INTERNATIONAL STANDARD

**ISO 19103**

First edition
2015-12-01

# Geographic information — Conceptual schema language

*Information géographique — Langage de schéma conceptuel*

Reference number
ISO 19103:2015(E)

# Contents

Page

## Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2. www.iso.org/directives

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received. www.iso.org/patents

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/TC 211, *Geographic information/Geomatics*.

This first edition of ISO 19103:2015 cancels and replaces the first edition (ISO/TS 19103:2005).

# Introduction

This International Standard of the ISO geographic information suite of standards is concerned with the adoption and use of a conceptual schema language (CSL) for developing computer interpretable models, or schemas, of geographic information. Standardization of geographic information requires the use of a formal CSL to specify unambiguous schemas that can serve as a basis for data interchange and the definition of interoperable services. An important goal of the ISO geographic information suite of standards is to create a framework in which data interchange and service interoperability can be realized across multiple implementation environments. The adoption and consistent use of a CSL to specify geographic information is of fundamental importance in achieving this goal.

There are two aspects to this International Standard. First, a CSL is selected that meets the requirements for rigorous representation of geographic information. This International Standard identifies the combination of the Unified Modeling Language (UML) static structure diagram with its associated Object Constraint Language (OCL) and a set of basic type definitions as the conceptual schema language for specification of geographic information. Secondly, this International Standard provides guidelines on how UML should be used to create geographic information models that are a basis for achieving the goal of interoperability.

One goal of the ISO geographic information suite of standards using UML models is that they will provide a basis for model based mapping to encoding schemas like the ones defined in ISO 19118, as well as a basis for creating implementation specifications for implementation profiles for various other environments.

This International Standard describes the general metamodel for use of UML in the context of the ISO geographic information series of standards. Aspects specifically dealing with the modelling of application schemas are described in ISO 19109.

This International Standard is a revision of a previous version from 2005. Changes are documented in Clause 5.

# Geographic information — Conceptual schema language

## 1 Scope

This International Standard provides rules and guidelines for the use of a conceptual schema language within the context of geographic information. The chosen conceptual schema language is the Unified Modeling Language (UML).

This International Standard provides a profile of the Unified Modelling Language (UML).

The standardization target type of this standard is UML schemas describing geographic information.

## 2 Conformance

### 2.1 Introduction

This International Standard defines three levels of conformance classes:

— UML version

— Data types

— Model documentation

To conform to this International Standard, the usage of a conceptual schema language shall satisfy all of the requirements specified in one of the three levels of conformance described below, with the corresponding abstract test suite in Annex A.

### 2.2 UML version conformance

#### 2.2.1 UML 2 conformance class

Table 1 describes the conformance class for UML 2.

**Table 1 — UML 2 conformance class**

| | |
|---|---|
| Conformance class identifier | UML2 |
| Standardization target type | UML2 schemas for geographic information |
| Dependency | ISO/IEC 19505-2:2012, Clause 2<br>OCL 2.3.1 |
| Requirements | All requirements in 6.2 to 6.12 except Requirement 2, and including Requirement 26. |
| Tests | All tests in A.1.2 |

#### 2.2.2 UML 1 to UML2 mapping conformance class

Table 2 describes the conformance class for mapping from UML 1.

**Table 2 — UML 1 to UML 2 mapping conformance class**

| | |
|---|---|
| Conformance class identifier | UML1 |
| Standardization target type | UML1 schemas for geographic information |
| Dependency | UML2<br>ConformantSchema<br>ISO/IEC 19501:2005, Clause 2 |
| Requirements | All requirements in Annex B |
| Tests | All tests in A.1.3 |

### 2.2.3 Conformant schema conformance class

Table 3 describes the conformance class for non-UML schemas.

NOTE Non-UML schemas are considered conformant if there is a well-defined mapping from a model in the source language into an equivalent model in UML and that this model in UML is conformant.

**Table 3 — Conformant schema conformance class**

| | |
|---|---|
| Conformance class identifier | ConformantSchema |
| Standardization target type | Schemas for geographic information |
| Dependency | UML2 |
| Requirements | Requirement 2 in 6.2. |
| Tests | All tests in A.1.4 |

## 2.3 Data types conformance

### 2.3.1 Introduction

Conceptual schemas that claim conformance with this International Standard may also state that they conform to a named subset of the concepts in the standard. These subsets may be used to document different levels of capabilities or complexities. This International Standard describes two levels of capabilities for the use of data types which are defined in Table 4 and 5.

### 2.3.2 Core types conformance class

Table 4 describes the conformance class for core data types.

**Table 4 — Core types conformance class**

| | |
|---|---|
| Conformance class identifier | CoreTypes |
| Standardization target type | Core types for geographic information |
| Dependency | UML2<br>ISO/IEC 11404:2007<br>ISO 8601:2004 |
| Requirements | All requirements in Clause 7 |
| Tests | All tests in A.2.1 |

### 2.3.3 Core and extension types conformance class

Table 5 describes the conformance class for core and extension data types.

**Table 5 — Core and extension types conformance class**

| | |
|---|---|
| Conformance class identifier | CoreExtendedTypes |
| Standardization target type | Core and extension types for geographic information |
| Dependency | CoreTypes<br>ISO 639<br>ISO 3166<br>RFC 3986 |
| Requirements | All requirements in Annex C |
| Tests | All tests in A.2.2 |

## 2.4 Model documentation conformance

### 2.4.1 Introduction

The UML diagrams and textual description of model elements in a model are most often presented in a document. The specific requirements in this International Standard for presentation of geographic information is an extension of the requirements imposed by UML 2. A separate conformance class is defined for this in Table 6.

### 2.4.2 Model documentation conformance class

Table 6 describes the conformance class for model documentation.

**Table 6 — Model documentation conformance class**

| | |
|---|---|
| Conformance class identifier | ModelDocumentation |
| Standardization target type | Documentation of UML schemas for geographic information |
| Dependency | UML2 |
| Requirements | All requirements in 6.16 |
| Tests | All tests in A.3 |

## 3 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 639 (all parts), *Codes for the representation of names and languages*

ISO 3166 (all parts), *Codes for the representation of names of countries and their subdivisions*

ISO 8601:2004, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 11404:2007, *Information technology — General-Purpose Datatypes (GPD)*

ISO/IEC 19501:2005, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*

ISO/IEC 19505-2:2012, *Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure*

NOTE Unified Modeling Language (UML), *version* 2.4.1, available at http://www.omg.org/spec/UML/

OCL 2.3.1, OMG *Object Constraint Language (OCL), version 2.3.1*, available at <http://www.omg.org/spec/OCL/>

RFC 3986 dated January 2005 on URI Syntax, available at <http://www.ietf.org/rfc/rfc3986.txt>

## 4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

**4.1**
**aggregation**
<UML> special form of *association* (4.4) that specifies a whole-part *relationship* (4.30) between the aggregate (whole) and a *component* (4.9) part

Note 1 to entry: See <UML> *composition* (4.10).

[SOURCE: UML 1]

**4.2**
**application**
manipulation and processing of data in support of user requirements

[SOURCE: ISO 19101-1:2014, 4.1.1]

**4.3**
**application schema**
*conceptual schema* (4.12) for data required by one or more *applications* (4.2)

[SOURCE: ISO 19101-1:2014, 4.1.2]

**4.4**
**association**
<UML> semantic *relationship* (4.30) that can occur between typed *instances* (4.20)

Note 1 to entry: A binary association is an association among exactly two classifiers (4.8) (including the possibility of an association from a classifier to itself).

[SOURCE: UML 2]

**4.5**
**attribute**
<UML> *feature* (4.17) within a *classifier* (4.8) that describes a range of values that *instances* (4.20) of the classifier may hold

[SOURCE: UML 1]

**4.6**
**cardinality**
<UML> number of elements in a set

Note 1 to entry: Contrast with *multiplicity* (4.24), which is the range of possible cardinalities a set can hold.

[SOURCE: UML 1]

**4.7**
**class**
description of a set of *objects* (4.25) that share the same *attributes* (4.5), *operations* (4.26), methods, *relationships* (4.30), and semantics

[SOURCE: UML 1]

**4.8**
**classifier**
<UML> mechanism that describes behavioural and structural *features* (4.17) in any combination

[SOURCE: UML 1]

**4.9**
**component**
<UML> representation of a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment

[SOURCE: UML 2]

**4.10**
**composition**
<UML> *aggregation* (4.1) where the composite *object* (4.25) (whole) has responsibility for the existence and storage of the composed objects (parts)

[SOURCE: UML 2]

**4.11**
**conceptual model**
*model* (4.23) that defines concepts of a universe of discourse

[SOURCE: ISO 19101-1:2014, 4.1.5]

**4.12**
**conceptual schema**
formal description of a *conceptual model* (4.11)

[SOURCE: ISO 19101-1:2014, 4.1.6]

**4.13**
**constraint**
<UML> condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element

[SOURCE: UML 2]

**4.14**
**data type**
specification of a *value domain* (4.37) with *operations* (4.26) allowed on values in this domain

EXAMPLE Integer, Real, Boolean, String and Date.

Note 1 to entry: Data types include primitive predefined *types* (4.36) and user definable types.

**4.15**
**dependency**
<UML> *relationship* (4.30) that signifies that a single or a set of model elements requires other model elements for their specification or implementation

Note 1 to entry: This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).

[SOURCE: UML 2]

**4.16**
**feature**
abstraction of real world phenomena

Note 1 to entry: A feature can occur as a *class* (4.7) or an *instance* (4.20). The full term feature type or feature instance can be used when only one is meant.

Note 2 to entry: In UML 2 the term feature is used for a property, such as *operation* (4.26) or *attribute* (4.5), which is encapsulated as part of a list within a *classifier* (4.8), such as *interface* (4.21), class or *data type* (4.14).

Note 3 to entry: See Annex D.2.

[SOURCE: ISO 19101-1:2014, 4.1.11 modified — Notes 1-3 have been added.]

**4.17**
**feature**
<UML> property of a *classifier* (4.8)

[SOURCE: UML 2]

**4.18**
**generalization**
<UML> taxonomic *relationship* (4.30) between a more general element and a more specific element of the same element type

Note 1 to entry: An *instance* (4.20) of the more specific element can be used where the more general element is allowed. See: *inheritance* (4.19).

[SOURCE: UML 2]

**4.19**
**inheritance**
mechanism by which more specific *classifiers* (4.8) incorporate structure and behaviour defined by more general classifiers

Note 1 to entry: See *generalization* (4.18).

**4.20**
**instance**
<UML> individual entity having its own value and possibly its own identity

Note 1 to entry: A *classifier* (4.8) specifies the form and behaviour of a set of instances with similar properties.

**4.21**
**interface**
<UML> *classifier* (4.8) that represents a declaration of a set of coherent public <UML> *features* (4.17) and obligations

Note 1 to entry: An interface specifies a contract; any classifier that realizes the interface must fulfil that contract. The obligations that can be associated with an interface are in the form of various kinds of *constraints* (4.13) (such as pre- and post-conditions) or protocol specifications, which can impose ordering restrictions on interactions through the interface.

[SOURCE: UML 2]

**4.22**
**metamodel**
*model* (4.23) that defines the language for expressing other models

Note 1 to entry: A model is an *instance* (4.20) of a metamodel, and a metamodel is an instance of a meta-metamodel.

[SOURCE: UML 2]

**4.23**
**model**
abstraction of some aspects of reality

[SOURCE: ISO 19109:2015, 4.15]

**4.24**
**multiplicity**
<UML> specification of the range of allowable *cardinalities* (4.6) that a set may assume

**4.25**
**object**
entity with a well defined boundary and identity that encapsulates state and behaviour

[SOURCE: UML 1]

**4.26**
**operation**
<UML> behavioural <UML> *feature* (4.17) of a *classifier* (4.8) that specifies the name, *type* (4.36), parameters, and *constraints* (4.13) for invoking an associated behaviour

[SOURCE: UML 2]

**4.27**
**package**
<UML> general purpose mechanism for organizing elements into groups

[SOURCE: UML 2]

**4.28**
**profile**
<UML> definition of a limited extension to a reference *metamodel* (4.22) with the purpose of adapting the metamodel to a specific platform or domain

[SOURCE: UML 2]

**4.29**
**realization**
<UML> specialized abstraction *relationship* (4.30) between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation of the latter (the client)

Note 1 to entry: Realization indicates *inheritance* (4.19) of behaviour without inheritance of structure.

[SOURCE: UML 2]

**4.30**
**relationship**
<UML> semantic connection among model elements

[SOURCE: UML 1]

**4.31**
**schema**
formal description of a *model* (4.23)

[SOURCE: ISO 19101-1:2014, 4.1.34]

**4.32**
**service**
distinct part of the functionality that is provided by an entity through *interfaces* (4.21)

[SOURCE: ISO 19119:2005, 4.1]

**4.33**
**stereotype**
<UML> extension of an existing metaclass that enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass

[SOURCE: UML 2]

**4.34**
**tagged value**
<UML> *attribute* (4.5) on a *stereotype* (4.33) used to extend a model element

[SOURCE: UML 2]

**4.35**
**template**
<UML> parameterized model element

[SOURCE: UML 2]

**4.36**
**type**
<UML> stereotyped *class* (4.7) that specifies a domain of *objects* (4.25) together with the *operations* (4.26) applicable to the objects, without defining the physical implementation of those objects

Note 1 to entry: A type can have *attributes* (4.5) and *associations* (4.4). See relationship to interface in Annex B.

[SOURCE: UML 1]

**4.37**
**value domain**
set of accepted values

EXAMPLE The range 3–28, all integers, any ASCII character, enumeration of values (green, blue, white).

## 5 Presentation and abbreviations

### 5.1 Presentation

The main technical content of this International Standard is found in Clauses 6 and 7. Clause 6 describes the profile of UML for modelling geographic information. Clause 7 defines a set of classes to be used as common elements, as standard UML does not prescribe the use of specific data types.

Annex A describes an abstract test suite for checking that the normative aspects of UML models are made according to the rules of this International Standard. Annex B defines rules for mapping a UML 1 model to a UML 2 model making the model conformant with this International Standard. Annex C describes additional extended data types for this profile. Annex D defines the UML profile formally. An introduction to conceptual schema languages can be found in Annex E. A set of modelling guidelines for information modelling and service modelling is described in Annex F. The general UML as defined in UML 2 is briefly described in Annex G.

### 5.2 Backwards compatibility to previous version of ISO 19103

A list of general descriptions of main additions, removals and changes can be found in Annex H.

The verb “deprecate” provides notice that the referenced portion of this International Standard is being retained for backwards compatibility with earlier versions but may be removed from a future version of this International Standard without further notice.

Text describing deprecated elements is written in *italics*.

### 5.3 Abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| CSL | Conceptual schema language |

| | |
|---|---|
| CSMF | Conceptual Schema Modelling Facility |
| EMOF | Essential Meta Object Facility |
| GFM | General Feature Model |
| IANA | Internet Assigned Numbers Authority |
| MOF | Meta Object Facility |
| OCL | Object Constraint Language |
| OMG | Object Management Group |
| ODP | Open Distributed Processing |
| SRS | Spatial Reference System |
| UML | Unified Modeling Language |
| UML 1 | Unified Modeling Language version 1.4.2 |
| UML 2 | Unified Modeling Language version 2.4.1 |
| uom | Unit of Measure |
| URI | Uniform Resource Identifier |
| url | Uniform Resource Locator |
| XML | eXtensible Markup Language |
| XMI | XML Metamodel Interchange |

## 6 The ISO 19103 UML Profile – Use of UML

### 6.1 Introduction

This International Standard contains a UML Profile for geographic information. Clause 6 provides rules and guidelines on the use of UML for specifying conceptual models within the domain of geographic information. It is based on general UML as defined in UML 2. Annex G contains an introduction to UML and follows the same structure as this clause, to make it easy to refer to the relevant standard UML concepts.

The profile is guidance for a set of limited additions to the UML standard to adapt it for geographic information. It should be noted that profiles are a lightweight extension mechanism to UML and if the need is a precise metamodel of allowed modelling constructs other approaches may be considered. One such approach is EMOF (Essential MOF) which is a precise metamodel for simple class modelling defined in the Meta Object Facility (MOF) Core Specification[7].

### 6.2 General use of UML

| Requirement 1. | The conceptual schema shall be modelled in conformance with UML 2. |
|---|---|

NOTE 1 Books, such as "UML User Guide"[1] and "UML Reference Manual"[2] contain further information. The book "UML Distilled"[4] is a shorter introductory text. More recent books on UML 2 are recommended.

Requirement 2. Conceptual schemas modelled in another CSL shall specify a mapping from the CSL to UML 2. If the conceptual schema is transformed into a UML 2 schema using the mapping, the resulting UML 2 schemas shall conform to the UML2 conformance class.

Non-UML schemas are considered conformant if there is a well-defined mapping from a model in the source language into an equivalent model in UML and that this model in UML is conformant.

| |
|---|
| Requirement 3. All normative class models shall contain definitions sufficient for understanding of all classes, attributes, associations, operations and appropriate data type definitions. |

It is the need for this completeness that makes it necessary to define this UML profile, as UML in general can be used on various levels of precision and completeness. Other kinds of UML diagrams may also be used as normative models.

| |
|---|
| Requirement 4. Each model shall have documented a clear description of its level of abstraction and class stereotypes may be added to help express the chosen abstraction level. |

The main abstraction levels are 'Abstract Schema' and 'Application Schema'. See Figure 4 for an example of types of content in these levels.

NOTE 2 UML 2 can also be used to specify schemas on the levels 'Metamodel' and 'Implementation Schema'.

a) Metamodel level

b) Abstract Schema level

c) Application Schema level

d) Implementation Schema level

## 6.3 Classifiers

Classifiers may serve different purposes in different contexts and models can be at different levels of abstraction. Classifiers defined according to this International Standard may represent abstract specifications where the characteristics of the classifier do not have to be directly implemented or can also represent concrete implementation models. Stereotypes provide a means to define different types of classifiers, which can also define the intended abstraction level of a model.

Stereotypes defined for this UML profile are described in 6.10.2. Specification (abstract) models may be realized in different ways and how this is done should be expressed in each case. See 6.8.4 on realization relationships for a common description of how to realize and use abstract models in implementation models.

The notion of abstract models and classifiers defined in models is that they are abstract, in the sense that the characteristics of the classifier do not have to be directly implementable. An abstract UML model must not be confused with a UML classifier marked as abstract. Marking a UML classifier abstract means that the classifier cannot be instantiated; only specializations of the classifier can occur as instances. Such abstract classifiers may be defined in models at different levels of abstraction.

Classifiers with keyword <<dataType>> (lacking identity) are usually defined in application or implementation schemas. In conceptual schemas they are considered to be abstract unless the schema in which they are defined specifically states otherwise.

## 6.4 Attributes

Attributes have no further requirements other than according to standard UML.

## 6.5 Enumerations and codelists

### 6.5.1 General rules

Requirement 5. An enumerated type declaration defines a list of valid mnemonic identifiers. Attributes of an enumerated type shall take values only from this list.

An enumerated type may be defined as:

— **enum { value1, value2, value3 }**

EXAMPLE 1 attr1: BuildingType, where BuildingType is defined as Enum BuildingType {public, private, tourist}.

Requirement 6. Enumerated types are modelled as classifiers with attributes representing the allowed values. Values of enumerated types shall follow the naming conventions for regular attribute names and should be mnemonic, unless an established name for the concept exists.

Requirement 7. As the values of enumerated types are concepts, each value shall have a definition for the value.

NOTE Values of enumerated types that have overlapping meanings or with some internal structure can be further described in separate tables outside of the UML model.

*Recommendation 1. It is recommended not to use meaningless initial attribute values (in the UML model) to represent alternative enumeration or code list values (codes) for the mnemonic identifiers.*

*Recommendation 2. For additional user understanding of the concepts it is recommended to have common language labels and definitions in one or several languages inside the model using tagged values or outside of the model, and create a mapping from these to each value in the model.*

EXAMPLE 2 For the values in Figure 2 there may be some mapping from the value crossTrack to English common language “cross track” and to Norwegian language “krysspor”.

There are two main types of enumerated types, enumerations and code lists. An enumeration is fixed at model time and cannot be extended, while a code list can be extended during the lifetime of the model.

### 6.5.2 Enumerations

Enumerations are modelled as described above with the addition of a keyword <<enumeration>> .

EXAMPLE Figure 1 shows an enumeration of the colours of the rainbow.

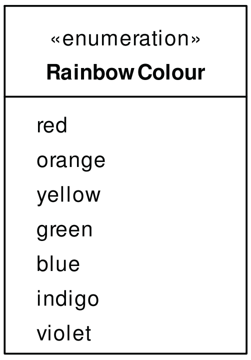


**Figure 1 — Example of enumeration**

The extension of an enumeration type will imply a schema modification and enumerations are used only for a set of values that is complete and no extensions to the set are possible, e.g. the seven colours of the rainbow. Enumerated lists that are stable in a particular version of a schema may also be modelled as enumerations.

| |
|---|
| Requirement 8. If extensions to an enumerated list are expected during the lifetime of the schema version, a code list shall be used. |

### 6.5.3 Code lists

A code list can be used to describe an open enumeration.

| |
|---|
| Requirement 9. Code lists shall be used if none or only a few of the allowed values are known, like a likely set, or an initial set. |

This means that it may need to be represented and implemented in such a way that it can be extended during system runtime. Code lists are modelled as classes with a <<CodeList>> stereotype. An example code list is shown in Figure 2.

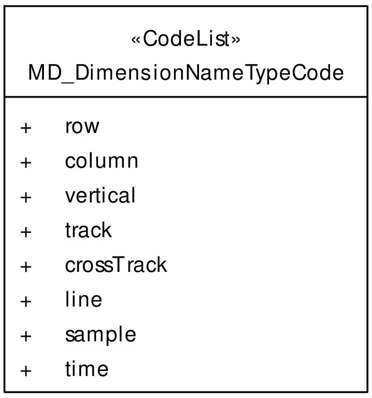


**Figure 2 — Example of code list**

*Recommendation 3. Extensions of a code list use the existing code list values and merely add additional unique values. These additional values should not replace an existing code by changing the name or definition, or have the same definition as an existing value.*

Code lists can furthermore be divided into several types, for example those that are managed by a single authority and those that can be extended by a user of the schema.

Code lists managed by a single external authority may carry a tagged value "codeList" whose value references the actual external code list. If the tagged value is set, only values from the referenced code list are valid.

*Recommendation 4. The value of the tagged value "codeList" should be a persistent URI identifying the code list.*

NOTE 1 The rules for governance of code list values belong to implementations. If no tagged value "codeList" is present, the code list can be extended freely.

NOTE 2 In the model this kind of externally managed code list can initially be without values, as the values can be added after the model is completed.

NOTE 3 To allow recipients of data to interpret code list values, it is helpful if the implementations of attributes with code list values also include a reference to the code list.

### 6.6 Data types

Clause 7 defines the set of core data types that are used in this UML profile. Other data types are also allowed within the context of the UML standards.

### 6.7 Operations

Operations have no further requirements other than according to standard UML.

### 6.8 Relationships

#### 6.8.1 General

Relationships have some additional requirements specified in 6.8.

#### 6.8.2 Associations

Requirement 10. All associations shall have multiplicity defined at each navigable end.

Requirement 11. Each navigable end shall have a role name.

*Recommendation 5. The role names of association ends should reflect the role that the classifier at the association end plays with respect to the classifier at the other end. It should not express the whole association.*

EXAMPLE 1 Facility ----- > Site

The role name at the Site end of the association should be site or hostingSite or location rather than locatedAt or hosts.

EXAMPLE 2 Facility ----- > Installation

The role name at the Installation end of the association should be installation or containedInstallation or part rather than contains or hasPart.

