TSMAD27-4.3.1B

S-100 – Part 1

Conceptual Schema Language

Contents

1-1 Scope 1
1-2 Conformance 1
1-3 Normative references 1
1-4 The S-100 UML Profile 2
1-4.1 Introduction 2
1-4.2 General usage of UML 2
1-4.3 Classes 2
1-4.4 Attributes 3
1-4.5 Basic data types 3
1-4.5.1 General considerations 3
1-4.5.2 Primitive types 3
1-4.5.3 Complex types 4
1-4.6 Predefined derived types 8
1-4.7 Codelist types 8
1-4.8 Enumerated types 10
1-4.9 Relationships and associations 10
1-4.9.1 Relationships 10
1-4.9.2 Association, composition and aggregation 11
1-4.10 Stereotypes 13
1-4.10.1 Use of standard UML stereotypes for class/classifier 13
1-4.11 Optional, conditional and mandatory – attributes and associations 14
1-4.12 Naming and name spaces 14
1-4.13 Notes 15
1-4.14 Packages 15
1-4.15 Documentation of models in S-100 16
~~1-1 Scope 1~~
~~1-2 Conformance 1~~
~~1-3 Normative references 1~~
~~1-4 The S-100 UML Profile 2~~
~~1-4.1 Introduction 2~~
~~1-4.2 General usage of UML 2~~
~~1-4.3 Classes 2~~
~~1-4.4 Attributes 3~~
~~1-4.5 Basic data types 3~~
~~1-4.5.1 General considerations 3~~
~~1-4.5.2 Primitive types 3~~
~~1-4.5.3 Complex types 4~~
~~1-4.6 Enumerated types 7~~
~~1-4.7 Relationships and associations 8~~
~~1-4.7.1 Relationships 8~~
~~1-4.7.2 Association, composition and aggregation 9~~
~~1-4.8 Stereotypes 11~~
~~1-4.8.1 Use of standard UML stereotypes for class/classifier 11~~
~~1-4.9 Optional, conditional and mandatory – attributes and associations 11~~
~~1-4.10 Naming and name spaces 12~~
~~1-4.11 Notes 13~~
~~1-4.12 Packages 13~~
~~1-4.13 Documentation of models in S-100 14~~

1-1 Scope

This Part defines the conceptual schema language and basic data types for use within the IHO community. It identifies the combination of the Unified Modeling Language (UML) static structure diagram, and a set of basic data type definitions as the conceptual schema language for specification of geographic information. (UML is a standardized general-purpose modelling language in the field of software engineering. It includes a set of graphical notation techniques to create abstract models of specific systems. UML combines the best practice from data modelling concepts such as entity relationship diagrams, work flow, object modelling and component modelling).

Secondly, this Part provides guidelines on how UML should be used to create standardized geographic information and service models that are a basis for achieving the goal of interoperability. Since it deals with the UML, a section with specific UML terms and definitions is provided, in addition to these terms being included in Annex 1 (Terms and Definitions).

1-2 Conformance

Any conceptual schema written for a specification that claims conformance to this part of S-100 shall conform to the rules set out in clause 5. This profile conforms to conformance class 2 of ISO 19106:2004.

1-3 Normative references

ISO 19103:2005(E), *Geographic information — Conceptual schema language*
ISO 8601:2004(E), *Data elements and interchange formats — Information interchange — Representation of dates and times*
ISO 19136: *Geographic Information – Geography Markup Language*
ISO 25964-1: *Information and documentation — Thesauri and interoperability with other vocabularies — Part 1: Thesauri for information retrieval.*
ISO 25964-2: *Information and documentation — Thesauri and interoperability with other vocabularies — Part 2: Interoperability with other vocabularies*
OGC 10-129r1: *Geographic Information – Geography Markup Language (GML) – Extended schemas and encoding rules*
OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2
RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. Internet Standard 66, IETF. URL: http://www.ietf.org/rfc/rfc3986.txt or http://www.rfc-editor.org/info/std66
RFC 2141, *URN Syntax*. R. Moats. IETF RFC 2141, May 1997. URL: http://www.rfc-editor.org/info/rfc2141
SKOS: *SKOS – Simple Knowledge Organization System – Reference*. W3C Recommendation, 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

1-4 The S-100 UML Profile

1-4.1 Introduction

This clause provides rules and guidelines on the use of UML within the field of geographic information.

The subclauses are structured as follows:

1) General usage of UML
2) Classes
3) Attributes
4) Basic data types
5) Enumerated types
6) Predefined derived types
7) Relationships and associations
8) Stereotypes
9) Optional, conditional and mandatory – attributes and associations
10) Naming and name spaces
11) Notes
12) Packages
13) Documentation of models

1-4.2 General usage of UML

UML (The Unified Modeling Language) shall be used in a manner that is consistent with UML 2. Normative models shall use class diagrams and package diagrams. Other UML diagram-types may be used informatively. All normative models shall contain complete definitions of attributes, associations, and appropriate data type definitions.

