INTERNATIONAL STANDARD

**ISO 19150-2**

First edition
2015-07-01

# Geographic information — Ontology —

## Part 2: Rules for developing ontologies in the Web Ontology Language (OWL)

*Information géographique — Ontologie —*

*Partie 2: Règles pour le développement d'ontologies dans le langage d'ontologie Web (OWL)*



Reference number
ISO 19150-2:2015(E)



ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

# Contents

Page

# Foreword

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

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

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

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

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

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

ISO 19150 consists of the following parts, under the general title *Geographic information — Ontology*:

— *Part 1: Framework*

— *Part 2: Rules for developing ontologies in the Web Ontology Language (OWL)*

*Semantic operators*, *Service ontology*, *Domain ontology registry* and *Service ontology registry* are planned to be covered in future Parts.

# Introduction

Fundamentally, ontology comes from philosophy and refers to the study of the nature of the world itself. The information technology and artificial intelligence communities borrowed this term of ontology for the explicit specification of a conceptualization.[2] Information technology and artificial intelligence consider that reality may be abstracted differently depending on the context from which "things" are perceived and, as such, recognize that multiple ontologies about the same part of reality may exist. In geographic information, ontology refers to a formal representation of phenomena of a universe of discourse with an underlying vocabulary including definitions and axioms that make the intended meaning explicit and describe phenomena and their interrelationships.[1] An ontology can be formalized differently ranging from weak to strong semantics: taxonomy, thesaurus, conceptual model, logical theory.[2]

Ontology is a fundamental notion for semantic interoperability on the Semantic Web since it defines the meaning of data and describes it in a format that machines and applications can read. As such, an application using data also has access to their inherent semantics through the ontology associated with it. Therefore, ontologies can support integration of heterogeneous data captured by different communities by relating them based on their semantic similarity. The W3C has proposed the Web Ontology Language (OWL) family of knowledge representation languages for authoring ontologies characterised by formal semantics on the Web.[4,11]

Semantics is an important topic in the field of geographic information. The meaning of geographic information is essential for their discovery, sharing, integration, and use. Geographic information standards have recognized this fact in the standards on rules for application schema (ISO 19109) and the methodology for feature cataloguing (ISO 19110),[7] which are both based on the General Feature Model (GFM). Basically, semantics relates phenomena and signs used to represent them (i.e. data) by the way of concepts. Typically, concepts are maintained in repositories called ontologies.

The ISO geographic information standards have chosen the conceptual modelling language UML[10,12] for the formal representation of abstraction of the reality. Additionally as introduced in ISO/TS 19150-1:2012, there is a need to provide formal representation of abstraction of the reality in OWL to support the Semantic Web. Accordingly, this part of ISO 19150 defines rules to convert UML static views of geographic information and application schemas into OWL ontologies in order to benefit and support interoperability of geographic information over the Semantic Web. These rules are required for:

— ontology description completeness;

— consistency in the set of OWL ontologies for geographic information;

— consistency in conversion of UML diagrams to OWL ontologies; and

— cohesion and unity between UML models and ontologies.

These rules are based on but also extend the OMG's Ontology Definition Metamodel.[11] OWL ontologies are complementary to UML static views and serve different purposes.

# Geographic information — Ontology —

# Part 2: Rules for developing ontologies in the Web Ontology Language (OWL)

## 1 Scope

This part of ISO 19150 defines rules and guidelines for the development of ontologies to support better the interoperability of geographic information over the Semantic Web. The Web Ontology Language (OWL) is the language adopted for ontologies.

This part of ISO 19150 defines the conversion of the UML static view modeling elements used in the ISO geographic information standards into OWL. It further defines conversion rules for describing application schemas based on the General Feature Model defined in ISO 19109 into OWL.

This part of ISO 19150 does not define semantics operators, rules for service ontologies, and does not develop any ontology.

## 2 Conformance

Any application ontology or profile claiming conformance with this part of ISO 19150 shall pass the requirements described in the abstract test suite, presented in Annex A.

The abstract test suite is organized in two conformance classes that address the following purposes:

— Conversion of a UML package from the ISO/TC 211 Harmonized Model to OWL (conformance class 19150-2owl-conf); and

— Formalization of an application schema in OWL (conformance class 19150-2app-conf).

## 3 Normative references

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

ISO 19103:—[1], *Geographic information — Conceptual schema language*

ISO 19107:2003, *Geographic information — Spatial schema*

ISO 19108:2002, *Geographic information — Temporal schema*

ISO 19109:—[2], *Geographic information — Rules for application schema*

ISO 19112:2003, *Geographic information — Spatial referencing by geographic identifiers*

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

ISO 19123:2005, *Geographic information — Schema for coverage geometry and functions*

1) To be published. (Revision of ISO/TS 19103:2005)

2) To be published. (Revision of ISO 19109:2005)

ISO 19156:2011, *Geographic information — Observations and measurements*

ISO 19157:2013, *Geographic information — Data quality*

W3C OWL 2, *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax* (W3C Recommendation 27 October 2009)

W3C OWL 2 RDF, *OWL 2 Web Ontology Language RDF-Based Semantics* (W3C Recommendation 27 October 2009)

W3C SKOS, *SKOS Simple Knowledge Organization System Reference* (W3C Recommendation 18 August 2009)

IETF RFC 5234, *Augmented BNF for Syntax Specifications: ABNF*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

# 4 Terms, definitions, abbreviations, and namespaces

## 4.1 Terms and definitions

**4.1.1**
**aggregation**
<UML> special form of *association* (4.1.6) that specifies a whole-part relationship between the aggregate (whole) and a component part

Note 1 to entry: See *composition* (4.1.12).

[SOURCE: ISO 19103:—[1)], 4.1]

**4.1.2**
**annotation**
<OWL> additional information associated to ontologies, entities, and axioms

[SOURCE: OWL]

**4.1.3**
**annotation property**
<OWL> element used to provide a textual *annotation* (4.1.2) for an *ontology* (4.1.29), axiom, or an IRI

[SOURCE: OWL]

**4.1.4**
**application ontology**
*ontology* (4.1.29) representing the concepts and relationships in an *application schema* (4.1.5)

**4.1.5**
**application schema**
*conceptual schema* (4.1.14) for *data* (4.1.16) required by one or more applications

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

**4.1.6**
**association**
<UML> semantic relationship that can occur between typed instances

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

[SOURCE: OMG UML, Superstructure, version 2.4.1, 7.3.3]



 

**4.1.7**
**attribute**
<UML> feature within a classifier that describes a range of values that instances of the classifier may hold

[SOURCE: ISO 19103:—[1)], 4.5]

**4.1.8**
**cardinality**
<UML> number of elements in a set

Note 1 to entry: Contrast with *multiplicity* (4.1.24), which is the range of possible cardinalities a set may hold.

[SOURCE: ISO 19103:—[1)], 4.6]

**4.1.9**
**class**
<OWL> set of *individuals* (4.1.20)

[SOURCE: OWL]

**4.1.10**
**class**
<UML> description of a set of objects that share the same *attributes* (4.1.7), *operations* (4.1.30), methods, relationships, and semantics

[SOURCE: ISO 19103:—[1)], 4.7]

**4.1.11**
**codelist**
value domain including a code for each permissible value

[SOURCE: ISO 19136:2007, 4.1.7]

**4.1.12**
**composition**
<UML> *aggregation* (4.1.1) where the composite object (whole) has responsibility for the existence and storage of the composed objects (parts)

[SOURCE: ISO 19103:—[1)], 4.10]

**4.1.13**
**conceptual model**
model that defines concepts of a *universe of discourse* (4.1.36)

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

**4.1.14**
**conceptual schema**
formal description of a *conceptual model* (4.1.13)

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

**4.1.15**
**constraint**
<UML> condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element

[SOURCE: ISO 19103:—[1)], 4.13]

**4.1.16**
**data**
reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing

[SOURCE: ISO/IEC 2382:2015, 2121272]

**4.1.17**
**data property**
<OWL> semantic *association* (4.1.6) between an *individual* (4.1.20) and a typed *literal* (4.1.21)

Note 1 to entry: Data properties were sometimes referred to as 'concrete properties' in Description Logic.

