TECHNICAL SPECIFICATION

ISO/TS 19103

First edition
2005-07-15

Geographic information — Conceptual schema language

Information géographique — Schéma de language conceptuel

Reference number
ISO/TS 19103:2005(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web 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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50 % of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed after three years in order to decide whether it will be confirmed for a further three years, revised to become an International Standard, or withdrawn. If the ISO/PAS or ISO/TS is confirmed, it is reviewed again after a further three years, at which time it must either be transformed into an International Standard or be withdrawn.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

IISO/TS 19103 was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics*.

Introduction

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

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

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

Geographic information — Conceptual schema language

1 Scope

This Technical Specification provides rules and guidelines for the use of a conceptual schema language within the ISO geographic information standards. The chosen conceptual schema language is the Unified Modeling Language (UML).

This Technical Specification provides a profile of the Unified Modeling Language (UML) for use with geographic information. In addition, it provides guidelines on how UML should be used to create standardized geographic information and service models.

2 Conformance

Any conceptual schema written for a specification, including a profile or functional standard, that claims conformance with this Technical Specification shall pass all of the requirements described in the abstract test suite in Annex A. Non-UML schemas shall be considered conformant if there is a well-defined mapping from a model in the source language into an equivalent model in UML and that this model in UML is conformant.

3 Normative references

The following referenced documents are indispensable for the application 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.

ISO 19101:2002, *Geographic Information — Reference model*

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

4 Terms, definitions and abbreviations

4.1 ISO/TS 19103 terms

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

4.1.1
application
manipulation and processing of data in support of user requirements

[ISO 19101]

4.1.2
application schema
conceptual schema for data required by one or more **applications**

[ISO 19101]

4.1.3
conceptual model
model that defines concepts of a universe of discourse

[ISO 19101]

4.1.4
conceptual schema
formal description of a **conceptual model**

[ISO 19101]

4.1.5
data type
specification of a **value domain** with **operations** allowed on **values** in this **domain**

EXAMPLE Integer, Real, Boolean, String, Date and SG Point (conversion of data into a series of codes).

NOTE Data types include primitive predefined types and user-definable types.

4.1.6
domain
well-defined set

NOTE Domains are used to define the domain set and range set of attributes, operators and functions.

4.1.7
feature
abstraction of real world phenomena

[ISO 19101]

NOTE 1 A feature may occur as a type or an instance. Feature type or feature instance should be used when only one is meant.

NOTE 2 In UML [8] a feature is a property, such as operation or attribute, which is encapsulated as part of a list within a classifier, such as interface, class or data type.

4.1.8
feature association
relationship that links instances of one **feature** type with instances of the same or a different **feature** type

[ISO 19109]

NOTE 1 A feature association may occur as a type or an instance. Feature association type or feature association instance is used when only one is meant.

NOTE 2 Feature associations include aggregation of features.

4.1.9
feature attribute
characteristic of a **feature**

[ISO 19101]

NOTE 1 A feature attribute has a name, a data type, and a value domain associated to it. A feature attribute for a feature instance also has an attribute value taken from the value domain.

NOTE 2 A feature attribute may occur as a type or an instance. Feature attribute type or feature attribute instance should be used when only one is meant.

4.1.10
feature operation
operation that every **instance** of a **feature** type may perform

[ISO 19110]

EXAMPLE 1 An operation upon a “dam” is to raise the dam. The result of this operation is to raise the level of water in a reservoir.

EXAMPLE 2 An operation by a “dam” might be to block vessels from navigating along a “watercourse”.

NOTE Feature operations provide a basis for feature type definition.

4.1.11
metadata
data about data

[ISO 19115]

4.1.12
metadata element
discrete unit of **metadata**

[ISO 19115]

NOTE 1 Metadata elements are unique within a metadata entity.

NOTE 2 Equivalent to an attribute in UML terminology.

4.1.13
schema
formal description of a model

[ISO 19101]

4.1.14
service
distinct part of the functionality that is provided by an entity through interfaces

[ISO/IEC TR 14252]

4.1.15
value domain
set of accepted values

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

4.2 UML terms

The following are UML terms that are adapted from ISO/IEC 19501.

4.2.1
actor
coherent set of roles that users of use cases play when interacting with these use cases

NOTE An actor may be considered to play a separate role with regard to each use case with which it communicates.

4.2.2
aggregation
special form of **association** that specifies a whole-part **relationship** between the aggregate (whole) and a **component** part

NOTE See composition.

4.2.3
association
semantic **relationship** between two or more **classifiers** that specifies connections among their **instances**

NOTE A binary association is an association among exactly two classifiers (including the possibility of an association from a classifier to itself).

4.2.4
attribute
feature within a **classifier** that describes a range of **values** that **instances** of the **classifier** may hold

NOTE 1 An attribute is semantically equivalent to a composition association; however, the intent and usage is normally different.

NOTE 2 "Feature" used in this definition is the UML meaning of the term and is not meant as defined in 4.1 of this Technical Specification.

4.2.5
behaviour
observable effects of an **operation** or event, including its results

4.2.6
cardinality
number of elements in a set

NOTE Contrast: multiplicity.

4.2.7
class
description of a set of **objects** that share the same **attributes**, operations, **methods**, **relationships** and semantics

NOTE A class may use a set of interfaces to specify collections of operations it provides to its environment. See: interface.

4.2.8
classifier
mechanism that describes behavioural and structural **features**

NOTE Classifiers include interfaces, classes, datatypes, and components.

4.2.9
component
modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of **interfaces**

NOTE A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files.

4.2.10
composition
form of **aggregation** which requires that a part **instance** be included in at most one composite at a time, and that the composite **object** is responsible for the creation and destruction of the parts

NOTE Parts with non-fixed multiplicity may be created after the composite itself, but once created they live and die with it (i.e. they share lifetimes). Such parts can also be explicitly removed before the death of the composite. Composition may be recursive. Synonym: composite aggregation.

4.2.11
constraint
semantic condition or restriction

NOTE Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML. See: tagged value, stereotype.

4.2.12
dependency
relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element)

4.2.13
generalization
taxonomic **relationship** between a more general element and a more specific element that is fully consistent with the more general element and contains additional information

NOTE An instance of the more specific element may be used where the more general element is allowed. See: inheritance.

4.2.14
inheritance
mechanism by which more specific elements incorporate structure and **behaviour** of more general elements related by **behaviour**

NOTE See generalization.

4.2.15
instance
entity that has unique identity, a set of **operations** can be applied to it, and state that stores the effects of the **operations**

NOTE See: object.

4.2.16
interface
named set of **operations** that characterize the **behaviour** of an element

4.2.17
metamodel
model that defines the language for expressing a model

4.2.18
method
implementation of an **operation**

NOTE It specifies the algorithm or procedure associated with an operation.

4.2.19
multiplicity
specification of the range of allowable cardinalities that a set may assume

NOTE Multiplicity specifications may be given for roles within associations, parts within composites, repetitions and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers. Contrast: cardinality.

4.2.20
object
entity with a well-defined boundary and identity that encapsulates state and **behaviour**

NOTE State is represented by attributes and relationships, behaviour is represented by operations, methods and state machines. An object is an instance of a class. See: class, instance.

4.2.21
operation
service that can be requested from an **object** to affect **behaviour**

NOTE 1 An operation has a signature, which may restrict the actual parameters that are possible.

NOTE 2 Definition from UML Reference Manual: A specification of a transformation or query that an object may be called to execute.

NOTE 3 An operation has a name and a list of parameters. A method is a procedure that implements an operation. It has an algorithm or procedure description.

4.2.22
package
general purpose mechanism for organizing elements into groups

NOTE Packages may be nested within other packages. Both model elements and diagrams may appear in a package.

4.2.23
refinement
relationship that represents a fuller **specification** of something that has already been specified at a certain level of detail

NOTE For example, a design class is a refinement of an analysis class

4.2.24
relationship
semantic connection among model elements

NOTE Kinds of relationships include association, generalization, metarelationship, flow and several kinds grouped under dependency.

4.2.25
specification
declarative description of what something is or does

NOTE Contrast: implementation.

4.2.26
stereotype
new type of modeling element that extends the semantics of the **metamodel**

NOTE Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of three extensibility mechanisms in UML. The others are constraint and tagged value.

4.2.27
tagged value
explicit definition of a property as a name-**value** pair

NOTE In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML; others may be user defined. Tagged values are one of three extensibility mechanisms in UML. The others are constraint and stereotype.

4.2.28
type
stereotyped class that specifies a **domain** of **objects** together with the **operations** applicable to the **objects**, without defining the physical implementation of those **objects**

NOTE A type may have attributes and associations.

4.2.29
value
element of a type **domain**

NOTE 1 A value may consider a possible state of an object within a class or type (domain).

NOTE 2 A data value is an instance of a data type, a value without identity.

4.3 Abbreviations

API	Application Programming Interface
CASE	Computer Aided Software Engineering
CORBA	Common Object Request Broker Architecture
CSL	Conceptual schema language
CSMF	Conceptual Schema Modeling Facility
DCOM/OLE	Distributed Compound Object Model/Object Linking and Embedding
GFM	General Feature Model
OCL	Object Constraint Language
ODMG	Object Database Management Group
OMG	Object Management Group
ODP	Open Distributed Processing
ODBC	Open Database Connection
SRS	Spatial Reference System
UML	Unified Modeling Language
url	Uniform Resource Locator
XML	Extended Markup Language
XMI	XML Metamodel Interchange

5 Organization

This Technical Specification contains a UML Profile which provides modeling guidelines for how to use UML for modeling compliant with the ISO geographic information standards.

The main technical content of this Technical Specification is found in Clause 6. An introduction to the general usage of UML is given in 6.1 and 6.2. The description of classes and attributes in 6.3 and 6.4 is based on general rules for UML. Data types described in 6.5 are developed for this Technical Specification, as standard UML does not prescribe the use of specific data types. More information on the necessary precision level of UML models required by this Technical Specification is provided in 6.6, 6.7 and 6.8. The conventions for defining optional attributes and associations are described in 6.9. Naming rules are described in 6.10.

Annex A describes an abstract test suite for checking that UML models are made according to the rules of this Technical Specification.

An introduction to conceptual schema languages can be found in Annex B, and a set of modeling guidelines for both information modeling and service modeling can be found in Annex C. The general UML as defined in ISO/IEC 19501 is briefly described in Annex D.

6 The ISO/TS 19103 UML Profile

6.1 Introduction

This clause provides rules and guidelines on the use of UML within the field of geographic information. It defines specific aspects for the use of UML compliant with the ISO geographic information standards. It is based on general UML as defined in ISO/IEC 19501 (Annex D). Annex D follows the same structure as this clause, to make it easy to refer to the relevant standard UML concepts.

The subclauses are structured as follows:

— General usage of UML

— Classes

— Attributes

— Data types

— Operations

— Relationships and associations

— Stereotypes and tagged values

— Optional, conditional and mandatory attributes and associations

— Naming and name spaces

— Packages

— Notes

— Constraints

— Documentation of models

6.2 General usage of UML

UML (The Unified Modeling Language) shall be used in a manner that is consistent with ISO/IEC 19501.

NOTE Books, such as “UML User Guide” [1] and “UML Reference Manual” [2] contain further information. The book “UML Distilled” [4] is a shorter introductory text.

All normative class models shall contain complete definitions of attributes, associations, operations and appropriate data type definitions. Other kinds of UML diagrams may also be used as normative models. It is the need for this completeness that makes it necessary to define this UML profile, as UML in general can be used on various levels of precision and completeness.

6.3 Classes

A class according to this Technical Specification is normally viewed as a specification and not as an implementation. Attributes are considered abstract and do not have to be directly implemented (i.e. as fields in a record or instance variables in an object). This is not in conflict with the process of encoding as described in ISO 19118, as this describes an external representation that does not have to be equivalent to the internal representation.

For this reason, conceptual schemas adhering to this profile of UML shall stereotype all classifiers as <<Type>> unless they specifically intend to specify internal representation.

NOTE Classifiers specified as <<DataType>> (lacking identity) are not usually assumed to be types in UML, but may be assumed to be abstract for conceptual schemas adhering to this profile of UML unless the schema in which they are defined specifically states otherwise.

For each class defined according to this Technical Specification, the set of attributes defined with this class, together with the sets of attributes of classes that are reachable directly or indirectly via associations, shall be sufficient to fully support the implementation of each operation defined for this particular class.

Use of multiple inheritance is an issue in many implementation environments and can cause problems if handled improperly in those environments. For this reason, the use of multiple inheritance should be minimized or avoided unless it is a fundamental part of the semantics of the type hierarchy. If used, the schema adhering to this UML profile shall not require that the implementation classes realizing the standards specified types replicate the inheritance model of the types.

6.4 Attributes

Attributes are used according to standard UML (see D.4).

6.5 Data types

6.5.1 General considerations

The classes and templates (parameterized types) defined in this clause are those that are usually defined by the development environment's data definition language. Each of these types can be represented in a variety of logically equivalent forms. The ones presented here are not meant to restrict the usage of other equivalent forms native to the chosen development environment. ISO/IEC 11404 presents an equivalent definition for most of the types and templates presented here.

The basic data types have been grouped into three categories, as shown in Figure 1:

a) Primitive types: Fundamental types for representing values, examples are CharacterString, Integer, Boolean, Date, Time, etc.

b) Implementation and collection types: Types for implementation and representation structures, examples are Names and Records, and types for representing multiple occurrences of other types, examples are Set, Bag and Sequence.

c) Derived types: Measure types and units of measurement.

The basic types are defined as abstract types, class name in italic, and appropriate representations will be defined by implementation and encoding mappings for the various subtypes.

The repertoire of basic data types is described in 6.5.2 – 6.5.7.

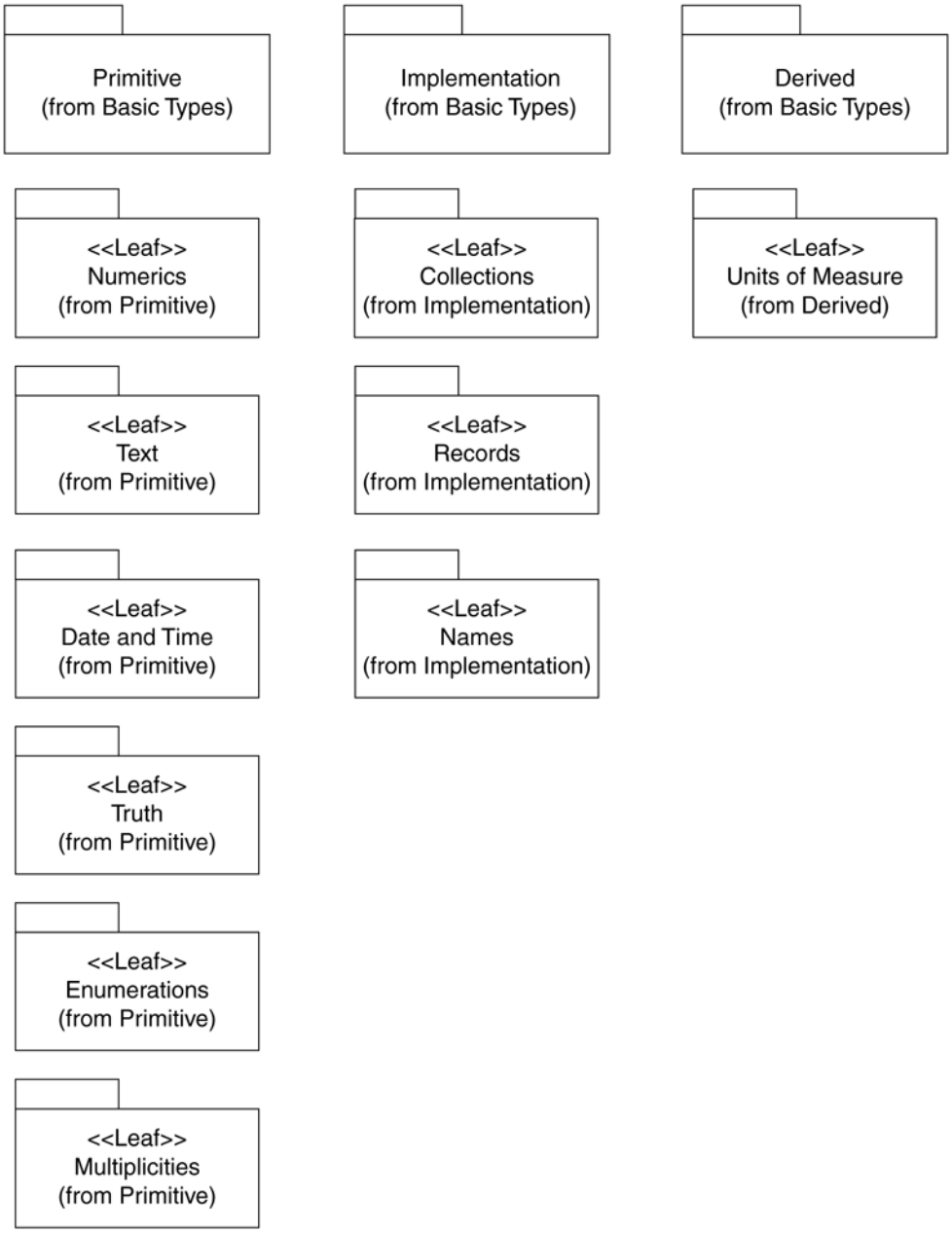


Figure 1 — The basic data types

6.5.2 Primitive types

6.5.2.1 General

Figure 2 provides an overview of the primitive types to be described next. Figure 3 provides an overview of the numeric types. Together with the class diagrams in the accompanying figures and the text of those figures, the content in each of the packages is described in 6.5.2.1 – 6.5.2.14.

