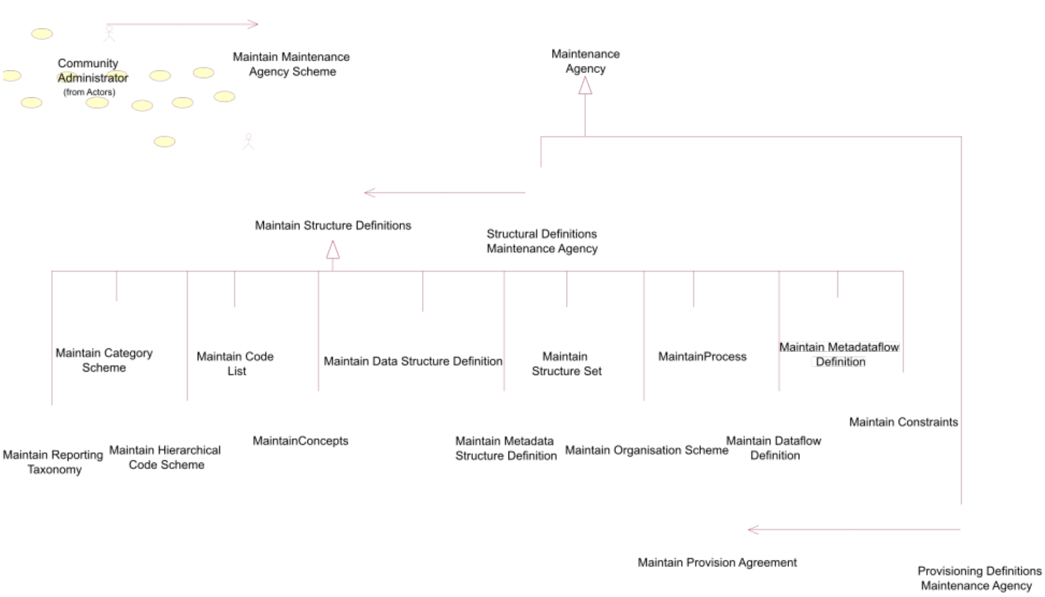

Figure 6 **Use cases for maintaining data and metadata structural and provisioning definitions**

Explanation of the Diagram

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

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

- maintaining structural definitions
- maintaining provisioning definitions

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

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

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

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

Publishing and Using Data and Reference Metadata

Use Cases

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

Explanation of the Diagram

Note that in this diagram "publishing" data and reference metadata is deemed to be the same as "reporting" data and reference metadata. In some cases the act of making the data available fulfils both functions. Aggregated data is published and in order for the Data Publisher to do this and in order for consuming applications to process the data and reference metadata its structure must be known. Furthermore, consuming applications may also require access to reference metadata in order to present this to the Data Consumer so that the data is better understood. As with the data, the reference metadata also needs to be formatted in accordance with a maintained structure. The Data Consumer and Metadata Consumer cannot use the data or reference metadata unless it is "published" and so there is a "data source" or "metadata source" dependency between the "uses" and "publish" use cases.

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

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.
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.

1.2.4 SDMX Base Package

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.

Base Structures - Identification, Versioning, and Maintenance

Class Diagram

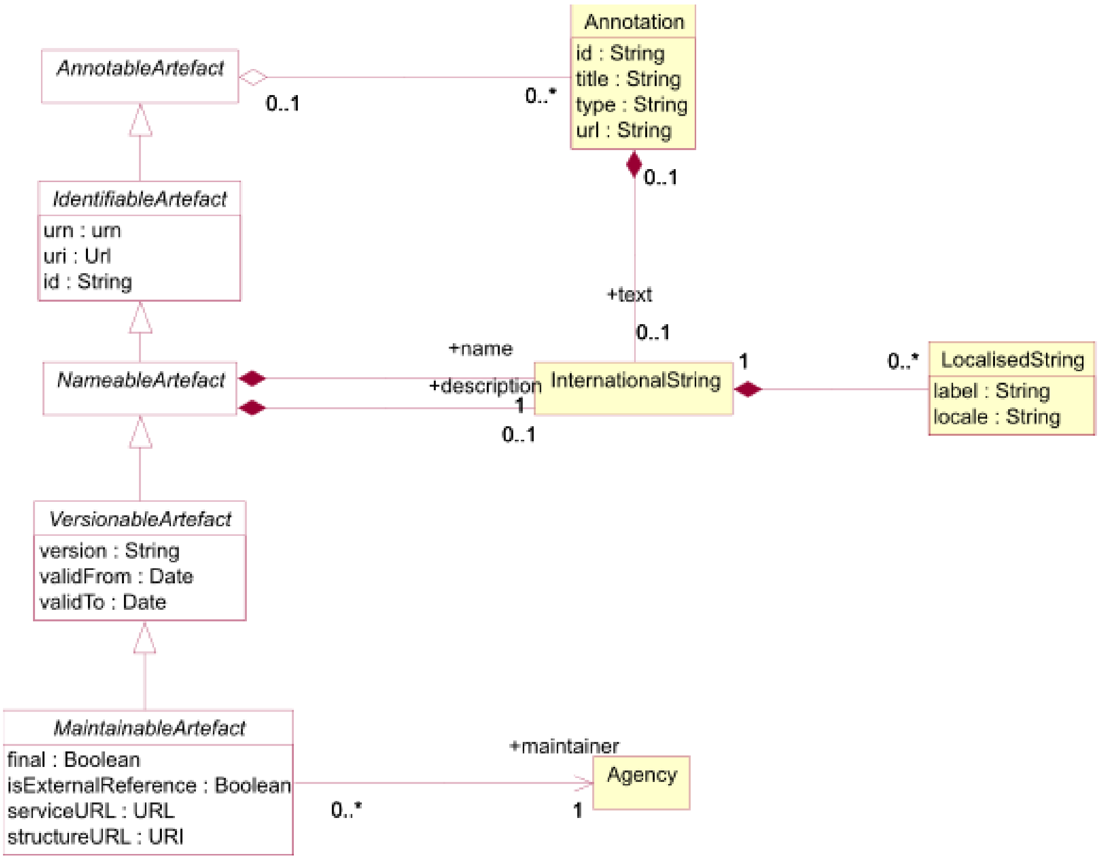

Figure 9: SDMX Identification, Maintenance and Versioning

Explanation of the Diagram

Narrative

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

All classes derived from the abstract class *AnnotableArtefact* may have Annotations (or notes): this supports the need 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 for identification. Concrete classes based on *IdentifiableArtefact* all inherit the ability to be uniquely identified.

The *NamableArtefact* is an abstract class that inherits from *IdentifiableArtefact* and in addition the +description and +name roles support multilingual descriptions and names for all objects based on *NameableArtefact*. 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 *NameableArtefact* 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 *Agency, and adds locational information (i.e. from where the object can be retrieved).* It is possible to define whether the artefact is draft or final with the final attribute.

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

Definitions

Class	Feature	Description
AnnotableArtefact	Base inheritance sub classes are: *IdentifiableArtefact*	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 International String provides the multilingual text content of the annotation via this role.
IdentifiableArtefact	Superclass is *AnnotableArtefact* Base inheritance sub classes are: NameableArtefact	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.
NameableArtefact	Superclass is *IdentifiableArtefact* Base inheritance sub classes are: *VersionableArtefact*	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.
VersionableArtefact	Superclass is *NameableArtefact* Base inheritance sub classes are: MaintainableArtefact	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
MaintainableArtefact	Inherits from *VersionableArtefact*	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 main-

Basic Inheritance

Class Diagram– Basic Inheritance from the Base Inheritance Classes

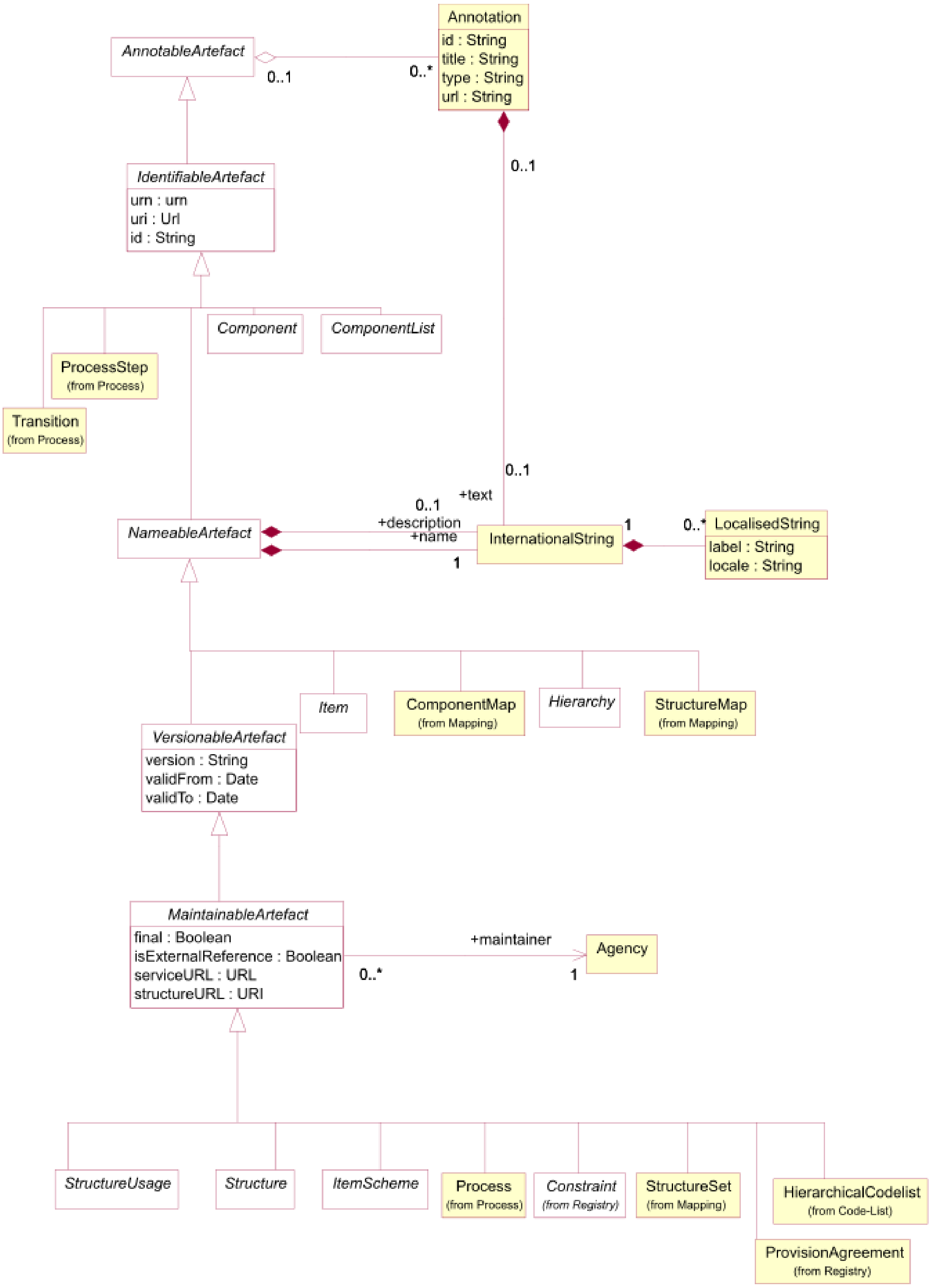

Figure 10: Basic Inheritance from the Base Structures

Explanation of the Diagram

Narrative

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

Data Types

Class Diagram

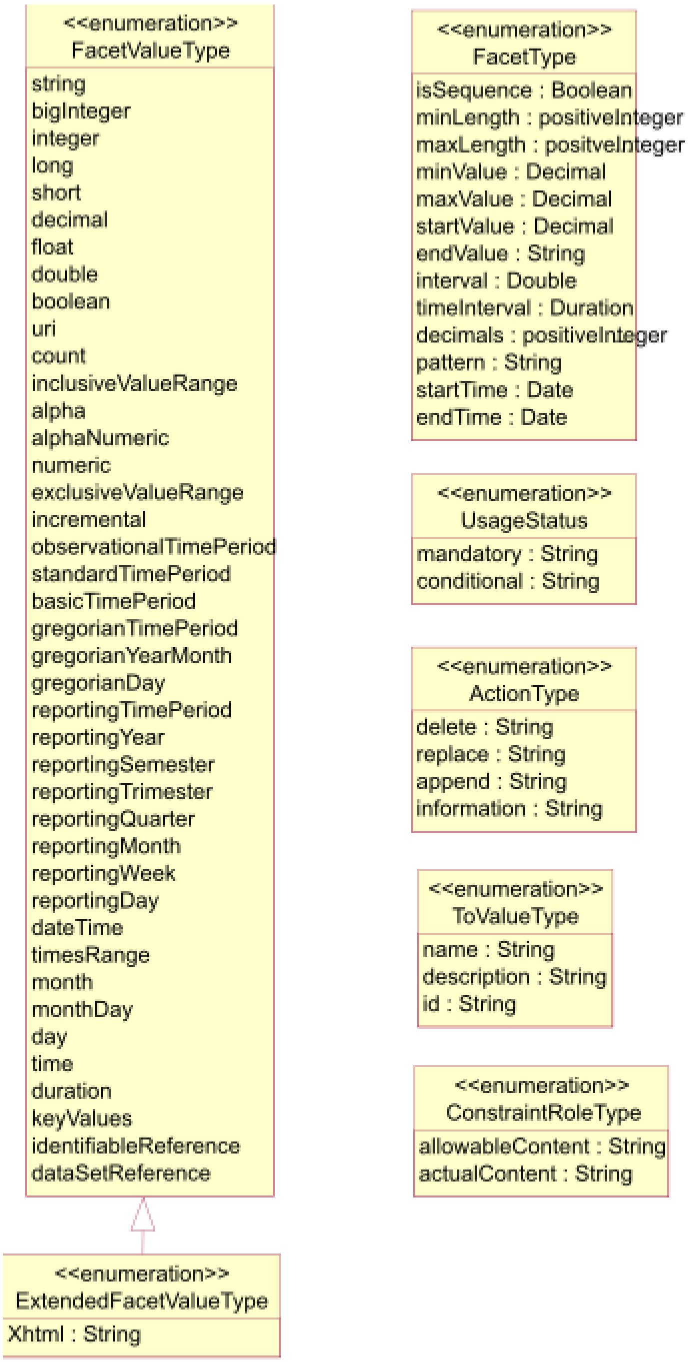

Figure 11: Class Diagram of Basic Data Types

Explanation of the Diagram

Narrative

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

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

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

- Append

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

- Replace

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

- Delete

  Data/Metadata is to be deleted.

- Information

Data and metadata are for information purposes.

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

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

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

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 ItemScheme is the basis for CategoryScheme, Codelist, ConceptScheme, *ReportingTaxonomy*, and *OrganisationScheme*.

Class Diagram

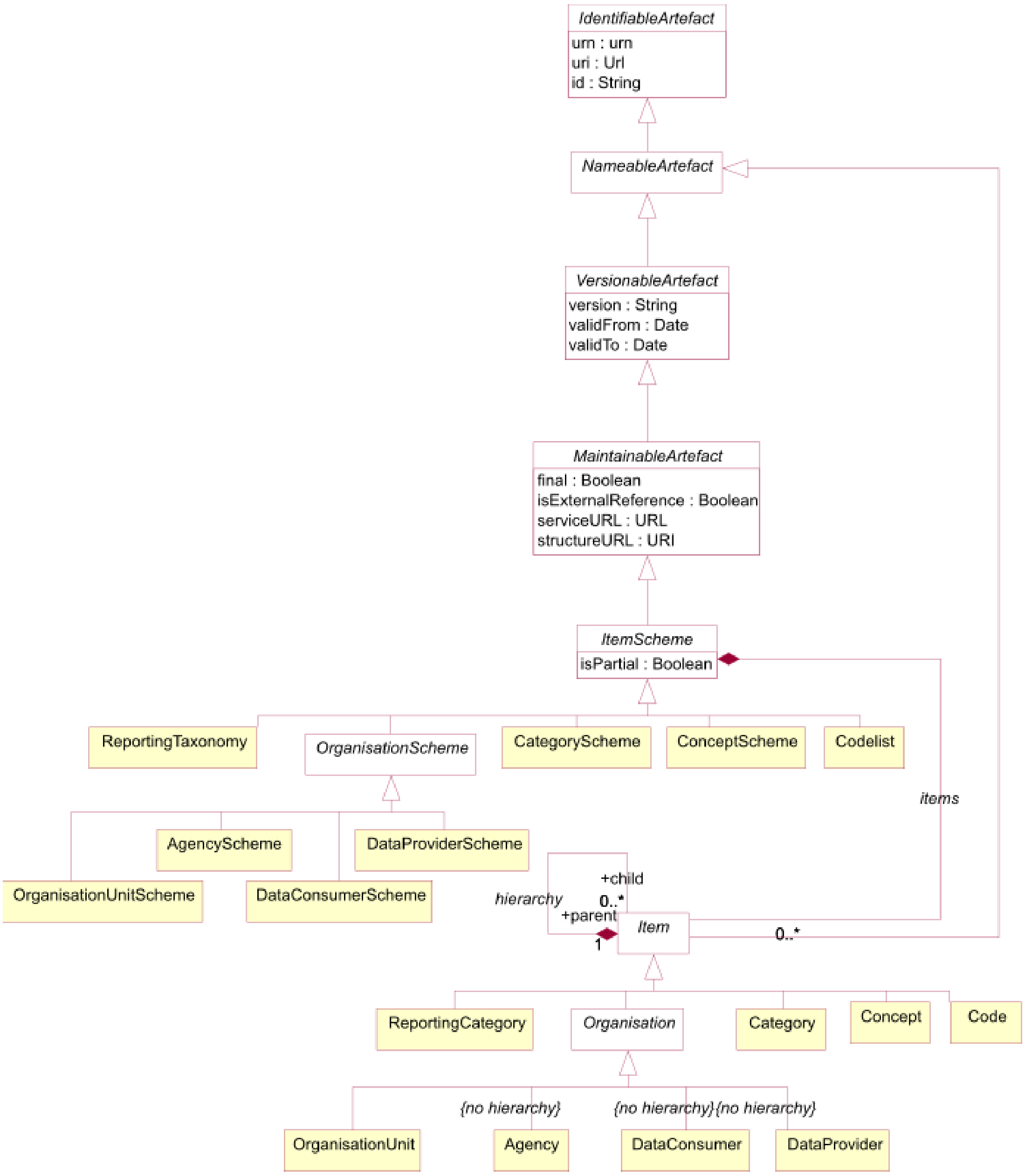

Figure 12 The Item Scheme pattern

Explanation of the Diagram

Narratve

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 gives it the ability to be annotated, have identity, naming, versioning and be associated with an Agency. An example of a concrete class is a CategoryScheme. The associated Category are *Items*.

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

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

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

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

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

Definitions

Class	Feature	Description
ItemScheme	Inherits from: *MaintainableArtefact* Direct sub classes are: CategoryScheme ConceptScheme Codelist ReportingTaxonomy *OrganisationScheme*	The descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common.
	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.
Item	Inherits from: *NameableArtefact* Direct sub classes are Category Concept Code ReportingCategory *Organisation*	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.