[SOURCE: OWL]

**4.1.18**
**datatype**
<OWL> entities that refer to a set of concrete *data* (4.1.16) values

EXAMPLE xsd:string, xsd:integer, xsd:decimal

Note 1 to entry: Datatypes are distinct from *classes* (4.1.9) of *individuals* (4.1.20), the latter are denoted by URIs and may be used by reference.

[SOURCE: OWL]

**4.1.19**
**generalization**
<UML> taxonomic relationship between a more general element and a more specific element of the same element type

Note 1 to entry: An instance of the more specific element can be used where the more general element is allowed.

[SOURCE: ISO 19103:—[1)], 4.18]

**4.1.20**
**individual**
instance of a *class* (4.1.9)

Note 1 to entry: It refers to a resource belonging to the extension of the class.

[SOURCE: Adapted from the OWL Web Ontology Language Guide]

**4.1.21**
**literal value**
**literal**
constant, explicitly specified value

EXAMPLE "1"^^xsd:integer, "abc"^^xsd:string

Note 1 to entry: This contrasts with a value that is determined by resolving a chain of substitution (e.g. a variable).

[SOURCE: ISO 19143:2010, 4.15]

**4.1.22**
**localName**
reference to a local object directly accessible from a *namespace* (4.1.25)

[SOURCE: ISO 19103:—[1)], modified – Derived from 7.5.5.1]

**4.1.23**
**metadata**
information about a resource

[SOURCE: ISO 19115-1:2014, 4.10]

**4.1.24**
**multiplicity**
<UML> specification of the range of allowable cardinalities that a set may assume

[SOURCE: ISO 19103:—[1)], 4.24]

**4.1.25**
**namespace**
<general> domain in which names, expressed by character strings, can be mapped to objects

Note 1 to entry: The names can be subject to local *constraints* (4.1.15) enforced by the namespace.

[SOURCE: ISO 19103:—[1)], modified – Derived from 7.5.2.1]

**4.1.26**
**namespace**
<RDF> common URI prefix or stem used in identifiers for a set of related resources

Note 1 to entry: The RDF namespace is concatenated with the *localName* (4.1.22) to create the complete URI identifier for an RDF resource. Every RDF resource is identified by a URI. In contrast, an XML namespace URI scopes a local XML component name, but there is no rule for combining these into a single identifier string.

**4.1.27**
**namespace**
<XML> collection of names, identified by a URI reference, that are used in XML documents as element names and *attribute* (4.1.7) names

[SOURCE: ISO/TS 19139:2007, 4.1]

**4.1.28**
**object property**
<OWL> semantic *association* (4.1.6) between a pair of *individuals* (4.1.20)

Note 1 to entry: Object properties have sometimes been referred to as 'abstract properties' in Description Logic.

[SOURCE: OWL]

**4.1.29**
**ontology**
formal representation of phenomena of a *universe of discourse* (4.1.36) with an underlying vocabulary including definitions and axioms that make the intended meaning explicit and describe phenomena and their interrelationships

[SOURCE: ISO 19101-1:2014, 4.1.26]

**4.1.30**
**operation**
<UML> behavioural <UML> feature of a classifier that specifies the name, type, parameters, and *constraints* (4.1.15) for invoking an associated behaviour

[SOURCE: ISO 19103:—[1)], 4.26]

**4.1.31**
**package**
<UML> general purpose mechanism for organizing elements into groups

Note 1 to entry: A package provides a *namespace* (4.1.27) for the grouped elements.

[SOURCE: ISO 19103:—[1)], 4.27]

**4.1.32**
**property restriction**
<OWL> special kind of *class* (4.1.9) description through the definition of *constraints* (4.1.15) on values and cardinalities

[SOURCE: OWL]

**4.1.33**
**qualified cardinality**
<OWL> *cardinality* (4.1.8) restriction that applies to *literals* (4.1.21) or *individuals* (4.1.20) that are connected by a *data property* (4.1.17) or an *object property* (4.1.28) and are instance of the qualifying range [*datatype* (4.1.18) or *class* (4.1.9)]

[SOURCE: OWL]

**4.1.34**
**source document**
document that contains the original definition of a resource

**4.1.35**
**stereotype**
<UML> extension of an existing metaclass that enables the use of platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass

[SOURCE: ISO 19103:—[1)], 4.33]

**4.1.36**
**universe of discourse**
view of the real or hypothetical world that includes everything of interest

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

**4.1.37**
**unqualified cardinality**
<OWL> *cardinality* (4.1.7) restriction that applies to all *literals* (4.1.21) or *individuals* (4.1.20) that are connected by a *data property* (4.1.17) or an *object property* (4.1.28)

[SOURCE: OWL]

## 4.2 Abbreviations

DL Description Logic

IRI Internationalized Resource Identifier

MOF MetaObject Facility

OMG Object Management Group

OWL Web Ontology Language (version 2)

RDF Resource Description Framework

RDFS RDF Schema

SKOS Simplified Knowledge Organization System

UML Unified Modeling Language



 

URI Universal Resource Identifier

XML eXtensible Markup Language

XSD XML Schema Definition

### 4.3 Namespaces

| | |
|---|---|
| dc | Dublin Core http://purl.org/dc/elements/1.1/[5] |
| dct | Dublin Core http://purl.org/dc/terms/ |
| owl | Web Ontology Language http://www.w3.org/2002/07/owl# |
| rdf | Resource Description Framework http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | RDF Schema http://www.w3.org/2000/01/rdf-schema# |
| skos | Simple Knowledge Organization System http://www.w3.org/2004/02/skos/core# |
| 19150-2owl | Requirements class for conversion of a UML package from the ISO/TC 211 Harmonized Model to OWL http://standards.iso.org/iso/19150-2/req/owl |
| 19150-2owl-conf | Conformance class for conversion of a UML package from the ISO/TC 211 Harmonized Model to OWL http://standards.iso.org/iso/19150-2/conf/owl |
| 19150-2app | Requirements class for formalization of an application schema in OWL http://standards.iso.org/iso/19150-2/req/applicationSchema |
| 19150-2app-conf | Conformance class for formalization of an application schema in OWL http://standards.iso.org/iso/19150-2/conf/applicationSchema |
| iso19150-2 | Base ontology elements required for formalization of UML models and application schemas in OWL http://def.isotc211.org/iso19150-2/2012/base# |
| exPk | Generic prefix for examples for UML and conceptual schema language rules http://def.isotc211.org/example/aPackage# |
| exPkCode | Generic prefix for examples of code-lists for UML and conceptual schema language rules http://def.isotc211.org/example/aPackage/code/ |
| myapp | Dummy prefix for examples, representing an application schema namespace |
| xsd | XML Schema Definition http://www.w3.org/2001/XMLSchema# |
| gfm | ontology describing the General Feature Model (ISO 19109:—[2]) |

## 5 Namespace

A namespace is a collection of names identified by a URI reference.[13] The definition of namespaces shall follow the rules for URI definition as documented in Annex B. In RDF applications (including OWL) every resource, including definitions and datatypes as well as individuals, is identified by an IRI.

NOTE The prefix that references a namespace within an ontology in OWL is identified in the namespace declaration as part of the ontology header element (see 6.3).

# 6 Rules for mapping ISO geographic information UML models to OWL ontologies

## 6.1 General

As introduced in ISO/TS 19150-1:2012, Clause 6 defines the conversion of UML static view modeling elements that are used in the ISO geographic information standards into OWL. These elements are name, package, class, stereotype, enumeration, code list, attribute, multiplicity, generalization/inheritance, association, aggregation, composition, and constraints.

ISO 19103:—[1)] defines the profile of UML used in conceptual modelling in ISO geographic information standards. Among others, three important aspects of the UML profile are:

a) every navigable association-end must have a role-name (ISO 19103:—[1)], 6.8.2);

b) class stereotypes «CodeList» and «Union» indicate a special behaviour different to normal classes (ISO 19103:—[1)], 6.10);

c) a set of primitive datatypes are provided.

Item a) means that the UML models under consideration map easily to the RDF model where all properties have semantic names. This allows for a more streamlined mapping from UML to OWL than, for example, the generic approach taken by OMG, which supports all the options implied by the MOF and UML meta model.

Item b) means that different UML-OWL transformation rules are required for classes with these stereotypes compared with standard classes. There are also standard stereotypes provided by UML, such as «enumeration».