1-4.3 Classes

A class is a description of a set of objects that share the same attributes, operations, methods, relationships, behaviour and constraints. 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 on implementation between 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. In S-100 operations are not used. A class may participate in associations.

A class according to the S-100 parts is viewed as a specification and not as an implementation.

The use of multiple inheritance shall be minimized, because it tends to increase model complexity.

An Abstract class is specified by having the class name in italics.

1-4.4 Attributes

UML notation for an attribute has the form:
$_{opt}$visibility$_{opt}$ name : $_{opt}$package ::$_{opt}$ $_{opt}$type$_{opt}$ $_{opt}$[multiplicity] $_{opt}$ $_{opt}$= initial value$_{opt}$ $_{opt}${property-string}$_{opt}$

An attribute must be unique within the context of a class and its supertypes, or else be a derived attribute, i.e. an attribute redefined from a supertype.

The visibility of attributes is shown by the symbols in Table 1-1. Protected and private visibility is normally not used in the standard specifications. The appropriate visibility symbols shall be used. The same visibility symbols are used for associations.

Table 1-1 — Visibility of Attributes

Symbol	Description
+	Public visibility
#	Protected visibility
-	Private visibility
/	Derived Attribute

All attributes must be typed and the type must exist, the constructed/defined types. A type must always be specified, there is no default type.
If no explicit multiplicity is given, it is assumed to be 1.
An attribute may define a default value, which is used when an object of that type is created. Default values are defined by explicit default values in the UML definition of the attribute.
The following properties can be used:

- readOnly – the value of the attribute cannot be changed and must be initialised.
- ordered – applies to attributes of a multiplicity of more than one in which the order of the elements is meaningful and must be maintained.

EXAMPLES + center: Point = (0,0) {readOnly}
+ origin: Point [0..1] // multiplicity 0..1 means that this is optional
+ controlPoints : Point [2..*] {ordered}

1-4.5 Basic data types

1-4.5.1 General considerations

The basic data types are grouped into two categories,:

1) Primitive types: Fundamental types for representing values, e.g. CharacterString, Integer, Boolean, Date, Time, etc.

2) Complex types: A combination of types, e.g. a combination of measure types and units of measurement.

The repertoire of basic data types is described in the following subclauses.

1-4.5.2 Primitive types

The following primitive types are supported in the S-100 UML Diagrams.

Table 1-2 — Data Types

Name	Description
Integer	A signed integer number, the representation of an integer is encapsulation and usage dependent. EXAMPLE 29, -65547
PositiveInteger	An unsigned integer number greater than 0.
NonNegativeInteger	An unsigned integer number greater than or equal to 0
Real	A signed real (floating point) number consisting of a mantissa and an exponent, the representation of a real is encapsulation and usage dependent. EXAMPLE 23.501, -1.234E-4, -23.0
Boolean	A value representing binary logic. The value can be either true or false.
CharacterString	A CharacterString is an arbitrary-length sequence of characters including accents and special characters from repertoire of one of the adopted character sets
Date	A date gives values for year, month and day according to the Gregorian Calendar. Character encoding of a date is a string which shall follow the calendar date format (complete representation, basic format) for date specified by ISO 8601. EXAMPLE 19980918 (YYYYMMDD)
Time	A time is given by an hour, minute and second. Character encoding of a time is a string that follows the local time (complete representation, basic format) format defined in ISO 8601. Time zone according to UTC is optional. EXAMPLE 183059 or 183059+0100 or 183059Z The complete representation of the time of 27 minutes and 46 seconds past 15 hours locally in Geneva (in winter one hour ahead of UTC), and in New York (in winter five hours behind UTC), together with the indication of the difference between the time scale of local time and UTC, are used as examples. Geneva: 1527460100 New York: 1527460500
DateTime	A DateTime is a combination of a date and a time type. Character encoding of a DateTime shall follow ISO 8601 (see above). EXAMPLE: 19850412T101530
TruncatedDateTime	A TruncatedDateTime allows a partial TM Position to be given. At least one of the following components must be present with omitted elements replaced by the appropriate number of hyphens. year – integer between 0000 - 9999 month – integer between 01-12 day – integer between 01 and 28, 29, 30, or 31 depending on year and month values time – Time type (see above)

1-4.5.3 Complex types

1-4.5.3.1 UnlimitedInteger

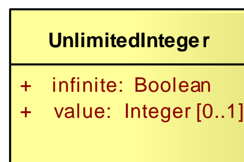

Figure 1-1 - UnlimitedInteger

A signed integer number whose value may be infinite.