Association names are principally for documentation purposes.

*Recommendation 6. Associations involving more than two classifiers should be avoided in order to reduce complexity and avoid model mapping problems.*

Association classes are associations where the association itself has attributes.

Requirement 12. Data types, which lack identity, shall be used only in attributes and strong aggregations (composition).

Data types (classifiers using keyword <<dataType>> ) have no identity and cannot be referenced. Therefore, a classifier representing a data type cannot take part in an association.

Requirement 13. A data type shall only have an outward pointing association if one of the following is true:

1) The target of the association is another data type and the association is a composition
2) The target of the association is a well-known immutable object for which a universally recognized identifier exists.

In case 1, the structure is equivalent to a member attribute of the data type. In case 2, the structure is equivalent to the data type having a member attribute whose value is the identity of the target.

EXAMPLE 3 In ISO 19107 the data type DirectPosition has a relation to SC_CRS which has a universally recognized identifier represented by inherited attribute identifier of type RS_Identifier. This is equivalent to DirectPosition having an attribute of type RS_Identifier from ISO 19115-1 to hold the coordinate system identity. Technically this is a violation of UML rules, but essentially it does not violate the intent. Figure 3 shows the two alternatives.

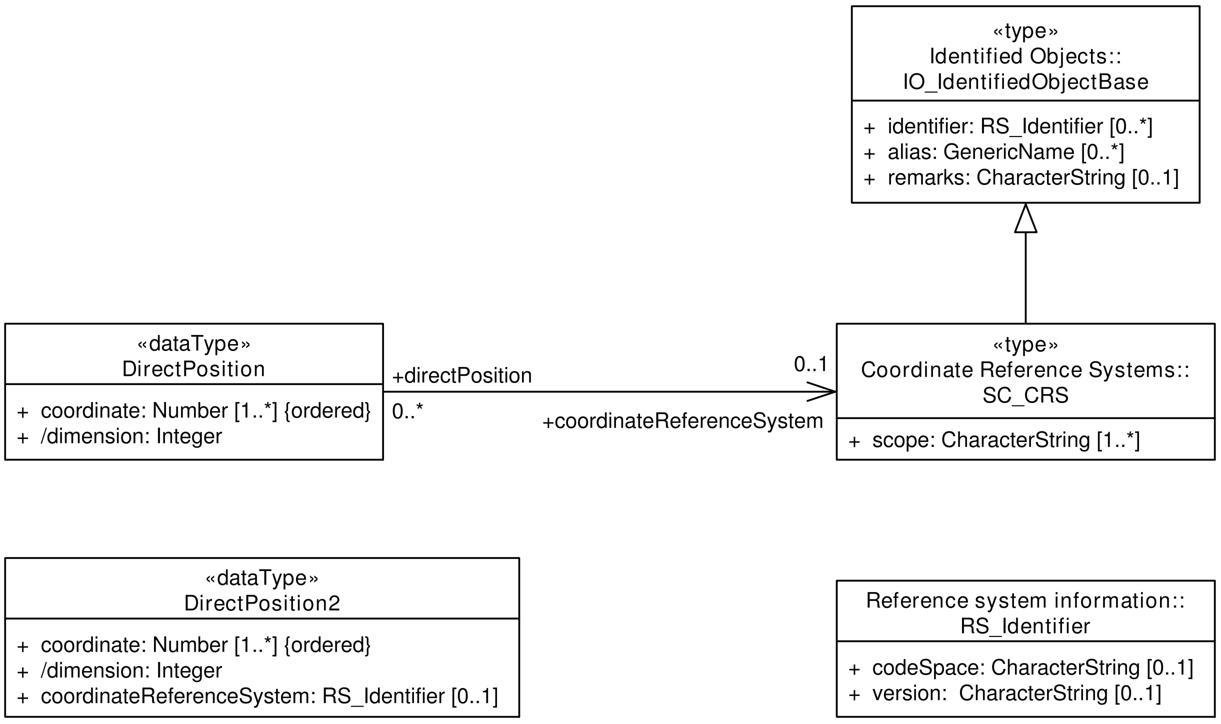

«type»
Identified Objects::
IO_IdentifiedObjectBase
+ identifier: RS_Identifier [0..*]
+ alias: GenericName [0..*]
+ remarks: CharacterString [0..1]
«dataType»
DirectPosition
+ coordinate: Number [1..*] {ordered}
+ /dimension: Integer
+directPosition
0..*
0..1
+coordinateReferenceSystem
«type»
Coordinate Reference Systems::
SC_CRS
+ scope: CharacterString [1..*]
«dataType»
DirectPosition2
+ coordinate: Number [1..*] {ordered}
+ /dimension: Integer
+ coordinateReferenceSystem: RS_Identifier [0..1]
Reference system information::
RS_Identifier
+ codeSpace: CharacterString [0..1]
+ version: CharacterString [0..1]

**Figure 3 — DirectPosition example: Use of Associations by datatypes**

In general, if a data type has an association role named “referenceToB” pointing to a type “B”, then it should be replaceable by an attribute “referenceToB” of type CharacterString (or similar type) that will contain an identity of a logically immutable instance of type “B”.

### 6.8.3 Generalizations

Use of multiple inheritance is an issue in many implementation environments and can cause problems if handled improperly in those environments. For this reason:

*Recommendation 7. The use of multiple inheritance should be minimized or avoided unless it is a fundamental part of the semantics of the type hierarchy.*

| Requirement 14. A generalization relationship is only allowed between classifiers at the same abstraction level. |
|---|

For example, if you subtype a class with keyword <<interface>> the specialized class will also be an interface.

### 6.8.4 Realizations

Realizations can be used for a variety of purposes. One is to relate model elements at different levels of abstraction. When used between classifiers, a realization can be thought of as inheritance of behaviour without inheritance of structure as described above. Consequently it is a convenient mechanism for describing shifts in abstraction levels where a concrete element realizes a more abstract element.

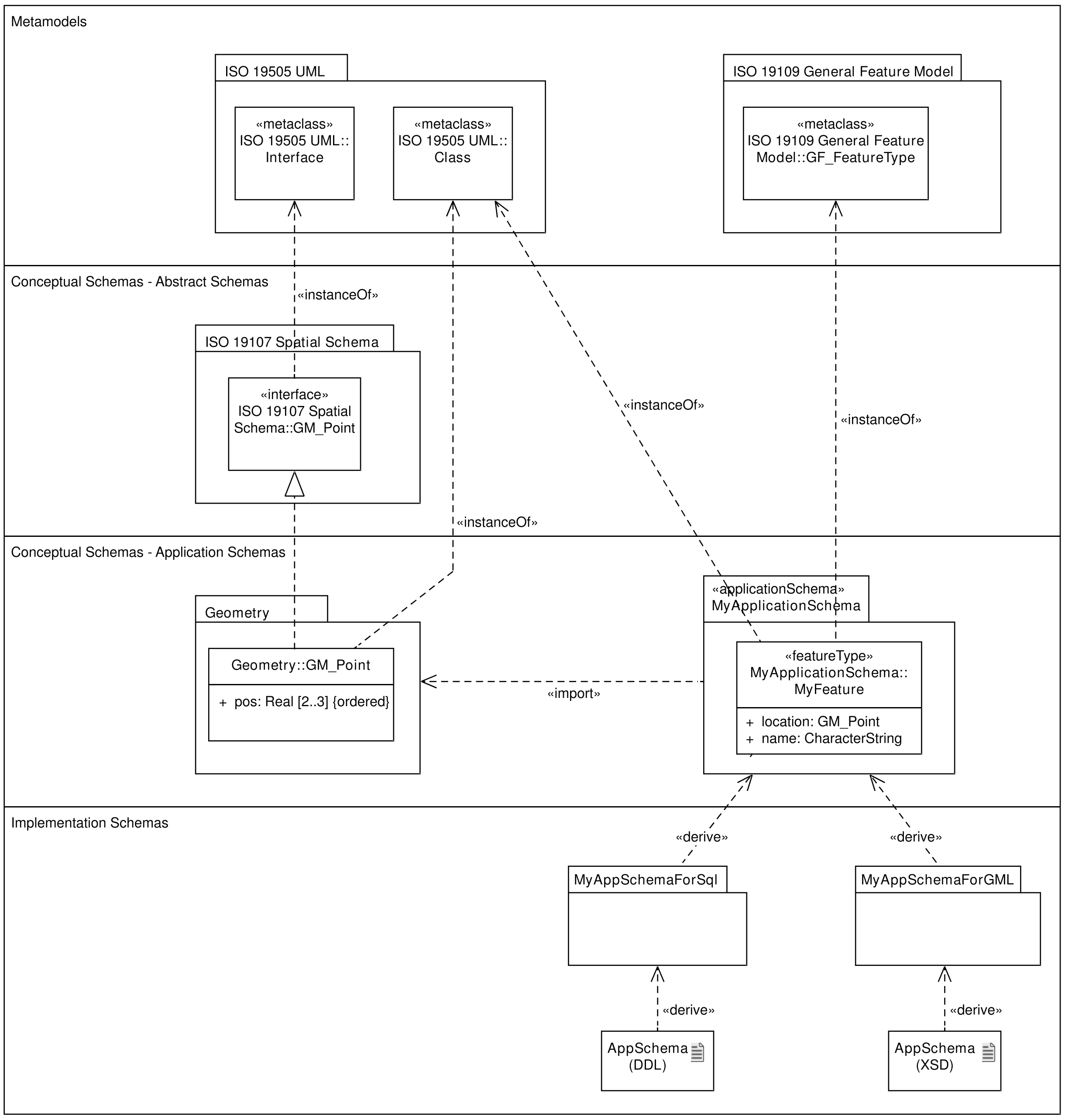

Metamodels
ISO 19505 UML
«metaclass» ISO 19505 UML:: Interface
«metaclass» ISO 19505 UML:: Class
ISO 19109 General Feature Model
«metaclass» ISO 19109 General Feature Model::GF_FeatureType
Conceptual Schemas - Abstract Schemas
«instanceOf»
ISO 19107 Spatial Schema
«interface» ISO 19107 Spatial Schema::GM_Point
«instanceOf»
«instanceOf»
«instanceOf»
Conceptual Schemas - Application Schemas
Geometry
Geometry::GM_Point
+ pos: Real [2..3] {ordered}
«import»
«applicationSchema» MyApplicationSchema
«featureType» MyApplicationSchema:: MyFeature
+ location: GM_Point
+ name: CharacterString
Implementation Schemas
«derive»
«derive»
MyAppSchemaForSql
MyAppSchemaForGML
«derive»
«derive»
AppSchema (DDL)
AppSchema (XSD)

**Figure 4 — Example of realization used between different levels of abstraction**

The use of realization is shown in Figure 4 where the interface GM_Point in package ISO 19107 is realized by a concrete class of the same name in a different package at the Application Schema level. Both classes reference the same information content, but it is the concrete one that may be implementable. This concept also applies to other abstract types defined in this International Standard.

## 6.9 Services

Behavioural aspects have no further requirements other than as in standard UML. Service architecture including approaches for identifying services is further described in ISO 19119.

## 6.10 Stereotypes and keywords

### 6.10.1 Introduction

Some additional basic stereotypes along with predefined stereotypes and special keywords of UML used in this International Standard are defined in 6.10.

| |
|---|
| Requirement 15. Conceptual models in the domain of geographic information shall use these stereotypes and keywords when appropriate, and shall not create alternatives with the exact same meaning. |

### 6.10.2 Stereotypes and keywords

Stereotypes make it possible to extend the UML and give classifiers different meaning. This International Standard specifies the following stereotypes and keywords for the UML profile for geographic information:

a) <<CodeList>> is a flexible enumeration that uses string values for expressing a list of potential values.

b) <<dataType>> is a set of properties that lack identity (independent existence and the possibility of side effects). A data type is a classifier with no operations, whose primary purpose is to hold information.

c) <<enumeration>> is a fixed list of valid identifiers of named literal values. Attributes of an enumerated type may only take values from this list.

d) <<interface>> is an abstract classifier with operations, attributes and associations, which can only inherit from or be inherited by other interfaces. Other classifiers may realize an interface by implementing its operations and supporting its attributes and associations (at least through derivation). See Annex B and mapping from UML 1 to UML 2.

e) <<Leaf>> is a package that contains definitions, without any sub packages.

f) <<Union>> is a type consisting of one and only one of several alternative datatypes (listed as member attributes). This is similar to a discriminated union in many programming languages. In some languages using pointers, this requires a “void” pointer that can be cast to the appropriate type as determined by a discriminator attribute.

Stereotypes are essential in the creation of code generators for UML models. Generally speaking, stereotypes act as flags to language compilers to determine how to create implementation models from the abstract models. Other stereotypes can be added in addition to the ones listed.

NOTE The names of stereotypes and keywords are not case-sensitive.

## 6.11 Optional, conditional and mandatory attributes and association ends

### 6.11.1 Mandatory

In UML, all attributes are by default mandatory. The possibility to show multiplicity for attributes and association role names provides a way of describing optional and conditional attributes.

An attribute that has a multiplicity with a lower bound of 1 is mandatory.

NOTE 1 The default multiplicity for attributes is 1.

An association whose target end is navigable and has a multiplicity with a lower bound of 1 at the target end is mandatory for the classifier at the source end.

NOTE 2 There is no default multiplicity for association ends. See Requirement 10.

#### 6.11.2 Optional

A property with a minimum multiplicity of 0 is optional.

#### 6.11.3 Conditional

An attribute or association may be specified as conditional, meaning that whether it is mandatory or optional depends on other model elements. It may depend simply on the presence of a value for another (optional) element or it may depend on the values of other elements. A conditional attribute or association is shown as optional with an OCL constraint attached.

### 6.12 Naming and namespaces

Naming conventions are used for a variety of reasons, mainly readability, consistency and as a protection against case sensitive binding. Rules and guidelines for naming are described below. It is highly recommended to adhere to the stated naming conventions, especially for implementation models.

| Requirement 16. The names of UML elements shall be case-insensitive unique within their namespace and not contain white space. |
|---|

*Recommendation 8. Names of UML elements should use precise and understandable technical names for classifiers, attributes, operations and parameters.*

EXAMPLE 1 Use index as opposed to n.

*Recommendation 9. Names of UML elements should use short parameter names when the parameter container or type carries meaning.*

EXAMPLE 2 use equals(other:GM_Object) as opposed to equals(otherGeometryObject:GM_Object)

NOTE Some older models may still use prefixes.

*Recommendation 10. Names of UML elements should combine multiple words as needed to form precise and understandable names without using any intervening characters (such as "_", "-"or space).*

EXAMPLE 3 computePartialDerivatives (not: compute Partial Derivatives or compute_Partial_Derivatives).

*Recommendation 11. For UML attributes and operation names, association roles and parameters, capitalize only the first letter of each word after the first word that is combined in a name. Capitalize the first letter of the first word for each name of a classifier, package, type specification and association names.*

EXAMPLE 4 (operation): computePartialDerivatives (not computepartialderivatives or COMPUTEPARTIALDERIVATIVES)

EXAMPLE 5 (class): CoordinateTransformation (not coordinateTransformation).

*Recommendation 12. UML elements should use documentation fields to further explain the meanings (semantics) of named elements.*

*Recommendation 13. UML elements should keep names as short as practical. Use standard abbreviations if understandable, skip prepositions and drop verbs when they do not significantly add to the meaning of the name.*

EXAMPLE 6 (operation) equals() instead of IsEqual(),

EXAMPLE 7 (operation) value() instead of getValue(),

EXAMPLE 8 (operation) initObject() instead of initializeObject()

EXAMPLE 9 (operation) length() instead of computeLength().

In most cases the namespace for an attribute, role or operation is the classifier in which the operation is defined, but it can also include the package name in which a classifier is defined. Where namespaces are classifier identifiers, they can take only one of two forms.

name: == classifier-name | package-name::classifier-name

The "::" is a resolution operator indicating the namespace of that which follows.

Unless there is a potential for confusion or a need for emphasis, the package name is not included.

## 6.13 Packages

Packages are a convenient mechanism for structuring models. A package is a general purpose namespace that can own any kind of model element. Packages may contain other packages. Packages may have dependency relationships to other packages, a dependency denotes that model elements in one package depend on model elements in another.

| Requirement 17. All package dependencies shall be shown in one or more package diagrams for each package. |
|---|

An example package structure of dependencies between the ISO geographic information standards is shown in Figure 5.

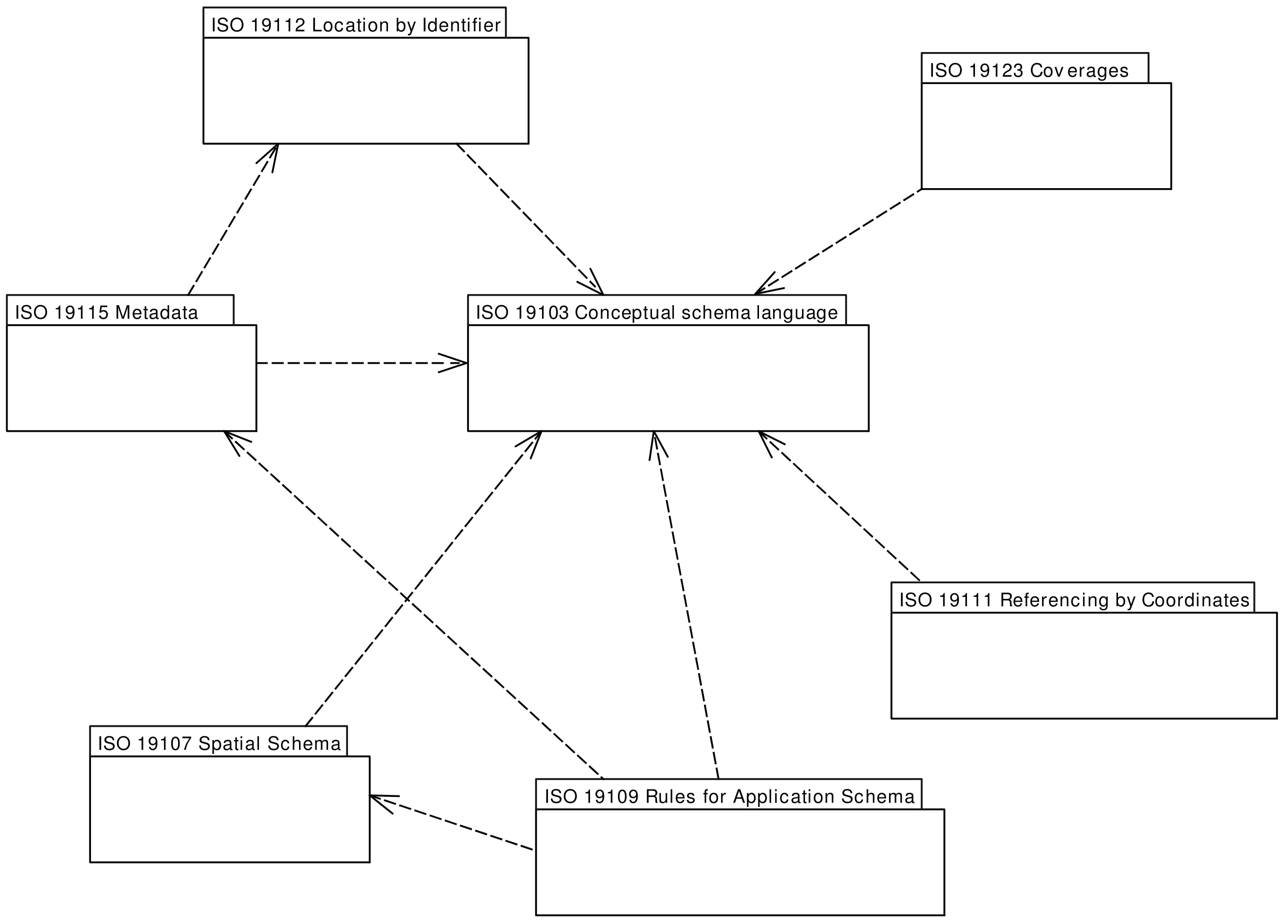

ISO 19112 Location by Identifier
ISO 19123 Cov erages
ISO 19115 Metadata
ISO 19103 Conceptual schema language
ISO 19111 Referencing by Coordinates
ISO 19107 Spatial Schema
ISO 19109 Rules for Application Schema

**Figure 5 — Example package structure**

## 6.14 Notes

Notes have no further requirements other than according to standard UML.

## 6.15 Constraints

Constraints are important as they add precision to the models. Expressing constraints at the conceptual level is important for validation, management and maintenance of data. Constraints may be described in a note, or in the classifier itself.

*Recommendation 14. Constraints should be expressed using the Object Constraint Language (OCL) as defined in OCL 2.0. In addition, constraints should be expressed in natural language in the accompanying documentation of the model.*

OCL defines a set of basic types similar to this profile. To fully use OCL in models defined by this profile a mapping between the basic types is necessary. Table 7 describes the mapping.

**Table 7 — Mapping between ISO 19103 basic types and OCL basic types**

| ISO 19103 | **OCL** |
|---|---|
| Integer | Integer |
| Real | Real |
| Boolean | Boolean |
| CharacterString | String |

## 6.16 Documentation of models

In addition to the presentation of the model in diagrams, it is necessary to document the semantics of the model.

*Recommendation 15. For the diagrams, readability concerns should be taken into account. Font size should be no smaller than 8 points after diagram scale is taken into account (a 12 pt font at 90 % is 12*0.9 = 10.8 pt).*

*Recommendation 16. For each package it is recommended to have a class diagram for the entire package (without any associations, operations or attributes shown).*

Requirement 18. All classifiers shall be documented in a "context diagram" where all attributes, operations and all relationships that are navigable from the central classifier are displayed.

Note that closely related classifiers can share a context diagram as long as the combined diagram is neither overly cluttered nor difficult to read.

Requirement 19. Each classifier shall have a definition describing its intended meaning or semantics.

Requirement 20. Each package, classifier, operation, attribute, association role, association and constraint shall have a textual description in the text near its context diagram.

Requirement 21. Package dependency diagrams shall be included, including external package dependencies.

Overview diagrams consisting of all major classes in a document may be included if found feasible. Overview diagrams can focus on the most important parts of the model, and package diagrams can also give a full and complete picture of the whole package content.

The textual description of a model may be presented in different ways. One approach is data dictionaries where all model elements are described in tables while another is to describe each model element in clauses and sub-clauses. This International Standard makes no recommendation for this documenting of models beyond Requirements 18 to 21.

## 7 Core data types

### 7.1 Introduction

The UML itself is not specific on the usage of data types, although the UML 2 Core package defines primitive data types. Clause 7 specifies a set of core data types for this International Standard.

| Requirement 22. Platform neutral models of geographic information shall use these core data types when appropriate, and shall not create alternative types with same meaning. |
|---|

Annex C describes additional data types that are extension to the core types. The data types defined in Clause 7 are those that are usually defined by the development environment's data definition language. Each of these types can be represented in a variety of logically equivalent forms. The ones presented here are not meant to restrict the usage of other equivalent forms native to the chosen development environment. ISO/IEC 11404 presents an equivalent definition for most of the types presented and is referenced where appropriate.

The types have been grouped into six categories:

a) Primitive types: Fundamental types for representing values, examples are CharacterString, Integer, Real, Boolean, Date and DateTime,

b) Collections,

c) Enumerated types,

d) Name types: types for representing name structures,

e) Any type,

f) Record types: types for representing structure.

The types are defined as abstract types and appropriate representations will be defined by implementation and encoding mappings. Encoding is further discussed in ISO 19118. The repertoire of data types is described in 7.2 to 7.8. An overview of the packages representing core data types is shown in Figure 6.

