



**International
Standard**

ISO 19103

**Geographic information —
Conceptual schema language**

Information géographique — Langage de schéma conceptuel

**Second edition
2024-09**

Reference number
ISO 19103:2024(en)

Full document details: [ISO 19103-2024](#)

© ISO 2024  **Documents**
Document Sharing Hub



COPYRIGHT PROTECTED DOCUMENT

© ISO 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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	v
Introduction	vi
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Symbols and abbreviated terms	11
5 Conformance	12
5.1 Conformance overview	12
5.2 Conceptual schemas modelled in UML	12
6 Overview	12
7 Use of UML	13
7.1 General use of UML	13
7.2 Classifiers	15
7.2.1 General	15
7.2.2 Classes	16
7.2.3 Data types	16
7.2.4 Enumerations	17
7.2.5 Interfaces	18
7.3 Features	19
7.3.1 General	19
7.3.2 Properties	19
7.3.3 Operations	23
7.4 Relationships	23
7.4.1 General	23
7.4.2 Associations	23
7.4.3 Generalizations	25
7.4.4 Realizations	25
7.4.5 Template bindings	26
7.5 Packages	26
7.6 Comments	28
7.7 Constraints	28
7.8 UML profile	28
7.9 Naming provisions	35
7.10 Diagrams	38
7.10.1 General	38
7.10.2 Package diagrams	39
7.10.3 Class diagrams	39
7.11 Reusable types	40
7.11.1 General	40
7.11.2 Core data types	40
7.11.3 Common types	40
8 Core data types	40
8.1 General	40
8.1.1 Relation with ISO/IEC 11404	40
8.1.2 Modelling choice for the core data types	42
8.2 Contents of the Core Data Types abstract schema	44
8.2.1 AnnualDate	44
8.2.2 AnnualMonth	44
8.2.3 Binary	44
8.2.4 Bit	45
8.2.5 Boolean	45
8.2.6 Character	45

8.2.7	CharacterString.....	45
8.2.8	Date.....	46
8.2.9	DateTime.....	46
8.2.10	Decimal.....	46
8.2.11	Digit.....	46
8.2.12	Integer.....	47
8.2.13	IRI.....	47
8.2.14	Measure.....	47
8.2.15	Number.....	48
8.2.16	PositionInTime.....	48
8.2.17	Rational.....	50
8.2.18	Real.....	50
8.2.19	RecurringPositionInTime.....	50
8.2.20	Sign.....	51
8.2.21	Time.....	51
8.2.22	URI.....	51
8.2.23	UUID.....	52
8.2.24	Vector.....	52
8.2.25	Year.....	53
8.2.26	YearMonth.....	53
Annex A	(normative) Abstract test suite.....	54
Annex B	(informative) Backward compatibility.....	57
Annex C	(informative) On conceptual schema languages.....	63
Annex D	(informative) UML notation reference.....	64
Annex E	(informative) Differences between UML 2.5.1 and UML 2.4.1.....	71
Annex F	(informative) Mapping between ISO 19103 and ISO/IEC 11404 data types.....	72
Annex G	(informative) Conceptual schema representations.....	75
Annex H	(informative) Code sets.....	76
Bibliography		86

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.iso211.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 metamodel. Models, metamodels and metamodels reside on different levels of “meta”.