1-4.5.3.2 Matrix

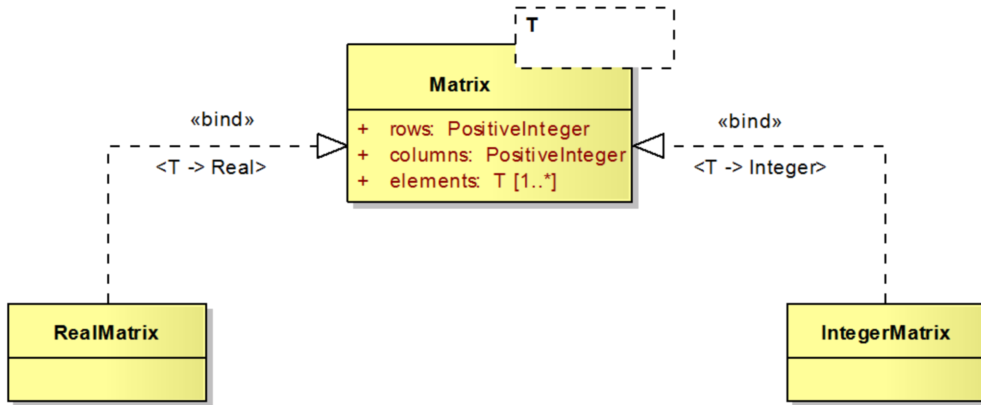

Figure 1-2 – Matrix

A grid of either real or integer elements.

1-4.5.3.3 S100_Multiplicity

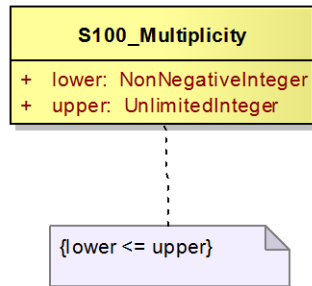

Figure 1-3 – S100_Multiplicity

Defines a multiplicity range from lower to upper. The upper boundary may be infinite.

1-4.5.3.4 S100_NumericRange

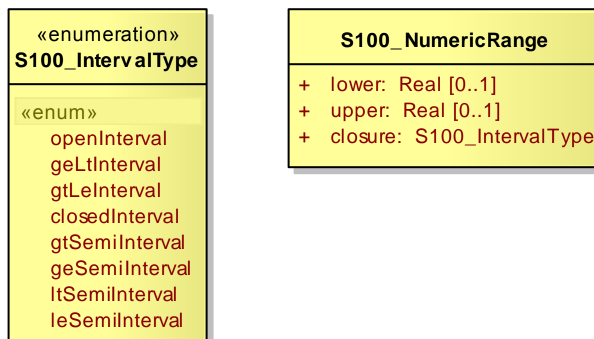

Figure 1-4 – S100_NumericRange

Specifies a numeric interval by its lower and upper boundary and the closure type of the interval.

NOTE The attribute **lower** must be used for all closures except **ltSemiInterval** or **leSemiInterval**. The attribute **upper** must be used for all closures except **gtSemiInterval** or **geSemiInterval**.

NOTE A single-value interval shall be encoded with **upper** = **lower** and set **closure** to **closedInterval**.

The closure of the interval is defined by the enumeration S100_IntervalType. The literals have the following meaning:

Table 1-3 — Interval Types

Name	Description	Notation	Definition (where a ≤ b)
openInterval	The open interval	*(a,b)*	$a < x < b$
geLtInterval	The right half-open interval	*[a,b)*	$a \le x < b$
gtLeInterval	The left half-open interval	*(a,b]*	$a < x \le b$
closedInterval	The closed interval	*[a,b]*	$a \le x \le b$
gtSemiInterval	The left half-open ray	*(a,∞)*	$a < x$
geSemiInterval	The left closed ray	*[a,∞)*	$a \le x$
ltSemiInterval	The right half-open ray	*(-∞,a)*	$x < a$
leSemiInterval	The right closed ray	*(-∞,a]*	$x \le a$

NOTE Intervals using the round brackets (or) as in the general interval (a,b) or specific examples (-1,3) and (2,4) are called **open intervals** and the endpoints are not included in the set. Intervals using the square brackets [or] as in the general interval [a,b] or specific examples [-1,3] and [2,4] are called **closed intervals** and the endpoints are included in the set. Intervals using both square and round brackets [and) or (and] as in the general intervals (a,b] and [a,b) or specific examples [-1,3) and (2,4] are called **half-closed intervals** or **half-open intervals**.

NOTE Intervals that have one of $\pm\infty$ as an end point are called rays or half-lines.

EXAMPLE The interval "(10,42)" indicates the set of all real numbers between 10 and 42 but does *not* include 10 or 42, the first and last numbers of the interval, respectively. The interval "[10,42]" includes every number between 10 and 42 *as well as* 10 and 42.

1-4.5.3.5 S100_UnitOfMeasure

A unit of measurement is a well defined comparator for a magnitude.

In S-100 a unit of measure is comprised of a name and optionally of a definition and a symbol.

