International Standard

ISO 19103

Second edition
2024-09

Geographic information — Conceptual schema language

Information géographique — Langage de schéma conceptuel

Reference number
ISO 19103:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

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 document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

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

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics,* in collaboration with the European Committee for Standardization (CEN) Technical Committee CEN/TC 287, *Geographic Information,* in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement).

This second edition cancels and replaces the first edition (ISO 19103:2015), which has been technically revised.

The main changes are as follows:

— conformance to UML 2.5.1 has been improved;

  — the UML profile has been improved and the stereotypes Leaf, CodeList and Union have been deprecated;

  — the collection data types, the name data types, the extension data types and data type Any have been removed;

— alignment with the data types described in ISO/IEC 11404:2007, Clause 8 and Clause 10 has been improved;

— the conformance classes for conceptual schemas modelled in UML 1.x and for conceptual schemas modelled in another conceptual schema language have been removed;

— the normative references have been updated, in particular:

  — addition of UML 2.5.1 and removal of ISO/IEC 19505-2:2012 (equivalent to UML 2.4.1, Superstructure[4]);

  — removal of the Object Constraint Language (OCL) specification.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document 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. An important goal of the ISO 19100 family of documents 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 document. First, a CSL is selected that meets the requirements for rigorous representation of geographic information. Several CSLs exist, of which two predominate in the geographic domain: the Unified Modeling Language (UML), specified by the Object Management Group (OMG), on the one hand, and the combination of the three Semantic Web specifications, the Resource Description Framework Schema (RDFS), the Web Ontology Language (OWL) and the Shapes Constraint Language (SHACL), specified by the World Wide Web Consortium (W3C), on the other hand. It was decided to continue using UML as it has proven its capability within the ISO 19100 family of documents, it supports a model-driven approach and it has a standardized graphical notation. This document identifies a subset of UML as the CSL for the specification of conceptual schemas. It also specifies a UML profile for the specification of conceptual schemas, and it specifies provisions on how to use UML and the UML profile to create conceptual schemas that are a basis for achieving the goal of interoperability. In addition, this document defines a set of core data type definitions for use in conceptual schemas.

One goal of the ISO 19100 family of documents using conceptual schemas specified in UML is that they will provide a basis for model-based mapping to encoding schemas like those defined in ISO 19118, as well as a basis for creating implementation specifications for implementation profiles for various other environments.

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

In accordance with the ISO/IEC Directives, Part 2, 2021, *Principles and rules for the structure and drafting of ISO and IEC documents*, in International Standards the decimal sign is a comma on the line. However, the General Conference on Weights and Measures (*Conférence Générale des Poids et Mesures*) at its meeting in 2003 passed unanimously the following resolution: "The decimal marker shall be either a point on the line or a comma on the line."[5] In practice, the choice between these alternatives depends on customary use in the language concerned. In the technical areas of geodesy and geographic information it is customary for the decimal point always to be used, for all languages. That practice is used throughout this document.

The name and contact information of the maintenance agency for this document can be found at www.iso.org/maintenance_agencies.

Geographic information — Conceptual schema language

1 Scope

This document specifies provisions for the use of a conceptual schema language within the context of modelling geographic information. The chosen conceptual schema language is a subset of the Unified Modeling Language (UML).

This document specifies a UML profile for modelling geographic information.

This document specifies a set of core data types for use in conceptual schemas.

The standardization target type of this document is conceptual schemas describing geographic information.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

UML 2.5.1: OBJECT MANAGEMENT GROUP (OMG). *Unified Modeling Language (UML)* [online]. Version 2.5.1. December 2017. Available at: https://www.omg.org/spec/UML/2.5.1

3 Terms and definitions

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

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
abstract
<information technology> filter out detail that is not within the scope of interest

Note 1 to entry: Abstracting facilitates the understanding of the essence of a *concept* (3.20) and allows for handling complexity.

Note 2 to entry: An act of abstracting is designated as an "abstraction". In the information technology domain, the term "abstraction" also represents concept *abstraction* (3.4).

3.2
abstract classifier
<UML> *classifier* (3.16) that has no direct *instances* (3.42)

Note 1 to entry: UML 2.5.1, 9.2.3.2 requires that every instance of an abstract classifier is an instance of one of its specializations.

Note 2 to entry: Adapted from UML 2.5.1, 9.2.3.2.

3.3
abstract schema
conceptual schema (3.23) that is not implementable without further specification

EXAMPLE The conceptual schemas for describing the spatial characteristics of geographic entities defined in ISO 19107:2019.

Note 1 to entry: An abstract schema can be applied to many domains.

Note 2 to entry: An abstract schema can be realized by an *application schema* (3.8).

3.4
abstraction
<information technology> result of an act of *abstracting* (3.1)

3.5
abstraction
<UML> *dependency* (3.30) that relates two *named elements* (3.50) or sets of named elements that represent the same *concept* (3.20) at different *levels of abstraction* (3.45) or from different viewpoints

Note 1 to entry: Adapted from UML 2.5.1, 7.7.3.3.

3.6
aggregation
shared aggregation
<UML> *binary association* (3.12) that specifies a *part-whole relation* (3.58) where the whole does not have responsibility for the existence of its parts

Note 1 to entry: A part can be included in more than one whole simultaneously.

Note 2 to entry: Adapted from UML 2.5.1, 9.5.3.

3.7
application
manipulation and processing of data in support of user requirements

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

3.8
application schema
conceptual schema (3.23) for data required by one or more *applications* (3.7)

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

3.9
association
<UML> semantic *relationship* (3.63) that can occur between *instances* (3.42) that have a *type* (3.70)

Note 1 to entry: An association is also a kind of *classifier* (3.16).

Note 2 to entry: Adapted from UML 2.5.1, 11.5.3.1.

3.10
attribute
<UML> *property* (3.61) owned by a *classifier* (3.16) other than an *association* (3.9)

Note 1 to entry: Adapted from UML 2.5.1, 9.5.3.

3.11
behavioural feature
<UML> *feature* (3.36) that specifies an aspect of behaviour

Note 1 to entry: Adapted from UML 2.5.1, 9.9.2.1.

3.12
binary association
<UML> *association* (3.9) having two member ends

Note 1 to entry: UML 2.5.1, 11.5.3.1 permits that the two member ends have the same *type* (3.70).

Note 2 to entry: Adapted from UML 2.5.1, 11.5.3.1.

3.13
cardinality
<UML> number of values

EXAMPLE The cardinality of a collection having three values is three.

Note 1 to entry: Cardinality is a characteristic of a collection.

Note 2 to entry: Adapted from UML 2.5.1, 7.5.3.2.

3.14
class
<UML> *classifier* (3.16) of a set of *objects* (3.54)

Note 1 to entry: Adapted from UML 2.5.1, 11.8.3.1.

3.15
class diagram
<UML> structure diagram where the primary symbols in the contents area are either *class* (3.14) symbols or *interface* (3.43) symbols, or both

Note 1 to entry: Adapted from UML 2.5.1, Annex A.

3.16
classifier
<UML> classification of *instances* (3.42) according to their *features* (3.36)

Note 1 to entry: A classifier is a kind of *type* (3.70).

Note 2 to entry: Adapted from UML 2.5.1, 9.2.1.

3.17
code set
code element set
code
code list
result of applying a coding scheme to all elements of a coded set

EXAMPLE The three-letter representations of airport names.

Note 1 to entry: The term "code" also represents the concept defined in ISO 19118:2011, 4.3.

[SOURCE: ISO/IEC 2382:2015, 2121556, modified — An additional admitted term "code list" has been added, the definition has been adjusted to use the terms used in Annex H, Note 1 to entry has been converted into an Example, Notes 2 and 3 to entry have been removed and a new Note 1 to entry has been added.]

3.18
comment
note
<UML> textual annotation that can be attached to a set of elements

Note 1 to entry: Adapted from UML 2.5.1, 7.8.2.1.

3.19
composition
<UML> *binary association* (3.12) that specifies a *part-whole relation* (3.58) where the whole has responsibility for the existence of its parts

Note 1 to entry: A part can only be included in at most one whole at a time.

Note 2 to entry: Adapted from UML 2.5.1, 9.5.3.

3.20
concept
unit of knowledge created by a unique combination of characteristics

Note 1 to entry: Concepts are not necessarily bound to particular *natural languages* (3.52). They are, however, influenced by social or cultural background, which often leads to different categorizations.

Note 2 to entry: This is the concept "concept" as used and designated by the term "concept" in terminology work. It is a very different concept from that designated by other domains such as industrial automation or marketing.

[SOURCE: ISO 1087:2019, 3.2.7]

3.21
conceptual formalism
set of modelling *concepts* (3.20) used to describe a *conceptual model* (3.22)

EXAMPLE Object-oriented modelling.

Note 1 to entry: One conceptual formalism can be expressed in several *conceptual schema languages* (3.24).

[SOURCE: ISO 19101-1:2014, 4.1.4, modified — Examples have been replaced.]

3.22
conceptual model
model (3.48) that defines *concepts* (3.20) of a *universe of discourse* (3.72)

Note 1 to entry: A model can include relations between concepts. A relation is a concept too.

[SOURCE: ISO 19101-1:2014, 4.1.5, modified — Note to entry added.]

3.23
conceptual schema
formal description of a *conceptual model* (3.22)

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

3.24
conceptual schema language
formal language (3.37) based on a *conceptual formalism* (3.21) for the purpose of representing *conceptual schemas* (3.23)

EXAMPLE UML, EXPRESS, IDEF1X.

Note 1 to entry: A conceptual schema language can be lexical or graphical. Several conceptual schema languages can be based on the same conceptual formalism.

[SOURCE: ISO 19101-1:2014, 4.1.7]

3.25
constraint
<UML> condition or restriction expressed in *natural language* (3.52) text or in a machine readable language for the purpose of declaring some of the semantics of an element or set of elements

Note 1 to entry: Adapted from UML 2.5.1, 7.8.3.1.

3.26
data type
set of distinct values, characterized by properties of those values, and by *operations* (3.55) on those values

EXAMPLE The data type "Boolean" with properties "unordered", "exact" and "non-numeric", and with operations "equal", "not", "and" and "or".

Note 1 to entry: Properties of data type values are ordered or unordered, exact or approximate, numeric or non-numeric and, if ordered, bounded or unbounded, as described in ISO/IEC 11404.

[SOURCE: ISO/IEC 11404:2007, 3.12, modified — Note to entry and example added.]

3.27
data type
<UML> *classifier* (3.16) whose *instances* (3.42) are distinguished only by their value

Note 1 to entry: Adapted from UML 2.5.1, 10.2.1.

3.28
data value
<UML> *instance* (3.42) of a *data type* (3.27)

Note 1 to entry: Adapted from UML 2.5.1, 7.5.3.2.

3.29
definition
representation of a *concept* (3.20) by an expression that describes it and differentiates it from related concepts

[SOURCE: ISO 1087:2019, 3.3.1]

3.30
dependency
<UML> *directed relationship* (3.32) which signifies that a single model element or a set of model elements requires other model elements for their specification or implementation

Note 1 to entry: A dependency signifies a supplier/client *relationship* (3.63) between model elements where the modification of a supplier can impact the client model elements. The complete semantics of the client element(s) are either semantically or structurally dependent on the *definition* (3.29) of the supplier element(s).

Note 2 to entry: Adapted from UML 2.5.1, 7.7.1 and 7.8.4.

3.31
designation
designator
label
representation of a *concept* (3.20) by a sign which denotes it in a domain or subject

Note 1 to entry: A designation can be linguistic or non-linguistic. It can consist of various types of characters, but also punctuation marks such as hyphens and parentheses, governed by domain-, subject-, or language-specific conventions.

Note 2 to entry: A designation can be a term including appellations, a proper name, or a symbol.

[SOURCE: ISO 1087:2019, 3.4.1, modified — An additional admitted term "label" has been added.]

3.32
directed relationship
<UML> *relationship* (3.63) between a collection of source model elements and a collection of target model elements

Note 1 to entry: Adapted from UML 2.5.1, 7.8.5.1.

3.33
enumeration
<UML> *data type* (3.27) whose values are named individually in the *model* (3.48)

Note 1 to entry: The set of *enumeration literals* (3.34) owned by an enumeration is ordered.

Note 2 to entry: Adapted from UML 2.5.1, 10.5.3

3.34
enumeration literal
<UML> user-defined *data value* (3.28) for an *enumeration* (3.33)

Note 1 to entry: In this case, the user is the modeller.

Note 2 to entry: Adapted from UML 2.5.1, 10.5.4.1.

3.35
extension
<UML> *association* (3.9) which indicates that the *properties* (3.61) of a *metaclass* (3.46) are extended through a *stereotype* (3.65)

Note 1 to entry: Adapted from UML 2.5.1, 12.4.1.1.

3.36
feature
<UML> characteristic

Note 1 to entry: Adapted from UML 2.5.1, 9.4.3.1.

3.37
formal language
language that is machine readable and has well-defined semantics

Note 1 to entry: Well-defined semantics will typically be model-theoretic semantics.

[SOURCE: ISO/IEC 21838-1:2021, 3.10]

3.38
generalization
<UML> taxonomic *directed relationship* (3.32) between a more general *classifier* (3.16) and a more specific classifier

Note 1 to entry: The more general classifier is called the parent, or the superclass if the classifier is a *class* (3.14). The more specific classifier is called the child. The generalization is directed from the child to the parent. The classifiers that can be reached by following the generalizations from a given classifier in the direction towards the more general classifiers are called the classifier's generalizations. The classifiers that can be reached by following the generalizations from a given classifier in the direction towards the more specific classifiers are called the classifier's specializations.

Note 2 to entry: Each *instance* (3.42) of the specific classifier is also an instance of the general classifier. The specific classifier inherits the *features* (3.36) of the more general classifier.

Note 3 to entry: Adapted from UML 2.5.1, 9.2.3.2 and 9.9.7.

3.39
identifier
linguistically independent sequence of characters capable of uniquely and permanently identifying that with which it is associated

[SOURCE: ISO 19135-1:2015, 4.1.5]

3.40
identity
inherent characteristic of an *instance* (3.42) that distinguishes it from all other instances

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.1865, modified — The term "property" has been replaced by "characteristic" in the definition, and the Note to entry has been removed.]

3.41
inheritance
<information technology> mechanism by which more specific entities incorporate structure and behaviour defined by more general entities

3.42
instance
individual entity

Note 1 to entry: The term "instance" represents the same concept as the term "particular" defined in ISO/IEC 21838-1:2021, 3.3.

3.43
interface
<UML> *classifier* (3.16) that represents a declaration of a set of coherent public *features* (3.36) and obligations that together constitute a coherent service

Note 1 to entry: An interface specifies a contract; UML 2.5.1, 10.4.3.1 requires that any *instance* (3.42) of a classifier that realizes the interface fulfils that contract. The obligations associated with an interface are in the form of *constraints* (3.25) (such as pre- and post-conditions) or protocol specifications, which can impose ordering restrictions on interactions through the interface.

Note 2 to entry: Adapted from UML 2.5.1, 10.4.3.1.

3.44
keyword
<UML> reserved word that is an integral part of the UML notation

Note 1 to entry: Keywords normally appear as text annotations attached to a UML graphic element or as part of a text line in a UML diagram. Keywords are enclosed in guillemets («keyword») and thus have the same notation as *stereotyped* (3.65) model elements.

Note 2 to entry: Adapted from UML 2.5.1, Annex C.

3.45
level of abstraction
abstraction level
indication of the amount of detail that is outside the scope of interest

Note 1 to entry: A *model* (3.48) at a high level of abstraction has a relatively low amount of detail.

3.46
metaclass
<UML> *class* (3.14) in a *metamodel* (3.47)

EXAMPLE The class "Interface" is a class in the UML metamodel and is therefore a metaclass.

Note 1 to entry: Adapted from UML 2.5.1, 6.2.

3.47
metamodel
model (3.48) that defines a modelling language

EXAMPLE The UML metamodel.

Note 1 to entry: A model is an *instance* (3.42) of a metamodel, and a metamodel is an instance of a meta-metamodel.

Note 2 to entry: Adapted from the MDA Guide.[15]

3.48
model
abstraction (3.4) of some aspects of reality

[SOURCE: ISO 19109:2015, 4.15]

3.49
multiplicity
<UML> specification of the valid *cardinalities* (3.13)

EXAMPLE An *instance* (3.42) of a collection specified as having a multiplicity of "1..3" has at least one value and has not more than three values.

Note 1 to entry: A multiplicity is a definition of an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound.

Note 2 to entry: Adapted from UML 2.5.1, 7.5.3.2 and 7.8.8.1.

3.50
named element
<UML> element in a *model* (3.48) that can have a name

Note 1 to entry: Adapted from UML 2.5.1, 7.8.9.

3.51
namespace
<UML> *named element* (3.50) that either owns or imports, or both, a set of named elements that can be identified by a name

Note 1 to entry: Adapted from UML 2.5.1, 7.8.10.1.

3.52
natural language
language which is or was in active use in a community of people, and the rules of which are mainly deduced from usage

[SOURCE: ISO 5127:2017, 3.1.5.02, modified — The Note to entry has been removed.]

3.53
n-ary association
<UML> *association* (3.9) having more than two member ends

Note 1 to entry: An association with three members ends is called a ternary association.

Note 2 to entry: Adapted from UML 2.5.1., 11.5.3.1.

3.54
object
<UML> individual with a state and *relationships* (3.63) to other individuals

Note 1 to entry: An object is an *instance* (3.42) of a *class* (3.14).

Note 2 to entry: Adapted from UML 2.5.1, 6.3.1.

3.55
operation
<UML> *behavioural feature* (3.11) of an *interface* (3.43), *data type* (3.27) or *class* (3.14)

Note 1 to entry: UML 2.5.1, 9.6.3.1 permits that an operation is directly invoked on *instances* (3.42) of its featuring *classifiers* (3.16). The operation specifies the name, *type* (3.70), parameters and *constraints* (3.25) for such invocations.

Note 2 to entry: Adapted from UML 2.5.1, 9.6.3.1.

3.56
package
<UML> element that is used to group elements, and provides a *namespace* (3.51) for the grouped elements

Note 1 to entry: Adapted from UML 2.5.1, 12.4.5.1.

3.57
package diagram
<UML> structure diagram that shows certain aspects of one or more *packages* (3.56)

Note 1 to entry: Adapted from UML 2.5.1, Annex A

3.58
part-whole relation
partitive relation
relation between two *concepts* (3.20) where one of the concepts constitutes the whole and the other concept constitutes a part of that whole

Note 1 to entry: A part-whole relation exists between the concepts "week" and "day", "molecule" and "atom".

[SOURCE: ISO 5127:2017, 3.1.7.06, modified — The preferred term and the admitted term have exchanged positions.]

3.59
primitive type
<UML> predefined *data type* (3.27) without any substructure

Note 1 to entry: A primitive type can have algebra and operations defined outside of UML, for example, mathematically.

Note 2 to entry: Adapted from UML 2.5.1, 10.2.3.2.

3.60
profile
<UML> *package* (3.56) that defines limited extensions to a reference *metamodel* (3.47) with the purpose of adapting the metamodel to a specific platform or domain

Note 1 to entry: Adapted from UML 2.5.1, 12.4.7.

3.61
property
<UML> *structural feature* (3.66)

Note 1 to entry: A property is owned by a *classifier* (3.16), an *association* (3.9) or another property.

Note 2 to entry: In the UML *metamodel* (3.47), property is the only kind of structural feature. UML 2.5.1 does not clearly specify the difference between a property and a structural feature that is not a property.

Note 3 to entry: Adapted from UML 2.5.1, 9.4.3.2, 9.5.3 and 9.9.17.1.

3.62
realization
<UML> *abstraction* (3.5) between two *named elements* (3.50) or sets of named elements, one representing a specification and the other representing an implementation of the latter

Note 1 to entry: The supplier represents the specification and the client represents an implementation of the specification.

Note 2 to entry: Adapted from UML 2.5.1, 7.8.14.1.

3.63
relationship
<UML> connection between elements

Note 1 to entry: Adapted from UML 2.5.1, 7.8.15.1.

3.64
schema
formal description of a *model* (3.48)

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

3.65
stereotype
<UML> model element that extends an existing *metaclass* (3.46) and 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

Note 1 to entry: A stereotype is contained in a *profile* (3.60).

Note 2 to entry: Adapted from UML 2.5.1, 12.4.9.1.

3.66
structural feature
<UML> *feature* (3.36) that specifies an aspect of structure

Note 1 to entry: A structural feature represents values held as part of the structure of the *instances* (3.42) of the element [e.g. a *classifier* (3.16)] that owns it.

Note 2 to entry: A structural feature is also a kind of *typed element* (3.71).

Note 3 to entry: Adapted from UML 2.5.1, 7.5.3.1 and 9.9.21.

3.67
tag definition
<UML> *property* (3.61) of a *stereotype* (3.65)

Note 1 to entry: Adapted from UML 2.5.1, 12.3.3.4.

3.68
tagged value
<UML> value of a *tag definition* (3.67) applied to a model element

Note 1 to entry: Adapted from UML 2.5.1, 12.3.3.4.

3.69
template
<UML> parameterized model element

Note 1 to entry: Adapted from UML 2.5.1, 7.3.1.

3.70
type
<UML> model element that specifies a set of allowed values

Note 1 to entry: The set of allowed values are known as the *instances* (3.42) of the type.

Note 2 to entry: In UML 2.5.1, the term "type" is also a preferred term for another concept. UML 2.5.1, 22.3 defines that concept as "*stereotype* (3.65) of *class* (3.14) that specifies a domain of *objects* (3.54) together with the *operations* (3.55) applicable to the objects, without defining the physical implementation of those objects".

Note 3 to entry: Adapted from UML 2.5.1, 7.5.3.1.

3.71
typed element
named element (3.50) that can have a *type* (3.70) specified for it

Note 1 to entry: UML 2.5.1, 7.5.3 requires that if a typed element has an associated type, then any value represented by the typed element (at any point in time) is an *instance* (3.42) of the given type. UML 2.5.1, 7.5.3 permits that a typed element with no associated type represents any value.

Note 2 to entry: Adapted from UML 2.5.1, 7.8.22.1.