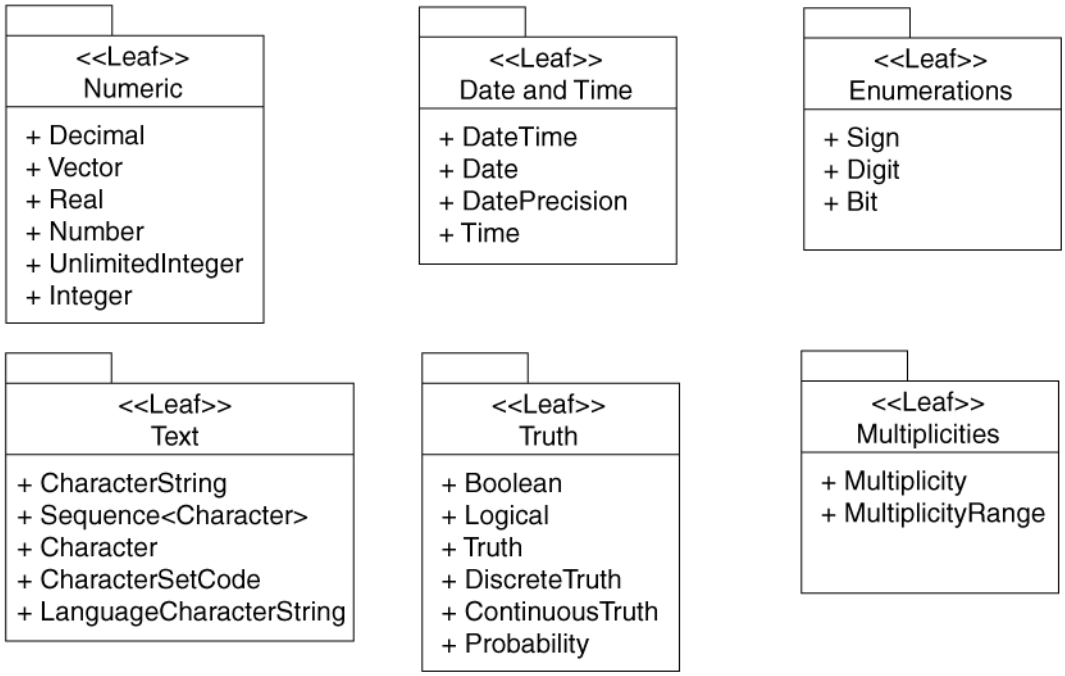


Figure 2 — Primitive types

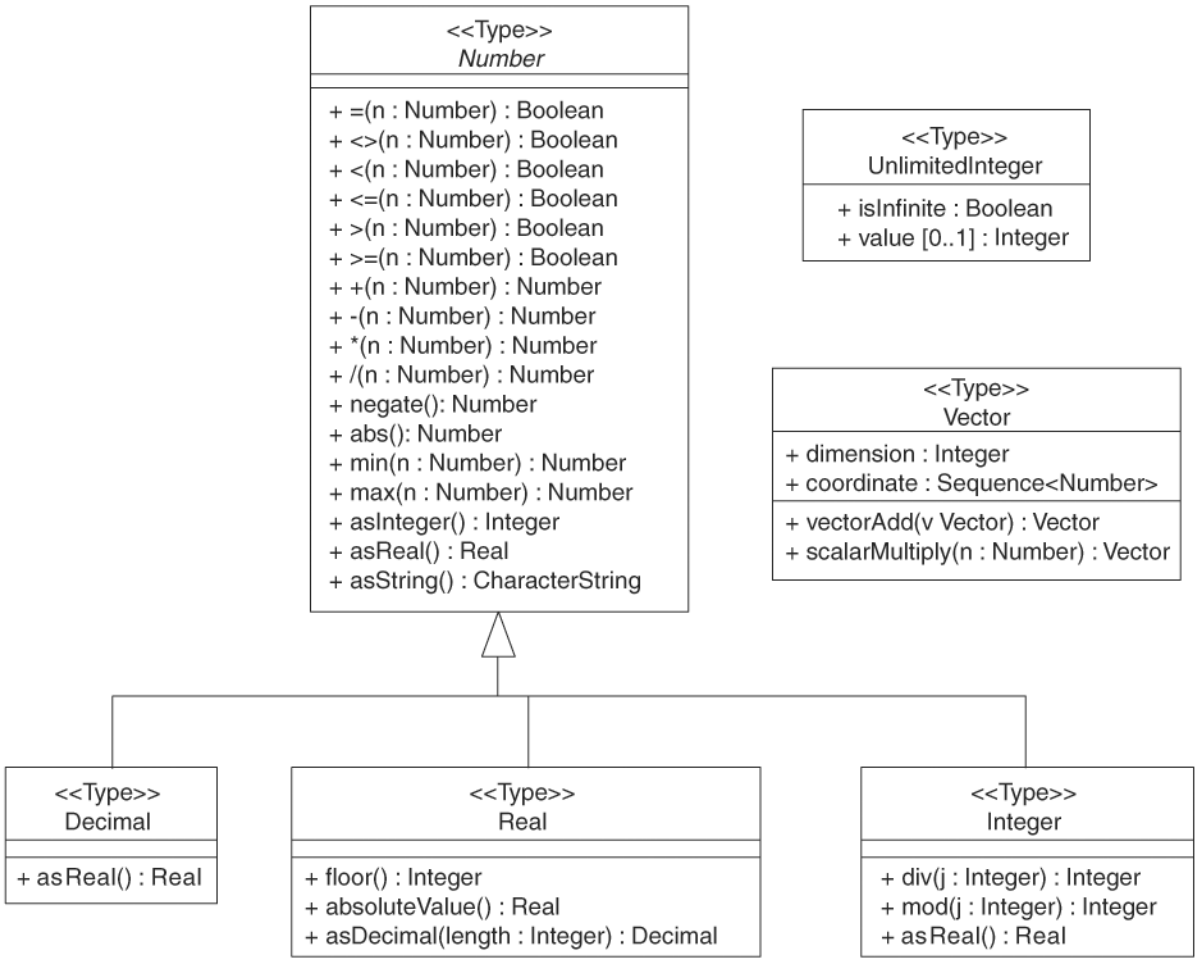


Figure 3 — Numeric types

6.5.2.2 Number

Number is the base type for all number data, giving the basic algebraic operations. Since all concrete types have finite representations, some parts of this algebra for most types exhibits some inaccuracy. For example, Integers cannot divide very well, and reals and decimals cannot avoid certain types of inaccuracies that depend on their representation semantics. Number is an *abstract* class.

6.5.2.3 Integer

Integer is a signed integer number; the length of an integer is encapsulation and usage dependent. It consists of an exact integer value, with no fractional part.

EXAMPLE 29, – 65547.

6.5.2.4 Decimal

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

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

EXAMPLE 12,75.

6.5.2.5 Real

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

EXAMPLE 23.501, – 1234E-4, – 23.0.

6.5.2.6 Vector

Vector is an ordered set of numbers called coordinates that represent a position in a coordinate system. The coordinates may be in a space of any number of dimensions, as for instance in an "n^{th} degree" polynomial spline.

EXAMPLE (123, 514, 150).

6.5.2.7 CharacterString

A CharacterString, as illustrated in Figure 4, is an arbitrary-length sequence of characters, including accents and special characters from the repertoire of one of the adopted character sets:

— ISO/IEC 10646: Universal Multi-Octet Coded Character Set (UCS), and

— ISO 8859.

EXAMPLE "Ærlige Kåre så snø for første gang."

The maximum length of a CharacterString is dependent on encapsulation and usage. A language tag may be provided for identification of the language of string values.

For an implementation mapping of a CharacterString, the handling of the following four aspects needs to be decided:

1) Representation of value,

2) Representation of character set,

3) Representation of encoding, and

4) Representation of language.

This can be handled for instance by choosing ISO/IEC 10646.

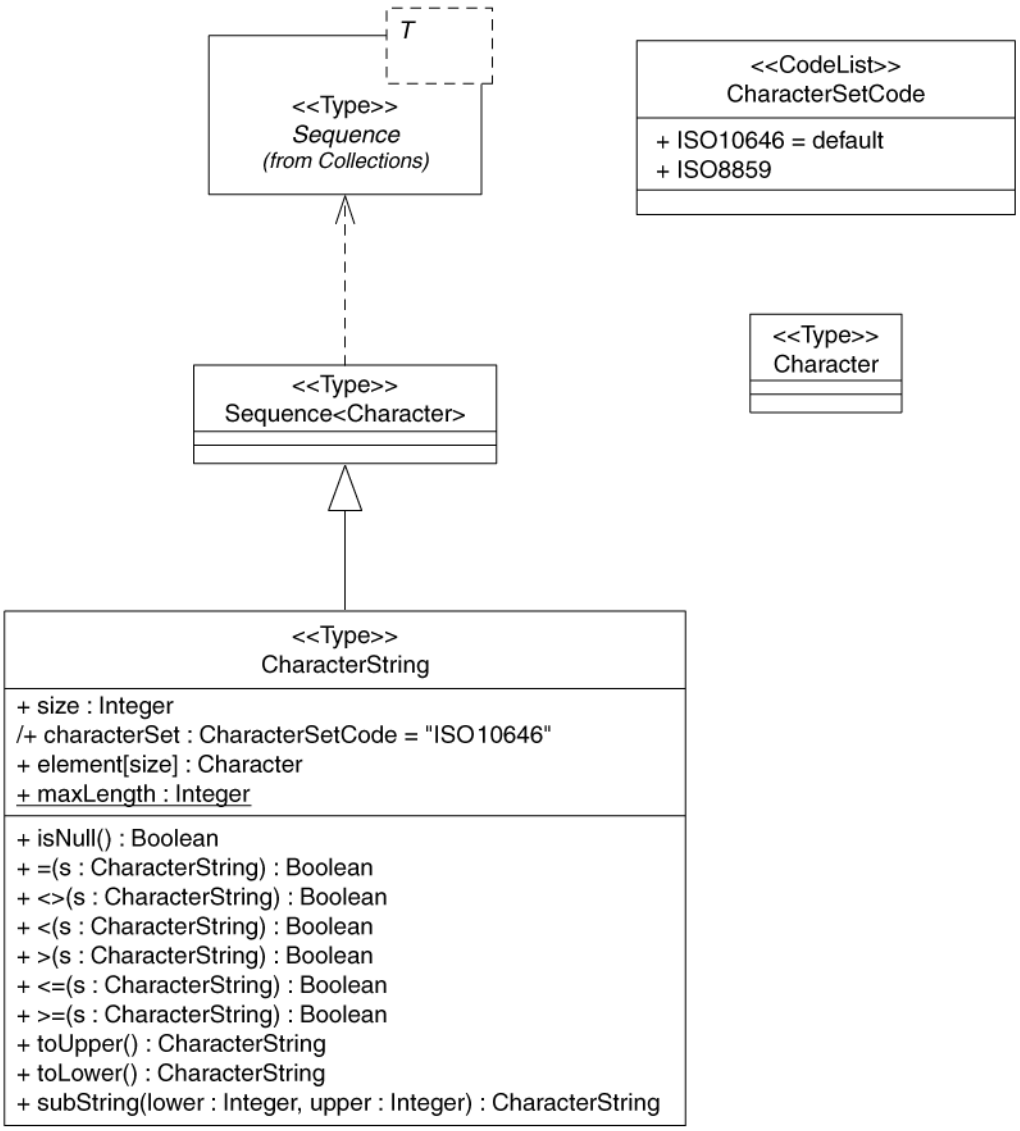


Figure 4 — Character string

6.5.2.8 Date

A date gives values for year, month and day. Figure 5 shows the Date type. Character encoding of a date is a string that shall follow the format for date specified in ISO 8601. Encoding is further discussed in ISO 19118 and ISO 19136. Principles for date and time are further discussed in ISO 19108.

EXAMPLE 1998-09-18.

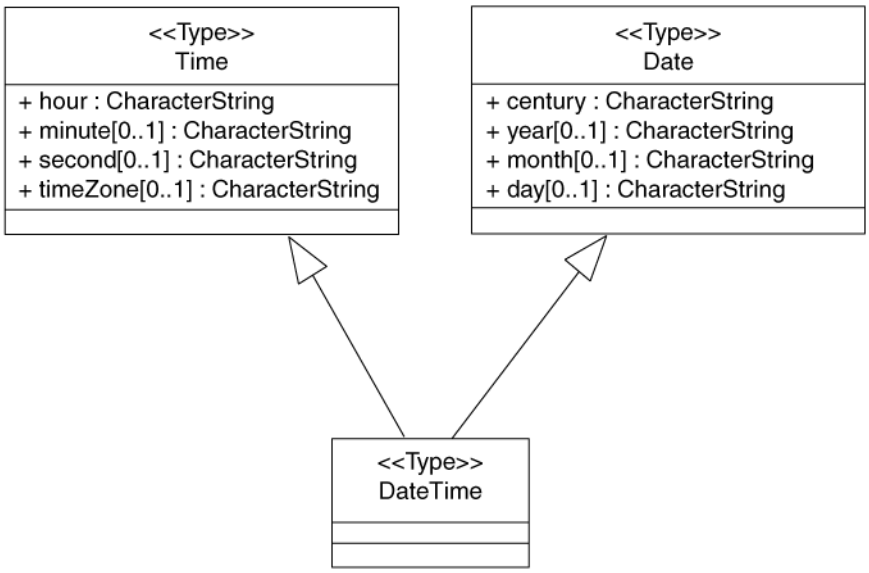


Figure 5 — Date and Time types

6.5.2.9 Time

A time is given by an hour, minute and second. Figure 5 shows the Time type. Character encoding of a time is a string that follows the ISO 8601 format. Time zone according to UTC is optional. Encoding is further discussed in ISO 19118 and ISO 19136. Principles for date and time are further discussed in ISO 19108.

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

6.5.2.10 DateTime

A DateTime is a combination of a date and a time type. Figure 5 shows the DateTime type. Character encoding of a DateTime shall follow ISO 8601. Encoding is further discussed in ISO 19118 and ISO 19136. Principles for date and time are further discussed in ISO 19108.

6.5.2.11 Boolean

Figure 6 shows values specifying TRUE or FALSE.

EXAMPLE TRUE or FALSE.

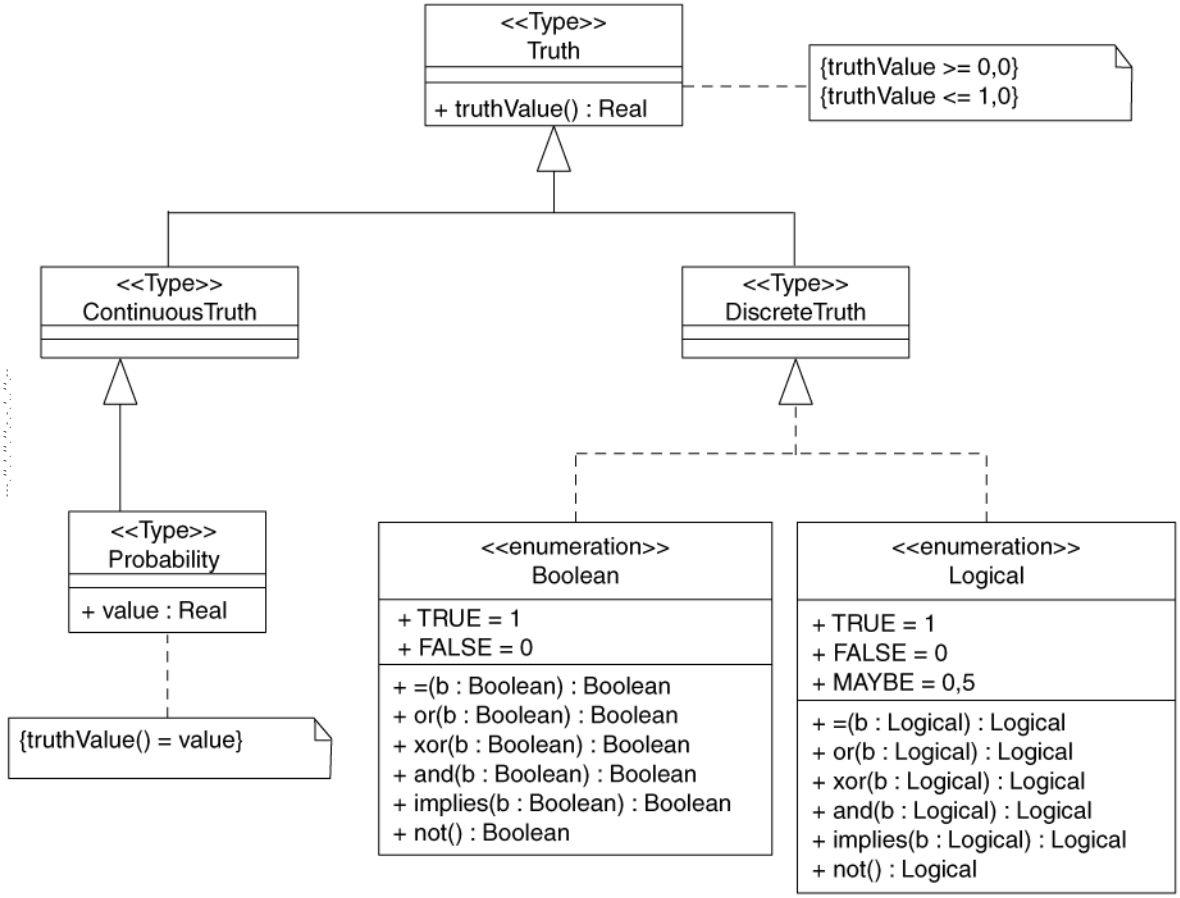


Figure 6 — Truth data types

6.5.2.12 Logical

Logical is a value specifying TRUE, FALSE or MAYBE (UNKNOWN).