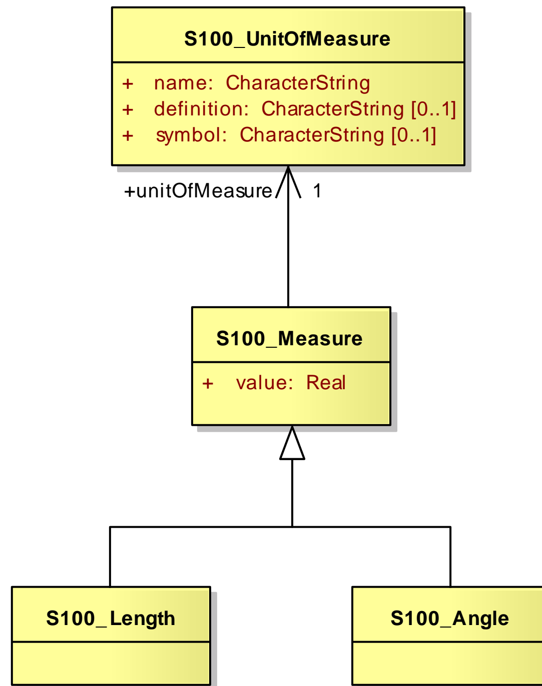

Figure 1-5 – S100_UnitOfMeasure

1-4.5.3.6 S100_Measure

A measure is the result of a measurement. A measurement is the estimation of the magnitude of some characteristic of an entity, such as its length or weight, relative to a unit of measurement. A measure consists of the actual magnitude (the value) and the unit of measurement.

1-4.5.3.7 S100_Length

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

1-4.5.3.8 S100_Angle

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

1-4.5.3.9 S100_IndeterminateDateTime

An indeterminate instant related by a specified temporal relation to a date and time specified in truncated date-time format. The temporal relations allowed are ‘before’ and ‘after’ and indicate respectively that the instant is before or after the time instant specified by the date-time component.

Figure 1-6 – S100_IndeterminateDateTime

1-4.6 Predefined derived types

Derived types are derived from the basic types or other derived types by restriction of the range of allowed values. The following derived types are defined in S-100. Product specifications may define additional derived types.

Table 1-4 — Predefined Derived Types

Name	Description	Derived From
URI	A uniform resource identifier as defined in RFC 3986. Character encoding of a URI shall follow the syntax rules defined in RFC 3986. EXAMPLE http://registry.iho.int	CharacterString
URL	A uniform resource locator (URL) is a URI that provides a means of locating the resource by describing its primary access mechanism (RFC 3986). EXAMPLE http://registry.iho.int	URI
URN	A persistent, location-independent, resource identifier that follows the syntax and semantics for URNs specified in RFC 2141. EXAMPLE urn:iho:s101:1:0:0:AnchorageArea	URI

1-4.7 Codelist types

CodeList types may be used for open enumerations whose membership cannot be known at the level of the product specification, for reuse of information model fragments, or for more efficient catalogue management. Specifically, they may be used:

a) for enumerations whose members are not all knowable at the level of the application schema;

b) for lists defined or controlled by external authorities;

c) for lists common to multiple S-100 domains;

d) if the set of allowed values needs to be extended without a major revision of the data specification;

e) long lists of potential values which would clutter or bloat feature catalogues.

For example, ISO 19115 (Metadata) defines several codelists, because it needs to define enumerated types whose membership is determined by domain and circumstances (e.g., distribution media).

A codelist type declaration defines either:

- a list of valid key-value combinations (i.e., code-value mappings) with a provision for allowing user communities to provide allowed values in a specified format; or,
- a dictionary (vocabulary) of key-value combinations in a known format, identifiable by a Uniform Resource Identifier and which can be located by the application of standard modern techniques for locating resources.

Code lists are modelled as classes that are stereotyped as <<Codelist>>. Code lists of the first form must list the known literals as attributes. In the second form, no attributes are listed.

A CodeList classifier must have tagged values which define its representation and extensibility, and may have a tagged value which hints at the anticipated encoding. Figure 1-7 shows two examples of codelists – the Languages codelist is an example of a codelist modelled as an extensible enumeration (indicated by the tagged values *asDictionary=false* and *extensible=true*) and the Countries codelist is an example of a codelist modelled by an external dictionary (indicated by tagged value *asDictionary=true*) whose location is given by its *vocabulary* tagged value.