Item c) requires that mappings from the UML classes representing datatypes to specific constructs using RDF, RDFS, OWL and the XSD datatypes accessible to OWL be defined.

The conversion rules are limited to OWL 2 RL, meaning that OWL RL shall be used for the definitions of the rules (see W3C OWL 2 and W3C OWL 2 RFD).

NOTE OWL2 RL profile corresponds approximately with OWL version 1 DL profile. This profile ensures a level of computability which is generally considered desirable for rigorous ontologies.

Clause 6 uses Augmented Backus Naur Form notation (see IETF RFC 5234), which is summarized in Annex C.

This part of ISO 19150 requires the use of standard HTTP URIs to identify resources in geographic information for the purpose of ontologies. The URI structures are defined in Annex B.

This part of ISO 19150 requires an ontology that defines additional annotation properties, properties and classes to support the representation of ISO geographic information UML models into OWL ontologies. This ontology and its namespace are documented in Annex D.

The requirements for representing a UML package from a standard in the series of ISO geographic information standards in OWL comprise a single requirements class (Table 1), identified as http://standards.iso.org/iso/19150-2/req/2owl and abbreviated as 19150-2owl.

**Table 1 — Requirements class for representing a UML package from a standard in the series of ISO geographic information standards in OWL**

| **Requirements class** | |
|---|---|
| 19150-2package = http://standards.iso.org/iso/19150-2/req/package | |
| **Target type** | Ontology |
| **Dependency** | http://www.w3.org/TR/owl2-syntax/ (OWL) |
| **Dependency** | http://tools.ietf.org/html/rfc3986 (URI Syntax) |
| **Dependency** | http://standards.iso.org/iso/19103/ed-2/en/ (Conceptual schema language) |



 

**Table 1** *(continued)*

| **Requirements class** | |
|---|---|
| 19150-2package = http://standards.iso.org/iso/19150-2/req/package | |
| **Requirement** | 19150-2package:name |
| **Requirement** | 19150-2package:ontologyName |
| **Requirement** | 19150-2package:rdfNamespace |
| **Requirement** | 19150-2package:className |
| **Requirement** | 19150-2package:datatypeName |
| **Requirement** | 19150-2package:propertyName |
| **Requirement** | 19150-2package:codeName |
| **Requirement** | 19150-2package:package |
| **Requirement** | 19150-2package:class |
| **Requirement** | 19150-2package:abstractClass |
| **Requirement** | 19150-2package:attribute-dataProperty |
| **Requirement** | 19150-2package:attribute-objectProperty |
| **Requirement** | 19150-2package:enumeration |
| **Requirement** | 19150-2package:codelist |
| **Requirement** | 19150-2package:codelistextension |
| **Requirement** | 19150-2package:union |
| **Requirement** | 19150-2package:multiplicity |
| **Requirement** | 19150-2package:relationship-generalization |
| **Requirement** | 19150-2package:relationship-association |
| **Requirement** | 19150-2package:relationship-aggregation |
| **Requirement** | 19150-2package:constraint |

## 6.2 Name

### 6.2.1 Scoping and namespaces

The first set of requirements deals with the construction of URIs used to identify ontology namespaces, classes, datatypes and properties, and is summarized in Annex B.

UML sets a number of restrictions and conventions on element names. Key restrictions are that (a) each class shall have a unique name within the context of a package, and (b) each attribute and role has a unique name within the context of a class. Hence, a UML package provides the namespace for its classes, and a UML class provides the namespace for its attributes and association roles. Therefore, an attribute name *attributeName* that is used in a *ClassA* can also be used in a *ClassB* both having a specific semantics in the context of each class (Figure 1). The class is therefore a namespace for its attributes.

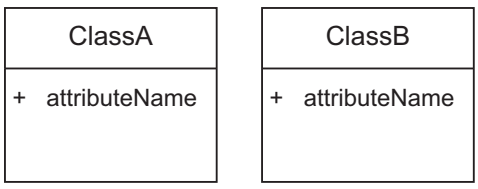


**Figure 1 — UML class and attribute names**

Naming rules in OWL inherit from RDF and are different to those in UML. Each resource (Class, DataType, Property) is denoted by a URI. A set of related resources are usually denoted by URIs with the same stem or base URI, known as the RDF Namespace. Within the context of an RDF namespace the localName for each class, datatype, and property must be unique and distinct from each other.

NOTE 1 OWL Ontologies are identified with a URI which is usually closely related to the namespace URI for the elements of the ontology. Under such a naming convention the resources are scoped to the ontology.

NOTE 2 The URIs used in an RDF document can be abbreviated using the QName syntax [XML Namespace], where the prefix stands for the URI of an RDF namespace.

NOTE 3 In OWL 2 the same URI can denote both a class and an individual. This is called 'punning'. However, use of punning limits reasoning performance and behaviour, so is generally avoided if possible. The profile of OWL used in this part of ISO 19150 does not allow punning.

The key resources that need naming within an OWL representation of a model are packages, classes, attributes and association roles.

Table 2 sets the general requirements for name.

**Table 2 — General name requirement**

| **Requirement** |
|---|
| 19150-2package:name |
| URIs for all resources in geographic information shall the these representation restrictions:<br>— no space characters;<br>— dash and underscore characters allowed;<br>— other punctuation characters from the source are replaced by underscore characters. |

### 6.2.2 Ontology name

The ontology name is constructed by appending an abbreviation of the package name to a base URI corresponding to the document or standard, with the "/" used as the separator. Table 3 sets the requirement for ontology name.

**Table 3 — Ontology name requirement**

| **Requirement** |
|---|
| 19150-2package:ontologyName |
| The ontology name URI shall be based on the name of the corresponding UML package, according to the expression:<br>`ontologyName = URIbase "/" umlPackageName`<br>— URIbase is a base URI in a domain owned by the organization that maintains the model or ontology (see Annex B),<br>— umlPackageName is an abbreviation for the name of the UML package corresponding to the ontology, which also follows the syntax rules for a segment as described in RFC 3986, 3.3.<br>umlPackageName shall follow these representation restrictions:<br>— lower case;<br>— only the semantic part of the package name is represented (see Annex B for examples). |

### 6.2.3 RDF namespace for ontology

The ontology namespace is constructed by appending a "#" character at the end of the ontology name. Table 4 sets the requirement for RDF namespace for ontology.



 

**Table 4 — RDF namespace for ontology**

| Requirement |
|---|
| 19150-2package:rdfNamespace |
| The RDF namespace for ontology items shall be based on the ontology name of the corresponding UML package, according to the expression:<br>`rdfNamespace = ontologyName "#"` |

NOTE The generic URI syntax uses the "#" style of URI to identify resources that are secondary to a primary resource (RFC 3986 §3.5). This is the most common pattern used to identify components of ontologies that describe classes and properties. As HTTP does not transmit the fragment after the "#" to the server, a request for a resource identified in this way results in the primary resource being retrieved, i.e. in this case a complete ontology.

### 6.2.4 Class name

Each class name is constructed by appending the UML class name to the RDF namespace. Table 5 sets the requirement for class name in ontology.

**Table 5 — Class name**

| Requirement |
|---|
| 19150-2package:className |
| The class URI shall be based on the name of the corresponding UML class, according to the expression:<br>`className = rdfNamespace umlClassName`<br>— umlClassName is the name of the corresponding UML Class.<br>umlClassName shall follow these representation restrictions:<br>— upper camel case. |

### 6.2.5 Datatype name

Datatype name follows a similar pattern to class name. The local name for a datatype is appended to the RDF namespace. Table 6 sets the requirement for datatype name in ontology.

**Table 6 — Datatype name**

| Requirement |
|---|
| 19150-2package:datatypeName |
| The datatype URI shall be based on the name of the corresponding UML class, according to the expression:<br>`datatypeName = rdfNamespace datatypeLocalName`<br>— datatypeLocalName is the local name of the datatype.<br>datatypeLocalName shall follow these representation restrictions:<br>— upper camel case. |

### 6.2.6 Property name

Attribute names are not required to be unique within a package in UML. However, it is uncommon to have multiple attributes with the same name with different semantics within a package. This part of ISO 19150 identifies a property by its localName scoped to the package where it is unique within the package, or scoped to its class where the same property name appears more than once in a package with different meanings. Table 7 sets the requirement for property name in ontology.