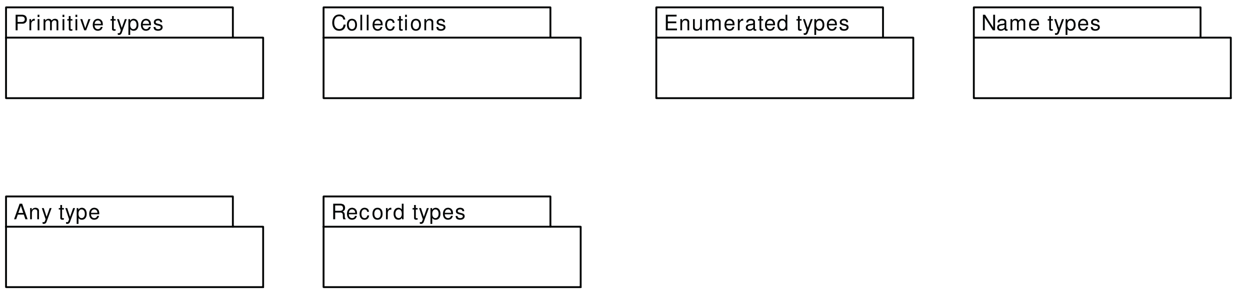


**Figure 6 — Overview of packages representing data types**

### 7.2 Primitive types

#### 7.2.1 General

Figure 7 provides an overview of the primitive types. Each type is described in 7.2.2 to 7.2.11.

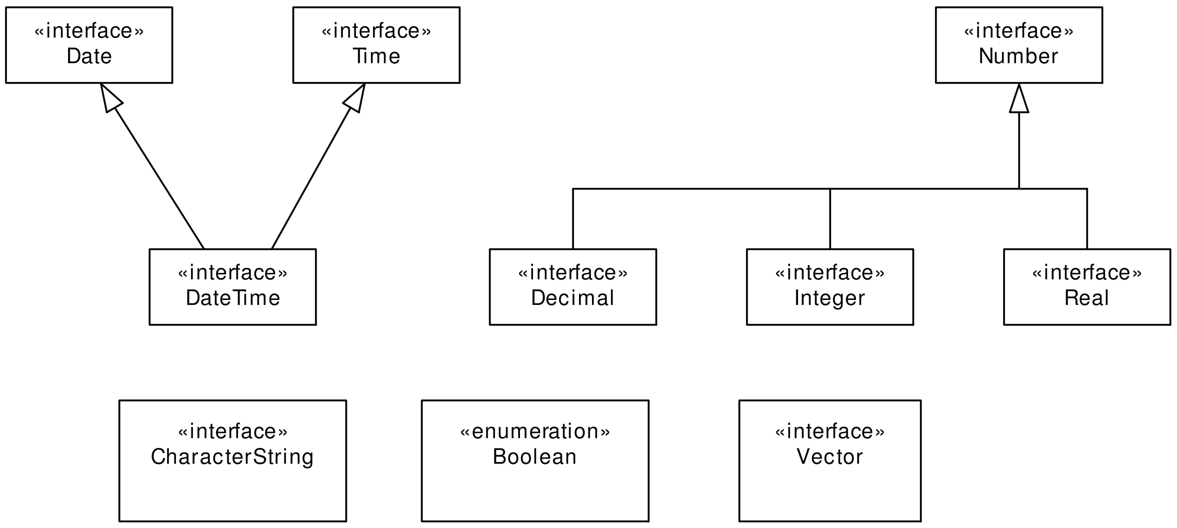


**Figure 7 — Primitive types**

### 7.2.2 Date

Date gives values for year, month and day as shown in Figure 8. Representation of Date is specified in ISO 8601. Principles for date and time are further discussed in ISO 19108.

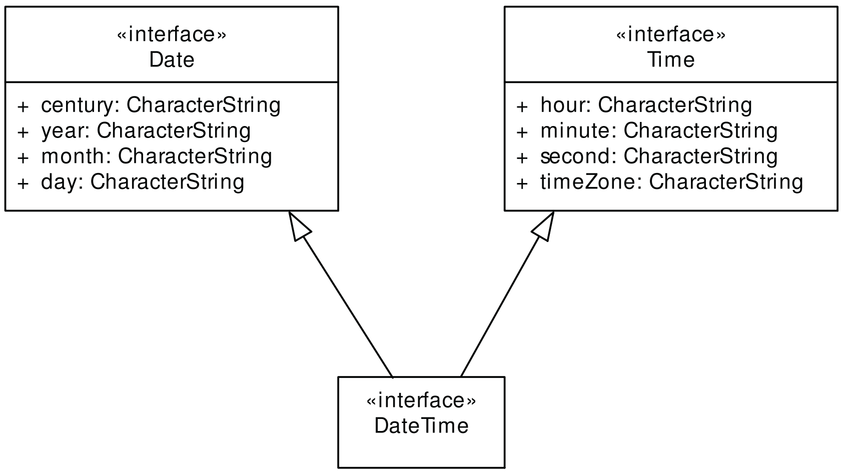


**Figure 8 — Date and Time types**

EXAMPLE 1998–09–18

### 7.2.3 Time

A time is given by an hour, minute and second as shown Figure 8. Representation of Time is specified in ISO 8601. Principles for date and time are further discussed in ISO 19108. ISO/IEC 11404 specifies other methods of representing time.

EXAMPLE 18:30:59 or 18:30:59+01:00

### 7.2.4 DateTime

A DateTime is a combination of a date and a time type as shown in Figure 8. Representation of DateTime is specified in ISO 8601. Principles for date and time are further discussed in ISO 19108. ISO/IEC 11404 specifies other methods of representing time.

### 7.2.5 Number

#### 7.2.5.1 Semantics

Number is the base type for all number data, giving the basic algebraic operations. Since all concrete types have finite representations, some parts of this algebra for most types exhibit some inaccuracy. For example, integers cannot always be divided accurately, and real and decimal numbers experience other types of inaccuracy that depend on their representation. The model for Number and other numeric types is shown in Figure 9.

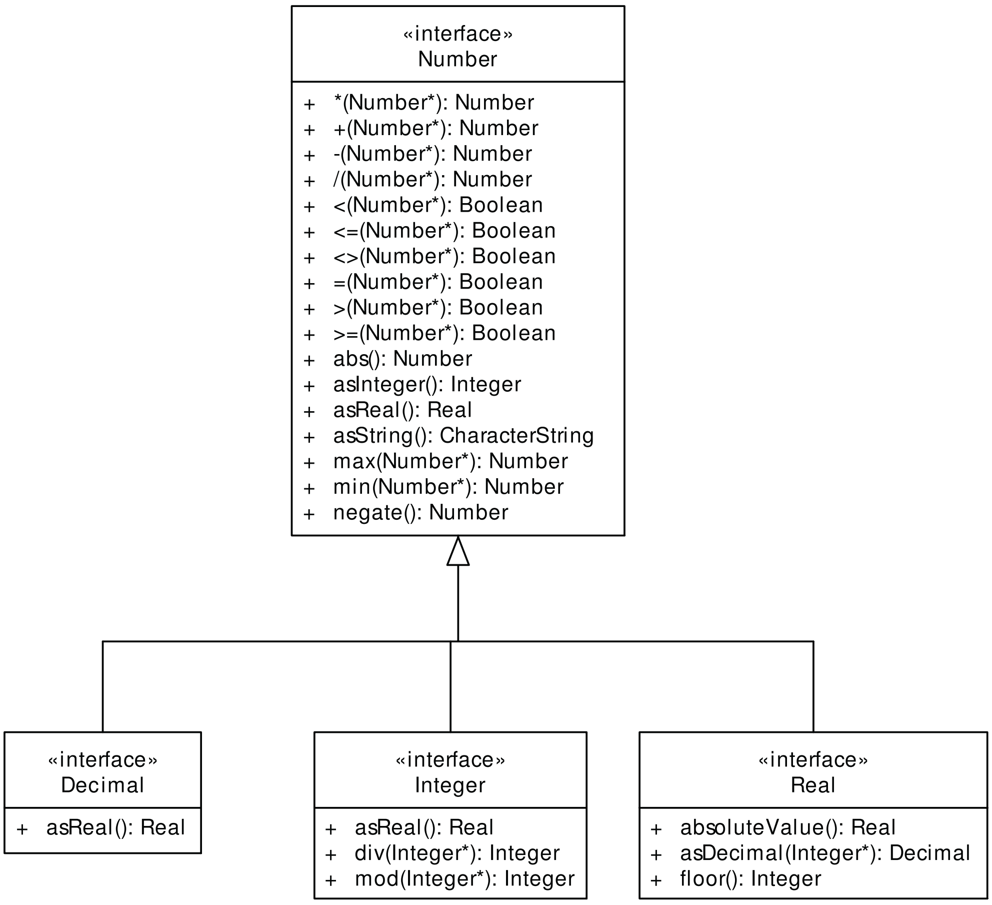


**Figure 9 — Number and subtypes**

#### 7.2.5.2 Operation *

Scale a number by another number.

#### 7.2.5.3 Operation +

Combine a number with another number.

#### 7.2.5.4 Operation -

Take away a number from another number.

#### 7.2.5.5 Operation /

Partition a number with another number.

#### 7.2.5.6 Operation <

A number is smaller than another number.

#### 7.2.5.7 Operation <=

A number is smaller than or equal to another number.

#### 7.2.5.8 Operation <>

A number is different from another number.

#### 7.2.5.9 Operation =

A number is equal to another number.

#### 7.2.5.10 Operation >

A number is larger than another number.

#### 7.2.5.11 Operation >=

A number is larger than or equal to another number.

#### 7.2.5.12 Operation abs

Absolute value of a number.

#### 7.2.5.13 Operation asInteger

Integer part of a number.

#### 7.2.5.14 Operation asReal

Real representation of a number.

#### 7.2.5.15 Operation asString

String representation of a number.

#### 7.2.5.16 Operation max

Largest value of the two numbers.

#### 7.2.5.17 Operation min

Smallest value of two numbers.

#### 7.2.5.18 Operation negate

Reverse the sign of a number.

### 7.2.6 Decimal

#### 7.2.6.1 Semantics

Decimal (see Figure 9) is a decimal data type in which the number represents an exact value, as a finite representation of a decimal number. It differs from the common binary real implementation in that it can represent 1/10 (one-tenth) without error, while binary real representation can only represent powers of exactly 1/2 (one-half). Since many currencies are decimal, these representations are preferred in dealing with such values. This is also true for mile markers, which are often given in decimals.

NOTE This differs from real, as real is an approximate value and Decimal is exact.

EXAMPLE 12.75.

#### 7.2.6.2 Operation asReal

Real representation of a decimal.

### 7.2.7 Integer

#### 7.2.7.1 Semantics

Integer (see Figure 9) is the mathematical data type comprising the exact integral values. See ISO/IEC 11404:2007, 8.1.7 for further description.

EXAMPLE 29, – 65547.

#### 7.2.7.2 Operation asReal

Real representation of a integer.

#### 7.2.7.3 Operation div

Integer quotient of division of one integer by another.

#### 7.2.7.4 Operation mod

Remainder of division of one integer by another.

### 7.2.8 Real

#### 7.2.8.1 Semantics

Real (see Figure 9) is a signed real (floating point) number consisting of a mantissa and an exponent, which represents a value to a precision given by the number of digits shown, but is not necessarily the exact value. The length of a real is encapsulation and usage dependent. The common binary real implementation uses base 2. Since such real numbers can approximate any measure where absolute accuracy is not possible, this form of numeric is most often used for measures. In cases were absolute accuracy is needed, such as currencies, then a decimal representation may be preferred (assuming the currency is decimal, such as the US dollar or British pound). Where there are no subunits possible, integer numbers may be preferred. A real can be considered as an integer part followed by a fractional part given in multiples of powers of 1/2 (halves). See ISO/IEC 11404:2007, 8.1.10 for further description.

EXAMPLE 23.501, – 1234E-4, – 23.0.

#### 7.2.8.2 Operation absoluteValue

Positive real value having the same magnitude as a real.

### 7.2.9 Vector

#### 7.2.9.1 Semantics

A vector is a quantity that has both magnitude and direction. It often can be represented as an ordered set of numbers called coordinates that represent a position in a coordinate system. Vector is shown in Figure 10.

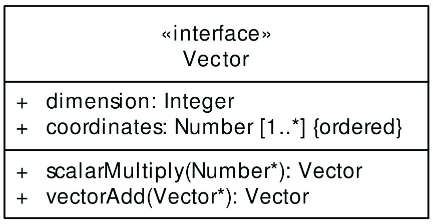

«interface»
Vector
+ dimension: Integer
+ coordinates: Number [1..*] {ordered}
+ scalarMultiply(Number*): Vector
+ vectorAdd(Vector*): Vector

**Figure 10 — Vector**

EXAMPLE (123, 514, 150)

#### 7.2.9.2 dimension

The number of dimensions in the space in which the vector is defined.

#### 7.2.9.3 coordinates

Sequence of numbers representing the vector.

#### 7.2.9.4 Operation scalarMultiply

Scale a vector by another vector.

#### 7.2.9.5 Operation vectorAdd

Combine a vector with another vector.

### 7.2.10 CharacterString

#### 7.2.10.1 Semantics

CharacterString is a family of data types which represent strings of symbols from standard character-sets. Semantics of CharacterString is in accordance with ISO/IEC 11404:2007, 10.1.5. Information about the language for interpreting the strings may if necessary be stated at an appropriate level. CharacterString is shown in Figure 11.

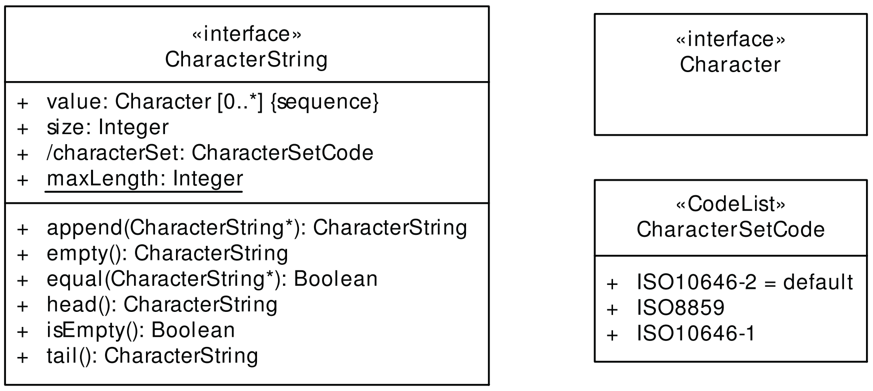

«interface»
CharacterString
+ value: Character [0..*] {sequence}
+ size: Integer
+ /characterSet: CharacterSetCode
+ maxLength: Integer
+ append(CharacterString*): CharacterString
+ empty(): CharacterString
+ equal(CharacterString*): Boolean
+ head(): CharacterString
+ isEmpty(): Boolean
+ tail(): CharacterString
«interface»
Character
«CodeList»
CharacterSetCode
+ ISO10646-2 = default
+ ISO8859
+ ISO10646-1

**Figure 11 — CharacterString**

EXAMPLE “Ærlige Kåre så snø for første gang.” (In Norwegian language.)

#### 7.2.10.2 size

Number of characters in the string.

#### 7.2.10.3 /characterSet

Character representation and encoding system of the string.

#### 7.2.10.4 maxLength

Maximum length of all instances of any string.

#### 7.2.10.5 Operation append

Add a string to the end of the string.

#### 7.2.10.6 Operation empty

The string containing no values.

#### 7.2.10.7 Operation equal

True if string is equal to another string.

#### 7.2.10.8 Operation head

First character in the string.

#### 7.2.10.9 Operation isEmpty

True if the string is an empty string

#### 7.2.10.10 Operation tail

String formed by deleting the first character from a string.

### 7.2.11 Boolean

#### 7.2.11.1 Semantics

Boolean is the mathematical data type associated with two-valued logic as defined in ISO/IEC 11404:2007, 8.1.1. Boolean specifies the values True or False. Boolean is shown in Figure 12.

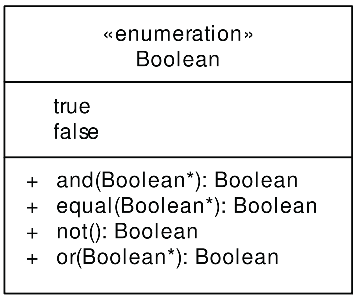


**Figure 12 — Boolean**

#### 7.2.11.2 true

Logically true.

#### 7.2.11.3 false

Logically false.

#### 7.2.11.4 Operation and

Logically true if both are true.

#### 7.2.11.5 Operation equal

Logically true if both are true or both are false.

#### 7.2.11.6 Operation not

Opposite logical value.

#### 7.2.11.7 Operation or

Logically true if any one are true.

## 7.3 Collections

### 7.3.1 General

UML 2 provides property keywords to define collection semantics for attributes. These property keywords actually belong to the multiplicity specifications of attributes and association ends. If the upper bound of the multiplicity is larger than one, then ordering and uniqueness indicators can be defined in brackets. Ordering can be ordered or unordered (default is unordered), while uniqueness can be unique or nonunique (default is unique). Keywords are defined for the combination of properties and are shown in Table 8.

**Table 8 — Keywords for combinations of ordering and uniqueness**

| Keyword | Combination |
|---|---|
| set | unordered, unique elements (default) |
| bag | unordered, nonunique elements |
| orderedSet | ordered, unique elements |
| list (or sequence) | ordered, nonunique elements |

EXAMPLE 1 position : DirectPosition [1..*] {set} instead of position : Set<DirectPosition>

EXAMPLE 2 coordinate : Number [0..*] {sequence} instead of coordinate : Sequence<Number>

This mechanism can replace the collection templates defined in the previous version of this International Standard. The collection templates are retained for direct backward compatibility and described in 7.3.2 to 7.3.5.

## 7.3.2 Collection templates

A collection type is a template type that contains multiple occurrences of instances of a specific type. There are different types of collections: set, bag and sequence. They have different semantics with regard to the ordering of the elements and the possible operations allowed on the collection. A collection type usually does not have an upper bound, i.e. a limit on the number of elements. The collection type is a template type because it takes a type as an argument. The type of the element in the collection is provided as an argument. The maximum number of elements may be specified as a constraint.

## 7.3.3 Set

A Set is a finite collection of objects, where each object appears in the collection only once. A set cannot contain any duplicated instances. The order of the elements of the set is not specified. The model for Set is shown in Figure 13. Formal definition of set is found in ISO/IEC 11404:2007, 8.4.2.

The generic type to be instantiated is Set<T> where T is the data type of the legal elements.

EXAMPLE Set<GM_Point> is instantiated from the generic Set<T> where T is the data type of the legal elements.

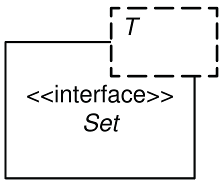


**Figure 13 — Collection types — Set type**

## 7.3.4 Bag

A bag may contain duplicate instances. As with a Set, there is no specified ordering among the elements of a bag. Bags are most often implemented through the use of proxies or reference pointers. The model for Bag is shown in Figure 14. Formal definition of bag is found in ISO/IEC 11404:2007, 8.4.3.

The generic type to be instantiated is Bag<T> where T is the data type of the elements of the bag.

EXAMPLE Bag<Integer> is instantiated from the generic Bag<T> where T is the data type of the legal elements.

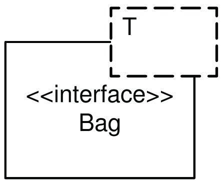


**Figure 14 — Collection types — Bag type**

### 7.3.5 Sequence

A Sequence is a Bag-like structure that orders the element instances. This means that an element may be repeated in a sequence. Sequences may be used as either lists or arrays. Arrays used in programming languages are sequences that can be directly indexed by an offset from the beginning of the sequence. Although lists are not semantically equivalent to arrays, the details of differences are in the implementation and do not affect the polymorphic interface described here.

A sequence is a collection that specifies a sequential ordering between its elements. A synonym to sequence is List. The model for Sequence is shown in Figure 15. Formal definition of sequence is found in ISO/IEC 11404:2007, 8.4.4.

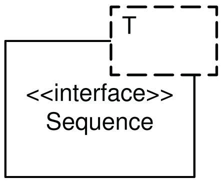


**Figure 15 — Collection type — Sequence type**

The generic type to be instantiated is Sequence<T> where T is the data type of the elements of the sequence.

EXAMPLE Sequence<String> is instantiated from the generic Sequence<T> where T is the data type of the elements.

Sequences can be used directly as a template type of a UML attribute, as in Sequence <T> , where T can be any type, or as type definitions.

## 7.4 Enumerated types

### 7.4.1 General

The defined enumerated types are shown in Figure 16 and described below. These types were defined in the previous version of this International Standard and are kept for direct backward compatibility.

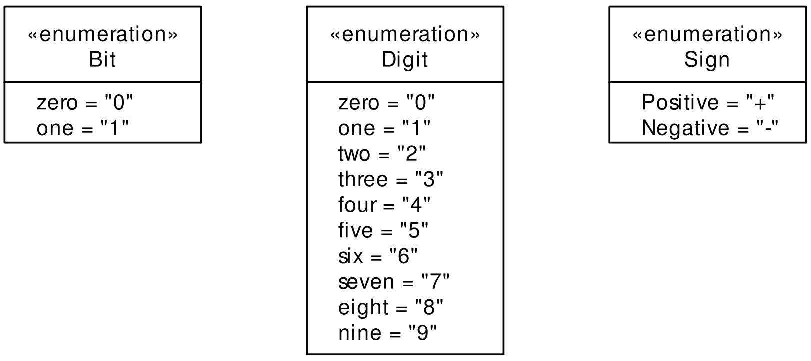


**Figure 16 — Enumerated types**

### 7.4.2 Bit

The simplest element of a binary stream. When represented as a character string, the usual notation is “0” or “1”, but other notations are possible, such as “.” and “X” which gives a better “density” view of a string. There is usually no semantic difference between these representations. Bit is shown in Figure 16.

### 7.4.3 Digit

The digits of a base 10 numbering scheme. The normal “Arabic” numerals are the common representation. Digit is shown in Figure 16.

### 7.4.4 Sign

A enumeration of sign, usually used in an algebraic system to distinguish between a positive value and a negative value, or between a base orientation or a reversal of a base orientation. These are commonly represented by a single character such as “+” or “-” but may sometimes carry an integer 1 for emphasis such as “+1”, or “-1” – There is no semantic difference between these two presentation objects. Sign is shown in Figure 16.

## 7.5 Name types

### 7.5.1 General

GenericName and subclasses of this class are used to create a generic scoped and local name structure for names in the context of namespaces. The model for name types is shown in Figure 17.