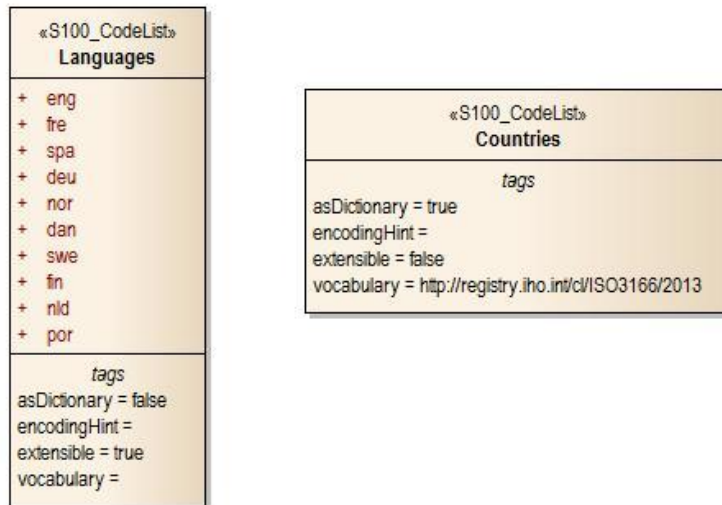

Figure 1-7 — Codelists

Implementations (and specific encodings) are allowed to depart from encoding hints. Different implementations may use different encoding schemes (and translation tables to other encoding schemes). For example preparation of a feature catalogue for an ISO 8211 encoding may transform a dictionary into an XML fragment which is merged into (or *Xinclude*'d in) the XML feature catalogue (obviously an additional procedure is needed for maintenance). This allows XML/GML encodings to use the dictionary while still allowing other encodings to function within their limitations.

The tagged values for S100 CodeLists are described in the table below.

Table 1-5 — Tagged values for codelists

Model	Tag			
	asDictionary	vocabulary	extensible	encodingHint
enumeration with pattern	false	(nil)	true: additional values permitted (default) false: additional values not permitted	enum: encode as ordinary enumeration (must have extensible=false) open: encode as union of list and pattern "other: \w{2,}" (default) + others as defined in product specifications
dictionary	true	(URI)	true: additional values permitted false: additional values not permitted (default)	enum: encode as ordinary enumeration (must have extensible=false) resource: encode as URI pointing to item in vocabulary (default) open: encode as URI identifying an item in either the specified vocabulary or another vocabulary + others as defined in product specification, or empty

1-4.71-4.8 Enumerated types

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

EXAMPLE

Figure 1-6 8 — Enumeration

Enumerations are modelled as classes that are stereotyped as <<enumeration>>. An enumeration class can only contain 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.

1-4.81-4.9 Relationships and associations

1-4.8.11-4.9.1 Relationships

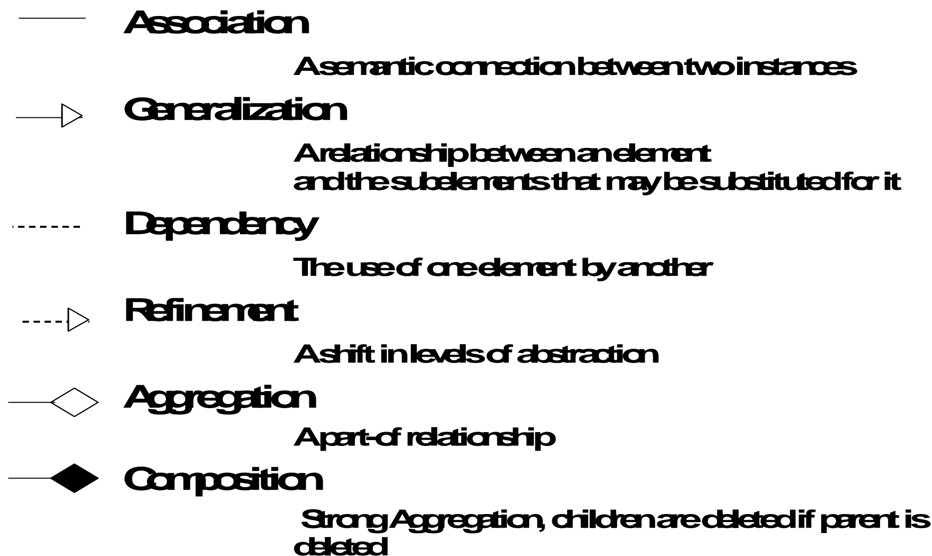

Figure 1-7 9 — Different kinds of relationships

A relationship in UML is a concrete semantic connection among model elements. Kinds of relationships include association, generalization, aggregation/composition, meta relationship, flow, and several kinds grouped under dependency. In ISO 19103 there is a clear distinction 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.”

In S-100, generalization, dependency 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.

~~1-4.8.2~~1-4.9.2 Association, composition and aggregation

An association in UML is the semantic relationship between two or more classifiers (e.g. 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 shall be used to represent a general relationship between two classes. The aggregation and composition associations shall be used to create part-whole relationships between two classes.

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

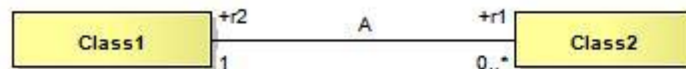

Figure 1-~~8~~10 — Association

Figure 1-10~~8~~ 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. 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 as 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.