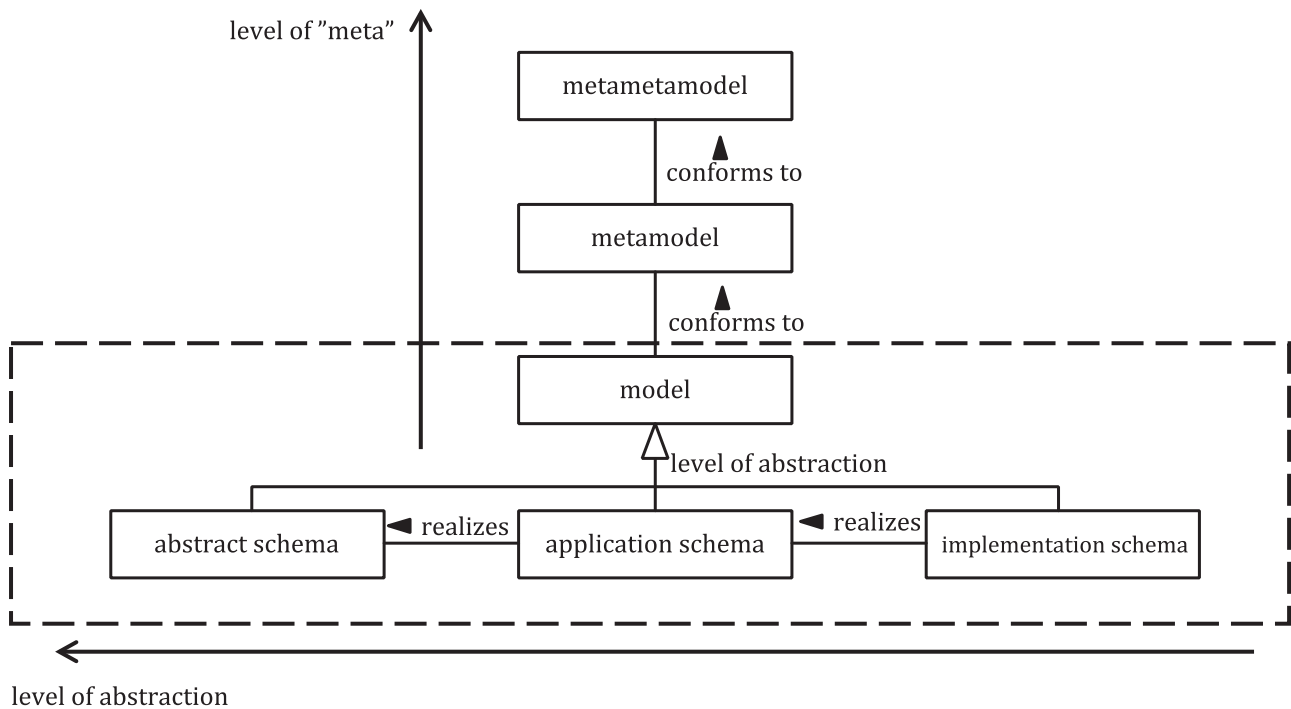


Figure 1 — Levels of abstraction versus levels of “meta”

[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.

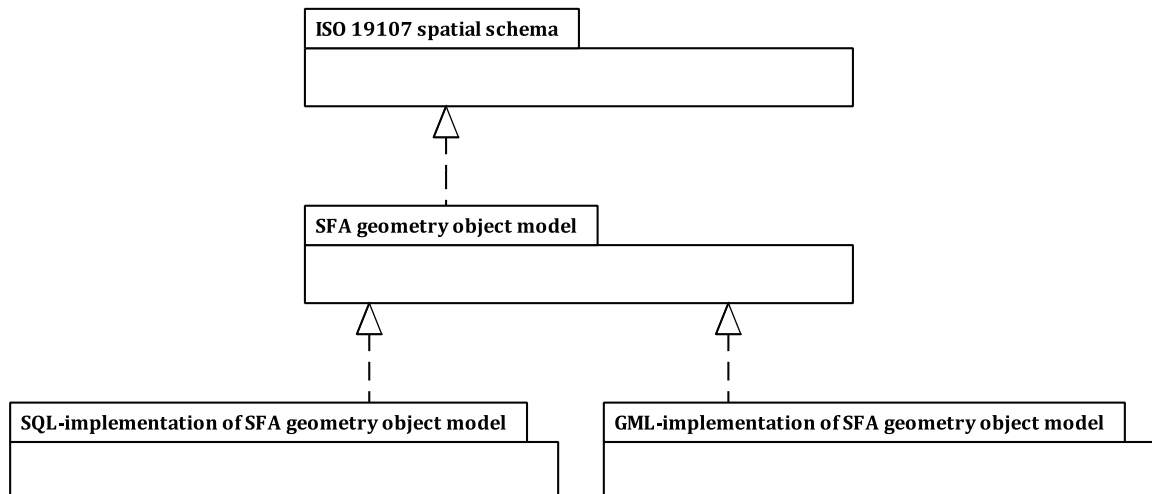


Figure 2 — Example of schemas at different levels of abstraction

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](#).