The Structure Pattern

Context

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

Class Diagrams

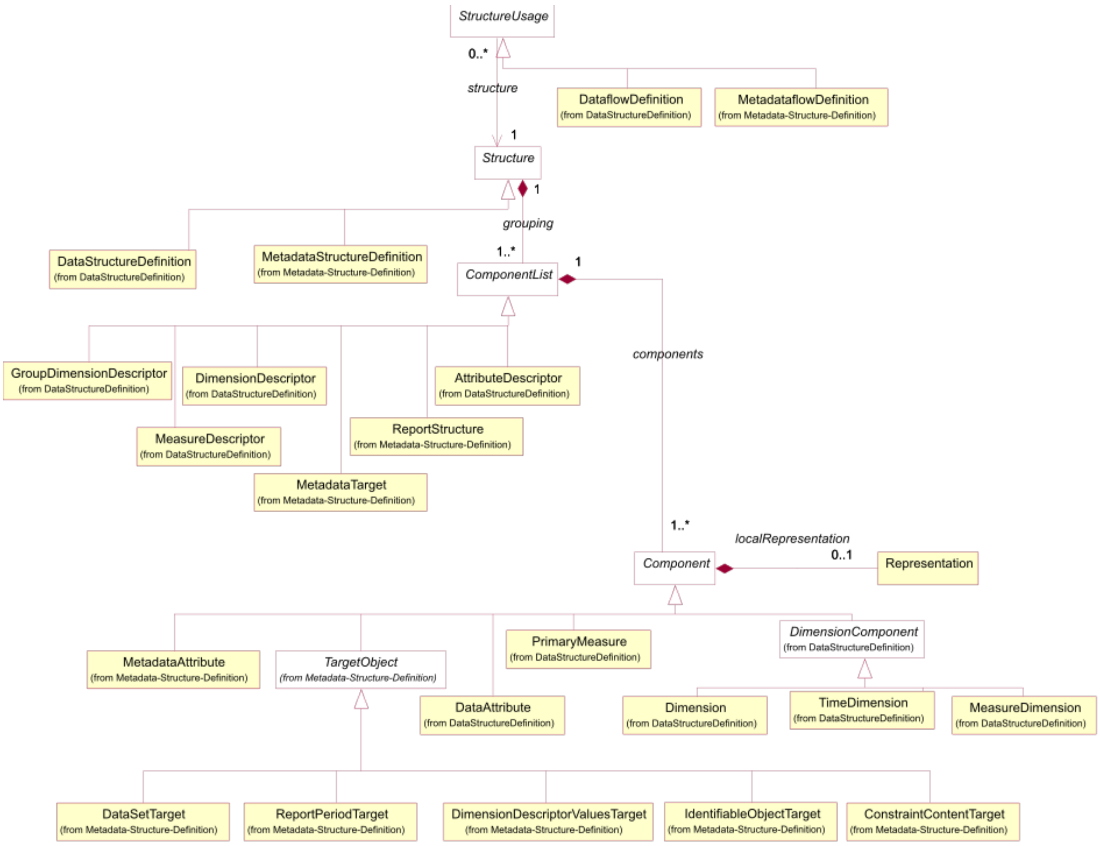

Figure 13: The Structure Pattern

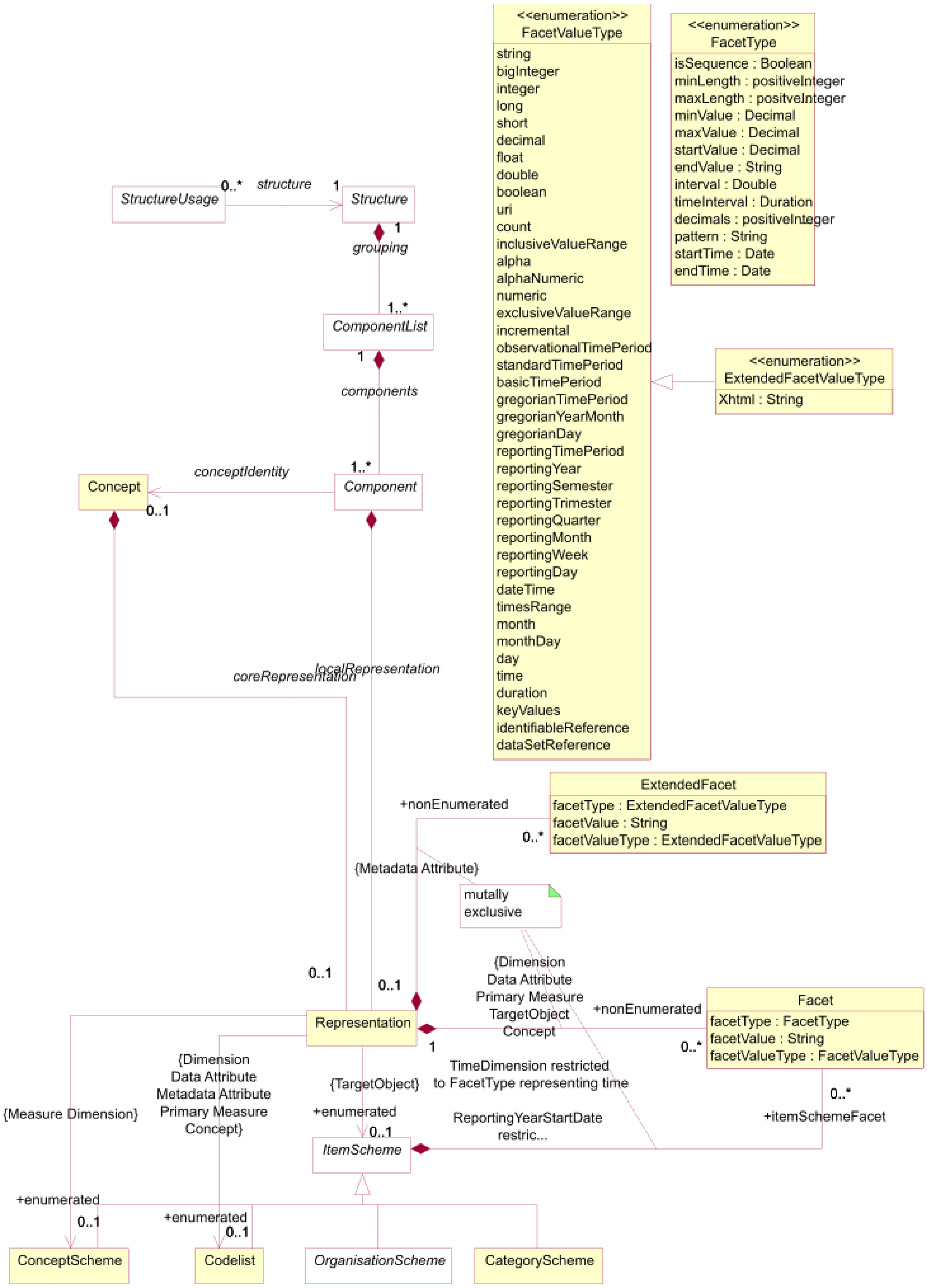

Figure 14: Representation within the Structure Pattern

Explanation of the Diagrams

Narrative

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

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

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

DimensionDescriptor: Dimension, Measure Dimension, Time Dimension

GroupDimensionDescriptor: Dimension, Measure Dimension, Time Dimension

MeasureDescriptor: PrimaryMeasure

AttributeDescriptor: Data Attribute

MetadataTarget: *TargetObject* and its sub classes

ReportStructure: MetadataAttribute

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

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

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

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

- The MeasureDimension must be enumerated and use a ConceptScheme
- The Dimension (but not MeasureDimension), DataAttribute, PrimaryMeasure, MetadataAttribute may be enumerated and, if so, use a Codelist
- The *TargetObject* may be enumerated and, if so, can use any ItemScheme (Codelist, ConceptScheme, *OrganisationScheme*, CategoryScheme, ReportingTaxonomy)
- The Dimension (but not MeasureDimension), Data Attribute, PrimaryMeasure, *TargetObject* may be non-enumerated and, if so, use one of more Facet, note that the FacetValueType applicable to the TimeDimension is restricted to those that represent time
- The MetadataAttribute may be non-enumerated and, if so, uses one or more ExtendedFacet

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

Definitions

Class	Feature	Description
StructureUsage	Inherits from: *MaintainableArtefact* Sub classes are: DataflowDefinition MetadataflowDefinition	An artefact whose components are described by a Structure. In concrete terms (sub-classes) an example would be a Dataflow Definition which is linked to a given structure – in this case the Data Structure Definition.
	structure	An association to a Structure specifying the structure of the artefact.
Structure	Inherits from: *MaintainableArtefact* Sub classes are: DataStructure Definition MetadataStructure Definition	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.
ComponentList	Inherits from: *IdentifiableArtefact* Sub classes are: DimensionDescriptor GroupDimension Descriptor MeasureDescriptor AttributeDescriptor MetadataTarget ReportStructure	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.
Component	Inherits from: *IdentifiableArtefact* Sub classes are: *PrimaryMeasure DataAttribute DimensionComponent TargetObject* MetadataAttribute	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
	+enumerated	Association to an enumerated list that contains the allowable content for the Component when reported

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

Representation Constructs

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

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**

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

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

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 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).

1.2.5 Specific Item Schemes

Introduction

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*. These concrete classes are:

- Codelist
- ConceptScheme
- CategoryScheme
- AgencyScheme, DataProviderScheme, DataConsumerScheme, OrganisationUnitScheme which all inherit from the abstract class *OrganisationScheme*
- Reporting Taxonomy

Inheritance View

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

Codelist

Class Diagram

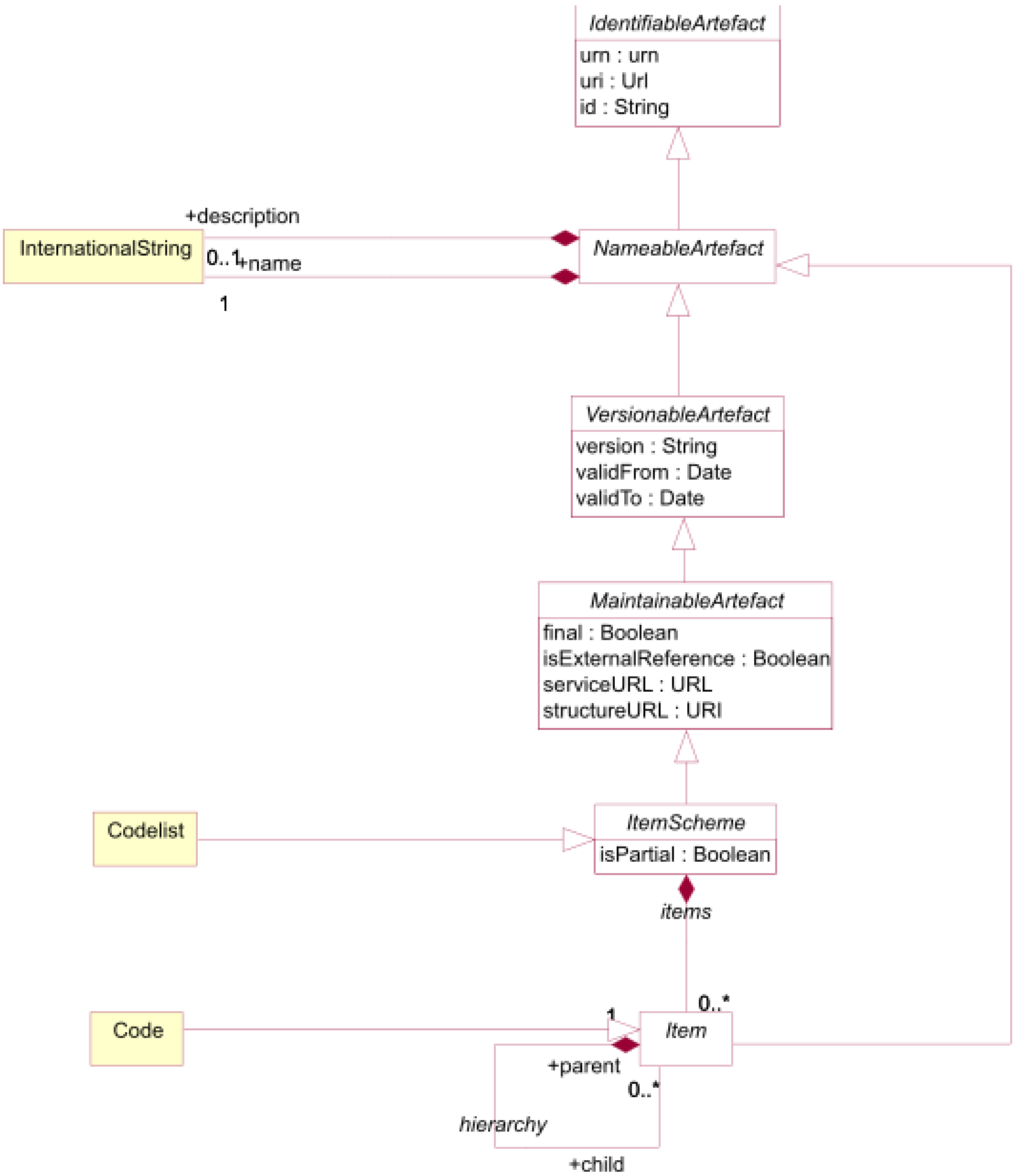

Figure 15 Class diagram of the Codelist

Explanation of the Diagram

Narrative

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

- id
- uri
- urn
- version
- validFrom
- validTo
- *isExternalReference*
- *serviceURL*
- *structureURL*
- final
- isPartial

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

- id
- uri
- urn

Both Codelist and Code have 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).

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

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

Definitions

Class	Feature	Description
Codelist	Inherits from *ItemScheme*	A list from which some statistical concepts (coded concepts) take their values.
Code	Inherits from *Item*	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.

Concept Scheme and Concepts

Class Diagram - Inheritance

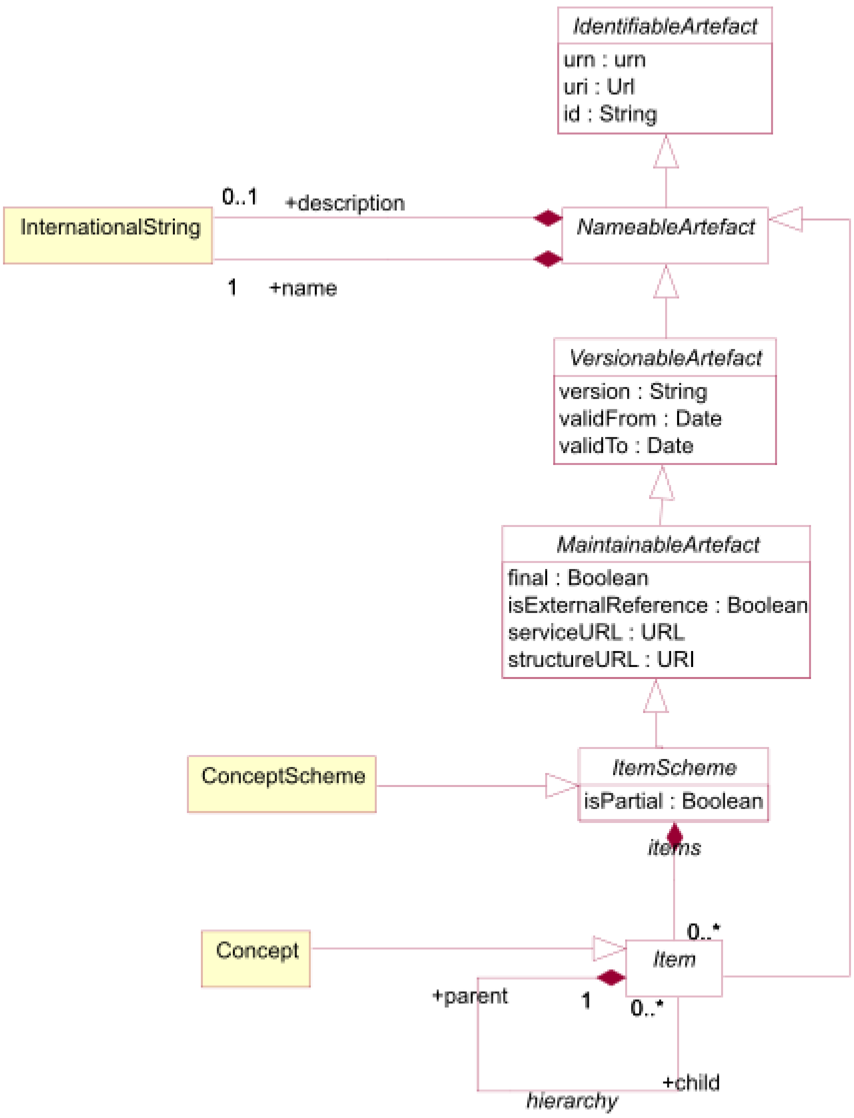

Figure 16 Class diagram of the Concept Scheme

Explanation of the Diagram

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

- id
- uri
- urn
- version
- validFrom
- validTo

- *isExternalReference*
- *registryURL*
- *structureURL*
- *repositoryURL*
- final
- isPartial

Concept inherits from Item and has the following attributes:

- id
- uri
- urn

Through the inheritance from *NameableArtefact* both ConceptScheme and Concept have 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).

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

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

Class Diagram - Relationship

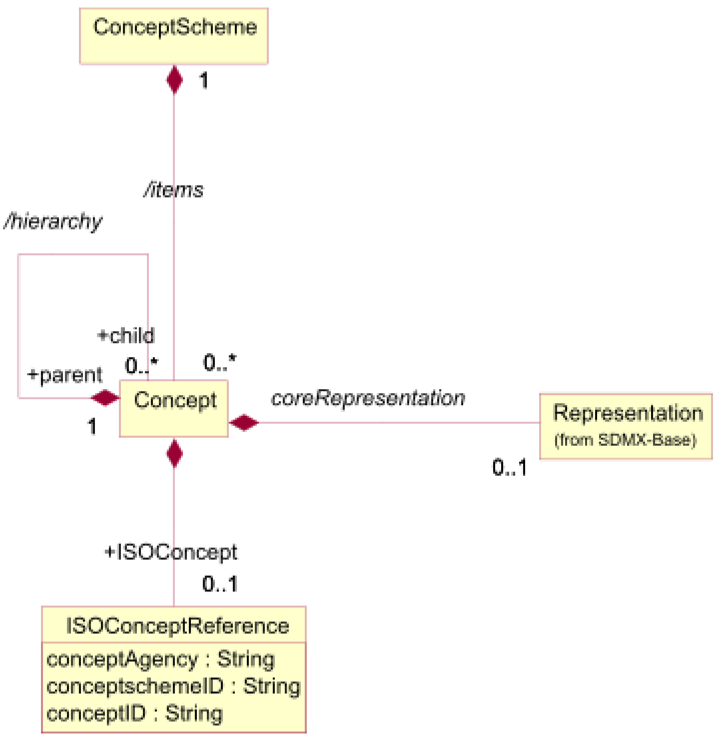

Figure 17: Relationship class diagram of the Concept Scheme

Explanation of the diagram

Narrative

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

The Concept can be associated with a *coreRepresentation*. The coreRepresentation is the specification of the format and value domain of the Concept when used on a structure like a DataStructureDefinition or a MetadataStructureDefinition, unless the specification of the Representation is overridden in the relevant structure definition. In a hierarchical ConceptScheme the Representation is inherited from the parent Concept unless overridden at the level of the child Concept.

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

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

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

Definitions

Class	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.
Concept	Inherits from *Item*	A concept is a unit of knowledge created by a unique combination of characteristics.
	/hierarchy	Associates the parent and the child concept.
	coreRepresentation	Associates a Representation.
	+ISOConcept	Association to an ISO concept reference.
ISOConceptReference		The identity of an ISO concept definition.
	conceptAgency	The maintenance agency of the concept scheme containing the concept.
	conceptSchemeID	The identifier of the concept scheme.
	conceptID	The identifier of the concept.