6.5.2.13 Probability

Probability is represented as a number greater than or equal to 0,0 and less than or equal to 1,0.

6.5.2.14 Multiplicity

Multiplicity defines the lower and upper bounds of the number of possible instances, as shown in Figure 7.

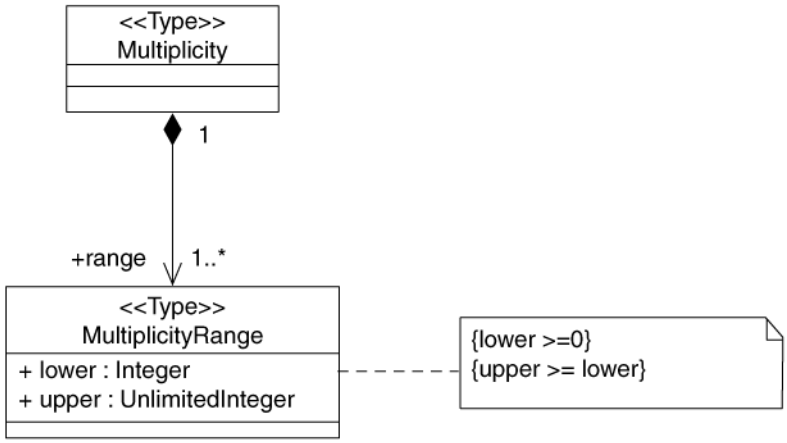


Figure 7 — Multiplicity

6.5.3 Collection and dictionary types

6.5.3.1 General

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

The major difference between collection types in this specification and in OCL is the inclusion of the TransfiniteSet (see Figure 8).

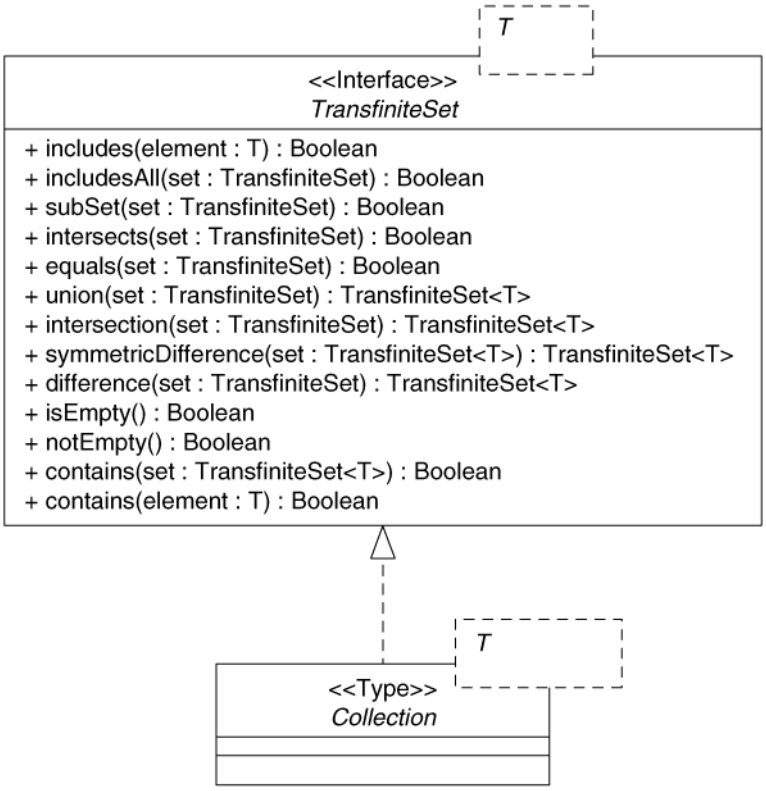


Figure 8 — Collection type

Protocols that do not depend upon the explicit instantiation of the membership of a set have been moved to this superclass level to define the behaviour of sets of possibly infinite size. Instantiations of this set would have to be done implicitly, via a Boolean test, instead of by explicit enumeration of members. Sometimes it is useful to specify and analyze a set which has not been explicitly instantiated.

6.5.3.2 Set

A Set is a finite collection of objects, where each object appears in the collection only once. A set shall not contain any duplicated instances. The order of the elements of the set is not specified. The model for Set is shown in Figure 9.

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

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

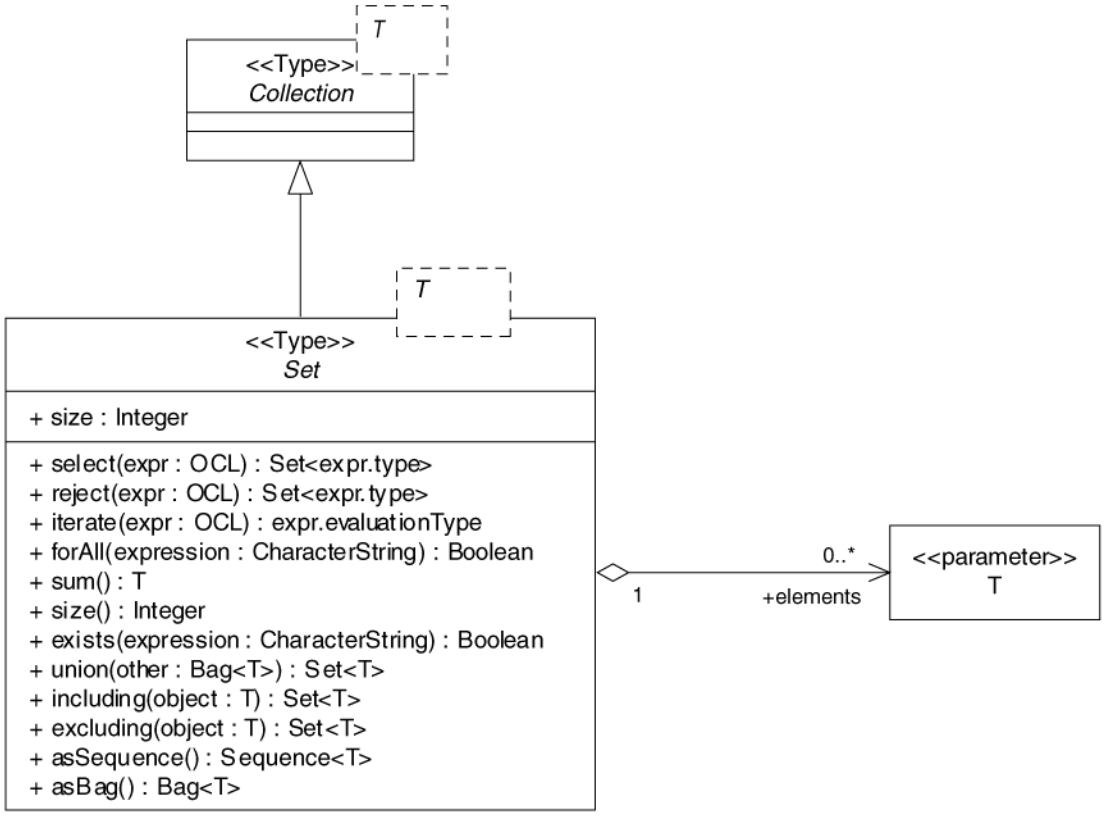


Figure 9 — Collection types — Set type

6.5.3.3 Bag

A bag may contain duplicate instances. As with a Set, there is no specified ordering among the elements of a bag. Bags are most often implemented through the use of proxies or reference pointers. The model for Bag is shown in Figure 10.

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

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

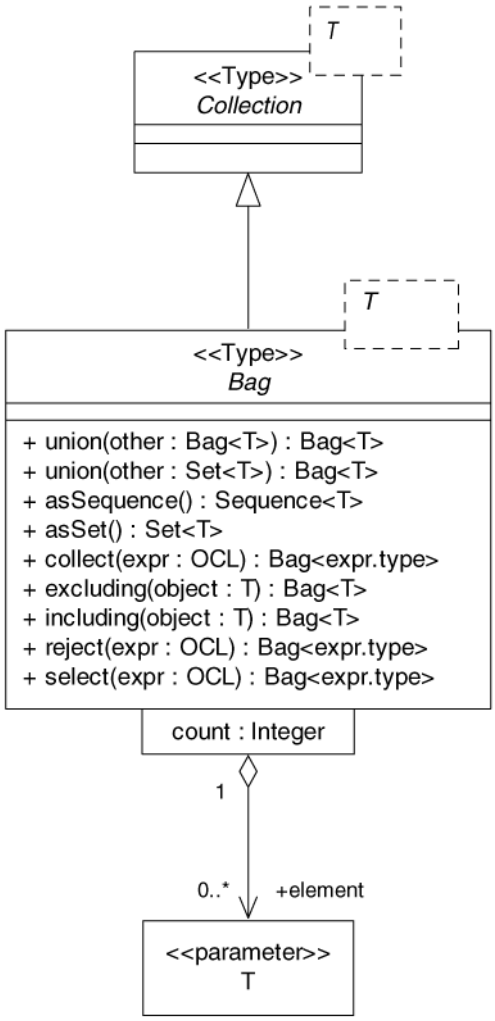


Figure 10 — Collection types — Bag type

6.5.3.4 Sequence

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

A sequence is a collection that specifies a sequential ordering between its elements. A synonym to sequence is List. A CircularSequence does not have a defined last element. The model for Sequence is shown in Figure 11.

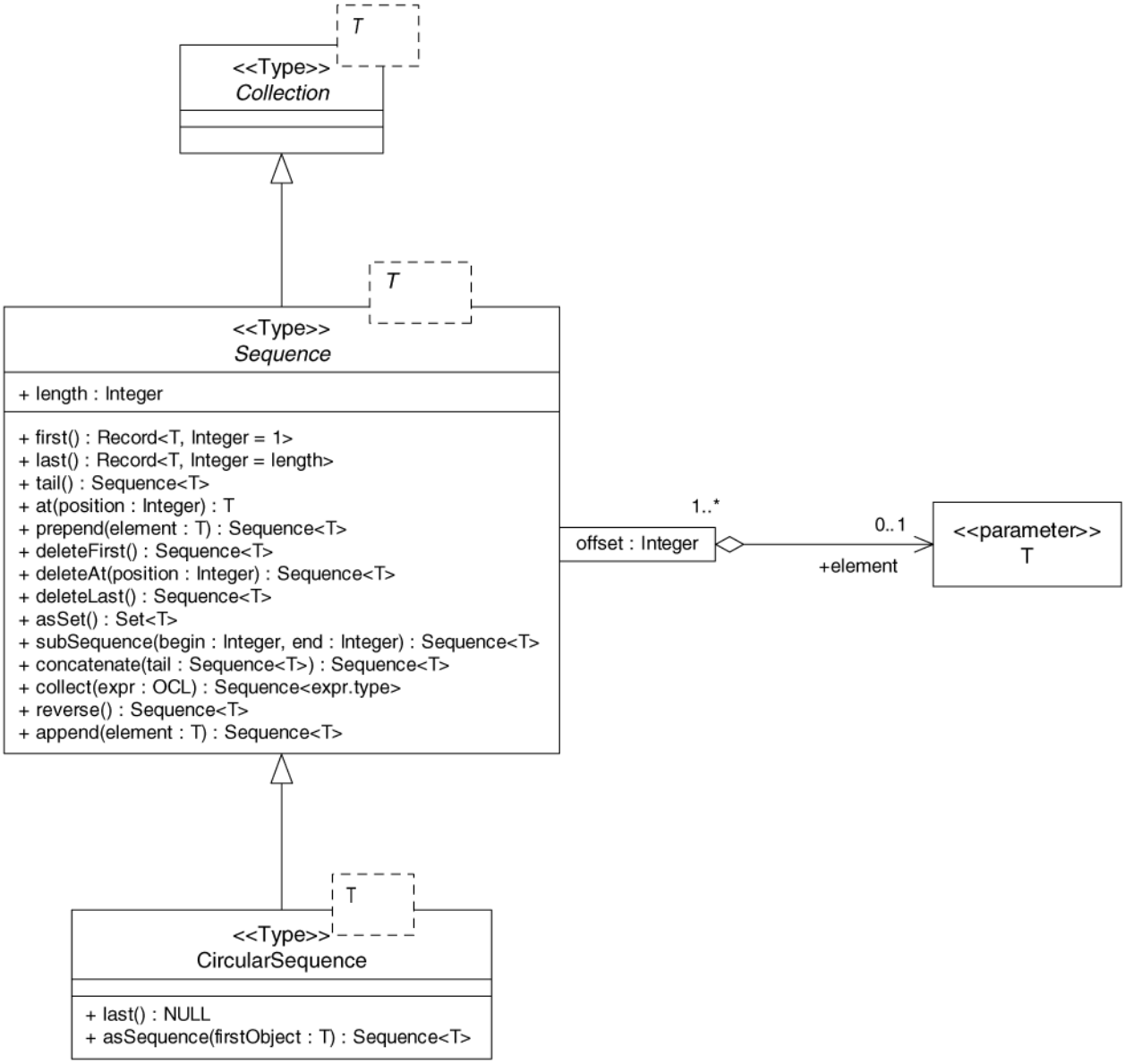


Figure 11 — Collection type — Sequence type

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

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

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

6.5.3.5 Dictionary

A dictionary is similar to an array, except that the lookup index for an array is expressed in integer numbers. Although any index type (KeyType) or return type (ValueType) is acceptable, the typical use within this Technical Specification will be to use character strings as the KeyType and numbers as the ValueType (see Figure 12).

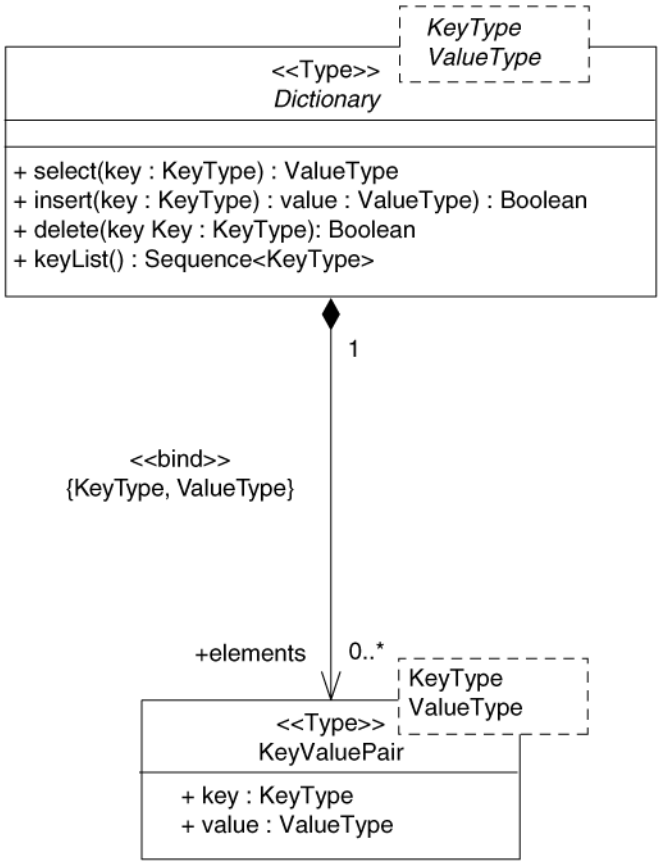


Figure 12 — Dictionary classes

6.5.4 Enumerated types

6.5.4.1 General

Enumeration types

enum { value1, value2, value3 }

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

EXAMPLE attr1 : BuildingType, where BuildingType is defined as Enum BuildingType { Public, Private, Tourist }.

6.5.4.2 Enumeration

Enumerations are modelled as classes that are stereotyped as <<Enumeration>>. An enumeration class can contain only simple attributes which represent the enumeration values. Other information within an enumeration class is void. An enumeration is a user-definable data type whose instances form a list of named literal values. Usually, both the enumeration name and its literal values are declared. The extension of an enumeration type will imply a schema modification for most implementation mappings, and one shall use enumeration only when it is clear that there will be no extensions. If extensions are expected, the type <<CodeList>> should be used. Example of Enumeration is shown in Figure 13.

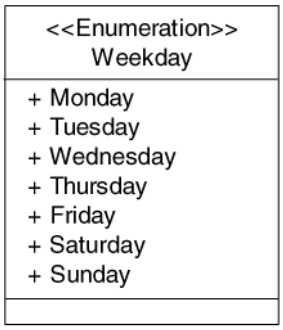
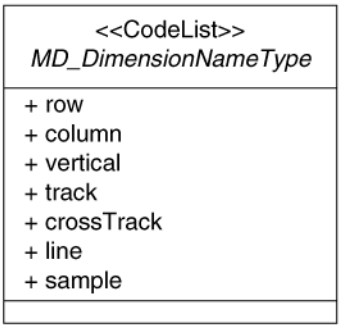


Figure 13 — Example of Enumeration

6.5.4.3 CodeList

CodeList can be used to describe an open enumeration. This means that it needs to be represented in such a way that it can be extended during system runtime. <<CodeList>> is a flexible enumeration that uses string values through a binding of the Dictionary type key and return values as string types; e.g. Dictionary (CharacterString, CharacterString). Code lists are useful for expressing a long list of potential values. If all the elements of the list are known, an enumeration shall be used; if only the likely values of the elements are known, a code list shall be used. Enumerated code lists may be encoded according to an International Standard, such as ISO 3166-1. Code lists are more likely to have their values exposed to the user, and are therefore often mnemonic. Different implementations are likely to use different encoding schemes (with translation tables back to other encoding schemes available). The chosen representation by an implementation of a codelist is to represent the value/code(key) pairs as attributes with a stereotyped <<CodeList>> with an attribute name for each value and the code(key) represented as an initial value. In the case when only attribute names are shown, the codes and their descriptions are equivalent. Examples of CodeLists are shown in Figure 14.