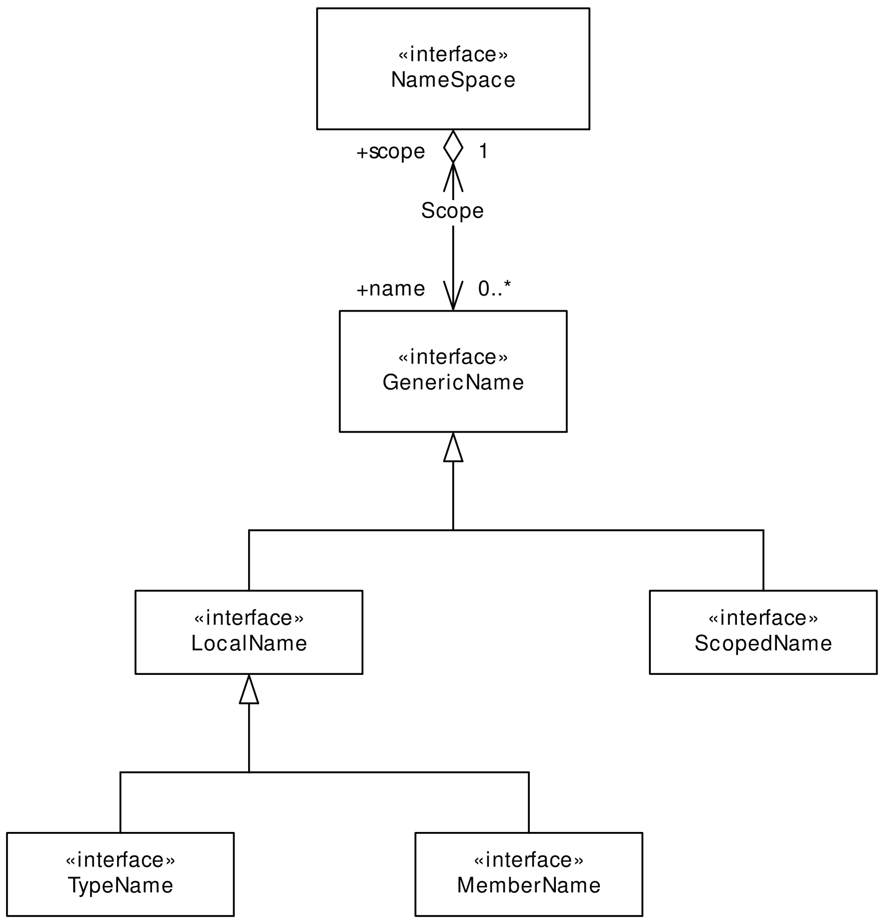


**Figure 17 — Name types**

### 7.5.2 NameSpace

#### 7.5.2.1 General

NameSpace defines a domain in which "names" given by character strings (possibly under local constraints enforced by the namespace) can be mapped to objects via a getObject operation. Examples include objects which form a namespace for their attributes, operations and associations, or schemas that form namespaces for their included data types or classes. The model for NameSpace is shown in Figure 18.

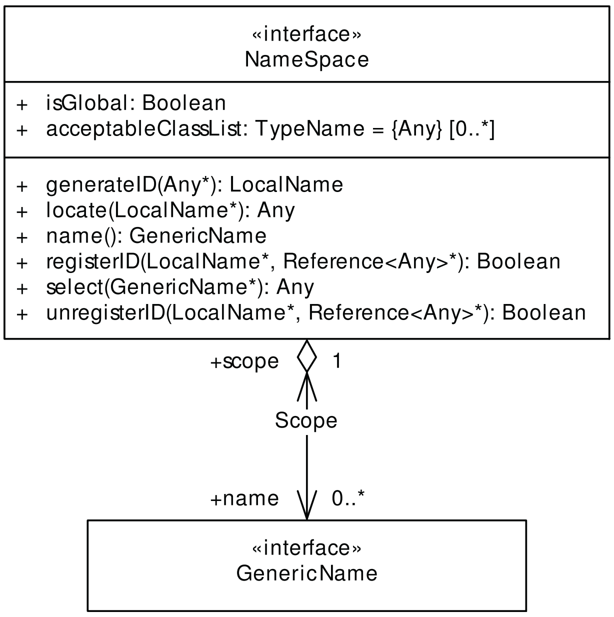

«interface»
NameSpace
+ isGlobal: Boolean
+ acceptableClassList: TypeName = {Any} [0..*]
+ generateID(Any*): LocalName
+ locate(LocalName*): Any
+ name(): GenericName
+ registerID(LocalName*, Reference<Any>*): Boolean
+ select(GenericName*): Any
+ unregisterID(LocalName*, Reference<Any>*): Boolean
+scope
1
Scope
+name
0..*
«interface»
GenericName

**Figure 18 — Namespace type**

### 7.5.2.2 isGlobal

True if the namespace is a globally unique namespace.

### 7.5.2.3 acceptableClassList

List of acceptable type names.

### 7.5.2.4 Operation generateID

Generate a unique name within the namespace.

### 7.5.2.5 Operation locate

Locate an object with a name within the namespace.

### 7.5.2.6 Operation name

Return the generic name.

### 7.5.2.7 Operation registered

Register an unused name as a unique name within the namespace.

### 7.5.2.8 Operation select

Return an object with the generic name.

### 7.5.2.9 Operation unregisterID

Remove a name from the namespace.

#### 7.5.2.10 Role name

NameSpace has a set of names, each specified by role name of type GenericName.

### 7.5.3 GenericName

#### 7.5.3.1 General

GenericName (see Figure 19) is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.

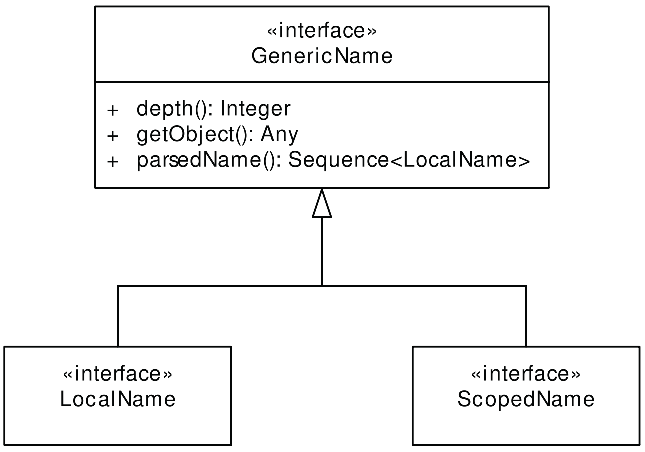


**Figure 19 — GenericName and subtypes**

#### 7.5.3.2 Operation depth

Depth of scope resolution chain.

#### 7.5.3.3 Operation getObject

Object having the generic name.

#### 7.5.3.4 Operation parsedName

Sequence of parsed local names.

#### 7.5.3.5 Role scope

Role scope of GenericName to NameSpace refers to the scope of instances of GenericName which is a namespace.

### 7.5.4 ScopedName

#### 7.5.4.1 General

ScopedName (see Figure 20) is a composite of a LocalName for locating another NameSpace and a GenericName valid in that NameSpace. ScopedName contains a LocalName as head and a GenericName, which might be a LocalName or a ScopedName, as tail.

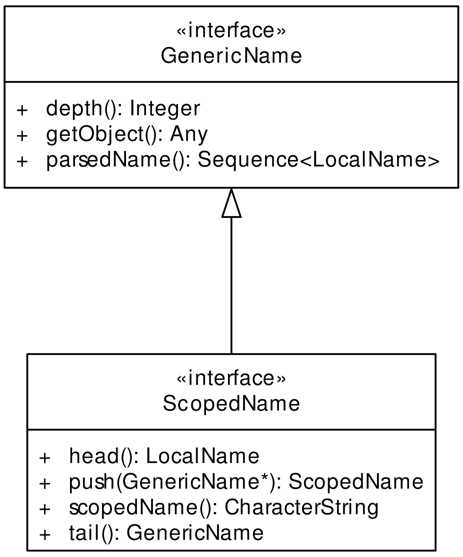


**Figure 20 — ScopedName**

#### 7.5.4.2 Operation head

Locate the namespace part of a scoped name.

#### 7.5.4.3 Operation push

Create a new scoped name.

#### 7.5.4.4 Operation scopedName

Return the scoped name.

#### 7.5.4.5 Operation tail

Locate the generic name part of a scoped name.

### 7.5.5 LocalName

#### 7.5.5.1 General

A LocalName (see Figure 21) references a local object directly accessible from the NameSpace.

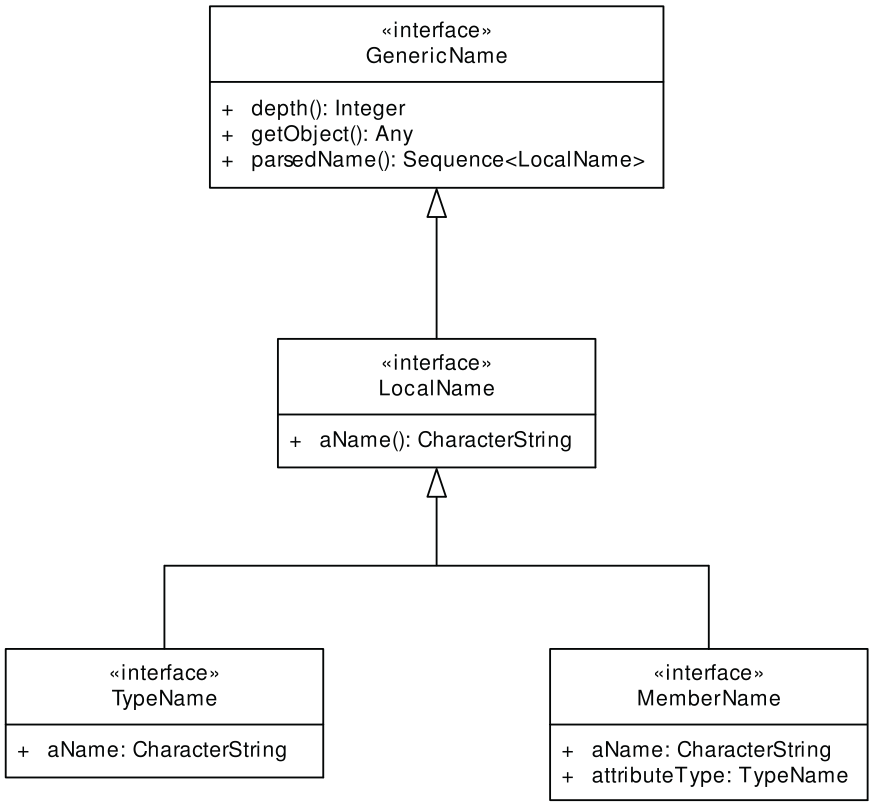


**Figure 21 — LocalName and subtypes**

#### 7.5.5.2 Operation aName

Return the local name.

### 7.5.6 TypeName

#### 7.5.6.1 General

A TypeName (see Figure 21) is a LocalName that references either a recordType or object type in some schema.

#### 7.5.6.2 aName

The name of the type.

### 7.5.7 MemberName

#### 7.5.7.1 General

A MemberName (see Figure 21) is a LocalName that references either an attribute slot in a record or recordType, an attribute, operation, or association role in an object instance or a type description in some schema.

#### 7.5.7.2 aName

The name of the member.

#### 7.5.7.3 attributeType

The type name of the member.

## 7.6 Any type

Any (see Figure 22) represents the universal root type which can be used as a root or base type in the type definition hierarchy. Any can be seen as an implicit super type of all other types. This use of Any is not strictly consistent with the semantics of the UML metamodel.

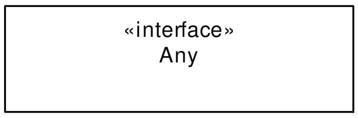


**Figure 22 — Any type**

## 7.7 Record types

### 7.7.1 General

Record and RecordType (see Figure 23) are described in 7.7.2 to 7.7.5.

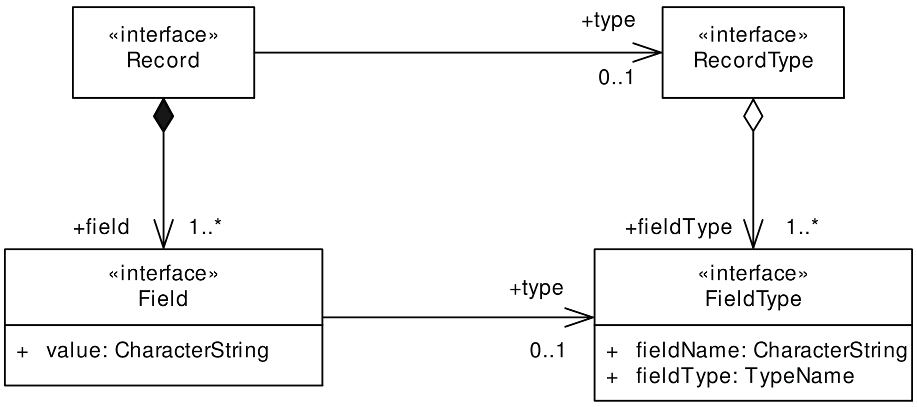


**Figure 23 — Record and RecordType**

### 7.7.2 Record

#### 7.7.2.1 General

Record generates a data type, called a record data type, whose values are heterogeneous aggregations of values of component data types, each aggregation having one value for each component data type, keyed by a fixed field-identifier [ISO/IEC 11404]. A record can have an associated RecordType. See ISO/IEC 11404 for further description.

#### 7.7.2.2 field

Member of a list of field value and field type pairs.

#### 7.7.2.3 type

Type of record.

### 7.7.3 RecordType

#### 7.7.3.1 General

A RecordType is a look up mechanism that associates attribute names to type names. Attribute names are locally mapped. Type names are most often either primitives or other types in the same schema as this record. Because the RecordType can control the structure of a set of records, it is essentially a metaclass for that set of records viewed as a class. See ISO/IEC 11404 for further description.

#### 7.7.3.2 fieldType

Member of a list of field types of a record.

### 7.7.4 Field

#### 7.7.4.1 General

A Field is a pair of a field value and its corresponding field type.

#### 7.7.4.2 value

Element of a value space that is consistent with its field type.

#### 7.7.4.3 type

Identifier of the field type.

### 7.7.5 FieldType

#### 7.7.5.1 General

A FieldType describes the name and type of a field.

#### 7.7.5.2 fieldName

Name of the field type.

#### 7.7.5.3 fieldType

Type of the field value.

## 7.8 NULL and EMPTY values

Several of the operations defined in this International Standard use NULL and EMPTY as possible values. NULL means that the value asked for is undefined. This International Standard assumes that all NULL values are equivalent. If a NULL is returned when an object has been requested, this means that no object matching the criteria specified exists. EMPTY refers to objects that can be interpreted as being sets that contain no elements. Unlike programming systems that provide strongly typed aggregates, this International Standard uses the mathematical tautology that there is one and only one empty set. Any object representing the empty set is equivalent to any other set that does the same. Other than being empty, such sets lack any inherent information, and so a NULL value is assumed equivalent to an EMPTY set in the appropriate context.

# Annex A
(normative)

# Abstract test suite

## A.1 UML versions

### A.1.1 General

UML models claiming conformance to this International Standard must conform with conformance class UML2 or conforming as described in A.1.2 and A.1.3. respectively.

### A.1.2 Models conforming to UML 2

a) Test Purpose: Verify that the model is according to the rules of UML 2.

b) Test Method: Inspect the model and the documentation of the model mapping.

c) Reference: 6.2, Requirements 1 and 3 to 17, and Annex D, Requirement 26.

d) Test Type: Capability.

### A.1.3 Models in UML 1 schemas conforming to UML 2

a) Test Purpose: Verify that the model is able to be mapped to a model according to the rules of UML 2.

b) Test Method: Inspect the model and the documentation of the model mapping.

c) Reference: 6.2, Annex B, Requirements 23 to 24 for mapping from UML 1, A.1.2.

d) Test Type: Capability.

### A.1.4 Models in other schema languages conforming to UML 2

a) Test Purpose: Verify that the model is able to be mapped to a model according to the rules of UML 2.

b) Test Method: Inspect the model and the documentation of the model mapping.

c) Reference: 6.2, Requirement 2, A.1.2.

d) Test Type: Capability.

## A.2 Data types

### A.2.1 Core types

a) Test Purpose: Verify the correct use of core data types in an application schema.

b) Test Method: Inspect the documentation of the application schema or profile.

c) Reference: Clause 7, Requirement 22

d) Test Type: Capability.

### A.2.2 Core and extension types

a) Test Purpose: Verify the correct use of core and extension data types in an application schema.

b) Test Method: Inspect the documentation of the application schema or profile.

c) Reference: Clause 7, Annex C, Requirement 25, A.2.1.

d) Test Type: Capability.

## A.3 Model documentation

a) Test Purpose: Verify the correct use content of the documentation.

b) Test Method: Inspect the documentation of the model.

c) Reference: 6.16, Requirements 18 to 21.

d) Test Type: Capability.

# Annex B
(normative)

# Rules for mapping UML 1 models to UML 2 models

There are a few differences between UML 1 as used in the previous version of the UML profile and UML 2 as defined in this International Standard. Because of this we need a way of mapping the old UML 1 based models to the new UML version. This annex specifies some of these mapping rules.

For a UML 1 model to be a valid UML 2 model the following model mapping is required:

Requirement 23. All types shall be mapped to interfaces, i.e. all classifiers with keyword <<type>> changes keyword to <<interface>> .

Requirement 24. All type to interface realizations shall be changed into simple interface – interface inheritance relationships.

# Annex C
(normative)

# Data types – extension types

## C.1 Introduction

Annex C specifies data types that are not considered to be core data types of the UML profile specified in this International Standard. These types are not directly related to how UML is used as a conceptual modelling language for geographic information and are regarded as extensions to the core data types defined in Clause 7.

| |
|---|
| Requirement 25. Platform neutral models of geographic information shall use these extended data types when appropriate, and shall not create alternative types with same meaning. |

An overview of the packages representing extension data types is shown in Figure C.1.

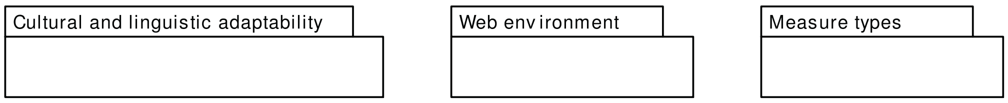


**Figure C.1 — Packages for extension data types**

## C.2 Cultural and linguistic adaptability

### C.2.1 General

Clause C.2 specifies data types that can include instances of information in specific languages. Figure C.2 provides an overview of the types supporting cultural and linguistic adaptability.

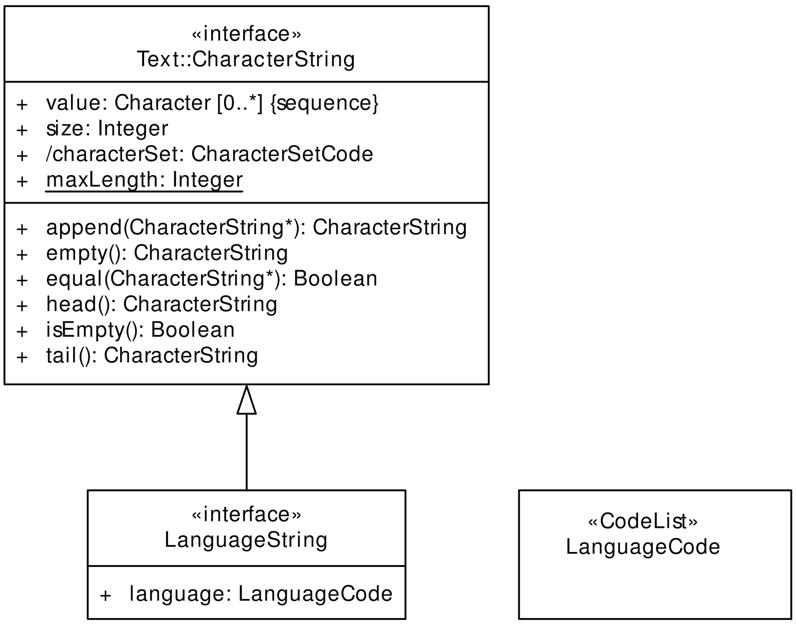

«interface»
Text::CharacterString
+ value: Character [0..*] {sequence}
+ size: Integer
+ /characterSet: CharacterSetCode
+ maxLength: Integer
+ append(CharacterString*): CharacterString
+ empty(): CharacterString
+ equal(CharacterString*): Boolean
+ head(): CharacterString
+ isEmpty(): Boolean
+ tail(): CharacterString
«interface»
LanguageString
+ language: LanguageCode
«CodeList»
LanguageCode

**Figure C.2 — Language specific text types**

### C.2.2 LanguageString

Alternative type to CharacterString to be used when strings in several different languages need to be modelled.

### C.2.3 language

Attribute language may store designation of the language used, as specified in code list LanguageCode with codes from any part of ISO 639.

## C.3 Web environment types

### C.3.1 General

Data types described here are specific to the World Wide Web environment where XML documents are typically processed. Figure C.3 defines the metadata required to describe elements specific to working in a web environment. Each data type is documented in C.3.2 to C.3.5.

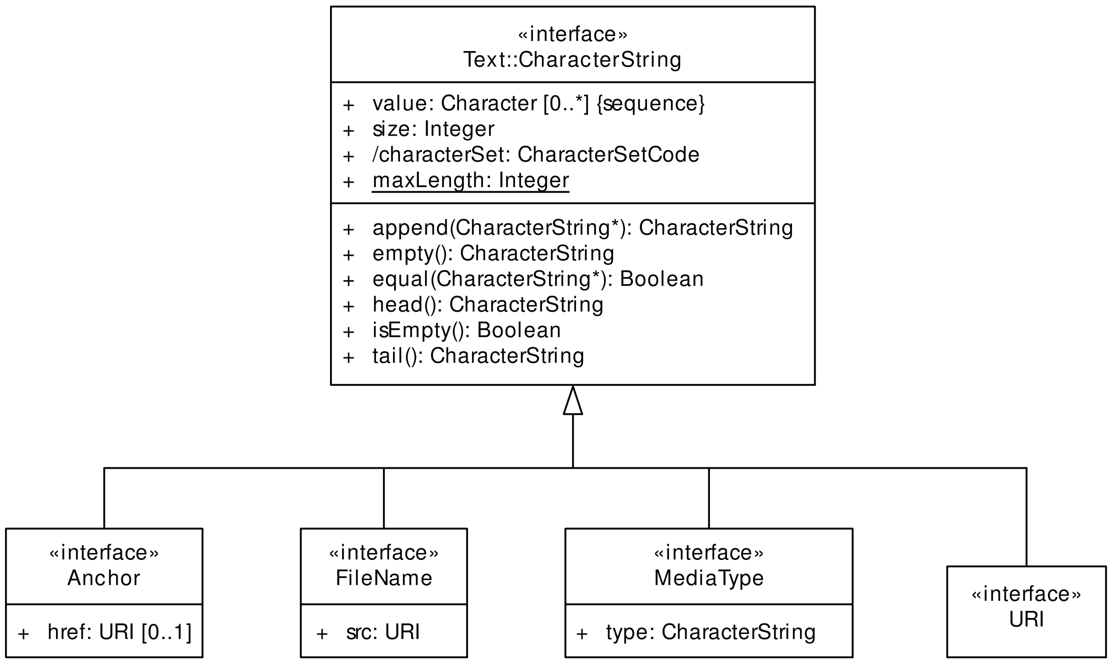

«interface»
Text::CharacterString
+ value: Character [0..*] {sequence}
+ size: Integer
+ /characterSet: CharacterSetCode
+ maxLength: Integer
+ append(CharacterString*): CharacterString
+ empty(): CharacterString
+ equal(CharacterString*): Boolean
+ head(): CharacterString
+ isEmpty(): Boolean
+ tail(): CharacterString
«interface»
Anchor
+ href: URI [0..1]
«interface»
FileName
+ src: URI
«interface»
MediaType
+ type: CharacterString
«interface»
URI

**Figure C.3 — Web environment types**

### C.3.2 Anchor

#### C.3.2.1 General

Supports hyper-linking capabilities and ensures a web-like implementation of CharacterStrings. The type Anchor (see Figure C.3) has been introduced to provide the ability to reference an external resource providing more information about the value of a CharacterString.

#### C.3.2.2 href

The attribute href supplies the data that allows an XLink application to find a remote resource (or resource fragment). [W3C XLINK]

### C.3.3 FileName

#### C.3.3.1 General

Supports explicitly referencing an external file corresponding to a property containing the name of the file. FileName is shown in Figure C.3.