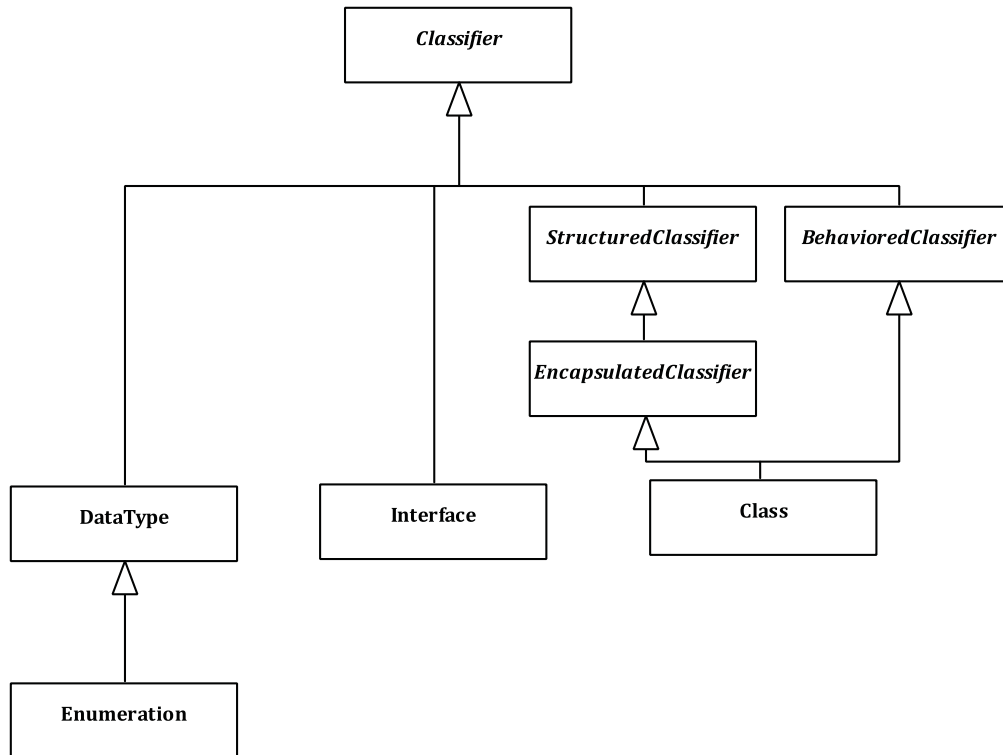


Figure 3 — Classifiers from the UML metamodel that are used in geographic information modelling

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.

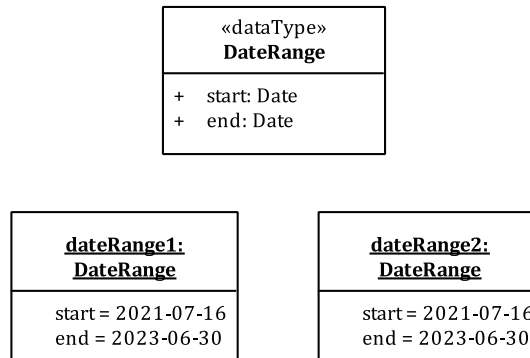


Figure 4 — Example of equality of data values

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.

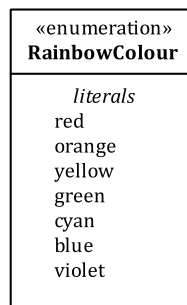


Figure 5 — Example of an enumeration with enumeration literals

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.

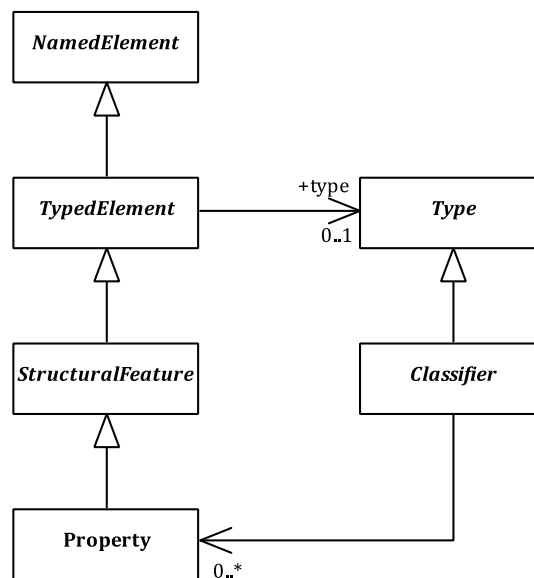


Figure 6 — Type and Property, from the UML metamodel

[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.