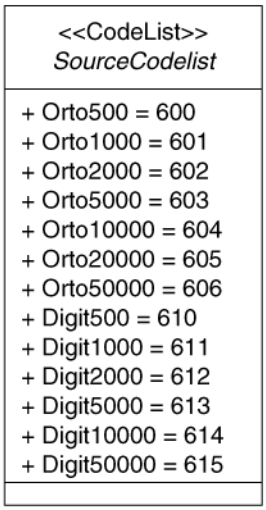


Figure 14 — Examples of CodeLists

6.5.5 Representation types: Record and RecordType

A Record is a list of logically related elements as (name, value) pairs in a Dictionary. A Record may be used as an implementation representation for features (see Figure 15).

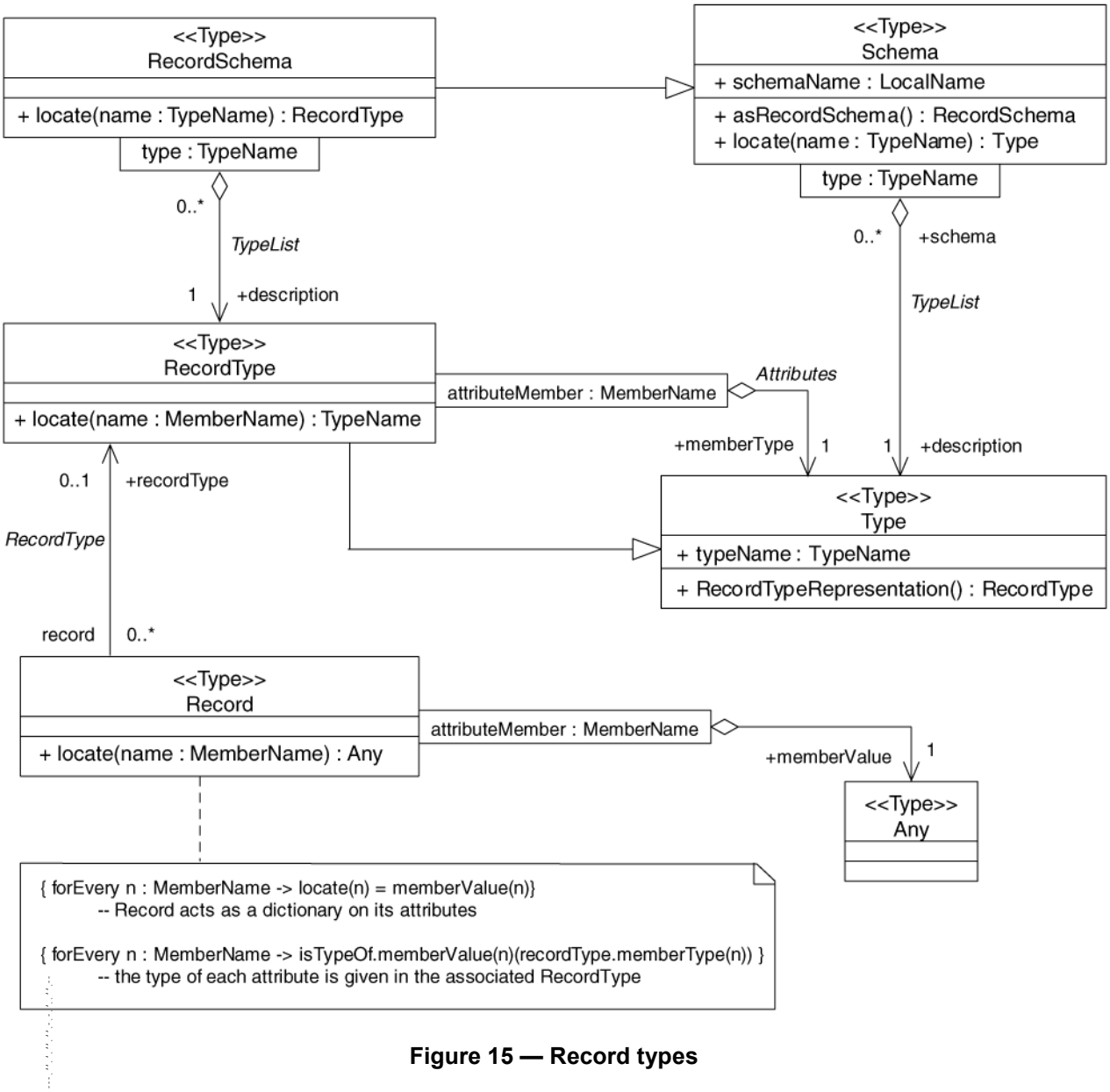


Figure 15 — Record types

6.5.6 Names

6.5.6.1 General

GenericName and subclasses of this are used to create a generic scoped and local name structure for type and attribute names in the context of namespaces.

6.5.6.2 NameSpace

NameSpace defines a domain in which “names” given by character strings (possibly under local constraints enforced by the namespace) can be mapped to objects via a getObject operation. Examples include objects which form a namespace for their attributes, operations and associations, or schemas that form namespaces for their included data types or classes. The model for NameSpace and Name types is shown in Figure 16.

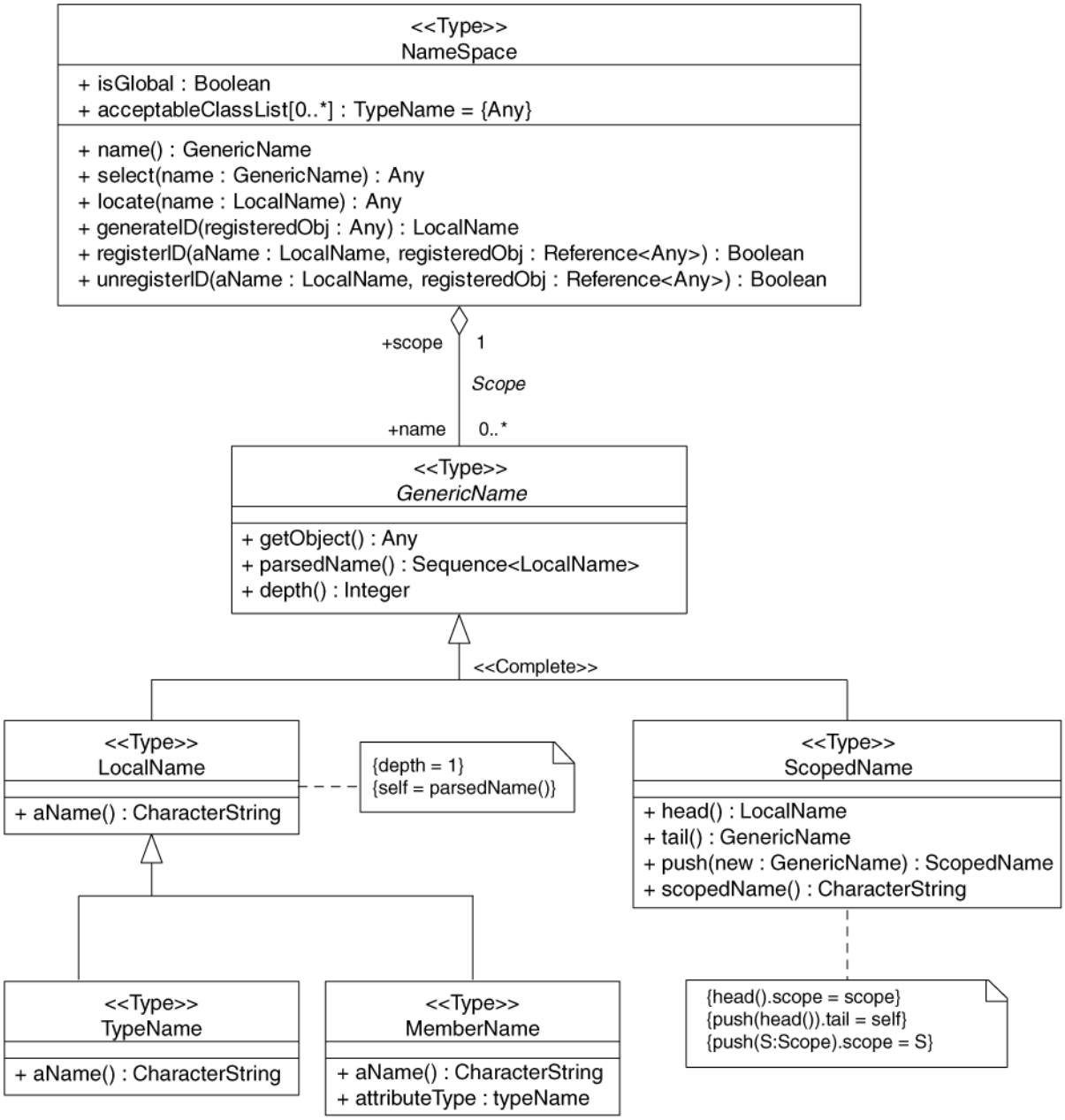


Figure 16 — Name types

6.5.6.3 GenericName

GenericName is the abstract class for all names in a NameSpace. Each instance of a GenericName is either a LocalName or a ScopedName.

6.5.6.4 ScopedName

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

6.5.6.5 LocalName

A LocalName references a local object directly accessible from the NameSpace.

6.5.6.6 TypeName

A TypeName is a LocalName that references either a recordType or object type in some schema. The stored value “aName” is the returned value for the “aName()” operation. This is the type name.

6.5.6.7 MemberName

A MemberName is a LocalName that references either an attribute slot in a record or recordType, an attribute, operation, or association role in an object instance or a type description in some schema. The stored value “aName” is the returned value for the “aName()” operation.

6.5.7 Derived types

6.5.7.1 General

Figure 17 gives an overview of the units of measure data types that are described in the following clauses.

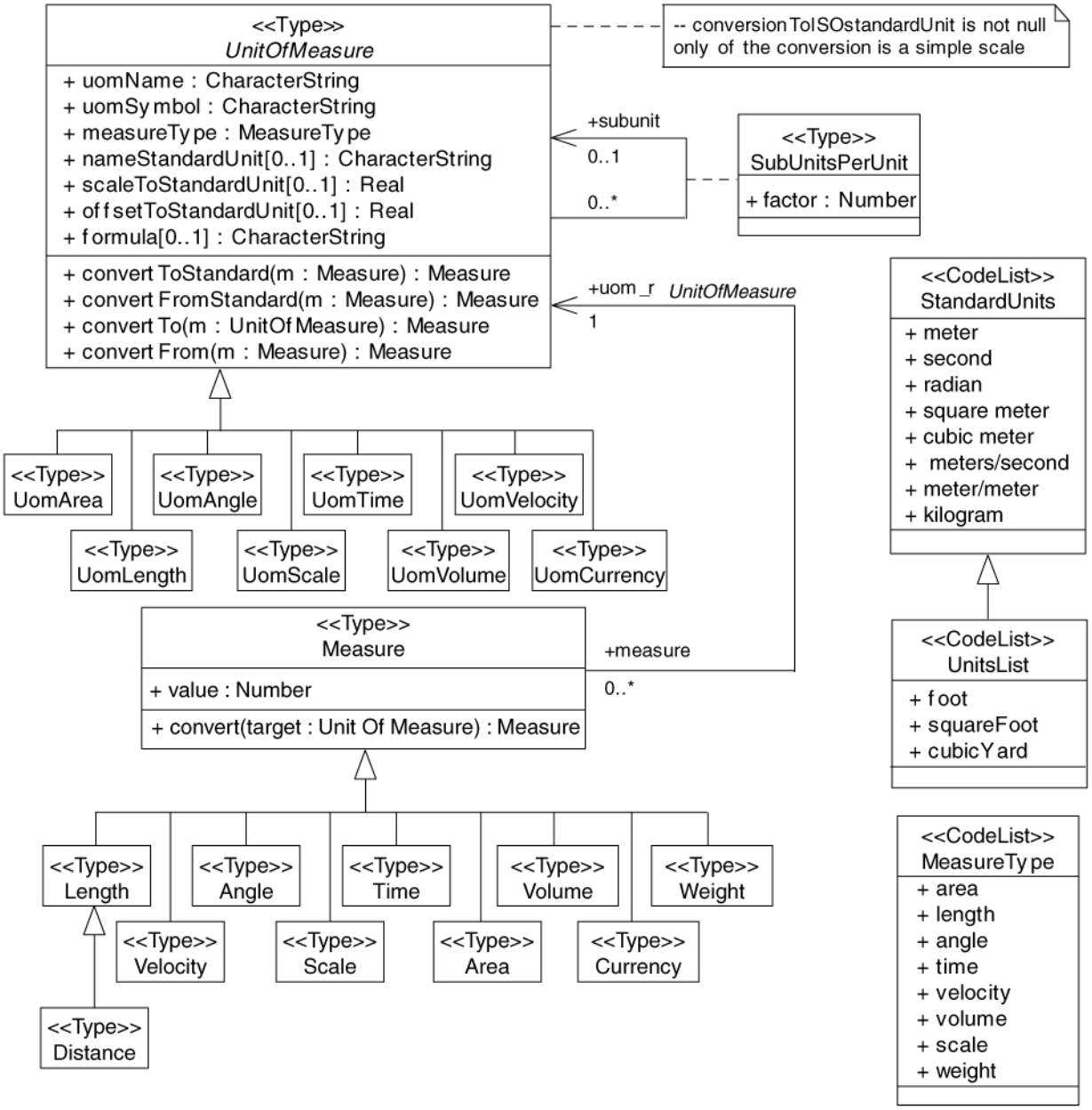


Figure 17 — Units of Measure types

6.5.7.2 Measure

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

6.5.7.3 UnitOfMeasure

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

The role uom (Figure 17) is constrained by the value of the attribute uom in each Measure subclass.

6.5.7.4 Area

Area is the measure of the physical extent of any two-dimensional geometric object.

6.5.7.5 UomArea

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

6.5.7.6 Length

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

6.5.7.7 Distance

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

6.5.7.8 UomLength

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

6.5.7.9 Angle

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

6.5.7.10 UomAngle

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

6.5.7.11 Scale

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

6.5.7.12 UomScale

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

6.5.7.13 Time

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

6.5.7.14 UomTime

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

6.5.7.15 Volume

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

6.5.7.16 UomVolume

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

6.5.7.17 Velocity

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

6.5.7.18 UomVelocity

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

6.5.7.19 AngularVelocity

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

6.5.7.20 UomAngularVelocity

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

6.5.8 NULL and EMPTY values

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

6.6 Operations

Operations are used according to standard UML, as described in D.6.

6.7 Relationships and associations

Relationships and associations are used according to standard UML, with the additional requirements specified in this subclause.

All associations shall have cardinalities defined for both association ends. At least one role name shall be defined. If only one role name is defined, the other will by default be inv_rolename.

If an association is navigable in a particular direction, the model shall supply a “role name” that is appropriate for the role of the target object in relation to the source object. Thus in a two-way association, two role names

will be supplied. The default role name is “the<target class name>” in which the target class is referenced from the source class (this is the default name in many UML tools). Association names are principally for documentation purposes.

Relationships involving more than two classes shall be avoided in order to reduce complexity. Cardinality for associations is specified as UML multiplicity specifications.

An association with role names can be viewed as similar to defining attributes for the two classes involved, with the additional constraint that updates and deletions are consistently handled for both sides. For one-way associations, it thus becomes equivalent to an attribute definition. A role thus serves as a pseudo-attribute. The recommendation for the ISO geographic information standards is to use the association notation for classes and interfaces, and to use attribute notation for data types.

6.8 Stereotypes and tagged values

6.8.1 General

Stereotypes and tagged values are used according to standard UML, as described in D.8, with an additional set of stereotypes.

6.8.2 Stereotypes

In this Technical Specification, the following stereotypes are defined:

a) <<CodeList>> is a flexible enumeration that uses string values through a binding of the Dictionary type key and return values as string types; e.g. Dictionary (String, String). If the elements of a list are completely known, an enumeration shall be used; if only the likely values of the elements are known, a code list shall be used.

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

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

Stereotypes are essential in the creation of code generators for UML models. Generally speaking, stereotypes act as flags to language compilers to determine how to create implementation models from the abstract. Enumeration and CodeList are prime examples of this. As opposed to the creation of a class, these classes generate small value domains, one value per “attribute” in the model. In an Enumeration, these values are usually represented as a 1-up number code for the attributes listed, and the list is assumed to be completely defined in the Abstract Specification. In a CodeList, the code values are not specified, and the list is extendible at any time. For examples, CodeLists can be used to implement the ISO country codes and language code used commonly in metadata specifications.

6.9 Optional, conditional and mandatory attributes and associations

6.9.1 Mandatory

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

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

NOTE The default multiplicity for attributes is 1.

An association that has a multiplicity with a lower bound of 1 at the target end is mandatory for the class at the source end.

6.9.2 Optional

A multiplicity of 0..1 or 0..* means that an attribute or association may be present or may be omitted, which means that it is optional.

The most appropriate way of specifying optionality is to use a tagged value. However, since different tools have different ways of handling and presenting tagged values, multiplicity for the sake of readability is used. Optionality means that a null value must be represented in the instance model, e.g. a place holder element or a null value. An optional or conditional attribute shall never have a default value defined.

6.9.3 Conditional

An attribute or association may be specified as conditional, meaning that whether it is mandatory or optional depends on other model elements. The dependencies may be by existence-dependence of other (optional) elements or by the values of other elements. A conditional attribute or association is shown as optional with an OCL constraint attached.

6.10 Naming and name spaces

Naming conventions are used for a variety of reasons, mainly readability, consistency and as a protection against case-sensitive binding.

The names of UML elements shall:

1) Use precise and understandable technical names for classes, attributes, operations, and parameters.

