The "Matrix" Model

Unified model for statistical data representation and processing

May 2007

Prepared by :

- *Vincenzo Del Vecchio (Introduction, The Architecture of Models, The Matrix Statistical Metamodel: Introduction, Historicity and other common properties, all of the" description" sections)*
- *Fabio Di Giovanni (The Matrix Statistical Metamodel and the Administration Model: all of the "diagram descriptions: definitions");*
- *Stefano Pambianco (All of the class diagrams, The External References Model, and the Appendix: A short guide to UML notation)*

We thank all the colleagues who put their knowledge at our disposal, made suggestions and revised the document.

The views expressed are those of the authors and do not involve the responsibility of the bank

CONTENTS

Conventions

In plain text the following conventions are used:

- ✓ References (Bibliography can be found at the end of the document) are shown using square brackets [*item_number*]
- ✓ Examples are shown in *italic* font and highlighted using a *box surrounding the text*
- ✓ MOF or UML classes: `ClassName` is shown in `Courier` font;
- ✓ MOF or UML attribute: `AttributeName` is shown in `Courier` font;
- ✓ MOF or UML association: `/AssociationName` is shown in `Courier` and prefixed by a "`/`";
- ✓ MOF or UML Instances: *`Instances`* of a Class are represented with the same ClassName and shown in *Italic* font;
- ✓ The names of the model entities are marked with an initial "M_" (for Matricial) to distinguish them from the similar terms used in the common mathematical language (e.g. M_Element, M_Variable, M_Set, …).

About the conventions adopted in the MOF and UML diagrams, see the guide on UML notation in the appendix.

Introduction

Statistical data are of great importance for the institutional functions of the Bank of Italy (see [3]). The activity of collecting, processing and disseminating statistics is extensive and complex ([1],[2]). For this reason, the Bank has drawn up a managerial, technological and methodological strategy (see [10]).

On a methodological plane, the topic of the representation of statistical data is crucial for the whole statistical activity. Many decades of experience in the field showed that a strong model is a key factor for the success of a statistical information system (SIS). In fact it makes it possible to achieve a powerful automation of the processing system, short time, low cost and reliability in designing and processing statistical information, high adaptability to the evolution of the information requirements, major support to harmonization, information documentation and usability and so on.

The Bank of Italy (BI) statistical information system is based on two main conceptualizations, the first one describing the general architecture of a SIS (conceived as a hierarchy of models),[1] the second one being the generic model devoted to defining the statistical data and the operations to be performed on them: the so called Matrix Model[2].

The general architecture of a SIS, summarized in the first part of this document and built following the trends of research and international organizations (see standards such as ISO and OMG), meets many goals at once, offering a synthetic and high-level vision of the SIS and connecting the different perspectives in which it is usually seen. In fact it makes it possible:

- to give a formal structure to the SIS and decompose it in parts;
- to obtain a self-consistent SIS, i.e. containing not only information but also its definition;[3]
- to make "active" the processing system of the SIS (i.e. software driven by the definitions of the data and the operations on them);
- to harmonize and standardize the terms, definitions and data of the SIS;
- to distinguish roles, fix competencies and assign them.

The Matrix Model emerged during a slow evolution over several decades. It is used for most of the BI statistical surveys and is a fundamental component of the architecture of the SIS of the BI. The principles on which it is based are widely treated in [4].

It is obvious that the Matrix Model is addressed to the administrators and final users of the SIS, and not to EDP specialist. In fact it is built making use of the basic notions of mathematics, set theory, probability theory and descriptive statistics methodology. Hence, its components and terminology are derived from those disciplines, are well known to statisticians and are based on a robust, simple and well proven theorization.

---

1 See [6], [8]

2 The name "Matrix" was coined to mean "having the form of a matrix", owing to the graphic representation used to define the data structure, which is a matrix of rows and columns.

3 See the notion of "infological completeness" [Sundgren];

The Matrix Model, together with a proper design of the processing system, allows the administrators and final users to become largely independent of EDP specialist intervention in managing and using the SIS. In fact the Model is built to be "active", i.e. able to drive the behaviour of the data processing software. Definitions directly given by users are intended to be interpreted and executed by the processing system. In this way, any change in the definitions leads to a corresponding change in data processing.

As a consequence, the system is self-documented, because the definitions that drive the system also document data and the operations performed.

Among the more important features of the Matrix Model, often not supported by commercial EDP products such as data base management systems and data warehouses, we can quote:

- historicity (the model can represent the historical changes of data and operations on them);
- the separate representations of abstractions (pure definitions) and data (measures), based on the basic notions of set theory and mathematics (element, set, variable, function), that makes it possible to manage the trade-off between the independence of different surveys and their coherence and harmonisation;
- the powerful and not redundant representation of hierarchical structures such as classifications and other hierarchies, based on the probability theory notion of "space of events";
- the powerful support for the definition of the data processing algorithms (transformations);
- the declarative language based on the methodology.

The model has been also presented in the Metanet project, although at the time a formalized description of it did not existed (see [5],[6]). Many other contributions derived from the Matrix Model have been made to international projects and works.

The "unified statistical dictionary" of the Bank of Italy, built for the representation of statistical concepts, data and transformations, is also based on the same model.

This document therefore fills a serious documentary gap for this topic and provides a first formal representation of the Matrix Model.

The architecture of models

Statistical data models

The principles underlying statistical data modelling in the Bank of Italy are described in [6]. Parts of that document are also repeated here. The statistical information system is described by different levels (layers) of modeling in a hierarchy in which the model of one level is described in terms of a hierarchically higher model and also describes one or more hierarchically lower models (see Figure 1).

In brief, starting from the reality that has to be described (call it the "zero" level), in the first level we have the data extensions, which are models of parts of the reality, followed, in the second level, by the data definitions, which are models of the data. The third level contains the methods to produce the data definitions, which are models of models of data (metamodels). Finally, the fourth level contains the methods that produce other methods, which are models of the metamodels (meta-metamodels).[4]

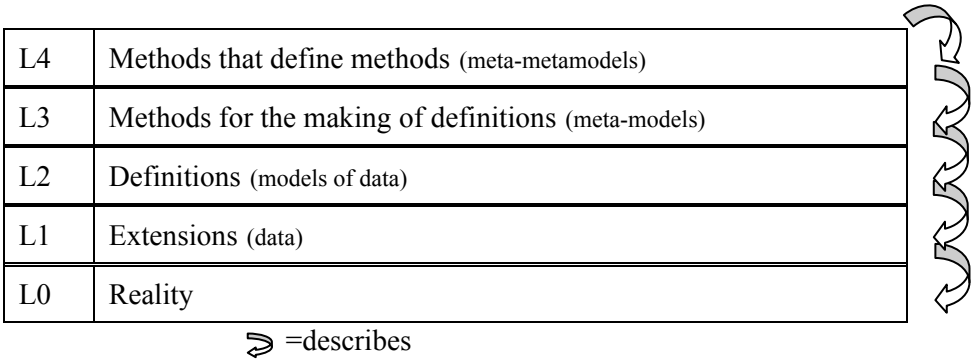


Figure 1. *The Hierarchy of Levels*

The specificity of the statistical field is located at the third level. A **statistical third-level model** is in fact considered the formal representation of a methodology for the statistical description of the reality (that is, a descriptive statistic methodology). A third-level model contains structures able to give a concrete and possibly formal shape to statistical methodological rules. The existence of a "statistical" third level is a consequence of the recognition of the specificity of the statistical methodology with respect to others, such as the EDP ones used for the implementation.[5]

A second-level model can be considered the definition of a specific statistical information segment,[6] that is, the definition of data and operations on them for a specific subject. Therefore, second-level models are specific subject-matter models produced using a certain statistical methodology (that is,

---

[4] Sometimes, in the EDP literature, "models" are called "schemas" and consequently "metamodels" are called "models" and "meta-metamodels" are called "meta-models". In this work, however, the term "schema" of a certain model (of a certain level) is used to indicate a lower level model produced using that model (e.g. a Matrix schema is a second level model produced using the Matrix model, that is a third level model).

[5] This consideration about the "statistical" hierarchy of models leaves temporarily aside the matter of the implementation of the information system, which will be discussed later with regard to EDP environments.

[6] The term is introduced to indicate a self-consistent part of a statistical information system, with an autonomous existence and evolution, such as a survey, a stove pipe, a processing line, etc.

a third-level model). The set of second-level models for a specific SIS (statistical information systems) is also named the "statistical dictionary" of the SIS. As in the general case, at the first level we find extensions of statistical data and at the zero level the reality to be described.

Note that the notion of "data model" (data that are the definition of other data) rather than "metadata" (data that describe other data in some way) is used because the former is more specific. For example, a "quality datum", a datum measuring or reporting the quality of another datum, can be considered a "metadatum" yet it is not the "model" of the original datum.[7]

The ideal situation would be to have only a single third-level methodology, able to model any other kind of statistical segment. In fact harmonization between models in a certain level (the second level in this case) happens to be much simpler if they are defined by means of the same higher level model, that is using the same modeling method. In practice the existence of many competing modeling methodologies cannot always be avoided. The effort to achieve a single statistical third level model drove the Bank of Italy to the construction of a "unified" statistical third level model (the Matrix metamodel, commonly referred to as the "Matrix model" for the sake of simplicity).

The aim of this document is to describe the Matrix statistical metamodel. To do so, a fourth-level model should be used, according to the general schema. Structures suitable to belong to fourth-level models are not specific to statistics, they are more general and can also be used in other fields (such as operational systems). That is to say that a fourth-level model contains structures able to define any kind of methodology and is possibly shared by all of them[8].

As in the third-level case, the ideal situation would be to have only a single fourth-level model, able to model any other kind of third-level methodology. In this document, the Meta Object Facility (MOF) model is used as the meta-metamodel, because it is the standard fourth-level model according to the Object Management Group (OMG).[9] The MOF model contains a subset of the construct used in UML, so MOF schema can be drawn using the notation used in UML.[10] MOF schema with UML notation are therefore used here to describe the Matrix metamodel. The Hierarchy of models used in this document for statistics modelling is displayed in Figure 2.

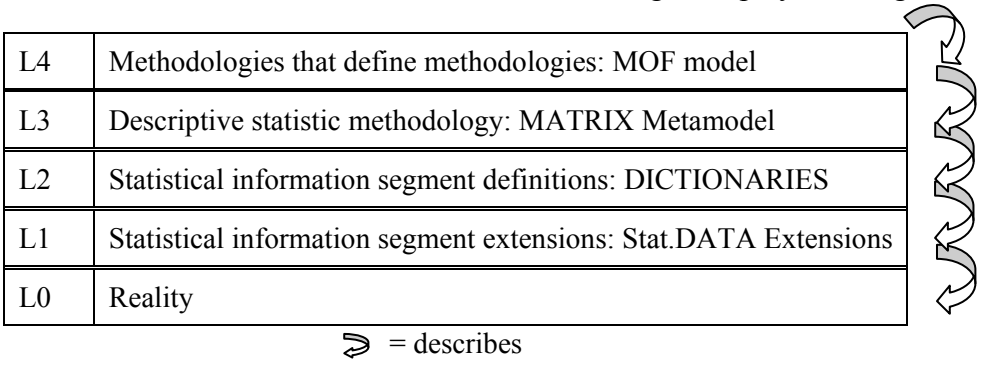


Figure 2. *The Hierarchy of Levels of Models in Statistics*

---

7 In the four-level model approach, both the original datum and the quality datum are considered level 1 data and have their definition in a level 2 model. So a level 1 model also contains metadata. The relationship between a datum and its "non-definition metadata" takes place within the same level, not between different ones.

8 An important feature of a fourth-level model is its self-describing property, that is, the ability of its structures to describe themselves and, therefore, to make levels higher than fourth superfluous.

9 An international organization supported by over 800 members: the major information system vendors, developers and users. OMG was founded in 1989 and promotes the theory and practice of object oriented technology in software development [12] [13].

10 See OMG Meta Object Facility specification, March 2000.

Other related models

Besides the models for statistical data representation and processing, some models on related topics are also treated in this document.

The first one concerns **references to external data**. To better specify and document statistics, it is often necessary to enrich the models in the statistical hierarchy with references to other data, sometimes not structured or structured differently than statistical ones, such as comments, notes, text documents, etc.

The second one defines the **administration system** of the statistical hierarchy models. As said in [8], the multi-level hierarchy of models provides a method to distinguish the competencies between different units, establishing a high-level link between the statistical information system structure and the organisation involved in running and using it, in term of roles and competencies.

References and administration rules are designed to be linked, if necessary, to every model and at every level of the statistical hierarchy. Furthermore, they are not considered specific to the statistical methodology. For these reasons, both references and administration rules are represented following a hierarchy used in the EDP field rather than the statistical one described before. The MOF is used as the unique common fourth-level model. According to OMG standards, UML is used as the third-level model. Models of the references and of the administration rules are placed at the second level. Individual references and administration rules are at the first level. See Figure 3.

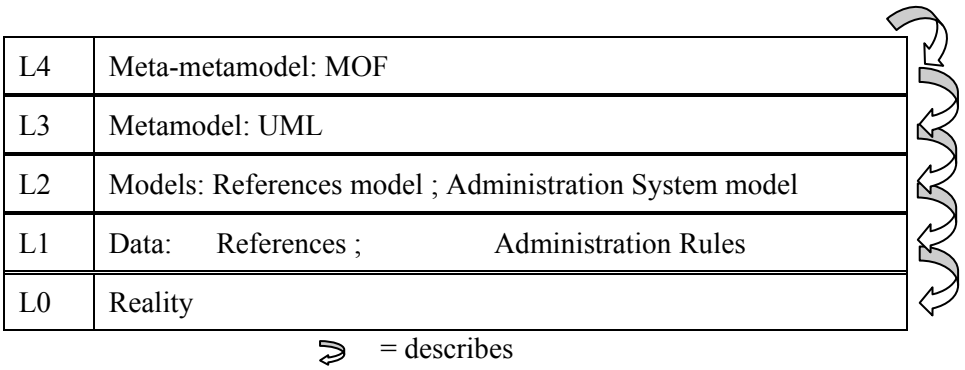


Figure 3. *Hierarchy of Models for "references to external data" and "administration system"*

So, level 2 models of "references to external data" and of the "administration system" are described in this document using UML.

The implementation model

One of the main goals of the architecture presented here for data representation and processing is the automation of the operative activities necessary to manage and use statistical data (see [10]). Going back to [8], we see that a model on a certain level can be used to process models in the lower level, acting as a specification. If this behaviour is enforced in practice, for example by means of

software artefacts, the system is called "active". In this case, the software artefacts must be generalized [9] and able to read the upper level model and apply it as a specification to perform consistent processing on lower level models [10].

We are used to seeing lower level models take shape in diagrams drawn in the higher level model language. In this document, for example, models are presented using MOF or UML. But diagrams, although very clear for humans, cannot be directly read by software artefacts. However, the model of each level can be represented as data and in this shape can be easily read by suitable software. Data can be also used by humans, directly or by means of automatically generated diagrams. For this reason, the master of a generic model to be is considered its implementation as data in an EDP environment, and the other forms are derived from it.

Because there are no tools to implement a model of the statistical hierarchy directly in an EDP environment, an EDP hierarchy has to be used for the implementation.

For example, using an XML instance to store data, the XML metamodel must be used as the third-level model, producing XML Schemas in the second level and XML instances in the first. In the same way, when data are stored in a relational data base, the relational metamodel and relational models must be used respectively in the third and second level.

Therefore, choosing a generic model of the statistical hierarchy, its structure has to be described in the second level of the EDP hierarchy (e.g. XML schemas or relational structures) and its content in the first level (XML instances or relational rows), independently of the level of the model in the statistical hierarchy.

The need arises to map the structures of the "statistical" model with respect to the EDP structures used to implement it (bearing in mind that many alternative implementations of the same statistical model can be realized).
This mapping itself could be represented according to a set of rules that could be seen as the "mapping representation" or "implementation" model.

Because in the Bank of Italy the "statistical" definitions are stored in a relational data base, an implementation model is used to map the models of the statistical hierarchy with the relational model (the implementation model is not presented in this document).[11]

[11] The implementation model itself is implemented as data. In this way the statistical processing software can be written according to the "statistical" model and made independent of the implementation choices.

The Matrix statistical metamodel

Introduction

The "Matrix" metamodel serves to support statistical data representation and processing. In fact, as well as representing the data, its major purpose is to permit their active processing, by designing a metadata system able to drive software artefacts by means of the definition of the data and the operations to be performed on them.[12]

Another important purpose of the metamodel is to allow the statistics definers[13] to give their definitions autonomously, possibly without the intervention of EDP people. This is feasible because the metamodel is the expression of a statistical methodology for describing statistical data and specifying operations on them and is based on concepts and language derived from mathematics and probability theory, rather than based on EDP.[14] Even if a more exhaustive description of the principles applied in building the representation can be found in [4], some notions are recalled below to improve the document's comprehensibility.

With regard to the Bank of Italy's EDP architecture for statistical processing, which consists of many function-specific processing packages ([11], [5]), the version of the Matrix statistical metamodel presented here is intended to embrace methodological constructs of general interest: it can be thought of as the "common" model. Like satellites, other "function-specific" models are related to the common model and complete the architecture, although they are not described here.

Symbols and diagrams are the same as in UML because the core structures of MOF and UML are the same. However, some UML indications are missing because the model presented here is not intended to give all of the information needed for an EDP implementation. A description of the UML conventions used in this document can be found in the appendix. On the other hand, the fact that the metamodel is historic is not rendered in UML notation: this is to say that the classes and the associations of the metamodel are in general dependent on time (the topic is treated in the "Historicity and other common properties" section). The identification attributes of the classes are not shown in the diagrams and descriptions either (see the appendix or the "Historicity and other common properties" section). The names of the items and structures of the model are preceded by an M_ (to mean "of the Matrix statistical metamodel") to distinguish them from terms used in the common language, according to principles set out in [6], [8] on terminology.

Firstly a synthetic and general overview of the metamodel is given, then, splitting the metamodel in parts, the analysis goes into more detail. Hence, most of the classes represented in the overall diagram are discussed in much more detail in the sections dedicated to the single parts of the model, where other more specific classes and associations are also introduced.

---

12 Software artefacts are "active" if they are driven by the upper level model of the model they have to process (see [6], [8]); an "active" system also helps to enforce "infological" completeness" [6], [8], [14], because a model cannot exist (because it cannot be produced) if the upper level model that defines it does not exist.

13 According to the roles defined in [6] and [8], statistics definers are administrators who apply a statistical methodology (third-level model) to produce the definition of one or more statistical segments (second-level models).

14 The metamodel is expressed at a "business" level rather than at a "technical" level, since in this case the statistics are the "business".

Overview of the metamodel

General metamodel description

The Matrix statistical metamodel can be divided into three sections, representing respectively:

- **Statistical concepts**, i.e. abstractions of interest for statistical purposes (see also [4]);
- **Statistical data**;
- **Transformations**, i.e. computations aimed to calculate data using as operands other data available in the system.

The aim of the "concepts" section of the metamodel is to allow a representation of abstractions independent of "data", defining abstractions autonomously and then referring to them within data definitions. In this way abstractions (and relations between them) can be defined once and shared by different data, making it possible to enforce coherence and harmonize statistical data that describe the same aspect of the real world.[15]

Many abstractions have to be used for a full description of statistical data. They are necessary to identify statistical units (e.g. people, firms, banks, bank operations, …), to define the groups of statistical units which data refer to, i.e. the key of the data (e.g. by age, gender, economic activity, place of residence, branch, country, …), and to specify the measures (e.g. income, earnings, …). For each of these categories it may also be necessary to specify additional meanings (country of birth, country of residence, …) and to list possible elements, that is single people, banks, gender, ages, countries … or groups of them.