Category Scheme

Context

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

Class diagram - Inheritance

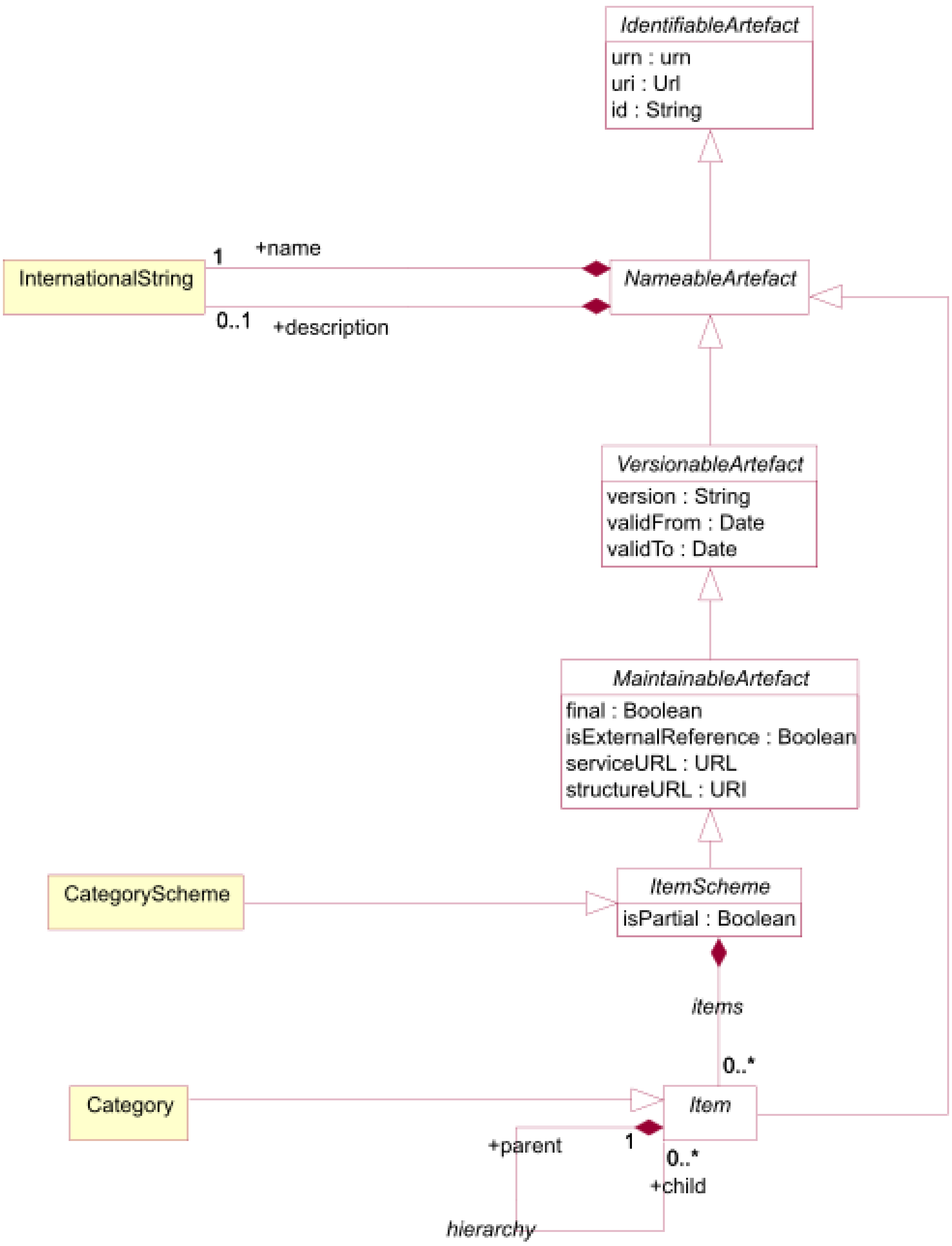

Figure 18 Inheritance Class diagram of the Category Scheme

Explanation of the Diagram

Narrative

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

- id
- uri
- urn
- version
- validFrom
- validTo
- *isExternalReference*
- *structureURL*
- *serviceURL*
- final
- isPartial

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

- id
- uri
- urn

Both CategoryScheme and Category have 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 on the model).

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

A partial CategoryScheme (where isPartial is set to "true") is identical to a CategoryScheme and contains the Category and associated names and descriptions, just as in a normal CategoryScheme. However, its content is a sub set of the full CategoryScheme. The way this works is described in section 3.5.3.1 on ItemScheme.

Class diagram - Relationship

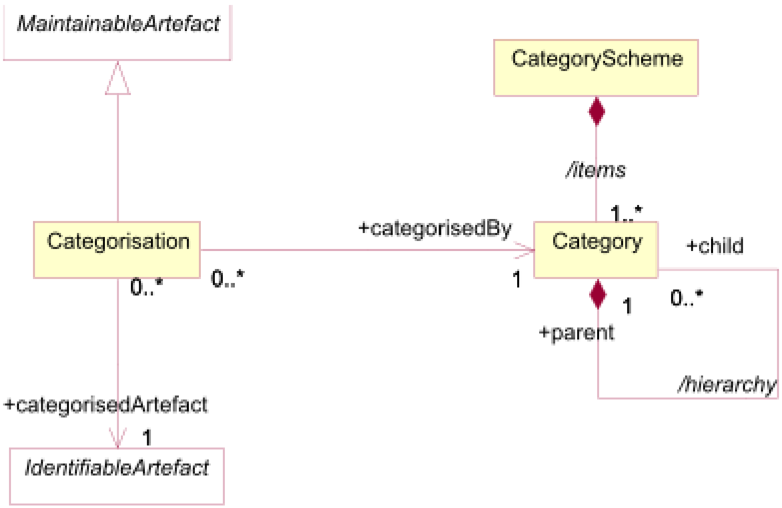

Figure 19: Relationship Class diagram of the Category Scheme

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

Definitions

Class	Feature	Description
CategoryScheme	Inherits from *ItemScheme*	The descriptive information for an arrangement or division of categories into groups based on characteristics, which the objects have in common.
	/items	Associates the categories.
Category	Inherits from *Item*	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 IdentifableArtefact with a Category.
	+categorisedArtefact	Associates the IdentifableArtefact.
	+categorisedBy	Associates the Category.

Organisation Scheme

Class Diagram

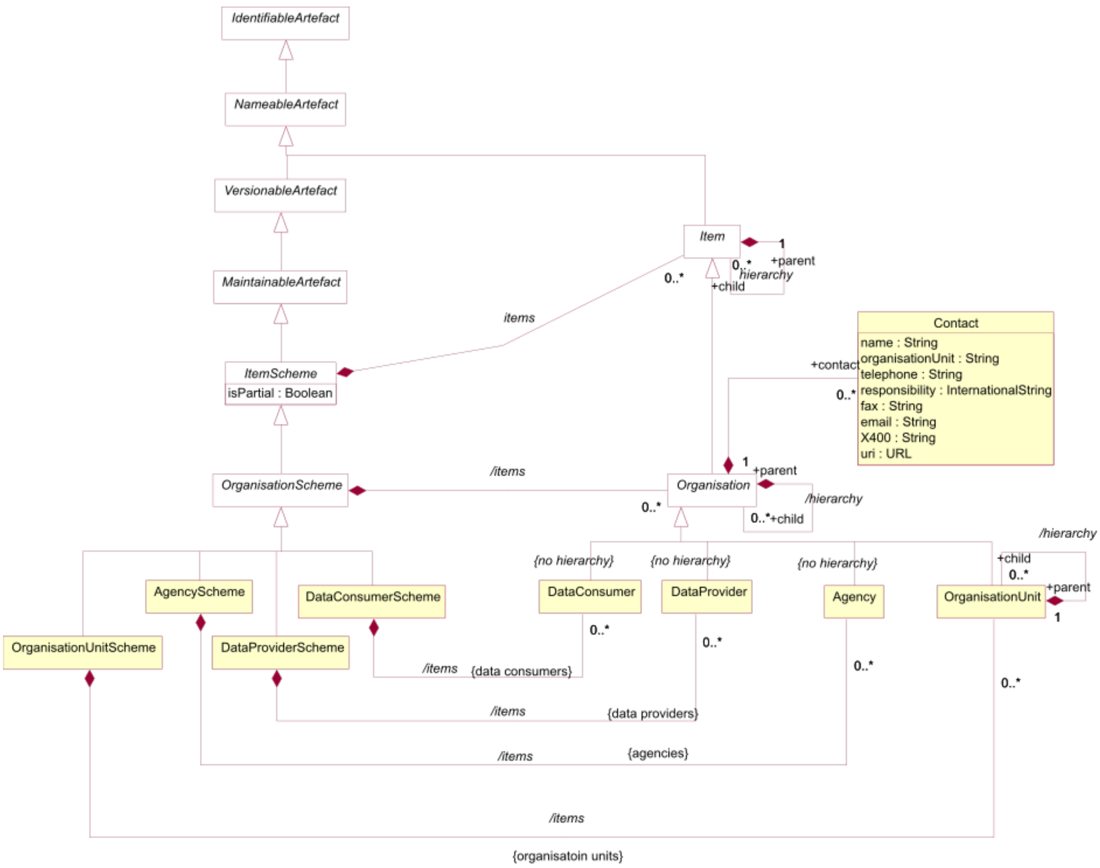

Figure 20 The Organisation Scheme class diagram

Explanation of the Diagram

Narrative

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

The *OrganisationScheme* can be one of four types:

1. AgencyScheme – contains Agency which is restricted to a flat list of agencies (i.e. there is no hierarchy). Note that the SDMX system of (Maintenance) Agency can be hierarchic and this is explained in more detail in the separate document "Technical Notes".
2. DataProviderScheme – contains DataProvider which is restricted to a flat list of agencies (i.e. there is no hierarchy).
3. DataConsumerScheme – contains DataConsumer which is restricted to a flat list of agencies (i.e. there is no hierarchy).
4. OrganisationUnitScheme – contains OrganisationUnit which does inherit the /hierarchy association from Organisation.

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

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

Definitions

Class	Feature	Description
Organisation-Scheme	Abstract Class Inherits from *ItemScheme* Sub classes are: *AgencyScheme DataProviderScheme DataConsumerScheme OrganisationUnitScheme*	A maintained collection of Organisations.
	/items	Association to the Organisations in the scheme.
Organisation	Inherits from *Item* 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.
	+contact	Association to the Contact information.
	/hierarchy	Association to child Organisations.
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 Maintenace Agencies.
	/items	Association to the Maintenance Agency in the scheme.
DataProvider-Scheme		A maintained collection of Data Providers.
	/items	Association to the Data Providers in the scheme.
DataConsumer-Scheme		A maintained collection of Data Consumers.
	/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 *Organisation*	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 *Organisation*	An organisation that produces data or reference metadata.
DataConsumer	Inherits from *Organisation*	An organisation using data as input for further processing.
OrganisationUnit	Inherits from *Organisation*	A designation in the organisational structure.

Reporting Taxonomy

Class Diagram

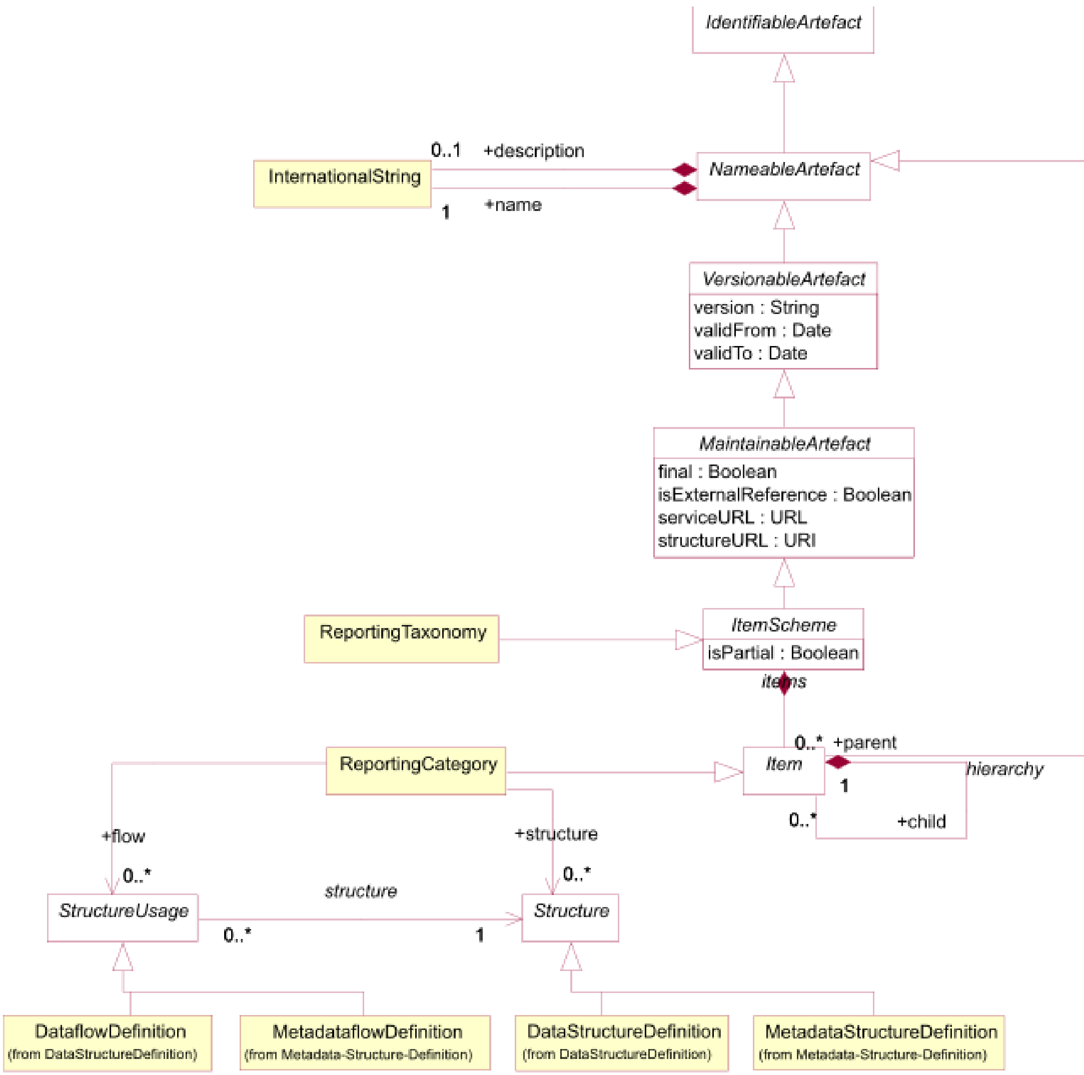

Figure 21: Class diagram of the Reporting Taxonomy

Explanation of the Diagram

Narrative

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

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

Section 1

1. linked to DatafowDefinition_1

2 linked to DatafowDefinition_2

Section 2

1 linked toDatafowDefinition_3

2 linked to DatafowDefinition_4

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

A partial ReportingTaxonomy (where isPartial is set to "true") is identical to a ReportingTaxonomy and contains the ReportingCategory and associated names and descriptions, just as in a normal ReportingTaxonomy However, its content is a sub set of the full ReportingTaxonomy The way this works is described in section 3.5.3.1 on ItemScheme.

Definitions

Class	Feature	Description
ReportingTaxonomy	Inherits from *ItemScheme*	A scheme which defines the composition structure of a data report where each component can be described by an independent Dataflow Definition or Metdataflow Definition.
	items	Associates the Reporting Category
ReportingCategory	Inherits from *Item*	A component that gives structure to the report and links to data and metadata.
	hierarchy	Associates child Reporting Category.
	+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.

1.2.6 Data Structure Definition and Dataset

Introduction

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

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 sub models these are shown in the same diagram, but are omitted from the more complex sub models for the sake of clarity. In these cases, the inheritance diagram below shows the full inheritance tree for the classes concerned with data structure definitions.

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

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

- Annotation - *AnnotableArtefact*
- Identification - *IdentifiableArtefact*
- Naming - *NameableArtefact*
- Versioning – *VersionableArtefact*
- Maintenance - *MaintainableArtefact*

Inheritance View

Class Diagram

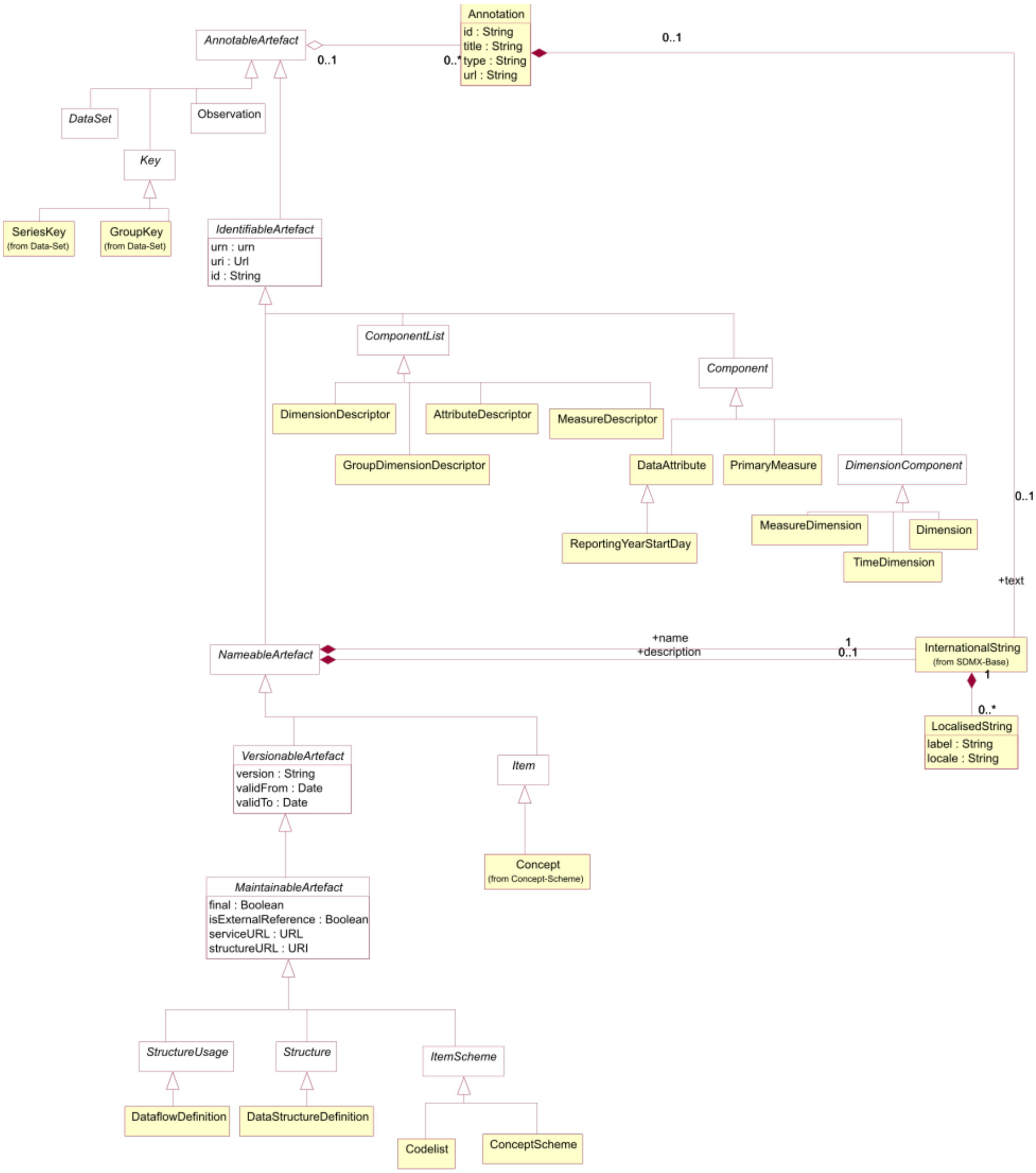

Figure 22 Class inheritance in the Data Structure Definition and Data Set Packages

Explanation of the Diagram

Narrative

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

- IdentifiableArtefact
- DataSet (and therefore StructureSpecificDataSet, GenericDataSet, GenericTimeSeriesDataSet Structure-SpecificTimeSeriesDataSet)
- Key (and therefore SeriesKey and GroupKey)

Those classes in the SDMX metamodel which require annotations and global identity are derived from *Identifi-ableArtefact* . These are:

- NameableArtefact
- ComponentList
- Component

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

- VersionableArtefact
- Item

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

- *MaintainableArtefact*

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

- *StructureUsage*
- *Structure*
- *ItemScheme*

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

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

- DataflowDefinition
- DataStructureDefinition

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

- DataStructureDefinition

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

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

- DimensionDescriptor – content is Dimension, MeasureDimension and Time Dimension
- DimensionGroupDescriptor – content is an association to Dimension, MeasureDimension, TimeDimension
- MeasureDescriptor – content is PrimaryMeasure
- AttributeDescriptor – content is DataAttribute

The classes that inherit from *Component* are:

- *PrimaryMeasure*
- DimensionComponent and thereby its sub classes of Dimension, MeasureDimension, and TimeDimension
- DataAttribute

The class that inherit from DataAttribute is:

- ReportingYearStartDay

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

Data Structure Definition – Relationship View

Class Diagram

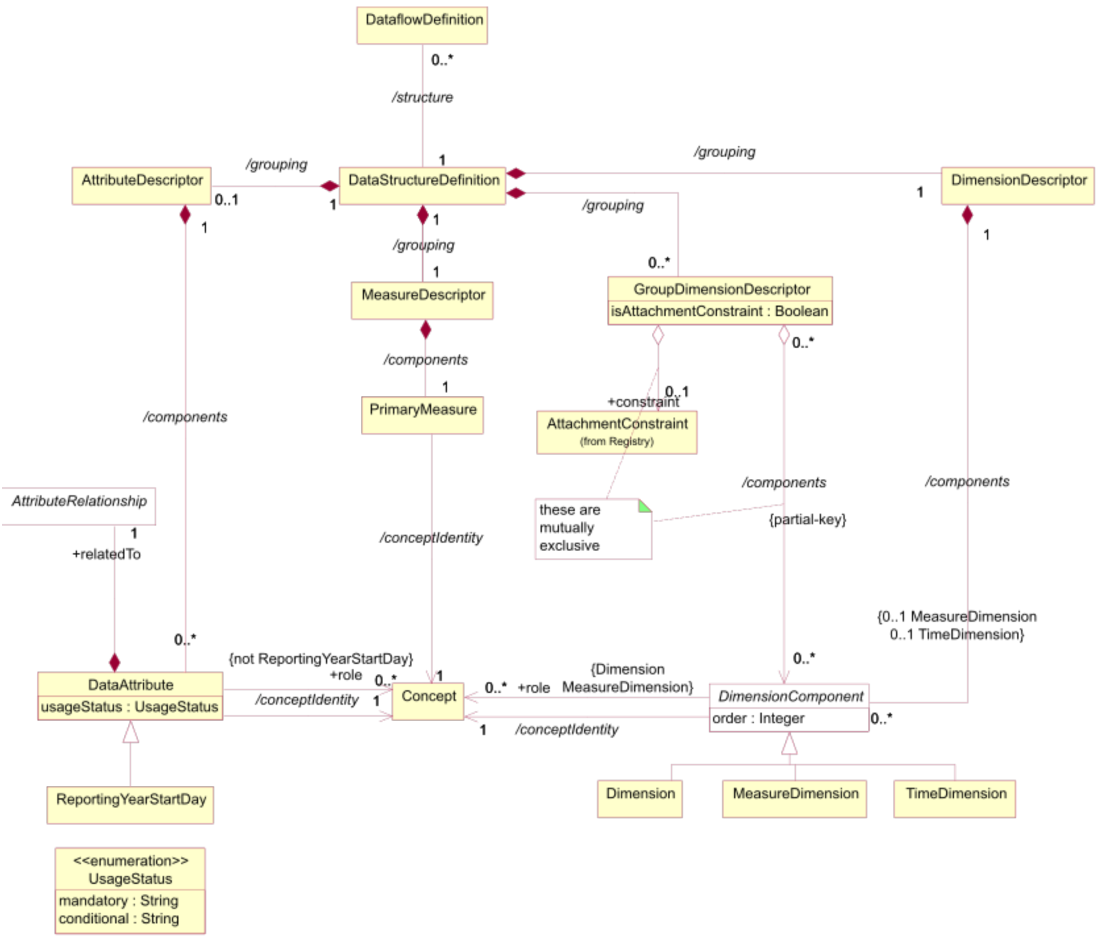

Figure 23 Relationship class diagram of the Data Structure Definition excluding representation

Explanation of the Diagrams

Narrative

A DataStructureDefinition defines the Dimensions, MeasureDimension, TimeDimension, *DataAttribute*s, and PrimaryMeasure, and associated Representation that comprise the valid structure of data and related attributes that are contained in a DataSet, which is defined by a DataflowDefinition.

The DataflowDefinition may also have additional metadata attached that defines qualitative information and Constraints on the use of the DataStructureDefinition such as the sub set of Codes used in a Dimension (this is covered later in this document – see "Data Constraints and Provisioning" section 9). Each DataflowDefinition has a maximum of one DataStructureDefinition specified which defines the structure of any DataSets to be reported/disseminated.

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

- Dimension
- MeasureDimension
- TimeDimension

Note that In the description here *DimensionComponent* can be oany or all of its sub classes i.e. Dimension, MeasureDimension, TimeDimension., and the term "DataAttribute" refers to both DataAttribute and its sub class ReportingYearStartDate.

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

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

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

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

There may be a maximum of one MeasureDimension specified in the DimensionDescriptor. The purpose of a MeasureDimension is to specify formally the meaning of the measures (because the PrimaryMeasure typically has a generic meaning e.g. observation value) and to enable multiple measures to be defined and reported in a *StructureSpecificDataSet*. Note that the MeasureDimension references a ConceptScheme as its Representation (see later) whereas a Dimension can have either an enumerated (Codelis*t*) or non-enumerated (Facet) representation. For a MeasureDimension the Concepts in the ConceptScheme comprise the list of allowable measures. This enables the representation for each individual measure (Concept) to be declared as the coreRepresentation of the Concept, thus overriding the Representation specified for the PrimaryMeasure for the observation value of this *Measure*Dimension *Concept.*

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

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

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

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

- DataSet (NoSpecifiedRelationship)
- Dimension or set of Dimensions (DimensionRelationship)
- Set of Dimensions specified by a GroupKey (GroupRelationship – this is retained for compatibility reasons – or +groupKey of the DimensionRelationship)
- Observation (PrimaryMeasureRelationship)

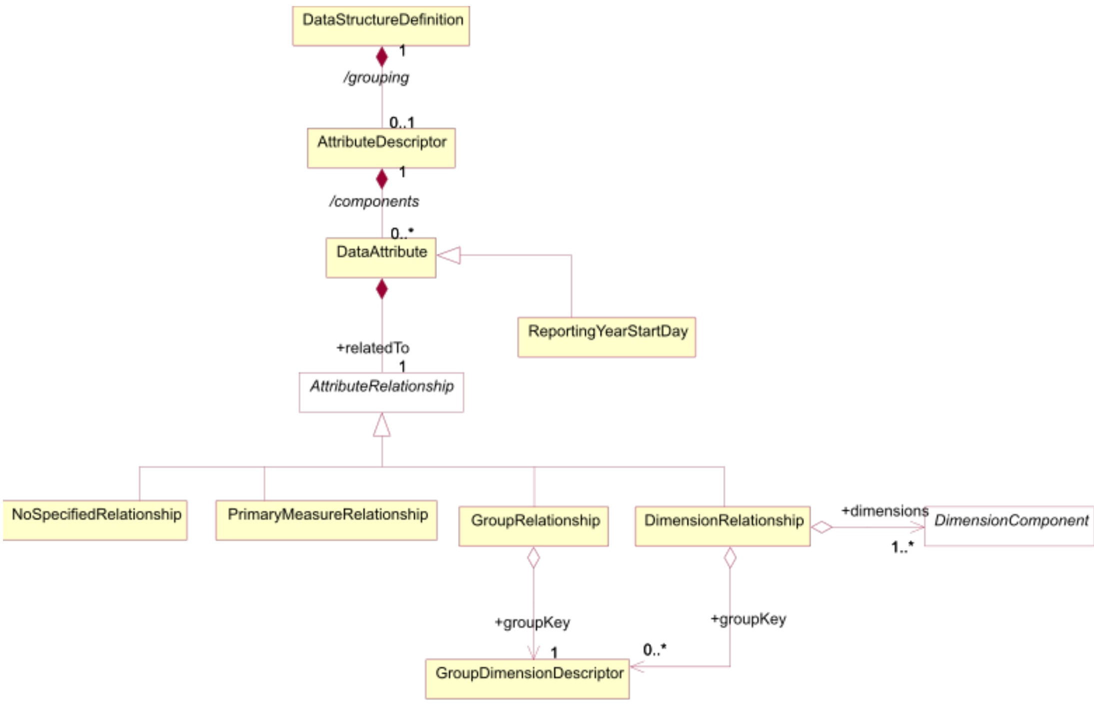

Figure 24: Attribute Attachment Defined in the Data Structure Definition

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

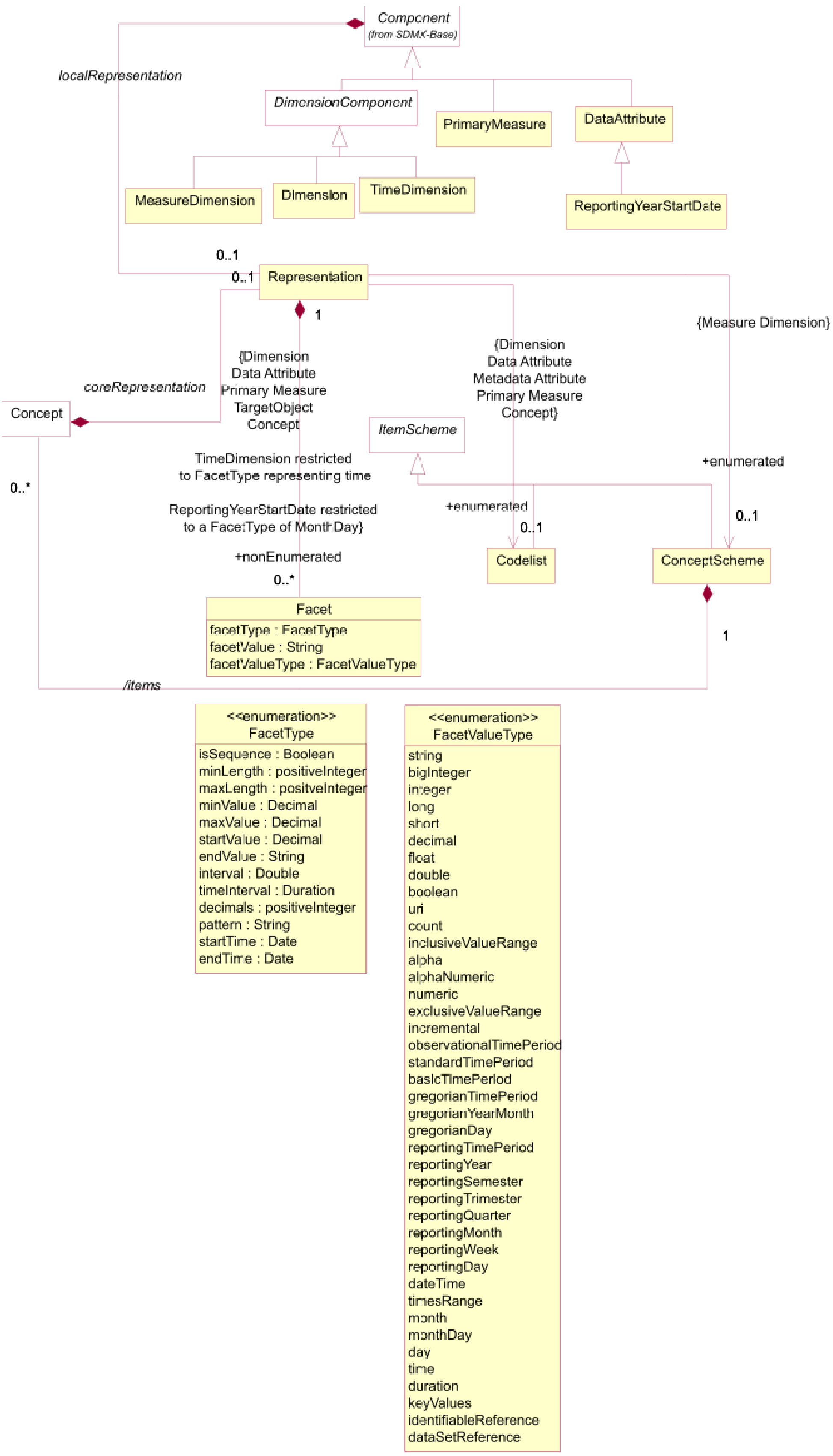
Component
(from SDMX-Base)
localRepresentation
DimensionComponent
PrimaryMeasure
DataAttribute
MeasureDimension
Dimension
TimeDimension
ReportingYearStartDate
0..1
0..1
Representation
1
{Measure Dimension}
{Dimension
Data Attribute
Primary Measure
TargetObject
Concept
{Dimension
Data Attribute
Metadata Attribute
Primary Measure
Concept}
coreRepresentation
Concept
ItemScheme
0..*
TimeDimension restricted
to FacetType representing time
+enumerated
ReportingYearStartDate restricted
to a FacetType of MonthDay}
+enumerated
0..1
0..1
+nonEnumerated
Codelist
ConceptScheme
0..*
Facet
facetType : FacetType
facetValue : String
facetValueType : FacetValueType
1
/items
<<enumeration>>
FacetType
isSequence : Boolean
minLength : positiveInteger
maxLength : positveInteger
minValue : Decimal
maxValue : Decimal
startValue : Decimal
endValue : String
interval : Double
timeInterval : Duration
decimals : positiveInteger
pattern : String
startTime : Date
endTime : Date
<<enumeration>>
FacetValueType
string
bigInteger
integer
long
short
decimal
float
double
boolean
uri
count
inclusiveValueRange
alpha
alphaNumeric
numeric
exclusiveValueRange
incremental
observationalTimePeriod
standardTimePeriod
basicTimePeriod
gregorianTimePeriod
gregorianYearMonth
gregorianDay
reportingTimePeriod
reportingYear
reportingSemester
reportingTrimester
reportingQuarter
reportingMonth
reportingWeek
reportingDay
dateTime
timesRange
month
monthDay
day
time
duration
keyValues
identifiableReference
dataSetReference

Figure 25: Representation of DSD Components

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

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

Definitions

Table 1.1: The general table 1

Class	Feature	Description
StructureUsage		See "SDMX Base".
DataflowDefinition	Inherits from *StructureUsage*	Abstract concept (i.e. the structure without any data) of a flow of data that providers will provide for different reference periods.
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.
GroupDimensionDescriptor	Inherits from *ComponentList*	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 Dimension components that comprise the group.
DimensionDescriptor	Inherits from *ComponentList*	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 *ComponentList*	A set metadata concepts that define the attributes of a Data Structure Definition.
	/components	An association to a Data Attribute component.
MeasureDescriptor	Inherits from *ComponentList*	A metadata concept that defines the measure of a Data Structure Definition.
	/components	
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 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 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".

continues on next page

Table 1.1 – continued from previous page

Class	Feature	Description
DataAttribute	Inherits from *Component*; Sub class *ReportingYear*, *StartDay*	A characteristic of an object or entity.

THIS IS AN ALTERNATE WAY OF CREATING TABLES THAT IS MORE CUMBERSONE BUT ALLOWS FOR MUCH MORE FLEXIBILITY SUCH AS MULTI-LINE AND LISTS ETC.

Table 1.2: The general table 2

Class	Feature	Description
StructureUsage	Feature	See "SDMX Base".
DataflowDefinition	Inherits from StructureUsage	Abstract concept (i.e. the structure without any data) of a flow of data that providers will provide for different reference periods.

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

Data Set – Relationship View

Context

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

Class Diagram

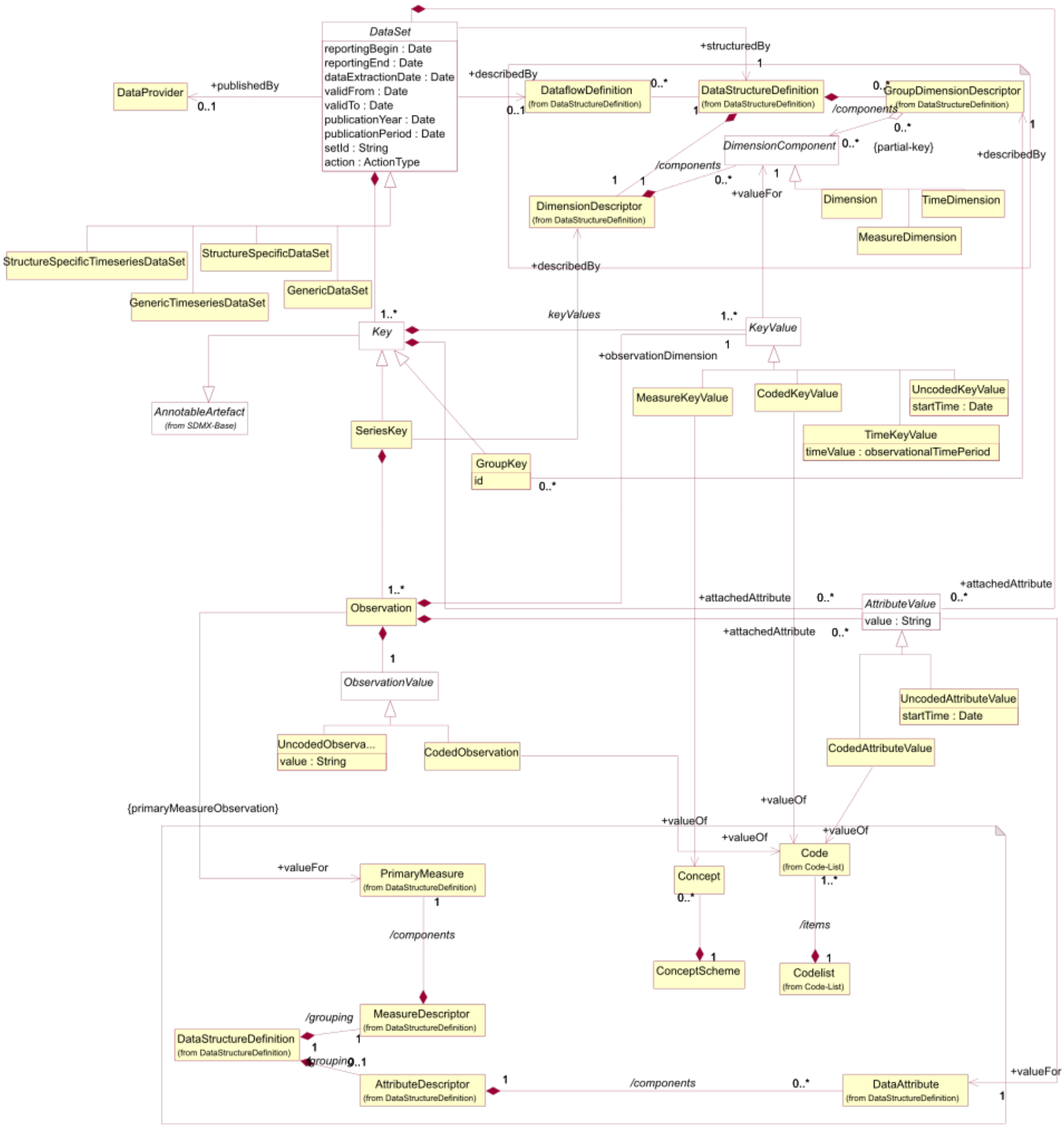

Figure 26 Class Diagram of the Data Set

Explanation of the Diagram

Narrative – Data Set

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

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

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

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

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

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

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

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

The ObservationValue can be coded - this is the CodedObservation – or it can be uncoded – this is the UncodedObservation.