3.72
universe of discourse
view of the real or hypothetical world that includes everything of interest

[SOURCE: ISO 19101-1:2014, 4.1.38]

3.73
usage
<UML> *dependency* (3.30) in which the client element requires the supplier element (or set of elements) for its full implementation or operation

Note 1 to entry: Adapted from UML 2.5.1, 7.8.23.1.

3.74
value domain
set of accepted values

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

Note 1 to entry: This is the same concept as "value space" from ISO/IEC 11404:2007, 3.61.

4 Symbols and abbreviated terms

CSL	conceptual schema language
GML	Geography Markup Language
HMMG	Harmonized Model Maintenance Group
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IRI	Internationalized Resource Identifier
ITU	International Telecommunication Union
JCGM	Joint Committee for Guides in Metrology
OCL	Object Constraint Language
OMG	Object Management Group
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RFC	Request for Comments
SBVR	Semantics of Business Vocabulary and Business Rules
SFA	simple feature access
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System

SQL	Structured Query Language
UML	Unified Modeling Language
uom	unit of measure
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language

5 Conformance

5.1 Conformance overview

This document defines one requirements class. Conformance with the requirements class shall be checked using the corresponding conformance class, which is specified in the abstract test suite in Annex A.

In this document, the identifiers of the requirements class, the provisions, the conformance class and the test cases are expressed as relative URI references. The base URI of these relative URI references is https://standards.isotc211.org/19103/-/2.

5.2 Conceptual schemas modelled in UML

The sole and core requirements class is given in Table 1.

Table 1 — Requirements class "Conceptual schemas modelled in UML"

Identifier	/req/conceptual-schema
Standardization target type	conceptual schemas describing geographic information
Dependencies	UML 2.5.1
Provisions	The provisions in Clause 7.
Corresponding conformance class	/conf/conceptual-schema (Clause A.2)

6 Overview

Clause 7 describes a subset of the Unified Modeling Language (UML), as specified in UML 2.5.1, for specifying conceptual schemas within the domain of geographic information. Clause 7 sets out requirements and recommendations that restrict the way this subset is used. Clause 7 also defines a UML profile, containing several stereotypes.

Clause 8 lists a set of core data types for use in conceptual schemas, many of which are UML representations of data types defined in ISO/IEC 11404:2007, Clause 8 and Clause 10.

Annex A contains an abstract test suite for checking the requirements specified in this document.

Annex B describes backward compatibility between ISO 19103:2015 and this document.

Annex C contains an introduction to conceptual schema languages. It can be useful to read this annex before reading Clause 7.

Annex D contains a reference of the notation for the main UML model elements mentioned in Clause 7. It can be useful to refer to this annex while reading Clause 7.

Annex E describes the changes between UML 2.4.1[4] and UML 2.5.1.

Annex F describes mappings between the data types from Clause 8 and the data types specified in ISO/IEC 11404:2007, Clause 8 and Clause 10.

Annex G briefly describes conceptual schema representations.

Annex H explains the concept of a "code set" and concepts related to it.

7 Use of UML

7.1 General use of UML

Clause 7 provides requirements and recommendations on the use of UML 2.5.1 for specifying conceptual schemas within the domain of geographic information.

Requirement 1 /req/conceptual-schema/uml

The conceptual schema shall conform to UML 2.5.1.

NOTE 1 The reader is expected to have familiarity with UML 2.5.1, as UML 2.5.1 is a normative reference of this document.

NOTE 2 The UML Best Practices repository[17] contains more information on UML in a geographic information context and links to other useful resources on UML. It is maintained by the Harmonized Model Maintenance Group (HMMG), an ISO/TC 211 advisory group.

NOTE 3 The previous edition of this document referenced UML 2.4.1[4] – also known as ISO/IEC 19505-1:2012 and ISO/IEC 19505-2:2012. The main changes between UML 2.4.1 and UML 2.5.1 are summarized in Annex E.

Requirement 2 /req/conceptual-schema/level-of-abstraction

The conceptual schema shall document its level of abstraction.

The three main levels of abstraction distinguished in the ISO 19100 family of documents are:

a) the abstract schema level;

b) the application schema level;

c) the implementation schema level.

The first two levels are relevant for conceptual schemas.

NOTE 4 The profile defined in 7.8 provides the AbstractSchema stereotype for documenting the level of abstraction of a conceptual schema. The UML profile defined in ISO 19109 provides the ApplicationSchema stereotype.

Figure 1 visually explains how schemas relate to schemas at different levels of abstraction and how schemas relate to their metamodel. Abstract schemas, application schemas and implementation schemas are all kinds of models. They reside all on the same level of "meta" (as indicated by the dashed rectangle). A model conforms to a metamodel (e.g. the UML metamodel), which in turn conforms to a metametamodel. Models, metamodels and metametamodels reside on different levels of "meta".

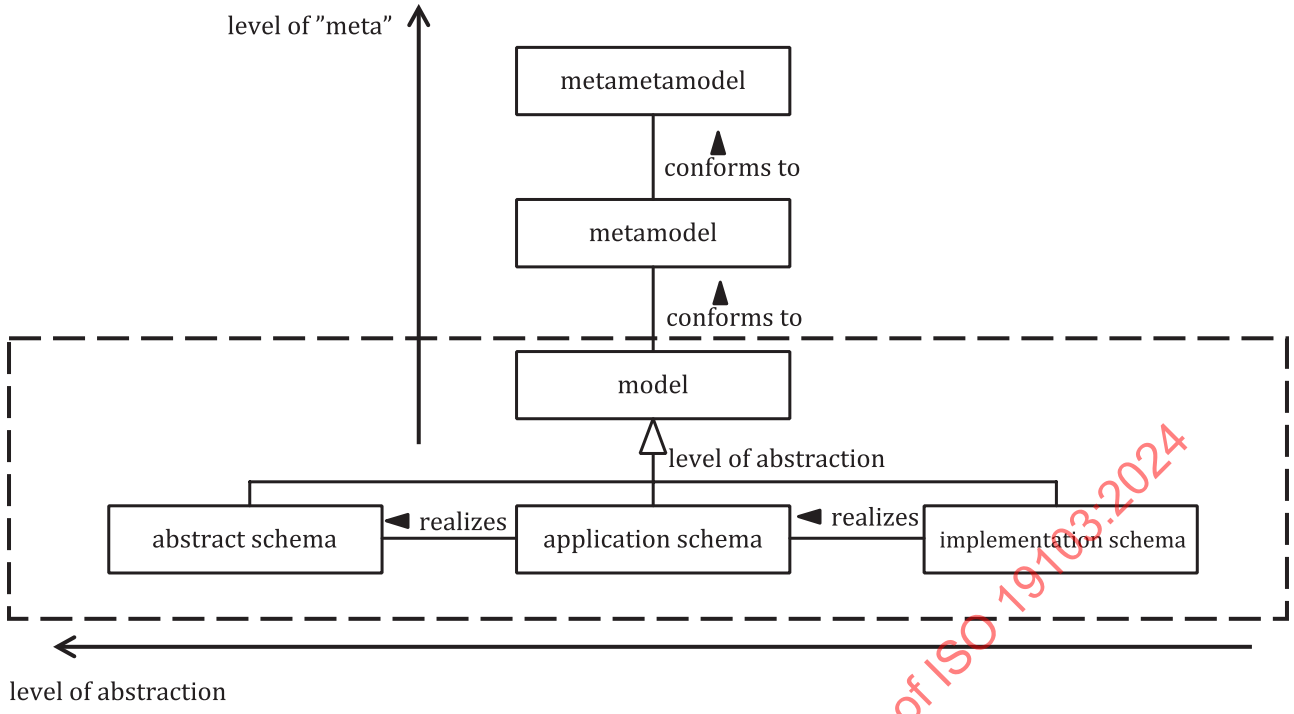


Figure 1 — Levels of abstraction versus levels of "meta"

Figure 2 gives an example of schemas at different levels of abstraction. The spatial schema for geometries defined in ISO 19107 is an abstract schema. It is (partly) realized by the simple feature access (SFA) geometry object model defined in the first part of the Simple feature access standard,[19] which resides at the application schema level of abstraction. The SFA geometry object model is in turn realized by both an SQL-implementation and a GML-implementation, in the second part of the Simple feature access standard[20] and the GML simple features profile,[21] respectively. The schemas in ISO 19107 and in the first part of the Simple feature access standard[19] are modelled in UML and conform to the UML metamodel.

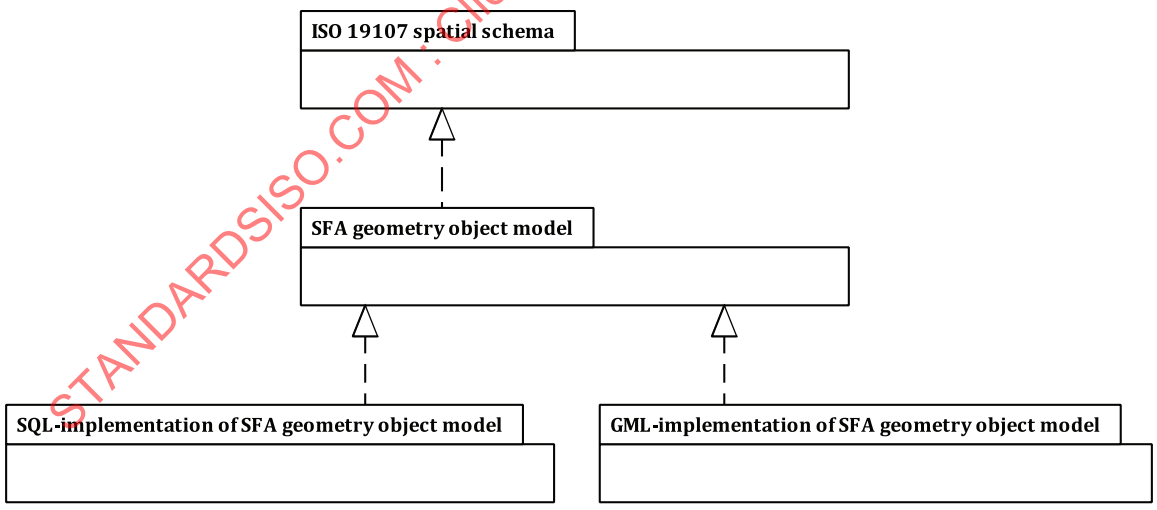


Figure 2 — Example of schemas at different levels of abstraction

7.2 Classifiers

7.2.1 General

Classes, data types, enumerations and interfaces are the main kinds of model elements used in conceptual schemas for geographic information. A class, a data type, an enumeration and an interface are all kinds of classifiers according to the UML metamodel, as illustrated in Figure 3. The classifiers class, data type, enumeration and interface are described in 7.2.2, 7.2.3, 7.2.4 and 7.2.5.

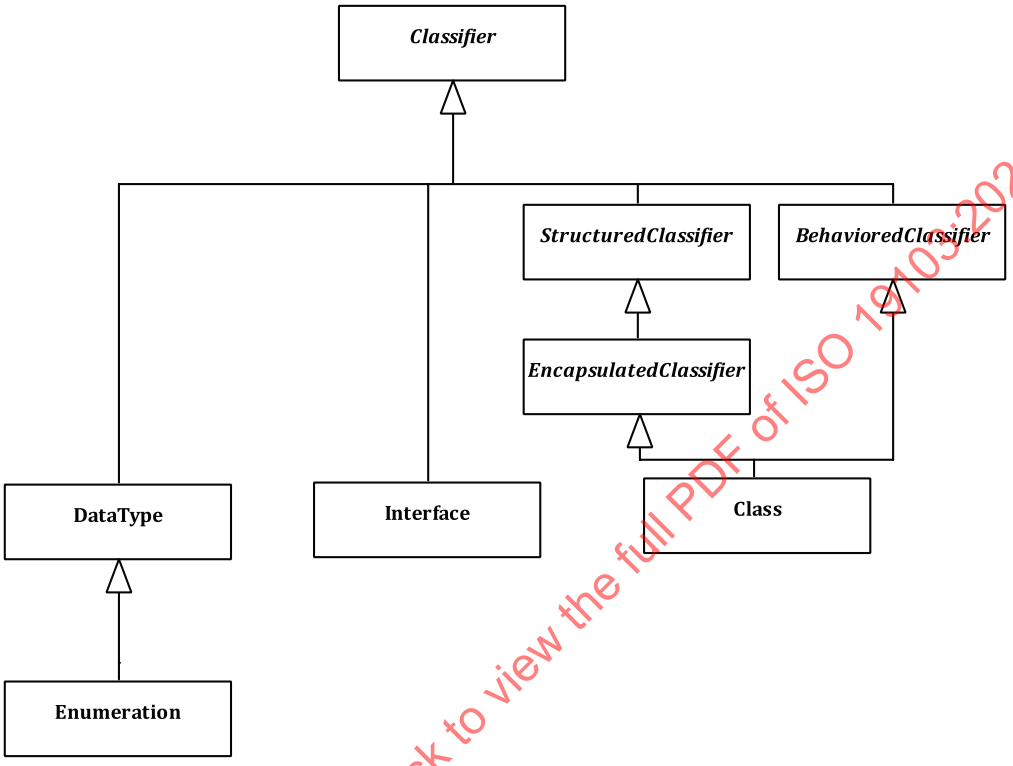


Figure 3 — Classifiers from the UML metamodel that are used in geographic information modelling

Recommendation 1 /rec/conceptual-schema/contents

If the conceptual schema is an abstract schema, the classifiers it contains should not be classes. If the conceptual schema is an application schema, the classifiers it contains should not be interfaces.

An interface defined in an abstract schema can be realized by other classifiers in an application schema. Both abstract schemas and application schemas can contain data types and enumerations.

NOTE 1 The UML metamodel defines more kinds of classifiers than the ones shown in Figure 3, for example "actor" and "use case".

NOTE 2 It is a common misconception that a data type is a class with a stereotype called "dataType". According to the UML metamodel a data type is not a kind of class (see also Figure 3). Similarly, an enumeration is not a kind of class, but a distinct kind of classifier. These misconceptions are caused by using guillemets («») in the notation for both keywords and stereotypes. Clause D.3 explains the difference between keywords and stereotypes in more detail.

NOTE 3 A classifier can be marked as being abstract, meaning that the classifier has no direct instances. In other words, every instance of the abstract classifier is an instance of one of its specializations (see 7.4.3). Abstract classifiers can be defined in schemas at different levels of abstraction.

NOTE 4 Certain UML metaclasses in Figure 3 are shown in italics, because they are abstract metaclasses. As explained in NOTE 3, an abstract metaclass can only be instantiated by instantiating one of its specializations. As a result, a modeller will not see a button labelled "Add Classifier", for example, in the graphical user interface of a tool conforming to UML 2.5.1.

NOTE 5 Metaclass PrimitiveType is another specialization of DataType (see UML 2.5.1, 10.2). It is not a part of the subset of UML used for modelling conceptual schemas of geographic information. Primitive types are typically used in implementation schemas; see Figure 32, for example.

7.2.2 Classes

Objects are categorized into classes. A class specifies the features that characterize the structure and behaviour of those objects (UML 2.5.1, 11.8.3.1).

NOTE 1 The terms class, object and feature have different meanings depending on the context or the domain. In this document, the terms are used in the same way as in UML 2.5.1.

Objects are distinguished by their identity; different objects can be uniquely identified. An object's identity is an inherent property of an object that distinguishes it from all other objects. If two objects have the same identity, those two objects are considered equal to each other.

NOTE 2 Equality of objects (instances of a classes) is based on object identity, whereas equality of data values (instances of data types) is based on value, see 7.2.3.

NOTE 3 More information and examples regarding instances with and without identity can be found in the literature on domain-driven design. In domain-driven design, instances with identity are called entities or reference objects, and instances without identity are called value objects (see Reference [22], for example).

In the real world, instances are often referred to indirectly: they do not necessarily have an identifier. In data management, however, assigning unique and immutable identifiers to instances is good practice, so instances can be referred to reliably. Therefore, a class commonly has a property that serves as an identifier. Two objects are then considered to be equal when their identifiers are equal. A property can be marked as being the identifier, or part of the identifier, for the classifier of which it is a property, as described in 7.3.2.

EXAMPLE 1 In the sentence "The coffee cup standing on the table over there", the coffee cup in question is referred to indirectly.

EXAMPLE 2 When two books have the same title, that does not necessarily mean that they are the same book. Books are assigned an International Standard Book Number (ISBN) to unambiguously identify them, and thus differentiate between them.

The context determines whether a real-world thing is treated as an object (an instance of a class) or as a data value (an instance of a data type).

EXAMPLE 3 An address can be represented as a class, having its own identifier. In another context, representing an address as a data type can be adequate.

EXAMPLE 4 Usually, names of persons and places are modelled as data types in conceptual schemas supporting public administration's base registries, because only the spelling of the name matters. However, in the context of an etymology name database, a name would be seen as an object.

7.2.3 Data types

A classifier whose instances are distinguished only by their value is a data type.

NOTE 1 The data type concept discussed in Clause 7, and in 7.2.3 in particular, is that described in 3.27, i.e. the data type concept as defined in UML 2.5.1. This is not the same concept as described in 3.26, which is the data type concept as defined in ISO/IEC 11404:2007, 3.12.

Two instances of a data type are considered to be equal when they have the same value. Two instances of a structured data type, i.e. a data type with attributes, are considered to be equal if and only if the structure is the same and the values of the corresponding attributes are equal (UML 2.5.1, 10.2.3.1).

NOTE 2 Equality of data values is based on value, whereas equality of objects is based on object identity, see 7.2.2.

Figure 4 gives an example of equality of data values. Data values "dateRange1" and "dateRange2" are instances of data type "DateRange". They are considered to be equal, as both their start dates (2021-07-16) and their end dates (2023-06-30) are the same.

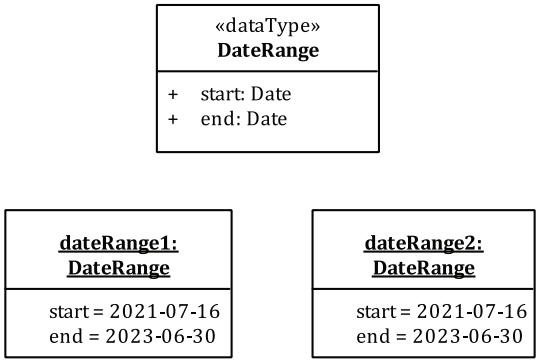


Figure 4 — Example of equality of data values

Data types are usually defined in application schemas or implementation schemas. Certain abstract schemas contain data types too, for instance data type DirectPosition specified in ISO 19107:2019, 6.2.9.

7.2.4 Enumerations

A data type whose values are enumerated in a model is an enumeration. This means that the conceptual schema editor names the values of an enumeration individually inside the conceptual schema. The values are called enumeration literals.

NOTE 1 The collection of enumeration literals owned by an enumeration is ordered as well as unique. That collection is therefore an ordered set. Subclause 7.3.2.3 elaborates on kinds of collections.

Figure 5 gives an example of an enumeration with enumeration literals, showing the seven main colours of the rainbow modelled as enumerations literals with names "red", "orange", etc.

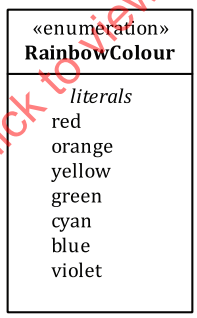


Figure 5 — Example of an enumeration with enumeration literals

When creating an implementation schema based on a conceptual schema, the implementation of the enumeration literals can be achieved in different ways. One way is using the same name in the implementation schema as in the conceptual schema. Another way is coding the values of the enumeration into code values in the implementation schema. Annex H elaborates on coding.

EXAMPLE The enumeration literals from the example in Figure 5 are implemented by code values in this XML schema fragment.

```
<simpleType name="RainbowColourType">
    <annotation>
        <documentation>classification of rainbow colors</documentation>
    </annotation>
    <restriction base="string">
        <enumeration value="1">
            <annotation>
                <documentation>red</documentation>
            </annotation>
        </enumeration>
        <enumeration value="2">
```

```
            <annotation>
                <documentation>orange</documentation>
            </annotation>
        </enumeration>
        <enumeration value="3">
            <annotation>
                <documentation>yellow</documentation>
            </annotation>
        </enumeration>
        <enumeration value="4">
            <annotation>
                <documentation>green</documentation>
            </annotation>
        </enumeration>
        <enumeration value="5">
            <annotation>
                <documentation>cyan</documentation>
            </annotation>
        </enumeration>
        <enumeration value="6">
            <annotation>
                <documentation>blue</documentation>
            </annotation>
        </enumeration>
        <enumeration value="7">
            <annotation>
                <documentation>violet</documentation>
            </annotation>
        </enumeration>
    </restriction>
</simpleType>
```

NOTE 2 Enumeration literals are different from properties (which are discussed in 7.3.2). An enumeration literal with a default value (also called an initial value) does not conform to UML 2.5.1, despite the fact that certain UML tools allow the specification of a default value for an enumeration literal, as internally, they deal with attributes and enumeration literals similarly. In other words, even if a tool would not prevent entering the code value for an enumeration literal in the graphical user interface element that records default values of attributes, the conceptual schema does still not conform to UML 2.5.1.

Adding, changing and/or removing an enumeration literal implies a conceptual schema modification. The modification can potentially trigger a modification of any implementation schema realizing the conceptual schema.

NOTE 3 If enumeration literals are coded, a simple enumeration literal name change does not trigger a modification of the implementation schemas.

7.2.5 Interfaces

A UML interface specifies a set of externally visible properties and operations.

Interfaces are usually defined in abstract schemas and are usually realized by other classifiers in schemas at a lower level of abstraction.

An interface can be realized in different ways, depending on the implementation context. The presence of a property in an interface implies that a realizing classifier allows a user to retrieve the value of this property from an instance of the realization, but it does not necessarily imply that a realizing classifier has this property in its internal structure. The realizing classifier can, for example, support retrieving the value of the property by means of a query operation.

EXAMPLE An implementation of Date, described in 8.2.8, provides a way to retrieve all the following properties: day, month and year.

The core data types defined in Clause 8 are modelled as UML interfaces, as they are intended to be used in conceptual schemas, independent of the implementation context.