The abstractions, called **`M_Concepts`**, are of three main types: **`M_Elements`**, **`M_Sets`**, and **`M_Variables`**.[16]

`M_Element` is the Matrix metamodel entity used to represent the single elements of the categories of statistical interest (e.g. a single country, economic activity, time value, …).

`M_Set` is used to define sets of `M_Element`s (e.g. lists of countries, economic activities, time values).

`M_Variable` is used to give an additional meaning to the generic `M_Element` belonging to an `M_Set`, named "definition set" of the variable (e.g. country of birth, country of work, country of residence, …)

The subclasses are linked by some basic integrity relationships.[17]

---

15 Sharing `M_Concept`s can also help to reduce the definition work of data and concept definers.

16 As explained in [4], each type of `M_Concept`s derives from the corresponding algebraic notions (variable, set and element of a set), but `M_Variable`s, `M_Set`s and `M_Element`s are generally "historical", i.e. their existence and properties are likely to be time dependent; in this case, considered at a specific moment, they correspond to their equivalent "algebraic" object (for example, an algebraic set), whereas over time they can be considered as a collection of algebraic objects, each referring to a specific moment (see also the historicity section).

17 Note that in the general case, due to the historicity of `M_Concept`s, the multiplicities of relations is referred to a single instant of time (considering the whole "time" axis, the multiplicity would always be "many" to "many")

- An *M_Set* may "contain" many *M_Elements*, an *M_Element* may belong to many *M_Sets*: the association between *M_Set* and *M_Element* represents the *M_Set* definition in terms of *M_Elements* (an *M_Set* is composed of *M_Elements*, from 0 to N);
- An *M_Variable* "is defined in" one and only one *M_Set*, many *M_Variables* can be defined in the same *M_Set*: the association between *M_Variable* and *M_Set* means that a variable has a definition set in which it takes values (one and only one); the values are the *M_Elements* belonging to the *M_Set*.

The second section of the metamodel concerns statistical data definitions. A statistical datum is considered to be the law which associates each pair consisting of a group of statistical units and a manifestation of time with the measure of one or more properties of the pair, as described in [4]. A single association is called an "observation". Therefore, a statistical datum is thought of as a mathematical function (statistical function), so the entity used to define data is called **M_Function**.

A statistical function has a domain made up of some *M_Variables* (the ones that identify groups of statistical units and time) assuming values in some *M_Sets*, and a co-domain also made of some *M_Variables* (the measures) assuming values in some other *M_Sets*. Both domain and co-domain are in general multi-dimensional. Therefore, many *M_Concepts* are generally needed to define an *M_Function*. The same *M_Concept* can be used to define many *M_Functions*.

The variables that define the *M_Function* domain, also called "independent", can be divided into two categories: grouping or time variables. The former is used to specify the groups of statistical units,[18] the latter to specify the time values.[19] In the general case, the domain can include many groups and many time values (a matrix of groups and times). Important function subtypes are the historical series, which have a domain made up of one group and many time values,[20] and the cross-sections, which have a domain made up of many groups and one time value.

Although at a conceptual level the difference between time and grouping variables can be significant, there are no structural differences in the representation and therefore such a distinction is not emphasized in the metamodel. Simply, time variables are the variables that have a definition set of time values. Likewise, the distinction between cross-sectional data, time series data and matrices of groups and time periods is left to the definition of the domain of the function. Furthermore, although the time is often emphasized in the exposition due to the significance of the topic, data that are not dependent on time are also admitted and no variable is considered strictly mandatory.[21].

The variables of the *M_Function* co-domain, also called "dependent", represent the kind of information we want to know about groups of statistical units and time (measures) and can be quantitative or qualitative.[22] There can be dependent variables, called "attributes", used, when needed, to describe properties of the observations.

---

[18] A group can also consist of a single object, so that so called microdata [14] and analytical data [4] can also be represented.

[19] A time value can correspond to a time period or to an instant (which is considered as a particular case of a period).

[20] GESMES/CB (CB is for Central Banks) also uses a multidimensional Key-family to define a homogeneous historical series family (i.e. an array).

[21] Data not dependent on time are also admitted.

[22] A variable is considered "quantitative" when it makes sense to perform mathematical operations in its definition set and "qualitative" if not.

The extensional form of a function (which is located at the first level of the four-level hierarchy) links each domain element to one co-domain element. According to the general rule, the extensional form cannot exist if the function definition does not exist (i.e. the second-level model that defines the first level and that complies with its third-level model, i.e. this metamodel).

As said before, the "transformations" section of the metamodel is used to define computations.

In this case as well, to ensure the system's consistency, it should be possible to execute only defined computations. Note that this constraint is satisfied when the software that executes computations is active, i.e. driven by the definition of the transformations.

The main goal of transformations is to define how to calculate new statistical data starting from existing ones. Nevertheless, the definition of "transformation" is more general: it is considered a computation that can have many operands and one result, all of which are called "transformation members" and they can be statistical data or statistical concepts. The main entity for defining transformations is the **`M_Transformation`**.

An *`M_Transformation`* can be viewed either from an external point of view (i.e. as a black box, considering its external relationships with the other Matrix metamodel entities) or from an internal one (i.e. as a white box, taking into consideration its internal structure).

Externally a transformation defines a link between the operands and the result. In addition, because the result of a transformation can be the operand of another one, the transformations definitions trace the calculus sequences of the statistical information system (see Figure 4).

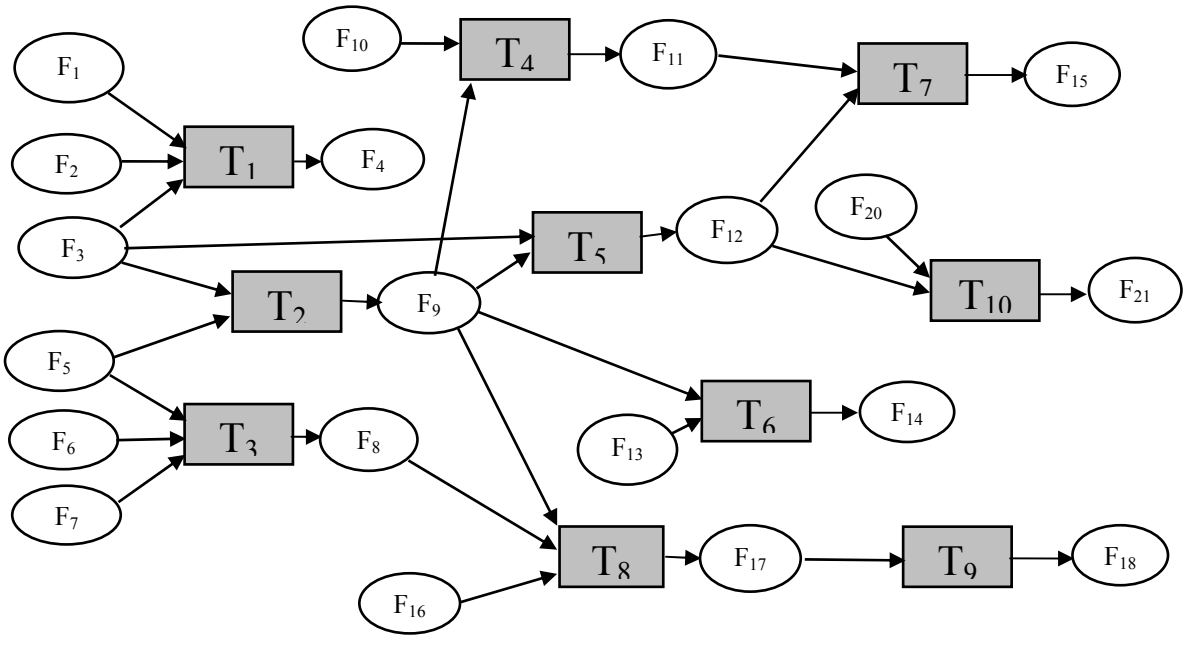


Figure 4: *Graphic example of calculus sequences (T_x=`M_Transformation` n.x; F_y=`M_Function` n. y)*

Internally (i.e. inside any single T_x box in the diagram above), transformations use "operators" to define the algorithms to be performed. The metamodel does not set the "grammar" of the operators:

they can be freely defined and refer, at a less conceptual level, to software routines able to perform them.[23]

Like concepts and data, transformations are also typically defined autonomously by the statistics definers[24] using existing operators (EDP work is needed only to develop the necessary routines when new operators have to be introduced).

The metamodel consisting of concepts, data and transformations con be thought of as an onion with three layers, in which any layer refers to the more internal ones (Figure 5):

- *M_Concepts* are in the internal layer, because they can be defined autonomously and independently of *M_Functions* and *M_Transformations* (however *M_Concepts* definitions are useful not "per se" but in defining *M_Functions* and *M_Transformations*);
- *M_Functions* are in the middle layer because they need *M_Concepts* to be defined; representing the data definitions of the information system, they can be considered the main point of the metamodel;
- *M_Transformations* are in the external layer; they are defined using *M_Function* and *M_Concepts* and specify the computation algorithms of calculated data.

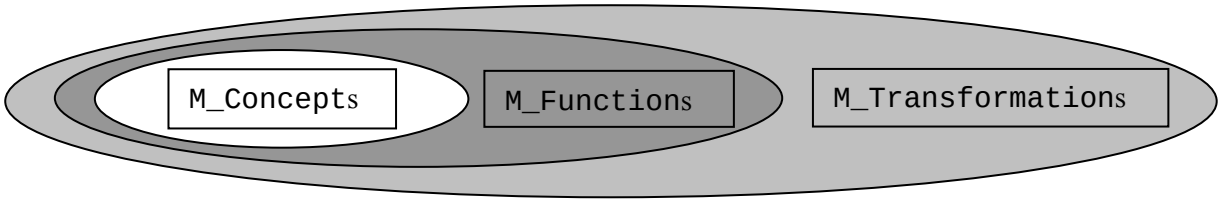


Figure 5: *The three parts of the Matrix statistical metamodel*

[23] Since the system is "active", each operator must have a routine able to perform it (in a sense the code of the software routine can also be considered the "extensional" form of the operator, i.e. its first level model)

[24] According to [6] and [8], they are the administrators who use a third-level model to produce second level ones.

General metamodel: class diagram

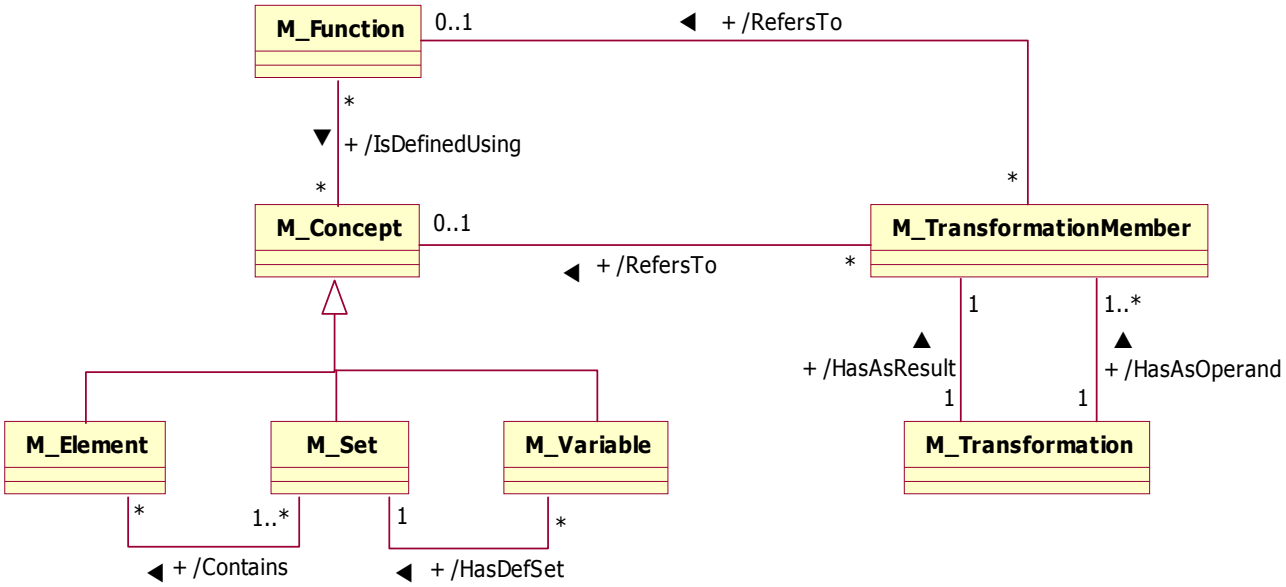


Diagram 1: *General metamodel*

Diagram description: definitions

M_Function

Statistical data definition conceived of as mathematical function. An *M_Function* is defined using *M_Concepts*. An *M_Function* is defined using any number of *M_Concepts* and an *M_Concept* can participate in the definition of any number of *M_Functions* (/IsDefinedUsing association).

M_Concept

Abstraction of interest for statistical purposes. An *M_Concept* can be referred to many times in the definition of other concepts, data and transformations. *M_Concepts* are of three types: *M_Variables*, *M_Elements*, *M_Sets*.

M_Element

Single element of a category of the reality. For example, for the category of "countries", *M_Elements* can be "Belgium", "Denmark", "France", "Germany", "Ireland", "Italy", "Luxembourg", "Nederland", and so on.

M_Set

Set of *M_Elements*. Examples of *M_Sets* for the category of "countries" are:
"the European countries", "the EC countries", "the Euro countries", "the Benelux countries" and so on. An *M_Set* contains any number of *M_Elements* (/Contains association). An *M_Element* can belong to any number of *M_Sets*.[25]

M_Variable

Generic unknown *M_Element* of an *M_Set* connected to a more specific meaning. Examples of variables for the category of countries are: "country of birth", "country of work", "country of residence", and so on. An *M_Variable* is defined in an *M_Set* (/HasDefSet association), that is the *M_Set* in which it can take values (the values are the *M_Elements* belonging to such an *M_Set*).

M_Transformation

Definition of a computation of an *M_Function* or an *M_Concept* from other *M_Functions* or *M_Concepts*. An *M_Transformation* has a result (/HasAsResult association) and can have any number of operands (/HasAsOperand association)

---

[25] But at least one

M_TransformationMember

Class representing the operands and the results of the *M_Transformations*. An *M_TransformationMember* can refer to an *M_Function* or an *M_Concept* (/RefersTo associations). The same *M_Function* or *M_Concept* can be referred to by any number of *M_TransformationMembers*.

Integrity constraints:

- The /RefersTo associations between *M_TransformationMember* and *M_Function* and between *M_TransformationMember* and *M_Concepts* are mutually exclusive (an *M_TransformationMember* refers to either an *M_Function* or an *M_Concept*, never both, never none).

Historicity and other common properties

The Matrix metamodel allows historical definitions, i.e. one of its features is to represent the history of concepts, functions and transformations considering them in a temporal perspective.

Firstly, the instances of the classes of the Matrix metamodel (*M_Instances*) have a life, i.e. a set of time instants in which they are regarded as "existing" and thus being valid instances.

Example 1:
The M_Elements " West Germany" and "East Germany" existed until 1989, the M_Element "Euro" has existed since 1999, the M_Element "Italian lira" existed until 2001, the M_Function "Number of inhabitants of European countries" exists from … to …

Secondly, for *M_Instances*, attributes and references to other *M_Instances* resulting from associations also have a time validity and can change over time.

Example 2:
Tthe M_Element "country of Greece" has belonged to the M_Set of the euro countries since 2002, the M_Element "country of …" has belonged to the M_Set of the European Community since …., the M_Set of European countries contains West Germany and East Germany until 1989, the M_Function "Number of inhabitants of European countries" has the M_Domain called "domain A" since …

To represent the historicity, a historical qualifier is attached to all of the classes of the Matrix metamodel (M_Classes): it specifies the "period", with regard to a conventional time called "reference time", in which the generic *M_Instance* exists and has constant attributes and references to other *M_Instances*.

Therefore, a generic *M_Instance* can have many data occurrences, each one corresponding to a single period of existence of the *M_Instance*. The periods of validity of the various occurrences of an *M_Instance* must not overlap one another. The whole life of an *M_Instance* corresponds to the "union" of the periods of validity of its occurrences.

In this context, due to the variability of the references in connection with time, the multiplicity of associations would always be "many" to "many". To avoid such a loss of descriptive power, in this document the multiplicity of the associations is described with reference to the single, generic time instant.

Attributes and associations that cannot change with time are specified within the integrity constraints as "invariable with time".

The historicity of the definitions allows the active software to use different algorithms to process data relevant to different reference times.

Another kind of historicity, treated in a different way, is related to "the information system time", i.e. the time period in which a certain definition is present in the meta-information system and can produce effects in the processing. This kind of historicity is automatically applied to all the definitions inserted, cancelled or modified in the system, logging them with the "system date and time", so that it is possible to know, at every instant, present or past, the definitions that are or were

effective and active for processing.[26] However, since this kind of historicity concerns the processing model more than the statistical one, it is not explicitly treated here.

Besides the historical qualifier, also other attributes are common to all the `M_Classes` and to their references to other `M_Classes` resulting from associations. The common attributes are the following:

- `Historical qualifier` [mandatory]: the period of validity in which *`M_Instance`* exists and has constant attributes and references to other *`M_Instances`*; the period is specified by the attributes: "start date" and "end date";[27]
- `Administration Model reference` [mandatory]: set of attributes providing the link between the *`M_Instance`* (or of a period of validity of an *`M_Instance`*) and the maintenance organization, specifying roles and competences for the administration; the topic will be discussed in the "External References and Administration Models" section;
- `External Information reference` [optional]: set of attributes providing the link between the *`M_Instance`* (or of a period of validity of an *`M_Instance`*) and other relevant information whatever structured, like text (e.g. comments, notes, documents) or other data differently structured than the Matrix metamodel.

In the diagrams and descriptions of this document, such attributes are not reported within the `M_Classes`, since it is assumed that they are inherited.

Still about the general properties, every class of the metamodel has an identifier, that is a set of attributes whose values uniquely identify an instance of the class. The identifiers are not shown in the diagrams and descriptions of this document although they are taken to exist for every class, to be mandatory and to be named by adding "Id" to the name of the class (e.g. `M_SetId`, `M_FunctionId`, …).

Another property, common to most of the metamodel classes, is the "Description", which provides the meaning of the instance in natural language (multilingual when needed). The presence of the `Description` property is pointed out for each metamodel class.

[26] This logging also records the user who makes the change and is used as an audit trail.

[27] For instances that don't depend on time, the start and the end dates are set equal to the first and last instant of the reference time conventionally represented in the system.

Statistical Concepts

Introduction

The metamodel for representing the statistical concepts is built around the idea that the abstractions used to describe the properties of a statistical population can be defined using the conceptual instruments of probability theory, especially the notions of event and space of events. In the present work, obviously, there is no interest in discussing the measure of probability, but rather in using the formal representation of the events given in the theory.

*A **chance phenomenon (or experiment, or random trial)** can be defined as a process that, once executed, produces a specific result from a predefined set of possible results called **space of results** or also **sample space** (note that if the result is not one of the predefined set, the experiment is considered as not executed).*

For example, if the experiment is tossing a coin, the space of the results is the set {head, tail}. In rolling a dice, the sample space is {1, 2, 3, 4, 5, 6}.