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

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

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

Definitions

Class	Feature	Description
DataSet	Abstract Class Sub classes GenericDataSet StructureSpecificDataSet GenericTime SeriesDataSet StructureSpecificTime SeriesDataSet	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)
	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

continues on next page

Table 1.3 – continued from previous page

GenericDataSet		A data format structure that is able to contain data corresponding to any Data Structure Definition.
StructureSpecific		A data format structure that contains data corresponding to one specific Data Structure Definition.
DataSet		
GenericTimeseries		A data format structure that is able to contain timeseries data corresponding to any Data Structure Definition.
DataSet		
StructureSpecific		A data format structure that contains timeseries data corresponding to one specific Data Structure Definition.
TimeseriesDataSet		
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.
	+attachedAttribute	Association to the Attribute Values relating to the Series Key or Group Key.
KeyValue	Abstract class Sub classes MeasureKeyValue TimeKeyValue CodedKeyValue UncodedKeyValue	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.
MeasureKeyValue	Inherits from *KeyValue*	The value of the Measure Dimension component of the key. The value is the Concept to which this class is associated.

continues on next page

Table 1.3 – continued from previous page

	+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.
TimeKeyValue	Inherits from *KeyValue*	The value of the Time Dimension component of the key.
CodedKeyValue	Inherits from *KeyValue*	The value of a coded component of the key. The value is the Code to which this class is associated.
	+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 *KeyValue*	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 Key	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 Key	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.
	+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.

continues on next page

Table 1.3 – continued from previous page

	+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”.
ObservationValue	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.
AttributeValue	Abstract class Sub classes *UncodedAttributeValue* *CodedAttributeValue*	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.
UncodedAttribute	Inherits from	An attribute value that has a text value.
Value	*AttributeValue*	
	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).
CodedAttribute	Inherits from	An attribute that takes it value from a Code in Code list.
Value	*AttributeValue*	

continues on next page

Table 1.3 – continued from previous page

	+value	Association to the Code that is the value of the Attribute Value. 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.

1.2.7 Cube

Context

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

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

Support for the Cube in the Information Model

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

- The HierachicalCodelist defines the (often complex) hierarchies of codes
- If required, the StructureSet can
  - group DataStructureDefinition that describe the cube
  - provide a mapping mechanism between the codes in the flat code lists used by the DataStructureDefinition and a HierarchicalCodelist where the HierarchicalCodelist uses code lists that are not used in the DataStructureDefinition

1.2.8 Metadata Structure Definition and Metadata Set

Context

The SDMX metamodel allows metadata:

1. To be exchanged without the need to embed it within the object that it is describing.
2. To be stored separately from the object that it describes, yet be linked to it (for example, an organisation has a metadata repository which supports the dissemination of metadata resulting from metadata requests generated by systems or services that have access to the object for which the metadata pertains. This is common in web dissemination where additional metadata is available for viewing (and eventually downloading) by clicking on an "information" icon next to the object to which the metadata is attached).

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

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

- metadata structure definition which has the following components:
  - the object types to which the metadata are to be associated (attached)
  - the components that, together, comprise a unique key of the object type to which the metadata are to be associated
  - the reporting structure comprising the metadata attributes that can be attached to the various object types (these attributes can be structured in a hierarchy), together with any constraints that may apply (e.g. association to a code list that contains valid values for the attribute when reported in a metadata set)
- the metadata set, which contains reported metadata

Inheritance

Introduction

As with the Data Structure Definition Structure, 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. The diagram below shows the full inheritance tree for the classes concerned with the MetadataStructureDefinition and the MetadataSet.

There are very few additional classes in the MetadataStructureDefinition package that do not themselves inherit from classes in the SDMX Base. In other words, the SDMX Base gives most of the structure of this sub model both in terms of associations and in terms of attributes. The relationship diagrams shown in this section show clearly when these associations are inherited from the SDMX Base (see the Appendix "A Short Guide to UML in the SDMX Information Model" to see the diagrammatic notation used to depict this). It is important to note that SDMX base structures used for the MetadataStructureDefinition are the same as those used for the DataStructure-Definition and so, even though the usage is slightly different, the underlying way of defining a MetadataStructure-Definition is similar to that used for defining a DataStructureDefinition.

Class Diagram - Inheritance

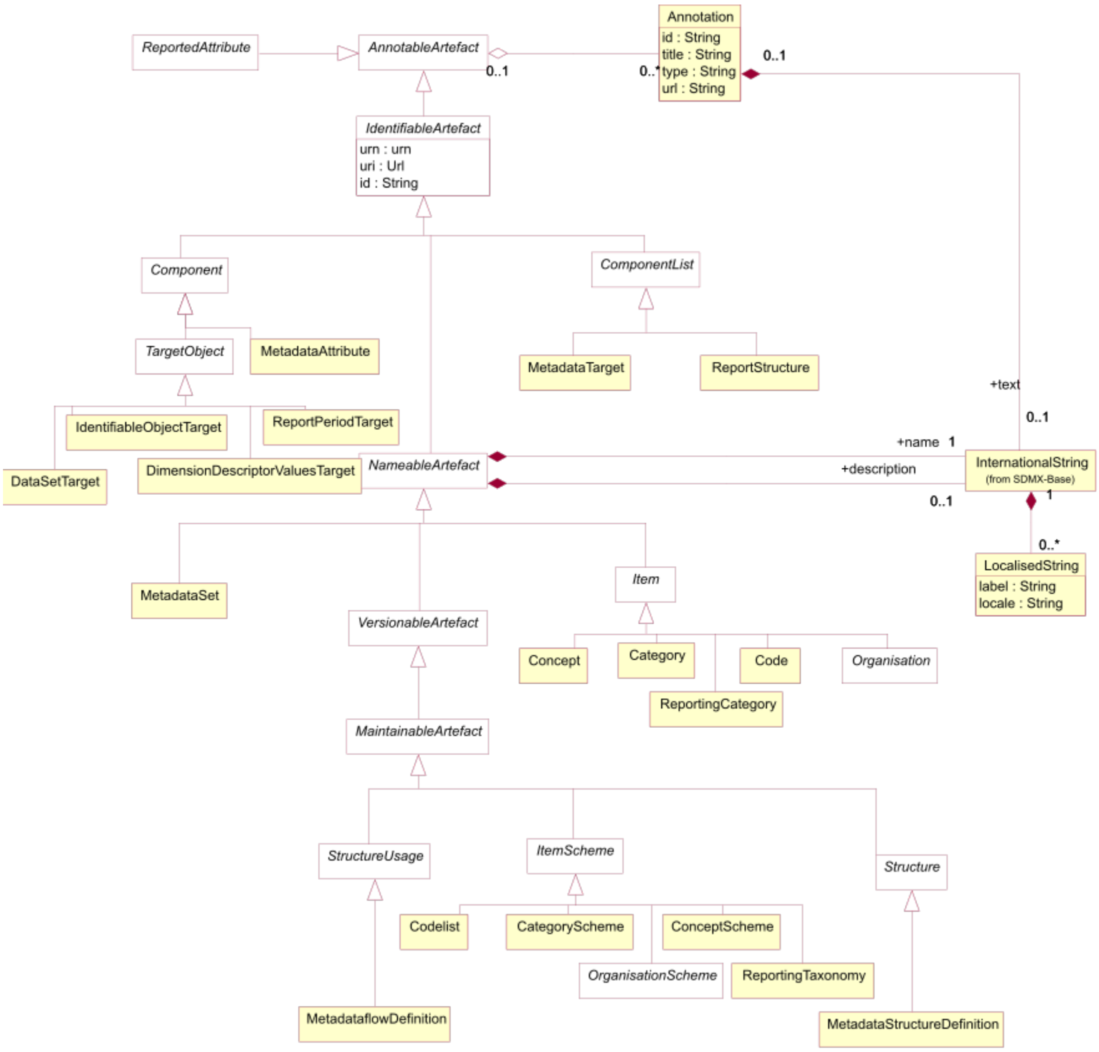

Figure 27: Inheritance class diagram of the Metadata Structure Definition

Explanation of the Diagram

Narrative

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

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

- *StructureUsage* (concrete class is MetadataflowDefinition)
- *Structure* (concrete class is MetadataStructureDefinition)

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

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

- MetadataTarget
- ReportStructure

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

- *Sub Classes of TargetObject*
- *MetadataAttribute*

Metadata Structure Definition

Introduction

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

Structures Already Described

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

- CategoryScheme
- ConceptScheme
- Codelist
- *OrganisationScheme*
- Reporting Taxonomy

Class Diagram – Relationship

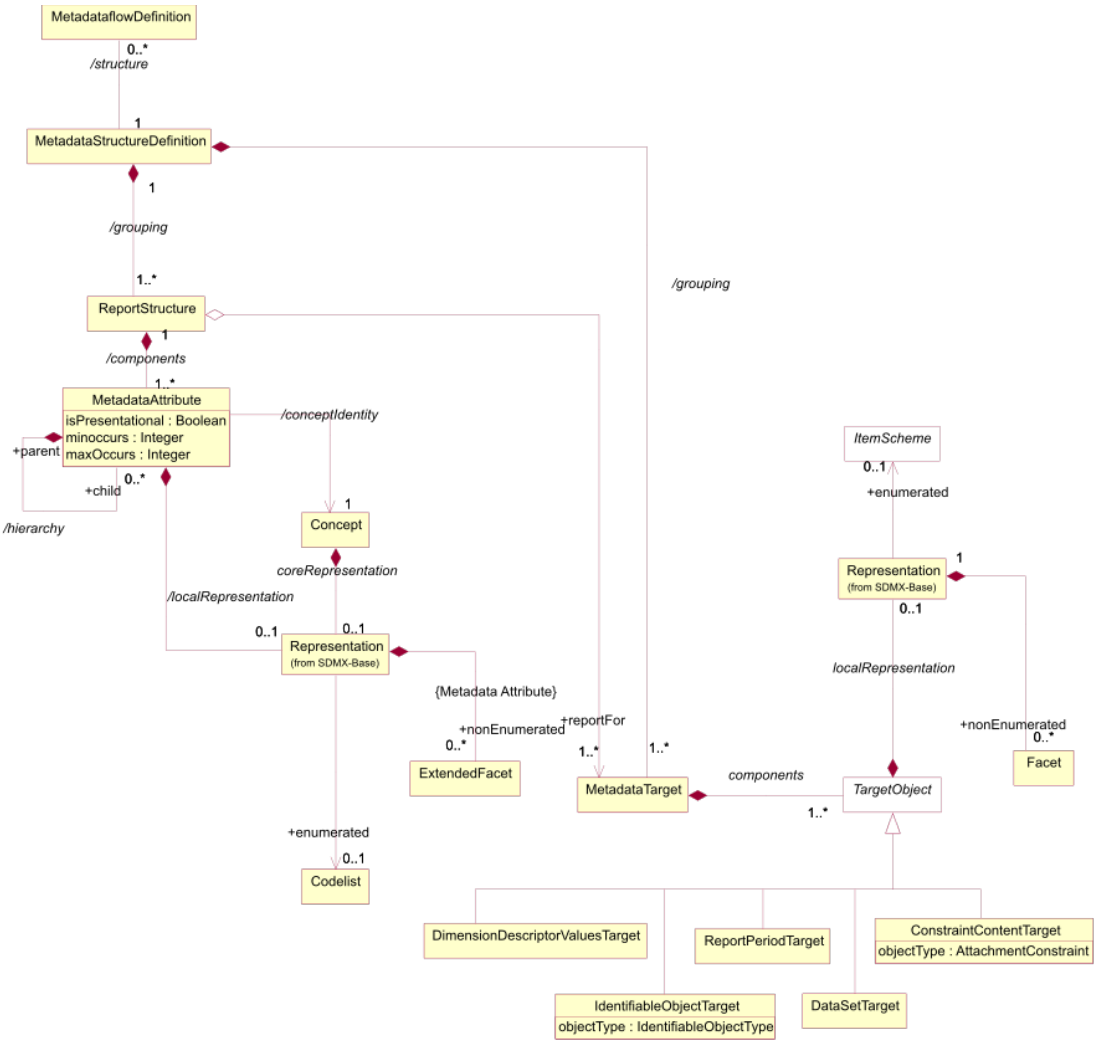

Figure 28: Relationship class diagram of the Metadata Structure Definition

Explanation of the Diagram

Narrative

In brief a MetadataStructureDefinition (MSD) defines:

- 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 *MetadataAttribute*s that can be associated with the object type identified in the referenced MetadataTargets, and hierarchical structure of the attributes

The MetadataTarget comprises one or more *TargetObject*s. The combination of *TargetObject*s identifies a specific object type to which metadata can be attached in a MetadataSet.

The *TargetObject* is one of the following:

- DimensionDescriptorValuesTarget - this allows the specification of a full or partial key (as used in a dataset) to be specified in a MetadataSet as the target object
- IdentifiableObjectTarget – this defines a specific object type, which can be any IdentifiableArtefact
- DataSetTarget – this specifies that the target object is a DataSet
- ReportPeriodTarget - this specifies that the report period must be present in the MetadataSet
- ConstraintContentTarget – this specifies that target object is the content of an AttachmentConstraint i.e. the part of the data set or metadata set identified by the content of an AttachmentConstraint

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

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

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

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

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

It can be seen from this that the specification of the object types to which a *MetadataAttribute* can be attached is indirect: the *MetadataAttributes* are defined in a ReportStructure which itself is attached to one or more MetadataTarget and the actual object is identified by the *TargetObject*s comprising the MetadataTarget. This gives a flexible mechanism by which the actual object types need not be defined in concrete terms in the model, but are defined dynamically in the MetadataStructureDefinition, in much the same way as the keys to which data observation are "attached" in a DataStructureDefinition. In this way the MetadataStructureDefinition can be used to define any set of *MetadataAttribute*s and any set of object types to which they can be attached.

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

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

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

Definitions

Class	Feature	Description
StructureUsage		See “SDMX Base”.
Metadataflow Definition	Inherits from: *StructureUsage*	Abstract concept (i.e. the structure without any metadata) of a flow of metadata that providers will provide for different reference periods.
	/structure	Associates a Metadata Structure Definition.
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 *ComponentList*	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.
TargetObject	Abstract Class Sub Classes DimensionDescriptorValuesTarget 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.
DimensionDescriptorValuesTarget	Inherits from *TargetObject*	The target object is the key of a data series.
IdentifiableObject Target	Inherits from *TargetObject*	The target object is a specified object type.
	objectType	Identifies the object type.
DataSetTarget	Inherits from *TargetObject*	The target object is a Data Set.
ReportPeriodTarget	Inherits from *TargetObject*	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 *TargetObject*	The target object is the data or reference metadata that is identified in the content of an Attachment Constraint.
ReportStructure	Inherits from: *ComponentList*	Defines a set of concepts that comprises the Metadata Attributes to be reported.
	/components	An association to the Metadata

Metadata Set

Class Diagram

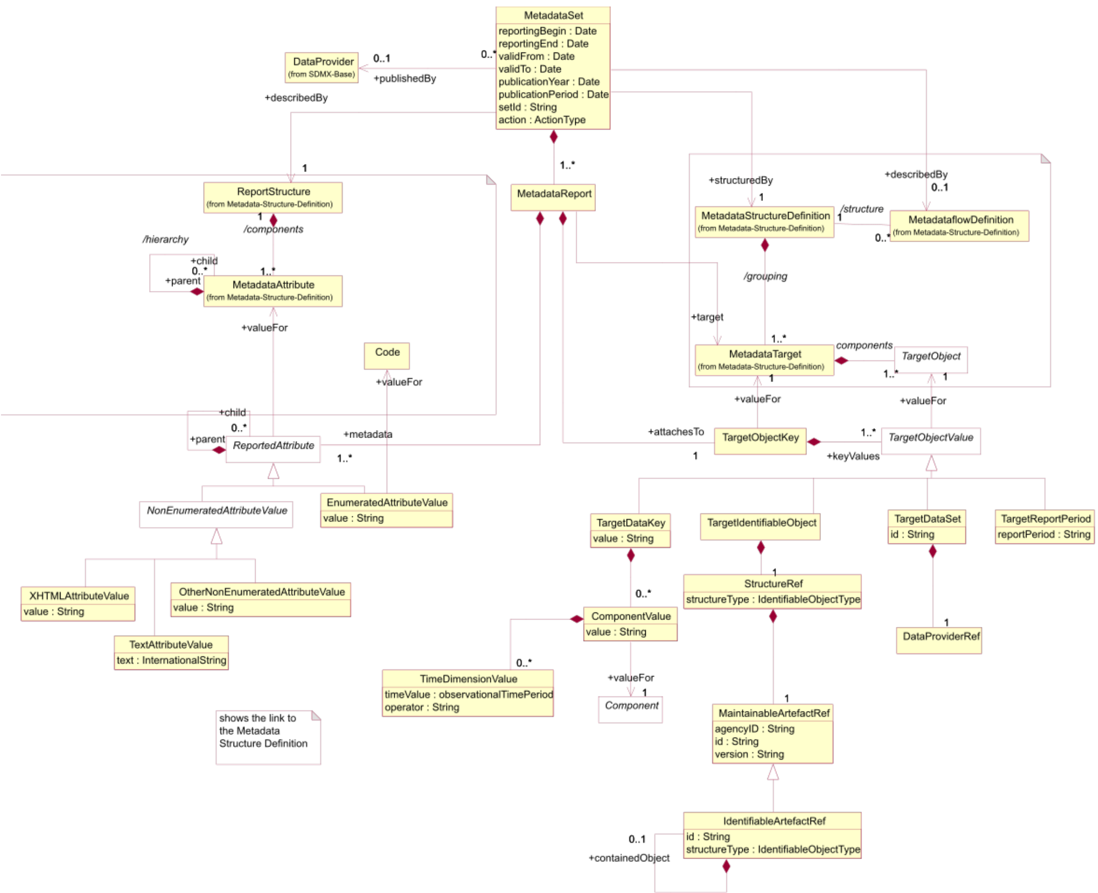

Figure 29: Relationship Class Diagram of the Metadata Set

Explanation of the Diagram

Narrative

Note that the MetadataSet must conform to the MetadataStructureDefinition associated to the MetadataflowDefinition for which this MetadataSet is an "instance of metadata". Whilst the model shows the association to the classes of the MetadataStructureDefinition, this is for conceptual purposes to show the link to the MetadataStructureDefinition. In the actual MetadataSet as exchanged there must, of course, be a reference to the MetadataStructureDefinition and the ReportStructure, and optionally a MetadataflowDefinition, but the MetadataStructureDefinition is not necessarily exchanged with the metadata. Therefore, the MetadataStructureDefinition classes are shown in the grey areas, as these are not a part of the MetadataSet itself.

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

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

The identified ReportStructure specifies which MetadataAttributes are expected as *ReportedAttribute*s. The identified MetadataTarget specifies the expected content of the TargetObjectKey i.e. it specifies the information required to identify the object for which the *ReportedAttribute*s are reported.

The TargetObjectValue can be one of:

- TargetDataKey – this can contain:
  - a SeriesKey (set of dimension values)
  - a SeriesKey plus a value or values (giving time range) for the TimeDimension (TimeDimensionValue)
  - a value of values for the TimeDimension
- TargetIdentifiableObject -this identifies any identifiable object (which includes both Maintainable and Identifiable objects
- TargetDataSet – this identifies a DataSet
- TargetReportPeriod – this specifies the report period for the Report

A simple text value for the *ReportedAttribute* uses the *NonEnumeratedAttributeValue* sub class of *ReportedAttribute* whilst a coded value uses the EnumeratedAttributeValue sub class.