7.3 Features

7.3.1 General

The specializations of UML features used for modelling conceptual schemas for geographic information are properties, which are structural features, and operations, which are behavioural features.

NOTE "Feature" as discussed in 7.3 refers to the concept "feature" as defined in UML 2.5.1, 9.4. It does not refer to the concept "feature" as defined in ISO 19101-1:2014, 4.1.11.

7.3.2 Properties

7.3.2.1 General

A property can be either an attribute of a classifier or a member end of an association (UML 2.5.1, 9.5.3). The latter is also called an "association end".

Subclause 7.3.2 specifies provisions for both kinds of properties. Subclause 7.4.2 specifies additional provisions for properties that are member ends of associations.

NOTE As a property is not a classifier, a property of a property is conceptually not supported by UML. A typical use for such a construct would be metadata at the property level.

7.3.2.2 Type of a property

According to the UML metamodel, a property is a typed element, i.e. a named element that can have a type. A type specifies a set of allowed values known as the instances of the type. See also UML 2.5.1, 7.5.3 and Figure 6.

UML 2.5.1, 7.5.3 requires that if a typed element has an associated type, then any value represented by the typed element (at any point in time) is an instance of the given type. UML 2.5.1, 7.5.3 permits that a typed element with no associated type represents any value.

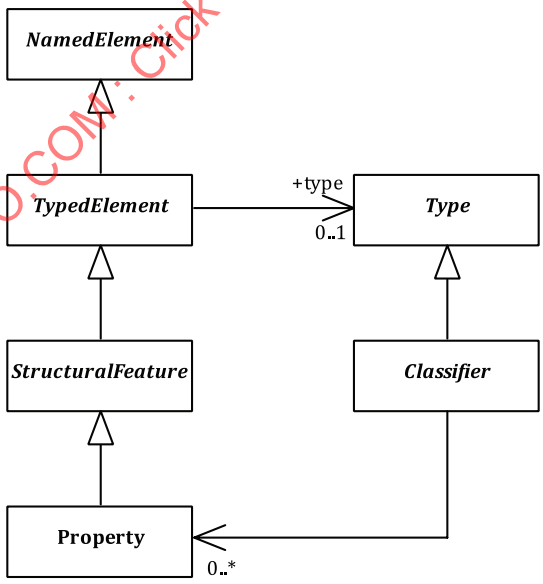


Figure 6 — Type and Property, from the UML metamodel

Figure 7 gives an example of an attribute with a type and an attribute without a type. The type of attribute id is CharacterString. No type is specified for attribute result. It can therefore have any value.

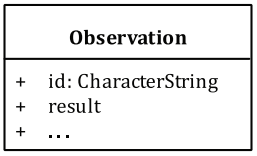


Figure 7 — Example of an attribute with a type and an attribute without a type

Figure 8 gives an example of association ends and their type. The type of association ends author and reader is Person.

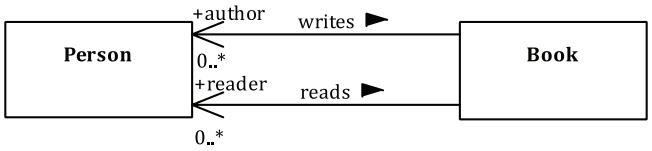
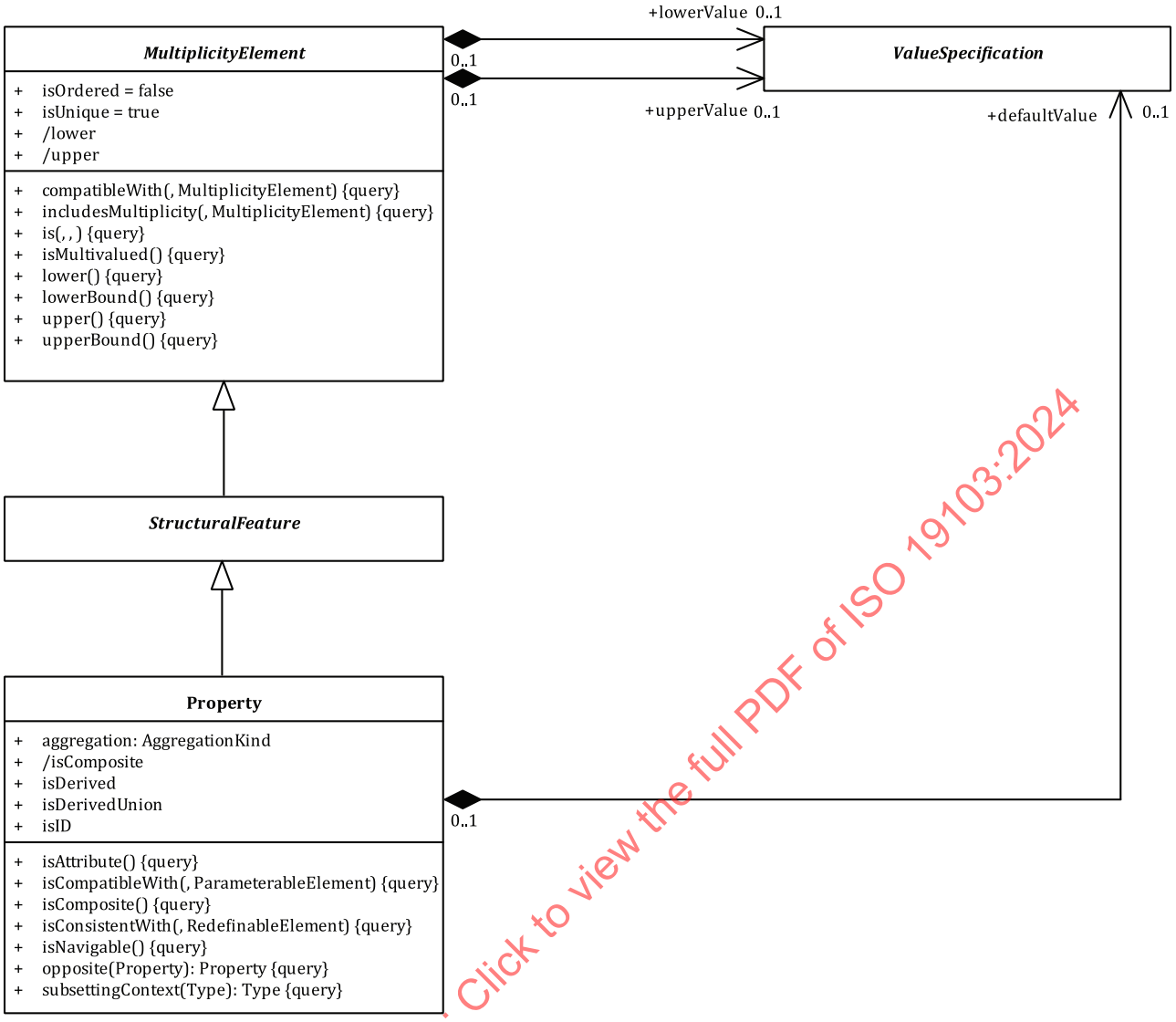


Figure 8 — Example of association ends and their type

7.3.2.3 Multiplicity of a property

The allowable number of instances of a property is specified by the multiplicity of that property. A multiplicity consists of a lower bound and an upper bound, as defined in Figure 9 and in UML 2.5.1, 7.8.8.7.



NOTE 1 The query “lowerBound()” returns the lower bound of the multiplicity as an integer, which is the “integerValue” of “lowerValue”, if this is given, and 1 otherwise (see UML 2.5.1, 7.8.8.7).

NOTE 2 The query “upperBound()” returns the upper bound of the multiplicity for a bounded multiplicity as an unlimited natural, which is the “unlimitedNaturalValue” of “upperValue”, if given, and 1, otherwise (see UML 2.5.1, 7.8.8.7).

NOTE 3 The query “isAttribute()” is true if the property is defined as an attribute of some classifier (see UML 2.5.1, 9.9.17.7).

Figure 9 — MultiplicityElement and Property, from the UML metamodel

The descriptions of the operations “lowerBound()” and “upperBound()” of “MultiplicityElement” (see Figure 9) can be rephrased as follows. The lower bound of a multiplicity is treated as being 1 if it is not explicitly indicated. Similarly, the upper bound of a multiplicity is treated as being 1 if it is not explicitly indicated. In other words, the default multiplicity of a property is 1..1. This is potentially not known to all users of conceptual schemas. Therefore, this document specifies Recommendation 2.

Recommendation 2 /rec/conceptual-schema/explicit-multiplicity

For each property, both the lower bound and the upper bound of the multiplicity should be explicitly specified.

A property is specified as either ordered or unordered (unordered is the default), and is specified as either unique or not unique (unique is the default); see also Figure 9.

For a multivalued property (a property with an upper bound greater than 1) the combination of the values of the metaproperties isOrdered and isUnique is used to specify that the collection of values in an instance of that property is of one of four kinds of collections. Table 2 gives an overview of the possible combinations of isOrdered and isUnique, including the kind of collection that corresponds to every combination. The notation for the kinds of collections is given in Table D.1.

For a property that is not multivalued, neither isOrdered nor isUnique has an effect (UML 2.5.1, 7.5.3.2).

Table 2 — Kinds of collections

Ordered	Unique	Kind of collection	See also[a]
no	yes	set	ISO/IEC 11404:2007, 8.4.3 and Reference [23]
yes	yes	ordered set	—
no	no	bag (= multiset)	ISO/IEC 11404:2007, 8.4.4 and Reference [24]
yes	no	sequence (= list = ordered bag)	ISO/IEC 11404:2007, 8.4.5 and Reference [25]

NOTE See also UML 2.5.1, Table 7.1.

[a] Column "See also" references resources providing more information about the kind of collection.

NOTE The mechanism described in this subclause is also reflected in the OCL 2.4 standard.[26] For more information, see OCL 2.4, 11.6 and OCL 2.4, 12.8, paragraph 1.

7.3.2.4 Identifier properties

A property can be marked as being the identifier – or part of the identifier – for the classifier that owns it by setting metaproperty isID to true (see 7.3.2.4). The notation for identifier properties is further described in D.1.2.

Figure 10 gives an example of an identifier property. Property isbn (ISBN = International Standard Book Number) uniquely identifies an instance of class Book.

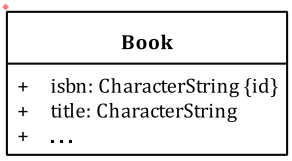


Figure 10 — Example of an identifier property

7.3.2.5 Categorization of properties

A property can be categorized as being mandatory, optional or conditional. This categorization is based on the lower bound of the property's multiplicity and any constraints on the property, as shown in Table 3.

Table 3 — Categorization of properties

Condition	Kind of property
multiplicity with a lower bound of at least 1	mandatory property
multiplicity with a lower bound of 0	optional property
multiplicity with a lower bound of 0 + constraint that specifies under what conditions instances of the property are present or not	conditional property

NOTE The categorization presented in Table 3 is often used in data dictionaries in ISO/TC 211 documents. Mandatory properties are indicated with "M", optional properties with "O" and conditional properties with "C".

7.3.3 Operations

This document does not express further requirements for operations in the conceptual schema, other than what is specified in UML 2.5.1, 9.6 and 9.9.11.

7.4 Relationships

7.4.1 General

Figure 11 illustrates the kinds of UML relationships used for modelling conceptual schemas for geographic information.

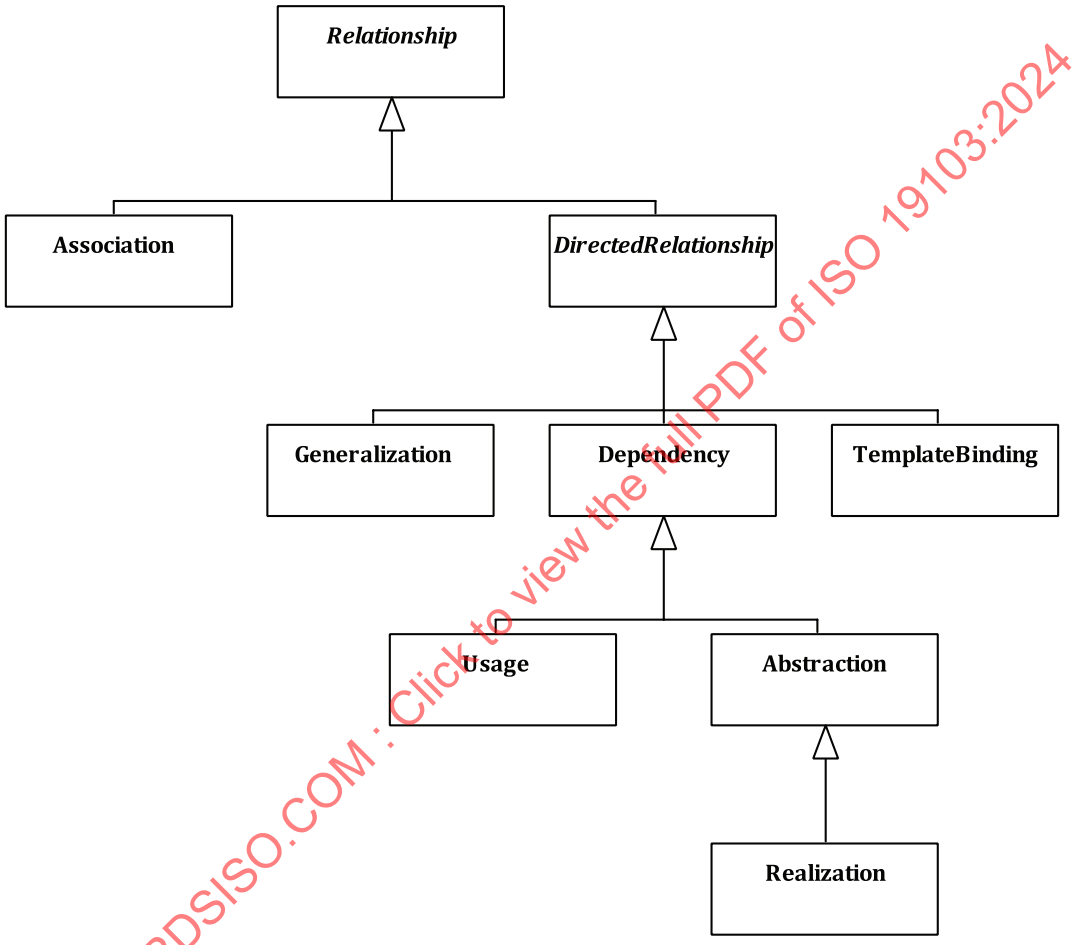


Figure 11 — Relationships from the UML metamodel that are used in geographic information modelling

7.4.2 Associations

7.4.2.1 General

Classifiers can be connected to each other by means of associations.

An association that has two member ends is called a binary association. An association that has more than two member ends is called an n-ary association. For example, a ternary association has three member ends.

A binary association can represent a part-whole relation. UML 2.5.1 distinguishes between two kinds of part-whole relations:

— aggregations, where the part end has shared aggregation;

— compositions, where the part end is compositely aggregated.

NOTE 1 According to UML 2.5.1, 9.5.3, the precise semantics of shared aggregation varies by application area and modeller.

NOTE 2 Aggregations and compositions are instances of the UML metaclass Association. This is because the UML metamodel does not define distinct metaclasses for representing part-whole relations, see also Figure 11.

Association classes are associations where the association itself has properties, see UML 2.5.1, 11.5.3.2.

7.4.2.2 Association names

Meaningful association names as well as reading directions (▶) aid a user in understanding the semantics of how the classifiers are connected, and hence reduce ambiguity.

Recommendation 3 /rec/conceptual-schema/association-names

Each binary association should have both a name as well as an indication of the direction in which the association is to be read.

Examples of associations with names are given in 7.9.

7.4.2.3 Navigable association ends

If a binary association's end has an open arrowhead (→), that association end is navigable. Navigability indicates that an object knows the object(s) it is connected to.

NOTE 1 UML 2.5.1, 11.5.4, defines the notation for a cross (×) for non-navigable association ends. However, using this notation is optional. It currently remains common practice to show the open arrowheads for navigable association ends and to suppress the crosses for non-navigable association ends. This practice is also followed in the diagrams in this document.

NOTE 2 It has been common practice to assume that a navigable association end is owned by the classifier at the opposite end. This practice remains current, even though UML 2.5.1, 11.5.4, deprecates it. Consequently, ownership is usually not explicitly indicated graphically in conceptual schemas that adhere to this document. This practice is also followed in the diagrams in this document. The graphical indication of ownership by a small filled circle (●), also called a dot, is explained and illustrated in UML 2.5.1, 11.5.4.

A navigation direction expresses a hint for how to implement the conceptual schema. Therefore, subsequent implementation is eased by defining the role name already in the conceptual schema.

Requirement 3 /req/conceptual-schema/navigable-association-end-name

Each navigable end of each binary association shall have a role name.

7.4.2.4 Use of associations

An association end owned by the classifier at the opposite end can also be represented using the attribute notation, instead of the association notation, according to UML 2.5.1, 11.5.4. The following conventions are common practice, see UML 2.5.1, 9.5.3, paragraph 2. They are also used in the diagrams in this document.

— A property whose type has identity is modelled using an association.

— A property whose type does not have identity is not modelled using an association.

UML classes have identity (see 7.2.2). UML data types have no identity (see 7.2.3). The core data types, modelled as UML interfaces and described in Clause 8, have no identity either. A property without a type can only be modelled using the attribute notation. See 7.3.2.2 for more information about the type of properties.

A data type can have a property that is modelled using an association. That is, however, not a very common situation.

Figure 12 gives an example of a data type having a property modelled using an association. Data type DirectPosition has property coordinateReferenceSystem with type CoordinateReferenceSystem. Any realization of interface CoordinateReferenceSystem will have identity, as indicated by property identifier. Property coordinateReferenceSystem is modelled using an association.

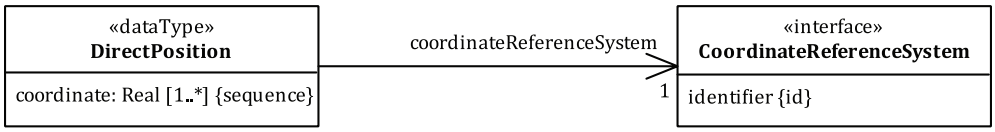


Figure 12 — Example of a data type having a property modelled using an association

7.4.2.5 N-ary associations and association classes

Recommendation 4 /rec/conceptual-schema/no-n-ary-associations

N-ary associations should be avoided in the conceptual schema in order to reduce complexity and avoid model mapping problems.

7.4.3 Generalizations

Requirement 4 /req/conceptual-schema/generalizations

Each generalization shall only connect classifiers that are instances of the same class from the UML metamodel.

For example, if a data type is specialized, the specific classifier will be a data type as well.

NOTE 1 When using generalization sets to group generalizations, the default constraints on a generalization set are {incomplete, overlapping}. An inconsistency regarding the default constraints was taken care of in UML 2.5.1, 9.7. See also https://issues.omg.org/issues/UML25-646.

NOTE 2 A generalization is owned by the more specific classifier (the child), see UML 2.5.1, 9.9.7.1.

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 5 applies.

Recommendation 5 /rec/conceptual-schema/no-multiple-inheritance

A classifier should not have more than one parent unless it is a fundamental part of the semantics of the type hierarchy.

7.4.4 Realizations

Realizations can be used for a variety of purposes. One is to relate model elements at different levels of abstraction, that is, to describe shifts in abstraction levels, where a concrete element (called the client, the realizing element or the realization) realizes a more abstract element (called the supplier, or the realized element). By comparison, generalizations relate elements at the same level of abstraction (see 7.4.3).

When used between classifiers, a realization can be thought of as inheritance of behaviour without inheritance of structure, as explained in Reference [27].

Figure 13 gives examples of realizations. The spatial schema for geometries defined in ISO 19107 and the simple feature access (SFA) geometry object model defined in the first part of the Simple feature access standard[19] are connected to each other by means of a realization. The interface Point defined in ISO 19107:2019, 6.4.13 and the class of the same name in the SFA geometry object model are also connected to each other by means of a realization. Class Point is also called a realization of interface Point.

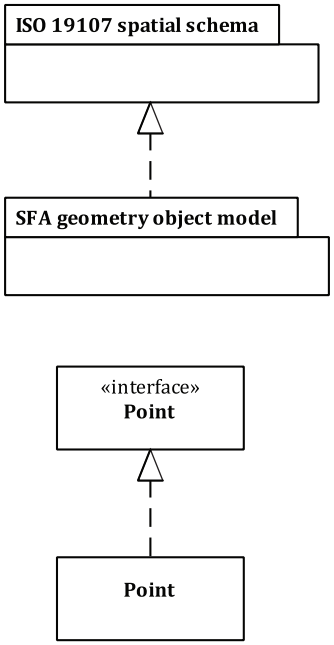


Figure 13 — Examples of realizations

7.4.5 Template bindings

Infrequently, template bindings can be useful in conceptual schemas for geographic information. This subclause does not elaborate on template bindings and templates in general, but the notation is given in D.1.3 and the relevant parts of the UML metamodel are described in UML 2.5.1, 7.3.

7.5 Packages

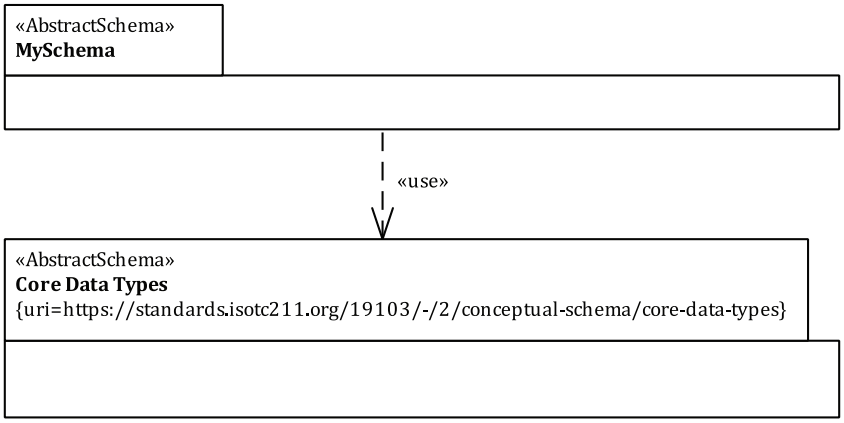
Packages are a convenient mechanism for structuring models. A package is a namespace that can own any kind of model element. Packages can contain other packages. Packages can have dependency relationships to other packages. A dependency denotes that model elements in one package depend on model elements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

Requirement 5 /req/conceptual-schema/dependencies

The conceptual schema shall have a dependency to another conceptual schema if a model element in the first schema depends on a model element in the other schema.