*An **event** is a set of possible results, that is a subset of the space of the results (an event "occurs" if the outcome of the experiment is one of the results of the set).*

*A **space of events** is, in simple terms, the space of any possible event (in general the power set of the sample space) or the space of the event of interest, that is a collection of subsets of the sample space closed under countable set operations (and therefore allowing to define measures).*

*The space of events includes the empty set (also called "**impossible event**" because it will never take place) and the set that comprises all of the possible results (also called "**certain event**" because it will always take place). An event that contains just one possible result is called "**elementary event**".*

For example, for the experiment of tossing a coin the space of events contains the following events:

Impossible event:	*{}*
Event A:	*{head}*
Event B:	*{tail}*
Certain event:	*{head, tail}*

In rolling a dice, the possible events are all the sets obtained using integers from 1 to 6, and therefore:

{},{1},{2},...,{6},{1,2},{1,3},...,{5,6},{1,2,3},{1,2,4},...,{4,5,6},...,...,...,{1,2,3,4,5,6}

Events are mutually exclusive if the occurrence of one of them prevents the occurrence of the others. Mutually exclusive events correspond to a disjointed set of results.

Events can be composed using logical operators (like and, or, not) obtaining other events of the same space: this corresponds to the composition using set operators (like intersection, union, complement respectively) of their set of results.

Following this approach, the depiction of a generic category of the reality is considered a chance phenomenon, e.g. the determination of time, territorial location, economic activity, currency, shares, economic operators (persons, firms, …), economic operations and so on.

Example 3:

For territorial location, the space of result can be defined as the set of the points of the earth's surface. Consequently, any (measurable) set of points of the earth's surface (i.e. each zone or territory) is considered an admissible event: regions, countries, continents, etc. are events represented as `M_Elements` *in the metamodel.*

In the case of time, a generic time instant can be assumed to be a possible result, the space of result can be the set of all the time instants (or a set of time instants of interest, for example a predefined period). The possible events are, besides the single time instants, also any (measurable) set of them, such as days, months, years, etc.

Still in the case of time, different experiments and sample spaces should be defined for the measure of "durations", because the result of the measure is not an absolute time but a length of time.

The identification of single objects, such as people or securities, is also considered a chance phenomenon, in which the possible results are the single objects and the events of interest are the elementary events, i.e. the events composed of a

single result.

Moreover, when historical changes have to be considered, the original chance phenomenon is associated with another one consisting in determining the time the original measure refers to, in this way obtaining a combined experiment whose outcomes are pairs made up of an outcome of the original experiment and an outcome of the reference time. An event of this experiment is an "historical" event.

Given two chance phenomena A and B, the combined chance phenomenon C consists in the execution of an experiment of A and of an experiment of B, so a possible result of C is the pair of a result of A and a result of B (the space of results of C is in general the Cartesian product of the spaces of results of A and B)

Note that the probability theory approach in defining the space of events also allows a smooth formalization of the so called missing values, i.e. values that indicate that the measure of a certain category was not successful, so that a significant value is unknown. In fact the "unknown" ("missing") result can be considered a proper possible result of the chance phenomenon (also different types of "unknown" results if needed) and consequently "unknown" or "missing" events can be defined as needed.

A proper choice of the events of interest allows the representation of a single level of detail of the category (for example the cities for the category of the territory) or different levels (for example cities, countries and continents). In the latter case, the metamodel allows the representation of the logical relationships between events, like the hierarchical structure between events of the different levels of a classification (for example, the composition of continents in term of countries and of countries in term of cities).

For simplicity, the presentation of the `M_Concepts` metamodel is divided into three parts. Firstly, the basic one, which contains the main entities and their relationships. The second one addresses the hierarchical relationships between `M_Elements`, in which the important topic of hierarchies and classifications is treated (such relationships are considered instant by instant, i.e. with time being equal). The third one deals with the relationships between `M_Elements` across time (such as the temporal phenomena of merging, breaking up, renaming).

Description of the statistical concepts metamodel

The space of results is registered in the M_SpaceOfResult class. Single results and their association to events are not represented.[28] The focal point of the concept metamodel is the definition of the events and spaces of events. Only the events of interest are defined.[29]

To identify the events uniquely, it is necessary to choose an "identifier", that is a set of identification attributes (e.g. the ISO code for countries; the NACE code for economic activities; the ISO code for currencies; name, surname, place and date of birth of people; the ISIN code for securities).

The M_EventsDomain (from now M_Domain) is the entity used to define a space of events with a given events identifier. An *M_Element* is the representation of an event in an *M_Domain* and it is identified by a specific "value" of its identifier [30]. An *M_Element* belongs to just one *M_Domain*. An *M_Domain* can contain many *M_Elements*.

The ideal situation, in a statistical information system, would be to have just a single identifier for each category of the reality. In practice, however, the coexistence of many identifiers cannot always be avoided, owing to the different needs to be satisfied, different coding of events at a different level of detail, different sources of data, different standards and organizations, and so on (e.g. the ISO and SWIFT codes for countries, …). Many *M_Domains* can correspond to the same *M_SpaceOfResult* because of the possible coexistence of more than one event identifier. The same event can be defined in many *M_Domains* of the same *M_SpaceOfResults*, producing different *M_Elements* with the same meaning (synonyms).[31]

Example 4:
Possible M_Domains for the category of the "territory":

Countries – Identifier: ISO code (FR= France ; ES=Spain)
Countries – Identifier: Bank of Italy code (001=United States; … ; 039=Italy; …)

The M_Set class, in the depicted context, contains the definition of the "sets of events" and the M_Variable class contains the definition of the "random variables". The relationships between M_Element, M_Set and M_Variable classes are the same as those described in the overall diagram section.

Moreover the diagram shows that an *M_Set* refers to an *M_Domain* (it can contain *M_Elements* belonging to just one *M_Domain*), and an *M_Variable* is defined on an *M_Domain* (it can assume values in *M_Sets* referring to just one *M_Domain*).

---

[28] This is obvious because the outcomes of an experiment can also be infinite.

[29] This makes the representation finite and does not limit the detail achievable, because an event can also contain a single result of the experiment.

[30] The value will be used to refer to the M_Element when needed, in particular in the data extensions, for this reason the term "value" will be often used as a synonym of M_Element.

[31] *M_Elements* with the same meaning can also exist in the same *M_Domain* (when the same event is defined more than once).

Statistical concepts metamodel: class diagram

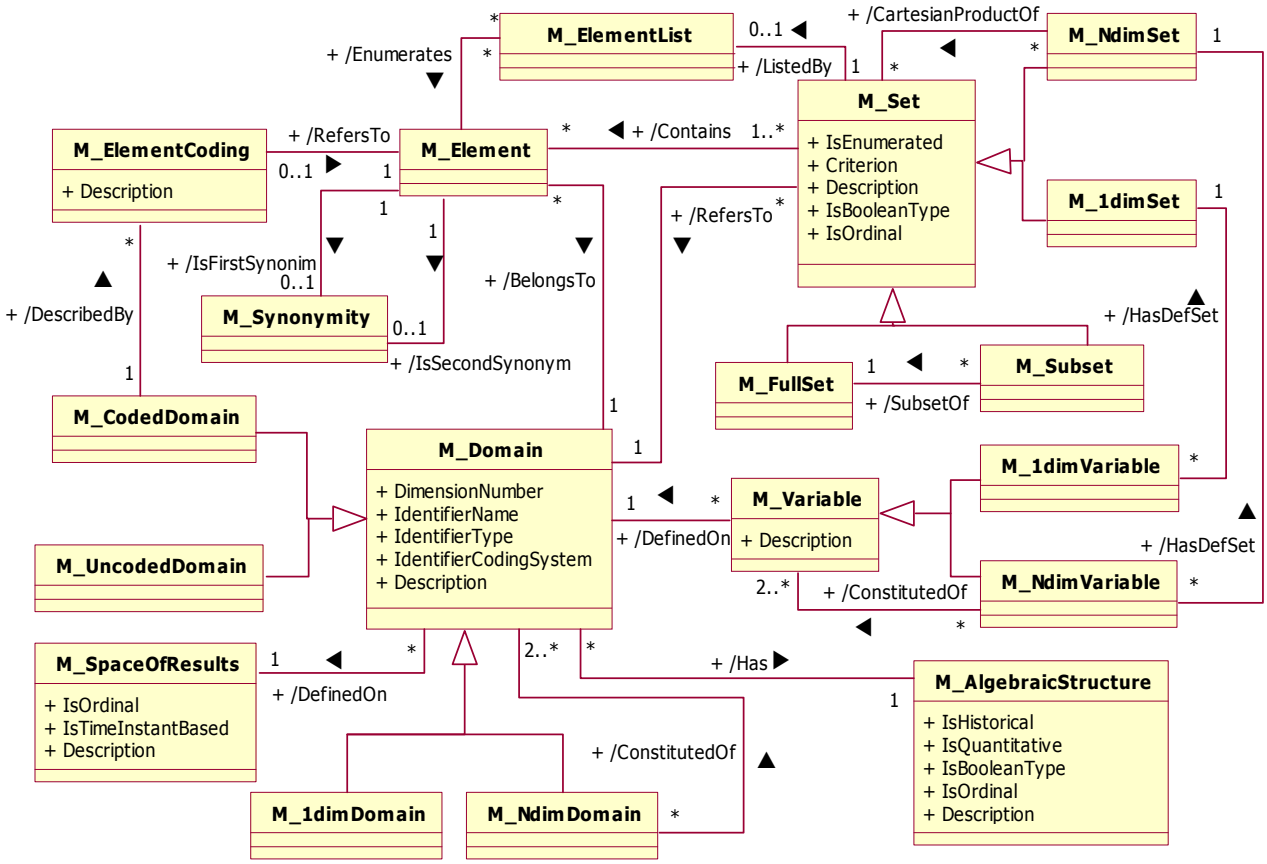


Diagram 2: *Statistical concepts metamodel*

Diagram description: definitions

M_SpaceOfResult

The sample space of the chance phenomenon that measures a category of the reality.

Attributes:

- IsOrdinal [mandatory]: the *M_SpaceOfResult* is ordinal[32] if an intrinsic ordering criterion exists between all of its points (i.e. all of the possible results of the chance phenomenon)
  Examples:
  - *The sample space of tossing a dice is ordinal, since the intrinsic ordering criterion is the natural numbers sequence;*
  - *The sample space of territorial zones is not ordinal*
- IsTimeInstantBased [mandatory]: the *M_SpaceOfResult* is based on time instants if its possible results are time instant[33].
- Description [mandatory].

Integrity constraints:

- IsOrdinal and IsTimeInstantBased attributes are invariable with time
- For a time instant based *M_SpaceOfResult* there is an intrinsic ordering criterion based on the sequence of time instants, therefore when IsTimeInstantBased is "true" IsOrdinal is also "true"

M_Domain
(and M_CodedDomain; M_UncodedDomain; M_1DimDomain; M_NDimDomain)

In simple terms, an *M_Domain* is the set of the elements of interest of a category that are identified by a given identifier (set of identification attributes). More formally, it is a representation of a space of events by means of a given identifier. An *M_Domain* contains (from zero to many) *M_Elements*, each representing an event (/BelongsTo association).

An *M_Domain* is defined on an *M_SpaceOfResults* (/DefinedOn association).

An *M_Domain* has an *M_AlgebraicStructure* (/Has association) from which it inherits algebraic properties.

An *M_Domain* can be coded or uncoded.
An *M_CodedDomain* is an *M_Domain* whose identifier is a conventional code that is not considered self-explanatory for understanding the meaning of the *M_Elements*. An *M_CodedDomain* is described by the M_ElementCoding class (/DescribedBy

---

[32] "Ordinal" means that it can be ordered following an intrinsic criterion

[33] Note that the space of result of time distances (durations) is not considered time instant based, because its results are not time instants.

association) which has the Description attribute that specifies the meaning of the M_Elements.
An *M_UncodedDomain* is an *M_Domain* whose identifier is considered self-explanatory, so nothing else is needed to specify the meaning of the *M_Elements*.

An *M_Domain* can be considered a Cartesian Space and can be mono (*M_1dimDomain*) or multi dimensional (*M_NdimDomain*). An *M_NdimDomain* is the representation of a combined space of events. An *M_NdimDomain* is constituted of *M_Domains* (/ConstitutedOf association) having a lower number of dimensions.

Attributes:

- DimensionNumber [mandatory]: number of dimensions of the *M_Domain*.
- IdentifierName [mandatory, for *M_1dimDomains* only]: it specifies the attribute that will be used to uniquely identify the *M_Elements* belonging to the *M_1dimDomain*. For *M_NdimDomains* the identifier is inferred from those of the component *M_1dimDomains*.
- IdentifierType [mandatory, for *M_1dimDomains* only]: it specifies the data type of the identifier for the *M_1dimDomains*. For *M_NdimDomains* the data type is inferred from those of the component *M_1dimDomains*.
- IdentifierCodingSystem [optional, for *M_1dimDomains* only]: it specifies the coding system used to express the IdentifierName for the *M_1dimDomains*, if the identifier is coded.
- Description [mandatory].

Integrity constraints:

- Coded/Uncoded characteristic is invariable with time
- IdentifierName attribute is invariable with time
- IdentifierType attribute is invariable with time
- DimensionNumber attribute is invariable with time
- The /DefinedOn association with M_SpaceOfResults is invariable with time
- The /Has association with M_AlgebraicStructure is invariable with time
- The /DescribedBy association with M_ElementCoding is invariable with time
- The /ConstitutedOf association is invariable with time
- An M_Domain is time instant based when it is defined on a time instant based M_SpaceOfResult (the IsTimeInstantBased property is inherited)

M_AlgebraicStructure

This class gathers some algebraic properties that an *M_Domain* can have as well as the set of operators allowed for its *M_Elements* in data calculation and processing.

Attributes:

- IsHistorical [mandatory]: the *M_AlgebraicStructure* is historical when it is time dependent. The existence and relationships of the *M_Elements* belonging to an *M_Domain* having a historical *M_AlgebraicStructure* are considered time dependent and are

qualified by a "period of validity"[34] (otherwise there is not time dependency and the validity is assumed equal to "always").

- `IsQuantitative` [mandatory]: the *`M_AlgebraicStructure`* is quantitative when it makes sense to perform mathematical operations such as addition, subtraction, multiplication, division and so on (note that this property cannot be merely inferred from the data type of the `M_Domain` class). The *`M_Elements`* belonging to an *`M_Domain`* having a quantitative *`M_AlgebraicStructure`* represent quantitative values.
  If the *`M_AlgebraicStructure`* is not quantitative, it is called "qualitative" and mathematical operations are not allowed.
- `IsBooleanType` [mandatory]: the *`M_AlgebraicStructure`* is Boolean when it makes sense to apply logical operators (e.g. "and", "or", "not") or set operators (e.g. intersection, union, complement). The *`M_Elements`* belonging to an *`M_Domain`* having a boolean *`M_AlgebraicStructure`* can represent any elements of the power set of the space of results (hence at any level of detail, e.g. cities, countries, continents and the whole world in a sample space of the territory); therefore they may not be mutually exclusive and can be composed by mean of logical operators to give other *`M_Elements`* of the *`M_Domain`* (a more detailed explanation can be found in the "Description of the hierarchies metamodel" section).
  If the *`M_AlgebraicStructure`* is not Boolean, it is called "partition type" or "partition". *`M_Elements`* belonging to an *`M_Domain`* having a partition *`M_AlgebraicStructure`* represent mutually exclusive events (hence at just one level of detail) and cannot be composed by mean of logical operators.
- `IsOrdinal` [mandatory]: the *`M_AlgebraicStructure`* is ordinal when it has an intrinsic ordering criterion.
- `Description` [mandatory].

Integrity constraints:

- The `IsHistorical`, `IsQuantitative`, `IsBooleanType`, `IsOrdinal` attributes are invariable with time
- A Quantitative *`M_AlgebraicStructure`* is partition type
- A Qualitative *`M_AlgebraicStructure`* can be either partition or Boolean type
- A Quantitative *`M_AlgebraicStructure`* is ordinal
- A Qualitative *`M_AlgebraicStructure`* can be ordinal or not
- A time instant based *`M_Domain`* can be Qualitative or Quantitative
- A time instant based *`M_Domain`* can be Boolean or partition
- An *`M_Domain`* defined on a "non-ordinal" *`M_SpaceOfResult`* cannot have an "ordinal" *`M_AlgebraicStructure`*
- An *`M_Domain`* defined on an "ordinal" *`M_SpaceOfResult`* can have an "ordinal" *`M_AlgebraicStructure`* (or not)
- An *`M_AlgebraicStructure`* of Boolean type is not ordinal

`M_Element`

This is the abstract class of all of the `M_Elements` of interest, independently of their concrete and explicit representation in the system.[35] In simple words, *`M_Elements`* are single elements of the

[34] In the general case, the periods of validity can be many, provided they don't overlap.

categories of the reality. More formally, an *M_Element* is the representation of an event of a chance phenomenon corresponding to the measure of a category of the reality.

An *M_Element* belongs to one and only one *M_Domain* (/BelongsTo association). For a full identification of an *M_Element*, the pair (*M_DomainId*, *M_ElementId*) is needed.
For *M_UncodedDomains* the *M_Element* meaning[36] is inferred from the value of the M_ElementId (which is considered self-explanatory), for *M_CodedDomains* it is provided by the Description attribute of the M_ElementCoding class.

Example 5:

M_CodedDomain: *Currency identified by "ISO" coding system*
Example of M_Element*:* *Value: "29" : "CHILEAN PESO"*

M_CodedDomain*: Geographical Locations identified by the Bank of Italy coding system*
Example of M_Elements*:* *Value: "21": "Denmark"*
Value: "29" "France"

M_UncodedDomain*: Geographical Locations identified directly by the name of the country*
Example of M_Elements*:* *Value: "Denmark"*
Value: "France"

Integrity constraints:

- The /BelongsTo association with the *M_Domain* class is invariable with time
- An *M_Element* has the same dimensions as the *M_Domain* it belongs to
- The dimensions of an *M_Element* are invariable with time
- For a Boolean *M_Domain* the definition of an *M_Element* corresponding to the certain event is mandatory

M_ElementCoding

The class provides the "descriptions" needed to understand the meaning of the *M_Elements* belonging to *M_CodedDomains* (see also M_CodedDomain and M_Element classes).
An *M_ElementCoding* refers to one and only one *M_Element* (/RefersTo association).

Attributes:

- Description [mandatory]

Integrity constraints:

- *M_Elements* belonging to *M_CodedDomains* must be referred to by an *M_ElementCoding*

---

[35] *M_Elements* can be defined explicitly by listing them or implicitly by giving a criterion whose outcome is the *M_ElementList* (see also the description of the M_Set class)

[36] The "meaning" of an M_Element consists in understanding which event is represented.

- *M_Elements* belonging to *M_UncodedDomains* cannot be referred to by an *M_ElementCoding*

M_Synonymity

Two or more *M_Elements,* belonging to the same *M_Domain* or to different *M_Domains* defined on the same *M_SpaceOfResult*, can have the same meaning, corresponding to the same event. They are called synonyms and the instances of this class establish the connection between pairs of them. Hence, an *M_Synonymity* refers to two *M_Elements* (/IsFirstSynonym and /IsSecondSynonym associations). When possible and suitable, one of the synonyms is given a pivot role, connecting all other synonyms to it through many *M_Synonymities*: in this way it is possible to avoid untidy chains of *M_Synonymities*.[37]

M_Set
(and M_Fullset, M_Subset, M_1dimSet, M_NdimSet)

An *M_Set* is a set of *M_Elements* belonging to the same *M_Domain*.

An *M_Set* refers to one and only one *M_Domain* (/RefersTo association). An *M_Set* has the same dimensions as the *M_Domain* it refers to, so it can be mono (M_1dimSet) or multi dimensional (M_NdimSet).

An *M_Set* can contain any number of *M_Elements* (/Contains association[38]).

The *M_Elements* that belong to the *M_Set* can be specified explicitly by listing the *M_Elements* (/ListedBy association with the M_ElementsList class), implicitly by mean of a criterion whose outcome gives the list of *M_Elements* (the Criterion attribute) or by both: in this case the list of *M_Elements* specified explicitly must be equal to the outcome of the criterion.