#### C.3.3.2 src

The attribute src provides a machine readable path to the location of a corresponding file.

### C.3.4 MediaType

#### C.3.4.1 General

Supports identification of the file type using the media type name and subtype name. MediaType is shown in Figure C.3. The base CharacterString value of the MediaType provides a human readable description; the type attribute is drawn from the Internet Assigned Numbers Authority - IANA's list of media types (http://www.iana.org/assignments/media-types/). The actual path to the file is stored elsewhere, in a different attribute.

#### C.3.4.2 type

Provides the media type name and subtype name.

EXAMPLE "text/plain" where text is type name and plain is subtype name.

### C.3.5 URI

Uniform Resource Identifier (URI), is a compact string of characters compliant with RFC 3986, which is used to identify or name a resource. URI is shown in Figure C.3.

## C.4 Measure types

### C.4.1 General

Figure C.4 gives an overview of measure and units of measure data types that are described in Clause C.4.

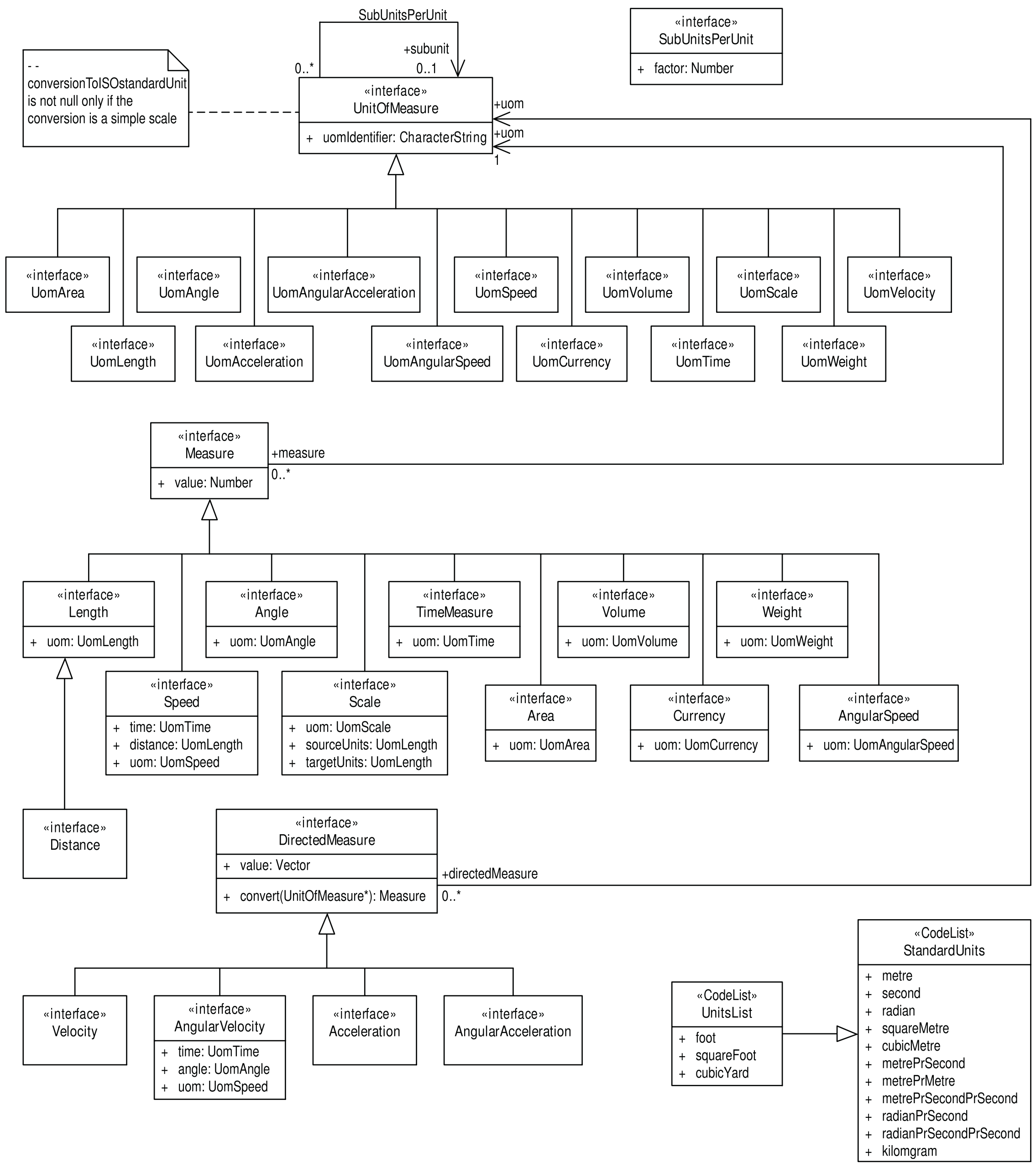


**Figure C.4 — Measure types**

### C.4.2 Measure

A Measure is the result from performing the act or process of ascertaining the value of a characteristic of some entity.

### C.4.3 DirectedMeasure

A DirectedMeasure is the result of ascertaining the value of a characteristic of some entity where direction is an essential aspect of the characteristic.

### C.4.4 UnitOfMeasure

A unit of measure is a quantity adopted as a standard of measurement for other quantities of the same kind.

The role uom (see Figure C.4) is constrained by the value of the attribute uom in each Measure subclass.

### C.4.5 Area

Area is the measure of the physical extent of any 2-D geometric object.

### C.4.6 UomArea

UomArea is any of the reference quantities used to express the value of area. Common units include square length units, such as square metres and square feet. Other common unit include acres (in the US) and hectares.

### C.4.7 Length

Length is the measure of distance as an integral, for example the length of curve, the perimeter of a polygon as the length of the boundary.

### C.4.8 Distance

Distance is used as a type for returning the separation between two points.

### C.4.9 UomLength

UomLength is any of the reference quantities used to express the value of the length, distance between two entities. Examples are the English System of feet and inches or the metric system of millimetres, centimetres and metres, also the System International (SI) System of Units.

### C.4.10 Angle

Angle is the amount of rotation needed to bring one line or plane into coincidence with another, generally measured in radians or degrees.

### C.4.11 UomAngle

UomAngle is any of the reference quantities used to express the value of angles. In the US, the sexagesimal system of degrees, minutes and seconds is frequently used. The radian measurement system is also used. In other parts of the world the grad angle, gon, measuring system is used.

### C.4.12 Scale

Scale is the ratio of one quantity to another, often unitless.

### C.4.13 UomScale

UomScale is any of the reference quantities used to express the value of scale, or the ratio between quantities with the same unit. Scale factors are often unitless.

### C.4.14 TimeMeasure

TimeMeasure is the designation of an instant on a selected time scale, astronomical or atomic. It is used in the sense of time of day.

### C.4.15 UomTime

UomTime is any of the reference quantities used to express the value of or reckoning the passage of time and/or date, (e.g. seconds, minutes, days, months).

### C.4.16 Volume

Volume is the measure of the physical space of any 3-D geometric object.

### C.4.17 UomVolume

UomVolume is any of the reference quantities used to express the value of volume.

### C.4.18 Speed

Speed is the rate at which someone or something moves, generally expressed as distance over time. It is distinct from velocity in that speed can be measured along a curve.

### C.4.19 UomSpeed

UomSpeed provides both the unit of measure of the distance and of the time, for example kilometres per hour, or metres per second.

### C.4.20 AngularSpeed

AngularSpeed (often known as angular velocity) is the magnitude of the rate of change of angular position of a rotating body. Angular speed is always positive.

### C.4.21 UomAngularSpeed

UomAngularSpeed provides both the unit of measure of the angle and of the time, for example degrees per second.

### C.4.22 Weight

Weight is the force exerted on the mass of a body by a gravitational field.

### C.4.23 UomWeight

UomWeight is any of the reference quantities used to express force, such as newton.

### C.4.24 Currency

Currency is a system of money in general use in a particular country or countries.

### C.4.25 UomCurrency

UomCurrency indicates the specific currency, such as one listed in ISO 4217[8].

### C.4.26 Velocity

Velocity is the instantaneous rate of change of position with time. It is usually calculated using the simple formula, the change in position during a given time interval.

### C.4.27 UomVelocity

UomVelocity is any of the reference quantities used to express the value of velocity.

### C.4.28 AngularVelocity

AngularVelocity is the instantaneous rate of change of angular displacement with time.

### C.4.29 UomAngularVelocity

UomAngularVelocity is any of the reference quantities used to express the value of angular velocity. It must include an indication of which direction is considered positive.

### C.4.30 Acceleration

Acceleration is the rate of change of velocity per unit of time.

### C.4.31 UomAcceleration

UomAcceleration provides both the unit of measure of the velocity and of the time, for example metres per second, per second.

### C.4.32 AngularAcceleration

AngularAcceleration is the rate of change of angular velocity per unit of time.

### C.4.33 UomAngularAcceleration

UomAcceleration provides both the unit of measure of the angular velocity and of the time, for example degrees per second, per second.

### C.4.34 SubUnitsPerUnit

Sub units per unit.

### C.4.35 StandardUnits

Standard units are the base and named derived units that form part of the International System of Units (SI), plus units which are based on SI units and are required for the other measures, for example metres per second.

### C.4.36 UnitsList

This code list contains other units which are not part of SI, but are in common use in some parts of the world.

# Annex D
(normative)

# Formal UML profile

## D.1 General

Annex D describes the ISO 19103 UML profile formally. That is, it describes how the concepts defined in this International Standard are related to the UML metamodel. In essence, all stereotypes defined in this International Standard are modelled as extensions to the UML metamodel.

## D.2 Overview of relevant UML metaclasses

The model in Figure D.1 shows the UML metaclasses that are extended in this profile.

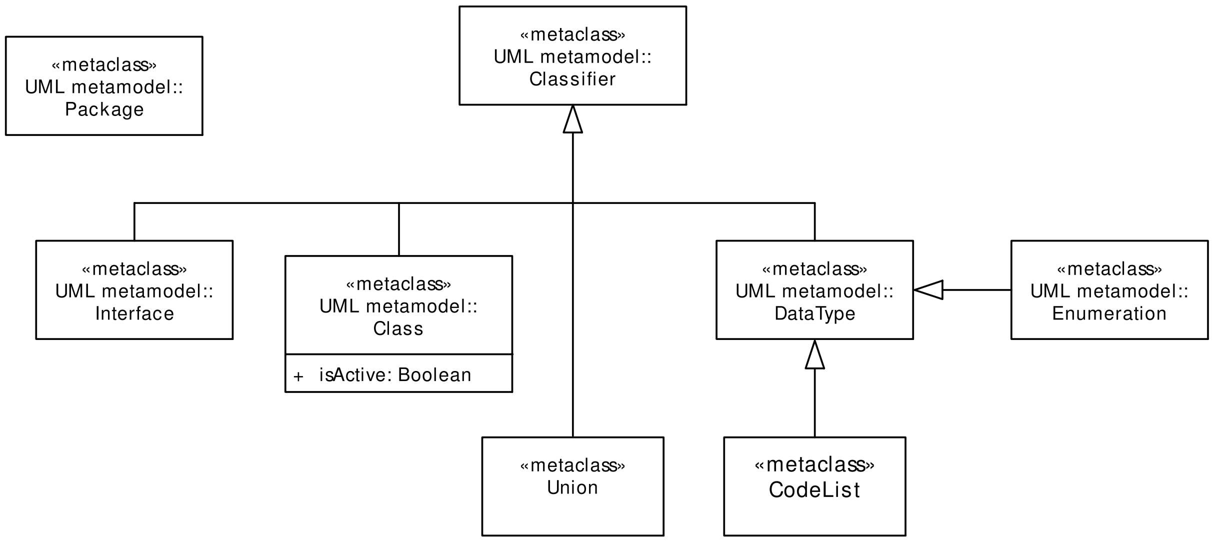

«metaclass» UML metamodel:: Package
«metaclass» UML metamodel:: Classifier
«metaclass» UML metamodel:: Interface
«metaclass» UML metamodel:: Class
+ isActive: Boolean
«metaclass» UML metamodel:: DataType
«metaclass» UML metamodel:: Enumeration
«metaclass» Union
«metaclass» CodeList

**Figure D.1 — UML metaclass overview**

## D.3 ISO 19103 UML profile

The model in Figure D.2 shows all stereotypes defined in this International Standard and their relation to the respective UML metamodel classes. The relation between a metaclass and a stereotype is defined by an extension, the stereotype extends the metaclass. Tagged values are defined as attributes for the stereotypes, for instance attribute ‘codeList’ of stereotype CodeList which represents a URI referencing the original code list maintained by e.g. some authority.

| Requirement 26. Schemas for geographic information shall, where appropriate, use the stereotypes “Leaf”, “Union” and “CodeList” as described in Annex D. |
|---|

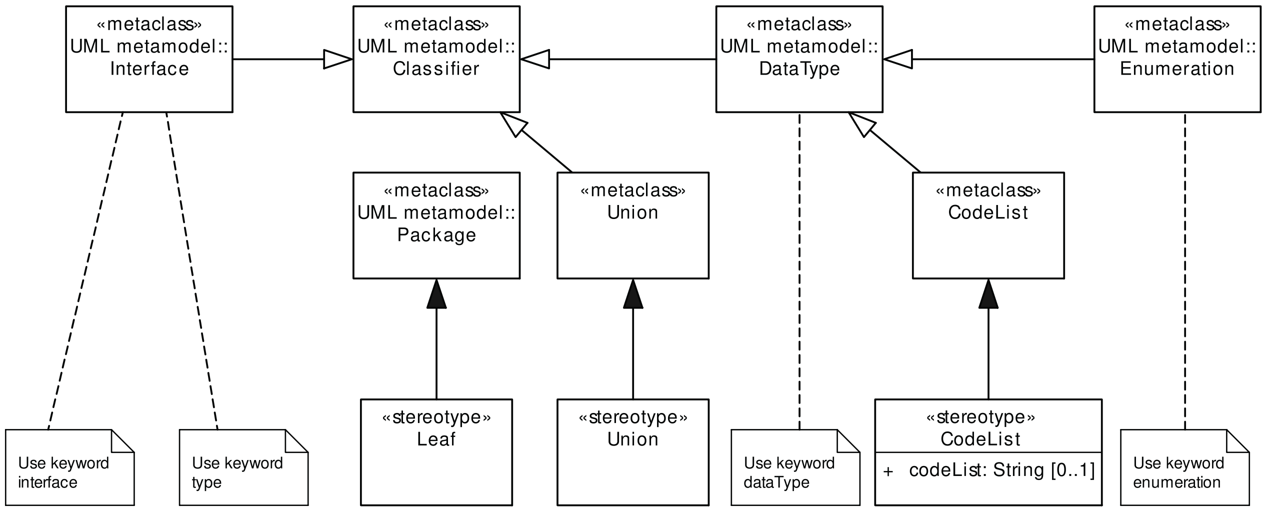

«metaclass» UML metamodel:: Interface
«metaclass» UML metamodel:: Classifier
«metaclass» UML metamodel:: DataType
«metaclass» UML metamodel:: Enumeration
«metaclass» UML metamodel:: Package
«metaclass» Union
«metaclass» CodeList
«stereotype» Leaf
«stereotype» Union
«stereotype» CodeList
+ codeList: String [0..1]
Use keyword interface
Use keyword type
Use keyword dataType
Use keyword enumeration

**Figure D.2 — ISO 19103 stereotypes and keywords**

These stereotypes and keywords can be applied to model elements to give them the semantics of the stereotypes and metaclasses. The first letter of applied stereotypes is normally capitalized, applied but stereotypes and keywords are case insensitive. See Figure D.3 for an example of the keyword interface applied to the class Any.

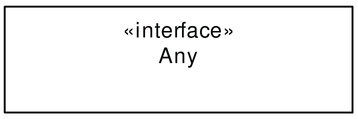


**Figure D.3 — Example of class Any with applied keyword interface**

# Annex E
(informative)

# On conceptual schema languages

## E.1 Reality and geographic concepts

Figure E.1 shows the ISO geographic information standards' approach of bringing together concepts and technology from the geographic information and information technology disciplines. The ISO geographic information standards were originally grouped into five areas: Framework and reference model, data models and operators, data administration, geographic information services and profiles and functional standards. The information technology input shows conceptual schema languages as one of the main inputs into this work.

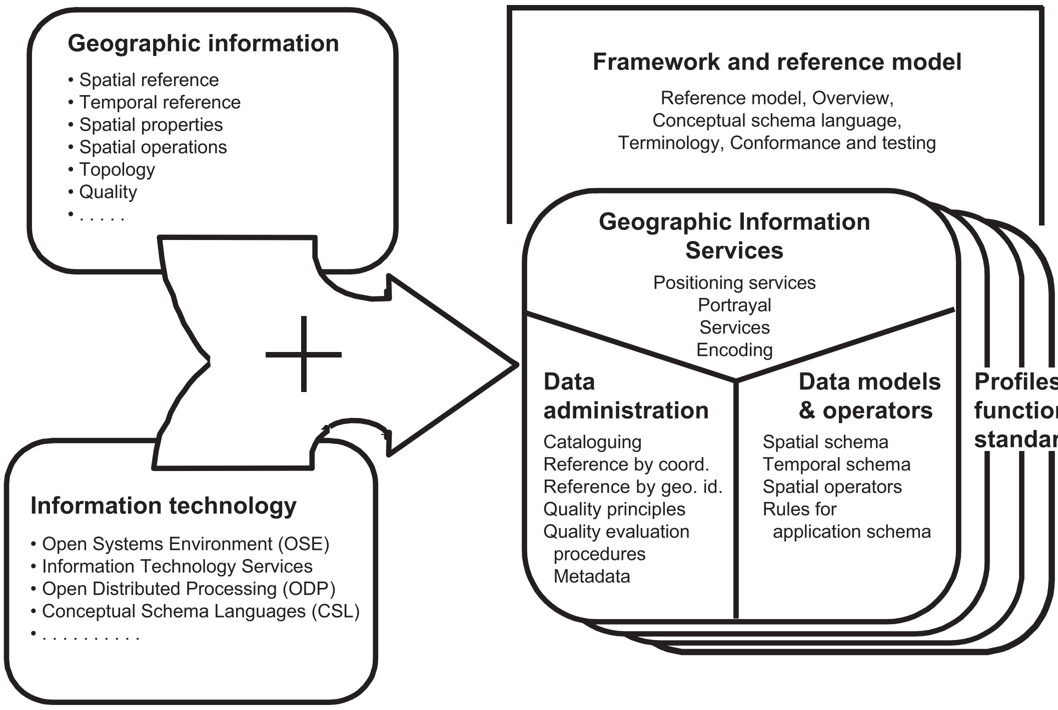


**Figure E.1 — ISO geographic information standards work based on both GIS and IT concepts**

## E.2 Universe of discourse and conceptual schema

A conceptual schema classifies objects into types and classes, identifying types of objects according to their properties (structure and behaviour) and associations between types of objects.

Figure E.2 describes the relationship between modelling reality and the resulting conceptual schema. A "universe of discourse" is a selected piece of "reality" (or a hypothetical world) that a human being wishes to describe in a model. The universe of discourse may include not only features such as watercourses, lakes, islands, property boundary, property owner and exploitation area, but also their attributes, their operations and the relationships that exist among such features. A universe of discourse is described in a "conceptual model". The "conceptual schema" is a rigorous description of a conceptual model for some universe of discourse. A conceptual schema that defines the universe of discourse relating to a specific application or a set of applications shall be called an "application schema". A conceptual schema also defines relationships between model elements and constraints on the elements and their relationships. These constraints define valid states of the modelled system. An actual universe of discourse can be digitally represented by data that are structured according to the specification in the corresponding conceptual schema.

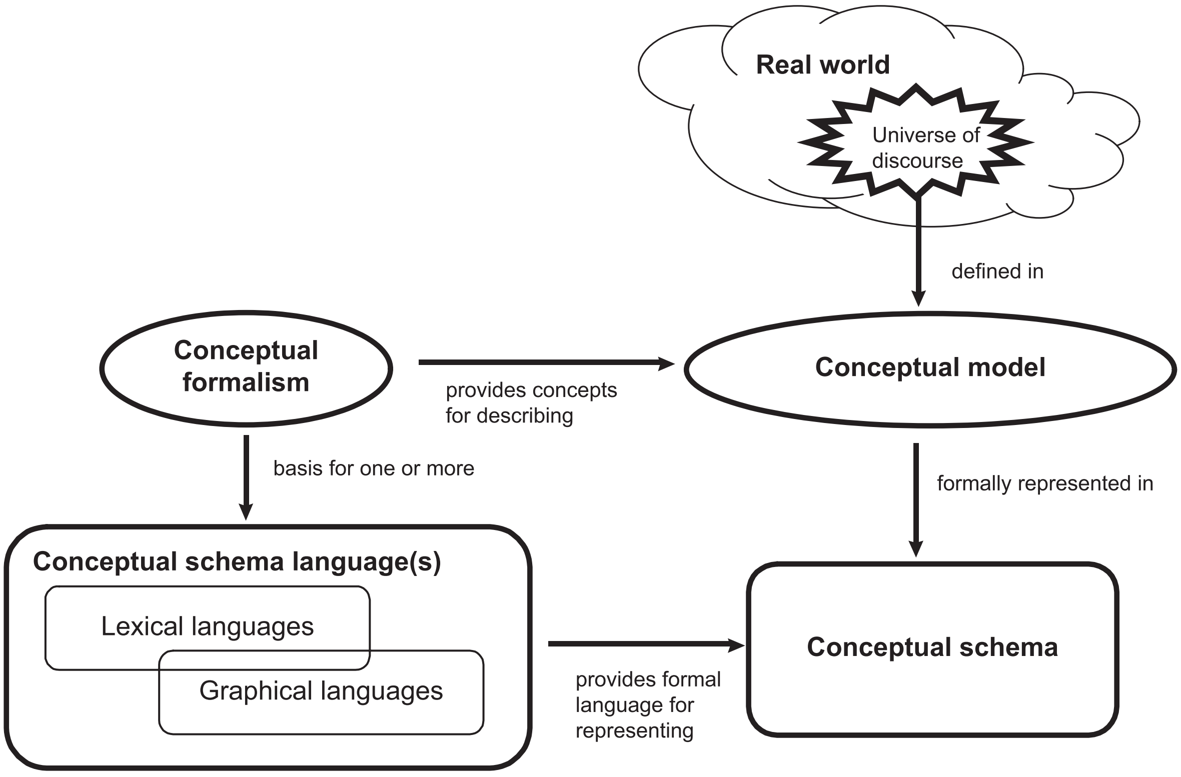


**Figure E.2 — From reality to conceptual schema**

A “conceptual schema language” is used to describe a conceptual schema. A conceptual schema language is a formal language that can be parsed by a computer or a human being. A conceptual schema language contains all linguistic constructs necessary to formulate a conceptual schema and to manipulate its contents.

A conceptual schema language is based upon a “conceptual formalism”. The conceptual formalism provides the rules, constraints, inheritance mechanisms, events, functions, processes and other elements that make up a conceptual schema language. These elements may also be used to create conceptual schemas that describe a given information system or information technology standard, if this is the chosen universe of discourse. A conceptual formalism provides a basis for the formal definition of all knowledge considered relevant to an information technology application. More than one conceptual schema language, either lexical or graphical, can adhere to and be mapped to the same conceptual formalism.

Conceptual schemas developed for parts of the ISO geographic information standards are represented using a conceptual schema language. These conceptual schemas are integrated into “application schemas” which define the structure of geographic data processed by computer systems. Encoding of geographic information in support of implementation is addressed in ISO 19118.