An appropriate specialization of dependency may be used, for example a usage (see also 7.4.1).

Figure 14 gives an example of a usage. The abstract schema MySchema uses one or more of the core data types from abstract schema CoreDataTypes.



NOTE A reference of the notation for relationships, including usages, is given in D.1.3.

Figure 14 — Example of a usage

Requirement 6 /req/conceptual-schema/directed-acyclic-graph

The conceptual schema shall not depend on another conceptual schema if the other schema depends on the first schema.

Requirement 6 is in the information technology domain also known as the acyclic dependencies principle.

A package can have a URI attribute which serves as the universally unique identifier of the package (UML 2.5.1, 12.2 and 12.4.5), as shown in Figure 15. This possibility is used:

— for the profile defined in 7.8;

— for the package referred to in Requirement 12 (7.11.2);

— for the package referred to in Recommendation 19 (7.11.3).

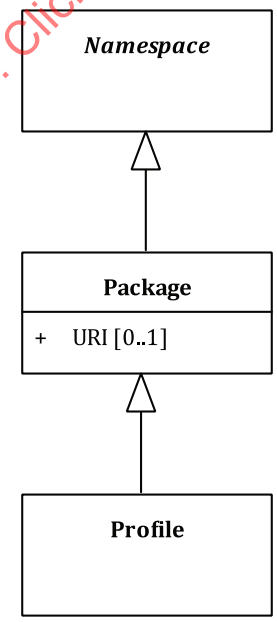


Figure 15 — Namespace, Package and Profile, from the UML metamodel

7.6 Comments

This document does not express further requirements to comments in the conceptual schema, other than what is specified in UML 2.5.1, 7.2 and 7.8.2.

7.7 Constraints

Constraints add precision to a model, by adding model constructs that cannot be expressed in pre-defined UML constructs. Expressing constraints at the conceptual level is important for validation, management and maintenance of data.

Requirement 7 /req/conceptual-schema/natural-language-constraints

Each constraint shall be expressed in natural language.

Recommendation 6 /rec/conceptual-schema/unambiguous-constraints

Each constraint should be unambiguous.

NOTE 1 Traditionally, the Object Constraint Language (OCL)[26] has been used to formulate unambiguous constraints for UML models. OCL is a textual and more formal language, and it is used in many of the constraints in the UML metamodel in UML 2.5.1.

NOTE 2 Specifications exist for writing unambiguous constraints in natural language. For example, the Semantics of Business Vocabulary and Business Rules (SBVR)[28] and RuleSpeak.[29]

Constraints can be shown in different ways, depending on the model elements they constrain (see also UML 2.5.1, 7.6.4).

Figure 16 gives an example of a constraint that constrains class ContactPoint. The constraint is expressed in both natural language and in OCL.

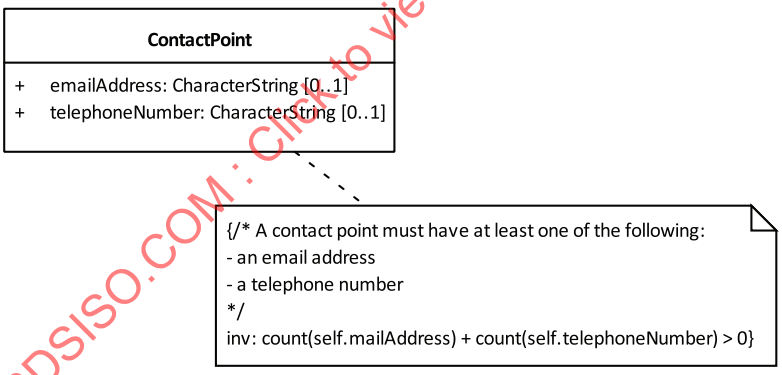


Figure 16 — Example of a constraint

7.8 UML profile

This document specifies a UML profile tailored for the geographic information domain. The main reasons for defining this profile are:

— to define properties at the metalevel that are applicable to many kinds of model elements and whose values improve semantic interoperability, in particular the metaproperties “designation”, “definition” and “description”;

— to be able to differentiate between different kinds of data types, and use that information when transforming a conceptual schema into an implementation schema.

NOTE 1 “Profile” as defined in UML 2.5.1 and as illustrated in Figure 15 is not the same concept as the concept “profile” defined in ISO 19106:2004. This is explained in Reference [31], 2.4.1.

Requirement 8 /req/conceptual-schema/uml-profile-application

The conceptual schema shall use the stereotypes of the profile with identifier https://standards.isotc211.org/19103/-/2/uml-profile when appropriate, and shall not create alternatives with the exact same meaning.

An overview of the contents of the profile is given in Figure 17. An overview of the stereotypes is presented in Table 4.

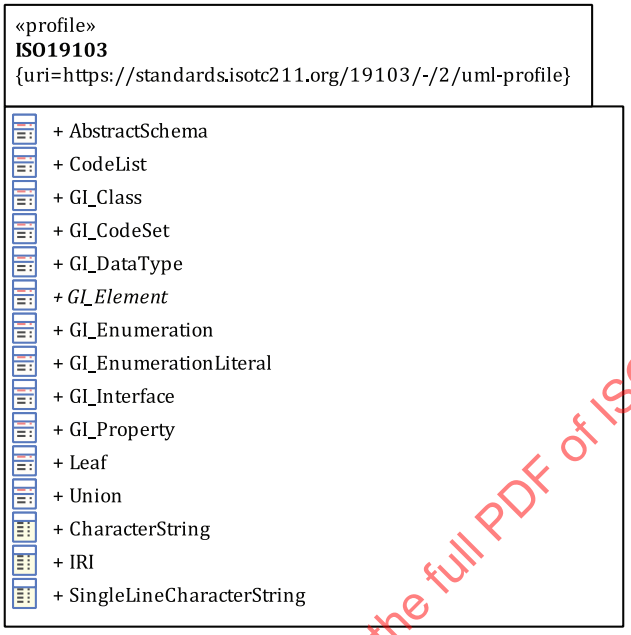


Figure 17 — UML profile – contents

Table 4 — Stereotypes in the UML profile for geographic information

Stereotype	Metaclass extended	Stereotype specialized	Figure
AbstractSchema	Package	GI_Element	Figure 21
CodeList[a]	Class, DataType, Enumeration		Figure 24
GI_Class	Class	GI_Element	Figure 20
GI_CodeSet	DataType	GI_Element	Figure 22
GI_DataType	DataType	GI_Element	Figure 20
GI_Element			Figure 18
GI_Enumeration	Enumeration	GI_Element	Figure 20
GI_EnumerationLiteral	EnumerationLiteral	GI_Element	Figure 20
GI_Interface	Interface	GI_Element	Figure 20
GI_Property	Property	GI_Element	Figure 20
Leaf[b]	Package		Figure 23
Union[b]	Class, DataType		Figure 24

[a] Deprecated; superseded by GI_CodeSet

[b] Deprecated.

The stereotypes GI_Class, GI_DataType, GI_Enumeration, GI_EnumerationLiteral, GI_Interface and GI_Property do not add additional semantics to UML. They merely enable adding tagged values (to classes,

data types, enumerations, enumeration literals, interfaces and properties, respectively) whose purpose is to improve semantic interoperability in a structured way.

NOTE 2 UML 1.5[32] recommends defining tag definitions in conjunction with a stereotype, but still allows tag definitions that are not associated with any stereotype. The latter practice is no longer permitted in the later UML specifications, including UML 2.5.1.

NOTE 3 UML 2.5.1 recommends choosing stereotype names that do not clash with the keyword for the extended model element (UML 2.5.1, 12.4.9.6). Therefore, most stereotypes' names are prefixed with "GI_", which stands for geographic information, the scope of ISO/TC 211. It was a deliberate choice to have the underscore as part of the prefix "GI_", to highlight the fact that it is indeed a prefix, and not really a part of the name itself. Note that the UML profile defined in this document is not a conceptual schema and is therefore not subject to the naming provisions specified in 7.9.

The semantic tagged values are defined on and inherited from the abstract stereotype GI_Element, as indicated in Table 4, Figure 18, Table 5 and Figure 20.

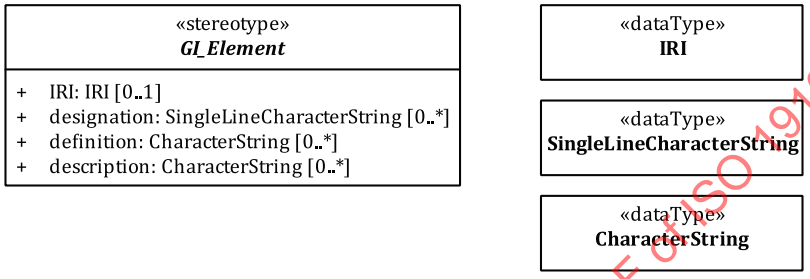


Figure 18 — UML profile – stereotype GI_Element

The data types used as type of the tagged values are the following:

— IRI: realization of the Internationalized Resource Identifier (IRI) type described in 8.2.13.

— CharacterString: realization of the CharacterString type described in 8.2.7.

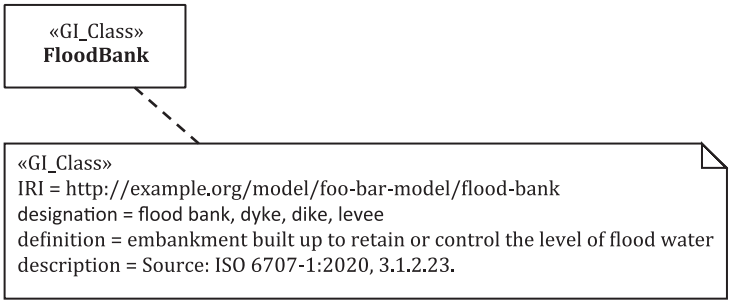
— SingleLineCharacterString: data type whose value domain is the value domain of CharacterString except for end of line characters and newline characters.

NOTE 4 The core data types defined in Clause 8 are defined as UML interfaces, which cannot be used directly in profiles according to UML 2.5.1, 12.3.3.1.1, paragraph 13. Therefore, the profile defines new data types, defined as UML data types, that realize the relevant core data types.

Table 5 — Definition and description of the properties of stereotype GI_Element

Name	Definition	Description
IRI	model element identifier in the form of a Internationalized Resource Identifier (IRI)	This identifier can be used directly in an encoding suitable for the Semantic Web.
designation	representation of a concept by a sign which denotes it in a domain or subject	Multiple designations may exist. Preferably, designations are expressed in a natural language.
definition	representation of a concept by an expression that describes it and differentiates it from related concepts	Multiple definitions may exist. It is a common practice to have a single definition for a concept.
description	expression which serves to give information in addition to the definition	Multiple descriptions may exist. Unlike a definition, a description includes for instance notes and examples; see ISO 704.

Figure 19 gives an example of the application of a specialization of GI_Element.



NOTE The cardinality of the value of "designation" of the class with name "FloodBank" is four, as this class has four designations: "flood bank", "dyke", "dike" and "levee". On the class diagram, the instances of "designation" are displayed as a comma-separated list. In the conceptual schema itself, however, the four instances of "designation" are present individually. See also UML 2.5.1, 12.3.4.

Figure 19 — Example of the application of a specialization of GI_Element

Properties designation, definition and description are multivalued (see Figure 18). This leaves open the possibility of creating multilingual conceptual schemas, thus conceptual schemas having model elements that are documented in more than one natural language. This document does not specify provisions for how to specify the natural language of the values of these properties.

Multilingual documentation of conceptual schemas can also be achieved by managing the translations of designations, definitions and descriptions separately from the conceptual schemas, for example in a register. The conceptual schema and the translations can be managed at a different pace and by different governance rules if the conceptual schema development and the translation processes are decoupled.

NOTE 5 Properties designation, definition and description, shown in Figure 18 and described in Table 5, are compatible with the properties of the same name defined for metaclass IdentifiedType in the General Feature Model defined by ISO 19109.

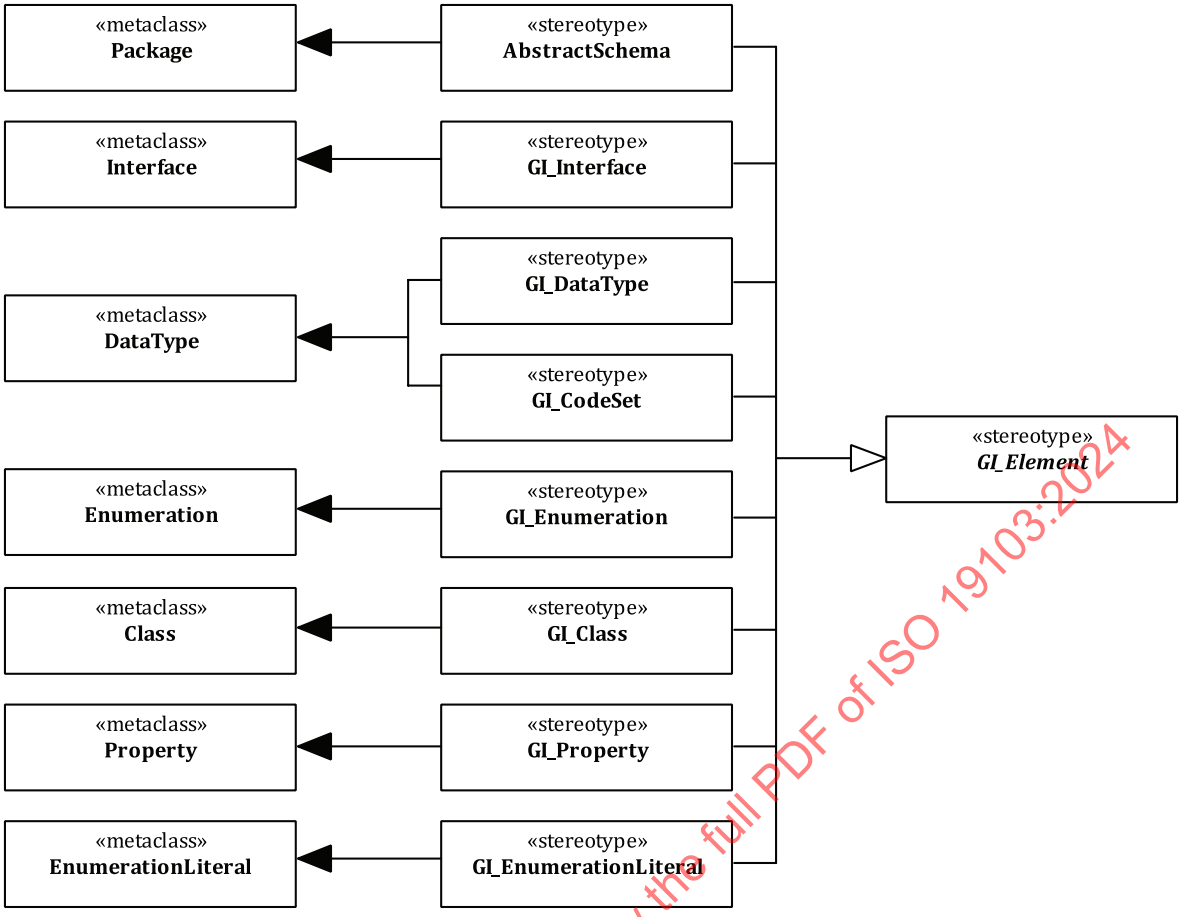


Figure 20 — UML profile – specializations of GI_Element

Recommendation 7 /rec/conceptual-schema/terminological-data

The conceptual schema should contain meaningful values for the tagged values inherited from stereotype GI_Element.

Recommendation 7 includes adding precise and unambiguous definitions to the elements of conceptual schema.

NOTE 6 Certain model elements can have designations that are self-explanatory and therefore do not need a definition.

NOTE 7 National or community rules can constrain the presence of definitions on certain or all model elements to be mandatory.

NOTE 8 Principles for terminology work, including the writing of definitions, are given in ISO 704. ISO 704:2022, 6.2 recommends that intensional definitions are used whenever possible. ISO 704:2022, 6.4.3.1 requires that intentional definitions are as concise as possible.

Stereotype AbstractSchema (Figure 21, Table 6) is used to declare that the level of abstraction of a particular conceptual schema is the abstract schema level (see 7.1).

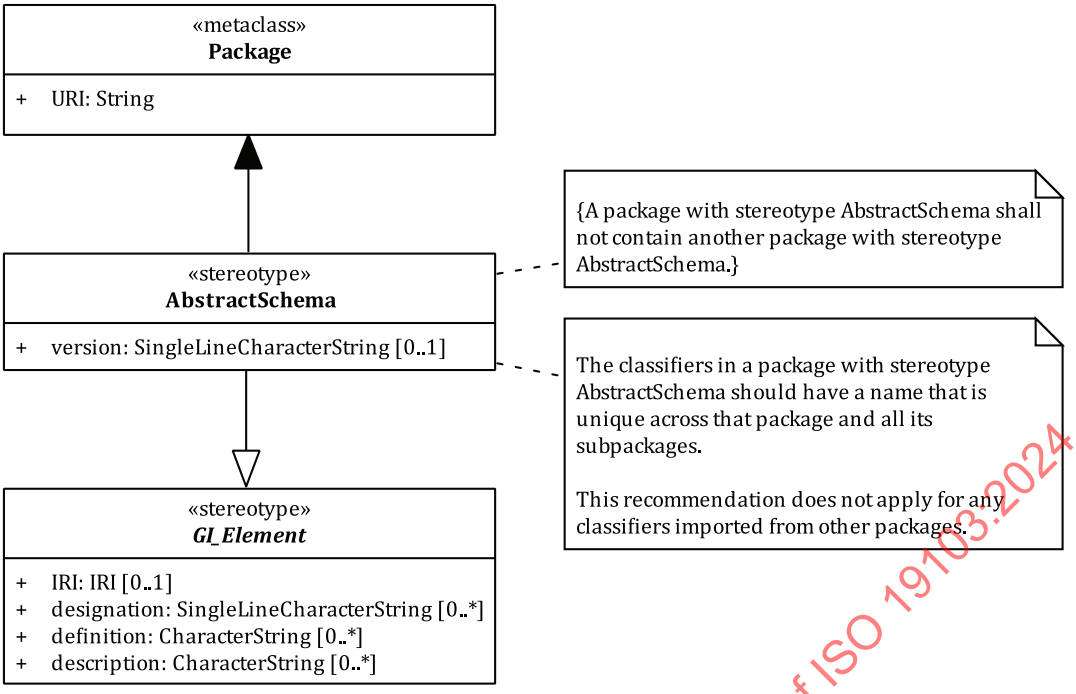


Figure 21 — UML profile – stereotype AbstractSchema

Table 6 — Definition and description of the properties of stereotype AbstractSchema

Name	Definition	Description
version	identifier assigned to a version of the abstract schema	

A data type with stereotype GI_CodeSet (Figure 22) represents a data type whose instances are the code values from a particular code set that is managed independently of and outside the conceptual schema. The code set is identified using the tagged values of the stereotype.

An enumeration and a data type with stereotype GI_CodeSet have in common that their values are specified individually. An enumeration's values are specified individually within the model the enumeration is defined in, whereas the values of a data type with stereotype GI_CodeSet are specified individually beyond the model the data type is defined in.

An enumeration and a data type with stereotype GI_CodeSet are different regarding the consequences of a change in their values. Adding, changing and/or removing an enumeration literal results in a conceptual schema modification. In comparison, a conceptual schema is not modified when the code set referenced by a data type with stereotype GI_CodeSet is changed.

The concepts code set and code value are explained in more detail in Annex H.

EXAMPLE 1 The code set for the representation of the names of languages as alpha-3 code values is identified by (and available at) http://id.loc.gov/vocabulary/iso639-2. This identifier is set as value for tagged value "IRI".

EXAMPLE 2 At the time of writing, ISO has not assigned official identifiers to the country code sets defined by ISO 3166-1. ISO 3166-1 defines three code sets, and therefore, using a URI that points to ISO 3166-1 does not identify which of the three code sets is meant. Instead, a possibility is to set "ISO 3166-1 alpha-2 country codes" as value for tagged value "designation".

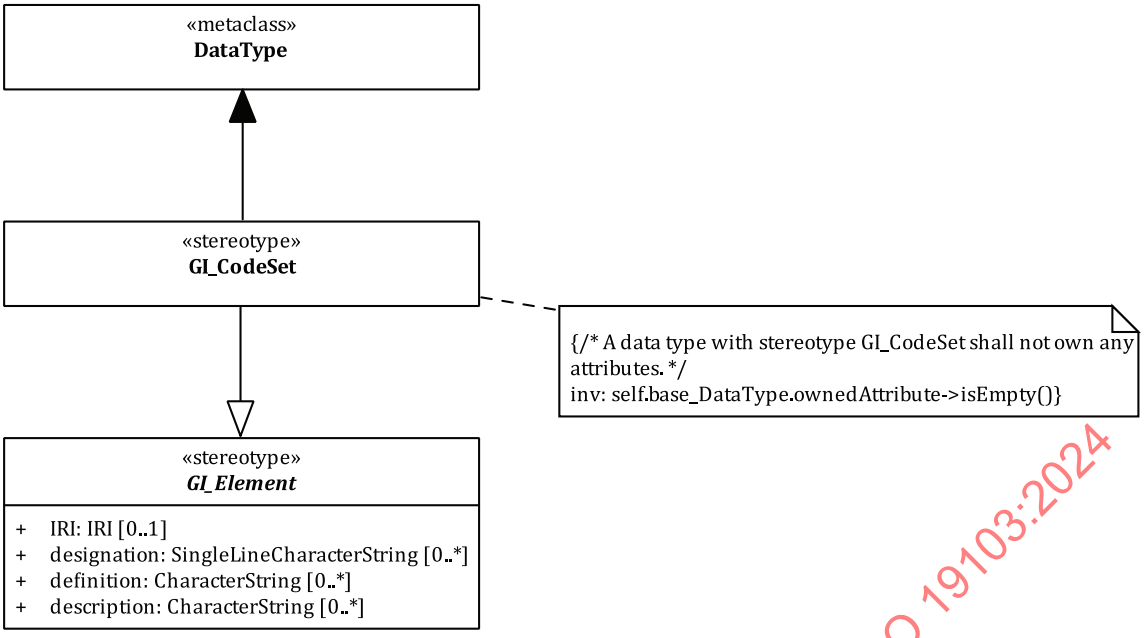


Figure 22 — UML profile – stereotype GI_CodeSet

This document deprecates the stereotypes Leaf, CodeList and Union. They are still included in the profile to maintain backward compatibility.