For an *M_NdimSet*, the *M_Elements* can also be specified through the Cartesian product of other *M_Sets*, each one referring to one of the component *M_Domains* of its *M_NdimDomain* (/CartesianProductOf association). In addition, a Criterion can be used to specify, if necessary, other *M_NdimSets* containing combinations of *M_Elements* to exclude from the *M_NdimSet* being defined (or to include, being the only admissible ones).

An *M_Fullset* is an *M_Set* that contains all of the *M_Elements* belonging to the *M_Domain*. For each *M_Domain*, one and only one *M_Fullset* exists, independently of time.

An *M_Subset* is a proper subset of the *M_Fullset* of the *M_Domain* (/SubsetOf association). For each *M_Domain* any number of *M_Subsets* can exist.

---

37 If "A","B","C","D" are synonym and § means "is a synonym of", it is normally better to specify A § B, A § C, A § D with "A" playing a pivot role rather than A § B, B § C, C § D, D § A, …, without any role.

38 The /Contains is the abstract association between an M_Set and its *M_Elements*, however specified, i.e. also for the *M_Elements* that are specified by means of a criterion and are not explicitly represented or listed; the explicit list of *M_Elements* is registered in the M_ElementList class.

By default, the *M_FullSet* of an *M_NdimDomain* is considered to be the Cartesian product of the *M_FullSet* of all the component *M_Domains*; if so, it is not necessary to specify explicitly its *M_Elements*. Otherwise, a Criterion can be used to specify other *M_NdimSets* containing combinations of *M_Elements* to exclude from the default Cartesian product (or to include, being the only admissible ones).

Note that the extension of the *M_Fullset* of an *M_NdimDomain* expresses the compatibility / incompatibility between *M_Elements* of the component *M_Domains*. Hence, defining explicitly *M_NdimDomains* and their *M_Fullsets* serves to express the mutual compatibility or incompatibility between events of different spaces of events (or also of the same one, if an *M_Domain* is combined with itself).

Attributes

- IsEnumerated [mandatory]: specifies whether an explicit extension of the *M_Set* is defined, i.e. if the *M_Set* is listed by an *M_ElementList* (/ListedBy association).
- Criterion [optional]: name of the algorithm that produces the *M_Set* extension, usable by active software to generate the list of *M_Elements* of the *M_Set*;[39] a criterion can also be used, in particular for *M_NdimSets*, to specify combinations to be excluded/included.
- Description [optional].
- IsBooleanType [mandatory]: an *M_Set* is of boolean type if it can contain *M_Elements* that are not mutually exclusive, otherwise it is of partition type (or "partition").
- IsOrdinal [mandatory]: an *M_Set* is ordinal when it has an intrinsic ordering criterion between its *M_Elements*.[40]

Integrity constraints:

- The /RefersTo association with *M_Domain* class is invariable with time
- The property of being the *M_Fullset* or an *M_Subset* is invariable with time
- The *M_Set* has the same dimensions as the *M_Domains* it refers to
- The dimensions of an *M_Set* are invariable with time
- An *M_Set* can contain (/Contains association) only *M_Elements* that belong to *(*/BelongsTo association) the *M_Domain* it refers to *(*/RefersTo association*)*
- An *M_Set* can contain (/Contains association) any number of *M_Elements*, An *M_Element* is contained by one or more *M_Sets* (at least to the *M_FullSet* of the *M_Domain*)
- For an *M_Set* either a Criterion (attribute of the M_Set class) or an explicit list of the *M_Elements* (/ListedBy association with M_ElementsList class) or a Cartesian product (/CartesianProductOf association) must exist (for the *M_FullSets* of an *M_NdimDomains*, it is always considered existing the Cartesian product of the *M_FullSets* of all the component *M_Domains*)
- If the attribute IsEnumerated is "true", the *M_Set* is listed by one and only one *M_ElementList* (the multiplicity of the /ListedBy association is one and only one);

---

[39] The criterion can be used to produce the extension of the *M_Set* and use it dynamically, even without storing it explicitly as an *M_ElementList*.

[40] Any *M_Set*, even if it is not ordinal, can be ordered by means of any number of conventional criteria (not intrinsic ones) that can be expressed in many ways: the topic is not treated here because at the moment it is not part of the "common" meta-model.

if IsEnumerated is "false", the *M_Set* is not listed (the multiplicity of the /ListedBy association is zero)

- The *M_Sets* inherit a number of properties from the *M_Domain* they refer to and from the related *M_AlgebraicStructure* and *M_SpaceOfResult*:
  - *M_Sets* referring to a Boolean type *M_Domain* can be of Boolean type or not
  - *M_Sets* referring to a Partition type *M_Domain* cannot be of Boolean type
  - *M_Sets* referring to quantitative *M_Domains* are quantitative
  - *M_Sets* referring to qualitative *M_Domains* are qualitative
  - *M_Sets* referring to an *M_Domain* defined on a non-ordinal *M_SpaceOfResult* are not ordinal
  - *M_Sets* referring to an *M_Domain* defined on an ordinal *M_SpaceOfResult* can be ordinal or not
  - An *M_Set* referring to an ordinal *M_Domain* is ordinal
  - An *M_Set* referring to a non-ordinal *M_Domain* can be ordinal or not
  - A subset of an ordinal *M_Set* is ordinal
- An *M_NdimSet* containing combinations of *M_Elements* to exclude (or to include being the only admissible ones) from another *M_NdimSet* being defined, can refer to a subset of the dimensions of the latter.

M_ElementsList

An *M_ElementsList* is the explicit list of the *M_Elements* belonging to an *M_Set*. An *M_ElementsList* enumerates the *M_Elements* of one and only one *M_Set* (/Enumerates association), an *M_Set* can be listed by one or no *M_ElementsList* (/ListedBy association).

Integrity constraints:

- If a Criterion is specified for the *M_Set*, the *M_ElementsList* must enumerate the *M_Elements* resulting from the execution of the criterion.

M_Variable
(and M_1dimVariable, M_NdimVariable)

An *M_Variable* is a generic unknown *M_Element* of an *M_Set* to which a specific additional meaning is given. An *M_Variable* can also be thought of as a characteristic (i.e. a property) abstracted from the objects it can be referred to. In the probability theory approach, an *M_Variable* is the random variable of the measure of a property.

The basic component of the *M_Variable* definition is the *M_Set* whose *M_Elements* are the admissible values of the *M_Variable* (also called here "defining *M_Set*" of the *M_Variable*: /HasDefSet associations).

The defining *M_Set* of an *M_Variable* refers to an *M_Domain* (/RefersTo association) and can be the *M_Fullset* or one of its *M_Subsets*. Therefore, an *M_Variable* is also defined on an *M_Domain* (/DefinedOn association) and has the same number of dimensions of the *M_Set* and *M_Domain* on which it is defined, i.e. can be mono (*M_1dimVariable*) or

multi dimensional (*M_NdimVariable*). An *M_NdimVariable* is constituted of *M_Variables* having a lower number of dimensions (/ConstitutedOf association).

Attributes:

- Description [mandatory]

Integrity constraints:

- The /DefinedOn association with the M_Domain class is invariable with time
- The /ConstitutedOf association between M_NdimVariable and M_Variable is invariable with time
- The number of dimensions of an *M_Variable* is invariable with time
- The *M_Variable* inherits the characteristics of the *M_Domain* and of the *M_Set* it is defined on:
    - An *M_Variable* is Boolean or partition type if its defining *M_Set* is of that kind
    - An *M_Variable* is qualitative or quantitative if its defining *M_Set* is of that kind
    - An *M_Variable* is ordinal if its defining *M_Set* is of that kind
    - An *M_Variable* is time instant based if its defining *M_Set* is of that kind

Description of the hierarchies metamodel

This section deals with the logical relationships between *M_Elements*, seen as probability theory events, and their organization in hierarchies.

Particularly suited to such a construction are the *M_Domains* having an *M_AlgebraicStructure* of Boolean type (attribute IsBoolean is "true"), because they allow a Boolean algebra to be defined within their *M_Elements*.[41]

A boolean algebra is an algebraic structure (i.e. a set of items and operators having given properties) that summarizes the properties of the set operators (e.g. intersection, union, complement) and the logical operators (e.g. "and", "or", "not").

A Boolean type *M_Domain* is the more general type of *M_Domain* and results from the probability theory notion of space of events.

In a space of events, events can be combined with logical operators. For example C = (A and B) is the event that occurs if both A and B occur; D = (A or B) is the event that occurs if A or B (or both) occur, E = (not A) is the event that occurs if A does not occur.
This corresponds, in the space of result, with set operations between the corresponding set of results. For example, the results set of C is the intersection of the results sets of A and B, the results set of D is the union of the results sets of A and B, the results set of E is the complement with respect to the certain event of the results set of A.

In the representation of the algebraic structure of a Boolean type *M_Domain*, the Matrix metamodel takes into consideration a specific operation: the "partition" operator (and its inverse operation "disjoint union").

Given an event "A", a partition of "A" is a set of mutually exclusive events "B_i" (i=1..n) whose "or" composition gives "A" again (the results sets of "B_i" are the partition of the results set of "A"). The inverse operation of the "partition" is the "or" of disjointed events (which corresponds to the union of disjointed results sets).

The M_ElementComposition and M_ElementCompositionItem classes allow such a relationship to be defined, by associating a set of components (*M_Elements* that must be mutually exclusive) to the compound *M_Element* that is their logical "or" (the same *M_ElementComposition*, considered in the opposite direction, relates a compound *M_Element* with a set of *M_Elements* that are its logical "partition").

An *M_ElementComposition* has a hierarchical structure (the compound is the parent, the components are the children) and it is the basic step in defining an *M_Hierarchy*, which is simply a group of *M_ElementsCompositions*. *M_Hierarchies* can have a structure made up of predefined levels (*M_Classifications*) or a free structure (*M_FreeHierarchies*).

Note that the metamodel is historical. For example, the components of an *M_ElementsComposition* and the *M_ElementCompositions* that are part of an *M_Hierarchy* can change with time. The operator of "partition" is intended to be applied time by time, i.e. time being equal.

---

[41] However, from a formal point of view, it is also possible to define hierarchies between different domains (including those of partition type) provided these domains share the same 'Space of Results'.

Hierarchies metamodel: class diagram

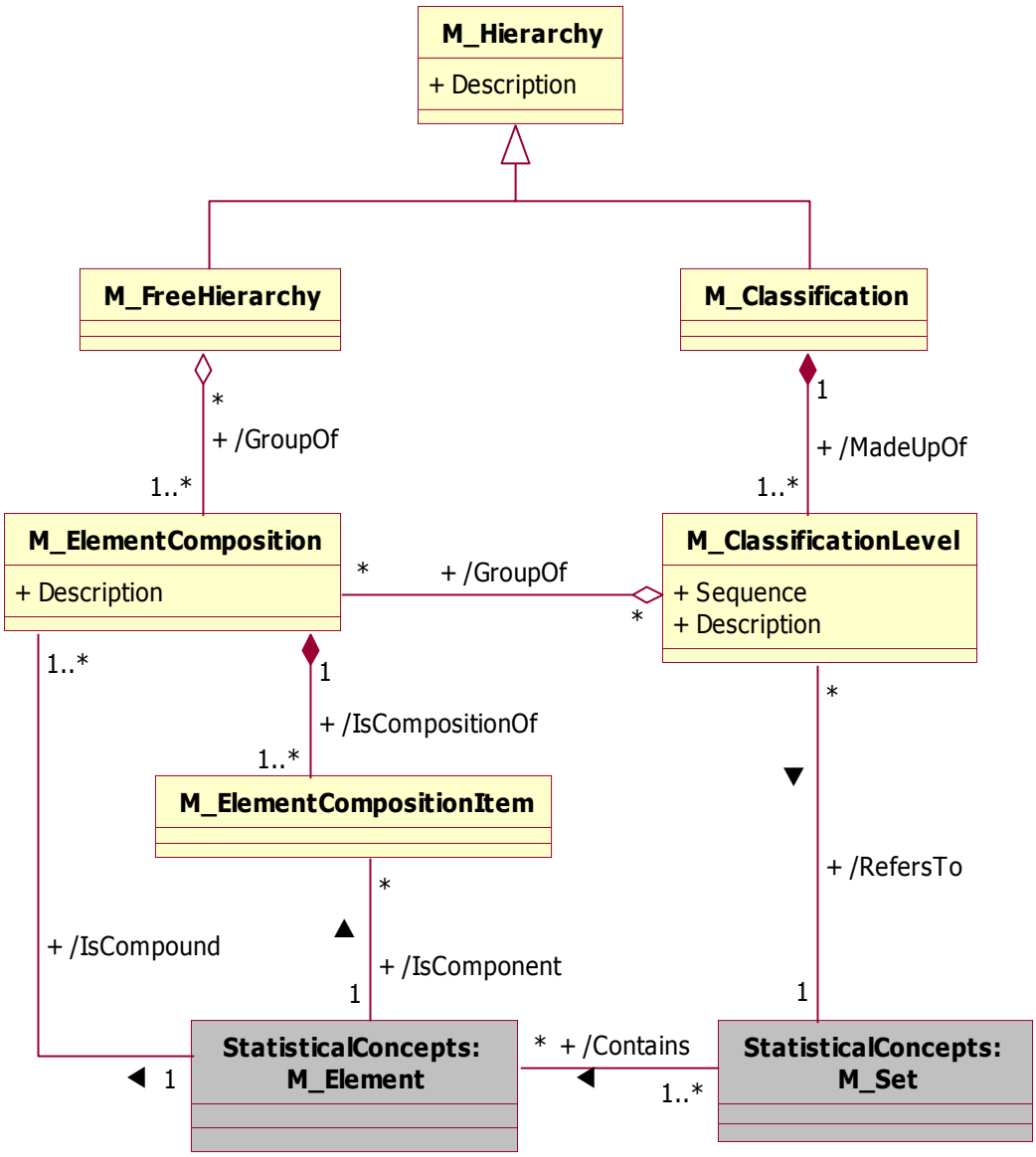


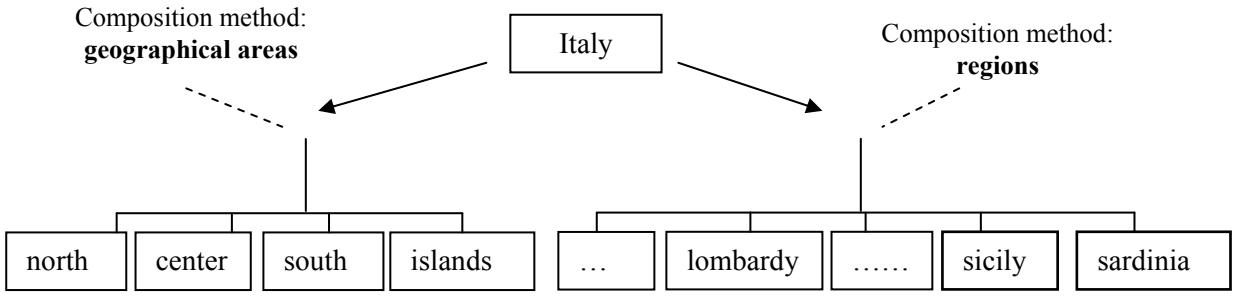
Diagram 3: *Hierarchies metamodel*

Diagram description: definitions

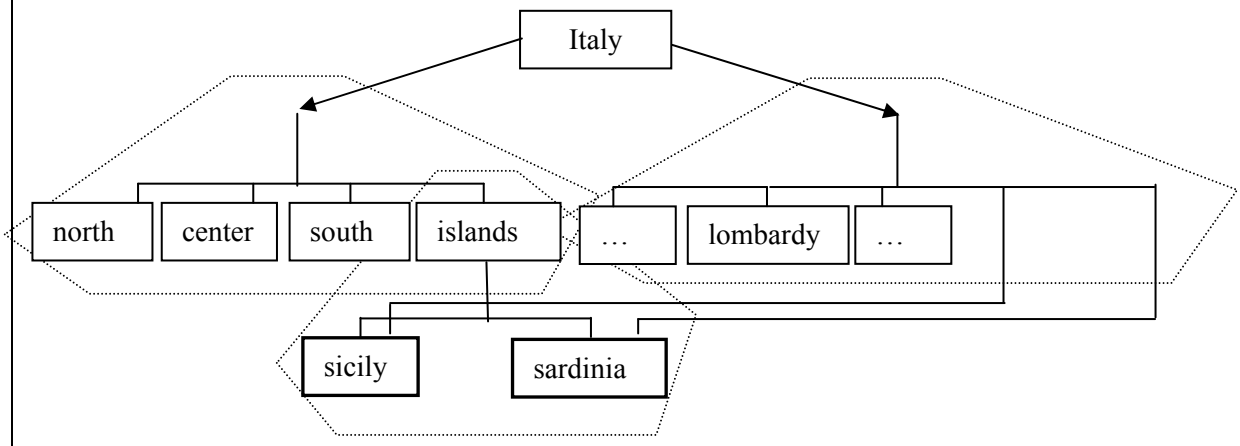
M_ElementComposition

An *M_ElementComposition* is the composition by means of "logical OR" of any number of mutually exclusive *M_Elements* to obtain a compound *M_Element* (/IsCompound association). Alternatively, it can be viewed as a way to decompose an *M_Element* into a set of mutually exclusive *M_Elements* whose "logical OR" returns the original *M_Element*. There can be many ways of decomposing a given *M_Element*, so that the same *M_Element* can be the "compound of" many different *M_ElementCompositions*.

Example 6:
For the M_Domain "territorial areas" it is possible to represent "ITALY" as the union of geographical areas north, center, south and islands or, alternatively, as the union of the Italian regions:



For the same M_Domain it is possible to specify other compositions between M_Elements by introducing, for example, the geographic area "Islands" in terms of regions; the hierarchical tree could be the following:



Note: "Italy", "Sicily" and "Sardinia" are defined once in the M_Element class and are simply referred to in the three M_ElementComposition definitions (one for "Islands" and two for "Italy").

Attributes:

- Description [optional]

M_ElementCompositionItem

Single item of the *M_ElementComposition* (/IsCompositionOf association) that refers to a single component *M_Element* (/IsComponent association).

M_Hierarchy

Hierarchical structure defined among the *M_Elements*. An *M_Hierarchy* is a group of *M_ElementCompositions* There are two possible kinds of hierarchies: the classification (see below M_Classification class) and the free hierarchy (see below the M_FreeHierarchy class).