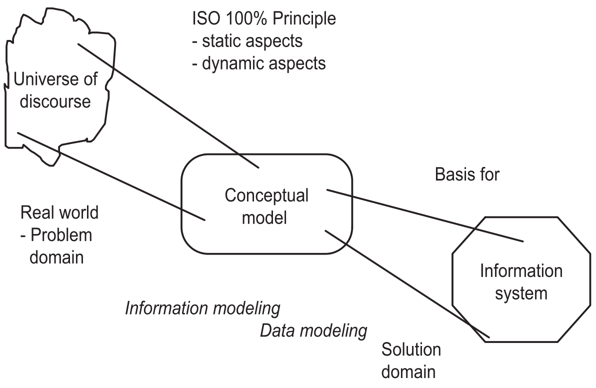


**Figure E.3 — The ISO 100 % principle for conceptual modelling**

Earlier ISO work (ISO/TR 9007) defined and established the 100 % principle for conceptual modelling (see Figure E.3). The goal is to represent 100 % of both the static and the dynamic aspects of the universe of discourse. This implies a need to represent both structure and behaviour for the area of interest.

Figure E.4 shows the metamodel layers of CSMF, the Conceptual Schema Modelling Facility.

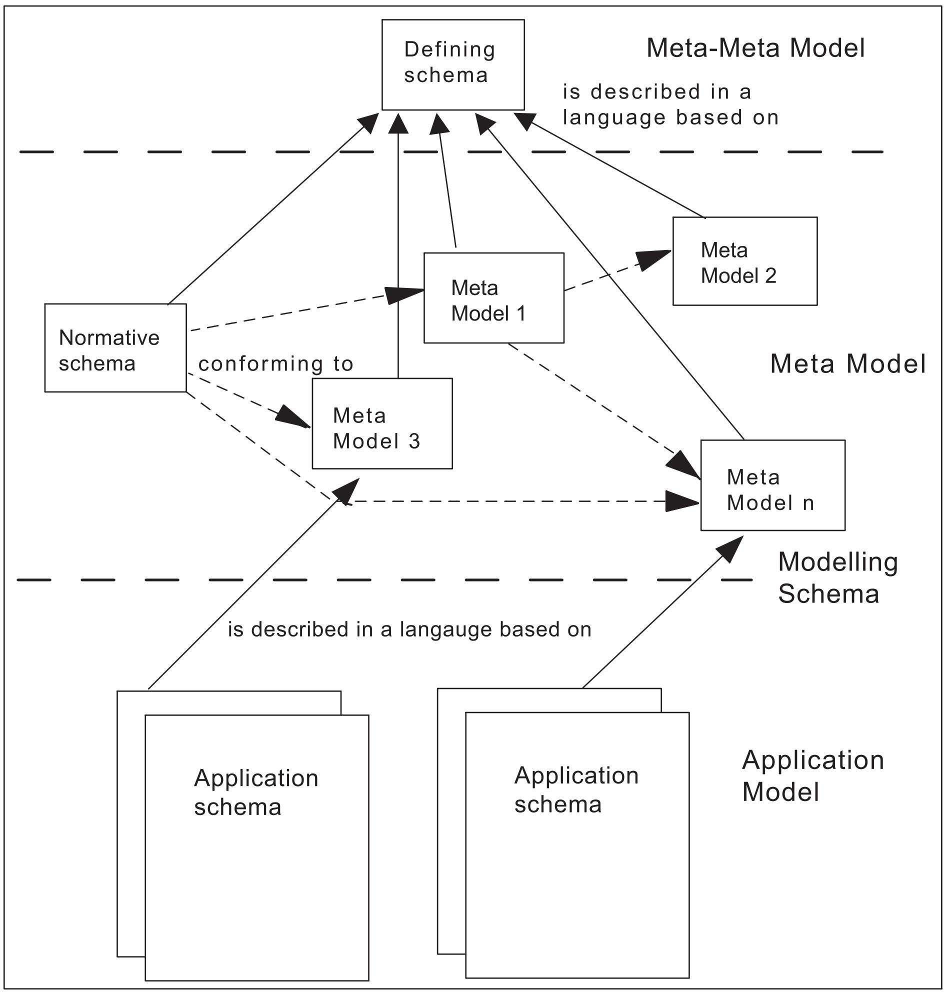


**Figure E.4 — Metamodel layers of CSMF — Conceptual Schema Modelling Facility**

## E.3 General Feature Model

The General Feature Model (also presented in ISO 19109) was developed independently of existing metamodels and has been used as a guide for defining rules for application schema modelling. The General Feature Model focuses on modelling features together with their associated attributes, relationships and operations. Attributes might have associated quality. The General Feature Model is mapped onto the UML metamodel in ISO 19109. Figure E.5 shows the relationship of the General Feature Model and UML.

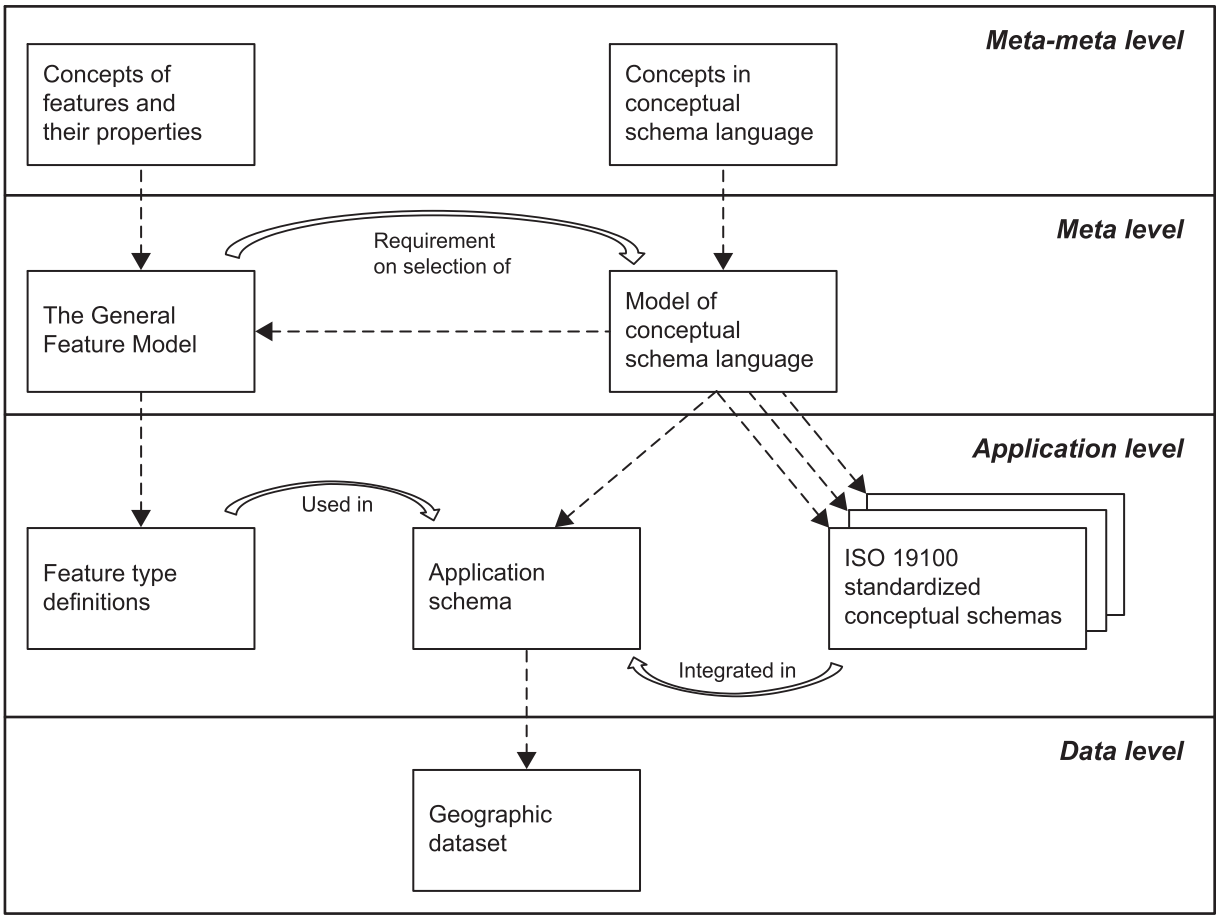

Meta-meta level
Concepts of features and their properties
Concepts in conceptual schema language
Meta level
Requirement on selection of
The General Feature Model
Model of conceptual schema language
Application level
Used in
Feature type definitions
Application schema
ISO 19100 standardized conceptual schemas
Integrated in
Data level
Geographic dataset

**Figure E.5 — Relationship of the General Feature Model and the UML model**

The General Feature Model identifies the following special attributes as particularly important for the geographic information domain:

— Temporal,

— Spatial (Position/Topology),

— Quality,

— Geographic identifier,

— Thematic attributes.

The General Feature Model identifies the following special relationships as important for the geographic information domain:

— Specialization,

— Aggregation,

— Spatial (Position/Topology),

— Logical/Association.

The General Feature Model is further presented in ISO 19109. Figure E.6 shows an extract of main classes of the General Feature Model.

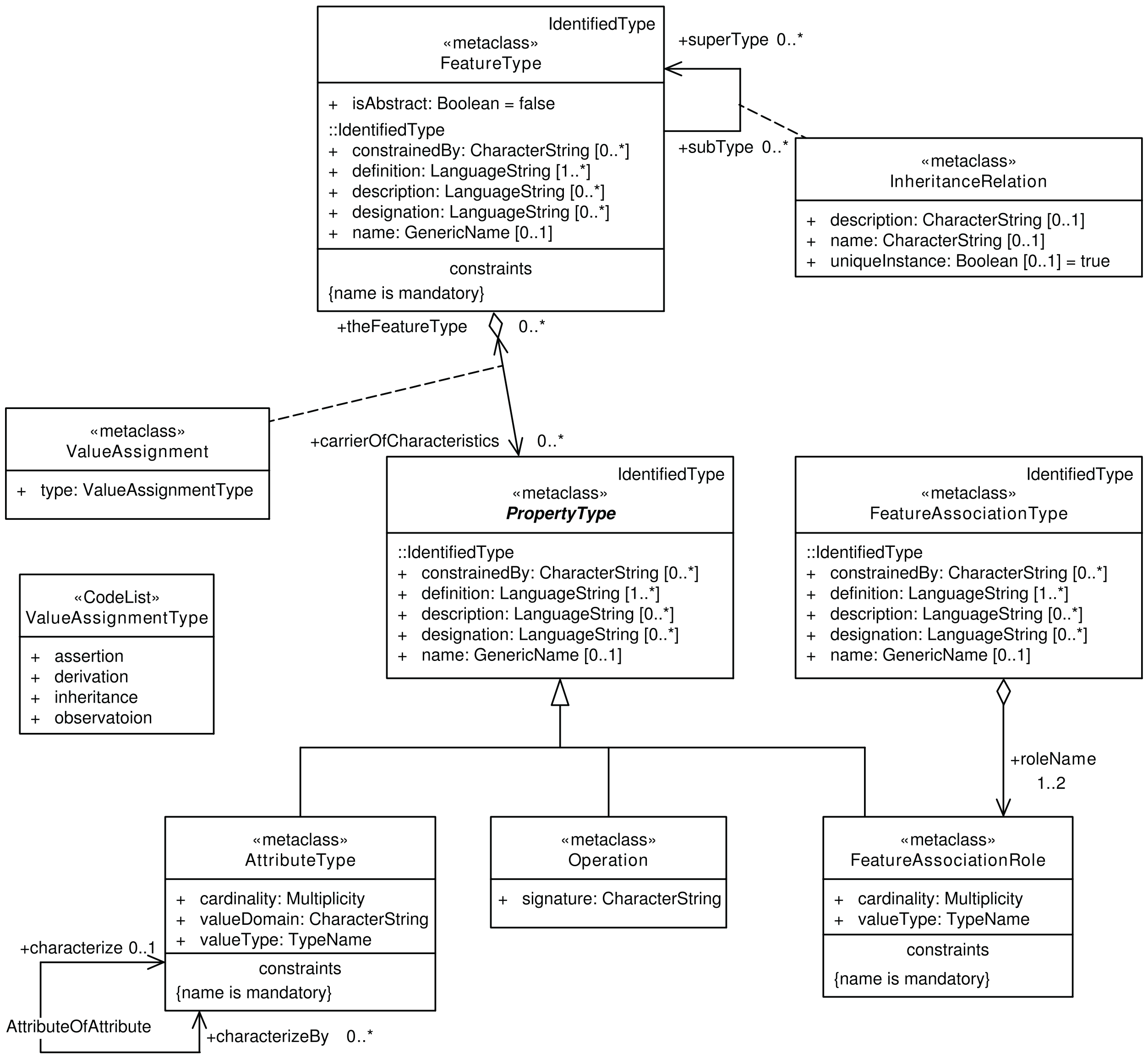

IdentifiedType
«metaclass»
FeatureType
+ isAbstract: Boolean = false
::IdentifiedType
+ constrainedBy: CharacterString [0..*]
+ definition: LanguageString [1..*]
+ description: LanguageString [0..*]
+ designation: LanguageString [0..*]
+ name: GenericName [0..1]
constraints
{name is mandatory}
+superType 0..*
+subType 0..*
«metaclass»
InheritanceRelation
+ description: CharacterString [0..1]
+ name: CharacterString [0..1]
+ uniqueInstance: Boolean [0..1] = true
+theFeatureType
0..*
«metaclass»
ValueAssignment
+ type: ValueAssignmentType
+carrierOfCharacteristics
0..*
IdentifiedType
«metaclass»
PropertyType
::IdentifiedType
+ constrainedBy: CharacterString [0..*]
+ definition: LanguageString [1..*]
+ description: LanguageString [0..*]
+ designation: LanguageString [0..*]
+ name: GenericName [0..1]
IdentifiedType
«metaclass»
FeatureAssociationType
::IdentifiedType
+ constrainedBy: CharacterString [0..*]
+ definition: LanguageString [1..*]
+ description: LanguageString [0..*]
+ designation: LanguageString [0..*]
+ name: GenericName [0..1]
«CodeList»
ValueAssignmentType
+ assertion
+ derivation
+ inheritance
+ observatoion
+roleName
1..2
«metaclass»
AttributeType
+ cardinality: Multiplicity
+ valueDomain: CharacterString
+ valueType: TypeName
constraints
{name is mandatory}
«metaclass»
Operation
+ signature: CharacterString
«metaclass»
FeatureAssociationRole
+ cardinality: Multiplicity
+ valueType: TypeName
constraints
{name is mandatory}
+characterize 0..1
AttributeOfAttribute
+characterizeBy 0..*

**Figure E.6 — The main classes of the ISO 19109 General Feature Model**

ISO 19109 contains a further description of the relationship between the General Feature Model and UML and the UML metamodel. ISO 19109 also includes a further description of terms such as feature, feature association, feature attribute and feature operation. The UML core metamodel is sufficiently similar to the ISO 19109 General Feature Model, with small changes, that it can be used as a baseline for meeting the requirements of the General Feature Model.

UML has been developed in a metamodel framework similar to CSMF, as shown in Figure E.7.

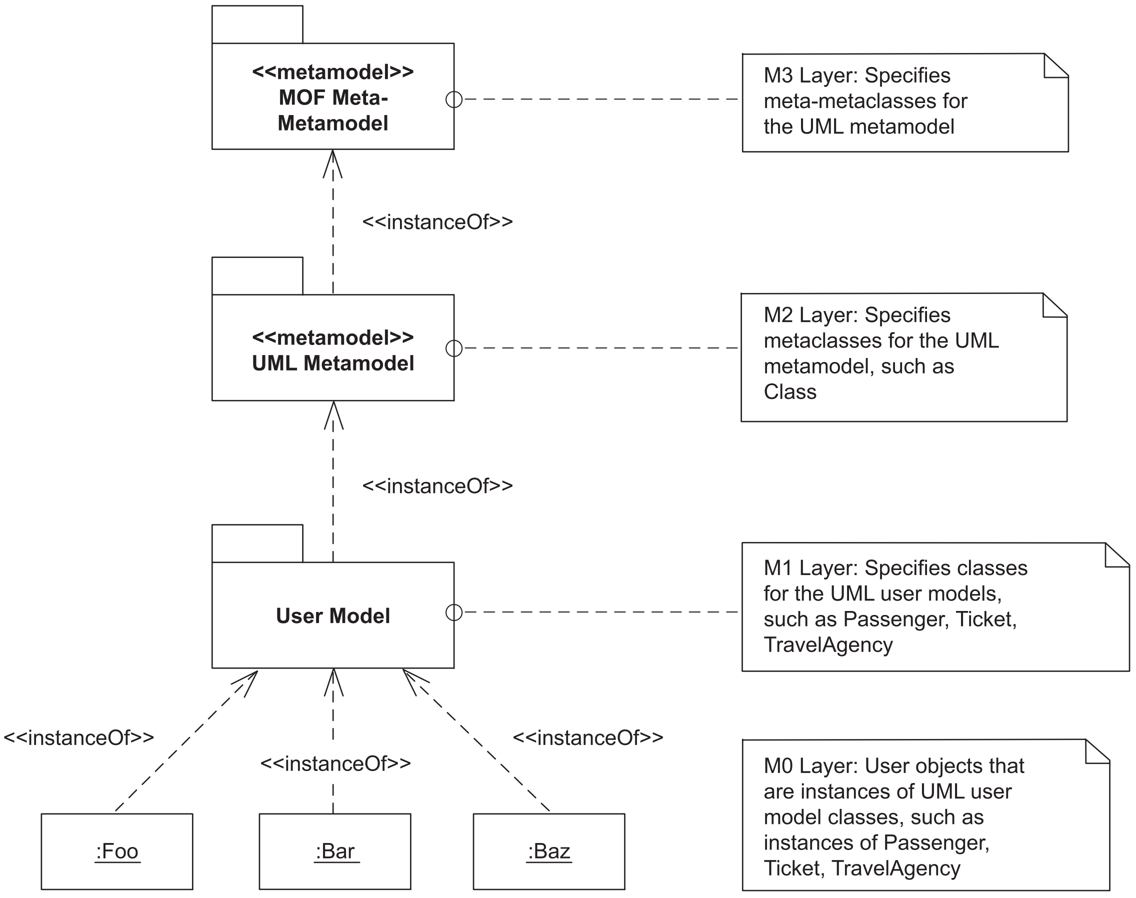

<<metamodel>>
MOF Meta-Metamodel
M3 Layer: Specifies meta-metaclasses for the UML metamodel
<<instanceOf>>
<<metamodel>>
UML Metamodel
M2 Layer: Specifies metaclasses for the UML metamodel, such as Class
<<instanceOf>>
User Model
M1 Layer: Specifies classes for the UML user models, such as Passenger, Ticket, TravelAgency
<<instanceOf>>
<<instanceOf>>
<<instanceOf>>
:Foo
:Bar
:Baz
M0 Layer: User objects that are instances of UML user model classes, such as instances of Passenger, Ticket, TravelAgency

**Figure E.7 — The metamodel architecture for UML**

## E.4 Architecture

ISO 19101-1 contains information on the use of the ISO RM-ODP viewpoint approach for modelling of geographic information and services.

The following is a short introduction to the goals and concepts in each of the ODP viewpoints, and the application of these viewpoints within the ISO geographic information standards (see Figure E.8).

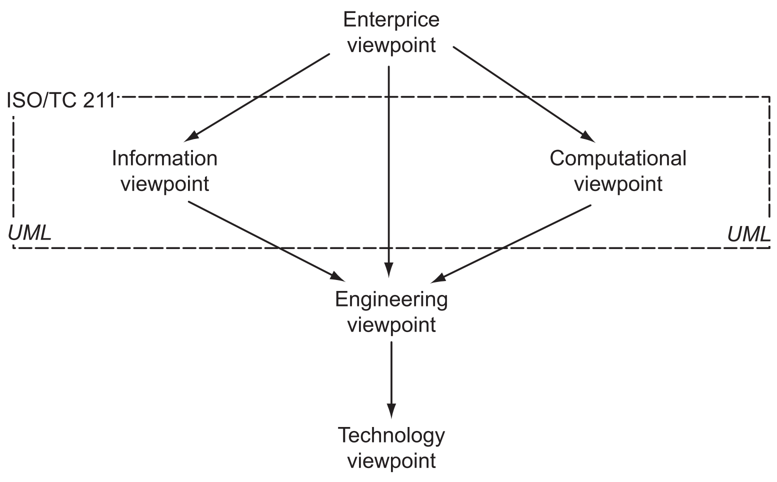


**Figure E.8 — Viewpoints in the ISO RM-ODP model**

**The enterprise viewpoint** is concerned with the purpose, scope and policies of the enterprise or business related to the specified systems and services. An enterprise specification of a service is a model of the service and the environment with which the service interacts. It describes the role of the service in the business, and the human user roles and business policies related to the service. The enterprise viewpoint provides a basis for defining the scope and context of the conceptual models for geographic information. The enterprise viewpoint will vary between different organizations and is thus used only for the generation of requirements. It is normally not part of the ISO geographic information standards.

**The computational viewpoint** is concerned with the interaction patterns between the components (services) of the system, described through their interfaces. A computational specification of a service is a model of the service interface as seen by a client, together with the required interactions that this service has with other services. The interactions among services may be described as a series of information sources and sinks. In geographic information, it is particularly important to show how a service is seen from a client perspective. The computational viewpoint is provided with implementation and technology neutral UML package and class diagrams and OCL constraints. Guidelines for the computational viewpoint are further described in Annex F.

**The information viewpoint** is concerned with the semantics of information and information processing. An information specification of an ODP system is a model of the information that it holds and of the information processing that it carries out. The information model is extracted from the individual components and provides a consistent common view on information which can be referenced by the specifications of information sources and sinks, and the information flows between them. For geographic information, the information viewpoint shows how the underlying information model (from the conceptual model) relates to the services identified in the computational viewpoint. This model is strongly related to the geographic information domain models. This will be described with UML package diagrams, UML class diagrams and OCL constraints[3]. This information model is implementation and technology neutral. Guidelines for the information viewpoint are further described in Annex F. Aspects dealing with the modelling of application schemas are described in ISO 19109.

**The engineering viewpoint** is concerned with the design of implementations within distributed, networked computing systems, i.e. the infrastructure required to support distribution. An engineering specification of an ODP system defines a networked computing infrastructure that supports the system structure defined in the computational specification, and provides the distribution transparencies that it defines. It describes mechanisms corresponding to the elements of the programming model, effectively defining an abstract machine which can carry out the computational actions, and the provision of the various transparencies to support distribution. ODP defines the following distribution transparencies: access, failure, location, migration, relocation, replication, persistence and transaction. In the ISO geographic information standards, engineering issues are separated from the functional specification in the information and computational viewpoints. Since the ISO geographic information standards focus on implementation neutral models, there is little emphasis on this viewpoint. It is assumed that the handling of various distribution transparencies will be done in the context of making implementation models in the technology viewpoint.

**The technology viewpoint** is concerned with the provision of an underlying infrastructure. A technology specification defines how a system is structured in terms of hardware and software components and underlying supporting infrastructure. In the ISO geographic information standards, it is important to show how a particular service can be mapped onto an underlying implementation technology such as XML Schema, SQL 3/ODBC, ODMG, CORBA, DCOM/OLE, Internet or similar infrastructure. This is also the area where we define mappings of the models from the information viewpoint onto underlying technologies with Geographic Information support, such as the OpenGIS API's for ODBC, CORBA, COM/OLE/DB and Internet. For this viewpoint, implementation specific UML models will be created in conformance with the implementation neutral models representing the information and computational viewpoints. This can be used as an intermediate step in the mapping down to implementation specifications directly in a language of the environment, such as XML Schema, CORBA IDL, COM IDL or SQL.