The stereotype Leaf (Figure 23) can be applied to packages that do not contain any other packages. It has no tag definitions.

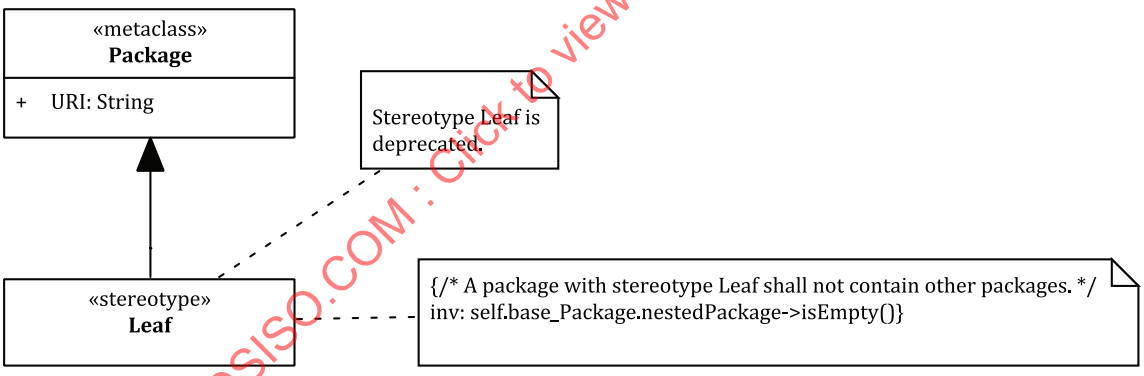


Figure 23 — UML profile – stereotype Leaf (deprecated)

CodeList and Union extend more than one metaclass (Figure 24), see also Annex B.

A classifier with stereotype Union is a data type that consist of one and only one of several alternative data types, which are modelled as attributes.

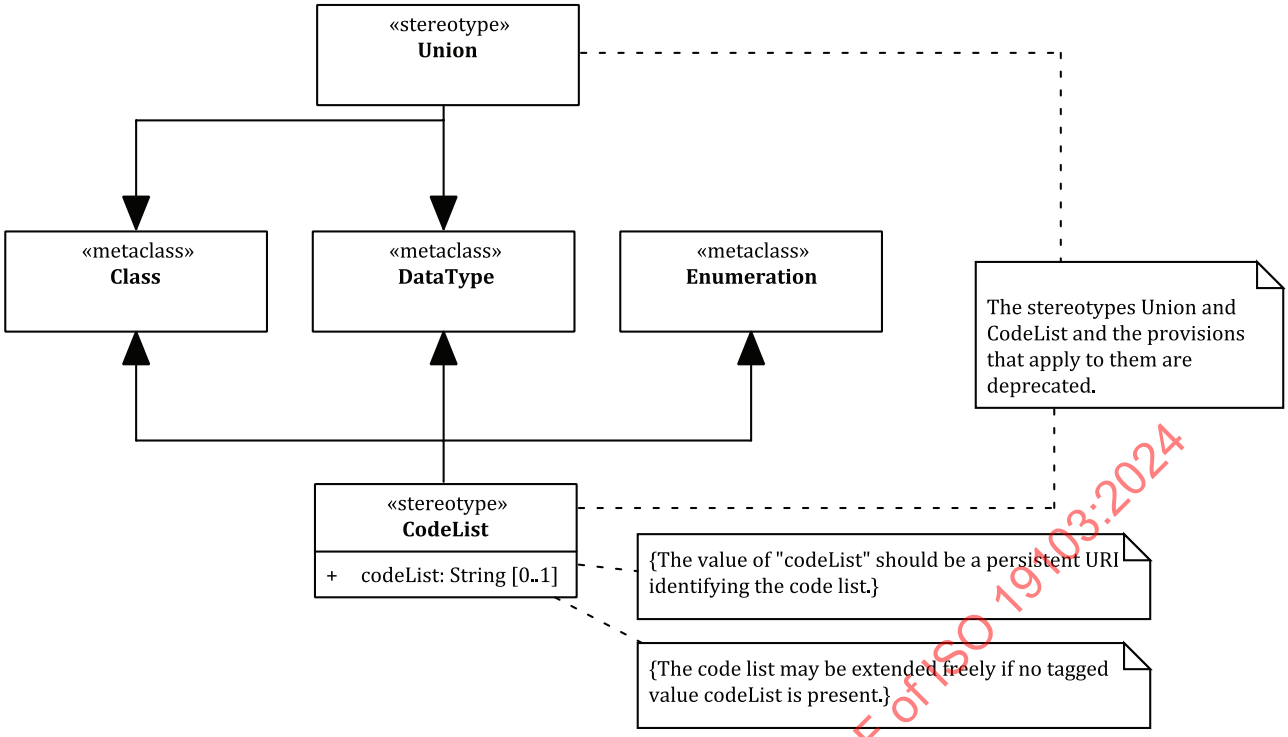


Figure 24 — UML profile – stereotypes CodeList and Union (deprecated)

7.9 Naming provisions

Provisions regarding naming are relevant for a variety of reasons, mainly readability, consistency and as a protection against case-sensitive binding.

Requirement 9 /req/conceptual-schema/unique-names

Each named element shall have a name that is unique within the element's namespace, regardless of the letter case of the name.

This means that in a conforming conceptual schema:

— names of classifiers are case-insensitive unique within the package to which those classifiers belong;

— names of properties are case-insensitive unique within the classifier that owns those properties.

NOTE 1 Case is a property of characters in certain alphabets whereby a character can be an uppercase or lowercase variant of a single letter. See ISO/IEC 10646 for more information.

Recommendation 8 /rec/conceptual-schema/precise-names

Each named element should have a technical name that is precise and understandable.

Multiple words may be combined to create precise and understandable names.

EXAMPLE 1 Use index as opposed to n.

EXAMPLE 2 Use computePartialDerivatives as opposed to compute.

Recommendation 9 /rec/conceptual-schema/property-names-measure

The name of each property with a measure as type should convey the semantics of the property.

Conveying the semantics of such a property can be done by reusing the names for quantities standardized in the ISO/IEC 80000 series.

NOTE 2 The ISO/IEC 80000 series standardizes names, symbols, definitions and units for quantities. In particular, ISO 80000-3 describes quantities of space and time. It includes descriptions of:

— acceleration

— angular acceleration

— angular measure

— angular velocity

— area

— depth

— diameter

— distance

— duration

— height

— length

— radius

— speed

— thickness

— velocity

— volume

— width

Figure 25 gives examples of properties with standardized quantity names. Properties crownDiameter, height and trunkDiameter reuse the standardized quantity names diameter and height.

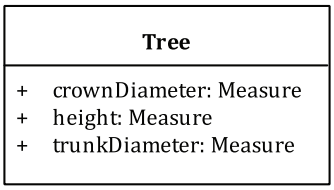


Figure 25 — Examples of properties with standardized quantity names

Measure is described in 8.2.14.

Recommendation 10 /rec/conceptual-schema/short-parameter-names

Each parameter name should be short when the parameter container or type carries meaning.

EXAMPLE 3 equals(other:GM_Object) as opposed to equals(otherGeometryObject:GM_Object)

NOTE 3 Some older models still use prefixes.

Recommendation 11 /rec/conceptual-schema/characters-in-names

Each named element should have a name that does not contain characters which, according to ISO/IEC 10646, belong to one of the following categories: Mark, Punctuation, Separator, Other.

For example, the use of underscores (_), hyphens (-) and spaces is discouraged. This is especially relevant when the names will be reused without modifications in an implementation schema, as implementation languages often forbid or discourage the use of those characters in names.

EXAMPLE 4 computePartialDerivatives (not: compute Partial Derivatives or compute_Partial_Derivatives)

NOTE 4 The package names in the Harmonized Model (available via the ISO/TC 211 resources site[37]) often do contain spaces, for readability reasons.

Recommendation 12 /rec/conceptual-schema/camel-case

Each package name and classifier name should follow the UpperCamelCase naming convention. Each property name, association name, operation name and parameter name should follow the lowerCamelCase naming convention.

With the UpperCamelCase naming convention, the words in compound names are joined together and each word's first letter is in upper case. With the lowerCamelCase naming convention, the words in compound names are also joined together, but the first letter is in lower case and the first letters of the subsequent words are in upper case.

EXAMPLE 5 (operation) computePartialDerivatives (not computepartialderivatives or COMPUTEPARTIALDERIVATIVES).

EXAMPLE 6 (class) CoordinateTransformation (not coordinateTransformation).

Recommendation 13 /rec/conceptual-schema/short-names

Each model element should have a name that is as short as practical.

This means for example:

— use standard abbreviations, provided they are understandable;

— skip prepositions;

— drop verbs when they do not significantly add to the meaning of the name.

EXAMPLE 7 (operation) equals instead of isEqual.

EXAMPLE 8 (operation) value instead of getValue.

EXAMPLE 9 (operation) initObject instead of initializeObject.

EXAMPLE 10 (operation) length instead of computeLength.

Recommendation 14 /rec/conceptual-schema/role-names

The role name of each association end should reflect the role that the classifier at the association end plays with respect to the classifier at the other end. The role name should not express the whole association.

Consequently, a proper role name is a noun or a noun phrase.

Recommendation 15 /rec/conceptual-schema/association-names

Each association name should contain a verb phrase.

The reason for this is that the association names and role names provide the user with the terminology for communicating about the conceptual schema.

Figure 26 gives an example of the application of Recommendation 14 and Recommendation 15. Role name "hostingSite" or "site" is preferred over role name "isLocatedAt" or "hosts" at the Site end of the association between "Facility" and "Site". On the other hand are "isLocatedAt" and "hosts" good names for the association itself. Role name "containedInstallation", "installation" or "part" is preferred over role name "contains" or "hasPart" at the "Installation" end of the association between "Facility" and "Installation".

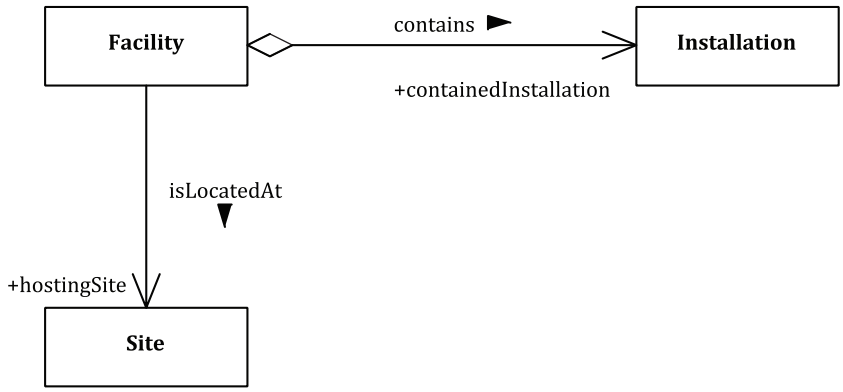


Figure 26 — Example of the application of Recommendation 14 and Recommendation 15 (1)

Figure 27 gives another example of the application of Recommendation 14 and Recommendation 15. Classes "Book" and "Person" are connected by two associations, making it even more important that the modeller carefully describes the semantics of each association. "Person" is named after the underlying nature of its objects, not after the roles those objects play. Indeed, the same person can be both an author and a reader of a book.

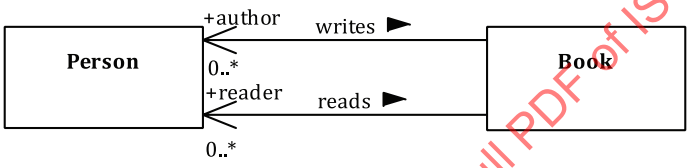


Figure 27 — Example of the application of Recommendation 14 and Recommendation 15 (2)

7.10 Diagrams

7.10.1 General

UML 2.5.1 defines two major kinds of diagrams: structure diagrams and behaviour diagrams. Structure diagrams show static structure, that is, aspects of a universe of discourse that are irrespective of time. On the other hand, behaviour diagrams show dynamic behaviour, that is aspects of a universe of discourse that change over time.

Structure diagrams and behaviour diagrams can be further categorized. The taxonomy of structure and behaviour diagrams is given in UML 2.5.1, Annex A. This document is not concerned with behaviour diagrams. As for structure diagrams, the following kinds are relevant (see also Figure 28):

— class diagrams;

— object diagrams;

— package diagrams;

— profile diagrams.

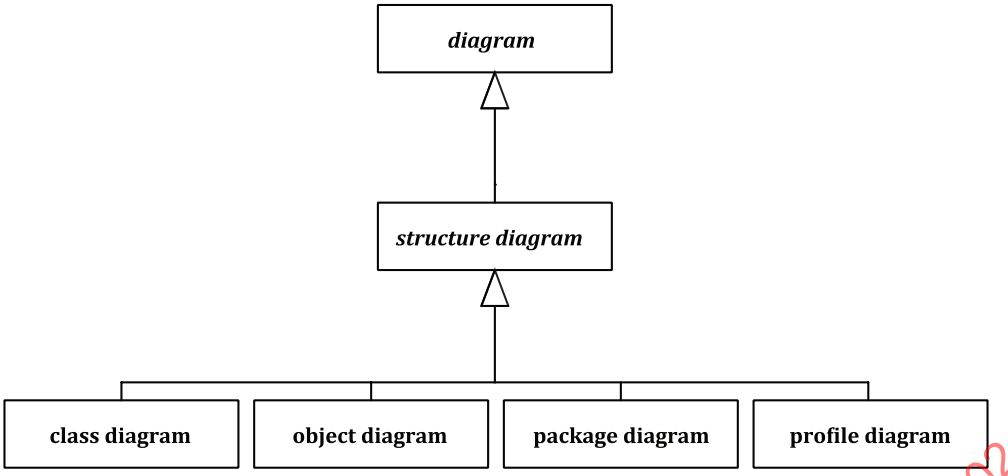


Figure 28 — Kinds of structure diagrams relevant to this document

Readable and understandable diagrams contribute to a better common understanding of a conceptual schema. Good practices regarding how to make such diagrams can be found in other resources addressing UML in general.

A modeller can choose to hide elements that are not relevant to the message a diagram wants to convey, to reduce the amount of information in the diagram, thereby making the diagram more comprehensible.

EXAMPLE Often, the role name and the multiplicity of a non-navigable association end will be hidden on class diagrams.

Recommendation 16 /rec/conceptual-schema/hide-qualified-names

The qualified names of named elements should not be included on diagrams, unless there is a potential for confusion or a need for emphasis.

UML 2.5.1, 7.4 and 7.8.9 elaborate on qualified names, named elements and namespaces.

7.10.2 Package diagrams

Requirement 10 /req/conceptual-schema/package-dependency-diagram

All dependencies of the conceptual schema shall be documented in a package dependency diagram.

See 7.5 for an example.

Recommendation 17 /rec/conceptual-schema/package-contents-diagram

Each package's contents should be documented in a package diagram.

See Figure 31 for an example.

7.10.3 Class diagrams

Typically, most of the diagrams showing (aspects of) a particular conceptual schema are class diagrams.

Recommendation 18 /rec/conceptual-schema/overview-diagram

The conceptual schema should be documented in an overview diagram.

An overview diagram focuses on the most important parts of a conceptual schema. An overview diagram usually does not show any attributes or operations.

Requirement 11 /req/conceptual-schema/context-diagrams

Each classifier shall be documented in a context diagram.

Closely related classifiers may share a context diagram if the combined diagram is neither overly cluttered nor difficult to read.

7.11 Reusable types

7.11.1 General

To improve the semantic interoperability of conceptual schemas, this document specifies provisions regarding the use of two sets of types at the conceptual level. Both sets are part of the Harmonized Model, which is available via the ISO/TC 211 resources site.[37]

7.11.2 Core data types

The first set of reusable types is described in Clause 8 of this document and contains common and generic data types, most of which are specified formally in ISO/IEC 11404:2007, Clause 8 and Clause 10.

Requirement 12 /req/conceptual-schema/core-data-types

The conceptual schema shall use the core data types in the abstract schema with identifier https://standards.isotc211.org/19103/-/2/conceptual-schema/core-data-types and not create new types with equivalent functionality.

The core data types are presented in Clause 8.

NOTE UML 2.5.1 does not prescribe the use of specific data types. UML 2.5.1 contains a package "PrimitiveTypes", described in UML 2.5.1, Clause 21. However, this package only defines a small set of primitive types that are commonly used in the definition of metamodels, namely Integer, Boolean, String, UnlimitedNatural and Real. This set is not sufficient for data interchange.

7.11.3 Common types

The second set of reusable types contains types that are less common and generic than the types in the first set, but are nonetheless considered to be shareable across multiple conceptual schemas. This set is managed in the Harmonized Model and is overseen by the HMMG under the ISO/TC 211 Harmonized Resources Maintenance Agency.

Recommendation 19 /rec/conceptual-schema/common-types

The conceptual schema should use the types in the package with identifier https://standards.isotc211.org/harmonized-model/common-types and not create new types with equivalent functionality.

8 Core data types

8.1 General

8.1.1 Relation with ISO/IEC 11404

ISO/IEC 11404 provides a specification for language-independent data types. In other words, ISO/IEC 11404 specifies data types at the conceptual level.

NOTE In Clause 8, the term "data type" refers to the data type concept as defined in ISO/IEC 11404:2007, 3.12, not to the data type concept as defined in UML 2.5.1 and discussed in 7.2.3.

In principle, data types from ISO/IEC 11404 can be mapped bidirectionally to internal data types from, for example, a programming language, a data format or a specification language. The concept of these bidirectional mappings is illustrated in Figure 29.

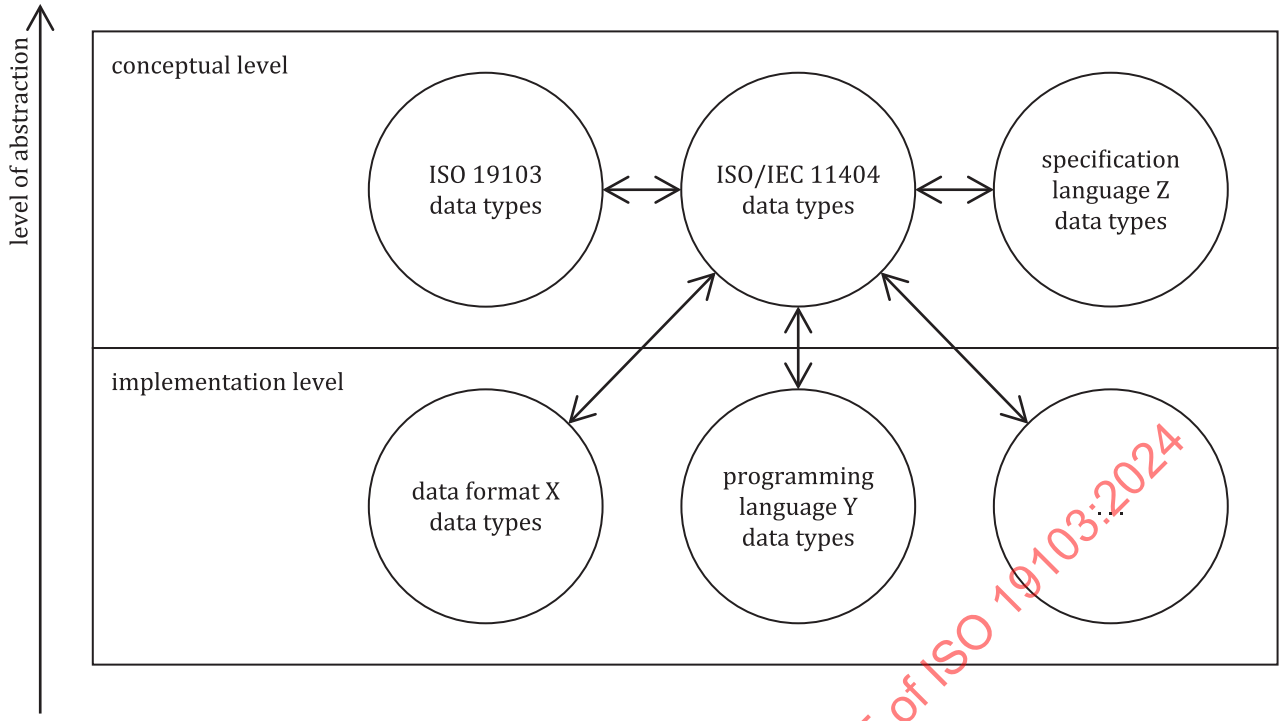


Figure 29 — ISO/IEC 11404 specification of language-independent data types

This document specifies only a UML representation of most of the standardized data types from ISO/IEC 11404:2001, Clause 8 and Clause 10. Consequently, the ISO 19103 data types reside at the conceptual level as well. Annex F contains the mapping between the ISO 19103 core data types and the ISO/IEC 11404 data types. Reuse of the ISO 19103 UML representations in conceptual schemas is required, as stated in 7.11.2.

NOTE 3 Certain data types from ISO/IEC 11404 are excluded because they are useful mainly in programming languages and not in conceptual schemas.

A conceptual schema can be implemented in one or more implementation schemas. The implementation includes mapping each ISO 19103 data type to an appropriate construct in the implementation schema language. In practice, this mapping is done directly between the ISO 19103 data types and the data types from the implementation schema language, not via the ISO/IEC 11404 data types, as illustrated in Figure 30. An ISO 19103 core data type can, for example, be mapped to:

— a built-in data type;

— a standardized encoding of the data type;

— a new, user-defined data type.