The *NonEnumeratedAttributeValue* can be one of:

- XHTMLAttributeValue – the content is XHTML
- TextAttributeValue – the content is textual and may contain the text in multiple languages
- OtherNonEnumeratedAttributeValue – the content is a string value that must conform to the Representation specified for the MetadataAttribute in the MetadataStructureDefinition for the relevant ReportStructure

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

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 ebd 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.

continues on next page

Table 1.4 – continued from previous page

	setId	Provides an identification of the metadata set.
	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.
	+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.
TargetObjectValue	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.

continues on next page

Table 1.4 – continued from previous page

	+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 *TargetObjectValue*	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.
TargetIdentifiable	Inherits from	Specifies the identification of an Identifiable object.
Object	*TargetObjectValue*	
StructureRef		Contains the identification of an Identifiable object.
	structureType	The object type of the target object.
Maintainable		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.
ArtefactRef		
Identifiable		
ArtefactRef		
	+containedObject	Association to a contained object in a hierarchy of Identifiable Objects such as a Transition in a Process Step.
TargetDataSet	Inherits from *TargetObjectValue*	Contains the identification of a Data Set
TargetReportPeriod	Inherits from *TargetObjectValue*	Contains the period covered by the Metadata Report.
ReportedAttribute	Abstract class Sub classes are: NonEnumeratedAttributeValue EnumeratedAttributeValue	The value for a Metadata Attribute.

continues on next page

Table 1.4 – continued from previous page

	+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.
NonEnumerated AttributeValue	Inherits from ReportedAttribute Sub class: XHTMLAttributeValue TextAttributeValue OtherNonEnumerated AttributeValue	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 Attribute-Value		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.
EnumeratedAttributeValue	Inherits from MetadataAttributeValue	The content of a Reported Attribute that is taken from a Code in a Code list.
	value	The Code value of the Reported Attribute.

continues on next page

Table 1.4 – continued from previous page

	+value	Association to a Code in the Code list specified in the Representation of the Metadata Attribute for which this Reported Attribute is the value 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.

1.2.9 Hierarchical Code List

Scope

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

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

The principal features of the Hierarchical Codelist are:

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

Inheritance

Class Diagram

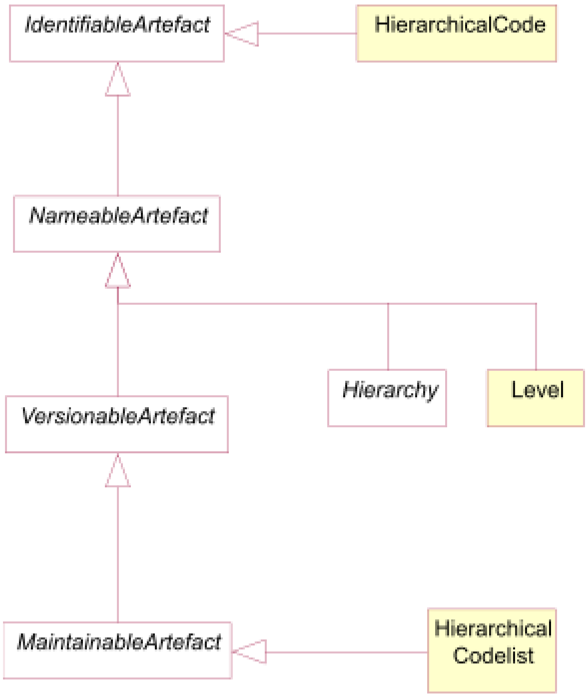

Figure 30: Inheritance class diagram for the Hierarchical Codelist

Explanation of the Diagram

Narrative

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

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

Definitions

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

Relationship

Class Diagram

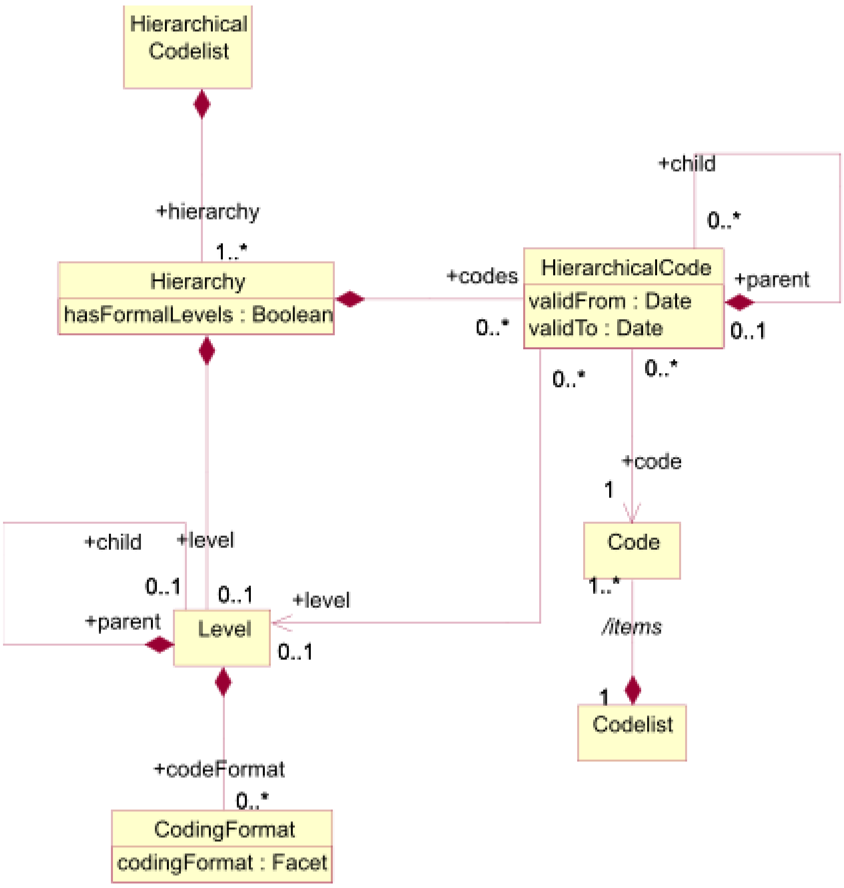

Figure 31: Relationship class diagram of the Hierarchical Code Scheme

Explanation of the Diagram

Narrative

The basic principles of the HierarchicalCodelist are:

1. The HierarchicalCodelist is a specification of the Codes comprising the scheme and the specification of the structure of the Codes in the scheme in terms of one or more *Hierarchy*.
2. The Codes in the HierarchicalCodelist are not themselves a part of the scheme, rather they are references to Codes in one or more external Codelists.
3. Any individual Code may participate in many Hierarchys, in order to give structure to the Hierarchical-Codelist.

4. The Hierarchy of Codes is specified in HierarchicalCode. This references the Code and its immediate child HierarchicalCodes.

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

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

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

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

Definitions

Class	Feature	Description
Hierarchical-Code	Inherits from:	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.
list	*MaintainableArtefact*	
	+hierarchy	Association to Hierarchies of Codes.
Hierarchy	Inherits from: *NameableArtefact*	A classification structure arranged in levels of detail from the broadest to the most detailed level.
	hasFormalLevels	If “true” this indicates a hierarchy where the structure is arranged in levels of detail from the broadest to the most detailed level. If “false” this indicates a hierarchy structure where the items in the hierarchy have no formal level structure.
	+codes	Association to the top-level Hierarchical Codes in the Hierarchy.
	+level	Association to the top Level in the Hierarchy.
Level	Inherits from *NameableArtefact*	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. In a “value based’ hierarchy this describes information about the HierarchicalCodes at the specified nesting level.
	+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
	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.

1.2.10 Structure Set and Mappings

Scope

A StructureSet allows components in one structure to be mapped to components in another structure of the same type. In this context the term "structure" is used loosely to include types of *ItemScheme*, types of *Structure*, and types of *StructureUsage*. The allowable structures that can be mapped, and the components that can be mapped within these structures are:

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	Hierachical 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

The StructureSet can contain one or more "maps" and can define related structures (via the association +relatedStructure) which group related DataStructureDefinitions, MetadataStructureDefinitions, DataflowDefinintions, MetadataflowDefinintions.

Structure Set

Class Diagram – Inheritance

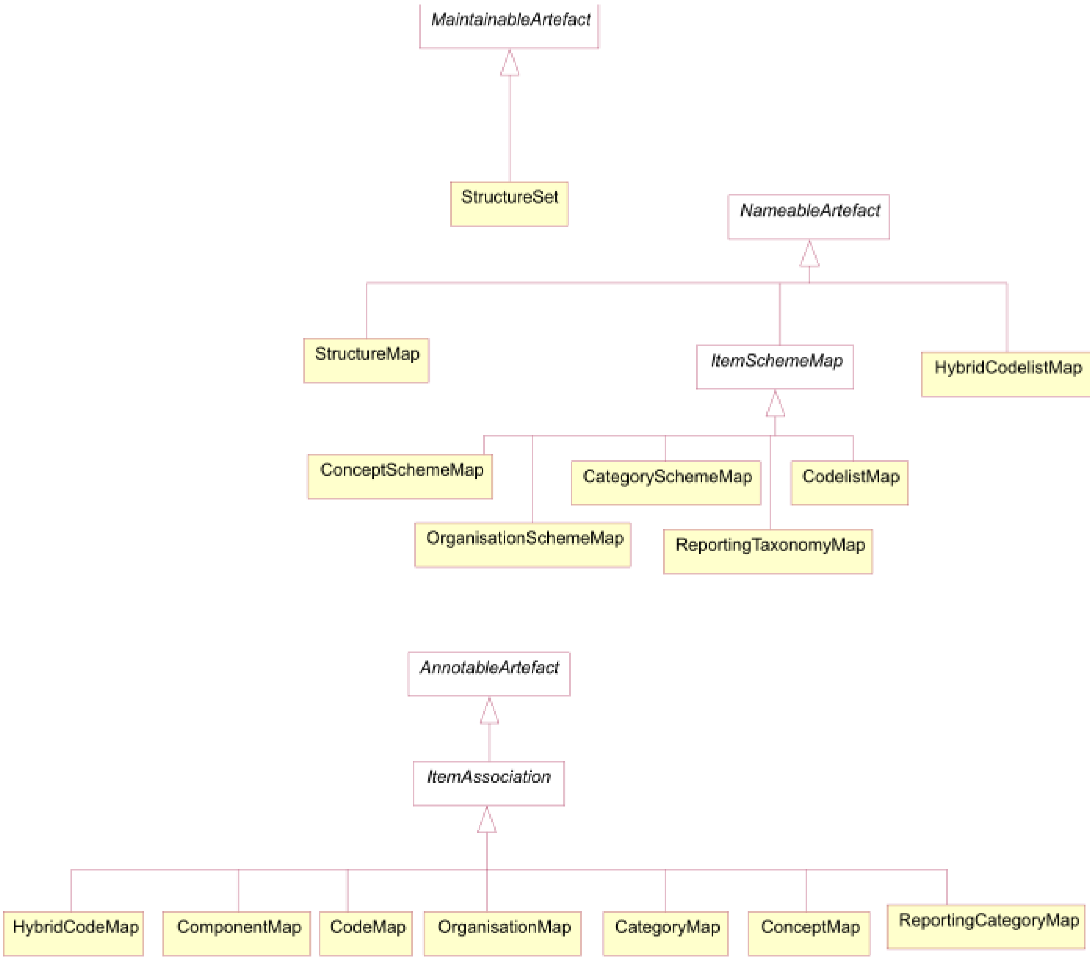

Figure 32: Inheritance Class Diagram of the Structure Set

Class Diagram – Relationship

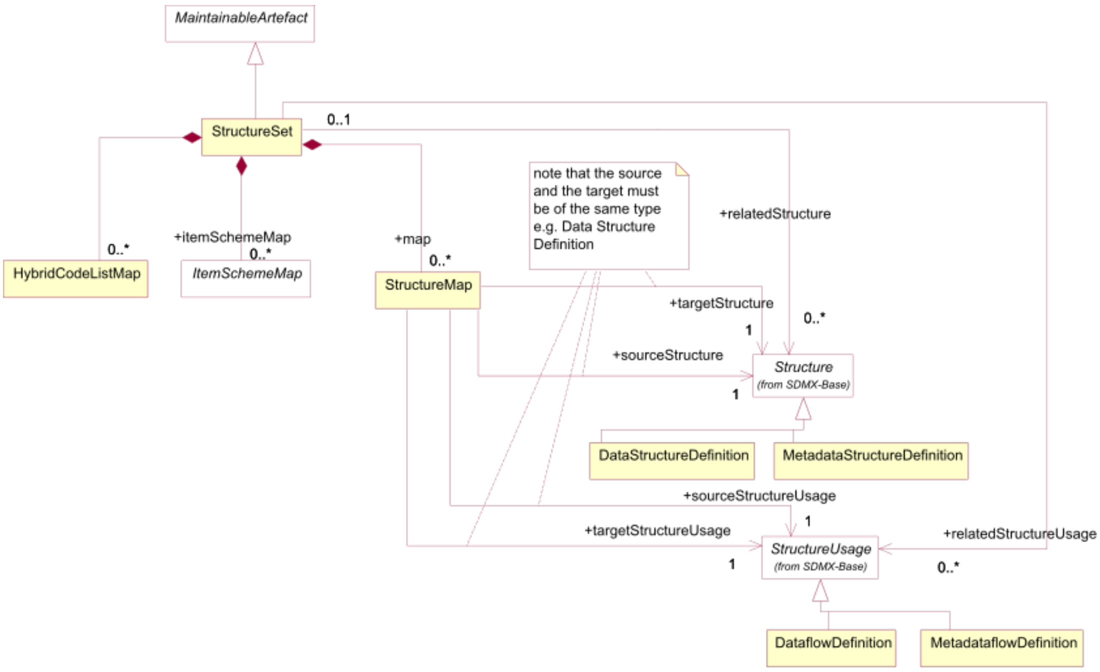

Figure 33: Relationship Class diagram of the Structure Set

Explanation of the Diagram

Narrative

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

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 HierachicalCodelist.

The StructureMap references two *Structure*s or *StructureUsage*s. In concrete terms these references will be to DataStructureDefinitions, MetadataStructureDefinitions, DataflowDefinitions or MetadataflowDefinitions.

Definitions

Class	Feature	Description
StructureSet	Inherits from *MaintainableArtefact*	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 *NameableArtefact*	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.

Structure Map

Class Diagram

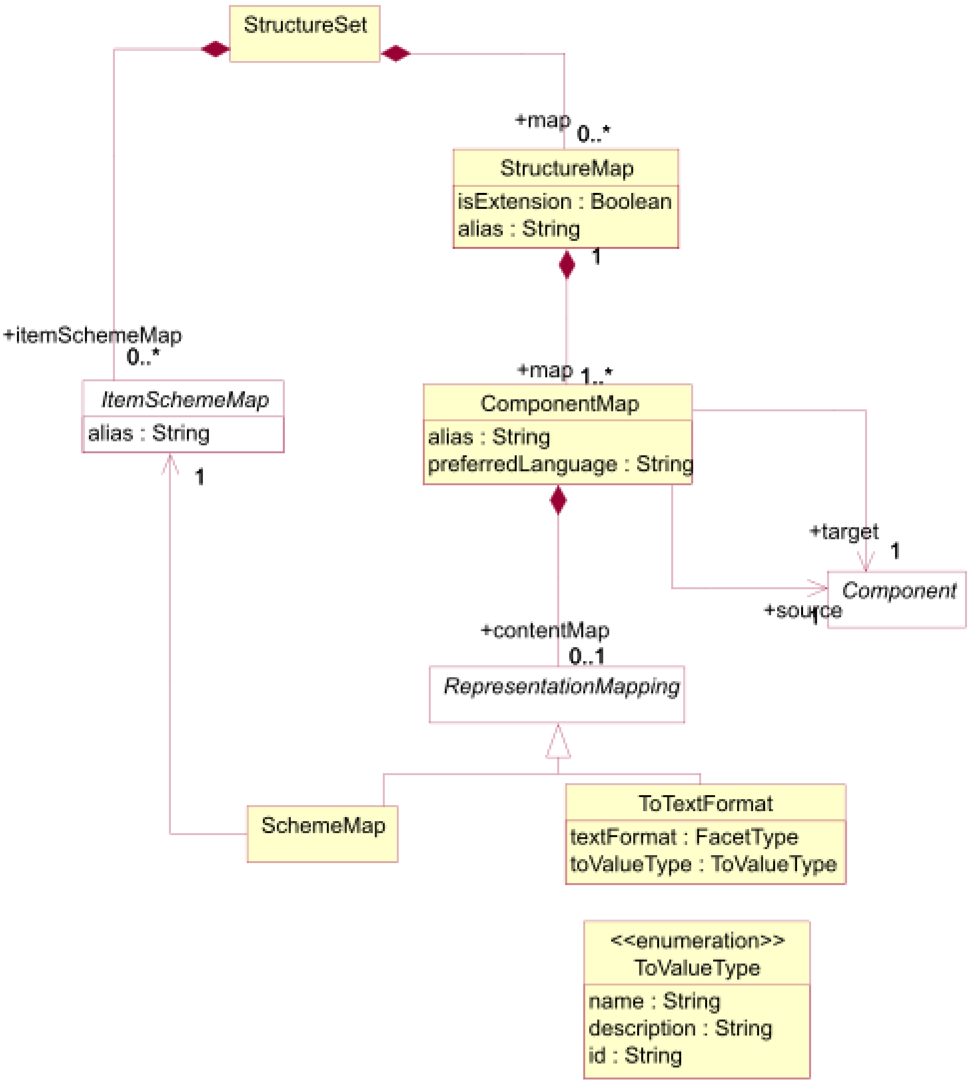

Figure 34: Class diagram of the Structure Map

Explanation of the Diagram

Narrative

The StructureMap contains a set of ComponentMaps, each one indicating equivalence between Components of the referenced *Structure*. ComponentMap has a *RepresentationMapping* which can be one of the concete classes of *ItemSchemeMap* (e.g. for a Dimension this would be a CodelistMap) or ToTextFormat which takes values: id, name, description. This instructs mapping tools to use the id, name or description of a coded component to determine equivalence with an uncoded component's value.

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

Definitions

Class	Feature	Description
StructureMap	Inherits from *NameableArtefact*	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 *AnnotableArtefact*	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.
Representation Mapping	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 *RepresentationMapping*	Associates an Item Scheme Map
ToTextFormat	Inherits from *RepresentationMapping*	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.

Item Scheme Map

Context

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

- ConceptScheme
- CategoryScheme
- *OrganisationScheme*
- Codelist
- ReportingTaxonomy

Class Diagram

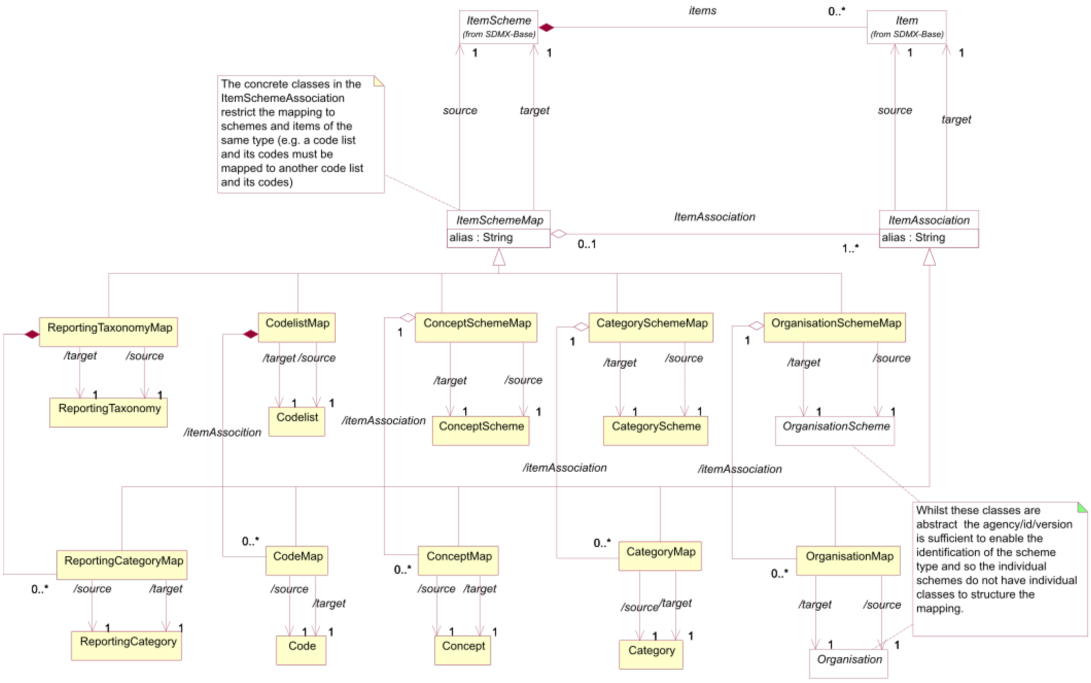

Figure 35: Class diagram of the Item Scheme Map

Explanation of the Diagram

Narrative

Both the ItemSchemeMap and the ItemAssociation inherit from NameableArtefact.