The UML (Unified Modeling Language) can be used to describe different kinds of models. One distinction of various kinds of models is into conceptual, specification and implementation models.

*Conceptual models* – describe the domain under study, independent of any computer or system representation of the model.

*Specification models* – describe both domain and system service models in an implementation neutral way, suitable for further detailing into an implementation model.

*Implementation models* – describe the realization of domain and system service models in an implementation specific way, related to the characteristics of the underlying platform.

The geographic domain models within the ISO geographic information standards have been developed as conceptual models and then further refined to specification models to be a suitable implementation neutral basis for system representation and data exchange. The service models within the ISO geographic information standards have been developed as specification models to reflect required system interfaces in an implementation neutral way. Some parts of the ISO geographic information standards also further describe implementation models for given specification models, to define corresponding implementation and platform specific models.

It is relevant to note that a conceptual modelling language can also be used to model software and hardware, if the domain under study is chosen to be a system. The benefit from this within the ISO geographic information standards is that the same conceptual schema language (UML) can be used for both domain models (information modelling) and for system models (service modelling).

# Annex F
(informative)

# Modelling guidelines

## F.1 Guidelines for modelling geographic information with UML

### F.1.1 General

The creation of UML models should preferably take place as a structured process. There are many proposed alternative processes for UML modelling, such as the Unified Software Development Process, Catalysis, the Select Perspective process, the Rational Unified Process, UML Components and others. None of these directly address the special concerns relating to standardization of information and service models in a standard suite such as the ISO geographic information standards, but are instead focused on the general development of systems and components. One of the differences is the need for models compliant with the ISO geographic information standards to specify implementation neutral (platform independent) models that can be profiled by creating mappings to different implementation specific (platform specific) models with different implementation environments.

Object Management Group (OMG) adopted the XMI specification for exchanging UML models using XML. The XMI specification allows for interoperability among UML tools.

The ISO geographic information standards focus on abstract, implementation-neutral UML models that can serve as specifications for implementations using various implementation mappings.

### F.1.2 General structure

A general structure of work products and models for a software development process using UML can be described as follows (adapted from the Unified Software Development Process):

— Business/Domain models

Domain model, Business Model, Business use cases (reverse/as-is, forward/to-be)

— Requirements models

Use case model, Actors, Architecture view, Glossary, User Interface Prototype

— Analysis/Architecture models

Architecture model – identify, service/control objects, information/entity/feature objects and boundary/interface objects, Analysis packages/classes, Use Case realization

— Design models (may be implementation neutral)

Design Model, Design System, Design Subsystem, Interfaces, Use Case realization – design, Design classes, Architecture view, Deployment model

— Implementation models

Implementation model, Components, Implementation subsystem, Interfaces, Integration build plan, Architecture view

— Test models

Test model, Test cases, Test plan/procedure, Test component, test evaluation

### F.1.3 General goals

The goal of the ISO geographic information standards is to create platform independent (implementation neutral) models. This will provide a baseline for creating platform specific (implementation specific) models for various environments, such as XML Schema, SQL, COM/OLE, CORBA, EXPRESS/SDAI and others. It is not necessary to standardize the process for how to create these models, only to specify the requirements for how to document the final models using UML, and this is the purpose of this International Standard.

The following guidelines assume that the problem domain has already been well understood, i.e. through the informal use of class modelling, to describe the essential concepts in the domain. It is also assumed that there is a general knowledge of the usage of the models – i.e. through a use-case based requirement specification and analysis, and that an analysis has been done to identify potential service types/objects and entity types/objects.

The following guidelines are focused on the process for creating implementation neutral models (abstract specifications) that later can be further transformed into detailed design and implementation models for various platforms and environments.

### F.1.4 General guidelines

The following "rules of general good behaviour" are not focused on the documents, but on the process. They are listed here as general guidance to those who do information and/or service modelling.

— Defined specialized terms used. Define everything that is not common software language. Avoid jargon, which requires one to recognize jargon.

— Do not use hidden assumptions. One of the purposes of modelling is to expose assumptions. Document the decisions, assumptions and reasons behind the model.

— Answer people's questions about the model, and document answers in the base document. Answers should be discussed in public technical committee meetings so that a consensus on controversial issues can be arrived at.

— Document the model in a UML tool, paying special attention to fill in all documentation fields when possible. If done properly, the content of the tool model is logically equivalent to the document. Make the tool model file available with the document so that all readers can import the classes without misinterpretation.

— Use precise technical names for attributes and operations to avoid confusion. Use documentation fields extensively. Do not reiterate class names inside the attribute names. Keep names short if possible. Keep naming style consistent.

— Do not include redundant interfaces. For example, do not include class attributes that record class associations that are shown in the class diagram; do not include class operations for modifying class associations that are shown in the class diagram.

— All general associations should have role names. The assigned names of associations need not be shown on class diagrams, when at least one role name is shown on that association.

### F.1.5 General interoperability

Interoperability requires consistency. If software modules are to be plug and play, they need to be plug compatible, which means that they have to do the same thing the same way. Even if identical services are provided by logically separate components, those services should be consistent.

All classes referenced in a model should be defined, as should all type-expressions used as attribute types, operation result types and operation input argument types.

Packages should not reinvent semantically equivalent interfaces if similar interfaces exist in other packages. They should instead be imported from these other packages. Diagrams should include visibility tags including package labels (necessary to maintain and document the proper dependency relations between packages). Imported classes should not be modified. An appropriate request for changes to the source (server) document should be filed instead.

When reusing operation signatures from other interfaces for semantically similar or semantically mappable operations, the order of parameters should be maintained. This way, object classes that realize both interfaces in question can use the same implementation method and signature.

Attributes and operations on imported classes and interfaces should not be listed in a client document.

The practice of maintaining full documentation of the same interface in two places makes the client document dependent on changes to those interfaces in the server document. Since an interface represents an abstract service, most minor changes in API protocols should not affect client package design. For imported classes, visibility tags include a "from <package-name>" label in the class name box. Most tools will do this automatically and some will go as far as to shade or colour the imported class. Unmarked classes in a diagram are assumed to be in the package to which the main classes belong.

In the following a separation is made between information modelling and service modelling. Information modelling typically deals with modelling related to persistent information that is handled by a geographical information system, and is typically the approach that will mostly be used to model geographic information. Service modelling deals with the modelling of system component interfaces, and is typically the approach that will mostly be used to model geographic services.

Since the emphasis for information models and service models is slightly different, it has been chosen to separate the modelling approach in a specialization for each of those.

## F.2 Guidelines for information modelling

### F.2.1 General

This is the approach to be taken for most of the modelling that is being done within the ISO geographic information standards and for modelling compliant with ISO geographic information standards. It describes phases for the use of UML. The main principles here are adapted from "Information Modeling: The EXPRESS way"[5]. These are general principles for information modelling, and also apply to other conceptual modelling languages than UML.

The focus here is on describing information models. In object-oriented modelling, one often separates objects/classes into various categories, such as interface/boundary classes that are responsible for system interfaces, service/control classes that are responsible for computational services, and entity/feature classes that represent the persistent model parts of a system. For information modelling, we are mainly concerned with the entity classes, and we therefore talk about entities.

The following modelling phases are recommended:

— Phase 0: Identify scope and context.

— Phase 1: Identify basic classes.

— Phase 2: Specify relationships, attributes and operations.

— Phase 3: Completion of constraints using text/OCL.

— Phase 4: Model definition harmonization – with sub-models and other work items.

The first three phases are of vital importance when creating the schemata compliant with the ISO geographic information standards. Phase 4 is also important when it comes to integrating the different schema into one consistent model.

### F.2.2 Phases of modelling

#### F.2.2.1 Phase 0: Identify scope and context

*Tasks:*

1) Identify the goals and scope of the models.

2) Identify related models and the wider context.

Major question that needs to be answered: What are the goals and scope of the model?

*Deliverable:* A written specification of goals, scope and context of models.

#### F.2.2.2 Phase 1a: Identify basic classes

*Tasks:*

1) Identify generic and specific concepts/terms.

2) Describe each of the classes so that they really differ.

3) Watch out for synonyms and homonyms.

4) Develop a glossary of concepts/terms.

Major questions that need to be answered:

— What are the classes according to the scope?

— What do we know about the attributes of the classes?

— Can we categorize the classes?

— What local consistency constraints apply to the classes?

— Are all the classes and attributes documented?

— Is the use of basic data types correct?

Minor concerns:

— Which, if any, combinations of attribute values uniquely identify a class instance?

— Is the existence of an instance of one class dependent on its usage by another class?

— Are the values of some attributes derivable from other attribute values?

*Deliverable:* A graphical UML class diagram with a written glossary of terms. The terms should be considered for input to the ISO geographic information standards glossary.

#### F.2.2.3 Phase 1b: Consistency with rules for application schema

Is the modelling approach consistent with the approach described in ISO 19109?

*Tasks:*

Check that the modelling approach, granularity of classes, etc. is consistent with the approach described in ISO 19109.

### F.2.2.4 Phase 2: Specify relationships, attributes and operations

Major questions to be asked:

— What are the relationships between the classes?

— Can we categorize the classes?

— Is the use of basic data types correct?

— Are additional attributes required to characterize a class?

— Are the values of some attributes derivable from other attribute values?

— Is the existence of an instance of one class dependent on its usage by another class?

— Which, if any, combinations of attribute values uniquely identify a class instance?

— Are all the classes, relationships and attributes documented?

— Is there a need for operations associated with the classes? If so, what are their intended behaviours, input- and output-arguments, and possible exceptions?

Minor question:

— For a complex model, is it partitioned into topic areas?

*Deliverable:* UML class model

### F.2.2.5 Phase 3: Completion of constraints

Major questions to be asked:

— What are the global consistency rules?

— Are all existence dependencies captured?

— Are all other cardinality constraints captured?

— Are all the constraints captured?

— Is the model well partitioned?

*Deliverable:* Plain text and OCL-based documentation of all constraints.

### F.2.2.6 Phase 4: Model definition harmonization

*Tasks:*

Do harmonization with sub-models and other parts.

When developing a large information model as in the ISO geographic information standards, the work is broken down into separate work items. This is done so that the series of standards can be done in parallel. Thus, there exist a number of modelling groups that develop individual pieces of the ISO geographic information standards, e.g. quality schema, metadata schema, spatial schema, etc. This work arrangement facilitates the rate of progress, but at a cost. The end result is that the different models may not fit together into an integrated information model, if no extra steps are taken to harmonize models.

Major questions to be asked when comparing with other schemas:

— Are there redundancies or conflicts?

— Are there any ambiguities?

— Are the models complete?

— Do the abstraction levels match?

The following checklist has been defined for development of Data/Information Models:

— Have scope and context of the model been defined according to the guidelines?

— Have basic classes in the model been defined according to the guidelines?

— Is the modelling approach consistent with the rules for application schema?

— Has UML been used according to the ISO geographic information standards UML Profile?

— Have relationships, attributes and operations been defined according to the guidelines?

— Have constraints been defined according to the guidelines?

— Have the models been harmonized with other models according to the guidelines?

## F.3 Guidelines for service modelling

### F.3.1 General

This is the approach to be taken for the modelling of geographic services, viewed as computational geographic software components within a distributed architecture. A reference architecture for geographic services is described in ISO 19101-1 and ISO 19119. The reader is also referred to a simple process for specifying components in the book "UML Components"[6]. This process describes how to specify components and services using UML.

The focus in this section is on describing service models. In object-oriented modelling, one often separates objects/classes into various categories, such as interface/boundary classes that are responsible for system interfaces, control/service classes that are responsible for computational services and entity classes that represent the persistent model parts of a system. For service modelling, we are mainly concerned with the service classes, and we therefore talk about service in the following modelling phases. However, most services handle and manipulate information/entity objects, and it is necessary to relate to the definitions made through previous or parallel information modelling.

The following modelling phases are recommended:

— Phase 0: Identify scope and context – use cases.

— Phase 1: Identify basic service responsibilities.

— Phase 2: Specify operations, attributes and service relationships.

— Phase 3: Completion of constraints on operations.

— Phase 4: Service definition harmonization.

### F.3.2 Phases of service modelling

#### F.3.2.1 Phase 0: Identify scope and context – use cases

*Tasks:*

1) Identify the goals of the service.

2) Identify related services and the wider context.

Major questions that need to be answered:

— What are the services/goals according to the scope?

— How are services grouped and associated with elements from the information models?

*Deliverable:* A written specification of the goals, scope and context of the service, as they relate to the scope of the work item. The specification could possibly be supported by use-case scenarios describing the potential use of the service.

#### F.3.2.2 Phase 1: Identify basic service responsibilities

*Tasks:*

1) Identify the basic responsibilities of a service.

2) Describe the basic interaction sequences between a client and the service.

3) Divide the service into logical sub-services/interfaces.

4) Specify collaboration/sequence diagrams.

*Deliverable:* An object model (UML) showing the basic interactions between the service (possibly described as a set of interfaces) and its potential clients. The basic interactions might be supplemented with UML collaboration/sequence diagrams.

#### F.3.2.3 Phase 2: Specify operations, attributes and service relationships

*Tasks:*

For each identified interface, do the following:

— Specify operations with input and output arguments and exceptions.

— Relate the arguments to the specified data/information models.

— Specify abstract attributes (as pair of set/get operations – only get for read-only).

— Specify those events that may be generated by the service.

— For complex services – provide a state diagram for allowed dynamic behaviour.

— Consider if new data/information objects need to be defined.

Minor questions:

— For a complex service, can it be partitioned into sub-services/interfaces areas?

*Deliverable:* Graphical UML class model with operation signatures.

#### F.3.2.4 Phase 3: Completion of constraints on operations

Major questions to be asked for each operation:

— What does this operation mean (explain semantics)?

— What does each parameter mean?

— Are there any constraints or preconditions on parameters?

— What does each user exception mean?

— Are standard exceptions used? What does each mean?

— Are there any other constraints or pre-/post-conditions?

— Are there any sequencing constraints in the use of this operation with other operations?

*Deliverable:* Plain text and OCL-based documentation of all operational constraints.

#### F.3.2.5 Phase 4: Service definition harmonization

Major questions:

— How does the service relate to and interact with other services?

— Are there any ambiguities?

— Are the models complete?

— Do the abstraction levels match?

The following items are a checklist for Service/Process Models:

— Have the service's scope and context been defined according to the guidelines?

— Have basic service responsibilities been defined according to the guidelines?

— Has UML been used according to the UML Profile?

— Have operations, attributes and service relationships been defined according to the guidelines?

— Have constraints on operations been defined according to the guidelines?

— Has the service specification been harmonized with other services according to the guidelines?

*Deliverable:* Revised UML models with updated documentation.

## F.4 Model harmonization

### F.4.1 Guidelines

The main checkpoint for model harmonization is to ensure that the various models are consistent with each other and can be used together in an application schema and in an actual application setting.

The following guidelines should be followed in the context of model harmonization.

1) Cosmetic – Produce a consistently styled output. Do not make changes to the content or structure of the model.

2) Editorial – Apply the principle of "one name and one idea".

3) Continuity – Identify redundancies and gaps between two schemata, remove declarations and add missing declarations.

4) Structural – Identify general underlying concepts that should be used in other parts of the model and rewrite the model.

5) Core based – Define a core information model that can be divided into separate models and developed in parallel. The core model should identify the context and the scope of the different submodels. This is a top-down strategy.

6) Evolutionary based – Start with an existing model and extend this model with a core based view. This is also called a middle-out strategy.

7) Model quality – The integrated model should not change the semantics associated with the different schemata.

### F.4.2 Checklist

The following aspects should be evaluated by each project team for parts that include UML models:

1) *Readability* – is the schema (information model) designed for a human reader?

2) *Scope* – are the defined scopes and assumptions of the different schemata in accordance with the scope of the standard?

3) *The nym principle* – is the principle of one name, one meaning, one definition followed? Be aware of synonyms and homonyms (nym).

4) *Context independence* – are the classes defined as context independent as possible? If the class is applicable for use in more than one context, it should not have defined context dependent constraints.

5) *Implementation independence* – are the models focused on describing what and not how, e.g. not focused on the efficiency of file exchange or implementation?

6) *Invariance* – the meaning of a class should not be dependent on the values of its attributes.

7) *Constraints* – are the value domains of the attributes and relationships appropriately restricted? This can be done by OCL expressions and relationship cardinalities.

8) *Reality* – are the declared classes in correspondence with reality?

9) *Redundancy* – are there redundancies that can lead to possible ambiguities in the model instances?

10) *Concepts* – are the underlying concepts expressed? Surface appearance, syntax descriptions and the use of optional attributes should be avoided.

11) *Hierarchies* – Are inheritance hierarchies used for data aggregation or vice versa?

12) *Basic data types* – are the attribute types correctly specified? Simple types such as integer, real, string, etc. carrying no semantics should be avoided.

13) *Deliverable:* Revised UML models.

# Annex G
(informative)

# Introduction to UML

## G.1 Introduction

Annex G is provided as a convenient introduction to the basic parts of UML that are used within and prescribed by the ISO geographic information standards. The normative document for use of UML is UML 2.

## G.2 General usage of UML

UML should be used in a manner that is consistent with UML 2. Books, such as "UML User Guide"[1] and "UML Reference Manual"[2] contain further information. The book "UML Distilled"[4] is a shorter introductory text.

## G.3 Classes

A class is a description of a set of objects that share the same attributes, operations, methods, relationships and semantics. A class represents a concept being modelled. Depending on the kind of model, the concept may be based on the real world (for a conceptual model), or it may be based upon implementation of either platform independent system concepts (for specification models) or platform specific system concepts (for implementation models).

A classifier is a generalization of a class that includes other class like elements, such as data types, actors and components. A UML class has a name, a set of attributes, a set of operations and constraints. A class may participate in associations.

An abstract class defines a polymorphic object class and cannot be instantiated. An Abstract class is specified by having the class name in italics, or alternatively, the keyword abstract may be placed in a property list below or after the class name.

The visibility of an attribute, operation, or association end defined in the context of a class may be limited in various ways. Figure G.1 shows symbols for derived element, visibility and class level definitions.

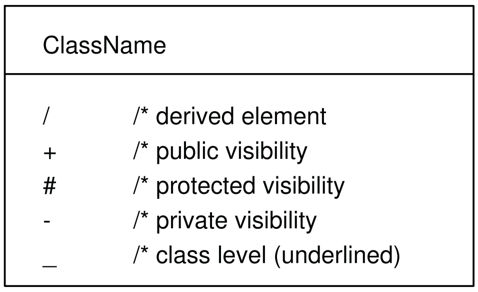


**Figure G.1 — Symbols for derived element, visibility and class level definitions**

The visibility of attributes, operations and association ends is shown by a symbol preceding the element name (role name in the case of an association end), as follows:

— - private, a private element is only visible inside the namespace that owns it.

— # protected, a protected element is visible to elements that have a generalization relationship to the namespace that owns it.

— + public, a public element is visible to all elements that can access the contents of the namespace that owns it.

— ~ package, a package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace. Only named elements that are not owned by packages can be marked as having package visibility. Any element marked as having package visibility is visible to all elements within the nearest enclosing package (given that other owning elements have proper visibility). Outside the nearest enclosing package, an element marked as having package visibility is not visible.

It is possible to define an attribute or operation to be at the class level (as opposed to being at the instance level) by underlining the attribute or operation definition.

A slash '/' preceding the name of a model element such as an attribute means that it is derived (calculated) from another element.

## G.4 Attributes

The following general notation for attributes is defined in UML 2.

<property> ::= [<visibility>] ['/'] <name> [':' <prop-type>] ['[' <multiplicity> ']'] ['='<default>] ['{' <prop-modifier> [',' <prop-modifier> ]* '}']

where:

• <visibility> is the visibility of the property.

<visibility> ::= '+' | '-' | '#' | '~'

•'/' signifies that the property is derived.

• <name> is the name of the property.

• <prop-type> is the name of the type of the property.

• <multiplicity> is the multiplicity of the property. If this term is omitted, it implies a multiplicity of 1 (exactly one).

• <default> is an expression that evaluates to the default value or values of the property.

• <prop-modifier> indicates a modifier that applies to the property.

<prop-modifier> ::= 'readOnly' | 'union' | 'subsets' <property-name> |

'redefines' <property-name> | 'ordered' | 'unique' | 'nonunique' | <prop-constraint>

where:

- readOnly means that the property is read only.
- union means that the property is a derived union of its subsets.
- subsets <property-name> means that the property is a proper subset

  of the property identified by <property-name>.
- redefines <property-name> means that the property redefines

  an inherited property identified by <property-name>.

- ordered means that the property is ordered.
- unique means that there are no duplicates in a multi-valued property.
- <prop-constraint> is an expression that specifies a constraint that applies to the property.

All redefinitions should be made explicit with the use of a {redefines <x> } property string. Matching features in subclasses without an explicit redefinition result in a redefinition that need not be shown in the notation. Redefinition prevents inheritance of a redefined element into the redefinition context thereby making the name of the redefined element available for reuse, either for the redefining element, or for some other.

EXAMPLE 1 + coordinates : Number [1..*] {sequence}

EXAMPLE 2 + origin : Point [0..1] / * multiplicity 0..1 means that this is optional.

Properties can be user defined. An attribute should be unique within the context of a class and its supertypes, unless it is an attribute redefined from a supertype. If an attribute is overridden, it should be treated as an abstract attribute that can be computed on demand according to the rules defined for its derivation.

A type for the attribute must always be specified; there is no default type.

If no explicit multiplicity is documented, the multiplicity is assumed to be 1.

An attribute may define a default value, which is used when an object of that data type is created. Default values are defined by explicit default values in the UML definition of the attribute. Default values are a necessary mechanism in some situations in conceptual models. As an example, ISO 19115-1, MD_DataIdentification class defaults the character set to ISO/IEC 10646.

EXAMPLE 3 + characterSet : MD_CharacterSetCode [0..*] = "utf8"

A UML profile may specify stereotypes applicable for attributes to extend them with additional semantics.

## G.5 Enumerations and code lists

### G.5.1 General

An enumerated type is a data type whose values are enumerated in the model as literals. 6.5 elaborates how enumerated types are used in this UML profile. Two variations of enumerated types are presented, enumerations and code lists.

Two primary types of codelists:

1) Where the codelist is defined or referenced and extended at runtime; extended by a user of the schema;

2) Where the codelist is a closely controlled/standardized vocabulary managed by an authority. If it is extended it is done so in an official published profile with an established registry and follows established rules for the extension.

### G.5.2 Guidelines for establishing codelists

1) Since a codelist is a controlled vocabulary that is meant to address a specific concept each codelist represents a separate subject/concept and each code should represent an unambiguous separate non-overlapping value (see G.5.3, Note) within that domain. As an example ISO 19115-1 has separate codelists for legal restrictions and security classification restrictions – even though these are fairly close in concept it is much better to not mix these concepts in one list as they have similar items with slightly different meanings. The antonymous alternative would be to combine all codelists into one!

2) Each item in the codelist contains a unique value (name or code) that all software will use to recognize the unique concept or option defined in a row of a codelists table. While numbers may be used, simple name identifier values are encouraged to enable human understanding of the code; this value need not be a proper name in any language;

3) A proper or concept name is provided as a label expressed in a given locale (language, country and character set) of the specific information community using the codelist.

4) Each code should have a unique unambiguous definition and describe a unique concept or option that supports the intent of the definition;

5) Codelists and their associated definitions are controlled in registers. For example ISO 19115-1:2014, Annex B is the base concept register for the ISO 19115-1 Codelists and is the base source for the establishment of user communities' registers.

### G.5.3 Guidelines for extending codelists

1) The codes defined in a standard are the only codes that can be interpreted by an application independently of any profile, i.e. each time a community decides to extend a codelist, it has to support the consequence of using a specific code (and accept the risk in terms of interoperability).