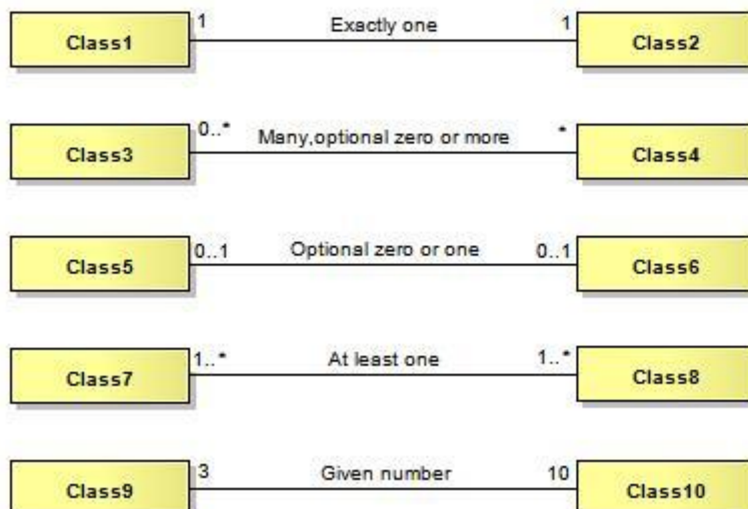

Figure 1-~~9~~ 11 — Specification of multiplicity

The number of instances that can participate at one end in an association (or attribute) is specified in Figure 1-11~~9~~.

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 1-12~~0~~ shows an example of an aggregation. The 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 shall be used when the containee objects (that 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 for sharing of the same objects in multiple aggregations. If a stronger aggregation semantics is required, composition shall be used as described below. It is possible also to define role name and multiplicity at the diamond shaped end as well.

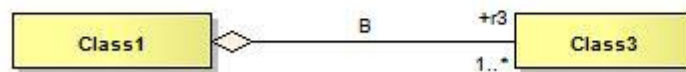

Figure 1-~~10~~ 12 — Aggregation

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 shall be used when the objects representing the parts of a container object, cannot exist without the container object. Figure 1-13~~0~~ 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) multiplicity for the owner class is always one. The containees, or parts, cannot be shared among multiple owners.

It is possible also to define role name at the diamond shaped end as well, but the multiplicity will always be at most one. Composition shall be used to have the semantic effect of containment. Composition should be used with care, in particular one should consider the different requirements from various application perspectives before introducing this constraint. 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.

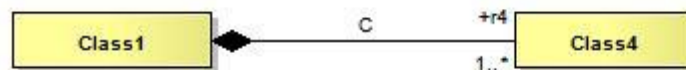

Figure 1-13~~1~~ — Composition (strong aggregation)

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.

All association ends (roles) representing the direction of a relationship must be named or else the association itself must be named. The name of an association end (the rolename) must be unique within the context of a class and its supertypes. The direction of an association must be specified. If the direction is not specified, it is assumed to be a two-way association. If one-way associations are intended, the direction of the association can be marked by an arrow at the end of the line. If only the association is named, the direction of the association shall be specified.

Every UML association has navigability attributes that indicate which player in the association has direct access to the association opposite role. The default logic for an unmarked association is that it is two-way. 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. 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 shall be minimized. One-way relations shall be used when that is all that is needed.

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 2-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 of secondary importance and actually are more for documentation purposes. Sometimes they can, however, be used for generating association-manager objects in environments that support associations as a first-class citizen concept.

Multiplicity refers to the number of relationships of a particular kind that an object can be involved in. If an association end were not navigable, putting a multiplicity constraint on it would require an implementation to track the use of association by other objects (or to be able to acquire the multiplicity through query). If this is important to the model, the association shall 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.

N-ary relationships, for N > 2 shall be avoided whenever possible, in order to reduce complexity. Multiplicity for associations are 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. The recommendation for S-100 is to use the association notation for all cases except for those involving attributes of basic data types.

~~1-4.9~~1-4.10 Stereotypes

~~1-4.9.1~~1-4.10.1 Use of standard UML stereotypes for class/classifier

In S-100 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) <<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. In S-100 Codelists taken from the ISO 19100 standards are classified as enumerations.

d) <<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”.'

e) <<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.

~~e)~~f) <<CodeList>> A data type whose instances form a list of named literals, some or all of whose members may not be known. The **CodeList** name is declared in the application schema. The list members may be described by either (i) a list of codes and corresponding literals augmented with a pattern allowing additional values conforming to a certain format, or (ii) a pointer to a resource consisting of a list of

code/literal mappings. The resource is called a vocabulary or dictionary. Tagged values attached to the **CodeList** declaration indicate which form is used and the location of the resource (generally as a URI). CodeLists should be used only when an enumeration is either unusable or inefficient (e.g., if the full list of values is not known to the specification authors or the list of allowed values is long, volatile, controlled by another authority, and/or shared by multiple domains

1-4.101-4.11 Optional, conditional and mandatory – attributes and associations

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

The default is mandatory which thus do not need to be specified. Where a multiplicity of 0..1 or 0..* is specified it means that this attribute may be present or may be omitted. A conditional attribute shall be shown as an optional attribute with a constraint statement in OCL. The condition shall be expressed as an OCL constraint in connection with the class declaration. This mean 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.