   — Correct: index

   — Incorrect: n

   — Short parameter names shall be used when the parameter type carries meaning; use Equals(other : GM_Object) as opposed to Equals(otherGeometryObject : GM_Object)

2) Combine multiple words as needed to form precise and understandable names without using any intervening characters (such as “_”, “-”, or space).

EXAMPLE ComputePartialDerivatives (not: Compute Partial Derivatives or compute_Partial_Derivatives).

3) For attributes and operation names, association roles and parameters, capitalize only the first letter of each word after the first word that is combined in a name. Capitalize the first letter of the first word for each name of a class, package, type-specification and association names.

EXAMPLE 1 ComputePartialDerivatives (not computepartialderivatives or COMPUTEPARTIALDERIVATIVES).

EXAMPLE 2 (Class): CoordinateTransformation (not coordinateTransformation).

4) Use documentation fields to further explain the meanings (semantics) of names.

5) Keep names as short as practical. Use standard abbreviations if understandable, skip prepositions, and drop verbs when they do not significantly add to the meaning of the name.

   — numSegment instead of numberOfSegments,

   — equals instead of IsEqual,

   — value() instead of getValue(),

   — initObject instead of initializeObject, and

   — length() instead of computeLength().

All classes shall have unique names. All classes shall be defined within some package. Class names shall start with an upper case letter. A class shall not have a name that is based on its external usage, since this may limit reuse. A class name shall not contain spaces. Separate words in a class name shall be concatenated. Each subword in a class name shall begin with a capital letter, such as “XnnnYmmm”.

To ensure global uniqueness of class names, every class name shall be defined with an alphabetic prefix separated from the body of the name by an underscore “_”. These prefixes are essentially name space specifiers and are used within the ISO 19100 series of standards to prevent confusion without using full package name prefixes in each place where a class name is used. Most prefixes are two characters but some are three characters and can refer to common XML namespace prefixes. Prefixes shall honour package boundaries, so that all classes within a high-level package share the same prefix, and no two high-level packages share a prefix. ISO 19107 uses the prefix GM in its geometry model and the prefix TP in its topology model. Other prefixes have been defined for other areas. The use of classname prefixes is not required in implementations where name spaces are properly handled.

The name of an association shall be unique within the context of a class and its supertypes.

In most cases using OCL, the name space is the class in which the operation is defined, but it can also include the package name in which a class is defined. Where name spaces are class identifiers, they can take only one of two forms.

type :== class-name | package-name::class-name

The “::” is a resolution operator indicating the name space of that which follows.

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

The UML naming scope with package::package::className allows for the same className to be defined in different packages. However, many UML tools do not currently allow for this. Therefore, a more restrictive naming convention is adopted:

— Although the model is case sensitive, all class names shall be unique in a case insensitive manner.

— Class name shall be unique across the entire model (so as not to create a problem with many UML tools).

— Package names shall be unique across the entire model (for the same reason).

— Every effort should be applied to eliminate multiple classes instantiating the same concept.

6.11 Packages

Packages shall be structured according to a three-level structure as follows:

1) Top-level structures are used for structuring the various main parts of a model.

2) Second-level (internal) packages contain only other subpackages.

3) Third-level (leaf) packages contain only class-diagrams.

This three-level structure makes the meaning of package dependencies clearer. A package shall also be used to represent an application schema. An application schema that contains any package defined in another schema shall also contain all of its dependencies. Application schemas shall contain at least one package diagram that specifies dependencies between the packages defined by the ISO geographic information standards and the packages defined by the application schema. An example package structure of dependencies between the ISO geographic information standards is shown in Figure 18.

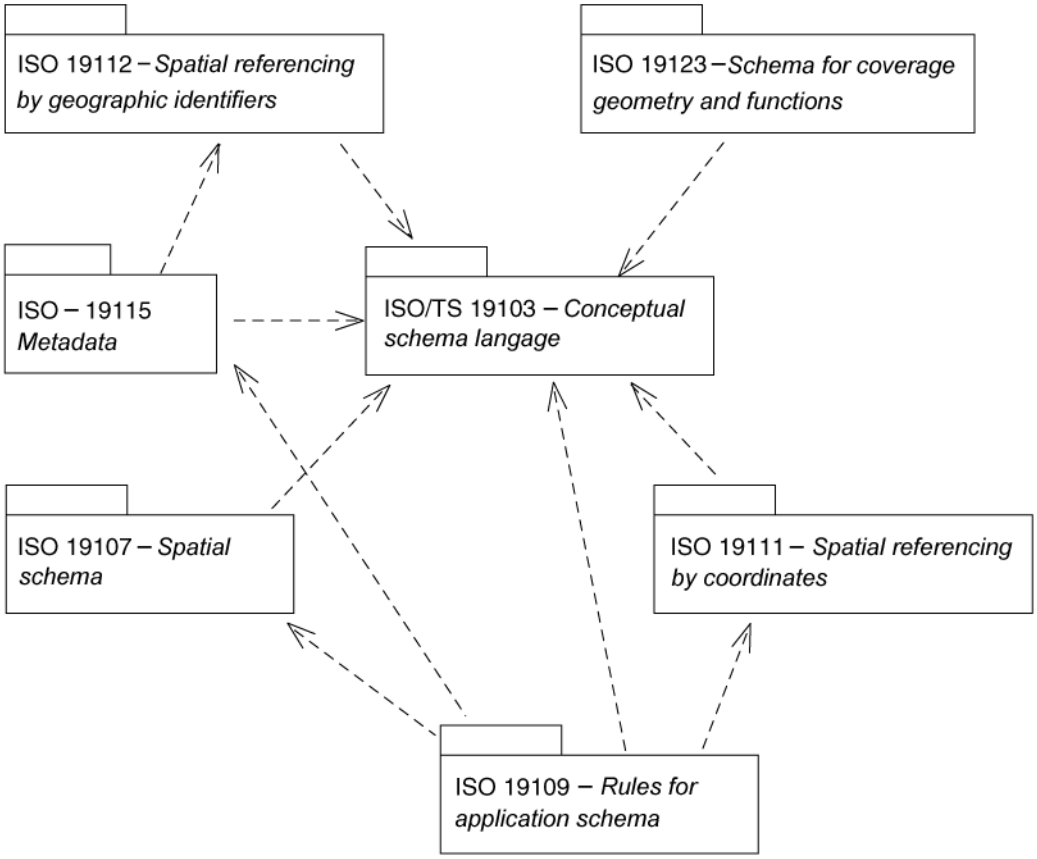


Figure 18 — Example package structure

Leaf packages are dependent on one another if their contained objects depend on one another. Internal packages are dependent only if, at some level, one of their leaves is dependent on leaves in the other. Actually, only leaf-to-leaf dependencies need to be shown. The extension of dependencies beyond the leaf is both redundant and difficult to maintain.

UML packages shall be used to group classes and their associations. Some rules for doing this are:

— All classes shall be part of some defined UML package.

— If a package component class is used in another package diagram, that diagram shall reference the original package to which the class belongs.

6.12 Notes

Notes are specified according to standard UML (see D.12).

6.13 Constraints

Constraints are specified according to the standard UML OCL, Object Constraint Language [3] (see D.12).

6.14 Documentation of models

In addition to the presentation of the model in diagrams, it is necessary to document the semantics of the model. The meaning of attributes, associations, operations and constraints shall be explained. The following template is suggested for documenting models.

For the diagrams, readability concerns shall be taken into account. Font size shall be no smaller than 8 points after diagram scale is taken into account (a 12-pt font at 90% is 12*0.9 = 10.8 pt). Each package shall have a class diagram for the entire package (without any associations, operations or attributes shown).

Each class shall have a context diagram (all of its associations, operation signatures and attributes shown). Each class shall have a definition describing its intended meaning or semantics. Each operation, attribute and relation shall have a textual description in the text near its context diagram. Note that closely related classes can share a context diagram as long as the combined diagram is neither overly cluttered nor difficult to read.

Most diagrams in the standard parts will be “context diagrams” which centre on a single class and display its attributes, operations and important relationships.

Package dependency diagrams shall be included, including external package dependencies. Overview diagrams consisting of all major classes in a document may be included if found feasible.

For each class there shall be a textual description of the semantics at least as follows:

<A>

General description of class A

class name: A

Specializes: a list of superclasses/supertypes

Attributes:

name: description of attribute meaning

Associations:

role: description of role meaning

Operations:

Operation: name

Description: <text>

Precondition: <text + OCL>

Input parameters: <description>

Output parameters : <description>

Return value: <description>

Exceptions: <description>

Postcondition: <text + OCL>

Constraints: < any sequencing constraints in the use of this operation with other operations?>

Other Constraints:

Any other constraints to be described. <text + OCL>

Annex A
(normative)

Abstract test suite

To check conformance of UML models to this Technical Specification, it is necessary to check that the models have been made following the rules in Clause 6 of this Technical Specification.

a) Test Purpose: Verify that the usage of UML has been done according to the rules in ISO/TS 19103.

b) Test Method: Inspect the diagrams, use the checking/validation capabilities of the UML tools that are being used.

c) Reference: ISO/TS 19103.

d) Test Type: Basic.

Verify that the usage of the following elements has been done according to 6.2 – 6.14.

— General usage of UML,

— Classes,

— Attributes,

— Data types,

— Operations,

— Relationships and associations,

— Stereotypes and tagged values,

— Optional, conditional and mandatory attributes and associations,

— Naming and name spaces,

— Packages,

— Notes,

— Constraints, and

— Documentation of models.

Annex B
(informative)

On conceptual schema languages

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

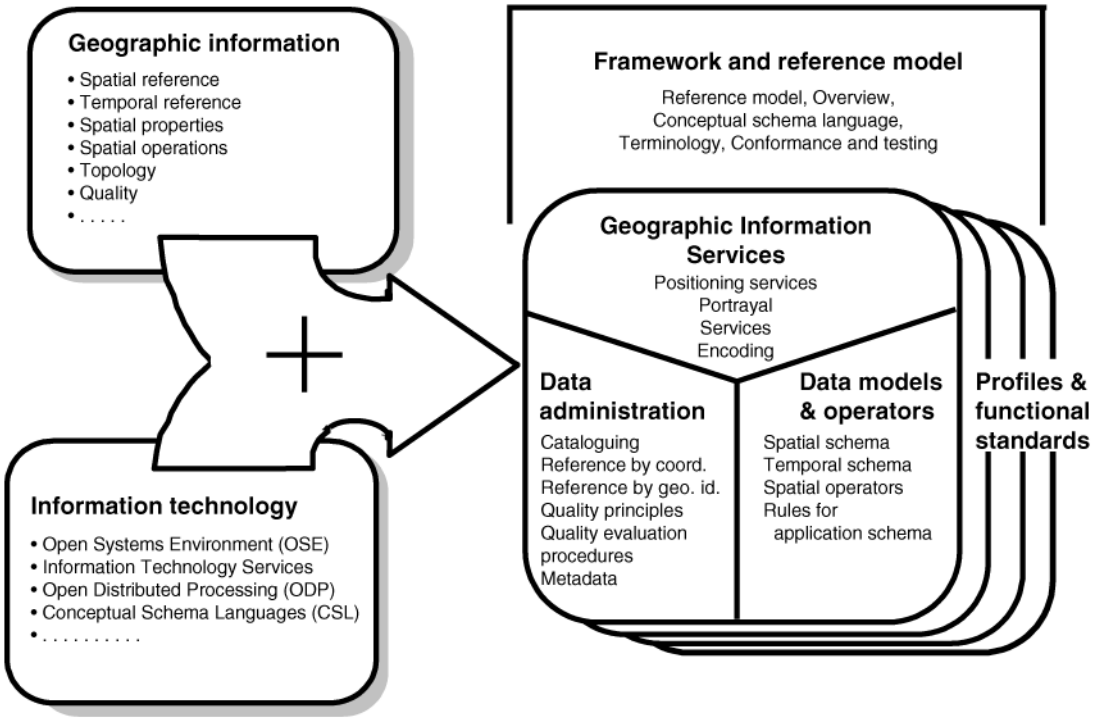


Figure B.1 — ISO geographic information standards work based on both GIS and IT concepts

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

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

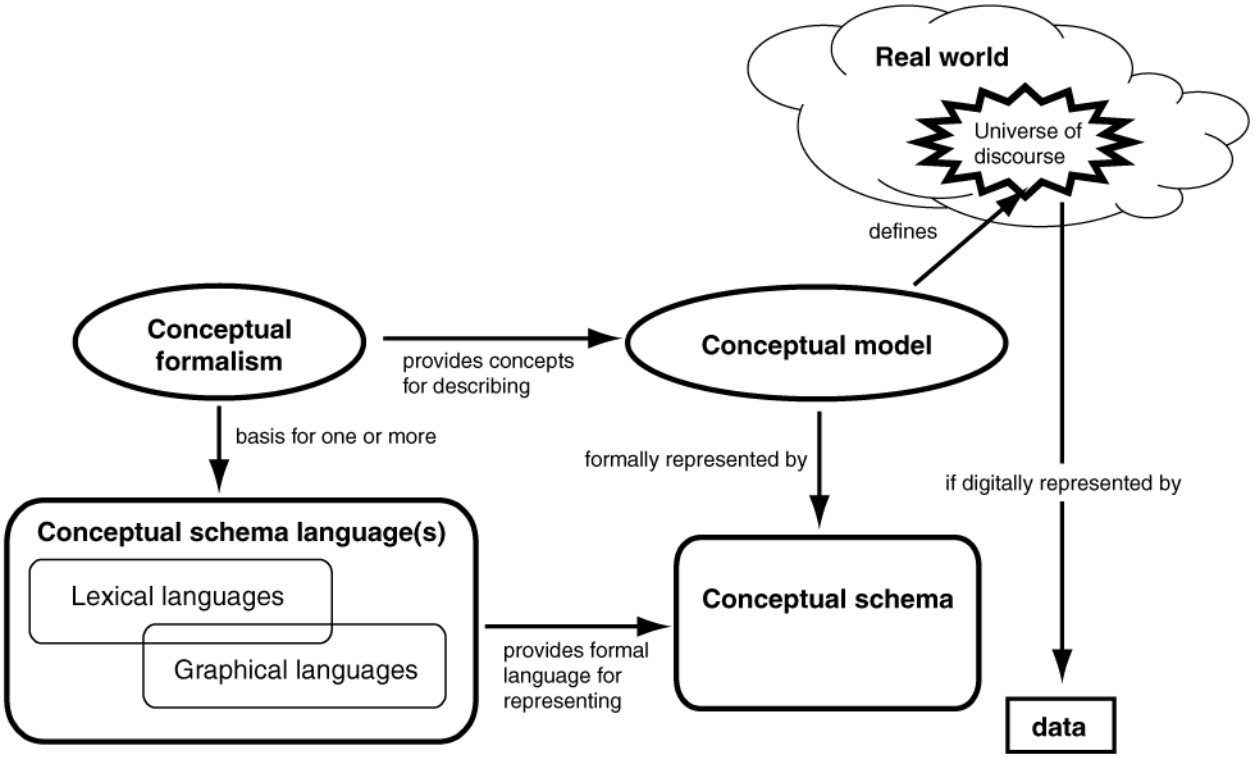


Figure B.2 — From reality to conceptual schema, based on a conceptual formalism and conceptual schema language

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

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

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

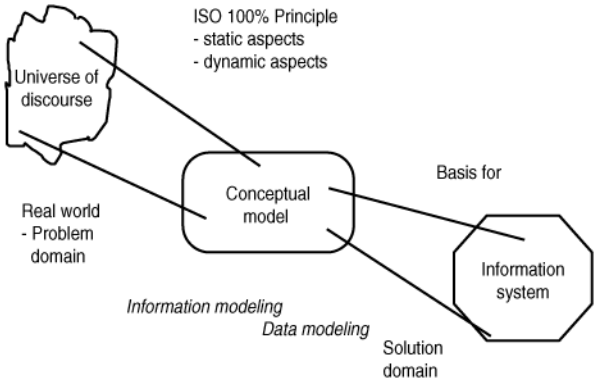


Figure B.3 — The ISO 100 % principle for conceptual modeling

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

Figure B.4 shows the metamodel layers of CSMF, the Conceptual Modeling Facility.