2) The extended code must not be used to replace an existing code by changing the name or definition.

3) Extended code lists follows the best practices for establishing codelists.

4) Type 2 codelists uses the existing codelist and merely add additional unique code/items.

NOTE Overlapping codes can be used in certain instances where both types and sub types are included, and users can provide a more general or a more specific code. Example, "geocentric", "geographic2D" and "geographic3D" are more specific types of "geodetic" reference systems:

**Table G.1 — Hierarchical codelist example**

| | Concept name (English) | Code | Definition |
|---|---|---|---|
| 1. | MD_ReferenceSystemTypeCode | | defines type of reference system used |
| 2. | geodetic | geodetic | coordinate reference system based on a geodetic datum (datum describing the relationship of a two- or three-dimensional coordinate system to the Earth) |
| 3. | geocentric | geocentric | geodetic CRS having a Cartesian 3D coordinate system<br>e.g. [geocentric] X,Y,Z |
| 4. | geographic3D | geographic3D | geodetic CRS having an ellipsoidal 3D coordinate system<br>e.g. latitude, longitude, ellipsoidal height |
| 5. | geographic2D | geographic2D | geodetic CRS having an ellipsoidal 2D coordinate system<br>e.g. latitude, longitude |
| 6. | projected | projected | coordinate reference system derived from a two-dimensional geodetic coordinate reference system by applying a map projection<br>e.g. easting, northing |
| 7. | vertical | vertical | one-dimensional coordinate reference system based on a vertical datum (datum describing the relation of gravity-related heights or depths to the Earth)<br>e.g. [gravity-related] height or depth |

**Table G.1** *(continued)*

| | Concept name (English) | Code | Definition |
|---|---|---|---|
| 8. | engineering | engineering | coordinate reference system based on an engineering datum (datum describing the relationship of a coordinate system to a local reference)<br><br>e.g. [local] x,y |
| 9. | image | image | coordinate reference system based on an image datum (engineering datum which defines the relationship of a coordinate system to an image)<br><br>e.g. row, column |
| 10. | design | design | engineering coordinate reference system in which the base representation of a moving object is specified<br><br>e.g. [local] x,y |

## G.6 Data types

The UML itself specifies only a limited set of primitive types. Clause 7 defines a set of core data types available in this profile, while Annex C defines a set of types that are extensions to the core types.

## G.7 Operations

An operation is specified by a text string that can be parsed into elements that describe the properties of the operation:

[<visibility>] <name> '(' [<parameter-list>] ')' [':' [<return-type>] ['{' <oper-property> [',' <oper-property>]* '}']]

where:

• <visibility> is the visibility of the operation (See G.3)

<visibility> ::= '+' | '-' | '#' | '~'

• <name> is the name of the operation.

• <return-type> is the type of the return result parameter if the operation has one defined.

• <oper-property> indicates the properties of the operation.

<oper-property> ::= 'redefines' <oper-name> | 'query' | 'ordered' | 'unique' | <oper-constraint>

where:

- redefines <oper-name> means that the operation redefines

  an inherited operation identified by <oper-name>
- query means that the operation does not change the state of the system.
- ordered means that the values of the return parameter are ordered.
- unique means that the values returned by the parameter have no duplicates.
- <oper-constraint> is a constraint that applies to the operation.
- <parameter-list> is a list of parameters of the operation in the following format:

  <parameter-list> ::= <parameter> [',' <parameter> ]*

<parameter> ::= [<direction> ] <parameter-name> ‘:’ <type-expression>

[‘[‘ <multiplicity> ’]’] [‘ = ’ <default> ] [‘{‘ <parm-property> [‘,’ <parm-property> ]* ‘}’]

where:

- <direction> ::= ‘in’ | ‘out’ | ‘inout’ (defaults to ‘in’ if omitted).
- <parameter-name> is the name of the parameter.
- <type-expression> is an expression that specifies the type of the parameter.
- <multiplicity> is the multiplicity of the parameter.
- <default> is an expression that defines the value specification for the default value of the parameter.
- <parm-property> indicates additional property values that apply to the parameter.

UML has defined four standard properties for operations: isQuery, sequential, guarded, concurrent.

There are several different categories of objects involved in an operation:

1) The object with which the operation is associated, referred to as “this” or “self” in most languages [the object before the dot “.” in a statement like object.operation()].

2) The objects passed as parameters that are used to control the behaviour of the operation.

3) The objects modified or returned by the operation.

In UML, objects are normally modified or accessed by their own methods. However, objects can be “passed by reference” so that any persistent object passed as a parameter may receive a cascading message. The object can therefore be modified, albeit indirectly, by any method to which it is passed. Thus object valued parameters are inherently “inout” in direction. Parameters that take data types as their values are either “in” or “out”.

Scope (class versus instance) and the value of the isQuery property (Boolean) should be documented in the text that describes the operation.

## G.8 Relationships

### G.8.1 General

A relationship in UML is a reified semantic connection among model elements. Kinds of relationships include association, aggregation/composition, generalization, realization and dependency, see Figure G.2. In Reference [2] a clear distinction is made between the general term “relationship”, and the more specific term “association”. Both are defined for class to class linkages, but association is reserved for those relationships that are in reality instance to instance linkages. Generalization, realization and dependency are class to class relationships. Aggregation and other object to object relationships, are more restrictively called “associations”. It is always appropriate to use the most restrictive term in any case, so in speaking of instantiable relationships, use the term “association”.

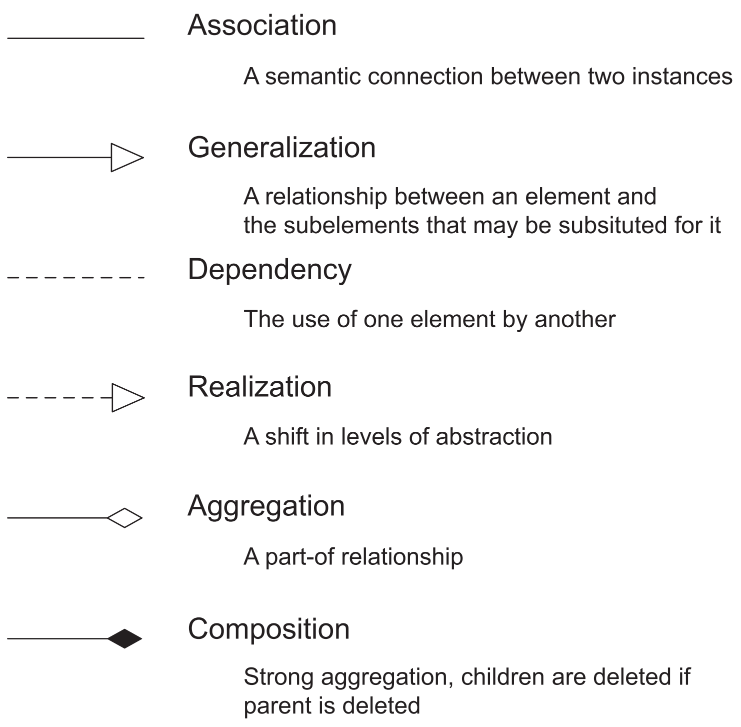

Association
A semantic connection between two instances
Generalization
A relationship between an element and the subelements that may be subsituted for it
Dependency
The use of one element by another
Realization
A shift in levels of abstraction
Aggregation
A part-of relationship
Composition
Strong aggregation, children are deleted if parent is deleted

**Figure G.2 — Different kinds of relationships**

In this UML profile dependency is used according to the standard UML notation and usage. In the following, the usage of association, aggregation, composition, generalization and realization is described further.

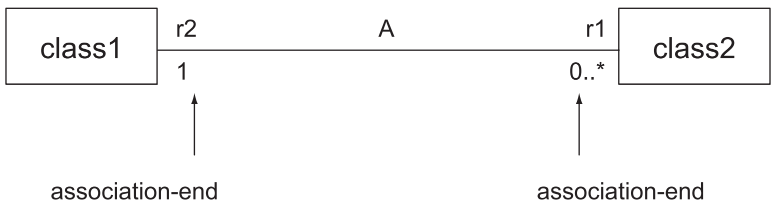
## G.8.2 Association, composition and aggregation

An association in UML is the semantic relationship between two or more classifiers (i.e. class, interface, type) that involves connections among their instances.

An association is used to describe a relationship between two or more classes. In addition to an ordinary association, UML defines two special types of associations called aggregation and composition. The three types have different semantics. An ordinary association should be used to represent a general relationship between two classes. The aggregation and composition associations should be used to create part whole relationships between two classes.

A binary association has a name and two association ends. An association end has a role name, a multiplicity statement and an optional aggregation symbol. An association end should always be connected to a class.

Figure G.3 shows an association named “A” with its two respective association-ends. The role name is used to identify the end of an association; the role name **r1** identifies the association-end which is connected to the class named class2. In this example, the role name **r1** identifies the nature of the relation **class2** has to **class1** through the association **A**. The multiplicity of an association-end can be one of exactly-one (1), zero-or-one (0..1), one-or-more (1..*), zero-or-more (0..*), an interval (n..m) or a set of given numbers (m, n, o, p). Viewed from the class, the role name of the opposite association-end identifies the role of the target class. We say that **class2** has an association to **class1** that is identified by the role **r2** and which has a multiplicity of exactly one. The other way around, we can say that **class1** has an association to **class2** that is identified by the role name **r1** with multiplicity of zero-or-more. In the instance model we say that **class1** objects have a reference to zero-or-more **class2** objects and that **class2** objects have a reference to exactly one **class1** object.



**Key**

| | |
|---|---|
| A | association |
| r1,r2 | role names |

**Figure G.3 — Association**

Figure G.4 specifies the notation that is used to specify the number of instances that can participate at one end of an association. The same notation is used for attributes, but then in the field [multiplicity], as discussed in G.4. If multiplicity is omitted at an association end the multiplicity value is not assumed to have any special value, and this will violate requirement 10.

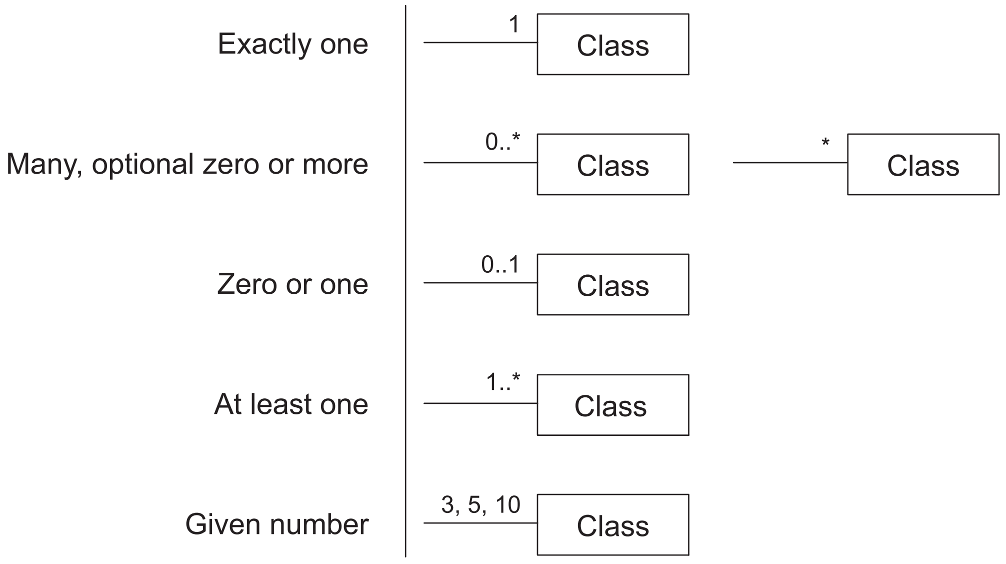
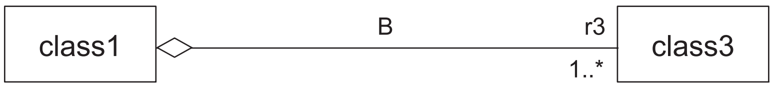


**Figure G.4 — Specification of multiplicity/cardinality**

## G.8.3 Aggregation

An aggregation association is a relationship between two classes, in which one of the classes plays the role of container and the other plays the role of a containee. Figure G.5 shows an example of an aggregation. The open diamond-shaped aggregation symbol at the association end close to **class1** indicates that **class1** is an aggregation consisting of **class3**. The meaning of this is that **class3** is a part of **class1**. In the instance model, **class1** objects will contain one or more **class3** objects. The aggregation association should be used when the containee objects, which represent the parts of a container object, can exist without the container object. Aggregation is a symbolic short-form for the part-of association but does not have explicit semantics. It allows multiple aggregations to share the same objects. If a stronger aggregation semantics is required, composition should be used as described below. It is possible also to define role name and multiplicity at the diamond shaped end as well.



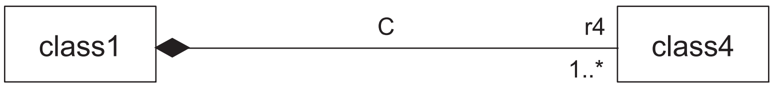
| | |
|---|---|
| **Key** | |
| B | association |
| r3 | role names |

**Figure G.5 — Aggregation**

### G.8.4 Composition

A composition association is a strong aggregation. In a composition association, if a container object is deleted then all of its containee objects are deleted as well. The composition association should be used when the objects representing the parts of a container object cannot exist without the container object. Figure G.6 shows a composition association, in which the diamond shaped composition symbol has a solid fill. Here **class1** objects consist of one or more **class4** objects, and the **class4** objects cannot exist unless the **class1** object also exists. The required (implied) cardinality for the owner class is always one. The containers, or parts, cannot be shared among multiple owners.

It is possible also to define role name at the diamond shaped end as well; the multiplicity will always be at most one. Composition should be used to have the semantic effect of containment and should be used with care. The application of the composition construct should be considered within the context of a model (rather than the scope), where context means the application domain within which the application must be consistent. This is in order to prevent problems where different applications have different requirements for composition.



| | |
|---|---|
| Key | |
| C | association |
| r4 | role names |

**Figure G.6 — Composition (strong aggregation)**

Every UML association has navigability attributes that indicate which class in the association can be followed to get information from the other end. The default logic for an unmarked association is that it is two-way. In the case of client-server relations, the association used is normally only navigable from client to server (one-way), indicated in the diagram by an arrowhead on the server side of the association. Associations that do not indicate navigability are two-way in that both participants have equal access to the opposite role. Two-way navigation is not common or necessary in many client-to-server operations, so the arrowhead has been introduced to be a hint for a more optimised implementation. The counterexample to this may be notification services, where the server often instigates communication on a prescribed event. The use of two-way relations that introduce unreasonable package dependencies should be minimized. One-way relations should be used when that is all that is needed.

Multiplicity refers to the number of relationships of a particular kind that an object can be involved in. If the target class of a one way relation is marked with a multiplicity, then this indicates to any implementation how many storage locations may be required by the source class of the relation. In most object languages, this would be implemented as an array of object references. The multiplicity indicates the constraints on the length of that array and the number of NULLs allowed. If an association end were not navigable, putting a multiplicity constraint on it would require an implementation to keep track of the use of the association by other objects (or to be able to acquire the multiplicity through query). If this is important to the model, the association should be two way navigable to make enforcement of the constraint more tenable. In other words, a one way relation implies a certain “don’t care” attitude towards the non navigable end.

### G.8.5 Generalization

UML 2 defines a generalization as a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier. For types, we often speak of supertype and subtype. An example of a generalization relationship is shown in Figure G.7.

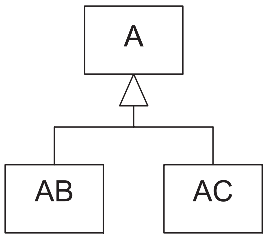


**Figure G.7 — Example of generalization relationship**

Class AB and AC are specializations of the generalized class A. All properties of A are inherited by AB and AC.

### G.8.6 Realization

A realization describes the relationship between a specification and its implementation; an indication of the inheritance of behaviour without the inheritance of structure [UML 2]. Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client). The meaning is that the client element must support all the behaviour of the supplier element but need not to match its structure or implementation.

The realization relationship is shown by a dashed path with a closed triangular arrowhead towards the specification element from the implementation element as shown in Figure G.8.

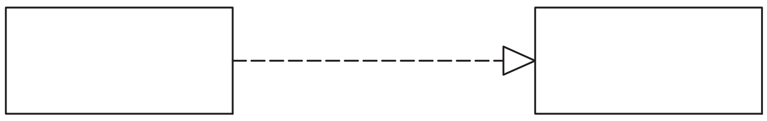


**Figure G.8 — UML notation for Realization**

## G.9 Services

Services are modelled according to general UML as described in UML 2.

## G.10 Stereotypes and tagged values

### G.10.1 General

Stereotypes and tagged values extend existing UML model elements to adapt them for different purposes. In earlier versions of UML stereotypes and tagged values could be attached to every UML element in a flexible way, but in UML 2 they are defined and used within a profile. Stereotypes are specific metaclasses and tagged values are in this respect metaattributes.

### G.10.2 Stereotypes

A stereotype is a new kind of model element defined within a profile based on an existing kind of model element. It is a model element that is used to classify (or mark) other UML elements so that they in some respect behave as if they were instances of new virtual or pseudo metamodel classes whose form is based on existing base metamodel classes. Names of new stereotypes must not have case-insensitive same name as predefined keywords, metamodel elements or other stereotypes. A stereotype may have properties just like a class and values of the properties for an applied stereotype can be referred to as tagged values.

UML defines a set of predefined stereotypes. This UML profile specifies some additional stereotypes. Stereotypes used by and defined in this International Standard are specified in 6.10.2.

More than one stereotype can be applied to a single model element. Any UML model element can be extended by a stereotype, for example package, class, association and attribute.

### G.10.3 Tagged values

A tagged value is a tag value pair that can be used to add properties to model elements in UML. In UML 2 tagged values can only be applied to model elements that use a stereotype with a tag definition. Tagged values are shown in the form tag = value where tag is the tag name and value is a literal value. Tagged values are especially relevant for code generation or configuration management.

EXAMPLE A tagged value attached to a classifier can be defined as follows: {author = “Joe Smith”, deadline = 31 March 1997, status = analysis}.

## G.11 Optional, conditional and mandatory attributes and association ends

Standard UML gives various ways to describe optional, conditional and mandatory attributes and association ends. This International Standard specifies in more detail one way to do this in 6.11.

## G.12 Naming and namespaces

UML does not specify any strict rules for naming and namespaces. This is elaborated more in 6.12.

## G.13 Packages

A UML package is a container that is used to group declarations of subpackages, classes and their associations. The package structure in UML enables a hierarchical structure of subpackages, class declarations and associations.

The packages, classes and attributes in a model can be identified by a qualified name or its innermost name if the context is given. The form of the qualified names is *packagename1::package:name2::classname.elementname1.elementname2*, where *packagename1* is the name of the outermost package, *packagename2* is a name which appears within the namespace of *packagename1*, *classname* is the name of a class that appears within the namespace of *packagename2* and *elementname1* is the name of an element within the namespace of *classname*. The:: symbol is used to separate package names from each other and from class names; the period (.) is used to separate class names from the names of elements within the namespace of a class. There is no limit to the depth of this namespace hierarchy.

EXAMPLE In the Spatial schema there is a subpackage named Geometry which defines a class named GM_Object. This class has an association with role name SRS (Spatial Reference System). The fully qualified name for this association is: Spatial.Geometry::GM_Object.SRS.

## G.14 Notes

Note boxes are used to comment on the model in general or specific item (i.e. class or association) of the model in particular, see Figure G.9. They can also be used for specifying constraints and conditions.

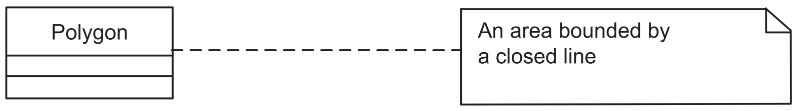


**Figure G.9 — Example note**

## G.15 Constraints

Constraints can be described using OCL 2. OCL is a set based language based on logical expressions of objects and object properties. An introductory text to OCL can be found in Reference [3].

## G.16 Documentation of models

UML 2 has no specific rules for documentation. 6.16 describes how models conforming to the profile defined in this International Standard shall be documented.

# Annex H
(informative)

# Backwards compatibility

## H.1 Introduction

Annex H is provided as a set of descriptions of changes from older concepts onto new improved concepts.

## H.2 Scope

Added standardization target type missing in previous version.

## H.3 Conformance

Added precise conformance classes missing in previous version.

## H.4 Normative references

Added references to ISO 639, ISO 3166, ISO 8601, ISO/IEC 11404, ISO/IEC 19505-2, RFC 3986 etc.

## H.5 Terminology

Added and revised several terms.

## H.6 UML Profile

Added much improved focus on requirements and recommendations.

Moved description of types to chapter 7 and to Annex C.

Added description of stereotypes and keywords <<dataType>>, <<CodeList>>, <<enumeration>>, <<interface>>.

## H.7 Core data types

Removed types UnlimitedInteger, Directory, Probability, Logical, Multiplicity, MultiplicityRange.

Added description of collections by using UML 2 multiplicity.

## H.8 Formal UML Profile

Added formal UML model describing the stereotypes in Annex D.

## H.9 Extension data types

Added MultilingualText and some web environment types.

Added some new subtypes of Measure and UnitOfMeasure.

Annex C is new and is provided to define Measure and the new conceptual models for cultural and linguistic adaptability and web environment. Some of these constructs are implemented and in use in existing encoding standards, though under a more restrictive scope. These constructs may in the future refer to the concepts in C, and describe their scope.

## H.10 Mapping from UML 1 to UML 2

Rules added to Annex B.

## H.11 Guidelines

Informative Annexes E, F and G revised.

## H.12 Backwards compatibility

Added summary of main changes in Annex H.

## H.13 Former assumptions

In earlier models a lack of explicit multiplicity on association ends may have been interpreted as being fixed to [1] or [0..*]. This is no longer allowed, and explicit multiplicity must be provided.

# Bibliography

[1] Booch G., Jacobsson I., Rumbaugh J. Unified Modeling Language User Guide. Addison-Wesley, Second Edition, 2005

[2] Rumbaugh J., Booch G., Jacobsson I. Unified Modeling Language Reference Manual. Addison-Wesley, Second Edition, 2004

[3] Warmer J.B., & Kleppe A.G. The Object Constraint Language — precise modeling with UML. Addison-Wesley Longman, Inc, 1997

[4] Fowler M. UML Distilled, A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, Third Edition, 2003

[5] Schenck D., & Wilson P. Information Modeling: The EXPRESS Way. Oxford University Press, 1993

[6] Cheesman J., & Daniles J. UML Components — A simple process for specifying component-based software. Addison-Wesley, 2000

[7] Meta Object Facility (MOF) Core Specification. version 2.0, Available at, http://www.omg.org/spec/MOF/2.0

[8] ISO 4217, *Codes for the representation of currencies*

[9] ISO 19101-1:2014, *Geographic information — Reference model — Part 1: Fundamentals*

[10] ISO 19107:2003, *Geographic information — Spatial schema*

[11] ISO 19108:2002, *Geographic information — Temporal schema*

[12] ISO 19109:2015, *Geographic information — Rules for application schema*

[13] ISO 19115-1:2014, *Geographic information — Metadata — Part 1: Fundamentals*

[14] ISO 19118:2011, *Geographic information — Encoding*

[15] ISO 19119:2005, *Geographic information — Services*

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

[17] ISO/TR 9007:1987, *Information processing systems — Concepts and terminology for the conceptual schema and the information base*

**ICS 35.240.70**

Price based on 81 pages