NOTE 4 Encoding is further discussed in ISO 19118.

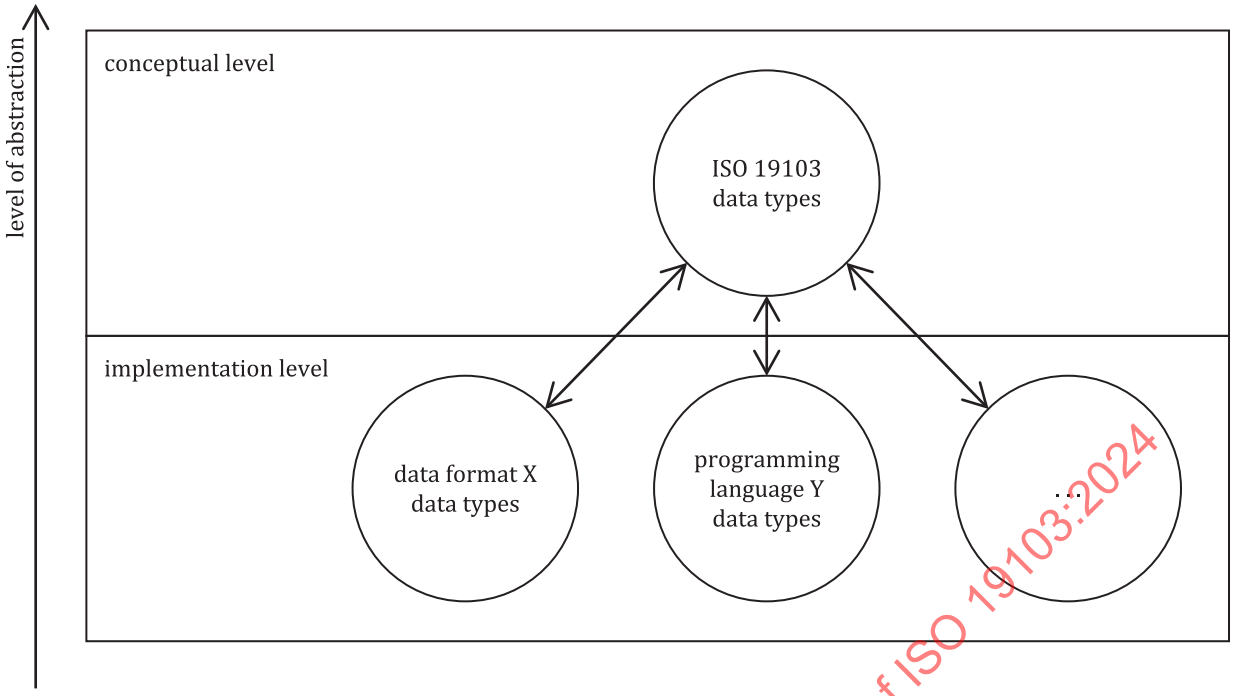


Figure 30 — Implementation of the ISO 19103 data types

8.1.2 Modelling choice for the core data types

The core data types are located in the abstract schema with identifier https://standards.isotc211.org/19103/-/2/conceptual-schema/core-data-types, see Figure 31.

Figure 31 — Contents of Core Data Types

The core data types are modelled as UML interfaces (7.2.5) because they reside at the conceptual level. They are intended to be used in other conceptual schemas, independent of the implementation context. The core data types are realized (7.4.4) by built-in data types from the implementation schema language in the most straightforward cases.

Figure 32 gives an example of a core data type realized by built-in data types from implementation schema languages. The interface CharacterString (see also 8.2.7) is realized by:

— the Extensible Markup Language (XML) primitive data type “string”;

— the Structured Query Language (SQL) primitive data type “CHARACTER VARYING”.

NOTE Certain data types categorized as defined data types in ISO/IEC 11404 are treated as primitive data types in other languages. For example, the XML string data type is categorized as a primitive data type in the XML Schema type system; see XML Schema Part 2: Datatypes.[38]

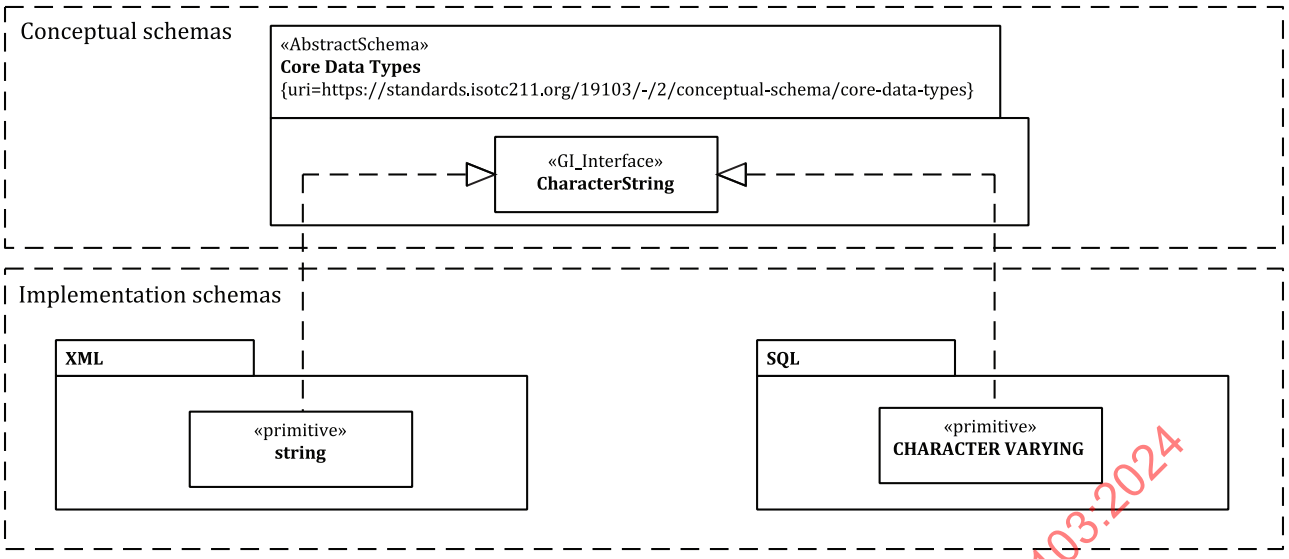


Figure 32 — Example of realizations of a core data type

8.2 Contents of the Core Data Types abstract schema

8.2.1 AnnualDate

AnnualDate (Figure 44, Table 7) represents a data type whose values are recurring positions in time (see 8.2.19) to a temporal resolution of a calendar day.

EXAMPLE 18 September.

8.2.2 AnnualMonth

AnnualMonth (Figure 44, Table 7) represents a data type whose values are recurring positions in time (see 8.2.19) to a temporal resolution of a calendar month.

EXAMPLE September.

8.2.3 Binary

Binary (Figure 33) represents a family of data types which are strings of bits (see 8.2.4), see ISO/IEC 11404:2007, 10.1.4.

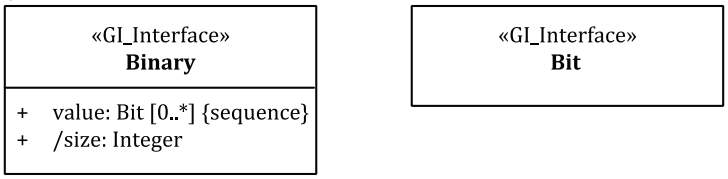


Figure 33 — Binary

EXAMPLE "01001010" is a binary of size 8.

8.2.4 Bit

Bit (Figure 34) represents the data type that has the two symbols designated 0 and 1 as value domain (see ISO/IEC 11404:2007, 10.1.3). Bit is the simplest element of a binary stream (see 8.2.3).

NOTE When encoded 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.

Figure 34 — Bit

EXAMPLE 1 0

EXAMPLE 2 1

8.2.5 Boolean

Boolean (Figure 35) is the mathematical data type associated with two-valued logic as defined in ISO/IEC 11404:2007, 8.1.1. Its value domain consists of the values true and false.

Figure 35 — Boolean

8.2.6 Character

Character (Figure 36) represents a family of data types whose value domains are character sets; see also ISO/IEC 11404:2007, 8.1.4.

The Internet Assigned Numbers Authority (IANA) maintains a register of character sets for use in the internet on https://www.iana.org/assignments/character-sets/character-sets (see also Figure 36 and Reference [39]). The register contains character sets defined by other processes and standards bodies, or specific profiles or combinations of such character sets (see RFC 2978[40]).

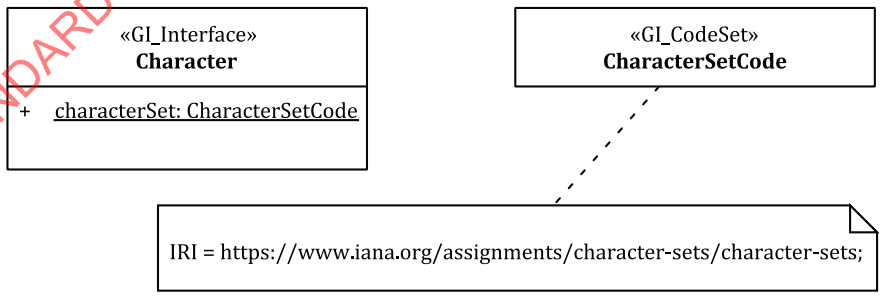


Figure 36 — Character

8.2.7 CharacterString

CharacterString (Figure 37) represents a family of data types which represent strings of symbols from standard character sets (see 8.2.6); see ISO/IEC 11404:2007, 10.1.5. Information about the natural language for interpreting the character strings can, if necessary, be stated at an appropriate level.

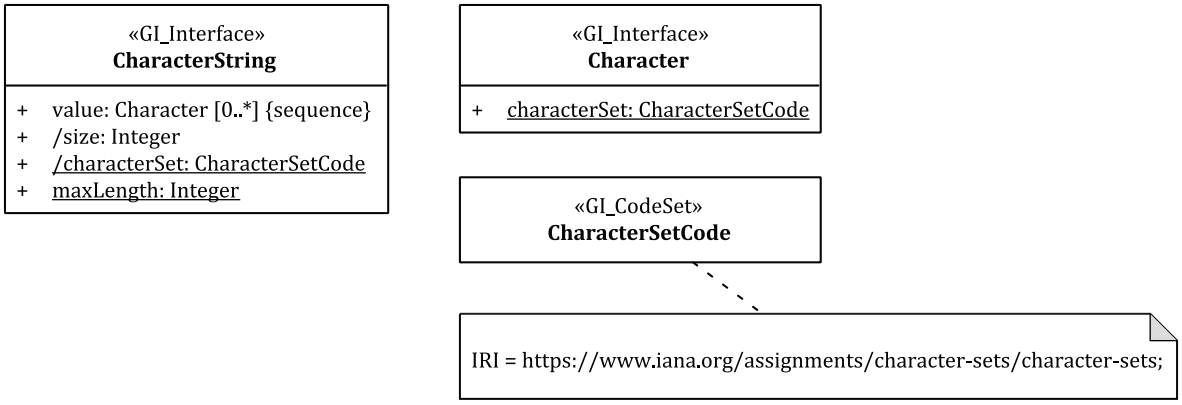


Figure 37 — CharacterString

EXAMPLE 1 “Ærlige Kåre så snø for første gang.” (in Norwegian).

EXAMPLE 2 “Honest Kåre saw snow for the first time.” (in English).

8.2.8 Date

Date (Figure 43, Table 7) represents what in ISO/IEC 11404:2007, 8.1.6, is described as a data type whose values are points in time (see 8.2.16) to a temporal resolution of a day.

EXAMPLE 1998-09-18.

8.2.9 DateTime

DateTime (Figure 43, Table 7) represents what in ISO/IEC 11404:2007, 8.1.6, is described as a data type whose values are points in time (see 8.2.16) to a temporal resolution of a second or a fraction of a second.

DateTime has properties timeShiftUtcHour, timeShiftUtcMinute, timeZone as defined in Table 7. In implementations, they may be set globally in, for example, the data specification, the conceptual schema or the metadata record for the data set. Consequently, these properties can become optional in a realization of DateTime.

EXAMPLE 1 1998-09-18T18:30:59+02:00.

EXAMPLE 2 1998-09-18T18:30:59+02:00 in the time zone identified by IANA time zone ID “Europe/Brussels”.

8.2.10 Decimal

Decimal (Figure 42) represents the mathematical data type whose value domain comprises the rational numbers (see 8.2.17) that can be expressed as a fraction p/q whose denominator q is a power of ten.

NOTE 1 A decimal is an exact value, it has a finite decimal representation. In comparison, a real is an approximate value.

NOTE 2 Decimals are useful for, for example, currencies and kilometre markers.

EXAMPLE $12.75 = \frac{1275}{100}$

8.2.11 Digit

Digit (Figure 38) represents the digits of a base 10 numbering scheme. The normal “Arabic” numerals are the common representation.

Figure 38 — Digit

EXAMPLE 1 2

EXAMPLE 2 9

8.2.12 Integer

Integer (Figure 42) represents the mathematical data type comprising the exact integral values. See ISO/IEC 11404:2007, 8.1.7 for further description.

EXAMPLE 1 29

EXAMPLE 2 −65547

8.2.13 IRI

An Internationalized Resource Identifier (IRI) (Figure 39) is a character string (see 8.2.7) that identifies a resource and that conforms to RFC 3987.[41] See RFC 3987 for more information and examples.

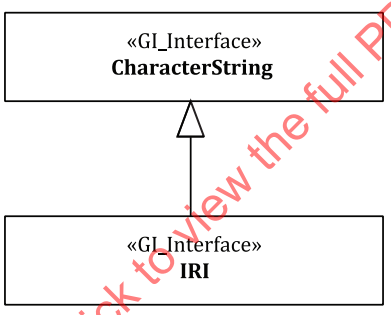


Figure 39 — IRI

8.2.14 Measure

Measure (Figure 40) represents a quantity value representing a measurement result (JCGM 200:2012, 2.10[42]).

UnitOfMeasure (Figure 40) represents a real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number (JCGM 200:2012, 1.9[42]).

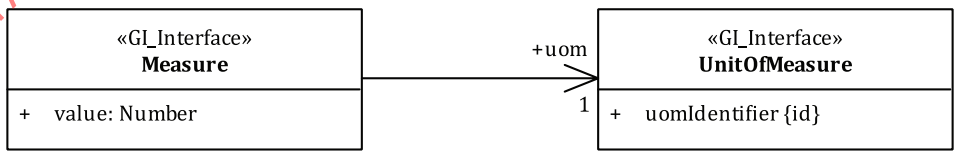


Figure 40 — Measure and UnitOfMeasure

A constraint (7.7) can specify an appropriate unit of measure (uom) for a property that has a measure (8.2.14) as its type.

Figure 41 gives an example of a constraint specifying an appropriate unit of measure. Class building has a constraint regarding the unit of measure for property height.

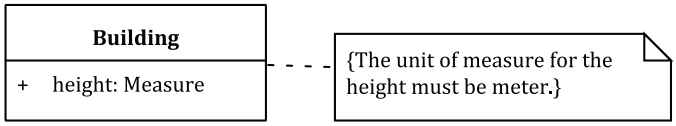


Figure 41 — Example of a constraint specifying an appropriate unit of measure

NOTE A measure can be a monetary measure.

8.2.15 Number

Number (Figure 42) represents 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.

Subtypes of Number are described in 8.2.12, 8.2.10, 8.2.17 and 8.2.18.

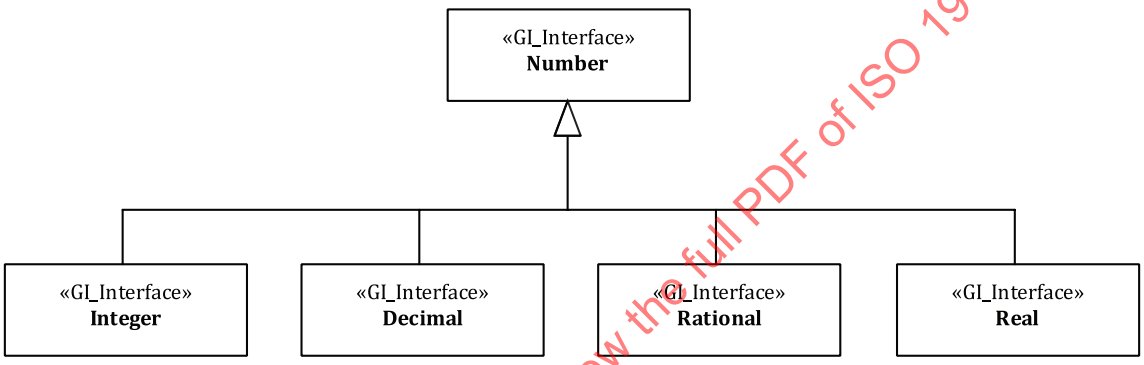


Figure 42 — Number and its subtypes

8.2.16 PositionInTime

PositionInTime (Figure 43) represents what in ISO/IEC 11404:2007, 8.1.6, is described as a data type whose values are points in time to a certain temporal resolution. The time scales used are the Gregorian calendar and the 24-hour clock system (see also ISO 34000).

NOTE Other time scales are out of scope of this document. Consequently, the temporal data types described in this document are not suited for certain domains such as geology and archaeology.

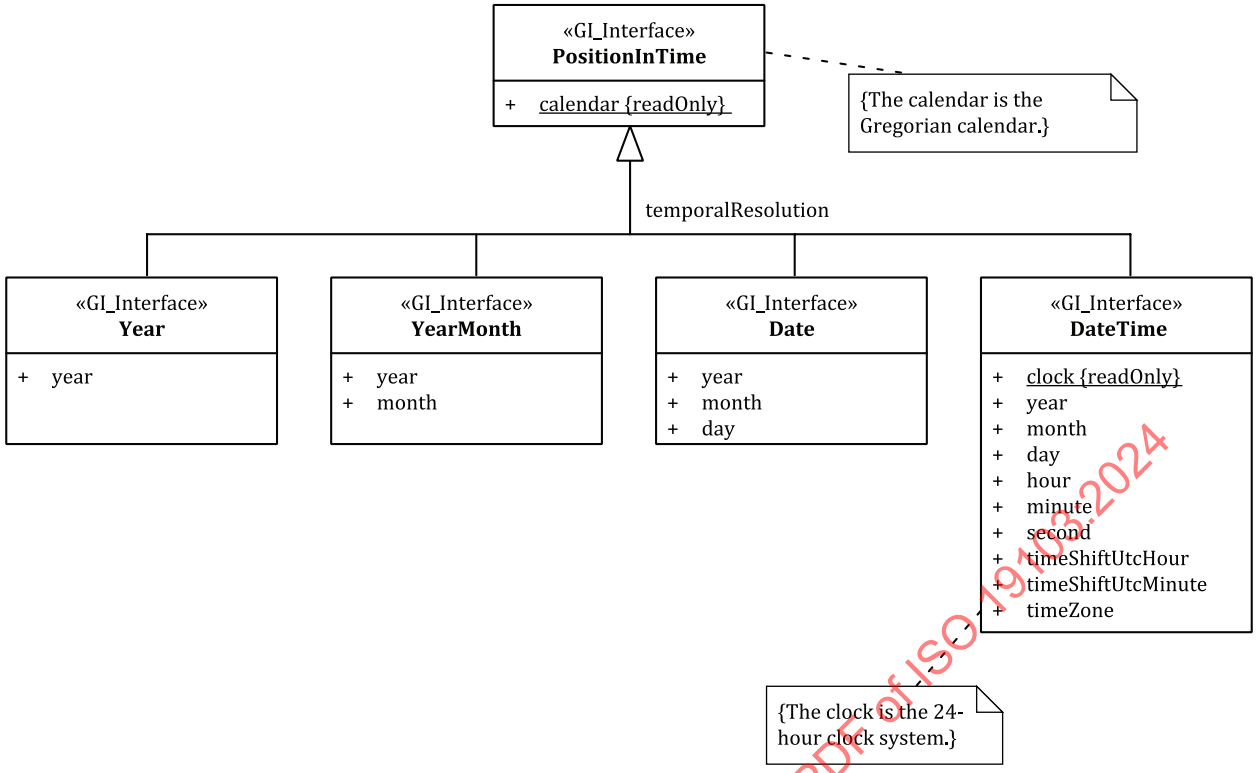


Figure 43 — Positions in time

The properties shown in Figure 43 are defined as in Table 7.

Table 7 — Properties of positions in times and recurring positions in times

Name	Designation	Definition	Description
calendar	calendar system	time scale that uses the time scale unit of calendar day as its basic unit	In this schema, the calendar system is the Gregorian calendar.
clock	clock system	time scale suited for intra-day time measurements	In this schema, the clock system is the 24-hour clock system, based on UTC.
year	calendar year	time scale unit defined by the calendar system	
month	calendar month	time scale unit resulting from a defined division of a calendar year, each containing a specific number of calendar days	In the Gregorian calendar, each calendar year is divided into 12 sequential calendar months, named January, February, March, April, May, June, July, August, September, October, November and December.
day	calendar day of month	ordinal number of a calendar day within a calendar month	In the Gregorian calendar, a calendar month consists of 28, 29, 30 or 31 days.
hour	clock hour	time scale unit whose duration represents one hour within the defined clock system	With the 24-hour clock system, each calendar day of month is divided into 24 clock hours.
minute	clock minute	time scale unit whose duration represents one minute within the defined clock system	With the 24-hour clock system, each clock hour is divided into 60 clock minutes.
second	clock second	time scale unit whose duration represents one second within the defined clock system	With the 24-hour clock system, each clock minute is divided into 59, 60 or 61 clock seconds, depending on whether there are leap seconds.
NOTE Most property designations and definitions are defined by ISO 34000.			

Table 7 *(continued)*

Name	Designation	Definition	Description
timeShiftUtcHour	hours of time shift against UTC	hours component of the difference between the marks attributed to the same instant between times of the time scale of the value and UTC	
timeShiftUtcMinute	minutes of time shift against UTC	minutes component of the difference between the marks attributed to the same instant between times of the time scale of the value and UTC	
timeZone	time zone	geographical region observing the same local time at any point in time	The two most common time zone information sources are the IANA Time Zone Database[44] and the set of Windows Time Zones.
NOTE Most property designations and definitions are defined by ISO 34000.			