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

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

Definitions

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

continues on next page

Table 1.5 – continued from previous page

	/source	Association to the source Code list.
	/target	Association to the target Code list.
CodeMap	Inherits from *ItemAssociation*	Associates a source and target Code.
	/source	Association to the source Code.
	/target	Association to the target Code.
CategorySchemeMap	Inherits from *ItemSchemeMap*	Associates a source and target Category Scheme.
	/source	Association to the source Category Scheme.
	/target	Association to the target Category Scheme.
CategoryMap	Inherits from *ItemAssociation*	Associates a source and target Category.
	/source	Association to the source Category.
	/target	Association to the target Category.
OrganisationSchemeMap	Inherits from *ItemSchemeMap*	Associates a source and target Organisation Scheme.
	/source	Association to the source Organisation Scheme.
	/target	Association to the target Organisation Scheme.
OrganisationMap	Inherits from *ItemAssociation*	Associates a source and target Organisation.
	/source	Association to the source Organisation.
	/target	Association to the target Organisation.
ReportingTaxonomyMap	Inherits from *ItemSchemeMap*	Associates a source and target Reporting Taxonomy.
	/source	Association to the source Reporting Taxonomy.
	/target	Association to the target Reporting Taxonomy.
ReportingCategoryMap	Inherits from *ItemAssociation*	Associates a source and target Reporting Category.
	/source	Association to the source Reporting Category.
	/target	Association to the target Reporting Category.

Hybrid Codelist Map

Class Diagram

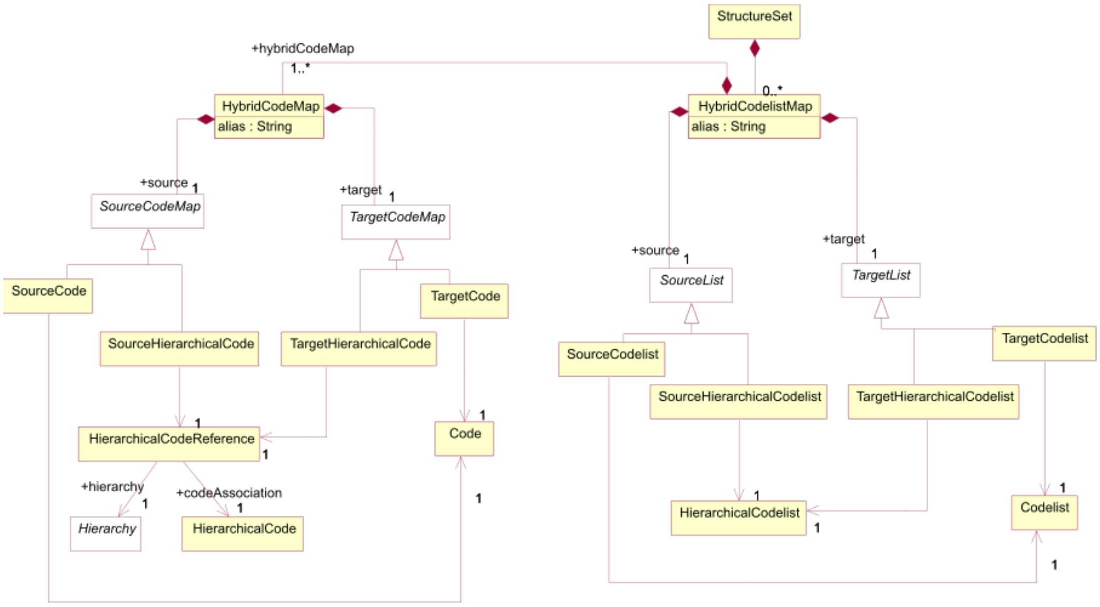

Figure 36: Class diagram of the Hybrid Codelist Map

Explanation of the Diagram

Narrative

The HybridCodelistMap maps the content of a Codelist and a HierachicalCodelist. It contains a mapping of the codes in the two schemes (HybridCodeMap). The HybridCodeMap maps either a Code or HierachicalCode to a Code or HierarchicalCode. The HierarchicalCode is identified by a combination of the Hierarchy and the HierarchicalCode.

Definitions

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

1.2.11 Constraints

Scope

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

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

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

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

These specifications are called *Constraint* in this model.

Inheritance

Class Diagram of Constrainable Artefacts - Inheritance

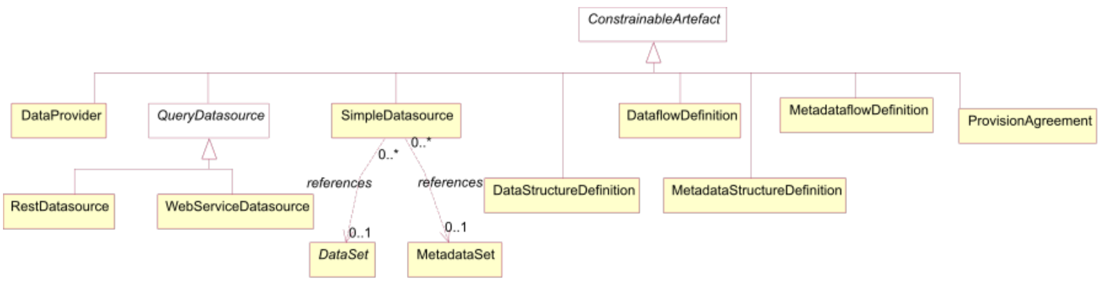

Figure 37: Inheritance class diagram of constrainable and provisioning artefacts

Explanation of the Diagram

Narrative

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

- DataflowDefinition
- ProvisionAgreement
- DataProvider – this is restricted to release calendar
- MetadataflowDefinition
- DataStructureDefinition
- MetadataStructureDefinition
- DataSet

- SimpleDataSource – this is a registered data source where the registration references the actual DataSet or MetadataSet
- QueryDataSource

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

Constraints

Relationship Class Diagram – high level view

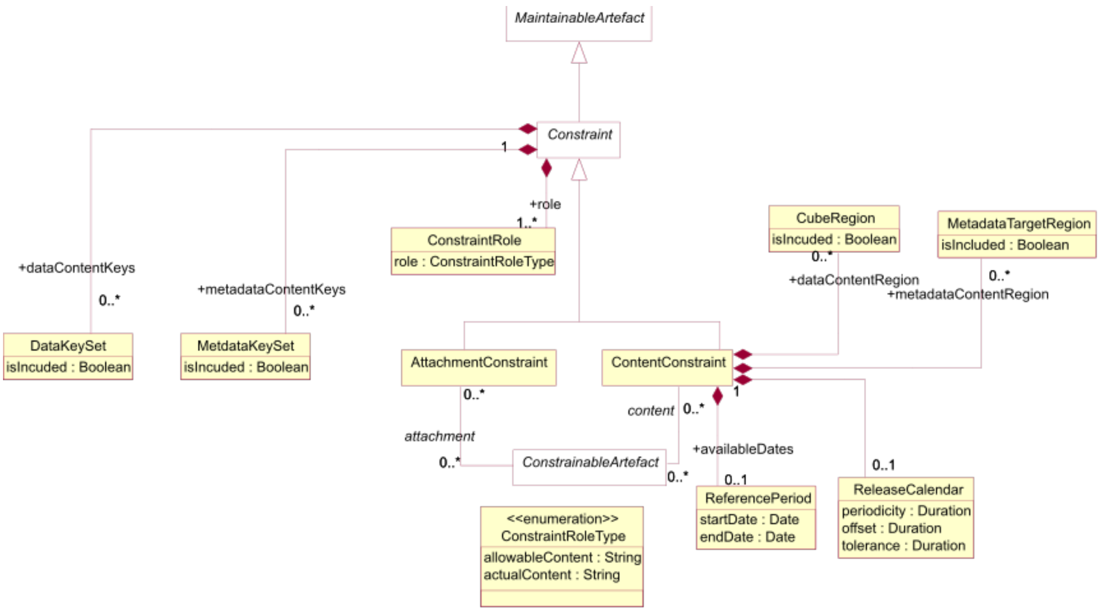

Figure 38: Relationship class diagram showing constraint metadata

Explanation of the Diagram

Narrative

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

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

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

1. ContentConstraint – is used solely as a mechanism to specify either the available set of keys (DataKeySet, MetadataKeySet) or set of component values (CubeRegion, MetadatTargetRegion) in a *DataSource* such as a DataSet or a database (*QueryDatasource*), or the allowable keys that can be constructed from a DataStructureDefinition. Multiple such constraints may be present for a *ConstrainableArtefact*. For instance, there may be a ContentConstraint that specifies the values allowed for the *ConstrainableArtefact* (role is allowableContent) which can be used for validation or for constructing a partial code list, whilst another constraint can specify the actual content of a data or metadata source (role is actualContent).
2. AttachmentConstraint – is used as a mechanism to define slices of the full set of data and to which metadata can be attached in a Data Set or MetadataSet. These slices can be defined either as a set of keys (KeySet) or a set of component values (CubeRegion). There can be many AttachmentConstraints specified for a specific AttachableArtefact.

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

Relationship Class Diagram – Detail

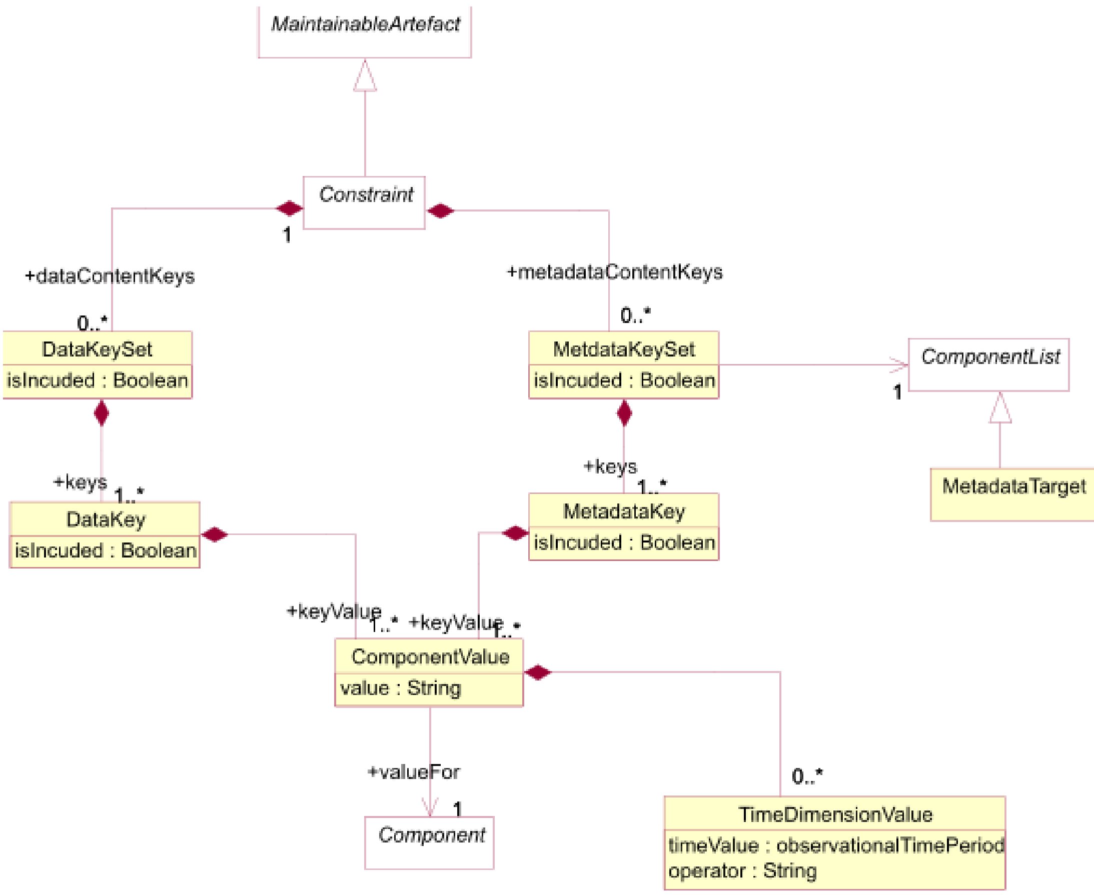

Figure 39: Constraints - Key Set Constraints

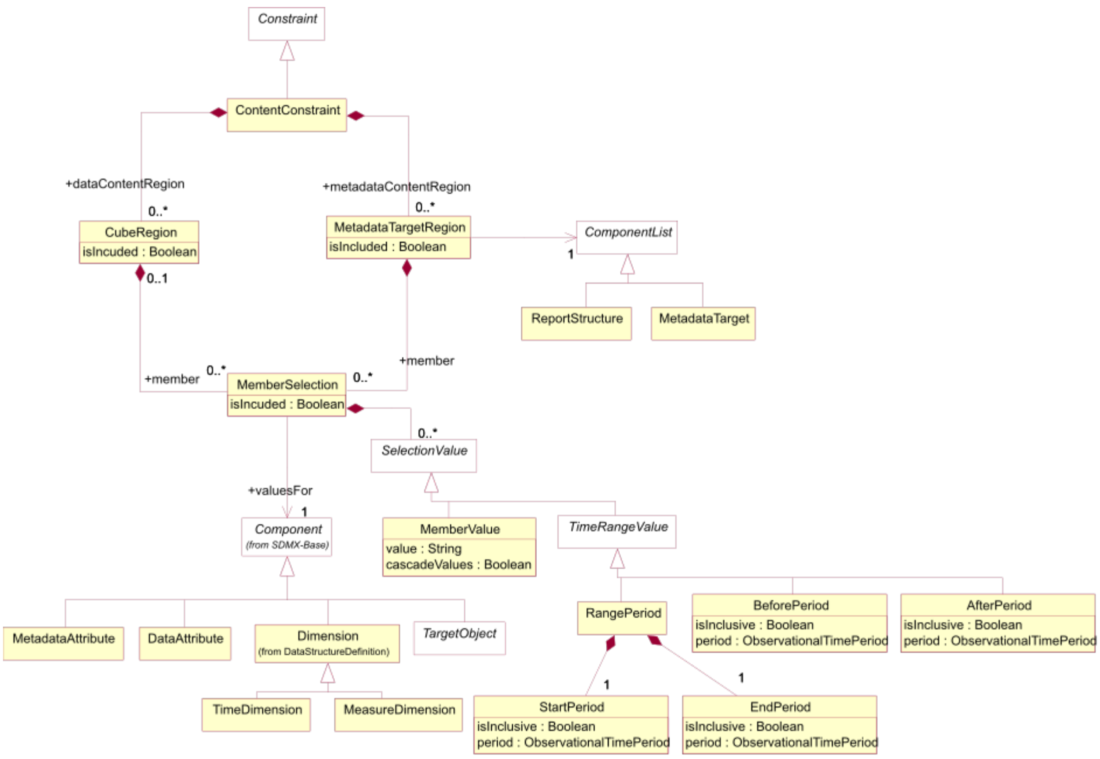

Figure 40: Constraints - Cube Region and Metadata Target Region Constraints

Explanation of the Diagram

A *Constraint* is a *MaintainableArtefact*.

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

1. As a set of keys that can be present in the DataSet (DataKeySet) or MetadataSet (MetadataKeySet). Each DataKey or MetadataKey specifies a number of ComponentValues each of which reference a *Component* (e.g. Dimension, TargetObject). Each ComponentValue is a value that may be present for a *Component* of a structure when contained in a DataSet or MetadataSet. The MetadataKeySet must also identify the MetadataTarget as there can be many of each of these in a MetadataStructureDefinition. For the DataKeySet the equivalent identification is not necessary as there is only one DimensionDescriptor and one AttributeDescriptor.
2. As a set of CubeRegions or MetadataTaregetRegions each of which defines a "slice" of the total structure (MemberSelection) in terms of one or more MemberValues that may be present for a *Component* of a structure when contained in a *DataSet* or MetadataSet.

The difference between (1) and (2) above is that in (1) a complete key is defined whereas in (2) above the "slice" defines a list of possible values for each of the *Component*s but does not specify specific key combinations. In addition, in (1) the association between *Component* and DataKeyValue or MetadataKeyValue is constrained to the components that comprise the key or identifier, whereas in (2) it can contain other component types (such as attributes). The value in ComponentValue.value and MemberValue.value must be consistent with the *Representation* declared for the *Component* in the DataStructureDefinition or MetadataStructureDefinition. Note that in all cases the "operator" on the value is deemed to be "equals". Furthermore, it is possible in a MemberValue to specify that child values (e.g. child codes) are included in the constraint by means of the cascadeValues attribute.

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

Definitions

Class	Feature	Description
Constrainable Artefact	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.
Constraint	Inherits from *MaintainableArtefact* Abstract class. Sub classes are: AttachmentConstraint ContentConstraint	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 Constrainable Artefact is linked.
	+metadataContentKeys	Association to a sub set of Metdata Key Sets (i.e. value combinations) that can be derived from the definition of the Structure to which the Constrainable Artefact is linke

continues on next page

Table 1.6 – continued from previous page

	+dataContentRegion	Association to a sub set of component values that can be derived from the Data Structure Definition to which the Constrainable Artefact is linked.
	+metadataContentRegion	Association to a sub set of component values that can be derived from the Metadata Structure Definition to which the Constrainable Artefact is linked.
ContentConstraint	Inherits from *Constraint*	Defines a Constraint in terms of the content that can be found in data or metadata sources linked to the Constrainable 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	Inherits from	Defines a Constraint in terms of the combination of component values that may be found in a data source, and to which a Constrainable Artefact may be associated in a structure definition.
Constraint	*Constraint*	
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.
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.

continues on next page

Table 1.6 – continued from previous page

MetadataKey		The values of a key in a metadata set.
	isIncluded	Indicates whether the Metdadata 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 Constrainable Artefact is linked.
TimeDimensionValue		The value of the Time Dimension component.
	timeValue	The value of the time period.
	operator	Indicates whether the specified value represents and exact time or time period, or whether the value should be handled as a range. A value of greaterThan or greaterThanOrEqual indicates that the value is the beginning of a range (exclusive or inclusive, respectively). A value of lessThan or lessThanOrEqual indicates that the value is the end or a range (exclusive or inclusive, respectively). 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)
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 Constrainable 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.
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 Constrainable Artefact is linked.

continues on next page

Table 1.6 – continued from previous page

	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 Constrainable Artefact is linked, which defines the valid Representation for the Member Values.
MemberValue		A single value of the set of values for the Member Selection.
	value	A value of the member.
	cascadeValues	Indicates that the child nodes of the member are included in the Member Selection (e.g. child codes)
TimeRangeValue	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 *TimeRangeValue*	The period before which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
AfterPeriod	Inherits from *TimeRangeValue*	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 *TimeRangeValue*	The period from which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
EndPeriod	Inherits from *TimeRangeValue*	The period to which the constrained selection is valid.
	isInclusive	Indication of whether the date is inclusive in the period.
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.

continues on next page

Table 1.6 – continued from previous page

	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 1st and the first release of the data
	tolerance	Period after which the data or metadata may be deemed late.