Attributes:

- Description [mandatory]

Integrity constraints:

- The property of being an *M_FreeHierarchy* or an *M_Classification* is time invariable
- In an *M_Hierarchy* an *M_Element* can be the compound *M_Element* (/IsCompound association) of one and only one *M_ElementComposition*

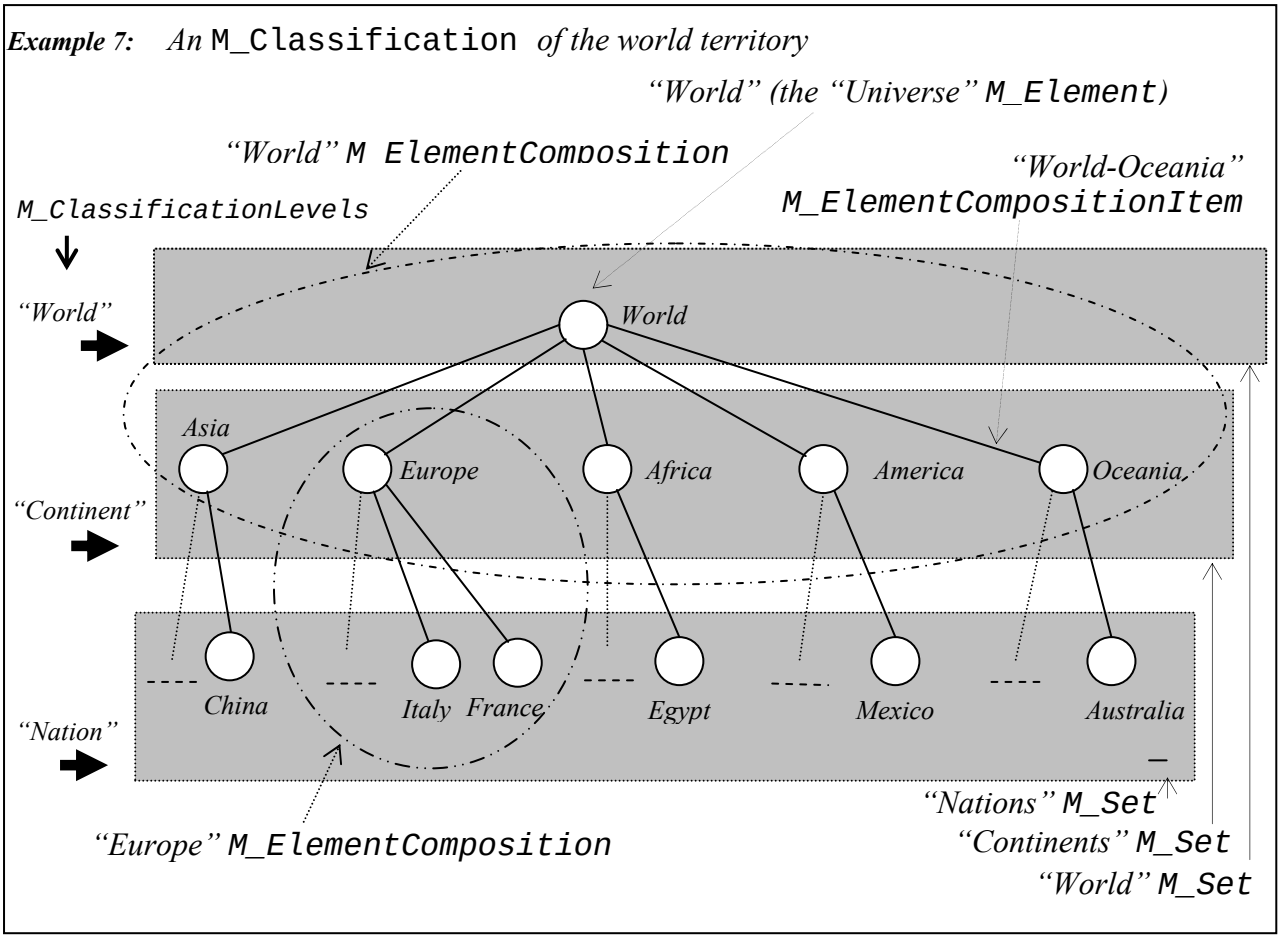
M_Classification

An *M_Hierarchy* made up of predefined and ordered levels (/MadeUpOf association with M_ClassificationLevel class) , each of them formed of mutually exclusive *M_Elements*, in which the *M_Elements* of a given level are obtained as a composition of *M_Elements* of the previous one. The composition of all the *M_Elements* of each *M_ClassificationLevel* gives the more general event of the hierarchy, called the "universe" or "root *M_Element*" of the *M_Classification*.

The systematic classification of phenomena and the naming of the classes provides the common language which makes consistent communication possible [cited after T.M.F. Smith]. Thus, a classification or nomenclature is an ordering system for the phenomena of a certain kind. It describes the division of a set of objects, such as economic activities, products, diseases or professions into classes. Each class has a unique identifier, its code. A classification can also be seen as a controlled language that enables the user to define concepts and to relate them to classes and codes [7].

Integrity constraints:

- The root *M_Element* (also the "universe" of the *M_Classification*) is mandatory
- All the *M_Elements* of an *M_Classification* imply the root *M_Element* (if any one of the *M_Elements* occurs the root *M_Element* also occurs)
- An *M_Element* in an *M_Classification* can be a component of one and only one *M_ElementComposition*



M_ClassificationLevel

The predefined level of an *M_Classification*. An *M_ClassificationLevel* refers to an *M_Set* (/RefersTo association) that contains the *M_Elements* of the *M_ClassificationLevel*. An *M_ClassificationLevel* has a group of *M_ElemenCompositions* (/GroupOf association) that relate the *M_ClassificationLevel* to the next one (of course except for the level for which there is not a subsequent one). Note that the *M_Elements* of an *M_ClassificationLevel* are associated through the /IsCompound association to the *M_ElementsCompositions* of the *M_ClassificationLevel*.

Attributes:

- Sequence [mandatory]: ordering number of the M_ClassificationLevel within the M_Classification
- Description [mandatory]

Integrity constraints :

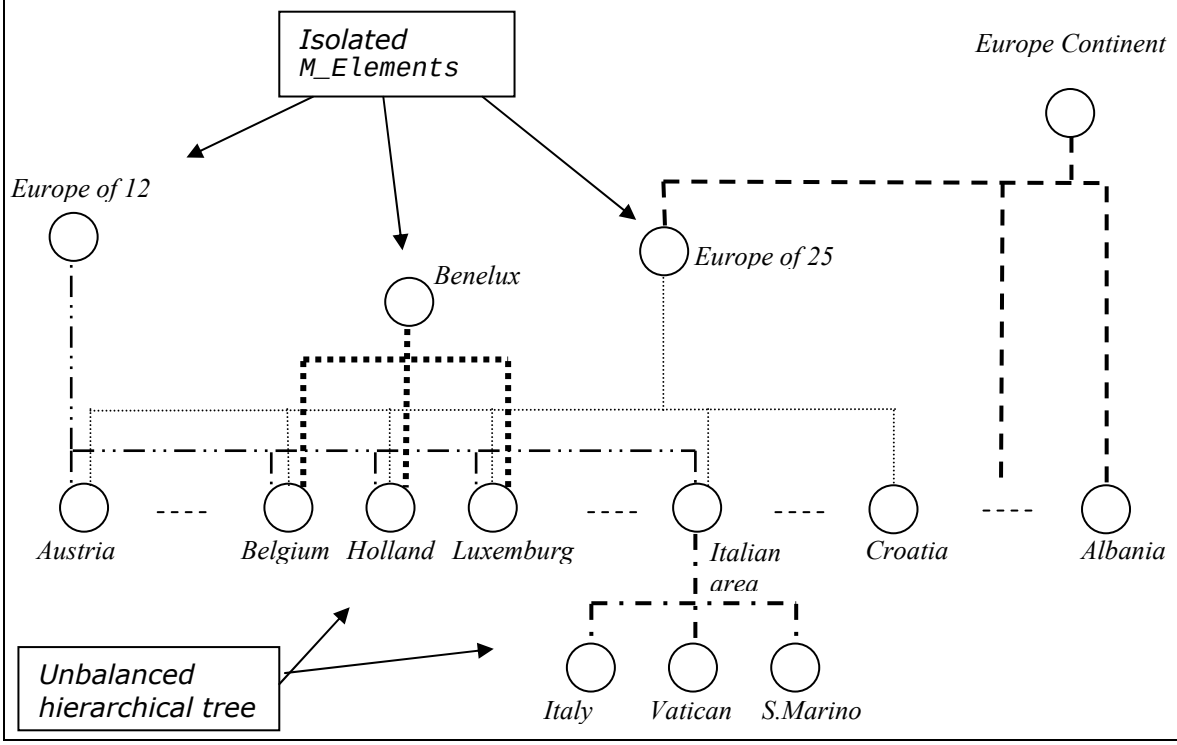
- An *M_ClassificationLevel* refers to an *M_Set* of partition type (i.e. an *M_Set* containing only mutually exclusive *M_Elements*)
- The first level of an *M_Classification* contains only the root *M_Element* (the "universe" of the *M_Classification*)
- The "logical or" of the *M_Elements* of a generic *M_ClassificationLevel* gives the root *M_Element* of the *M_Classification*

Example 8:
For the WORLD M_Element *the following compositions are equivalent:*
*WORLD = **U** [Asia, America, ...]* *(Union of the Continents)*
*WORLD = **U** [Italy, France, Japan,...]* *(Union of the Nations)*

M_FreeHierarchy

The *M_Hierarchy* with free *M_ElementCompositions* (/GroupOf association) defined among the *M_Elements*.

Example 9:
A free Hierarchy (circles indicate M_Elements*, dotted lines indicate* M_ElementCompositions*):*



As shown in the example above, in a free hierarchy there can be *M_Elements* that are not part of a level ("isolated" *M_Elements*) and the hierarchical tree can be unbalanced (there can be branches with different numbers of *M_ElementsCompositions* in the hierarchy).

Integrity constraints:

- An *M_FreeHierarchy* is not made up of predefined levels
- An *M_FreeHierarchy* does not necessarily have a root *M_Element* that represents the more general (or less detailed) event of the hierarchy
- An *M_Element* can be a component of many *M_ElementCompositions* of the *M_FreeHierarchy*

Description of 'Cross Time M_Element Correlations' metamodel

This section deals with the representation of the equivalence among events before and after a temporal discontinuity. This is also called the "correlation" among `M_Elements` across time (also the "time correlation") and it is needed in the case of temporal discontinuities in the `M_Elements` definition.

The word "event" has been used, until now, with the meaning it has in probability theory, e.g. a set of results of a random experiment; but "event" in natural language can also be used to indicate that something happens at a given time (and can cause a change of the "events", with a probabilistic meaning, that the `M_Elements` represent).

A "time correlation" is the definition of the equivalence between "events" (probabilistic meaning) valid before and after the temporal discontinuity.

> ***Example 10:***
>
> *real world category: Geographical Location*
>
> *What happened: West Germany and East Germany reunited*
> *Time Instant: 1989*
> `M_Elements`*: West Germany, East Germany, Germany*
> *Type: merge*
>
> *What happened: Czechoslovakia divided into Czech Republic and Slovakia*
> *Time Instant: 1993*
> `M_Elements`*: Czech Republic, Slovak Republic, Czechoslovakia*
> *Type: break up*

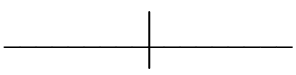
Discontinuities and hence time correlations can exist only for historical `M_Domain` (e.g. `M_Domains` that have a historical `M_AlgebraicStructure`).

The equivalence is established between an `M_Set` valid before and an `M_Set` valid after the temporal discontinuity. Neither the before nor the after `M_Set` can be empty.

There are different types of time correlations, depending on the number of `M_Elements` in each `M_Set` and on the `M_Elements` contained in the `M_Sets` before and after the temporal discontinuity.

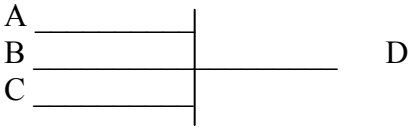
More precisely (in the following diagrams the temporal discontinuity is indicated by a vertical line):

<u>One before, one after</u>: change of name (e.g. change of coding system)

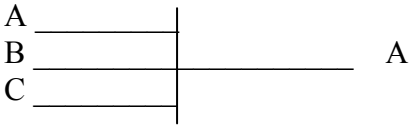


Many before, one after:

Merge (A, B, C merge and become D)

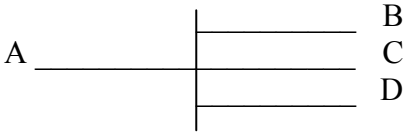


Incorporation (A incorporates B and C)

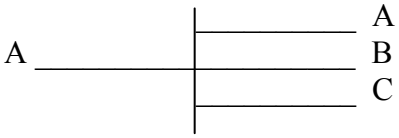


One before, many after:

Break up / fragmentation (A breaks up becoming B, C, D)

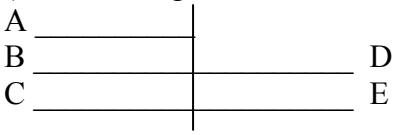


Hiving off (some parts of A are hived off and become B and C)

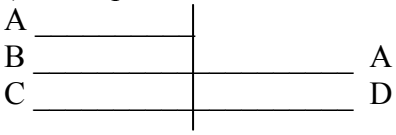


Many before, many after: merge and break up

(A, B, C merge and the outcome breaks up becoming D and E)



(A incorporates B, C and hives off D)



Note that the same *M_Element* does not correspond to the same set of results before and after the time correlation.[42]

[42] Although the set of results of an *M_Element* is not represented in the metamodel, in principle it must be thought as historical, i.e. containing results that can vary depending on the time.

Cross time M_Element correlations metamodel: class diagram

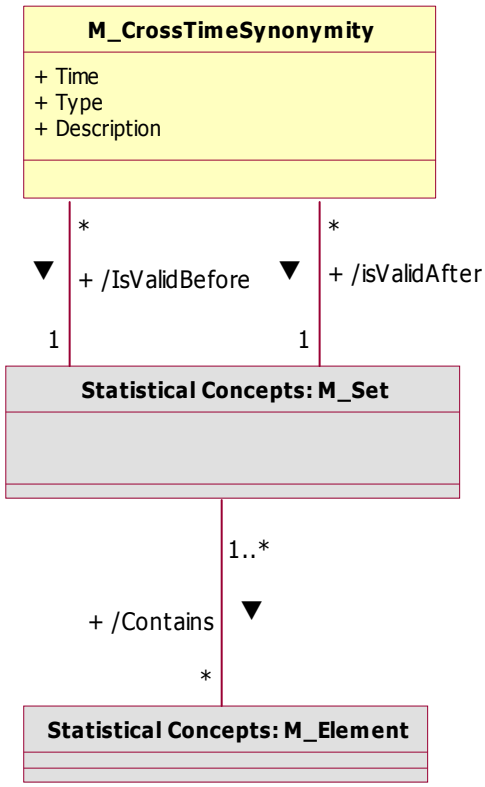


Diagram 4: *Cross time M_Element correlations metamodel*

Diagram description: definitions

M_CrossTimeSynonymity

This class establishes the equivalence between one *M_Set* valid before and one *M_Set* valid after the temporal discontinuity (/IsValidBefore and /IsValidAfter relationships).

Attributes:

- Time [mandatory]: the time instant when the temporal discontinuity happens
- Type [mandatory]: the type of temporal discontinuity.
- Description [optional]

Statistical Data

Introduction

As mentioned in the previous sections, in the Matrix metamodel statistical data are thought of as mathematical functions.

A mathematical function is an ordered triple (X,Y,G), where X and Y are sets (called the domain and codomain of the function), G is a subset of the cartesian product of X and Y such that each element x of X occurs exactly once as the first coordinate.
G is also called the graph of the function and is composed of ordered pairs (x,y) with the property that for an element x there is no more than one element y such that x is related to y.
The variable values x and y are called respectively "independent" and "dependent" variables. The function is also denoted by y= f(x).
The domain of a function X is the set of elements that occurs as the first coordinate in the pairs (x, y) of the relation G. If x is not in the domain, then f(x) is not defined.
The codomain (also the range) of a function is the set Y of elements y that can occur as the second coordinate in the pairs (x, y) of the relation G.
The variable x and y and the sets X and Y can be multidimensional, i.e. x=(x1, … , xn), y=(y1, … , ym)

The definition of a function consists in the specification of the set of "a priori" admissible values for the function occurrences (x,y),43 while the extension of a function is its graph "G".

Considering the statistical information system as a data warehouse, the definition is the "a priori" specification of the data structure whereas the "extension" is the set of observed values stored in physical archives (i.e. "a posteriori"). The definition is prepared before inserting data in the system and is used to specify the statistical data structure to providers, receivers, end users and software artefacts that process the data (check, store, retrieve, calculate, …).

The set of the admissible values of the function is the Cartesian product X*Y (or a subset of it), therefore the first (and maybe also the last) step of the definition of a function is the specification of the independent and dependent variables with their set of admissible values.

Example 11:

Function $\sqrt{x_1 + x_2} = y$

the domain dimensions are { x_1, x_2} and the co-domain dimension is { y };
the domain values are couples of values (x_1, x_2) belonging to the Cartesian product of admissible values of x_1 (any real value greater than or equal to 0) and x_2 (any real value), therefore { 4, 1} and { 16, -50} are elements of the function domain but {-8, 0} is not;
the codomain values are the admissible values of "y" (any real value),
the "a priori" admissible values for the function are the ordered triples (x_1, x_2, y), in which x_1 must be greater than or equal to zero and x_2 and y can assume any value.

When the set of admissible values of the function is a proper subset of X*Y, it is necessary to specify the combinations of the values of the variables that must be excluded because "a priori" they are impossible (or, conversely, the "a priori" possible combinations that must not be excluded). This can be done by defining in some way groups of combinations to be excluded or not (for example using Cartesian products of appropriate sets of two or more variables of the function).

43 "A priori" in this case means "before knowing the graph G".

Example 12:

Function "total amount of monthly loans in 2005 broken down by counterparty residence (Europe or non--Europe), counterparty location (European countries for Europeans and continents for non-Europeans), currency"

Independent Variables : Time, Residence of the Counterparty, Location of the Counterparty, Currency;
Dependent Variable: Total amount of loans

*The X*Y Cartesian product is obtained by specifying the three one dimensional sets of admissible values:*

- *{Jan2005,Feb2005,… ,Dec2005} for Time*
- *{European, Non--European) for the Residence of the counterparty*
- *{Austria, France, Germany, … , Africa, America. Asia,…} for the Location of the Counterparty*
- *{USD, euro, UK .Pound, …} for the Currency*
- *(from 0 to 10 [18] for the Total amount of loans*

Each occurrence of the function is a vector of 5 values such as {Jan 2005, European, France, USD, 15.687.300.000}, {Feb 2005, European, Italy, Euro, 56.789.000.000}. But not all of the points of such a Cartesian product are admissible, in fact a "European" can be located only in a European country and a non-European only in a continent (except for Europe). To specify impossible combinations (or alternatively possible combinations), the following Cartesian products can be defined:

First Cartesian product (containing combinations to exclude):

- *{European} for the Residence of the counterparty*
- *{Africa, Asia, America, … } for the Location of the Counterparty*

Second Cartesian product (containing combinations to exclude):

- *{Non-Eutopean) for the Residence of the counterparty*
- *{Austria, France, Germany,…} for the Location of Counterparty*

As for statistical concepts, also statistical data can be defined using the conceptual instruments of probability theory as stated in [4], For the purpose of this section, however, it is sufficient to remember that elements (values), sets and variables that are necessary to specify the function must be defined as M_Concepts.

As said above, the graph of the function (also called the extension of the function) is the correspondence law between the elements of the function domain and the elements of the function codomain. Each pair *(x,y)* is called "observation" or "occurrence".

Often, for mathematical functions, the correspondence law can be described by means of a synthetic rule that allows the dependent variable to be calculated once the dependent one is given, e.g. $y=x^2$

For statistical data, however, a synthetic rule is not available. The way to describe the graph of a function is to use a "correspondence table", e.g. a table where the pairs (x,y) are listed:

X	Y
0	0
1	1
2	4
3	9
….	….

In the hierarchy of models, the graph is located in the first level. According to the general integrity rules, a graph cannot exist if its definition does not exist in the second level (that is the definition of the function given according the third level metamodel, i.e. the Matrix one).

If the definition of the function exists, however, nothing can be said about the existence of the extension. Since the definition serves to specify "a priori" the objective of a statistical survey whereas the graph is the final result, the latter can certainly be missing if the survey has not yet been executed (or, for calculated data, if the calculus has not yet been made). The graph can also be partially known if the survey is not complete (for example, for periodical surveys, the graph exists for the past but not for the future).

When the information on the knowledge of the graph cannot be inferred directly from the graph itself, as is often the case for statistical data, it has to be represented in another way: the "knowledge domain" is the set of points in which the extension of the function is known[44] and it is also called the "status" of the function.