**Table 7 — Property name**

| Requirement |
|---|
| 19150-2package:propertyName |
| The property URI shall be based on the name of the corresponding UML attribute or association role name according to the expression:<br>`propertyName = rdfNamespace [umlClassName "."] propertyLocalName`<br>`propertyLocalName = umlAttributeName / umlRoleName`<br>— umlAttributeName means the name of a UML attribute,<br>— umlRoleName means the name of a UML association role.<br>umlAttributeName and umlRoleName shall follow these representation restrictions:<br>— lower camel case.<br>NOTE In addtition to its propertyLocalName, a propertyName can optionally include its umlClassName when the property is scoped to the class or if the same propertyLocalName may be used by another property of another class with a different semantics. |

### 6.2.7 Names for codelists and their members

Codelist classes and their members require specific treatment. As described in 6.8.2, a codelist class shall be implemented in OWL/RDF as both an OWL Class and as a SKOS ConceptScheme, and its members as SKOS Concepts.

SKOS (Simplified Knowledge Organization System) is a W3C standard that has been broadly adopted for vocabulary formalization. SKOS is formulated as an OWL ontology, and instances can be integrated with an OWL ontology. SKOS supports the codelist requirements of membership and extensibility. SKOS also supports the recording of basic thesaurus-like semantic relationships between members (broader, narrower, etc).

SKOS ConceptScheme and SKOS Concept resources are each individuals, as they have a specific *rdf:type* property that relates them to the class to which they belong. In the OWL context, the distinction between individuals and other ontology elements is important, and is often reflected in the use of different styles of URI to denote these different kinds of resources.

In particular, while classes and properties are normally used primarily in the context of the other classes and properties in an ontology, individuals are commonly accessed individually, Hence, while '#' URIs are commonly used to denote classes and properties in an ontology, it is common to denote individuals using '/' URIs. Table 8 sets the requirement for code names in ontology.

**Table 8 — Code name**

| Requirement |
|---|
| 19150-2package:codeName |
| The URI for a codelist shall be based on the name of the corresponding UML class stereotyped «CodeList» according to the expression:<br>`codeNamespace = ontologyName "/code/"`<br>`conceptSchemeName = codeNamespace className`<br>`collectionName = codeNamespace className "Collection"`<br>— className means the name of the class stereotyped «CodeList».<br>className shall follow these representation restrictions:<br>— upper camel case.<br>The URI for a codelist value shall be based on the name of the corresponding UML class stereotyped «CodeList» and the name of the codelist value (presented in UML similarly as a class attribute) according to the expression:<br>`conceptName = conceptSchemeName "/" umlCodelistValueName`<br>— umlCodelistValueName means the name of a codelist value.<br>umlCodelistValueName shall follow these representation restrictions:<br>— lower camel case. |

## 6.3 Package

### 6.3.1 UML notation

PACKAGE is the UML mechanism to group elements and to provide a namespace for the members of the group. A package may contain other packages. The UML notation of a PACKAGE is depicted as in Figure 2.

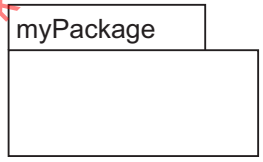


**Figure 2 — UML PACKAGE notation**

PACKAGEs may be stereotyped to extend their semantics.

### 6.3.2 OWL notation

In OWL, 'ontology' is the resource that provides for the grouping of elements. It may be uniquely identified by an IRI (see Annex B). It may also provide additional information with ontology annotation such as the version, the name or comments. It may also import other ontologies.

### 6.3.3 Rules

The UML model of each ISO geographic information standard corresponds to one or more Ontology <OWL> using an *owl:Ontology* declaration. An *rdfs:label* annotation property declaration provides the full name of the corresponding UML PACKAGE.

A *dct:source* annotation property declaration provides the title of the Ontology <OWL> reference document or standard and, when applicable, the hierarchy of PACKAGE names.

An *owl:versionInfo* declaration provides the date corresponding to the reference document or standard. If no document or standard corresponds to the Ontology <OWL>, the *owl:versionInfo* declaration provides the date of the official release of the Ontology <OWL>. The date is given according to ISO 8601.[6]

In the case where a UML PACKAGE depends on one or more other packages, each dependency package shall correspond to a specific Ontology <OWL>. The *owl:imports* declaration is used to set the dependency between Ontologies <OWL>.

Table 9 sets the requirement for the description of PACKAGEs in Ontology <OWL>.

**Table 9 — Package**

| **Requirement** |
|---|
| 19150-2package:package |
| The UML model of each ISO geographic information standard shall correspond to one or more ontologies.<br>An Ontology <OWL> shall import the ontology denoted http://def.isotc211.org/iso19150/-2/2012/base to include the resources supporting this International Standard.<br>An Ontology <OWL> shall be annotated with:<br>— the full name of the corresponding UML PACKAGE, using *rdfs:label*,<br>— the source for the ontology, using *dct:source* for a citation for the standard containing the definition, according to the expression:<br>`title = [documentNumber "," ] documentTitle [*[" : " packageName]]`<br>— documentNumber means the number of the document;<br>— documentTitle means the title of the document;<br>— packageName means the name of the package;<br>— the version date of the reference document or the ontology, using *owl:versionInfo*. |
| NOTE Integration of both the resources of this part of ISO 19150, and other dependencies between ontologies, use the *owl:imports* mechanism. |

Additional annotation for the Ontology <OWL> may be provided such as:

— human-readable description, using an *rdfs:comment* declaration,

— version IRI, using an *owl:versionIRI* declaration to specify the prior version of the Ontology <OWL>,

— information on prior version, using an *owl:priorVersion* declaration,

— backward compatibility, using an *owl:backwardCompatibleWith* declaration to specify the previous version of the Ontology <OWL> that is compatible with the current version of the containing Ontology <OWL> and using an *owl:incompatibleWith* declaration to specify the prior version of the Ontology <OWL> that is incompatible with the current version of the containing Ontology <OWL>,

— deprecation, using an *owl:deprecated* declaration with the value equal to *"true"^^xsd:boolean* if the Ontology <OWL> is deprecated, and

— superseding ontology, using a *dct:replaces* declaration to identify the Ontology <OWL> it supersedes and using a *dct:isReplacedBy* declaration to identify the Ontology <OWL> that has superseded it.

The following example illustrates the specification of an Ontology <OWL>.

NOTE This example and all the following examples are not complete definitions. The prefix descriptions provided in 4.3 can help reading the examples.

EXAMPLE

RDF/Turtle serialization

</iso19107/2003/SpatialSchema> a owl:Ontology ;



 

rdfs:label “ISO 19107:2003 Spatial Schema” ;

dct:source “ISO 19107:2003, Geographic information – Spatial schema” ;

owl:versionInfo “2003-05-01” ;

owl:imports <http://www.isotc211.org/tc211ontologies/iso19107/2003/7/Geometry> .

RDF/XML serialization

```
<rdf:RDF xmlns="http://def.isotc211.org">
   <owl:Ontology rdf:about="/iso19107/2003/SpatialSchema">
      <rdfs:label>ISO 19107:2003 Spatial Schema</rdfs:label>
      <dct:source>ISO 19107:2003, Geographic information – Spatial schema</dct:source >
      <owl:versionInfo>2003-05-01</owl:versionInfo>
      <owl:imports rdf:resource="http://www.isotc211.org/tc211ontologies/iso19107/2003/7/Geometry"/>
   </owl:Ontology>
</rdf:RDF>
```

## 6.4 Class

### 6.4.1 UML notation

CLASS is the UML classifier that describes a set of objects sharing the same specifications of features, constraints, and semantics. CLASSes may be abstract or concrete. The CLASS classifier may contain attributes and operations. The UML notation of a CLASS is depicted as in Figure 3.

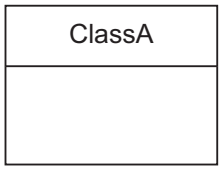


**Figure 3 — UML CLASS notation**

### 6.4.2 OWL notation

Class <OWL> is the resource corresponding to a set of individuals.

### 6.4.3 Rules

A UML CLASS corresponds to a Class <OWL> using an *owl:Class* declaration.