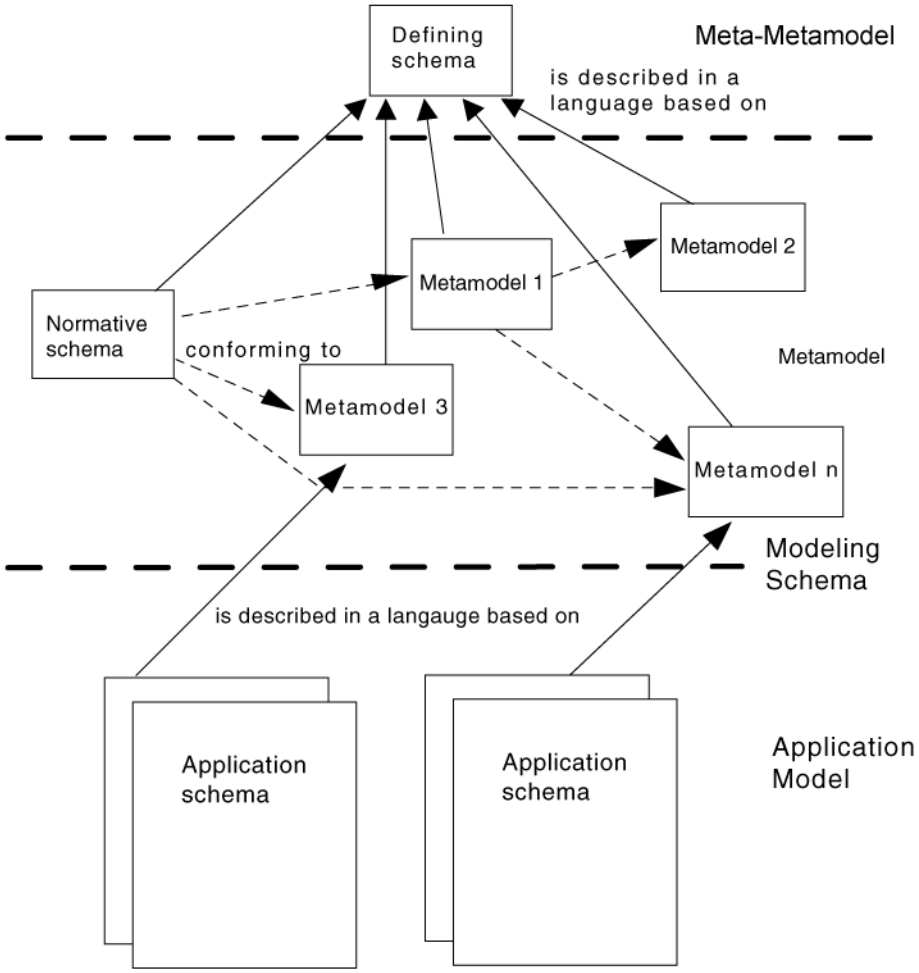


Figure B.4 — Metamodel layers of CSMF — Conceptual Schema Modeling Facility

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

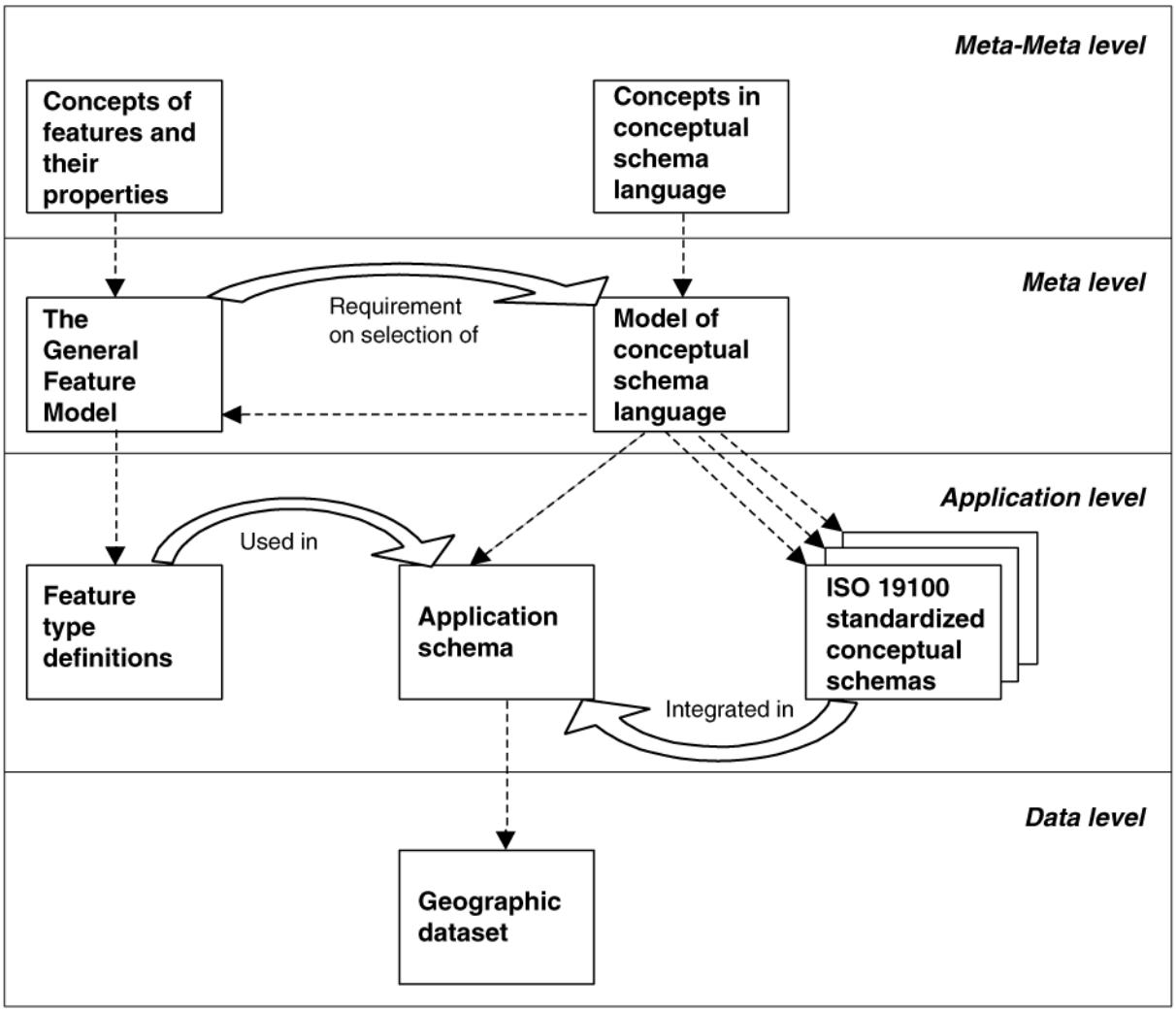


Figure B.5 — Relationship of the General Feature Model and the UML model

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

— Temporal,

— Spatial (Position/Topology),

— Quality,

— Geographic identifier, and

— Thematic attributes.

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

— Specialization,

— Aggregation,

— Spatial (Position/Topology), and

— Logical/Association.

The General Feature Model is further presented in ISO 19109. Figure B.6 shows the kernel of the General Feature Model.

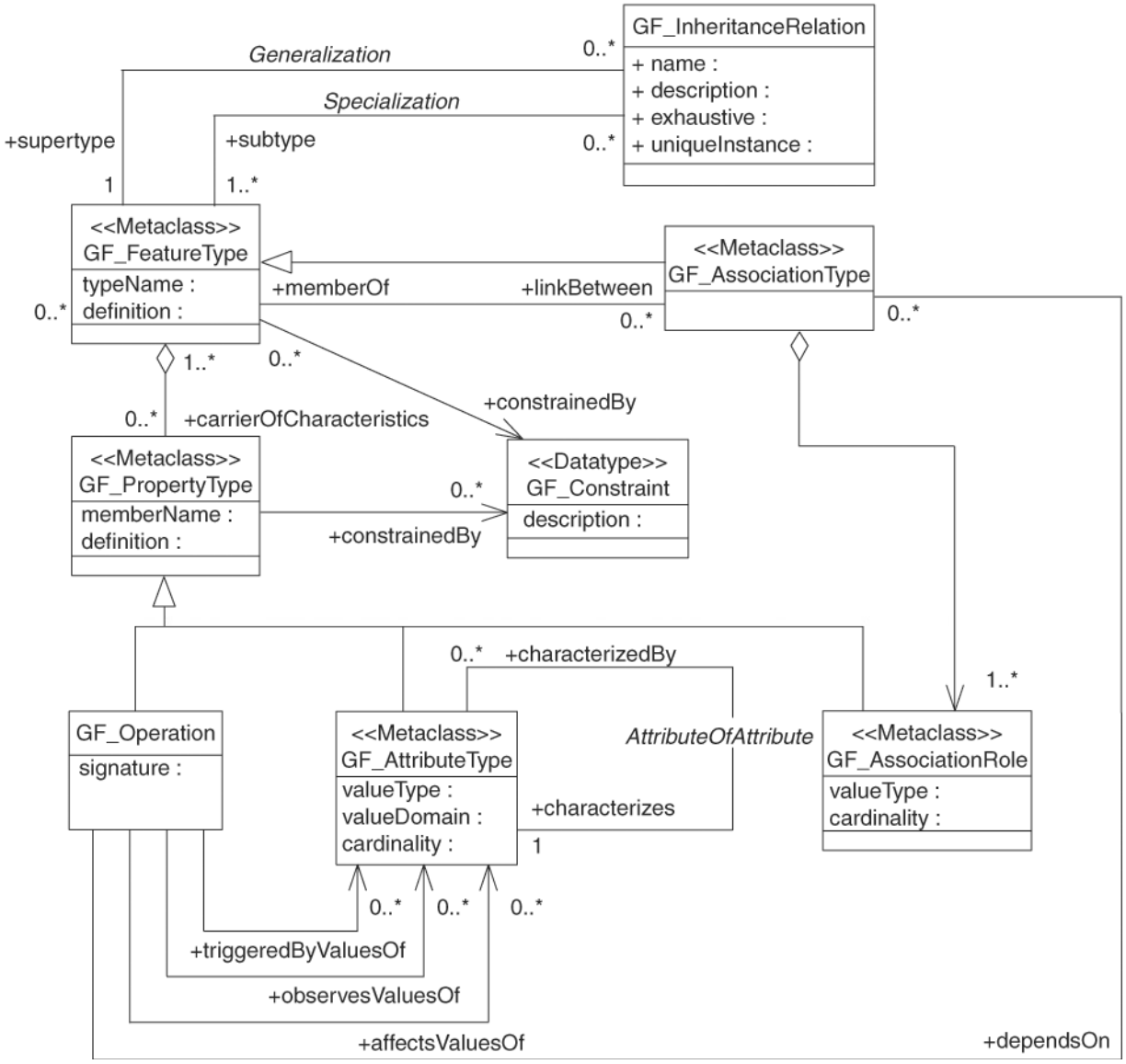


Figure B.6 — The kernel of the ISO 19109 General Feature Model

ISO 19109 contains a further description of the relationship between the General Feature Model and UML and the UML metamodel. ISO 19109 also includes a further description of terms such as feature, feature association, feature attribute and feature operation. Figure B.7 shows the essential parts of the UML core metamodel.

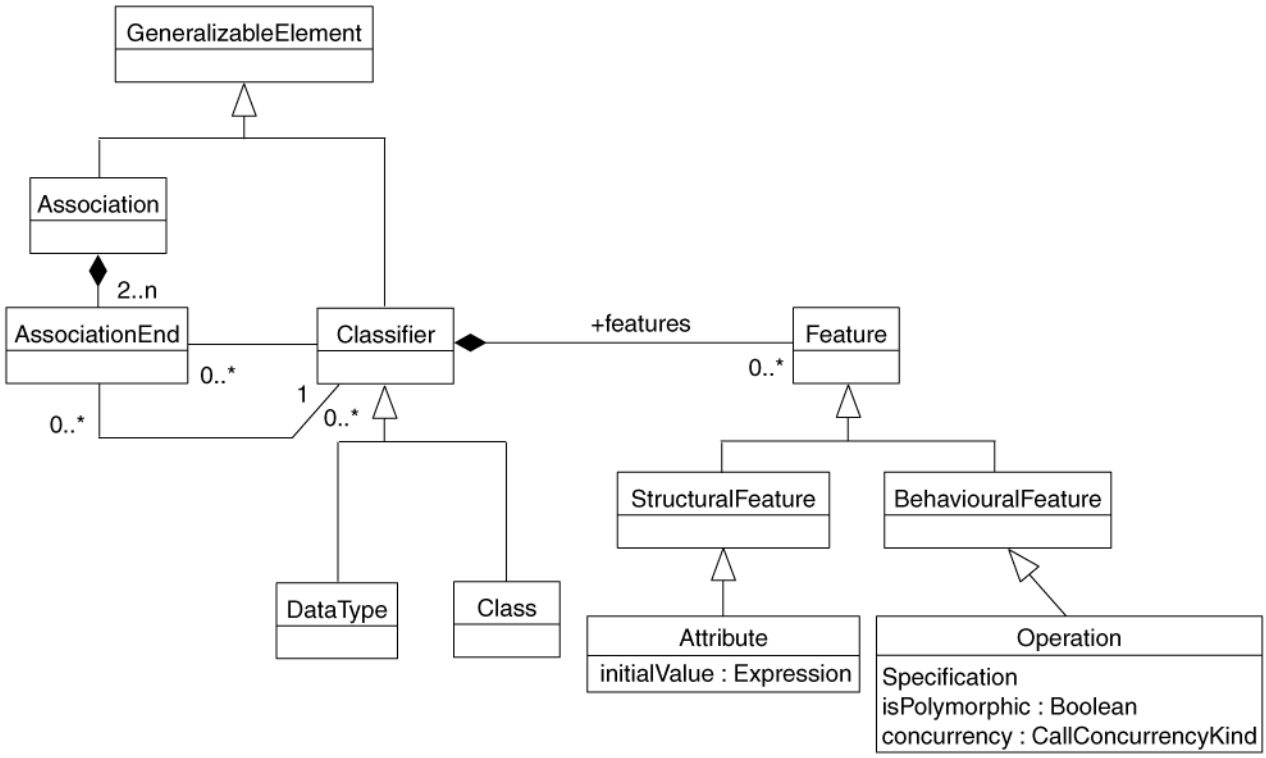


Figure B.7 — UML core metamodel

The UML core metamodel is sufficiently similar to the ISO 19109 General Feature Model, with small changes, that it can be used as a baseline for meeting the requirements of the General Feature Model.

NOTE Feature in Figure B.7 is a different concept from feature as defined in the General Feature Model.

UML has been developed in a metamodel framework similar to CSMF, as shown in Figure B.8.

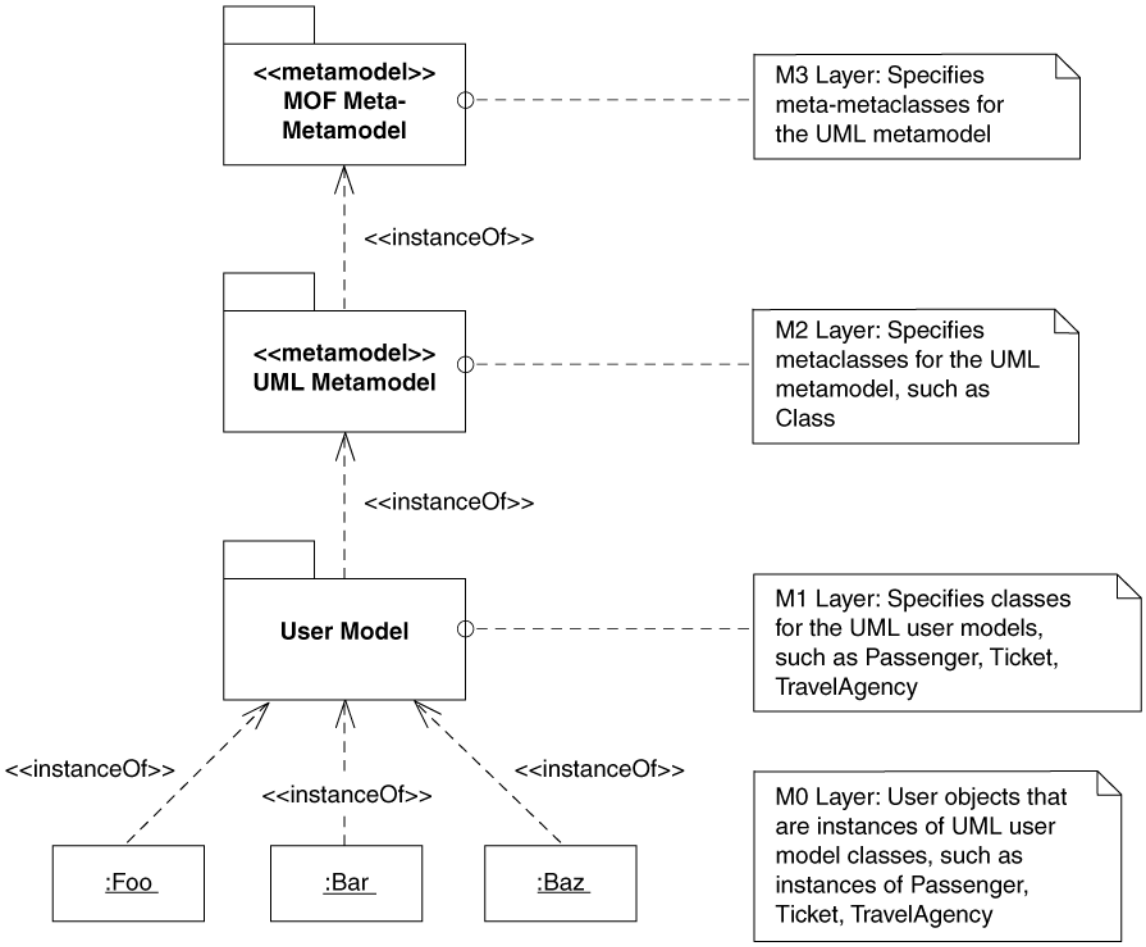


Figure B.8 — The metamodel architecture for UML

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

The following is a short introduction to the goals and concepts in each of the ODP viewpoints, and the application of these viewpoints within the ISO geographic information standards (see Figure B.9).

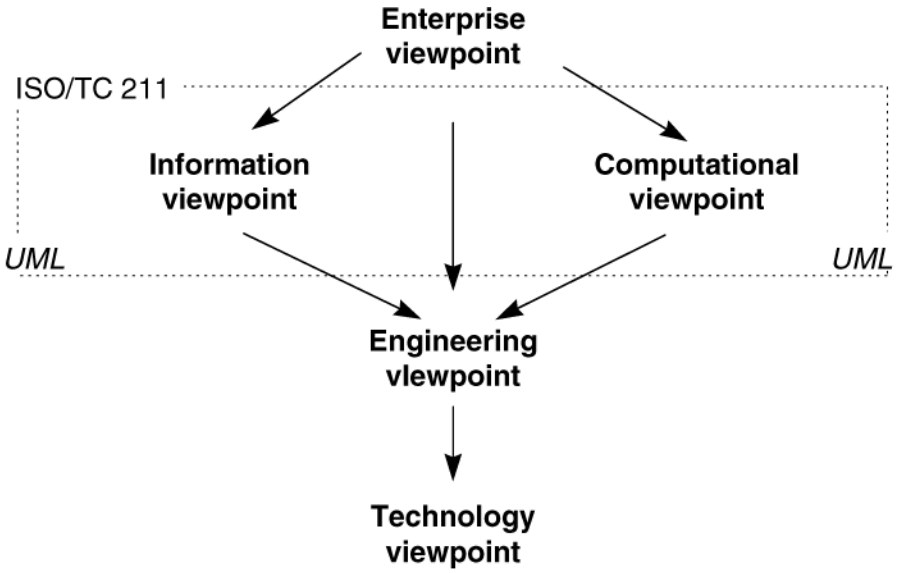


Figure B.9 — Viewpoints in the ISO RM ODP model

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

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

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

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

the ISO geographic information standards focus on implementation neutral models, there is little emphasis on this viewpoint. It is assumed that the handling of various distribution transparencies will be done in the context of making implementation models in the technology viewpoint.