Subclauses 8.2.8, 8.2.9, 8.2.25 and 8.2.26 give examples of instances of subtypes of PositionInTime. Most examples are in the form of date and time expressions using the extended format as defined in ISO 8601-1.

8.2.17 Rational

Rational (Figure 42) represents the mathematical data type comprising the rational numbers, see ISO/IEC 11404:2007, 8.1.8. A rational number is a number (see 8.2.15) that can be expressed as a fraction p/q where p and q are integers and $q \neq 0$. A rational number is said to have numerator p and denominator q, see also Reference [46].

EXAMPLE $\frac{37}{3}$

8.2.18 Real

Real (Figure 42) represents a family of data types which are computational approximations to the mathematical data type comprising the "real numbers". Specifically, each real data type designates a collection of mathematical real values which are expressed to some finite precision and are distinguishable to at least that precision. See ISO/IEC 11404:2007, 8.1.10 for further description.

NOTE Since real numbers can approximate any measure where absolute accuracy is not possible, this form of numeric is most often used for measures. In cases where absolute accuracy is needed, such as currencies, then a decimal representation can be preferred (assuming the currency is decimal, such as the US dollar or British pound). Where there are no subunits possible, integer numbers can be preferred.

EXAMPLE 1 23.501

EXAMPLE 2 –1234E-4

EXAMPLE 3 – 23.0

8.2.19 RecurringPositionInTime

RecurringPositionInTime (Figure 44, Table 7) represents a data type whose values are series of consecutive time intervals of identical duration. Subtypes of RecurringPositionInTime are described in 8.2.1, 8.2.2 and 8.2.21.

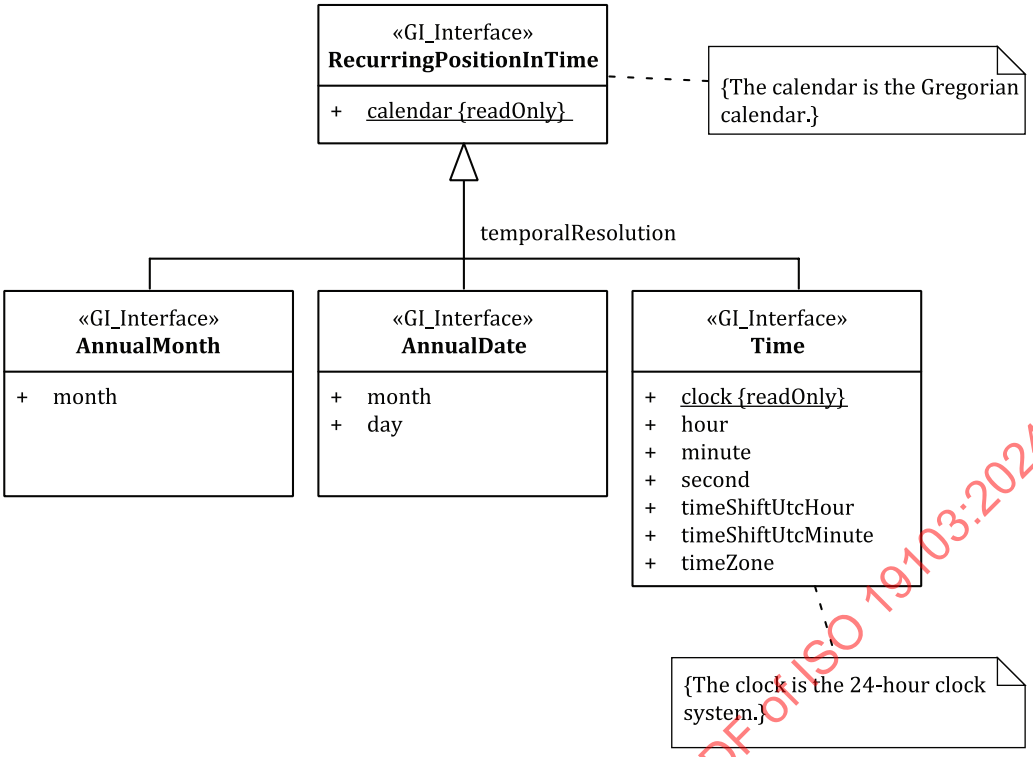


Figure 44 — Recurring positions in time

8.2.20 Sign

Sign (Figure 45) represents the signs "+" and "−" that are 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.

NOTE A sign is commonly represented by a single character such as "+" or "−" but can sometimes carry an integer 1 for emphasis such as "+1", or "−1". There is no semantic difference between these two presentation objects.

Figure 45 — Sign

EXAMPLE 1 +

EXAMPLE 2 −

8.2.21 Time

Time (Figure 44, Table 7) represents a data type whose values are recurring positions in time (see 8.2.19) to a temporal resolution of a second or a fraction of a second.

EXAMPLE 1 18:30:59

EXAMPLE 2 18:30:59+02:00 in the time zone identified by IANA time zone ID "Europe/Brussels".

8.2.22 URI

A Uniform Resource Identifier (URI) (Figure 46) is a character string that identifies a resource and that conforms to RFC 3986.[47] A URI is also an IRI (see 8.2.13). See RFC 3986 for more information and examples.

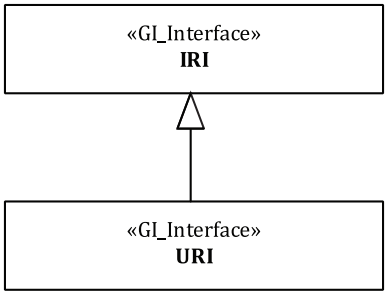


Figure 46 — URI

8.2.23 UUID

A Universally Unique Identifier (UUID) (Figure 47) is a 128-bit identifier that is unique across both space and time, with respect to the space of all UUIDs. The generation of UUIDs does not require a registration authority for each single identifier.

The UUID concept is defined in Reference [48]. Generation of UUIDs is specified in RFC 9562[49] or its successors.

NOTE RFC 9562 is related to ISO/IEC 9834-8, see RFC 9562, Clause 1 for more information. ISO/IEC 9834-8 is identical to ITU. X.667.[51]

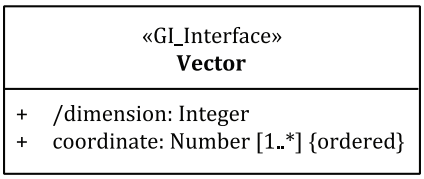
Figure 47 — UUID

EXAMPLE a2cd16ec-95c5-4c9e-968e-2bc3b068f98b (formatted as a string).

8.2.24 Vector

Vector (Figure 48) represents a quantity having direction as well as magnitude (ISO 19123-1:2023, 3.1.51). It is often represented as an ordered set of numbers called coordinates that represent a position in a coordinate system.

Vectors can be added together and multiplied by scalars. The reader is referred to reference works in mathematics for more information, e.g. Reference [53] and Reference [54].



NOTE Property “dimension” is a derived property. In an instance of a vector, the value of property “dimension” equals the cardinality of the value of property “coordinate”.

Figure 48 — Vector

EXAMPLE The vector in a three-dimensional space that goes from the origin to the point with coordinates (123, 514, 150) can be written as (123, 514, 150).

8.2.25 Year

Year (Figure 43, Table 7) represents what in ISO/IEC 11404:2007, 8.1.6, is described as a data type whose values are points in time (see 8.2.16) to a temporal resolution of a year.

EXAMPLE 1998

8.2.26 YearMonth

YearMonth (Figure 43, Table 7) represents what in ISO/IEC 11404:2007, 8.1.6, is described as a data type whose values are points in time (see 8.2.16) to a temporal resolution of a month.

EXAMPLE 1998-09

Annex A
(normative)

Abstract test suite

A.1 Overview of conformance classes

Clause A.2 contains the conformance class “Conceptual schemas modelled in UML”, which has the identifier /conf/conceptual-schema. See also 5.2.

A.2 Conceptual schemas modelled in UML

A.2.1 Abstract test case 1

Identifier	/conf/conceptual-schema/uml
Reference	/req/conceptual-schema/uml
Test method	Pass if the conceptual schema conforms to UML 2.5.1. NOTE Defining how to test whether a conceptual schema conforms to UML is out of scope for this document. UML 2.5.1 does not specify an abstract test suite this document can refer to. A tool for designing UML models can offer validation functionality.

A.2.2 Abstract test case 2

Identifier	/conf/conceptual-schema/level-of-abstraction
Reference	/req/conceptual-schema/level-of-abstraction
Test method	— Pass if the stereotype of the conceptual schema is the stereotype “AbstractSchema” of the UML profile https://standards.isotc211.org/19103/-/2/uml-profile. — Pass if the level of abstraction is documented in another way (manual inspection).

A.2.3 Abstract test case 3

Identifier	/conf/conceptual-schema/navigable-association-end-name
Reference	/req/conceptual-schema/navigable-association-end-name
Test method	Find all classifiers in the conceptual schema and find all binary associations connected to the classifiers. For each binary association, pass if each end that is navigable has a name defined.

A.2.4 Abstract test case 4

Identifier	/conf/conceptual-schema/generalizations
Reference	/req/conceptual-schema/generalizations
Test method	Pass if each generalization connects two classifiers that are instances of the same class from the UML metamodel. NOTE Many tools that are specifically designed for dealing with UML models ensure this already.

A.2.5 Abstract test case 5

Identifier	/conf/conceptual-schema/dependencies
Reference	/req/conceptual-schema/dependencies
Test method	Find all classifiers in the conceptual schema, find all their properties, find all the types of these properties, and find the conceptual schemas in which these types are defined and that are different from the conceptual schema under test. Pass if a dependency is present from the conceptual schema under test to the found conceptual schemas.

A.2.6 Abstract test case 6

Identifier	/conf/conceptual-schema/directed-acyclic-graph
Reference	/req/conceptual-schema/directed-acyclic-graph
Test method	Find all the conceptual schemas that the conceptual schema depends on directly and indirectly by searching through the conceptual schema's dependencies recursively. Construct a directed graph where each node represents a conceptual schema and each directed edge represents a dependency. Pass if no cycles are detected within the dependency graph that involve the conceptual schema under test.

A.2.7 Abstract test case 7

Identifier	/conf/conceptual-schema/natural-language-constraints
Reference	/req/conceptual-schema/natural-language-constraints
Test method	Pass if all constraints in the conceptual schema are expressed in natural language (manual inspection).

A.2.8 Abstract test case 8

Identifier	/conf/conceptual-schema/uml-profile-application
Reference	/req/conceptual-schema/uml-profile-application
Test method	Pass if the conceptual schema uses the stereotypes of the profile with identifier https://standards.isotc211.org/19103/-/2/uml-profile when appropriate and in accordance with any tagged values and constraints defined for the stereotypes.

A.2.9 Abstract test case 9

Identifier	/conf/conceptual-schema/unique-names
Reference	/req/conceptual-schema/unique-names
Test method	Find all classifiers in the conceptual schema. Pass if all the following are true: — each classifier has a name that is case-insensitive unique in the context of the package in which it is contained; — each property has a name that is case-insensitive unique in the context of the classifier in which it is contained.

A.2.10 Abstract test case 10

Identifier	/conf/conceptual-schema/package-dependency-diagram
Reference	/req/conceptual-schema/package-dependency-diagram
Test method	Take all dependencies found in /conf/conceptual-schema/dependencies. Pass if each dependency is present in the same package diagram.

A.2.11 Abstract test case 11

Identifier	/conf/conceptual-schema/context-diagram
Reference	/req/conceptual-schema/context-diagram
Test method	Pass if each classifier and its properties are documented in a dedicated class diagram. NOTE Using a naming pattern for context diagrams can aid automated verification. For example, “Context diagram <classifier name>”.

A.2.12 Abstract test case 12

Identifier	/conf/conceptual-schema/core-data-types
Reference	/req/conceptual-schema/core-data-types
Test method	Pass if the conceptual schema does not use types that are not the core data types but are equivalent to them.

Annex B
(informative)

Backward compatibility

A conceptual schema that conforms to ISO 19103:2015 does not automatically conform to this document.

Table B.1 describes the changes that can cause a conceptual schema to no longer conform to this document as well as the changes that deprecate certain practices. For each change, a suggestion is made for how to update a conceptual schema to make it conformant to this document.

Table B.2 provides a structured overview of the mapping between the provisions in this document and the provisions in ISO 19103:2015. The mapping is not necessarily exactly one-to-one.

Table B.1 — Backward compatibility overview

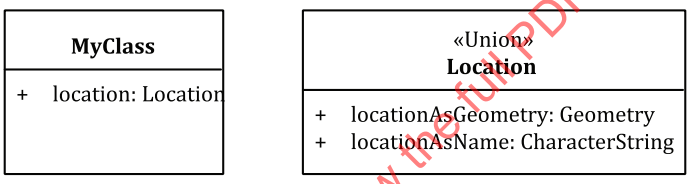
Description of change	Suggested action
Introduction of dedicated tagged values for containing the designations, definitions and descriptions of the concepts described in a conceptual schema.	Apply the stereotypes defined in 7.8 to the model elements. Move the designations, definitions and descriptions from the tool-specific documentation fields to the dedicated tagged values. NOTE 1 A copy of the contents of the tagged values can be added, in an automated way, to the tool-specific documentation fields, if representations of the conceptual schema depend on those fields to be filled out.
Deprecation of the Leaf stereotype.	Remove the Leaf stereotype from the packages that have it.
Deprecation of the Union stereotype. The Union stereotype extends more than one meta-class, (see Figure 24 and compare with ISO 19103:2015, Figure D.2). This is because the previous editions of this document defined this stereotype in an ambiguous way, which resulted in implementations taking different approaches. Figure B.1 a) gives an example of a classifier with the Union stereotype.	Two options exist. If the original intention was to have a property whose type is one from a set of mutually exclusive types: — update the properties who had the union as a type so they either have no type anymore (see 7.3.2.2) or have a type that is a common supertype of all the mutually exclusive types; — remove the union; — add a constraint specifying what type the property is allowed to have. Figure B.1 b) gives an example of the application of this option on the model given in Figure B.1 a). If the original intention was to have several properties, of which only one can be present in an instance: — replace the property that uses the union with the same number of attributes as defined in the union and give them each a multiplicity 0..1; — remove the union; — add a constraint specifying that those properties are mutually exclusive. Figure B.1 c) gives an example of the application of this option on the model given in Figure B.1 a).

Table B.1 *(continued)*

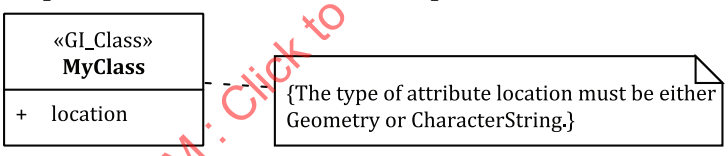
Description of change	Suggested action
Deprecation of the CodeList stereotype. The CodeList stereotype extends more than one meta-class, see Figure 24 and compare with ISO 19103:2015, Figure D.2. This is because the previous editions of this document defined this stereotype in an ambiguous way, which resulted in implementations taking different approaches.	Remove the CodeList stereotype and the tagged value codeList from the elements that have the stereotype. Remove the code values from the element, if present. If the schema is not supposed to specify what code set to use, then do all the following: — remove the element completely; — model the properties who had it as type without a type. If the schema is supposed to specify which code set to use, then do all of the following: — change the element to a data type if it is a class or an enumeration; — apply stereotype GI_CodeSet to the element; — update the tagged values of the element to indicate which code set is to be used and where it can be found. Annex H elaborates on code sets.
Removal of the data type Any (see ISO 19103:2015, 7.6).	Model the property without a type, see 7.3.2.2.
Removal of the data types Set, Sequence and Bag (see ISO 19103:2015, 7.3).	For each property with data type Set, Sequence or Bag, do all of the following: — change the property to a multivalued property; — change the type of the property to an appropriate data type; — set metaproperties “isUnique” and “isOrdered” to the correct values, see Table 2. EXAMPLE 1 Replace “position: Set<DirectPosition>” with “position: DirectPosition [1..*]” or “position: DirectPosition [1..*] {unordered, unique}”. EXAMPLE 2 Replace “coordinate: Sequence<Number>” with “coordinate: Number [0..*] {sequence}”, “coordinate: Number [0..*] {seq}” or “coordinate: Number [0..*] {ordered, nonunique}”. See Table D.1 for more information about the notation for multivalued properties.
Removal of the name types (see ISO 19103:2015, 7.5).	Search for the name data types in the package with identifier https://standards.isotc211.org/harmonized-model/common-types, see 7.11.3.
Removal of the extension data types (see ISO 19103:2015, Annex C).	Search for the extension data types in the package with identifier https://standards.isotc211.org/harmonized-model/common-types, see 7.11.3.

Table B.1 *(continued)*

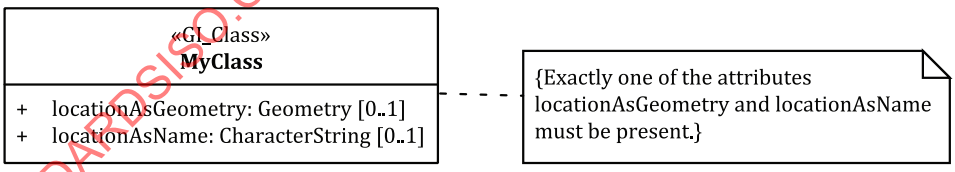
Description of change	Suggested action
Removal of the possibility to create a conforming conceptual schema by means of a mapping from another language conceptual schema language or from an older version of UML. Older conceptual schemas based on UML 1.4.2, also known as ISO/IEC 19501:2005, can contain classes using the type stereotype, defined in UML 2.5.1, 22.3 as "stereotype of class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects". Figure B.2 a) gives an example of such a class. NOTE 2 See also ISO 19103:2015, 4.36 and Annex B, and ISO/TS 19103:2005, 6.3, D.3 and D.8.3. NOTE 3 The type stereotype is present in both ISO/IEC 19501:2005 and UML 2.5.1. A type can be realized by one or more implementation classes, see UML 2.5.1, Clause 22 and Reference [27]. Both the type and the implementation class stereotype are predefined standard stereotypes, defined in the Standard Profile of UML.	If the conceptual schema is based on ISO/IEC 19501:2005, do all of the following: a) change all the classes with the type stereotype to interfaces, and apply stereotype GI_Interface; b) change all the realizations that have one of the classifiers that were changed in list item a) as source and that have an interface as target to generalizations. Figure B.2 b) gives an example of the result of the application of this action on Figure B.2 a).



a) Example of a classifier with the deprecated Union stereotype

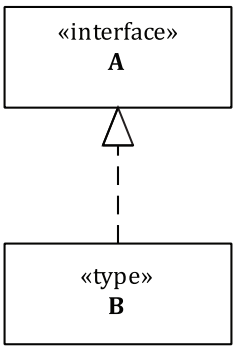


b) Example of a property whose type is one from a set of mutually exclusive types



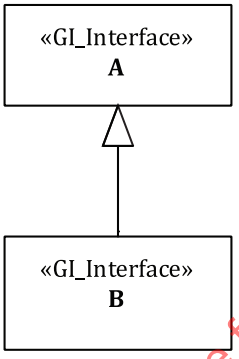
c) Example of several properties of which only one can be present in an instance

Figure B.1 — Example of the possible options to deal with the deprecation of the Union stereotype



NOTE B is a class with the “type” stereotype applied to it. The notation for classifiers is summarized in D.1.1.

a) Example of a type



b) Example of an interface converted from a type

Figure B.2 — Example of how to upgrade a conceptual schema containing types

Table B.2 — Provision mapping

Provision in ISO 19103:2015	Provision in this document	Note
Requirement 1	Requirement 1	
Requirement 2	—	This document addresses conceptual schemas modelled in UML, not conceptual schemas modelled in another conceptual schema modelling language.
Requirement 3	Recommendation 7	
Requirement 4	Requirement 2	
Requirement 5	—	
Requirement 6	—	
Requirement 7	Requirement 8, Recommendation 7	
Requirement 8	—	UML 2.5.1 defines an enumeration, 7.8 defines a data type with the GI_CodeSet stereotype, and Annex H explains what code sets are in general. This information provides the basis for the choice of the appropriate model element.
Requirement 9	—	UML 2.5.1 defines an enumeration, 7.8 defines a data type with the GI_CodeSet stereotype, and Annex H explains what code sets are in general. This information provides the basis for the choice of the appropriate model element.

Table B.2 *(continued)*

Provision in ISO 19103:2015	Provision in this document	Note
Requirement 10	Recommendation 2	The default multiplicity of 1..1 applies to all properties, thus to attributes and to association ends.
Requirement 11	Requirement 3	
Requirement 12	—	See 7.4.2.4.
Requirement 13	—	See 7.4.2.4.
Requirement 14	Requirement 4	
Requirement 15	Requirement 8	
Requirement 16	Requirement 9, Recommendation 11	
Requirement 17	Requirement 10	
Requirement 18	Requirement 11	
Requirement 19	Recommendation 7	
Requirement 20	—	See Annex G.
Requirement 21	Requirement 10	
Requirement 22	Requirement 12	
Requirement 23	—	This document addresses only conceptual schemas modelled in accordance with UML 2.5.1, not with earlier versions of UML.
Requirement 24	—	This document addresses only conceptual schemas modelled in accordance with UML 2.5.1, not with earlier versions of UML.
Requirement 25	Recommendation 19	
Recommendation 1	—	See 7.2.4.
Recommendation 2	Recommendation 7	
Recommendation 3	—	Rules for user extensions of code sets are specified by the owner of the code set. The management of code sets is not a standardization target of this document.
Recommendation 4	Requirement 8	
Recommendation 5	Recommendation 14	
Recommendation 6	Recommendation 4	
Recommendation 7	Recommendation 5	
Recommendation 8	Recommendation 8	
Recommendation 9	Recommendation 10	
Recommendation 10	Recommendation 11	
Recommendation 11	Recommendation 12	
Recommendation 12	Recommendation 7	
Recommendation 13	Recommendation 13	
Recommendation 14	Requirement 7, Recommendation 6	
Recommendation 15	—	Good practices regarding how to make readable and understandable diagrams can be found in other resources addressing UML in general.
Recommendation 16	Recommendation 18	
—	Requirement 5	
—	Requirement 6	
—	Recommendation 1	

Table B.2 *(continued)*

Provision in ISO 19103:2015	Provision in this document	Note
—	Recommendation 3	
—	Recommendation 6	
—	Recommendation 9	
—	Recommendation 15	
—	Recommendation 16	
—	Recommendation 17	

Annex C
(informative)

On conceptual schema languages

A conceptual schema classifies instances, identifying kinds of instances according to their characteristics (structure and behaviour) and relations with other instances.