A Class <OWL> is annotated with a label, its definition, and the source of its definition. The label provides the name of the Class <OWL> as used in the UML CLASS and uses a *rdfs:label* declaration. The definition provides the semantic of the Class <OWL> and uses a *skos:definition* declaration. The source of the definition identifies the resource defining this Class <OWL>. It uses a *rdfs:isDefinedBy* declaration to specify the IRI of the source document.

Table 10 sets the requirement for the description of CLASSes in Ontology <OWL>.

**Table 10 — Class**

| Requirement |
|---|
| 19150-2package:class |
| A UML CLASS shall correspond to a Class <OWL> using an *owl:Class* declaration.<br>A Class <OWL> shall be annotated with:<br>— a label, using *rdfs:label*,<br>— the source of the definition, using *rdfs:isDefinedBy* for the IRI of the source document. |

Additional annotation for the Class <OWL> may be provided such as its definition, using a *skos:definition* declaration, and if the Class <OWL> is deprecated, using an *owl:deprecated* annotation property declaration with its value set to *"true"^^xsd:boolean*.

The following example illustrates the specification of a Class <OWL>.

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "ClassA definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA">
    <rdfs:label>ClassA</rdfs:label>
    <skos:definition>ClassA definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
</owl:Class>
```

## 6.5 Abstract class

### 6.5.1 UML notation

In UML, a CLASS can be specified as ABSTRACT. This means that such a CLASS cannot be instantiated and is usually subtyped by implementable or concrete CLASSes. Although they are not instantiable, ABSTRACT CLASSes are useful for classification purposes and, as such, provide important meaning. The UML notation of an ABSTRACT CLASS is depicted as in Figure 4 with its name shown in italic characters.

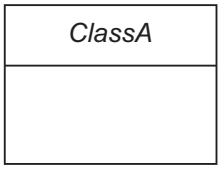


**Figure 4 — UML ABSTRACT CLASS notation**

### 6.5.2 OWL notation

OWL does not have a built-in mechanism to identify a Class <OWL> as abstract.

### 6.5.3 Rules

A UML ABSTRACT CLASS shall correspond to a Class <OWL>. The Class <OWL> is annotated to identify it as abstract.

To support identification of Class <OWL> as abstract, this part of ISO 19150 defines an annotation property *iso19150-2:isAbstract* formalized in the *base* ontology (Annex D):

**OWL Definition 1**

RDF/Turtle serialization

```
iso19150-2:isAbstract a owl:AnnotationProperty ;
rdfs:domain owl:Class;
rdfs:range xsd:Boolean .
```

RDF/XML serialization

```
<owl:AnnotationProperty rdf:about="&iso19150-2;isAbstract">
   <rdfs:range rdf:resource="&xsd;boolean"/>
   <rdfs:domain rdf:resource="&owl;Class"/>
</owl:AnnotationProperty>
```

This annotation property provides the mechanism for documenting if a Class <OWL> is abstract. *iso19150-2:isAbstract* is set to *"true"^^xsd:boolean* when a Class <OWL> is abstract. The default value is *false* when *iso19150-2:isAbstract* is not specified.

Table 11 sets the requirement for the description of ABSTRACT CLASSes in Ontology <OWL>.

**Table 11 — Abstract class**

| **Requirement** |
|---|
| 19150-2package:abstractClass |
| A UML ABSTRACT CLASS shall correspond to a Class <OWL>. The Class <OWL> shall be annotated to identify it as abstract using an *iso19150-2:isAbstract* annotation property declaration. |

The following example illustrates the identification of an abstract 'class.'

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
   rdfs:label "ClassA" ;
   skos:definition "ClassA definition" ;
   rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
```

iso19150-2:isAbstract “true”^^xsd:boolean .

RDF/XML serialization

```
<owl:Class rdf:about=”&exPk;ClassA”>
  <rdfs:label>ClassA</rdfs:label>
  <skos:definition>ClassA definition</skos:definition>
  <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
  <iso19150-2:isAbstract rdf:datatype=”&xsd;boolean”>true</iso19150-2:isAbstract>
</owl:Class>
```

## 6.6 Attribute

### 6.6.1 UML Notation

In UML, an ATTRIBUTE is a property related to a classifier (i.e., a class or a datatype) by the way of ownedAttribute. It associates an instance of a class to a value or set of values of the type specified in the attribute declaration. The general notation for an attribute is:

```
property = [visibility] [“/”] name [“:”prop-type] [“[“ multiplicity “]”]
[“=” default] [“{“ prop-modifier *[“,” prop-modifier] “}”]
```

- `visibility = “+” / “-” / “#” / “~”`
  - “+” means public;
  - “-” means private;
  - “#” means protected;
  - “~” means package;
- “/” signifies that the property is derived;
- name is the name of the property;
- prop-type is the name of the type of the property;
- multiplicity is the multiplicity of the property. If this term is omitted, it implies a multiplicity of 1 (exactly one);
- default is an expression that evaluates to the default value or values of the property;
- `prop-modifier = “readOnly” / “union” / “subsets” property-name / “redefines” property-name / “ordered” / “unique” / “nonunique” / prop-constraint`
  - “readOnly” means that the property is read only;
  - “union” means that the property is a derived union of its subsets;
  - “subsets” property-name means that the property is a proper subset of the property identified by <property-name>;
  - “redefines” property-name means that the property redefines an inherited property identified by <property-name>;

— "ordered" means that the property is ordered;

— "unique" means that there are no duplicates in a multi-valued property;

— prop-constraint is an expression that specifies a constraint that applies to the property. It is usually shown in the compartment below the class name compartment of the class notation.

The UML notation of an ATTRIBUTE is depicted as in Figure 5.

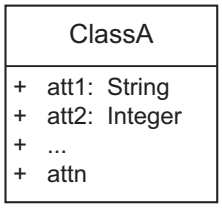


**Figure 5 — UML ATTRIBUTE notation**

### 6.6.2 OWL notation

OWL defines two kinds of resources for ATTRIBUTEs: Data Property <OWL> and Object Property <OWL>.

A Data Property <OWL> connects a (literal) data value to the property of the Class <OWL>. It is defined with an *owl:DatatypeProperty* declaration. A Data Property <OWL> is connected to a Class <OWL> by specifying its domain with an *rdfs:domain* declaration. The target type of a Data Property <OWL> is specified by its range with an *rdfs:range* declaration. Table 12 provides a mapping of the ISO 19103:—[1)] datatypes to OWL datatypes.

**Table 12 — Datatype mapping**

| | ISO 19103 **types** | **OWL datatypes** | **gcoDataypes (GCOLiteral[a])** |
|---|---|---|---|
| Prim-itive types | CharacterString | xsd:string | gco:CharacterString |
| | Date | xsd:date (in conjuction with rdf:XMLLiteral) | gco:Date |
| | DateTime | xsd:dateTime | gco:DateTime |
| | Integer | xsd:integer | gco:Integer |
| | Decimal | xsd:decimal | gco:Decimal |
| | Real | xsd:double | gco:Real |
| | Boolean | xsd:boolean | gco:Boolean |
| Name types | GenericName | <URI> xsd:Name | *gco:AbstractGenericName* |
| | LocalName | xsd:NCName | gco:LocalName |
| | ScopedName | <URI> | gco:ScopedName |
| | TypeName | xsd:NCName | gco:TypeName |
| | MemberName | xsd:Name | gco:MemberName |

[a] Allowed XML representations in conjunction with the use of *iso19150-2:GCOLiteral*

Additionally, this part of ISO 19150 defines the *iso19150-2:GCOLiteral* datatype, which allows the representation of ISO 19103:—[1)] datatypes in RDF based on the gco namespace as defined in ISO/TS 19139:2007[9] (see Table 12).

**OWL Definition 2**

RDF/Turtle serialization

```
iso19150-2:GCOLiteral a rdfs:Datatype ;
    rdfs:label “GCOLiteral” ;
    rdfs:isDefinedBy <http://standards.iso.org/iso/ts/19139/ed-1/en/> ;
    owl:equivalentClass rdf:XMLLiteral .
```

RDF/XML serialization

```
<rdfs:Datatype rdf:about=”&iso19150-2;GCOLiteral”>
    <rdfs:label>GCOLiteral</rdfs:label>
    <rdfs:isDefinedBy>http://standards.iso.org/iso/ts/19139/ed-1/en/</rdfs:isDefinedBy>
    <owl:equivalentClass rdf:resource=”&rdf;XMLLiteral”/>
</rdfs:Datatype>
```

The cardinality (i.e. the number of occurrences) for data properties can be from zero to multiple. How to set cardinality is specified in 6.10.