1.2.12 Data Provisioning

Class Diagram

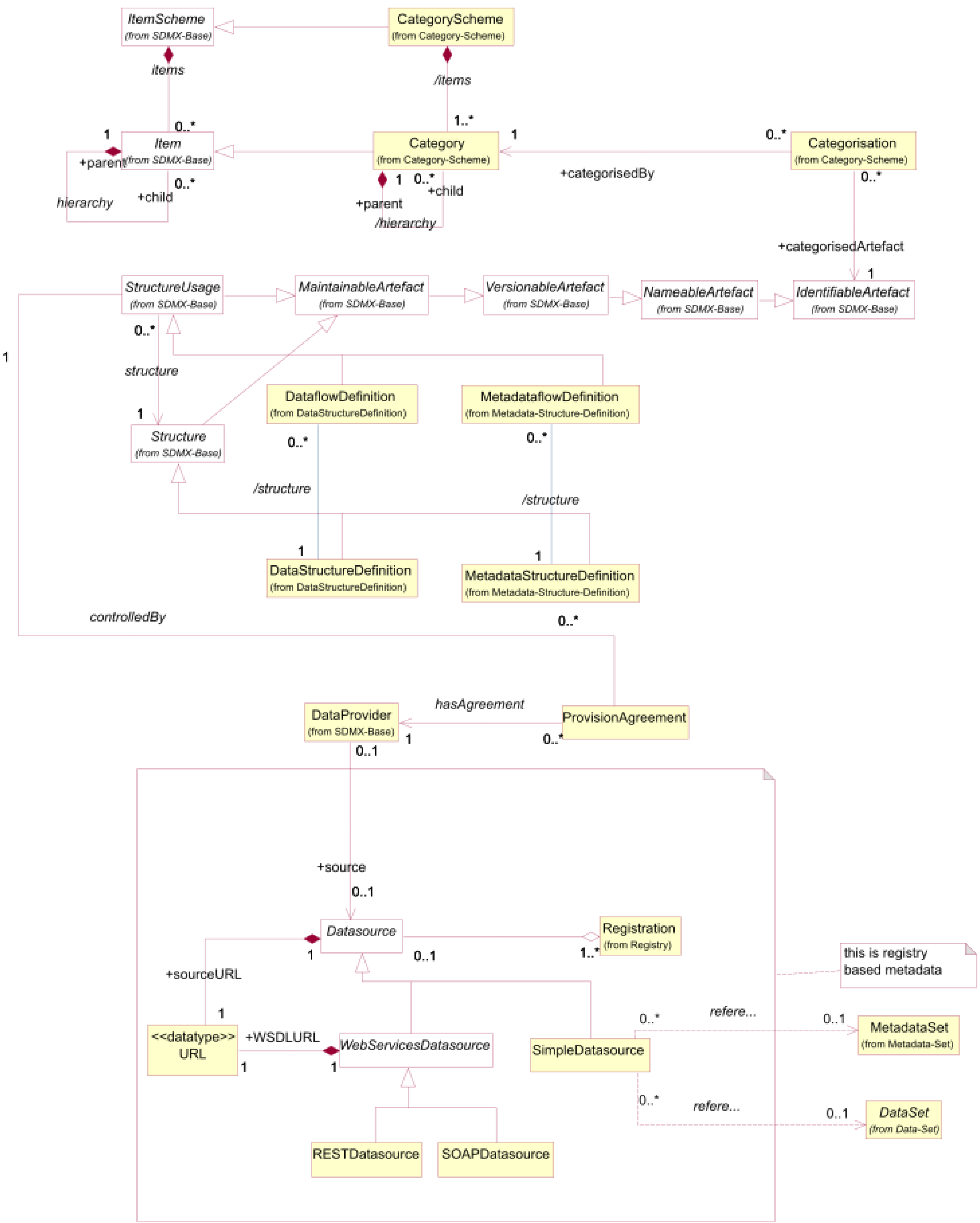

Figure 41: Relationship and inheritance class diagram of data provisioning

Explanation of the Diagram

Narrative

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

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

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

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

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

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

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

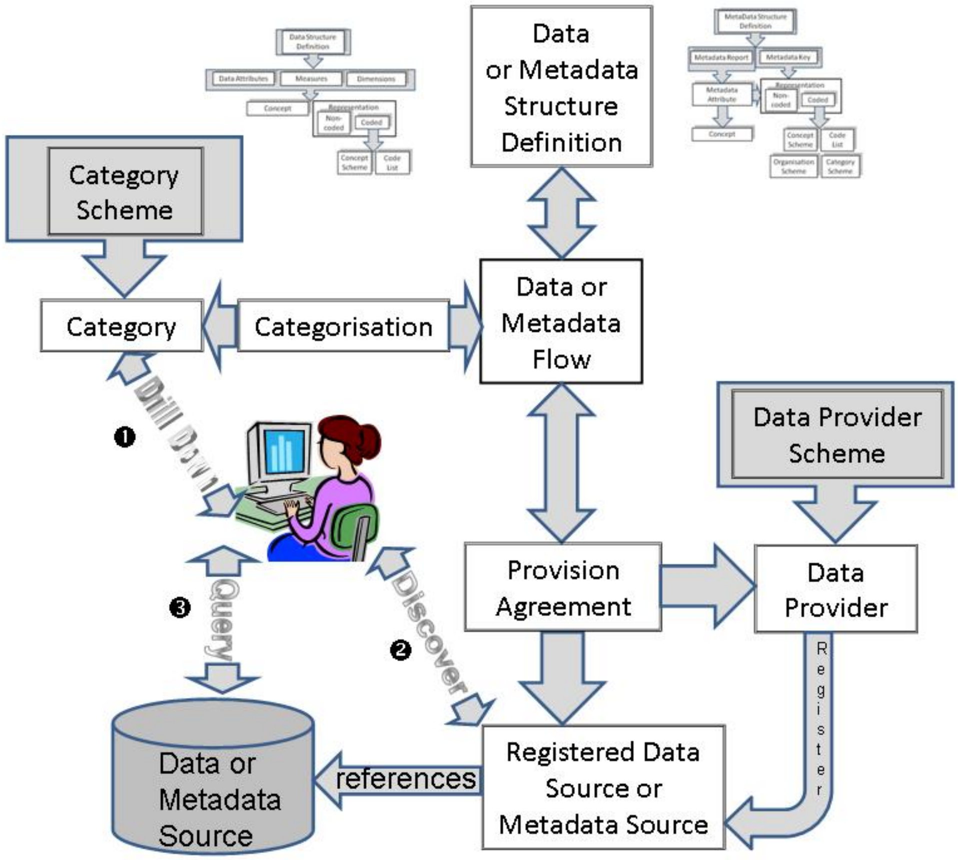

Figure 42: Schematic of the linking of structural metadata to data and metadata registration

Definitions

Class	Feature	Description
StructureUsage	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.
	+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.
Datasource	Abstract class: Sub classes are: SimpleDatasource *WebServices Datasource*	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.
*WebServices	Abstract class:	A data or reference metadata source which can process a data or metadata query.
Datasource*	Inherits from: *Datasource* Sub classes are: RESTDatasource SOAPDatasource	
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 Ser-

1.2.13 Process

Introduction

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

Model – Inheritance and Relationship view

Class Diagram

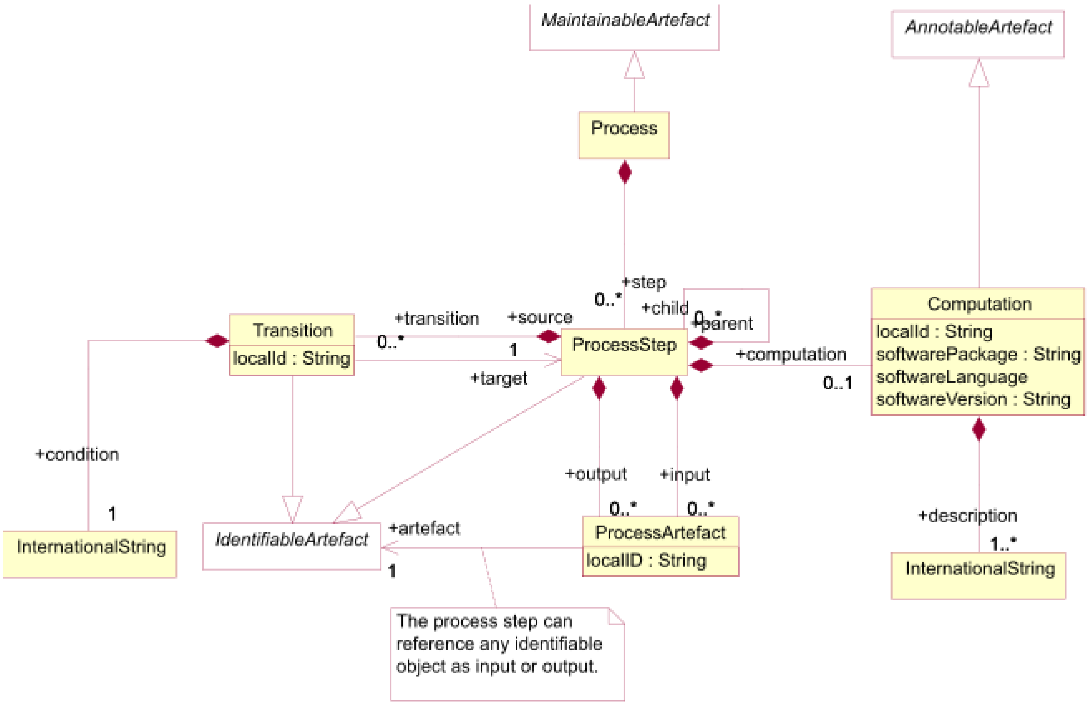

Figure 43: Inheritance and Relationship class diagram of Process and Transitions

Explanation of the Diagram

Narrative

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

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

Definitions

Class	Feature	Description
Process	Inherits from *Maintainable*	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 *IdentifiableArtefact*	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.
Transition	Inherits from *IdentifiableArtefact*	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.

1.2.14 Transformations and Expressions

Scope

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

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

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

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

http://www.omg.org/cwm

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

Model - Inheritance View

Class Diagram

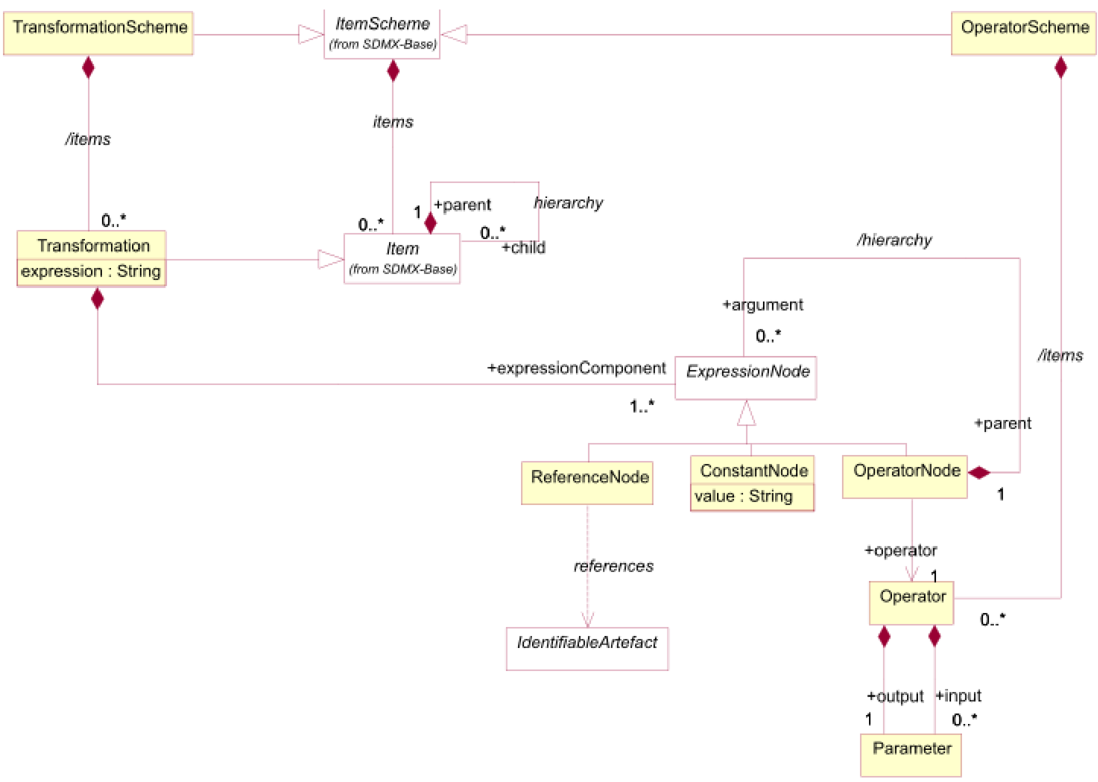

Figure 44: Inheritance and relationship class diagram of transformation classes

Explanation of the Diagram

Narrative

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

1. A TransformationScheme which comprises one or more Transformations.
2. An OperatorScheme which comprises one or more *Operator*s.
3. An ExpressionNodeScheme scheme which contains one or more ExpressionNodes..

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.

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 its Parameters.

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

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

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

Although the model defines the data structures that are used to contain a syntax neutral expression, the model itself does not specify a syntax neutral expression grammar. Alternatively, the function can be described in a text form either as an unstructured explanation of the function, or as a more formal language like BNF[2].

The data structures work as follows:

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

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

The expression of a Transformation (for example, for the Einstein equation calculus, "E = M*(C**2)") may be decomposed in a hierarchy of ExpressionNodes (in the example, "M", "C", "2", *, **). The ExpressionNode can be a ReferenceNode, a ConstantNode or an OperatorNode. The ReferenceNode references an identifiable model artefact (in the example, "M" and "C"). The ConstantNode is by definition a constant value (in the example "2"). The OperatorNode references an Operator in the OperatorScheme (in the example *, **). The Transformation has an association to its component ExpressionNodes.

The hierarchy of the ExpressionNodes conveys the order in which the operators are applied in the expression and is obtained by means of the /hierarchy association of the OperatorNode class, in which the child ExpressionNodes are the arguments of the parent OperatorNode. The child ExpressionNodes must correspond to the formal parameters of the Operator referenced by the parent OperatorNode in the correct sequence. The (child) ExpressionNode can be the result of another operation (that is another OperatorNode) or can be a Constant or can be a reference to an *IdentifiableArtefact* (ReferenceNode). All *IdentifiableArtefacts* in the SDMX-IM have a unique urn comprising the values of the individual objects that identify it. The structure of this urn is defined in the Registry Specification. An example would be the urn of a code which comprises the agency:code-list-id.code-id – an actual example is "urn:sdmx:org.sdmx.infomodel.codelist.Code=TFFS:CL_AREA(1.0).1A".

[2] BNF: Backus Naur Form

Definitions

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

1.2.15 Appendix 1: A Short Guide To UML in the SDMX Information Model

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.

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.

Figure 45 Actor

The **use cas**e 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"

Figure 46 Use case

Figure 47 Actor and use case

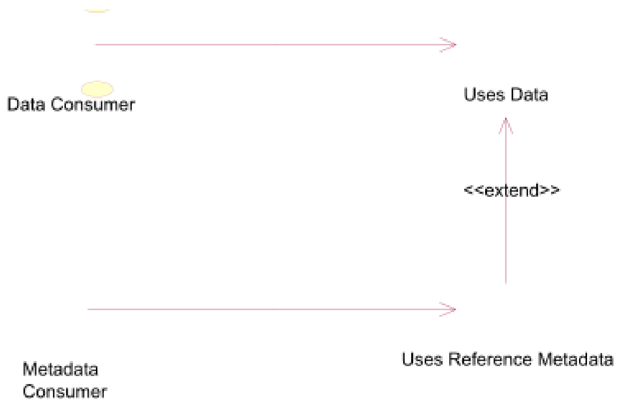

Figure 48 Extend use cases

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

Classes and Attributes

General

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

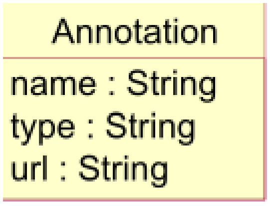

Figure 49 Class and its attributes

Figure 49 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 50 Abstract and concrete classes

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

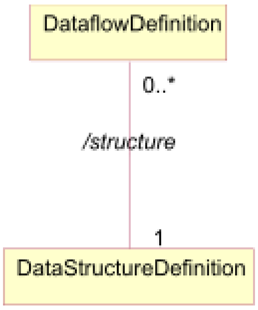

Figure 51 A simple association

Here the DataflowDefinition class has an association with the DataStructureDefinition class. The diagram shows that a DataflowDefinition can have an association with only one DataStructureDefinition (1) and that a DataStructureDefinition can be linked to many DataflowDefinitions (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

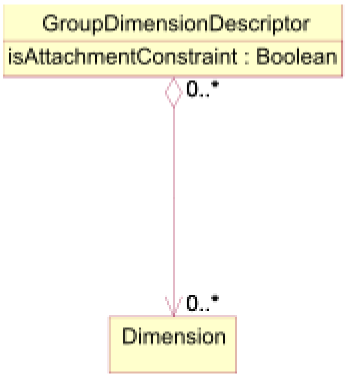

Figure 52: A simple aggregate association

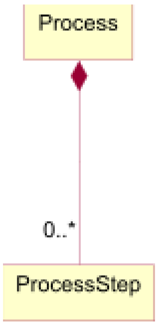

Figure 53 A composition aggregate association

An association with an aggregation relationship indicates that one class is a subordinate class (or a part) of another class. In an aggregation relationship. There are two types of aggregation, a simple aggregation where the child class instance can outlive its parent class, and a composition aggregation where

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

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 54 shows a simple aggregation between CategoryScheme and Category called */items* (this means it is derived from the association between the super classes – in this case between the *ItemScheme* and the *Item,* and another between Category called */hierarchy*.

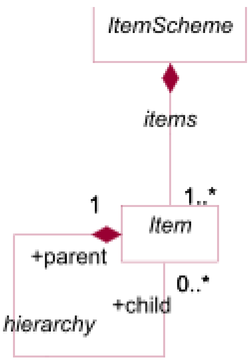

Figure 54 Association names and end names

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

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

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

Navigability

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

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

Figure 55 One way association

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

Inheritance

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

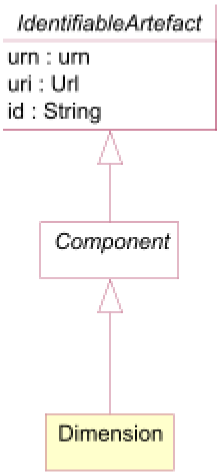

Figure 56 Inheritance

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

Derived association

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

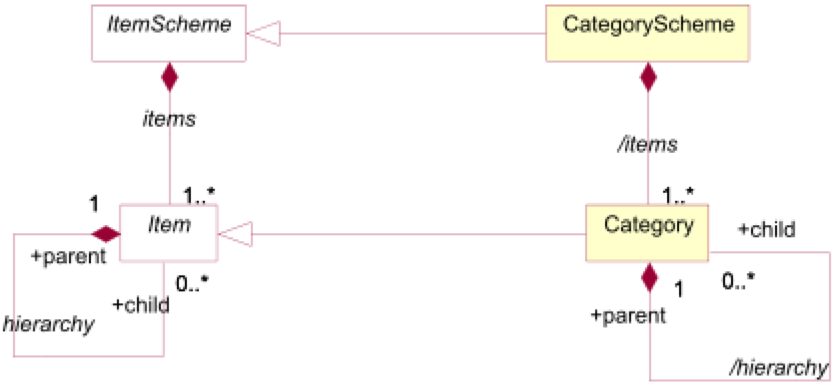

Figure 57 Derived associations