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

Figure B.10 shows the architectural reference model from ISO 19101. A conceptual schema language is used to describe both the services and the spatial domain models that are shared or transferred. The spatial domain models include the spatial schema, the temporal schema and the metadata schema.

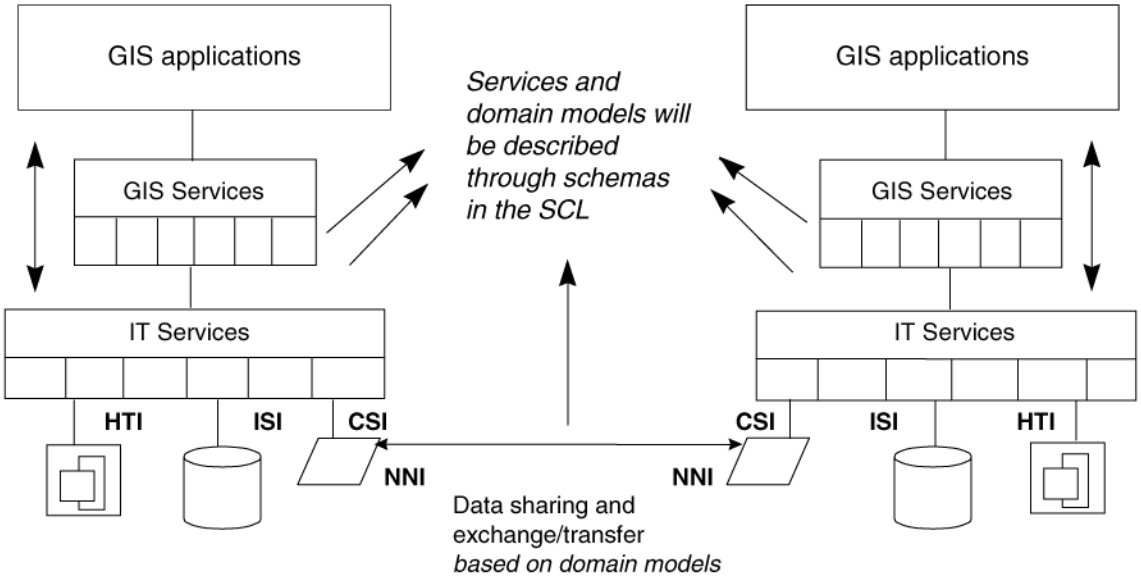


Figure B.10 — ISO 19101 architectural reference model

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

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

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

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

The geographic domain models within the ISO geographic information standards have been developed as conceptual models and then further refined to specification models to be a suitable implementation neutral basis for system representation and data exchange. The service models within the ISO geographic information standards have been developed as specification models to reflect required system interfaces in an

implementation neutral way. Some parts of the ISO geographic information standards also further describe implementation models for given specification models, to define corresponding implementation and platform specific models.

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

Annex C
(informative)

Modeling guidelines

C.1 Guidelines for modeling with UML

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

Object Management Group (OMG) in January 1999 adopted the XMI specification for exchanging UML models using XML. In the future, the XMI specification will allow for interoperability among UML tools. A new diagram interchange format based on XMI, also including graphical layout, is expected to be standardized by OMG. In the future, the ISO geographic information standards UML models should be made available using the XMI specification and the new diagram interchange format. In the meantime, models should be made electronically available using modeling tools. Many UML modeling tools are able to read formats of other tools.

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

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

— Business/Domain models

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

— Requirements models

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

— Analysis/Architecture models

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

— Design models (may be implementation neutral)

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

— Implementation models

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

— Test models

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

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

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

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

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

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

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

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

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

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

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

— All general associations should have role names, excluding composition (closed diamond), aggregation (open diamond), and generalization (triangle or inheritance) associations. The assigned names of associations need not be shown on class diagrams, when at least one role name is shown on that association.

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

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

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

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

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

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

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

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

C.2 Guidelines for information modeling

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

Phases of modeling

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

The following modeling phases are recommended:

— Phase 0: Identify scope and context.

— Phase 1: Identify basic classes.

— Phase 2: Specify relationships, attributes and operations.

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

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

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

Phase 0: Identify scope and context

Tasks:

1) Identify the goals and scope of the models.

2) Identify related models and the wider context.

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

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

Phase 1a: Identify basic classes

Tasks:

1) Identify generic and specific concepts/terms.

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

3) Watch out for synonyms and homonyms.

4) Develop a glossary of concepts/terms.

Major questions that need to be answered:

— What are the classes according to the scope?

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

— Can we categorize the classes?

— What local consistency constraints apply to the classes?

— Are all the classes and attributes documented?

— Is the use of basic data types correct?

Minor concerns:

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

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

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

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

Phase 1b: Consistency with rules for application schema

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

Tasks:

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

Phase 2: Specify relationships, attributes and possibly operations

Major questions to be asked:

— What are the relationships between the classes?

— Can we categorize the classes?

— Is the use of basic data types correct?

— Are additional attributes required to characterize a class?

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

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

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

— Are all the classes, relationships and attributes documented?

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

Minor questions:

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

Deliverable: UML class model

Phase 3: Completion of constraints

Major questions to be asked:

— What are the global consistency rules?

— Are all existence dependencies captured?

— Are all other cardinality constraints captured?

— Are all the constraints captured?

— Is the model well partitioned?

Deliverable: English and OCL-based documentation of all constraints.

Phase 4: Model definition harmonization

Do harmonization with sub-models and other parts.

When developing a large information model as in the ISO geographic information standards, the work is broken down into separate work items. This is done so that the series of standards can be done in parallel. Thus, there exist a number of modeling groups that develop individual pieces of the ISO geographic information standards of standards, e.g. quality schema, metadata schema, spatial schema, etc. This work arrangement facilitates the rate of progress, but at a cost. The end result is that the different models may not fit together into an integrated information model, if not extra steps are taken to harmonize models.

Major questions to be asked when comparing with other schemas:

— Are there redundancies or conflicts?

— Are there any ambiguities?

— Are the models complete?

— Do the abstraction levels match?

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

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

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

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

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

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

— Have constraints been defined according to the guidelines?

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

C.3 Guidelines for service modeling

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

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

The following modeling phases are recommended:

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

— Phase 1: Identify basic service responsibilities.

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

— Phase 3: Completion of constraints on operations.

— Phase 4: Service definition harmonization.

Phase 0: Identify scope and context – use cases

Tasks:

1) Identify the goals of the service.

2) Identify related services and the wider context.

Major questions that need to be answered:

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

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

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

Phase 1: Identify basic service responsibilities

Tasks:

1) Identify the basic responsibilities of a service.

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

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

4) Specify collaboration/sequence diagrams.

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

Phase 2: Specify operations, attributes and service relationships

For each identified interface, do the following:

— Specify operations with input and output arguments and exceptions.

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

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

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

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

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

Minor questions:

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

Deliverable: Graphical UML class model with operation signatures.

Phase 3: Completion of constraints on operations

Major questions to be asked for each operation:

— What does this operation mean (explain semantics)?

— What does each parameter mean?

— Are there any constraints or preconditions on parameters?

— What does each user exception mean?

— Are standard exceptions used? What does each mean?

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

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

Deliverable: English and OCL-based documentation of all operational constraints.

Phase 4: Service definition harmonization

Major questions:

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

— Are there any ambiguities?

— Are the models complete?

— Do the abstraction levels match?

Deliverable: Revised UML models with updated documentation.

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

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

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

— Has UML been used according to the UML Profile?

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

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

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

C.4 Model harmonization

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

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

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

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

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

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

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

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

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

Checklist

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

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

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

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

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

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

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

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

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

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

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

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

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

Deliverable: Revised UML models.

Annex D
(informative)

Introduction to UML

D.1 Introduction

This annex is provided as a convenient introduction to the basic parts of UML that are used within and prescribed by the ISO geographic information standards. The normative document for use of UML is ISO/IEC 19501.

This annex is structured as follows:

— General usage of UML,

— Classes,

— Attributes,

— Data types,

— Operations,

— Relationships and associations,

— Stereotypes and tagged values,

— Optional, conditional and mandatory attributes and associations,

— Naming and name spaces,

— Packages,

— Notes, and

— Constraints.

D.2 General usage of UML

UML should be used in a manner that is consistent with ISO/IEC 19501. Books, such as “UML User Guide” [1] and “UML Reference Manual” [2] contain further information. The book “UML Distilled” [4] is a shorter introductory text.

D.3 Classes

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

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

UML defines two special kinds of classes through the stereotypes <<Interface>> and <<Type>>.

An <<Interface>>, see Figure D.1, is a named set of operations that characterizes the behaviour of an element. An <<Interface>> can also be shown as a lollipop icon. An <<Interface>> in UML contains no attributes and does not specify its implementation.

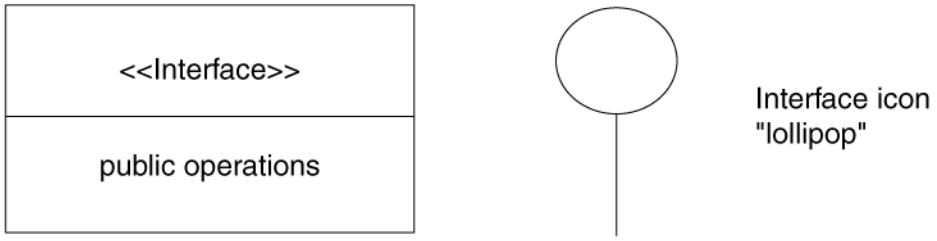


Figure D.1 — The stereotype <<Interface>>

A <<Type>>, see Figure D.2, is a stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A type may have abstract attributes and associations. To say that attributes are abstract means that their specification does not imply that they have to be concretely implemented as instance variables.

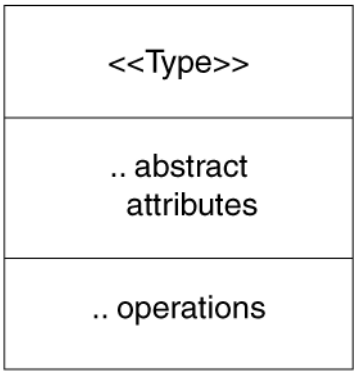


Figure D.2 — The stereotype <<Type>>

An <<Interface>> cannot have attributes, however, the notion of abstract attributes in a <<Type>> can be used as the place to describe the necessary abstract state. For instance, a point can have an abstract state for x and y coordinate, but an actual representation based on radian and degree that is used to calculate x and y. For objects that may be passed “by value” such as in the transfer of features, it is necessary that sufficient information be provided about the abstract state. This means that an “interface only” based specification cannot be used. The stereotype <<Type>> can be used for the specification of interfaces and abstract state.

Interfaces and types can be used according to the UML standard stereotypes, meaning set of operations, and set of operations and abstract attributes respectively.

An abstract class defines a polymorphic object class and cannot be instantiated. An abstract class differs from an <<Interface>> in that it may have attributes and may implement some of its operations. An <<Interface>> is equivalent to an abstract class with no attributes and no implementation of operations. An Abstract class is specified by having the classname in italics, or by the tagged value {Abstract} placed next to the classname.

The visibility of an attribute, operation, or association end defined in the context of a class may be limited in various ways. Private visibility means that an attribute or operation is only visible within the class within which it is defined; an association is only navigable from the class at its source end. Protected visibility ('#') means an attribute or operation is visible only within the class within which it is defined and to its subclasses; an association is only navigable from the class at its source end and from its subclasses. Public visibility ('+') means that an attribute or operation is visible externally to any class within the same package as the class within which it is defined; an association is navigable from any class associated with the class within which it is defined. Private and protected visibilities are normally not used in the ISO geographic information standards. Figure D.3 shows symbols for derived element, visibility and class level definitions.

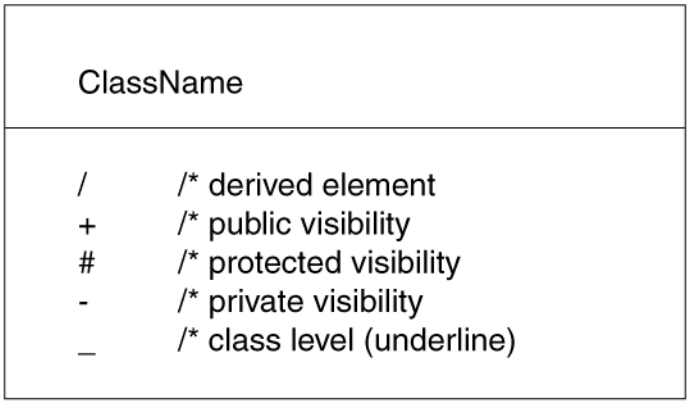


Figure D.3 — Symbols for derived element, visibility and class level definitions

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

- private

protected

+ public

Some UML tools use various graphical icons for the visibility symbols, but UML standard symbols are used in the ISO geographic information standards.

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

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

D.4 Attributes

UML notation for an attribute has the form:

[<<stereotype>>] [visibility] name [multiplicity] [:type] [= initial value] [{property-string}]

where optional elements are enclosed in brackets, and:

stereotype is the name of the stereotype (D.8) to which the attribute belongs, if any.

visibility specifies the visibility (D.3) of the attribute.

name is a character string that identifies the name of the attribute.

multiplicity specifies the number of values (D.7.2) that an instance of a class may have for a given attribute.

type-expression identifies the data type of the attribute.

initial value specifies the default value for the attribute.

property-string contains any tagged values applied to the attribute.

EXAMPLES
+ centre : Point = (0,0) {frozen}
+ origin [0..1] : Point / * multiplicity 0..1 means that this is optional.

Properties can be user-defined. Some predefined properties are: changeable, addOnly, frozen (const).

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

A data type must always be specified; there is no default type.

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

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

D.5 Data types

Clause 6 defines the set of data types to be used.

D.6 Operations

Operations are presented in UML class diagrams in compliance with the UML Notation Guide.

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

visibility name (parameter-list) : return-type-expression [{ property-string }]

where optional elements are enclosed in brackets, and:

visibility specifies the visibility (D.3) of the operation.

name is a character string that identifies the name of the operation.

parameter-list is a list of parameters (see below).

return-type-expression specifies the data type or types of the value returned by the operation.

property-string contains any tagged values applied to the operation.

An element of the property-list has the form:

[kind] name : type-expression [= default-value]

where:

kind is in, out, or inout, with the default in if absent.

name is the name of the parameter.

Type-expression identifies the data type of the parameter.

Default-value is an initial value for the parameter.

The values for *kind* have the following meanings:

in – The parameter value is read, but not modified, by the operation.

out – The parameter value is created by the operation and may be read upon the operation's return to the point of call.

inout – The parameter is read and modified by the operation and may be read upon the operation's return to the point of call. Exceptions can be defined as class elements using the stereotype <<Exception>>. Exceptions can be linked to operations through a dependency relationship.

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

Notes should be used with the stereotypes <<precondition>> and <<postcondition>> to describe pre- and post conditions for operations, and notes should be used with the stereotype <<invariant> to describe class invariants.

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

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

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

3) The objects modified or returned by the operation.

"An operation specifies a transformation on the state of the target object (and possibly the state of the rest of the system reachable from the target object), or a query that returns a value to the caller of the operation." [2]

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

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

D.7 Relationships and associations

D.7.1 Relationships

A relationship in UML is a reified semantic connection among model elements. Kinds of relationships include association, generalization, aggregation/composition, and several kinds grouped under dependency, see Figure D.4. In [2] a clear distinction is made between the general term "relationship," and the more specific term "association". Both are defined for class to class linkages, but association is reserved for those relationships that are in reality instance to instance linkages. "Generalization", "realization" and "dependency"

are class to class relationships. “Aggregation” and other object to object relationships, are more restrictively called “associations”. It is always appropriate to use the most restrictive term in any case, so in speaking of instantiable relationships, use the term “association.”

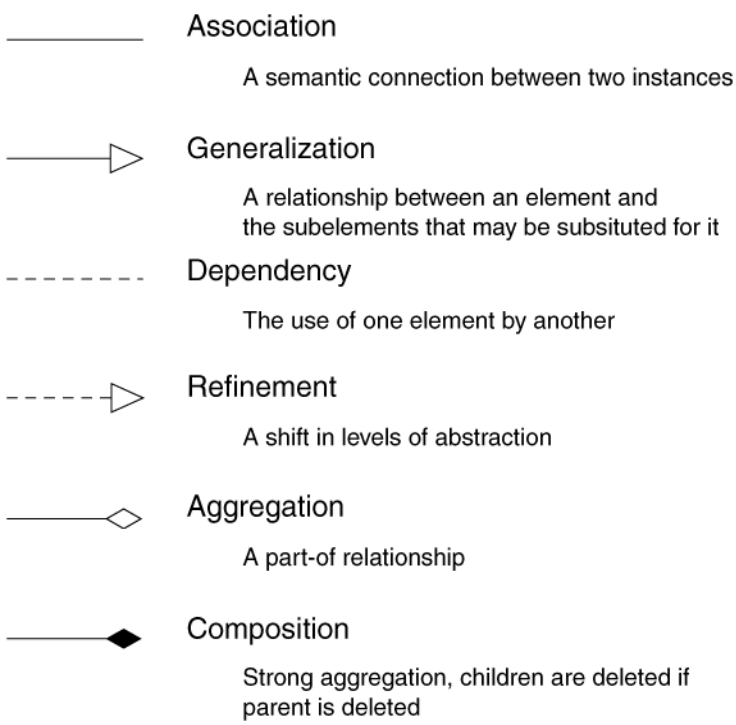


Figure D.4 — Different kinds of relationships

In the ISO geographic information standards, generalization, dependency, metarelationship, flow and refinement are used according to the standard UML notation and usage. In the following, the usage of association, aggregation and composition is described further.