An Object Property <OWL> connects an instance or an individual of a Class <OWL> to the property of the Class <OWL>. It is defined with an *owl:ObjectProperty* declaration. An Object Property <OWL> is connected to a Class <OWL> by specifying its domain with an rdfs:domain declaration. The target type of an Object Property <OWL> is specified by its range with an rdfs:range declaration. The possible value types for the rdfs:range declaration are any valid Class <OWL>, The arity for Object Property <OWL> can be from zero to multiple. How to set arities is defined in 6.9.

### 6.6.3 Rules

#### 6.6.3.1 OWL data property rules

A UML ATTRIBUTE described by a data value (e.g. number, character string, Boolean value, etc.) corresponds to a Data Property <OWL>. It is connected to the Class <OWL> it is associated with using an *rdfs:domain* declaration. The type of data values is specified using an *rdfs:range* declaration.

A Data Property <OWL> is annotated with a label and its source document. The label provides the name of the UML ATTRIBUTE and uses a *rdfs:label* declaration. The source document identifies the resource defining this UML ATTRIBUTE. It uses *rdfs:isDefinedBy* to declare the IRI of the source document.

Table 13 sets the requirement for the description of ATTRIBUTEs through Data Property <OWL> in Ontology <OWL>.

**Table 13 — Attribute – OWL data property**

| Requirement |
|---|
| 19150-2package:attribute-dataProperty |
| A UML ATTRIBUTE described by a data value shall correspond to a Data Property <OWL> and use an *owl:DatatypeProperty* declaration. It shall be connected to the Class <OWL> it is associated with using an *rdfs:domain* declaration. The type of data values shall be specified using an *rdfs:range* declaration.<br>A Data Property <OWL> shall be annotated with:<br>— a label for the name of the UML ATTRIBUTE, using *rdfs:label*,<br>— a source document defining this UML ATTRIBUTE, using *rdfs:isDefinedBy* to declare the IRI of the resource. |

Additional annotation for the Data Property <OWL> may be provided such as:

— definition, using a *skos:definition* declaration to provide the semantics of the UML ATTRIBUTE, and

— deprecation, using an *owl:deprecated* declaration, which is set to *"true"^^xsd:boolean* if the Data Property <OWL> is deprecated.

The following example illustrates the definition of a 'data property' for an attribute definition of a 'class'.

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA.att1 a owl:DatatypeProperty ;
    rdfs:label "att1" ;
    skos:definition "att1 definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:domain exPk:ClassA ;
    rdfs:range xsd:string .
```

RDF/XML serialization

```
<owl:DatatypeProperty rdf:about="&exPk;ClassA.att1">
    <rdfs:label>att1</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:domain rdf:resource="&exPk;ClassA"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

NOTE As illustrated in this example, att1 is restricted to the domain of ClassA. This is consistent with the UML metamodel that bind attributes to classes. This UML to OWL conversion preserves this binding.

### 6.6.3.2 OWL object property rules

A UML ATTRIBUTE described by another UML CLASS corresponds to an Object Property <OWL>. It is connected to the Class <OWL> resources it is associated with using an *rdfs:domain* declaration. The types of the Object Property <OWL> resources are specified using an *rdfs:range* declaration.

A Object Property <OWL> is annotated with a label and its source document. The label provides the name of the UML ATTRIBUTE and uses a *rdfs:label* declaration. The source document identifies the resource defining this UML ATTRIBUTE. It uses *rdfs:isDefinedBy* to declare the IRI of the source document.

Table 14 sets the requirement for the description of ATTRIBUTEs through Object Property <OWL> in Ontology <OWL>.

**Table 14 — Attribute – OWL object property**

| **Requirement** |
|---|
| 19150-2package:attribute-objectProperty |
| A UML ATTRIBUTE described by another UML CLASS shall correspond to a Object Property <OWL> and use an *owl:ObjectProperty* declaration. It shall be connected to the Class <OWL> resources it is associated with using an *rdfs:domain* declaration. The type of the Object Property <OWL> resources shall be specified using an *rdfs:range* declaration.<br>An Object Property <OWL> shall be annotated with:<br>— a label for the name of the UML ATTRIBUTE, using *rdfs:label*, and<br>— a source document defining this UML ATTRIBUTE, using *rdfs:isDefinedBy* to declare the IRI of the resource. |

Additional annotation for the Object Property <OWL> may be provided such as:

— definition, using a *skos:definition* declaration to provide the semantics of the UML ATTRIBUTE, and

— deprecation, using an *owl:deprecated* declaration, which is set to *"true"^^xsd:boolean* if the Object Property <OWL> is deprecated.

The following example illustrates the definition of an 'object property' for an attribute definition of a 'class'.

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA.att1 a owl:ObjectProperty ;
    rdfs:label "att1" ;
    skos:definition "att1 definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:domain exPk:ClassA ;
    rdfs:range exPk:ClassB .
exPk:ClassB a owl:Class ;
    rdfs:label "ClassB" ;
    skos:definition "ClassB definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
```

RDF/XML serialization

```
<owl:ObjectProperty rdf:about="&exPk;ClassA.att1">
    <rdfs:label>att1</rdfs:label>
    <rdfs:isDefinedBy>sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:domain rdf:resource="&exPk;ClassA"/>
```

```
    <rdfs:range rdf:resource="&exPk;ClassB"/>
</owl:ObjectProperty>
<owl:Class rdf:about="&exPk;ClassB">
    <rdfs:label>ClassB</rdfs:label>
    <skos:definition>ClassB definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
</owl:Class>
```

## 6.7 Enumerated type

### 6.7.1 Enumeration

#### 6.7.1.1 UML notation

In UML, an ENUMERATION corresponds to a datatype defining a closed list of valid mnemonic identifiers. The list is part of the model as an enumeration literal. As such, the extension of an enumeration type implies a schema modification for most implementation mappings. An enumeration shall be used only when the membership is logically closed, so it is clear that there are no possible extensions. Attributes of a given enumeration type can take values only from this list. The notation for an ENUMERATION uses the STEREOTYPE «enumeration» and lists members as CLASS ATTRIBUTEs, as shown in Figure 6.

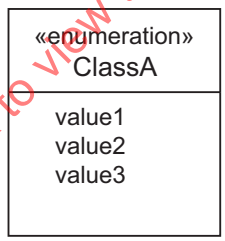


**Figure 6 — UML ENUMERATION notation**

#### 6.7.1.2 OWL notation

OWL supports enumerations of literals directly using a Datatype <RDFS> specified with an *rdfs:Datatype* declaration and an *owl:oneOf* restriction declaration over a collection specification.

#### 6.7.1.3 Rules

A UML ENUMERATION corresponds to a Datatype <RDFS> using an *rdfs:Datatype* declaration. The Datatype <RDFS> specifies the restricted list of literals using an *owl:oneOf* declaration over an *rdf:List* declaration of enumerated values.

The Datatype <RDFS> is annotated with a label and its source document. The label annotation provides the name of the UML ENUMERATION – i.e. the name of the UML ENUMERATION CLASS and uses a *rdfs:label* declaration. The source document identifies the resource defining this enumeration. It uses a *rdfs:isDefinedBy* declaration to specify the IRI of the resource.

Table 15 sets the requirement for the description of ENUMERATIONs in Ontology <OWL>.

**Table 15 — Enumeration**

| Requirement |
|---|
| 19150-2package:enumeration |
| A UML ENUMERATION shall correspond to a Datatype <RDFS> using an *rdfs:Datatype* declaration. The Datatype <RDFS> shall specify the restricted list of literals using an *owl:oneOf* declaration over an *rdf:List* declaration of enumerated values.<br>The Datatype <RDFS> shall be annotated with:<br>— a label for the name of the UML ENUMERATION, using *rdfs:label*, and<br>— a source document defining this UML ENUMERATION, using *rdfs:isDefinedBy* to declare the IRI of the resource. |

Additional annotation for the Datatype <RDFS> may be provided such as a definition annotation, which provides the semantics of the UML ENUMERATION, using a *skos:definition*.

The following example illustrates the definition of an enumeration.

EXAMPLE 1

RDF/Turtle serialization