Observation
+ id: CharacterString
+ result
+ ...

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.

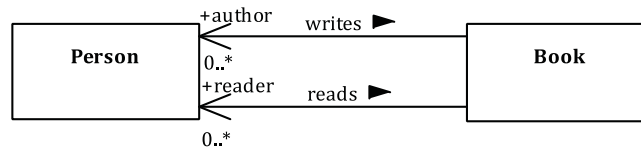
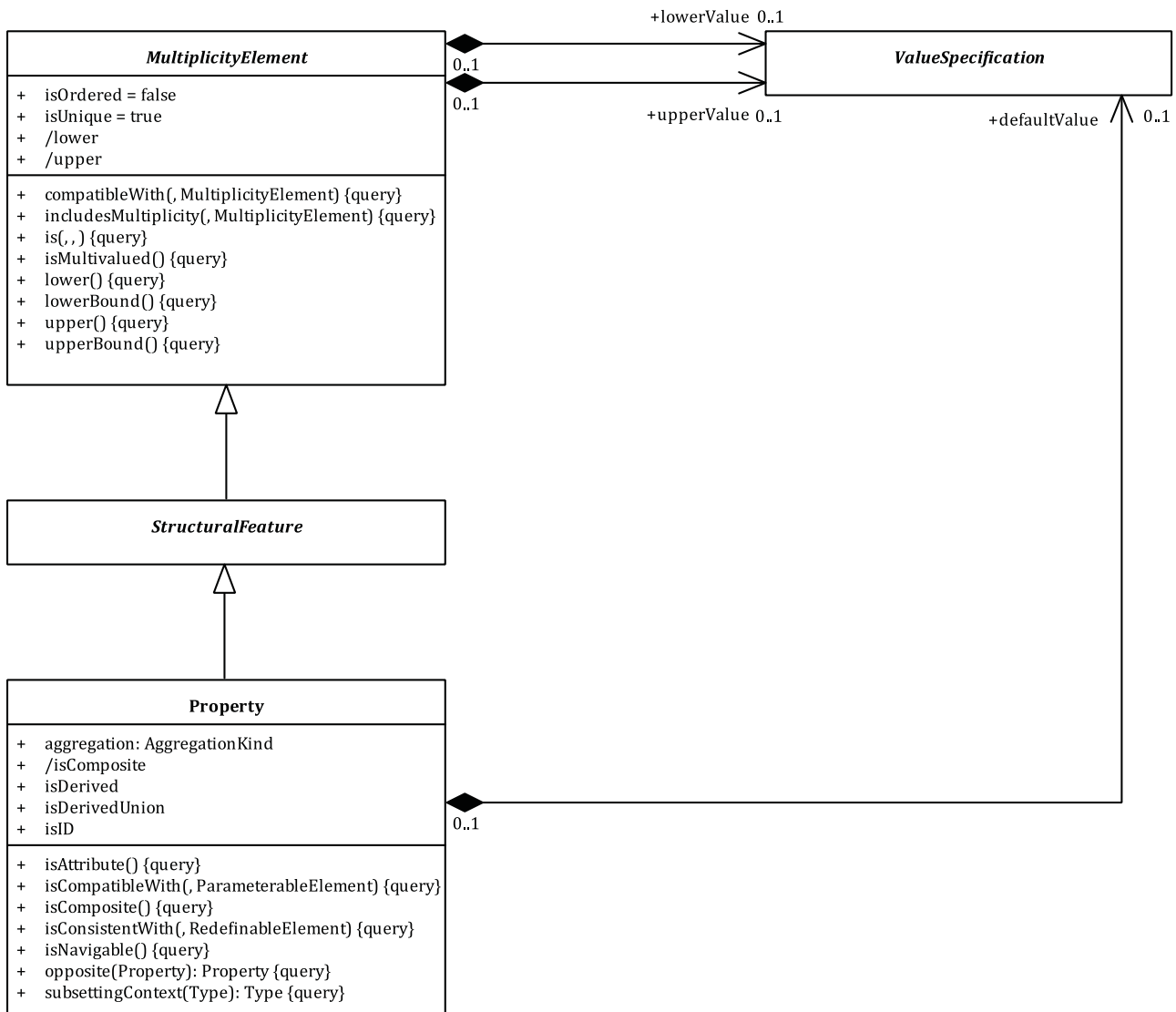