A special type of function consists of those that give information about the other functions belonging to the information system: they are called "functions about functions".[45] The "status" of a function can be considered a first sort of "function about functions", but many other types can exist, such as quality information, data specifying attributes of other data (e.g. if "stock" or "flow", if "measured" or "estimated").[46]

The presentation of the statistical data metamodel is divided into two parts, the first regarding the definition of the function and valid for every type of function, the second concerning the specificities of functions about functions.

[44] Remember that the definition domain expresses admissible values, i.e. the "a priori" values for which statisticians want to know the function and not the "a posteriori" values for which the function is known.

[45] As also noted in the "Architecture of Models" section, such functions are not meant to be the "model" (i.e. the level 2 definition) of the other functions they refer to; rather they have an extension in level 1 and a definition in level 2, like the other data bearing information about the reality.

[46] When the attributes are relevant to the occurrences of the function, they can be included as dependent variables in the same function as they describe.

Description of the statistical data metamodel

Statistical data are identified and registered by means of the M_Function class.

The Cartesian product that matches or contains the set of "a priori" admissible values of the *M_Function* is described by means of *M_StructureItem*. Every *M_StructureItem* is a dimension of such a Cartesian product.[47] An *M_Function* has many *M_StructureItems* (/HasAsDimension association), an *M_StructureItem* is a dimension of one and only one *M_Function*.

An M_*StructureItem* is the use of an *M_1dimVariable* within an *M_Function* with a specific role (e.g. dependent or independent) and with a set of admissible values (that is a subset of the defining *M_Set* of the *M_Variable)*. Hence, an *M_StructureItem* refers to an *M_1dimVariable* and an *M_1dimSet*, both defined in the "Description of the statistical concepts metamodel" section. The *M_1dimSet* referred to is also called the "domain in use for the *M_1dimVariable* in the *M_Function*".

The independent *M_Variables* in an *M_Function* can be quantitative or qualitative. If more than one independent *M_Variable* exist in an *M_Function*, each one can be quantitative or qualitative independently of the others.

If the set of "a priori" admissible values of the *M_Function* matches the Cartesian product of the *M_StructureItems*, the *M_Function* definition is complete. Otherwise, the specification of combinations to be excluded (or to be maintained) is made using the M_CombinationsGroup class.

Every *M_CombinationsGroup* identifies a group of combinations that can be excluded (/NotAllows association) or allowed (/Allows association). An *M_Function* can allow (or not allow) any number of *M_CombinationsGroups*. An *M_CombinationsGroup* can be allowed or not allowed by any number of *M_Functions*. An *M_CombinationsGroup* is the use of an *M_NdimVariable* with an *M_NdimSet*: the former qualifies its dimensions, the latter contains the combinations. An *M_Function* can refer only to *M_CombinationsGroups* whose dimensions are a subset of the dimensions of the *M_Function*.

With regard to historicity, as in the general case, an *M_Function* can be considered a set of mathematical functions, one for each "reference time" value. Consequently, the *M_StructureItems* and the *M_CombinationsGroups* of the *M_Function* can change with time.

The domain of the *M_Function* "X" can be obtained from the Cartesian product of the independent *M_StructureItems*, excluding or maintaining possible *M_CombinationsGroup* between them. The codomain "Y" can be obtained in the same way from the dependent *M_StructureItems*.

---

[47] Note that for some commercial data warehouses the term "dimension" is only used for independent variables of the data, while here it is used for both independent and dependent variables.

For the domain all the "a priori" admissible values must also be present in the graph G, if this is known (remember that "each element x of X occurs exactly once as the first coordinate"), while this is not true for the codomain.[48] The set of values of the dependent `M_Variables` that really occur in the graph G is also called the "actual codomain".

[48] Do not be confused by cases in which, as a convention, some function occurrences are not explicitly represented in the graph (e.g. when the function is equal to "zero"); in such cases some admissible independent values apparently do not occur in the graph, but this is only because the graph is not fully represented.

Statistical data: class diagram

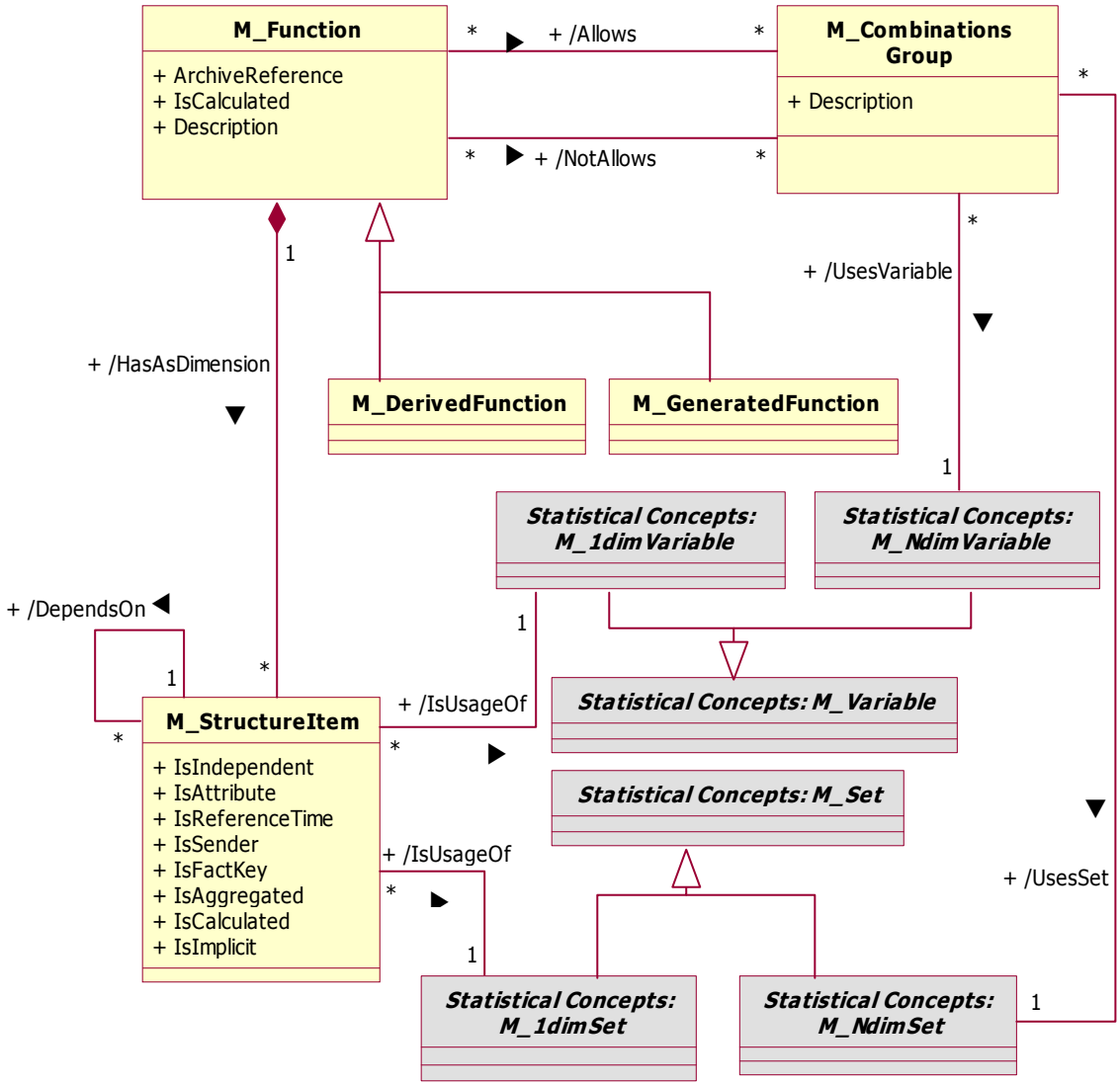


Diagram 5: *Statistical data metamodel*

Diagram description: definitions

M_Function
(And M_GeneratedFunction, M_DerivedFunction)

The M_Function class is the register of the data definitions.

An *M_Function* can be an *M_GeneratedFunction* or an *M_DerivedFunction*. An *M_GeneratedFunction* is an "original" or "primary" *M_Function* of the information system, not defined in terms of other *M_Functions*. An *M_DerivedFunction*, on the contrary, is defined in terms of other *M_Functions*.[49]

An *M_Function* has as dimensions any number of *M_StructureItems* (/HasAsDimension association).

An *M_Function* can allow (or not allow) any number of *M_CombinationsGroup* (/Allows and /NotAllows associations).

Attributes:

- IsCalculated [mandatory]: an *M_Function* is "calculated" if its graph "G" is obtained as the output of a calculus performed within the statistical information system; if not calculated, an *M_Function* is "collected" from external sources, i.e. its graph can be "reported" from reporting persons or institutions, extracted, transformed and loaded ("ETL" in data warehouse language) from other EDP databases or applications, "drawn up" from "statistics producers" (sorts of administrators, see [6]) and so on.
- ArchiveReference [mandatory]: the identifier of the physical archive (or archives) in which the *M_Function* extension is (or can be) stored.
- Description [mandatory].

Integrity constraints:

- An *M_GeneratedFunction* is "collected" (it cannot be "calculated" because it is not defined in terms of other *M_Functions*).
- An *M_DerivedFunction* can be collected or calculated (if an *M_DerivedFunction* is collected, it is defined in terms of other *M_Functions* but its calculus is not performed within the information system).
- An *M_Function* cannot "allow" and "not allow" *M_CombinationsGroups* at the same time.

M_StructureItem

An *M_StructureItem* is a mono-dimensional component of the Cartesian structure of an *M_Function*.

---

[49] The distinction is expressed as an "is a" in the diagram rather than with an attribute because it is referred to later in the M_Transformation diagram.

An *M_StructureItem* is the use of an *M_1dimVariable* (/IsUsageOf association) as the dimension of the *M_Function*. At the same time, an *M_StructureItem* is the use of an *M_1dimSet* (/IsUsageOf association) as the set of allowable values (also the "domain in use") for the M_1dimVariable referred to within the *M_Function*.

In addition to the main functional dependence, i.e. the one between all the independent *M_Variables* and the dependent ones, other functional dependencies can exist in an *M_Function*, in which an *M_StructureItem* is dependent on a group of *M_StructureItems* that do not coincide with the independent ones. Such dependencies are expressed by the /DependsOn association. In this case, the values of the dependent *M_Variable* are determined by the values of the *M_Variables* it depends on.

Attributes:

- IsIndependent [mandatory]: if "true", the *M_1dimVariable* referred to is independent (also called the "key" variable); if "false", it is dependent.
- IsAttribute [optional]: if "false", the *M_1dimVariable* referred to describes a property type relevant to the whole set of independent *M_Variables* of the *M_Function* (a "proper measure" of the *M_Function*), if "true" it describes a property type of all the occurrences or some of them (an "attribute" of the *M_Function*).[50]
- IsReferenceTime [mandatory]: if "true", the *M_StructureItem* is independent and refers to a time instant based *M_Variable* that plays the role of reference time.
- IsSender [mandatory]: if "true" the *M_StructureItem* refers to an *M_Variable* that contributes to the identification of the sender of the observation (e.g. the reporting agent for primary reporting, the country for secondary reporting).
- IsFactKey [mandatory]: if "true" the *M_StructureItem* is independent and has a meaning that refers to other *M_StructureItems* representing measures. For example, the two following data structures are semantically equivalent, but in the latter the *M_StructureItem* referring to the variable "Economic Phenomenon" is a "fact key"[51] because it gives a meaning to the generic measure "Total Amount", that would be meaningless "per se":

  Data structure no.1

CUSTOMER RESIDENCE	CURRENCY	LOANS - TOTAL AMOUNT	CURRENT ACCOUNTS - TOTAL AMOUNT
Rome	Euro	1.000.000	2.000.000

  Data structure no.2

CUSTOMER RESIDENCE	CURRENCY	ECONOMIC PHENOMENON	TOTAL AMOUNT
Rome	Euro	Loans	1.000.000
Rome	Euro	Current account	2.000.000

- IsAggregated [mandatory]: if "true", the *M_Set* referred to contains one and only one *M_Element*, describing a property common to all the observations of the *M_Function*.

---

50 Note that an "attribute" is not an independent *M_Variable*, therefore the *M_Fucntion* graph cannot contain two different observations that differ only in their attribute values.

51 "Fact key" is a term taken from data warehouse terminology.

- `IsCalculated [mandatory]:` if "true", the value of the *M_StructureItem* in the *M_Function* graph is calculated from the values of other *M_StructureItems*.[52]
- `IsImplicit [mandatory]:` if "true", the *M_StructureItem* does not appear explicitly in the graph of the *M_Function*, but it is useful to express some additional meaning and to harmonize the definition space of *M_Functions* with different explicit *M_StructureItems*.

Integrity constraints:

- The *M_Variable* and the *M_Set* used by an *M_StructureItem* must refer to the same *M_Domain* and the *M_Set* must be a subset of the *M_Variable* defining *M_Set*
- An *M_Variable* can be used by any number of *M_StructureItems*.
- An *M_Set* can be used by any number of `M_StructureItems`.
- If `IsAttribute` is "true" then `IsIndependent` is "false".
- In an *M_Function* structure there cannot be two *M_StructureItems* referring to the same *M_Variable* in the same role of dependent or independent variable.
- In an *M_Function* structure there does not necessarily exist a fact key, a reference time or a sender
- If one or more of the independent *M_Variables* of the *M_Function* refers to an *M_Domain* with a historical *M_AlgebraicStructure*, the *M_Function* must have a reference time dimension.
- For an *M_Function* no more than one fact key, reference time and sender can exist, but it is possible to identify each of them by means of more than one *M_StructureItem*; in this case, for all the *M_StructureItems* that identify the fact key (or the reference time, or the sender), the `IsFactKey` (or the `IsReferenceTime` or the `IsSender`) is "true"; as an example for the "fact key", let us examine the following data structure, semantically equivalent to those described above:

Data Structure no.3

CUSTOMER RESIDENCE	CURRENCY	ECONOMIC PHENOMENON	TYPE OF MEASURE	AMOUNT
Rome	Euro	Loans	Total	1000000
Rome	Euro	Current accounts	Total	2000000

In this structure, the *M_StructureItems* referring to the *M_Variables* "EconomicPhenomenon" and "TypeOfMeasure" have the `IsFactKey` property. Of course, this kind of possibility has a practical usefulness when there is more than one possible "type of measure" (e.g. total, min, max, mean, variance, …).

- If `IsFactKey` is "true", `IsIndependent` is also "true"
- If `IsReferenceTime` is "true", `IsIndependent` is also "true"
- If `IsSender` is "true", `IsIndependent` is also "true"
- If `IsCalculated` is "true", then `IsIndependent` is "false"
- When an *M_Function* has more than one "proper measure", the `/DependsOn` relationship can be used to associate the proper attributes to each "proper measure". For example:

Example 13:
Independent variables: Time, Branch, Currency;
Proper measures: total amount of loans, total number of operations;
Attributes: "precision"
`/DependsOn` *Association: The"precision" attribute depends on the "total amount of loans"*

[52] The algorithm is defined as an `M_Transformation`.

- If `IsImplicit` is "true", then `IsAggregated` is "true".

M_CombinationsGroup

An *M_CombinationsGroup* is a group of combinations of values of given *M_1dimVariables*[53] to be excluded from (or allowed in[54]) the Cartesian product of the *M_StructureItems* to specify the set of "a priori" admissible values of the *M_Function*.

Because of an *M_Function* inherits the properties of the *M_Variables* used and the properties of their *M_Domains*, no *M_CombinationsGroup* definition is needed when the exclusions/inclusions are already specified for the *M_NdimSet* that the *M_NdimVariable* is defined on or from the *M_Fullset* of its *M_Domain*. To this end, consider that the *M_Function* implicitly inherits the properties of all the *M_NdimVariables* obtainable as a composition of any number of the *M_1dimVariables* used by its *M_StructureItems*.

The specification of combinations of events to be excluded (or not) is made in different ways depending on the extent to which they are impossible (or possible): if "always", it appears in the definition of the *M_Fullset* of the relevant *M_NdimDomain*, if "for a given *M_NdimVariable*", it is made in the relevant *M_NdimSet*, if "for given *M_Functions*", it is made by means of an *M_CombinationsGroup*.

An *M_CombinationsGroup* uses an *M_NdimVariable* (`/UsesVariable` association) and an *M_NdimSet* in which the *M_NdimVariable* takes values (`/UsesSet` association). A "combination" is a value of the *M_NdimVariable*, which is an *M_Element* of the *M_NdimSet*.[55]

An *M_CombinationsGroup* can be allowed (`/Allows` association) or not allowed (`/NotAllows`) by any number of *M_Functions*, provided every *M_1dimVariable* that composes the *M_NdimVariable* used by the *M_CombinationsGroup* is also part of the *M_Function* structure (i.e. is used by an *M_StructureItem* of the *M_Function*). If the *M_CombinationsGroup* is allowed, it implies that all the combinations not belonging to it are "not allowed".

The choice between the `/Allows` and the `/NotAllows` association depends on the ease of definition: when only a few combinations have to be excluded, it would probably be better to use the `/NotAllows`, if almost all the combinations have to be excluded, it would probably be better to use the `/Allows`, defining the allowed ones because they are fewer.

An *M_CombinationsGroup* is a relation (a subset of the Cartesian product) between a number of *M_Variables* and relevant *M_Sets*, therefore a typical *M_CombinationsGroup* looks like this:

---

[53] A combination is thought of as a set of values, one for each *M_1dimVariable*; therefore, a combination is a value of the *M_NdimVariable* that is constituted of the given *M_1dimVariables*

[54] I.e. the allowed combinations are maintained and all the other combinations are excluded

[55] In other words, the *M_NdimSet* contains the combinations.

Var1	*Var2*
1	*A*
1	*B*
2	*B*
..	..

Attributes:

- `Description` [mandatory]

Integrity constraints:

- The *`M_Variable`* and the *`M_Set`* used by an *`M_CombinationsGroup`* must refer to the same *`M_Domain`*
- An *`M_Variable`* can be used by any number of *`M_CombinationsGroups`*.
- An *`M_Set`* can be used by any number of *`M_CombinationsGroups`*.

Description of the functions about functions metamodel

The functions about functions are data that give information about other data. They are defined using the same metamodel as a generic function[56] but in addition they make references to the functions which they give information on.

An *M_FunctionAboutFunction* can provide information about any number of *M_Functions*; information on an *M_Function* can be given by means of any number of *M_FunctionAboutFunctions* (/ProvidesInformationOn association).

The level of detail of the information can vary from a minimum (i.e. referred to the whole *M_Function*) to a maximum (i.e. referred to the single *M_Function* occurrence[57]), passing through all the intermediate possibilities (e.g. the occurrences characterized by some values of some *M_Variables* of the *M_Function*). The independent *M_Variables* of an *M_FunctionAboutFunction* are therefore a convenient subset of the independent *M_Variables* of the described *M_Functions*. By convention, the property expressed by a value of the *M_FunctionAboutFunction* is referred to the observations of the described *M_Functions* having the same values for the common independent *M_Variables*. The *M_Function* name described can also be included as independent *M_Variable* in the *M_FunctionAboutFunction* structure.

There can be many types of function about function, depending on the information they contain. The main types are "attribute" data, "status" data and "quality" data. Although these three cases only are represented in the class diagram below, the list of types of *M_FunctionAboutFunction* is considered open ended, i.e. it can be extended by the Administrator of the *M_Functions* to include also cases of interest in the specific environment.

"Attribute" data describe attributes of the *M_Functions* referred to, i.e. properties of the *M_Functions* occurrences (e.g. true value, estimated value, seasonally adjusted, …). The "attribute" data can act as a specification for producing the described data or can simply be documentation.