An attribute may be defined as conditional, meaning that it is optional depending on other attributes. The dependencies may be by existence-dependence of other (optional) attributes or by the values of other attributes. A conditional attribute is shown as optional with a conditional expression attached. The condition shall be written in a note directly associated with the attribute, or with the class and the name of the attribute on the first line. A conditional attribute shall never have a default value defined.

If unspecified, the default multiplicity for associations is 0..*, and the default multiplicity for attributes is 1.

1-4.111-4.12 Naming and name spaces

All classes shall have unique names. All classes shall be defined within a 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 name shall begin with a capital letter, such as “XnnnYmmm”.

To ensure global uniqueness of class names, all class names shall be defined with bi-alpha prefixes. Bialpha prefixes allows for the use of _ after, such as in GM_Object. The geometry model uses bialpha prefixes (GM and TP). Other prefixes should be defined for other areas.

The name of an association must be unique within the context of a class and its supertypes or else it must be derived.

Attribute names shall start with a lower-case letter.

Example: firstName, lastName.

Precise technical names should be used for attributes and operations to avoid confusion.

Example: alphaCodeIdentifier, dateOfLastChange

Documentation fields should be used extensively to describe element.

Don't reiterate class names inside the attribute names. Keep names short if possible.

Example: class S-100_WorkingGroup, attribute workingGroupName.

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 should:

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

Example: index not i

2) For attributes and association roles 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: computePartialDerivatives (not computepartialderivatives or COMPUTEPARTIALDERIVATIVES)

   Example: CoordinateTransformation (not coordinateTransformation)

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

   - numSegment instead of numberOfSegments
   - Equals instead of IsEqual
   - value() instead of getValue()
   - initObject instead of initializeObject
   - length() instead of computeLength()

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:

1) Although the model is case sensitive, all class name should be unique in a case insensitive manner.
2) Class name should be unique across the entire model (so as not to create a problem with many UML tools).
3) Package names should be unique across the entire model. (for the same reason).
4) Every effort should be applied to eliminate multiple classes instantiating the same concept.

1-4.121-4.13 Notes

Note boxes are used to comment on the model in general or on a specific item (i.e. class or association) of the model.

Figure 1-142 — Example note

1-4.131-4.14 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. A package shall be used to represent a schema.

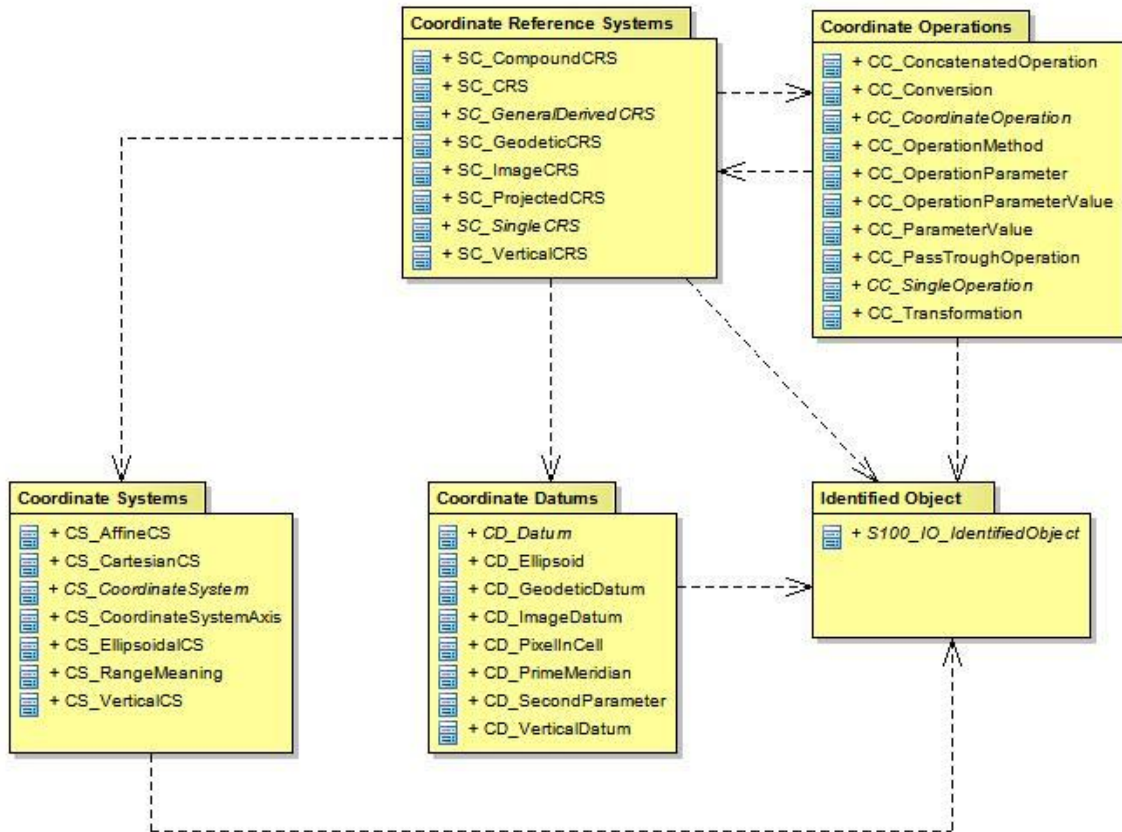

Figure 1-153 — Example package structure

The packages, classes and attributes in the schema model can be identified by a qualified name. The form of the qualified names is *name1 : :name2 : :name3*, where *name1* is the name of the outermost package, *name2* is a name which appears within the namespace of *name1*, and *name3* is a name that appears within the namespace of *name2*. The standard UML ": :" symbol shall be used as a name separator. 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.