Figure 8 — Example of association ends and their type

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.

Book
+ isbn: CharacterString {id}
+ title: CharacterString
+ ...

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.

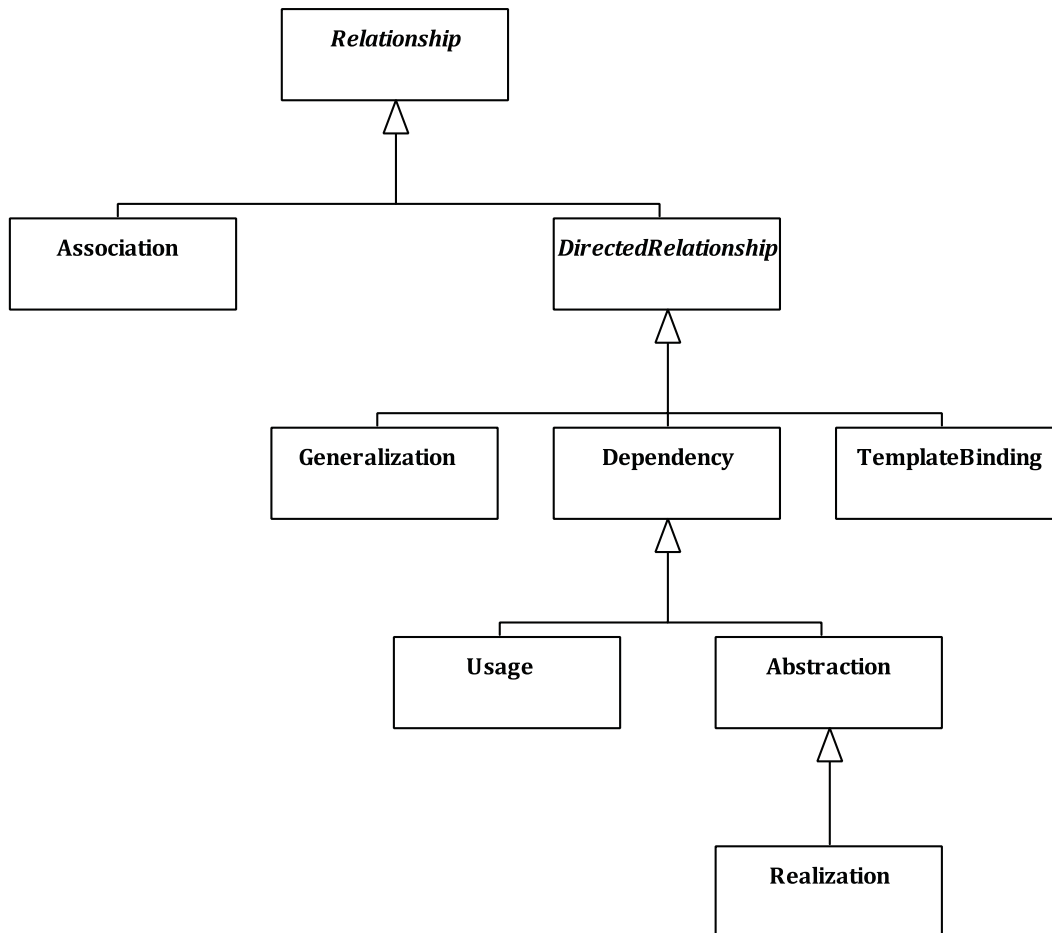


Figure 11 — Relationships from the UML metamodel that are used in geographic information modelling

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;