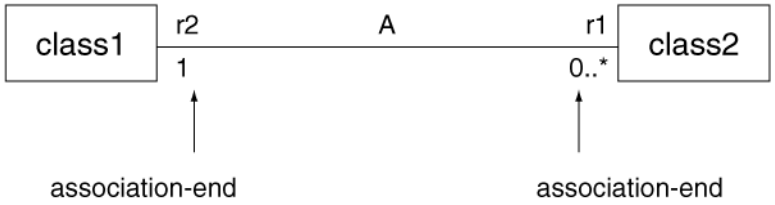
D.7.2 Association, composition and aggregation

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

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

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

Figure D.5 shows an association named “A” with its two respective association-ends. The role name is used to identify the end of an association; the role name **r1** identifies the association-end which is connected to the class named class2. In this example, the role name **r1** identifies the nature of the relation **class2** has to **class1** through the association **A**. The multiplicity of an association-end can be one of exactly-one (1), zero-or-one (0..1), one-or-more (1..*), zero-or-more (0..*) or an interval (n..m). Viewed from the class, the role name of the opposite association-end identifies the role of the target class. We say that **class2** has an association to **class1** that is identified by the role **r2** and which has a multiplicity of exactly one. The other way around, we can say that **class1** has an association to **class2** that is identified by the role name **r1** with multiplicity of zero-or-more. In the instance model we say that **class1** objects have a reference to zero-or-more **class2** objects and that **class2** objects have a reference to exactly one **class1** object.



Key

A association

r1,r2 role names

Figure D.5 — Association

Figure D.6 specifies the notation that is used to specify the number of instances that can participate at one end of an association. The same notation is used for attributes, but then in the field [multiplicity], as discussed in D.4.

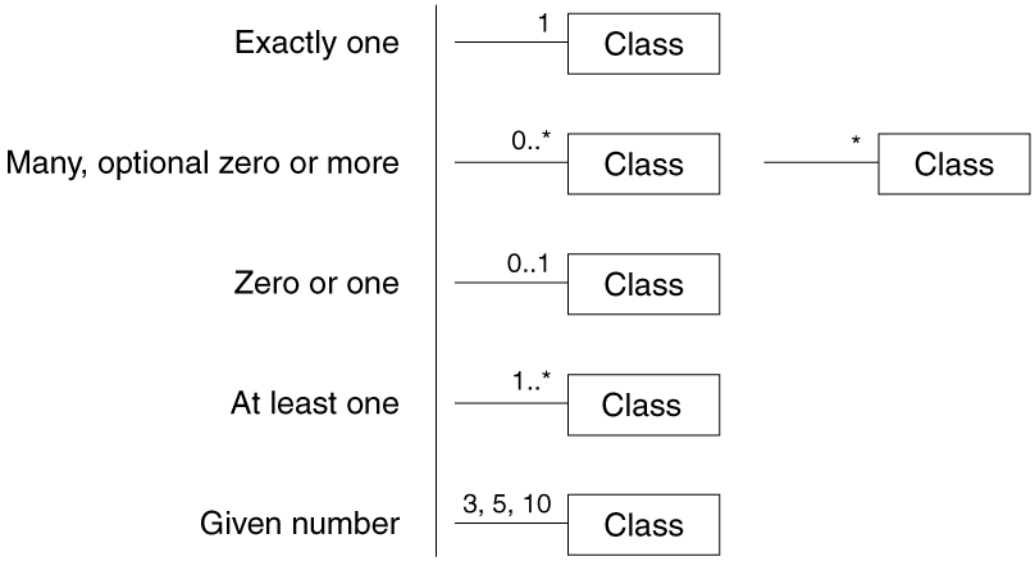
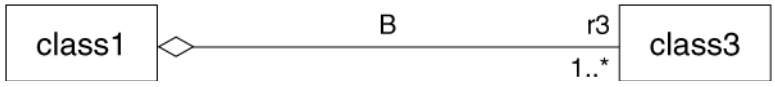


Figure D.6 — Specification of multiplicity/cardinality

An aggregation association is a relationship between two classes, in which one of the classes plays the role of container and the other plays the role of a containee. Figure D.7 shows an example of an aggregation. The open diamond-shaped aggregation symbol at the association-end close to **class1** indicates that **class1** is an aggregation consisting of **class3**. The meaning of this is that **class3** is a part of **class1**. In the instance model, **class1** objects will contain one-or-more **class3** objects. The aggregation association should be used when the containee objects, which represent the parts of a container object, can exist without the container object. Aggregation is a symbolic short-form for the part-of association but does not have explicit semantics. It allows multiple aggregations to share the same objects. If a stronger aggregation semantics is required, composition should be used as described below. It is possible also to define role name and multiplicity at the diamond shaped end as well.



Key

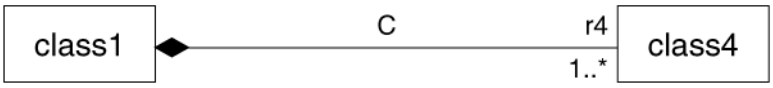
B association

r3 role name

Figure D.7 — Aggregation

A composition association is a strong aggregation. In a composition association, if a container object is deleted then all of its containee objects are deleted as well. The composition association should be used when the objects representing the parts of a container object cannot exist without the container object. Figure D.8 shows a composition association, in which the diamond-shaped composition symbol has a solid fill. Here **class1** objects consist of one-or-more **class4** objects, and the **class4** objects cannot exist unless the **class1** object also exists. The required (implied) cardinality for the owner class is always one. The containers, or parts, cannot be shared among multiple owners.

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



Key

C association

r4 role name

Figure D.8 — Composition (strong aggregation)

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

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

D.8 Stereotypes and tagged values

D.8.1 General

UML contains three extensibility mechanisms: stereotypes, tagged values and constraints, see D.12 for constraints. A UML stereotype is an extension mechanism for existing UML concepts. It is a model element that is used to classify (or mark) other UML elements so that they in some respect behave as if they were instances of new virtual or pseudo metamodel classes whose form is based on existing base metamodel classes. Stereotypes augment the classification mechanisms on the basis of the built-in UML metamodel class hierarchy. Therefore, names of new stereotypes must not clash with predefined metamodel elements or other stereotypes.

Tagged values are another extensibility mechanism of UML. A tagged value is a tag-value pair that can be used to add properties to any model element in UML, i.e. it can extend an arbitrary existing element in the UML metamodel or extend a stereotype.

D.8.2 Tagged values

Tagged values are Name (tag) separator (=) value (of the tag) properties on an element - relevant for code generation or configuration management (can be applied to all UML elements). Figure D.9 shows the UML metamodel for stereotypes and tagged values

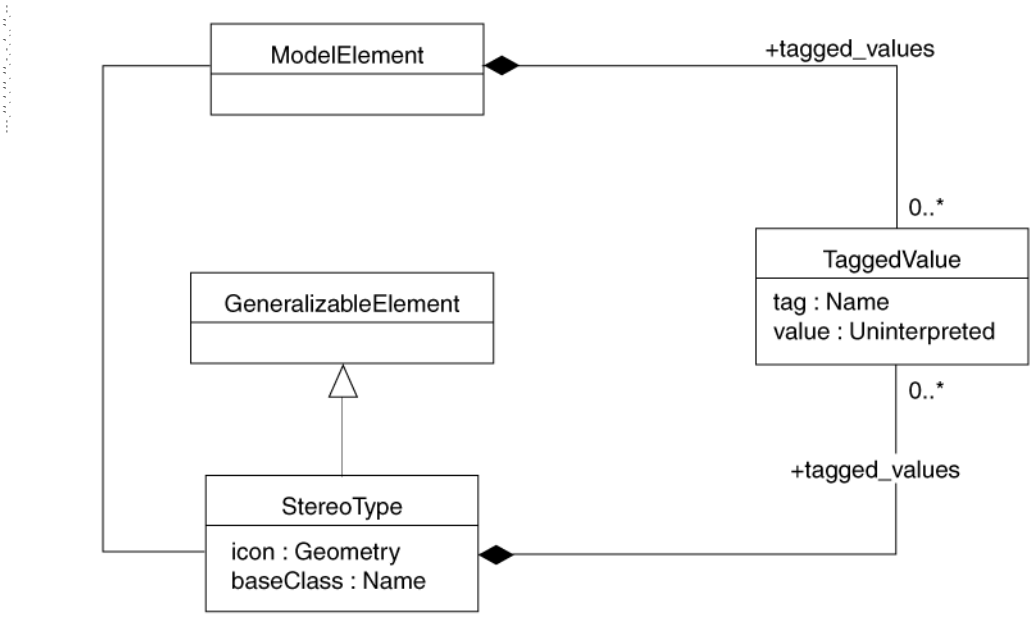


Figure D.9 — UML stereotypes and tagged values

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

D.8.3 Stereotypes

In this Technical Specification, the following stereotypes are used:

a) <<Interface>> a definition of a set of operations that is supported by objects having this interface.

b) <<Type>> a stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A type may have attributes and associations.

c) <<Control>> for a class whose primary purpose is to provide a service and does not represent particular data in itself — a standard UML extension from the Unified Software Development Process.

d) <<Entity>> meaning a class representing information-carrying, potentially persistent objects. A standard UML extension from the Unified Software Development Process.

e) <<Boundary>> meaning a class representing an external interface for a system. A standard UML extension from the Unified Software Development Process.

f) <<Enumeration>> A data type whose instances form a list of named literal values. Both the enumeration name and its literal values are declared. Enumeration means a short list of well-understood potential values within a class. Classic examples are Boolean that has only 2 (or 3) potential values TRUE, FALSE (and NULL). Most enumerations will be encoded as a sequential set of Integers, unless specified otherwise. The actual encoding is normally only of use to the programming language compilers.

g) <<Exception>> a signal the underlying execution machinery raises in response to behavioural faults.

h) <<MetaClass>> A class whose instances are classes. Metaclasses are typically used in the construction of metamodels. The meaning of metaclass is an object class whose primary purpose is to hold metadata about another class. For example, “FeatureType” and “AttributeType” are metaclasses for “Feature” and “Attribute”.

i) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). Data types include primitive predefined types and user-definable types. A DataType is thus a class with few or no operations whose primary purpose is to hold the abstract state of another class for transmittal, storage, encoding or persistent storage.

The stereotypes above are standard UML stereotypes used in geographic information.

D.9 Optional, conditional and mandatory attributes and associations

Standard UML gives various ways to describe optional, conditional and mandatory attributes and associations. This Technical Specification specifies in more detail one way to do this.

D.10 Naming and name spaces

UML does not specify any strict rules for naming and name spaces. This is elaborated more in Clause 6.

D.11 Packages

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

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

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

D.12 Notes

Note boxes are used to comment on the model in general or specific item (i.e. class or association) of the model in particular, see Figure D.10. They can also be used for specifying constraints and conditions (see 6.13).

Figure D.10 — Example note

D.13 Constraints

A constraint is shown as a text string in braces ({}). Constraints can be described using the Object Constraint Language [3], a set based language based on logical expressions of objects and object properties. An OCL constraint is a valid OCL expression of the boolean type, i.e. an expression with a *true* or *false value*.

Constraints can also be explained using text, see example in Table D.1 of constraints on Collection types.

Table D.1 — Textual constraints on Collection types

Collection types	Declaration	Description
Set	Set (T)	A collection that contains instances of a valid OCL type. Every element should be of the same type. A Set does not contain duplicate elements and any instance may only be present once. Set has no ordering of its elements.
Bag	Bag (T)	A Bag is like a Set, but may contain duplicated elements. Bag has no ordering of its elements.
Sequence	Sequence (T)	A Sequence is like a Bag but the elements are ordered.

Both OCL-expressions and text descriptions can be placed in UML NOTE boxes, see Figure D.11.

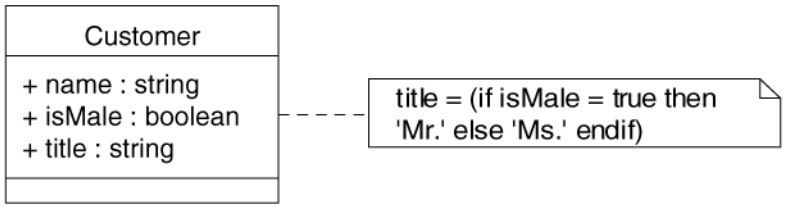


Figure D.11 — Example of OCL constraint attached to model elements

OCL can be used to describe constraints on various parts of a model, including constraints on:

— classes,

— attributes and associations of classes,

— operations of classes (pre- and post-conditions), and

— abstract classes.

Tables D.2 - D.5 describes standard operations on OCL types.

Table D.2 — Standard OCL boolean operations

operation	notation	result type
or	a or b	Boolean
and	a and b	Boolean
exclusive	A xor B	Boolean
negation	not a	Boolean
equals	a = b	Boolean
not equals	a <> b	Boolean
implies	a implies b	Boolean
if then else	if a then b else b' endif	Type of b and b'

Table D.3 — Standard OCL integer/real operations

operation	notation	result type
equals	a = b	Boolean
not equals	a <> b	Boolean
less	a < b	Boolean
more	a > b	Boolean
less or equal	a <= b	Boolean
more or equal	a >= b	Boolean
plus	a + b	Integer or Real
minus	a - b	Integer or Real
multiplication	a * b	Integer or Real
division	a / b	Real
modulus	a.mod(b)	Integer
integer division	a.div(b)	Integer
absolute value	a.abs	Integer or Real
maximum of a and b	a.max(b)	Integer or Real
minimum of a and b	a.min(b)	Integer or Real
round	a.floor	Integer

EXAMPLE
(3.2).floor / 3 = 1
1.175 * (-8.9).abs — 10
7.max(3) = 7.

Table D.4 — OCL string operations

operation	expression	result type
concatenation	string.concat(string)	String
size	string.size	Integer
to lower case	string.toLower	String
to upper case	string.toUpper	String
substring	string.substring(int, int)	String
equals	string1 = string2	Boolean
not equals	string1 <> string2	Boolean

EXAMPLE
'ISO/TC211'.size = 9
('ISO' = 'ISO') = true
'ISO/'.concat ('TC211') = 'ISO/TC211'.

Table D.5 — OCL collection operations

operation	description
size	The number of elements in the collection.
count(object)	The number of occurrences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter are present in the current collection.
isEmpty	True if the collection contains no elements.
notEmpty	True if the collection contains one or more elements.
iterate(expression)	Expression is evaluated for every element in the collection. The resulttype depends on the expression.
sum()	The addition of all elements in the collection. The elements must be of a type supporting addition (like Real, Integer).
exists(expression)	True if expression is true for at least one element in the collection.
forAll(expression)	True if for all elements expression is true.

User defined attributes from a UML model can be used freely in an OCL expression. User defined operations can be used only if they have no side effects, i.e. if they don't alter the state of the object.

OCL also offers operations on objects ('=' (same object) '<>' (different objects) 'oclType' (type of object), 'oclIsTypeOf' (true if direct type), 'oclIsKindOf' (true if type or subtype of), 'oclAsType' (returns type)) and on types (''name', 'attributes', 'associationEnds', 'operations', 'supertypes', 'allSuperTypes', 'allInstances').

Bibliography

[1] BOOCH, G., JACOBSSON, I. and RUMBAUGH, J. *UML User Guide*, 1999, Addison-Wesley ISBN 0-201-57168-4

[2] RUMBAUGH, J., BOOCH, G. and JACOBSSON, I. *UML Reference Manual*, 1999, Addison-Wesley ISBN 0-201-57168-X

[3] WARMER, J.B. and KLEPPE, A.G. *The Object Constraint Language — precise modeling with UML*: Addison-Wesley Longman, Inc., 1997

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

[5] SCHENCK, D. and WILSON, P. *Information Modeling: The EXPRESS Way*, Oxford University Press, 1993

[6] CHEESMAN, J. and DANILES, J. *UML Components — A simple process for specifying component-based software*, Addison-Wesley, 2000, ISBN 0201708515

[7] *Catalog of OMG Modeling and Metadata Specifications*. Available at <http://www.omg.org/technology/documents/modeling_spec_catalog.htm>

[8] *UML Resource Page*. Available at <http://www.uml.org>

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

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

[11] ISO 19109:2005, *Geographic information — Rules for application schema*

[12] ISO 19110:2005, *Geographic information — Methodology for feature cataloguing*

[13] ISO 19115:2003, *Geographic information — Metadata*

[14] ISO 19118:2005, *Geographic information — Encoding*

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

[16] ISO 19136:—[1)], *Geographic information — Geography Markup Language (GML)*

[17] ISO/IEC 11404:1996, *Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes*

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

[19] ISO/IEC 8859, all parts, *Information technology — 8-bit single-byte coded graphic character sets*

[20] ISO 8601:2000, *Data elements and interchange formats — Information interchange — Representation of dates and times*

[21] ISO 3166-1:1997, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

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

1) To be published.

ICS 35.240.70

Price based on 67 pages