"Status" data give information about the so-called "knowledge domain" of the *M_Functions*, i.e. the availability of the observations in the data warehouse. In many cases, in fact, the points at which the *M_Function* is known cannot be inferred completely from the *M_Function* graph and have to be declared explicitly (or simply this may be appropriate). It is useful, for example, to document whether the expected respondents of a survey have responded or not, the dates for which a periodical survey has been executed, the cases in which derived data have also been calculated (or not).

"Status" data can be necessary, for example, when the "known" graph is not completely represented, intentionally omitting some observations, such as those in which the measures are zero.[58] The lack of an observation results in uncertainty if the measure is unknown or zero.

---

[56] Note that this approach ensures that the data about data is defined and treated exactly like the described data and that they can be stored in the same data warehouse.

[57] In this case, the information can also be put in the *M_Function* itself in the form of additional attributes.

[58] Such a convention is common for example for quantitative *M_Functions* where the possible combinations are

***Example 14*:** a "status" data [59]:

M_Function	***Sender***	***Date***	***Knowledge***
F1	*A*	*2005.12.31*	*Known*
F1	*B*	*2005.12.31*	*Known*
F2	*B*	*2005.12.31*	*Not known*
..	..	..	..

The "status" data of the *M_Functions* are usually produced by the software artefacts that process the graph.

The "quality" data describe some quality attribute of the referenced *M_Functions*. They usually measure "if" and "how" the observations in the data warehouse conform to a given set of not mandatory but recommended "integrity" rules.

Example 15:
M_Function no. 1: amount of loans classified by currency, counterparty country, activity sector
M_Function no. 2: amount of loans classified by currency, maturity
Integrity rule: the M_Functions no .1 and no .2, aggregated at the "currency" detail, should coincide
Quality data: Unbalance of M_Functions(no. 2 – no. 1) classified by currency (the unbalance measures the amount of the error)

In this example, the quality data can be calculated by means of an *M_Transformation* (see in the "Description of the transformations metamodel" section). The basic idea is that statistical data can also be defined and calculated to give a "measure" of the quality of other data. The operands are the "input" data (whose quality has to be measured) and the result is the quality data. Note that in this case the integrity rule is formally expressed as an *M_Transformation* definition. However, quality data are not necessarily calculated.

Quality data can also contain information on formal errors (e.g. missing or unrecognized values, lack of integrity).

---

so many that it would be burdensome to represent them all (e.g. to store also observations whose measure is equal zero), and only significant observations are exchanged / processed / stored. This choice, however, makes the interpretation of the "observations not physically available" ambiguous: in fact their measures could be "zero" or the observations not have been sent by the provider / not stored / not calculated and therefore "unknown".

[59] The example is taken from the Bank of Italy information system (in the real case the dependent variable, whose *M_Elements* are 1=known and 0=unknown, is not expressed explicitly, because the zero values are omitted and the "1" values are implied by the presence of the occurrence).

Functions about functions: class diagram

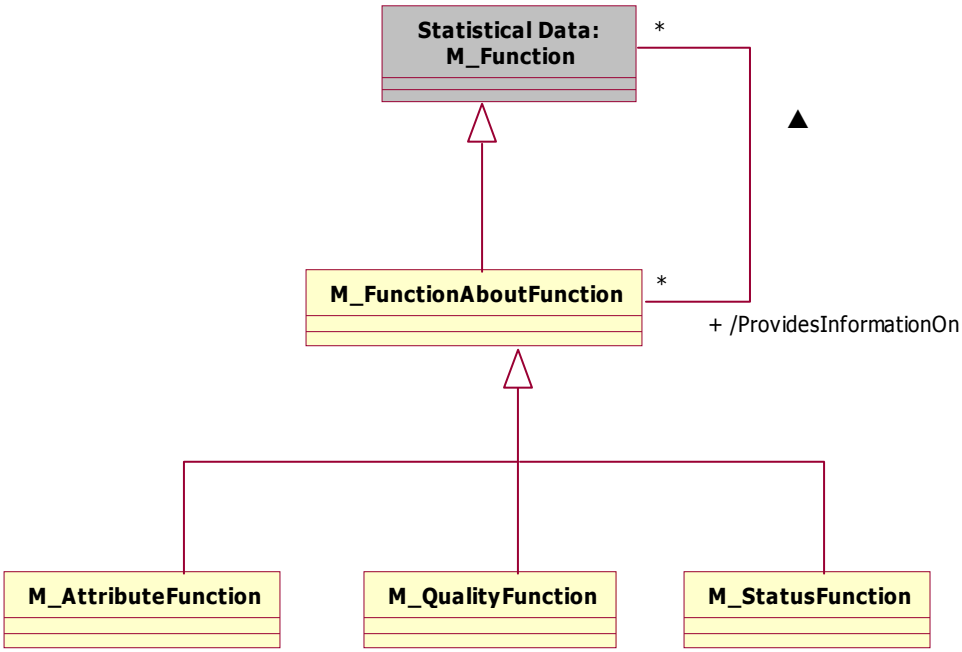


Diagram 6: *Functions about functions metamodel*

Transformations

Description of the transformations metamodel

An *M_Transformation* is thought of as a process that calculates a "result" by applying an "algorithm" to one ore more "operands". The result and the operands are called *M_TransformationMembers*. Each *M_TransformationMember* can be either an *M_Function* or an *M_Concept*.

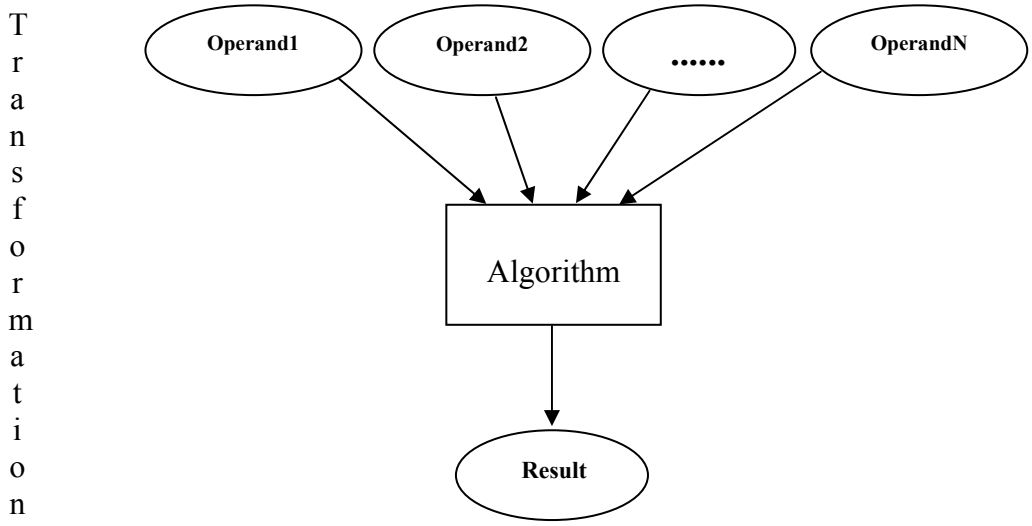


Figure 5: *Transformation*

The external view of an *M_Transformation* (Figure 5) links the input data to the calculated data (which in turn can be inputs of other *M_Transformations*), documents the calculus sequences, allows the active software to determine the calculus order and to establish which calculated data have to be refreshed when collected data changes.

The internal view of an *M_Transformation* deals with the way its algorithm is specified.[60] In fact it can be composite, can make use of many *M_Operators* and combine them in a hierarchical graph, exactly like in an expression. In the following example, given two functions F_1 and F_2 having the same *M_Variables*, an *M_Transformation* calculates the *M_Function* F_3 as the percentage of F_1 on the sum of F_1 and F_2 using the operators +, *, / defined among *M_Functions*:

$$F_3 = F_1 / (F_1 + F_2) * 100$$

This *M_Transformation* has three steps, the first calculates the partial result:

$$P_1 = (F_1 + F_2),$$

the second produces another partial result:

$$P_2 = F_1 / P_1$$

and the third obtains the final result as:

$$F_3 = P_2 * 100.$$

[60] The internal view of an *M_Transformation* is fully compliant with the "expression metamodel" of the CWM foundation packages (see [13]), but in addition it is historical.

This is the hierarchical graph of the *M_Transformation* in the example:

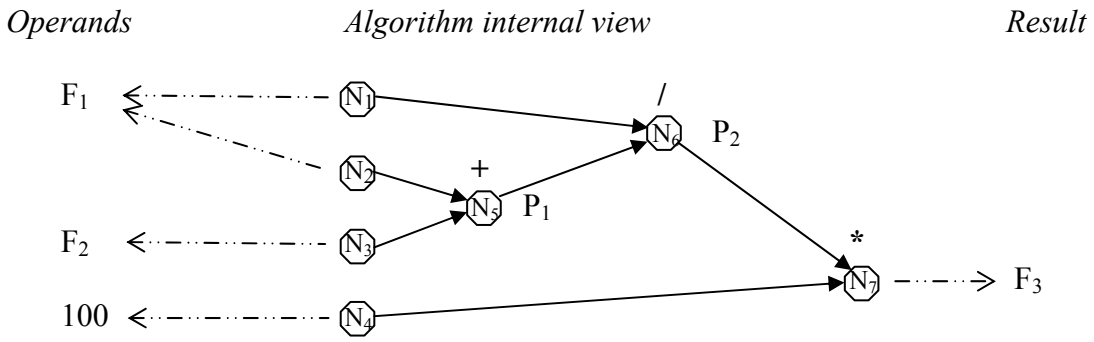


Figure 6: *Transformation graph*

The steps of an *M_Transformation* and the references to the external operands are called *M_TransformationNodes* (shown as circles in the graph above). There exist three types of *M_TransformationNodes*: those that reference an *M_Operator* (+, *, /) called *M_OperatorNodes*, those that reference a costant (100) called *M_ConstantNodes*, and those that reference an item of the model that is either an *M_Function* or an *M_Concept* (F_1, F_1 again, F_2) called *M_ModelItemNodes*.

The links between the steps (shown with arrows in the graph above) are represented in the "class diagram" by mean of an association. The links are hierarchical and oriented from the "child" (the arrow tail) toward the "parent" (the arrow head). A "parent" must be an *M_OperatorNode*.

Note that while the operands and the result must be defined in the proper classes of the metamodel, the partial results (P_1 and P_2 in the example) are not defined, because they serve only for the calculation.

M_Operators and *M_Constants* are registered in the classes having the same name. The grammar of the *M_Operators* is left free, so very different kinds of operators can be introduced as necessary and incrementally (e.g. algebraic, logical, statistical, for data manipulation, and so on). Obviously, the type of child nodes (i.e. *M_Function* of various types, *M_Variable*, *M_Set*, *M_Element*, scalar constant, matrix constant, …) must be equal to the type of the input parameters provided for by the parent *M_Operator*. An operator can be described by a text structured as an expression in which other operators can be composed (`ExpressionTemplate` attribute of the `M_Operator` class). When an *M_Operator* must be "actively" used by automatic processes, it makes reference to a software routine able to perform it (such a routine is automatically invoked by the software that performs the calculus sequence).

Even if an *M_Transformation* provides the specification of a calculus, the calculation is not necessarily performed within the information system. For example, the definition of the *M_Tranformations* can be used to tell the data sender how to calculate the data to be sent or can be used by the data sender to document how the data sent were calculated. For this reason, the *M_TransformationResult* class refers to (`/RefersTo` association) the *M_DerivedFunctions* (see the "Description of the statistical data metamodel" section) rather than the "calculated" ones.

As for the general case, the classes and the associations of the transformations section are also historical. Consequently the *M_Transformation* structure can change with respect to the time: the whole *M_Transformation* can be considered as a set of component *M_Transformations*, one for each “reference time” value.

Transformations metamodel: class diagram

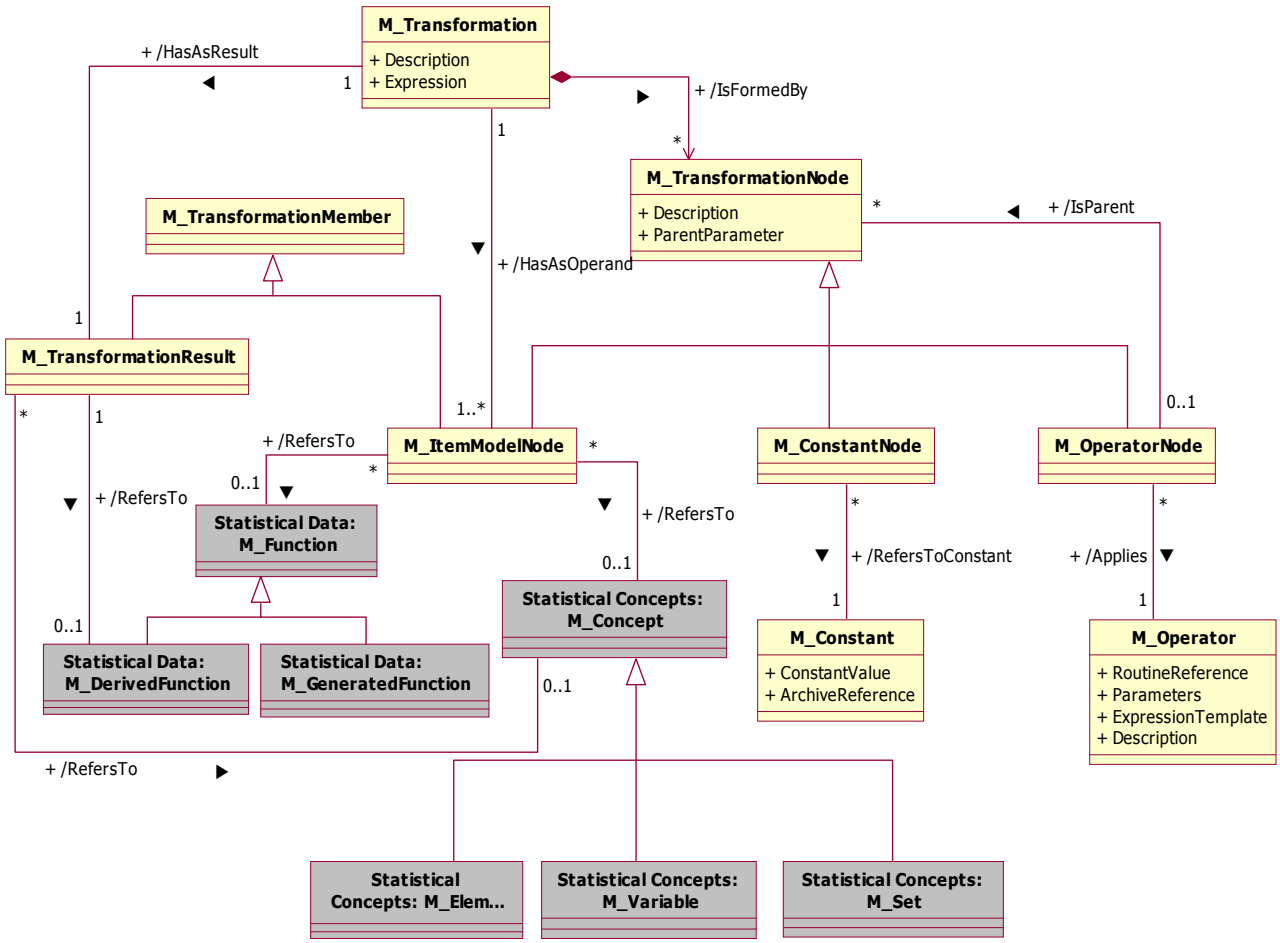


Diagram 7: *Transformations metamodel*

Diagram description : definitions

M_Transformation

This class registers the *M_Transformations* defined in the information system. An *M_Transformation* is an expression (i.e. is a finite and ordered sequence of operators applied on some operands to give a result) that is defined through a hierarchical tree and, optionally, through an equivalent textual expression (Expression attribute).

An *M_Transformation* has one result (/HasAsResult association) and many operands (/HasAsOperand association).

An *M_Transformation* is formed by any number of *M_TransformationNodes* (/IsFormedBy association) that are structured as a tree, in which the result of an *M_TransformationNode* is the argument supplied to another *M_TransformationNode*, that is the immediately higher level node in the tree, and there is a single highest level *M_TransformationNode* (the root node), that gives the result of the whole *M_Transformation*.

Attributes:

- Expression [optional]: a textual but structured representation of the *M_Transformation* expression that uses proper identifiers for operators and operands and uses special symbols (e.g. brackets) to specify the sequence of the applied operators.
- Description [optional]

Integrity rules

- The operands of an *M_Transformation* are the *M_ItemModelNodes* it consists of.
- The structure of the textual expression, if present, must match the structured tree of the *M_TransformationNodes*

M_TransformationMember

An *M_TransformationMember* is a result (*M_TransformationResult*) or an operand (*M_ItemModelNode*) of an *M_Transformation*.

M_TransformationResult

- Class of the results of the *M_Transformations*. An *M_TransformationResult* refers either to an *M_DerivedFunction* or to an *M_Concept*, never to both (/RefersTo association)

Integrity rules:

- The *M_TransformationResult* is the outcome of the root node of the *M_Transformation*

- An *M_TransformationResult* refers either to an *M_DerivedFunction* or to an *M_Concept*, never to both
- An *M_Function* can be referred to by one and only one *M_TransformationResult* (in other words, an *M_Transformation* specifies the only method to be used to calculate the *M_Function* graph)
- An *M_Concept* can be referred to by any number of *M_TransformationResults* (in other words, an *M_Transformation* specifies one of the possible methods to obtain an *M_Concept*)

M_TransformationNode
(and M_ItemModelNode, M_OperatorNode, M_ConstantNode)

The class registers the nodes of the *M_Transformations*, i.e. their "atomic steps". An *M_TransformationNode* can be of different types: an *M_ItemModelNode* (which refers to an item of the Matrix metamodel: an *M_Concept*, an *M_Function*), an *M_OperatorNode* (which refers to a composition law) or an *M_ConstantNode* (a constant used in the expression, also with composite structure).

An *M_ItemModelNode* is a use of an item of the metamodel as an external operand of an *M_Transformation* (/HasAsOperand association) and refers to either an *M_Function* or an *M_Concept* (/RefersTo associations). An *M_Function* or an *M_Concept* can be referred to by many *M_ItemModelNodes*.
An *M_OperatorNode* is a use of an *M_Operator* within an *M_Transformation* (/Applies association).
An *M_ConstantNode* is a use of an *M_Constant* within an *M_Transformation* (/RefersToConstant association).

An *M_TransformationNode* participates in the hierarchical tree structure and can have 0 or 1 parent *M_OperatorNode* (/IsParent association). The root node of an *M_Transformation* has no parent node, the other nodes have one parent node. An *M_OperatorNode* is the parent of any number of *M_TransformationNodes*.

Attributes:
- Description [optional]
- ParentParameter [mandatory, except for the root node]: the parameter of the *M_Operator* used by the parent *M_OperatorNode*, which the child node is given to as an argument;

Integrity rules:
- An *M_TransformationNode* belongs to one and only one *M_Transformation*
- *M_TransformationNodes* for which an *M_OperatorNode* is the parent must correspond to the parameters of the applied *M_Operator* in type and multiplicity (every *M_TransformationNode* must be the argument of one of the parameters of the *M_Operator*).

M_Constant

This class represents the constant values that are usable in the *M_Transformations* (through the *M_ConstantNodes*). Scalar or structured constants are allowed.
The value of an *M_Constant* can be represented either by using the ConstantValue attribute (e.g. when the constant is a scalar quantity such as a number or a character string) or by providing a link to an external data structure containing the constant values (a flat file, a relational table and so on).

Attributes:

- ConstantValue [optional]: the constant value
- ArchiveReference [optional]: the link (url, location, etc.) to the external structure containing the constant value.