```
exPk:ClassA a rdfs:Datatype ;
    rdfs:label “ClassA” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    owl:oneOf (“value1” “value2” “value3”) .
```

RDF/XML serialization

```
<rdfs:Datatype rdf:about=”&exPk;ClassA”>
    <rdfs:label>ClassA</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <owl:equivalentClass>
        <rdfs:Datatype>
            <owl:oneOf>
                <rdf:List >
                    <rdf:first rdf:datatype=”&xsd;string”>value1</rdf:first>
                    <rdf:rest>
                        <rdf:List >
                            <rdf:first rdf:datatype=”&xsd;string”>value2</rdf:first>
                            <rdf:rest>
                                <rdf:List>
                                    <rdf:first rdf:datatype=”&xsd;string”>value3</rdf:first>
                                    <rdf:rest rdf:resource=”&rdf;nil”/>
                                </rdf:List>
```

```
                    </rdf:rest>
                  </rdf:List>
                </rdf:rest>
              </rdf:List>
            </owl:oneOf>
          <rdfs:Datatype>
        <owl:equivalentClass>
      </rdfs:Datatype>
```

This part of ISO 19150 connects a Data Property <OWL> of a Class <OWL> to a Datatype <RDFS> of enumerated values with the use of an *rdfs:range* declaration as shown in the following. In this example, a Data Property <OWL> *att1* is defined and associated with ClassB. Its range is specified by the *rdfs:Datatype* ClassA defined in Example 1.

The following example illustrates the definition of an enumeration.

EXAMPLE 2

RDF/Turtle serialization

```
exPk:ClassB.att1 a owl:DatatypeProperty ;
    rdfs:label “att1” ;
    skos:definition “att1 definition” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:domain exPk:ClassB ;
    rdfs:range exPk:ClassA .
```

RDF/XML serialization

```
<owl:DatatypeProperty rdf:about=”&exPk;ClassB.att1”>
    <rdfs:label>att1</rdfs:label>
    <skos:definition>att1 definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:domain rdf:resource=”&exPk;ClassB”/>
    <rdfs:range rdf:resource=”&exPk;ClassA”/>
</owl:DatatypeProperty>
```

### 6.7.2 Code list

#### 6.7.2.1 UML notation

ISO 19103:—[1)] defines the UML extension CODELIST. A CODELIST corresponds to a datatype defining an open list of likely mnemonic identifiers. It expresses a list of potential values. Its UML notation is very

similar to an ENUMERATION. A CODELIST is used only when use of other values apart from those in the initial list is allowed. The UML notation for a CODELIST uses the STEREOTYPE «CodeList» and lists the initial members as CLASS ATTRIBUTEs, as shown in Figure 7.

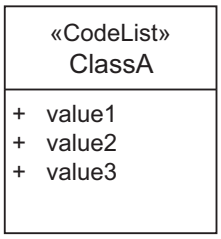


**Figure 7 — UML CODELIST notation**

#### 6.7.2.2 OWL notation

SKOS has been broadly adopted for vocabulary formalization. SKOS supports the codelist requirements of membership and extensibility.

#### 6.7.2.3 Rules

A CODELIST is implemented as a Class <OWL> and as a *skos:ConceptScheme*. The code list Class <OWL> is a subclass of *skos:Concept*. Since a concept scheme resource is an RDF individual, the concept scheme and OWL classes corresponding to the CODELIST are identified by different URIs for compliancy with OWL as used in this part of ISO 19150. The link between the OWL class and the concept scheme representing the CODELIST must therefore be formally asserted with an explicit property.

The members of the CODELIST are implemented as RDF resources that are members of the code list Class <OWL>, i.e. as individuals whose *rdf:type* is the code list Class <OWL>. Since the code list Class <OWL> is a subclass of *skos:Concept*, this entails that each member is itself an individual whose *rdf:type* is *skos:Concept*. The code list members are associated with the concept scheme using the standard *skos:inScheme* property. The concept scheme representing the code list is associated with its members using the standard *skos:hasTopConcept* property.

Table 16 sets the requirement for the description of CODELISTs in Ontology <OWL>.

**Table 16 — Code list**

| **Requirement** |
|---|
| 19150-2package:codelist |
| A CODELIST shall correspond to a Class <OWL>, a ConceptScheme <SKOS>, and a Collection <SKOS>. The Class <OWL> shall be a subclass of *skos:Concept*. The SKOS concept scheme shall be related to the Class <OWL> using a *dct:isFormatOf property*. Each member of the CODELIST shall correspond to an individual whose type is the Class <OWL> corresponding to the CODELIST, and with a *skos:inScheme* property whose value is the ConceptScheme <SKOS> corresponding to the CODELIST. Additionally, each member of the CODELIST shall also be member of the Collection <SKOS> using a *skos:members* declaration. Each of the resources shall be annotated with the following:<br>— a label, using *rdfs:label*,<br>— a source for the definition, using *rdfs:isDefinedBy* for the IRI of the resource. |

The following example illustrates the construction of a CodeList using the OWL and SKOS elements.

EXAMPLE

RDF/Turtle serialization

exPk:ClassA a owl:Class ;

subClassOf skos:Concept ;

```
    rdfs:label “ClassA” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
exPkCode:ClassA a skos:ConceptScheme ;
    skos:prefLabel “ClassA - ConceptScheme” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    dct:isFormatOf exPk:ClassA .
exPkCode:ClassA/value1 a exPk:ClassA , skos:Concept ;
    skos:prefLabel “value1” ;
    skos:inScheme exPkCode:ClassA ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
exPkCode:ClassA/value2 a exPk:ClassA , skos:Concept ;
    skos:prefLabel “value2” ;
    skos:inScheme exPkCode:ClassA ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
exPkCode:ClassA/value3 a exPk:ClassA , skos:Concept ;
    skos:prefLabel “value3” ;
    skos:inScheme exPkCode:ClassA ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> .
exPkCode: ClassACollection a skos:Collection ;
    skos:prefLabel “ClassA - Concepts” ;
    skos:members exPkCode: ClassA/value1 , exPkCode: ClassA/value2 , exPkCode: ClassA/value3 .
```

RDF/XML serialization

```
<owl:Class rdf:about=”&exPk;ClassA”>
    <rdfs:label>ClassA</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:subClassOf rdf:resource=”&skos;Concept”/>
</owl:Class>
<skos:ConceptScheme rdf:about=”&exPkCode;ClassA”>
    <skos:prefLabel>ClassA - ConceptScheme</skos:prefLabel>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <dct:isFormatOf rdf:resource=”&exPk;ClassA”/>
</skos:ConceptScheme>
<exPk:ClassA rdf:about=”&exPkCode;ClassA/value1”>
    <rdf:type rdf:resource=”skos:Concept”/>
```