Figure C.1 describes the relationship between modelling reality and the resulting conceptual schema. A universe of discourse is a selected piece of the real world (or a hypothetical world) that a human being wishes to describe in a model. The universe of discourse can include not only phenomena such as watercourses, lakes, islands, boundaries of pieces of land or owners of pieces of land, but also their characteristics and the relations that exist among such phenomena. A universe of discourse is defined 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 is called an application schema. A conceptual schema also defines relations between model elements and constraints on the elements and their relations. These constraints define valid states of the modelled system. An actual universe of discourse can be digitally represented by data that is structured according to the specification in the corresponding conceptual schema.

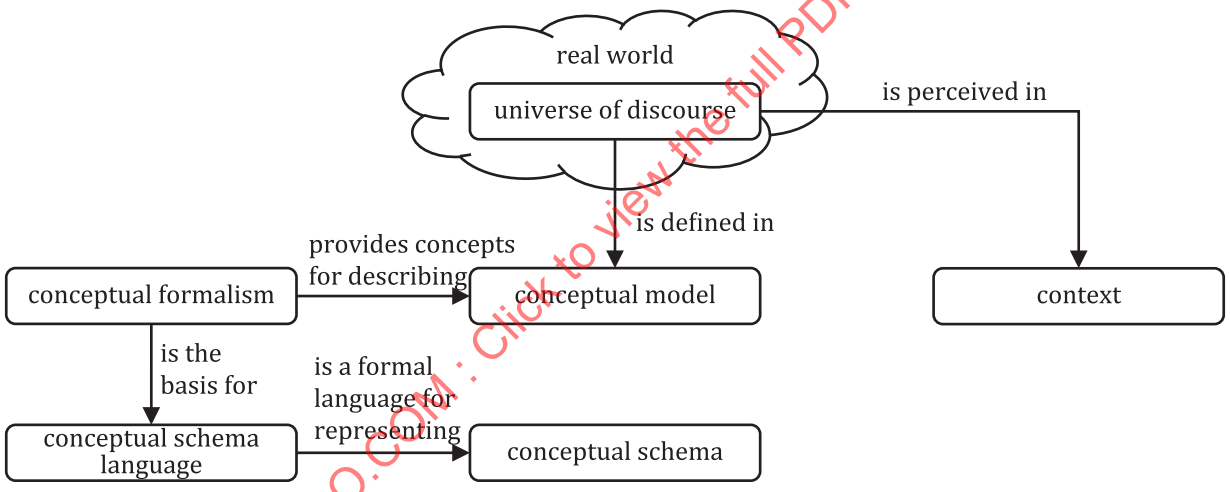


Figure C.1 — From reality to conceptual schema

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 can 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.

Annex D
(informative)

UML notation reference

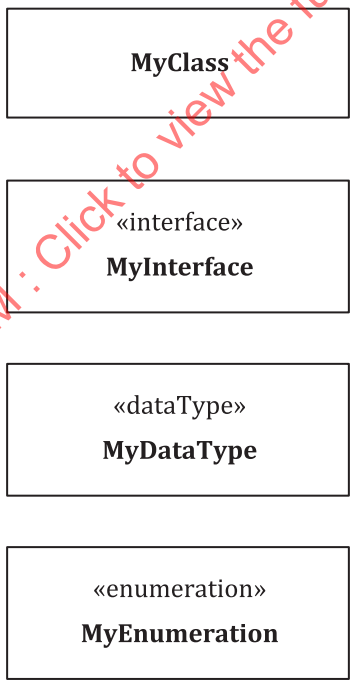
D.1 Notation for UML elements used for modelling conceptual schemas

D.1.1 Notation for classifiers

The default notation for a classifier in a UML diagram is a solid-outline rectangle containing the classifier's name. If the default notation is used for a classifier, a keyword corresponding to the metaclass of the classifier is shown in guillemets («keyword») above the name. No keyword is needed to indicate that the metaclass is class. This default notation is illustrated in Figure D.1.

When a stereotype is applied to a model element, the name of the stereotype is shown as well, either enclosed in a separate pair of guillemets or in the same pair of guillemets as the keyword. This is illustrated for classifiers in Figure D.2 and explained in Clause D.3.

More information about the notation for classifiers, data types (including enumerations), interfaces and classes is given in UML 2.5.1, 9.2.4.1, 10.2.4, 10.4.4, 11.4.4, respectively. The notation for model elements that have a stereotype applied is specified in UML 2.5.1, 12.3.4.



NOTE In this figure, the attributes and operations compartments are suppressed.

Figure D.1 — Default notation for classifiers used in geographic information modelling without a stereotype

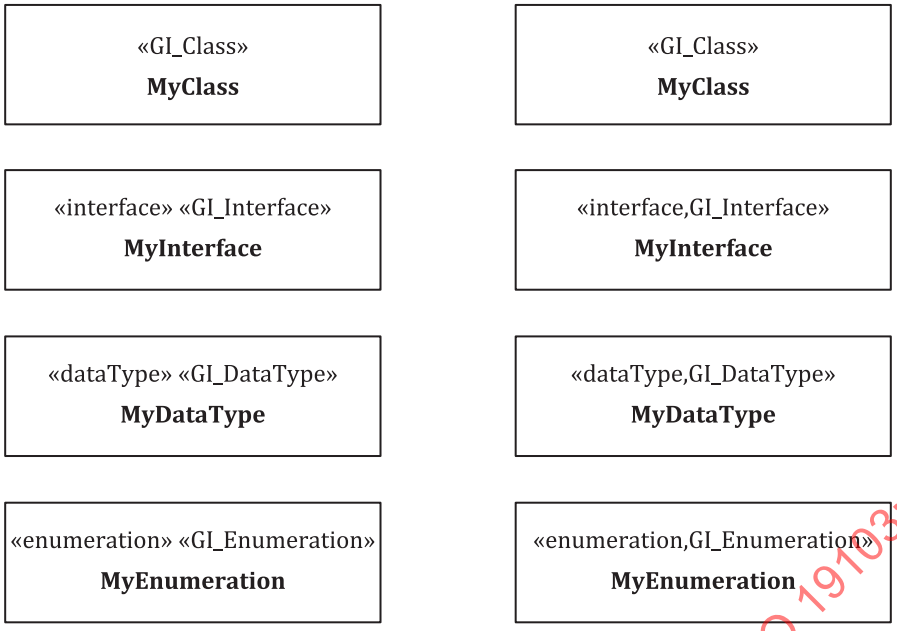


Figure D.2 — Two possible default notations for classifiers used in geographic information modelling with a stereotype

NOTE The tool used to create all the UML diagrams in this document except Figure D.1, Figure D.2 and Figure D.7, does not fully conform to UML 2.5.1 regarding the notation for stereotyped classifiers, shown in Figure D.2. In particular, the keywords "interface" and "dataType" are missing in the diagrams created by that tool in the case of interfaces and data types, respectively. The purpose of Figure D.2 is to demonstrate the possible, correct notations.

D.1.2 Notation for properties

The notation for properties is defined in UML 2.5.1, 9.5.4. UML 2.5.1, 9.5.5 gives examples. Figure 7, Figure 8 and Figure 10 give examples as well.

The notation for a property optionally ends with one or several property modifiers. Such a property modifier can be used with multivalued properties to indicate the kind of collection, see 7.3.2.3. Table D.1 lists the property modifiers used for the four kinds of collections described in Table 2.

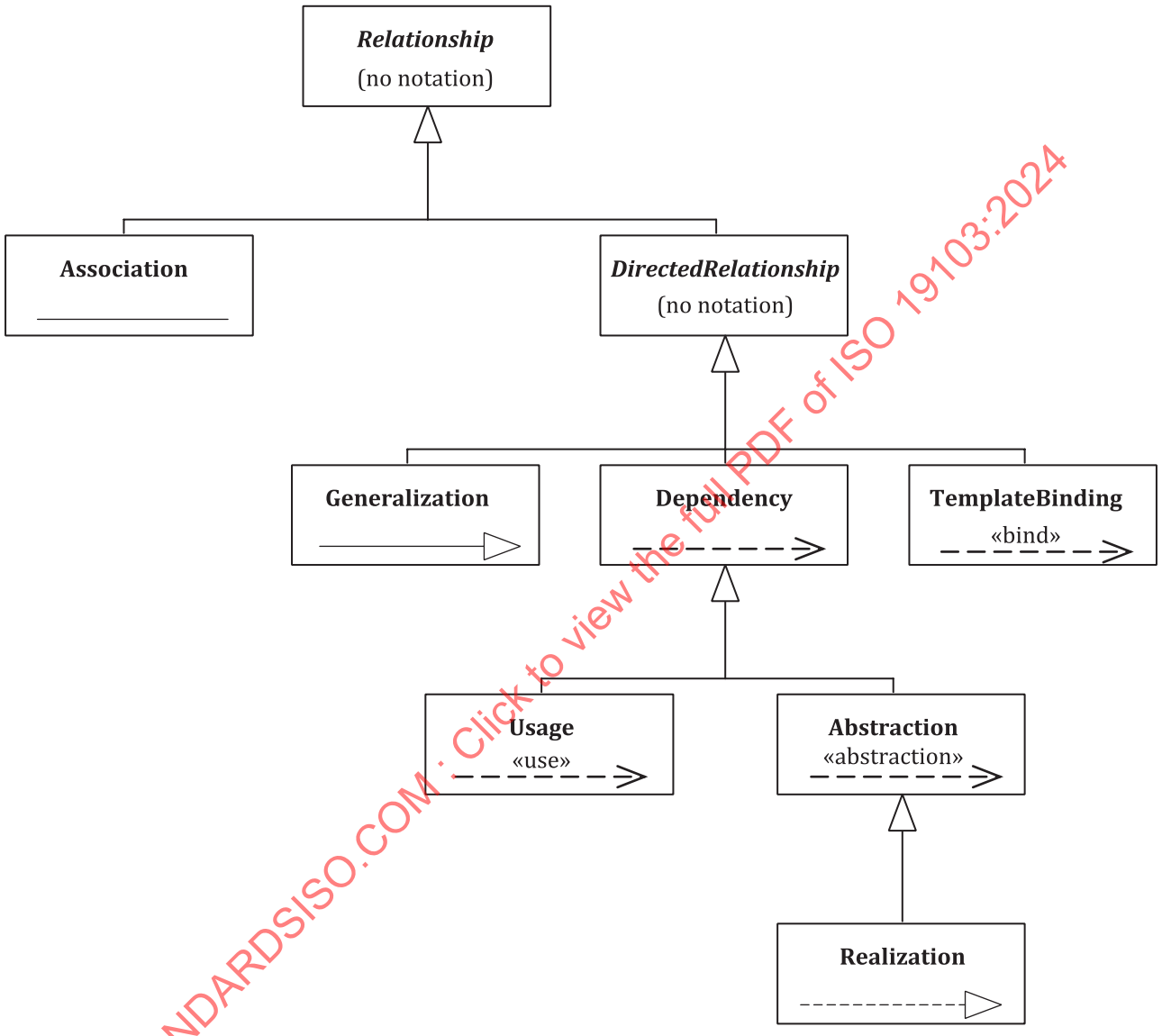
Table D.1 — Notation for the different kinds of collections

Kind of collection	Property modifiers
set	no modifier or {unordered, unique}
ordered set	{ordered} or {ordered, unique}
bag (= multiset)	{nonunique} or {unordered, nonunique}
sequence (= list = ordered bag)	{seq} or {sequence} or {ordered, nonunique}
NOTE 1 Metaproperty "isUnique" (see 7.3.2.3) is by default set to true in the UML metamodel, therefore the property modifier "unique" can be omitted.	
NOTE 2 Metaproperty "isOrdered" (see 7.3.2.3) is by default set to false in the UML metamodel, therefore the property modifier "unordered" can be omitted.	
NOTE 3 A UML tool typically provides controls in its graphical user interface to set the metaproperties "isUnique" and "isOrdered". The tool then gives the property one of the notations described in this table based on the values of those metaproperties.	
NOTE 4 In UML 2.2, the property modifier for a bag was changed from {bag} to {nonunique}. Not all UML tools conform to this new notation yet.	

D.1.3 Notation for relationships

Figure D.3 illustrates the notation defined in UML 2.5.1 for associations, generalizations, dependencies, usages, abstractions, realizations and template bindings.

The reader is referred to UML 2.5.1, 7.3.4, 7.7.4, 9.2.4.2 and 11.5.4 for more information about the notation for template bindings, dependencies (including usages, abstractions and realizations), generalizations and associations, respectively.



NOTE 1 Relationship and DirectedRelationship are abstract metaclasses and have therefore no notation.

NOTE 2 "use", "abstraction" and "bind" are keywords, not stereotypes names; see Clause D.3.

NOTE 3 The tool used to create the UML diagrams in this document does not show template bindings correctly regarding the arrowhead: they are displayed with a hollow arrowhead (like realizations) instead of an open arrowhead. Earlier versions of the UML specification were inconsistent regarding this notation. See e.g. https://issues.omg.org/issues/UML25-479.

Figure D.3 — Relationships used in geographic information modelling and their notation

Associations symbols can have several adornments:

— names;

— visibility symbols;

— multiplicities;

— property modifiers (see D.1.2);

— diamond (hollow or filled).

The notation for associations is shown in Figure D.4. See UML 2.5.1, 11.5.4 for more information about the notation for associations.

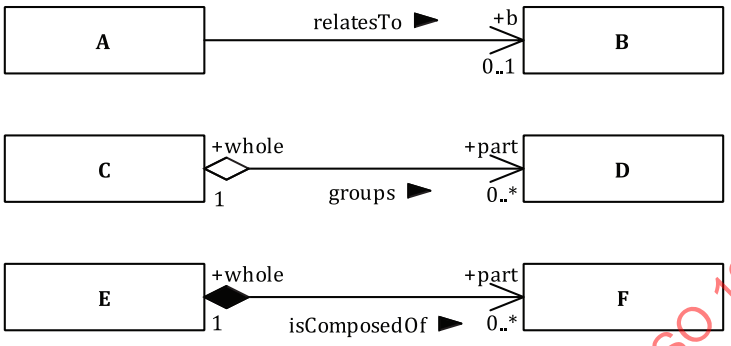


Figure D.4 — Notation for associations

D.1.4 Notation for comments

A comment is shown as a rectangle with the upper right corner bent. This rectangle is also known as a note symbol. The note symbol contains the body of the comment. The connection to each element that is annotated by the comment is shown by a separate dashed line. This is stated in UML 2.5.1, 7.2.4 and illustrated in Figure D.5.

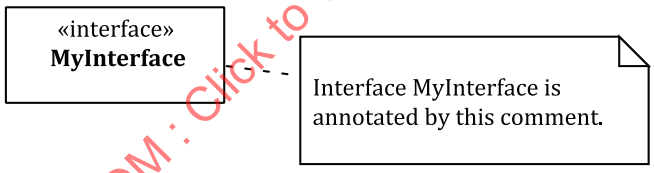


Figure D.5 — Notation for comments

D.1.5 Notation for constraints

Constraints can be shown in different ways, depending on the model elements they constrain (see UML 2.5.1, 7.6.4). The notation used in the figures in this document consists of placing the constraint expression within curly brackets and placing the resulting character string in a note symbol that is attached by dashed lines to the constrained elements. If the constraint has a name, the constraint name and a colon are placed before the constraint expression. This notation is illustrated in Figure D.6.

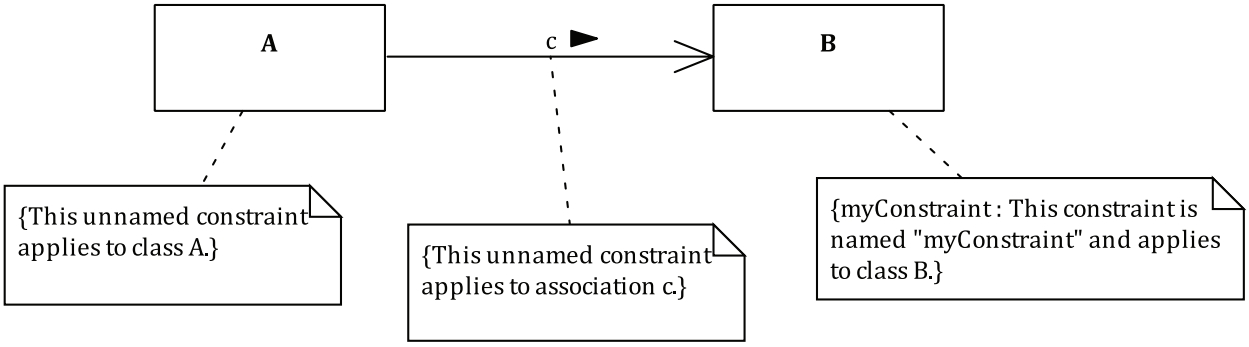


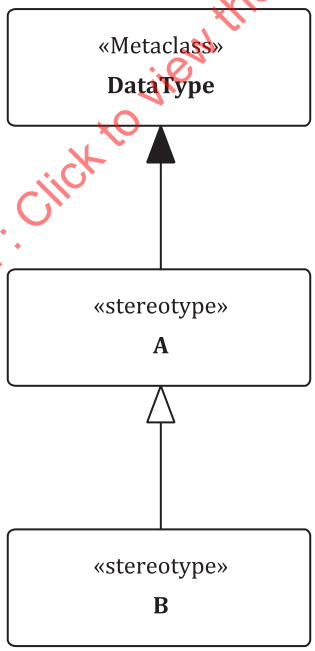
Figure D.6 — Notation for constraints

D.2 Notation for UML elements used for modelling profiles

A profile uses the same notation as package, with the addition that the keyword “profile” is shown before or above the name of the package enclosed in guillemets (UML 2.5.1, 12.3.4).

A stereotype can extend a metaclass from the UML metamodel, as exemplified in Figure D.7. A stereotype uses the same notation as a class, with the addition that the keyword “stereotype” is shown before or above the name of the class, enclosed in guillemets (UML 2.5.1, 12.3.4). «Metaclass» is shown before or above the name of the class that represent a metaclass from the UML metamodel (UML 2.5.1, 11.4.4 and 22.3).

Figure D.8 shows the relationships that can be used in profiles. Stereotype A extends metaclass DataType. Stereotype B specializes stereotype A.



NOTE The tool used to create all the UML diagrams in this document except Figure D.1, Figure D.2 and Figure D.7 shows the stereotype of a class that represent a metaclass from the UML metamodel as «metaclass» instead of «Metaclass». See also Figure 20, Figure 21, Figure 22, Figure 23 and Figure 24.

Figure D.7 — Notation for metaclasses and stereotypes in UML profiles

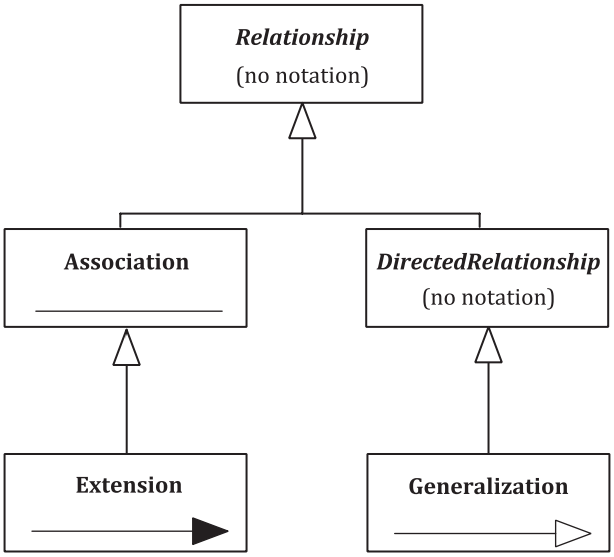


Figure D.8 — Relationships used in profiles and their notation

D.3 Difference between UML keywords and stereotypes

Keywords and stereotypes are two different concepts in UML, although both are shown in guillemets, as explained in the following quotation ("[...]" means that some text that is not important here has been left out).

> UML keywords are reserved words that are an integral part of the UML notation and normally appear as text annotations attached to a UML graphic element or as part of a text line in a UML diagram.
>
> In UML, keywords are used for three different purposes:
>
> a) To distinguish a particular UML concept (metaclass) from others sharing the same general graphical form. For instance, the «interface» keyword in the header box of a classifier rectangle is used to distinguish an Interface from other kinds of Classifiers.
>
> b) To distinguish a particular kind of relationship between UML concepts (meta-association) from other relationships sharing the same general graphical form. For example, dashed lines between elements are used for a number of different relationships, including Dependencies, [...], and so on.
>
> c) To specify the value of some modifier attached to a UML concept (meta-attribute value). [...]
>
> Keywords are always enclosed in guillemets («keyword»), which serve as visual cues to more readily distinguish when a keyword is being used.
>
> [...]
>
> In addition to identifying keywords, guillemets are also used to distinguish the usage of stereotypes. This means that:
>
> — Not all words appearing between guillemets are necessarily keywords, and
>
> — words appearing in guillemets do not necessarily represent stereotypes.
>
> [SOURCE: UML 2.5.1, Annex C]

The keywords for the model elements of the UML metamodel are listed in UML 2.5.1, Annex C. The keywords for the model elements that are relevant for this document are listed in Table D.2.

Table D.2 — Keywords for UML model elements used in geographic information modelling

UML metaclass	Keyword
Abstraction	abstraction
Class	(no keyword) (in a model); Metaclass (in a profile)
DataType	dataType
Enumeration	enumeration
Interface	interface
Profile	profile
Stereotype	stereotype
TemplateBinding	bind
Usage	use