M_Operator

This class registers the *M_Operators* applicable within the *M_Transformations* (through the *M_OperatorNodes*). An *M_Operator* can be applied by many *M_OperatorsNodes*.

An *M_Operator* is a composition law that composes a certain number of input parameters and gives one or more output parameters. Input and output parameters have a predefined data type and multiplicity.

An *M_Operator* has as attribute a human readable description and a link to a software routine able to perform it. An *M_Operator* can be composite, i.e. constituted of a structured symbolic expression that makes use of other *M_Operators* and of the generic parameters of the *M_Operators* (ExpressionTemplate attribute), for example imagine an operator that calculates the percentage of an *M_Function* with respect to another *M_Function*:

Operator : %
Expression template: $F_0 = (F_1 / F_2) * 100$
F_0 = *expression result (Type=*M_Function*, multiplicity=1)*
F_1 and F_2 = *expression operands (each one of type =* M_Function *and multiplicity = 1)*
*/ , ** = *operators used in the expression*

Attributes:

- RoutineReference [optional]: the link (url, location, etc.) to the external structure containing the software able to perform the operator.
- Parameters [mandatory]: Specification of the parameters of the *M_Operator*, with their role (input or output), data type and multiplicity
- ExpressionTemplate [optional]: Character string containing the symbolic expression that describes the algorithm.
- Description [optional].

Other related models

The representation of the **references to external data** and of the **administration system** of the statistical hierarchy models is based on two proper models.

References and administration rules are designed to be linked to every model at every level of the statistical hierarchy.

The basic idea is to define a generic entity "resource" (that will be a "documentable resource" in the Reference Model and an "administrable resource" in the Administration Model) so that the components of any model at any level of the statistical hierarchy can be considered as instances of this entity.

The properties and relationships of the entity "resource", therefore, will be valid for all the statistical hierarchy components.

***Example**:*
A "note" could be linked to a generic `DocumentableResource`.
Since any component of the statistical hierarchy is a DocumentableResource, a note could be linked in the same way to any kind of instance at any level of that hierarchy (e.g. to the `M_Domain` *concept, which is an instance of the level 3 Matrix metamodel, or to the '**Securities**'* `M_Domain`*, which is an instance of the level 2 model)*

A detailed description of the entity "resource" is beyond the scope of this document; however, broadly speaking, "resources" can have an XML or a relational representation, i.e., in the relational case, each "resource" can be represented by one or more "relational rows" and can be identified using the corresponding "relational key".

External references model

External references: class diagram

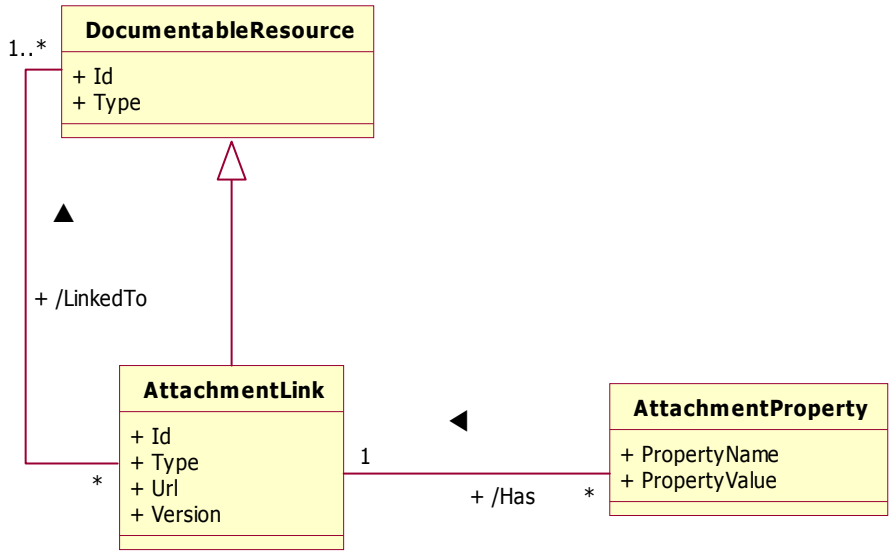


Diagram 8: *External references metamodel*

Diagram description: definitions

`DocumentableResource`

A generic resource is a ***`DocumentableResource`*** when further information can be attached to (`/LinkedTo`) the resource: the information can be structured or not-structured. All the components of the statistical hierarchy and the ***`AttachmentLinks`*** themselves are ***`DocumentableResources`*** (an ***`AttachmentLink`*** can be linked to another ***`AttachmentLink`***).

Attributes:

- `Id` [mandatory]: the unique resource identifier within the system; the identifier is based on the relational representation of the resources.
- `Type` [mandatory]: resource type; this property can be used to classify different types of resources in the information system.

`AttachmentLink`

This class represents the link to the additional information (i.e. notes, documents,..) that can be linked to the resources (`/LinkedTo`). Through its link an attachment is itself a resource. Note that through this class the model describes the way attachments are linked to resources but does not establish any rule about attachment structure.

Attributes:

- `Id` [mandatory]: `:` the unique attachment identifier within the system
- `Type` [optional]`:` this property can be used to classify different types of attachment in the information system (Notes, Documents, …).
- `Url` [mandatory] `:` specifies the location where the attachment is stored to permit its retrieval.
- `Version` [optional] `:` identifier of the version of the attachment.

`AttachmentProperty`

It is possible to enrich the information about the attachment (`/Has` relationship) by specifying further *`AttachmentProperties`*. Since an attachment is not a model entity, its properties are not fixed and can be defined according to the attachment type.

Attributes:

- `Name` [mandatory]`:` property name
- `Value` [mandatory]`:` property value

Administration model

Administration: class diagram

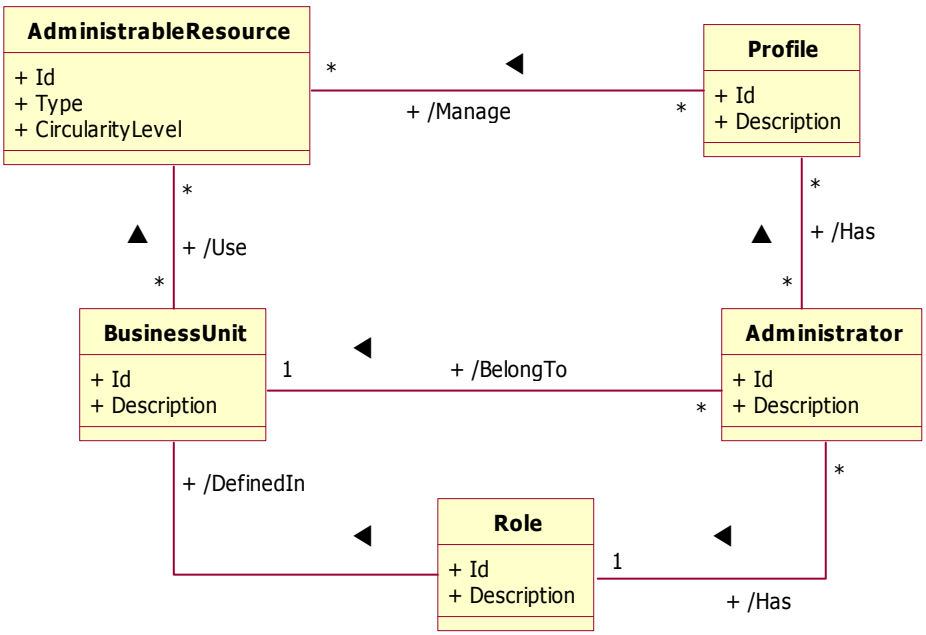


Diagram 9: *Administration metamodel*

Diagram description: definitions

`AdministrableResource`

A generic resource is classified as an ***`AdministrableResource`*** when it is needed to document the roles and competencies necessary to administer it.

An ***`AdministrableResource`*** has information associated with it about the administrators (users who are responsible for the maintenance of the resource) and about the circularity level (the use of the resource in the system can be free or restricted to specific company areas). Hence, for each ***`AdministrableResource`*** it is necessary to define the set of ***`Profiles`*** authorized for its management (`/Manage` association) and the set of company business areas that are allowed to use it (`/Use` association).

Attributes:

- `Id` [mandatory]: the unique resource identifier within the system; the identifier is based on the relational representation of the resources.
- `Type` [optional]: resource type; this property can be used to classify different types of resources in the information system.
- `CircularityLevel` [mandatory]: specifies the circularity level of the resource, related to the fact that its management is based on specific requisites and needs:
    - PU: "public" resource, available for all the ***`BusinessUnits`***.

- PR: “private” resource, available for specific *BusinessUnits*.
- UT: “user” resource, available for a specific user.

Profile

Represents the authorization profile assigned to maintain (/Manage association) the *AdministrableResource*. A resource can be managed by many *Profiles* and the same *Profile* can manage many resources. *Profiles* refer to a number (/Has association) of *Administrators*.

Attributes:

- Id [mandatory]: profile identifier.
- Description [mandatory]: profile description.

Administrator

Represents the “logical” user assigned to the maintenance of resources. An *Administrator* must be associated with one or more *Profiles* (/Has association) through which the *Administrator* can be linked to the *AdministrableResource*. The *Administrator* has a *Role* (/Has association) within the *BusinessUnit* he belongs to (/BelongTo association).

Attributes:

- Id [mandatory]: administrator identifier.
- Description [mandatory]: administrator description.

Role

Represents the business roles defined within a *BusinessUnit* (/DefinedIn association). Each *BusinessUnit* comprises many *Roles* and each *Role* can be associated with many *Administrators*.

Attributes:

- Id [mandatory]:: role identifier.
- Description [mandatory]:: role description.

BusinessUnit

Represents the component of the organization. *BusinessUnits* use (/Use association) *AdministrableResources*.

Attributes:

- Id [mandatory]: business unit identifier.
- Description [mandatory]: business unit description.

APPENDIX
A short guide to UML notation

Introduction

This section gives a brief overview of the use of UML notation in this document. The examples are taken from the diagrams.

Among the UML modelling tools, only the "class diagram" is used.

Because the diagrams are not intended to give all the information needed for an EDP implementation, some UML indications are not used (see below). On the other hand, the historicity of the metamodel is not rendered in UML notation (this is to say that the classes and the associations of the metamodel are in general dependent on time: this topic is also treated in the "Historicity and other common properties" section).

Classes and their Attributes

In UML, an object class is something of interest for the user and models the characteristics and the behaviour of some objects.

The UML graphic representation of a class is a rectangle split into three compartments. The top compartment is for the class name (which is mandatory), the second is for attributes (characteristics) and the last is for operations (behaviours). In this document only the data structures are modelled, so behaviours are not indicated and then the third compartment is left empty.

As a naming convention, the names of classes and attributes are assembled using one or more words linked together and capitalizing the first letter of each word. The names of the classes of the Matrix metamodel are preceded by an "M_" to distinguish them from terms used in the common language, according to the terminology principles set out in [6], [8].

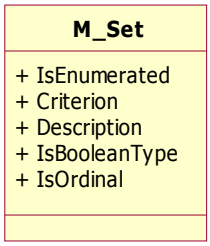


Figure A1 Class and its attributes

In Figure "A1" the name of the class is `M_Set` and its attributes are `IsEnumerated`, `Criterion`, `Description`, `IsBooleanType` and `IsOrdinal`[61] (note that the full identity of an attribute includes its class, e.g. a complete attribute name is "`M_Set.Description`").

61 Attributes named "Is…Something…" have two possible values: "true" and "false".

In UML the attribute name is preceded by the symbol "+" (to mean public accessibility) or "-" (private). In this document all the attributes are considered "public".

Every class of the metamodel has an identifier, which is an attribute (or more than one) whose value uniquely identifies an instance of the class. The identifiers are not shown in the diagrams of this document although they are presumed to exist for every class, to be mandatory and to be named by adding "Id" after the name of the class (e.g. `M_SetId`, `M_FunctionId`, …).

Attributes referring to other classes, whose existence is implied by an association, are not explicitly indicated.

The UML distinction between "abstract" and "concrete" classes is not indicated.

Because the metamodel is divided into parts, it is possible to see the same class in more than one diagram. The complete description of the class meaning and of its attributes, however, is presented in one diagram only (the diagram of the section to which the class is most related, usually the first diagram in which the class appears). In the other diagrams the class is drawn just to be referred to, the rectangle is painted with a grey background and its name is preceded by the name of the diagram the class belongs to.

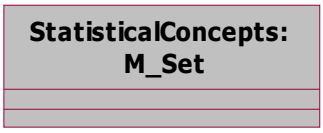


Figure A2 - Class belonging to another class diagram

Inheritance

A class can be specialized by subclasses that inherit all the attributes (and operations) of the superclass, but not the associations. A subclass can have attributes (and operations) in addition to those of the superclass.

Figure A3 – Subclass that inherits from a superclass

In Figure A3 the class `M_FreeHierarchy` is a subclass of the `M_Hierarchy` class.

Associations

Because of the historicity of the metamodel, the links between the instances of the classes can vary with time. In this context, the multiplicity of the associations would always be many to many. To

avoid such a loss of descriptive power, in this document the multiplicity of the associations is described with reference to the single, generic time instant.

The navigability of associations is intended to be always in both directions. In addition to the classical UML notation, a full black triangular arrow specifies the sense in which the name of the relationship should be read.

There are various kinds of association types that make it possible to explain the relationships existing between classes.

Simple association

A simple association links two classes to each other by specifying the multiplicity of the link.

In UML it is possible to specify a variety of "multiplicity" rules. The ones used in the diagrams are:

0..1	: Zero or one
1	: One and only one
1..*	: One or many
*	: Zero, one or many (any number)

Figure A4 - Simple association

Figure A4 shows the simple association between class A and class B. The multiplicity specifies that an object of B can be associated with one and only one object of A and that an object of A can be related to one or many objects of B.

It is possible to give a name to an association. Naming associations is useful to describe their purposes. Like class and attribute names, the names of associations are also assembled using one or more words linked together and capitalizing the first letter of each word. Every association name is preceded by '/' and the visibility indicator (i.e. '+' for public, '-' for private). In this document all the associations are considered "public".

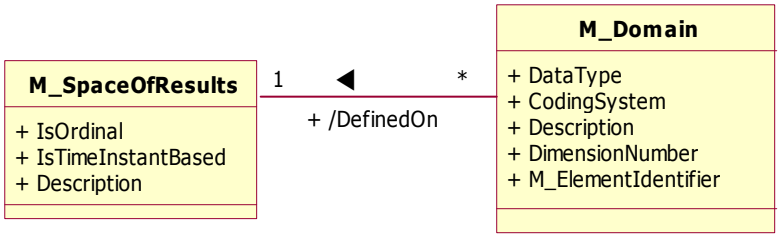


Figure A5 Association names

In the case of two classes joined by more than one association it is extremely useful to name the purpose of the association to explain the diagram completely. Figure A6 shows this case.

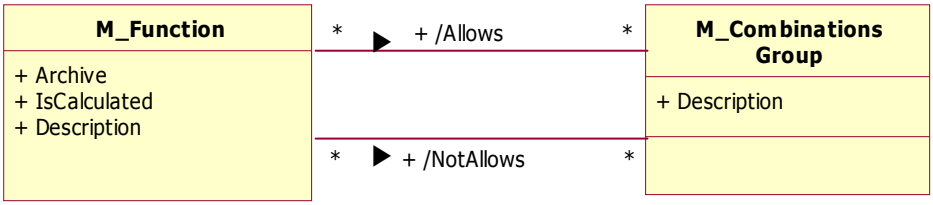


Figure A6 Association names

Aggregation

Simple Aggregation

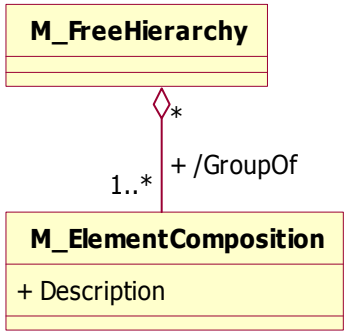


Figure A7 A simple aggregate association

Aggregation is a special kind of association indicating that a set of classes (subordinate classes) are linked to build another class (aggregated or parent class). In a simple aggregation association, an instance of a subordinate class can survive the related instance of the parent class. The symbol representing an aggregation relationship is an unfilled diamond shape on the side of the aggregated class. Figure A7 shows an example of an aggregation relationship between `M_FreeHierarchy` and `M_ElementComposition`.

Composition

The composition (or composite aggregation) relationship is a stronger kind of aggregation. In a composition the lifecycle of the instance of a subordinate class is influenced by that of the parent. An instance of a subordinate class cannot survive the related instance of the parent class. The UML convention represents composition as an aggregation, but in this case the diamond shape is filled. Figure A8 shows a composition relationship between `M_Function` and `M_StructureItem`.

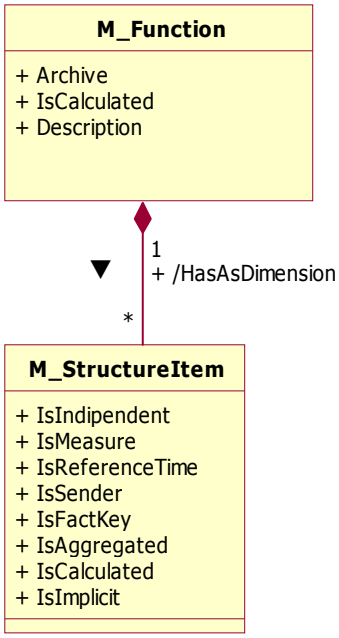


Figure A8 An aggregate association by composition

REFERENCES

[1] Banca d'Italia – "The monetary and financial statistics of the Bank of Italy" – March 1994.

[2] Banca D'Italia - "L'informazione statistica nell'attività della Banca centrale" (a cura del Comitato per le statistiche creditizie e finanziarie, coordinatore C. Conigliani), Tematiche istituzionali - Ottobre 1996.

[3] Ciampi C.A. "La statistica nell'attività della Banca d'Italia" - intervento all'Università degli Studi di Roma La Sapienza - Banca d'Italia - Bollettino economico - n. 20 - 1993.

[4] Del Vecchio V. - "Statistical data and concepts representation" (English translation of "La Rappresentazione dei dati e dei concetti statistici" - Banca D'Italia - Tematiche Aziendali - settembre 1997)

[5] Del Vecchio V. - "The Banca d'Italia's active statistical meta-information system" – European Commission Information Society Technologies Programme - MetaNet Project - Proceedings of 1st MetaNet Conference - 2-4 April 2001

[6] Del Vecchio V. – "The multi-level model of the statistical information system in the Bank of Italy" – (Bank of Italy internal paper) - October 2002

[7] European Board for Edi Standardisation / EG6 Statistics – "CLASET- Concepts and terms related to classifications & exchange of classifications – European Commission CLASET initiative – EG6/WG3/mda/96005 – June 1996.

[8] Froeschl K.A.; Grossmann W; Del Vecchio V.; *The Concept of Statistical Metadata,* – European Commission Information Society Technologies Programme - MetaNet Project - Deliverable 5 – Modeling levels in statistical information systems. [February 2003]

[9] Maggiolini M. – "The software generalization process" – (Bank of Italy internal paper) - March 1999

[10] Maggiolini - "Statistical Data Processing Strategy in Banca d'Italia" - (Bank of Italy internal paper) - October 2002

[11] Milani, P. – "Procedural steps and software support in statistics processing" - (Bank of Italy internal paper) - October 2002

[12] Object Management Group - "Meta Object Facility (MOF) Specification" – March 2000.

[13] Object Management Group - "Common Warehouse Metamodel (CWM) specification" – February 2000.

[14] Sundgren B. - "Statistical Metainformation and Metainformation Systems" - Statistic Sweden R&D Report -1991:11.

.