1-4.141-4.15 Documentation of models in S-100

In addition to the diagrams, it is necessary to document the semantics of the model. The meaning of attributes, associations, operations and constraints needs to be explained. This is done by means of context tables. A context table is defined for each class; it has the following columns:

- Role Name
- Name
- Description
- Multiplicity
- Data Type
- Remarks

The Role Name column specifies what property of the class is described in this row. Possible values are:

- Class – The class itself
- Attribute – An attribute of that class
- Association – An association to another class
- Enumeration – An enumerated data type
- Literal – A value of an enumerated data type

The Name column contains the name of the property. For association this is the role name used for the given class. In the Description column the semantics of the property are given. The Multiplicity column contains the number of occurrences of the property in the class. This also describes which properties are mandatory and which are optional. The Data Type column contains the name of the data type of the property. In the Remarks column additional information about the property can be expressed. This includes constraints or conditions. For the documentation of enumerated types the Multiplicity and Data Type column are not used.

The following Example illustrates the use of context Tables:

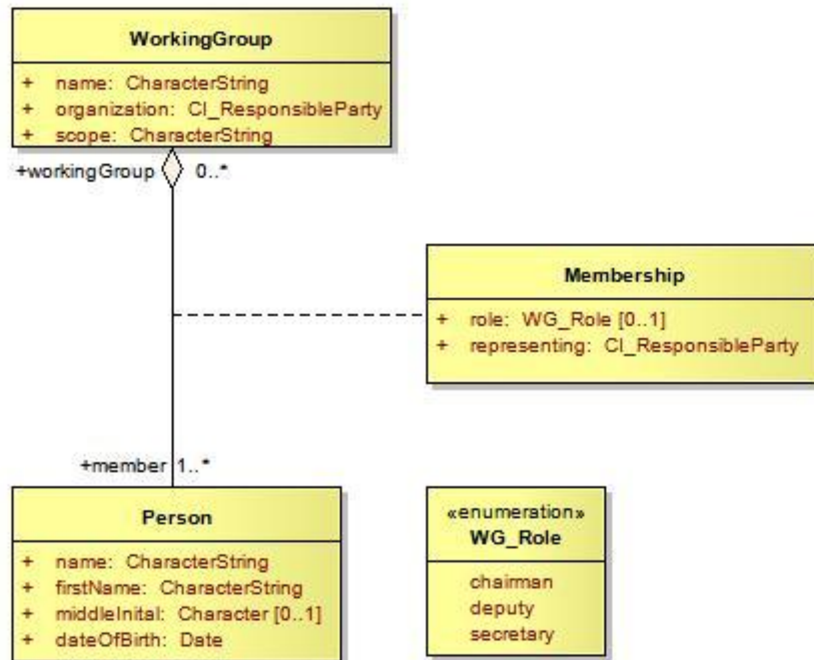

Figure 1-164

Role Name	Name	Description	Multiplicity	Data Type	Remarks
Class	WorkingGroup	A group of experts doing some useful work	-	-	
Attribute	name	The name of the working group	1	CharacterString	
Attribute	organization	The organization responsible for the working group	1	CI_ResponsibleParty	
Attribute	scope	The reason why so many people travel around the world	1	CharacterString	
Association	member	A person that is designated to contribute to the group	1..*	Person	

Role Name	Name	Description	Multiplicity	Data Type	Remarks
Class	Person	A human being	-	-	
Attribute	name	The name of the person	1	CharacterString	
Attribute	givenName	The first name of the person	1	CharacterString	
Attribute	middleInitial	The middle initial of the person	0..1	Character	
Attribute	dateOfBirth	The date when the person was born	1	Date	
Association	workingGroup	A working group the person contributes to	0..*	WorkingGroup	

Role Name	Name	Description	Multiplicity	Data Type	Remarks
Class	Membership	A class describing the membership of a person in a working group	-	-	
Attribute	role	The role that the person has in the working group	0..1	WG_Role	Ordinary member have no role
Attribute	representing	The organization which is represented by the person in the working group	1	CI_ResponsibleParty	

Role Name	Name	Description	Remarks
Enumeration	WG_Role	The roles people can have in a working group	
Literal	chairman	The gov'nor	
Literal	deputy	His best friend	
Literal	secretary	Poor man (or woman) has to have his (or her) fingers always on the keyboard	