```
    <skos:prefLabel>value1</skos:prefLabel>
    <skos:inScheme rdf:resource="exPkCode:ClassA"/>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
</exPk:ClassA>
<exPk:ClassA rdf:about="&exPkCode;ClassA/value2">
    <rdf:type rdf:resource="skos:Concept"/>
    <skos:prefLabel>value2</skos:prefLabel>
    <skos:inScheme rdf:resource="exPkCode:ClassA"/>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
</exPk:ClassA>
<exPk:ClassA rdf:about="&exPkCode;ClassA/value3">
    <rdf:type rdf:resource="skos:Concept"/>
    <skos:prefLabel>value3</skos:prefLabel>
    <skos:inScheme rdf:resource="exPkCode:ClassA"/>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
</exPk:ClassA>
<skos:Collection rdf:about="&exPkCode;ClassACollection">
    <skos:prefLabel>ClassA - Concepts</skos:prefLabel>
    <skos:members rdf:parseType="Collection">
        <skos:Concept rdf:resource="&exPkCode;ClassA/value1"/>
        <skos:Concept rdf:resource="&exPkCode;ClassA/value2"/>
        <skos:Concept rdf:resource="&exPkCode;ClassA/value3"/>
    </skos:members>
</owl:Collection>
```



In the example shown above all the members of the code list appeared in the UML class. The corresponding SKOS resources are identified by URIs clearly related to the URI denoting the concept scheme, which is also in the same domain with the same owner as the classes and properties in the ontology. This is appropriate for the code list members defined in the original UML model. However, additional code list members will not necessarily be denoted by a URI in the same domain.

Table 17 sets the requirement for additional CODELISTs items in Ontology <OWL>.

**Table 17 — Code list extension**

| **Requirement** |
|---|
| 19150-2package:codelistextension |
| If a CODELIST is extended with additional items that were not identified in the original model, the URI denoting a new member shall be in a URI domain appropriate to its governance. This should not be the same as the URI domain for the initial members unless the new members are defined by the same authority. |

The SKOS semantic relations are available to record relations amongst items in code lists, where these are known. The mapping relations (broadMatch, closeMatch, exactMatch, narrowMatch, relatedMatch) are used to map to related items in other schemes, while the thesaurus relations (broaderTransitive, broader, narrower, narrowerTransitive, related) are used to record relationships with other items in the same scheme.

Relationships amongst members of a code list may be recorded using *skos:broaderTransitive*, *skos:broader*, *skos:narrower*, *skos:narrowerTransitive*, *skos:related* properties.

Relationships between members of a code list and members of other concept schemes may be recorded using *skos:broadMatch*, *skos:closeMatch*, *skos:exactMatch*, *skos:narrowMatch*, *skos:relatedMatch* properties.

## 6.8 Union class

### 6.8.1 UML notation

In UML, a UNION CLASS is composed of member classes, one of which must be instantiated at run time. A UNION CLASS thus supports selection from an arbitrary set of CLASSes. This may be compared with generalization hierarchies, in which specification of a generalized CLASS in a model supports selection of one of its specializations. The UML notation for a UNION uses the STEREOTYPE «Union» and lists the members as CLASS ATTRIBUTEs, as shown in Figure 8.

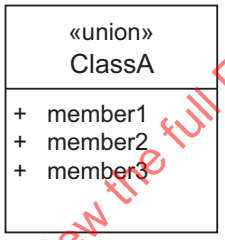


**Figure 8 — UML UNION notation**

### 6.8.2 OWL notation

In OWL, a UNION is implemented using the *owl:unionOf* declaration.

### 6.8.3 Rules

A UML UNION class corresponds to a Class <OWL> using an *owl:unionOf* property whose value is a collection of classes.

Table 18 sets the requirement for UNION CLASS in Ontology <OWL>.

**Table 18 — Union class**

| **Requirement** |
|---|
| 19150-2package:union |
| A UNION class shall be implemented as a Class <OWL> with the members of the union provided as values of an *owl:unionOf* property. |

The following example illustrates the construction of a Union using the OWL elements.

EXAMPLE

RDF/Turtle serialization

owl:equivalentClass [

a owl:Class ;

```
        owl:unionfOf (exPk:ClassA exPk:ClassB)
    ] .
    exPk:ClassA a owl:Class .
    exPk:ClassB a owl:Class .
```

RDF/XML serialization

```
    <owl:equivalentClass>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="&exPk;ClassA"/>
                <owl:Class rdf:about="&exPk;ClassB"/>
        </owl:unionOf>
    </owl:Class>
    </owl:equivalentClass>
```

## 6.9 Multiplicity

### 6.9.1 UML notation

In UML, MULTIPLICITY specifies the allowable cardinalities for the instantiation of an element. It is expressed by the pair of lower and upper bounds of the number of times the element can be instantiated. The lower and upper bounds are non-negative integers. The lower bound shall be equal or greater than "0"; the upper bound shall be greater than "0", shall be equal or greater than the lower bound, and can be infinite. The default values for lower and upper bounds is "1." The notation for multiplicity is as follows:

```
Multiplicity = lower-bound ".." upper-bound
```

— lower-bound means minimum number of times the element can be instantiated;

— upper-bound means maximum number of times the element can be instantiated; the star character (*) can be used to represent an unlimited (or infinite) value.

### 6.9.2 OWL notation

OWL defines cardinalities of elements through property Restrictions <OWL>. It allows the specification of minimum, maximum, and exact cardinalities. A cardinality value is always a non-negative integer. The minimum cardinality specifies the smallest number of individuals that are connected to the class individuals through the property. The maximum cardinality specifies the largest number of individuals that are connected to the class individuals through the property. The exact cardinality specifies the strict number of individuals that are connected to the class individuals through the property. Cardinalities can be qualified or unqualified. Qualified cardinality only applies to the type connected to the property by its range definition whereas unqualified applies to all individuals that are connected to the property.

### 6.9.3 Rules

A UML MULTIPLICITY corresponds to a Restrictions <OWL> on a property (Data Property <OWL> or Object Property <OWL>) using an *owl:Restriction* declaration in combination with cardinality specifications. Cardinality specifications are restricted to the use of cardinalities only with the use of



 

*owl:cardinality*, *owl:minCardinality*, and *owl:maxCardinality* declarations. This rule aligns with the Data Property <OWL> and Object Property <OWL> rules for range specification.

NOTE When no cardinalities are specified, the default values are "0" for the minimum cardinality and "unlimited" for the maximum cardinality.

Table 19 sets the requirement for the description of MULTIPLICITY in Ontology <OWL>.

**Table 19 — Multiplicity**

| **Requirement** |
|---|
| 19150-2package:multiplicity |
| UML MULTIPLICITY shall correspond to Restrictions <OWL> on a property (Data Property <OWL> or Object Property <OWL>) using an *owl:Restriction* declaration in combination cardinality specifications. Cardinality specifications are restricted to the use of cardinalities only using *owl:cardinality*, *owl:minCardinality*, and *owl:maxCardinality* together with *owl:allValuesFrom*. |

Examples 1 to 4 below show how to specify cardinality restriction on Class <OWL> for properties. In Example 1, the minimum cardinality of the property *att1* of *ClassA* is set to "1" and the maximum cardinality is unlimited, since the maximum cardinality is not specified.

NOTE In examples 1 to 5, the definition of the *exPk:ClassA.att1* property follows its definition from either of the examples in 6.6.3.1 and 6.6.3.2.

EXAMPLE 1

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "Class A definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:minCardinality "1"^^xsd:nonNegativeInteger ;
        ] ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:allValuesFrom exPk:ClassB ;
        ] .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA">
```

```
<rdfs:label>ClassA</rdfs:label>
<skos:definition>Class A definition</skos:definition>
<rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
        <owl:allValuesFrom rdf:resource="&exPk;ClassB"/>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

In the following example, the maximum cardinality of the attribute *att1* of *ClassA* is set to "3" and the minimum cardinality is "0", since to minimum cardinality is not specified.

EXAMPLE 2

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "Class A definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:subClassOf
        [
        a owl:Restriction ;
        owl:onProperty exPk:ClassA.att1 ;
        owl:maxCardinality "3"^^xsd:nonNegativeInteger ;
        ] ;
    rdfs:subClassOf
        [
        a owl:Restriction ;
        owl:onProperty exPk:ClassA.att1 ;
```

```
        owl:allValuesFrom exPk:ClassB ;
    ] .
```

RDF/XML serialization

```
<owl:Class rdf:about="& exPk;ClassA">
    <rdfs:label>ClassA</rdfs:label>
    <skos:definition>Class A definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">3</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:allValuesFrom rdf:resource="&exPk;ClassB"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

NOTE The OWL syntax for restrictions is somewhat convoluted. In natural language this can be read as "ClassA is a subclass of the classes about which the only thing we know is that their members have a maximum of three att1 properties".

In the following example, the minimum cardinality of the attribute *att1* of *ClassA* is set to "1" and the maximum cardinality is set to "3".

EXAMPLE 3

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "Class A definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
```

```
        owl:onProperty exPk:ClassA.att1 ;
        owl:minCardinality "1"^^xsd:nonNegativeInteger ;
      ] ;
    rdfs:subClassOf
      [
        a owl:Restriction ;
        owl:onProperty exPk:ClassA.att1 ;
        owl:maxCardinality "3"^^xsd:nonNegativeInteger ;
      ] ;
    rdfs:subClassOf
      [
        a owl:Restriction ;
        owl:onProperty exPk:ClassA.att1 ;
        owl:allValuesFrom exPk:ClassB ;
      ] .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA">
  <rdfs:label>ClassA</rdfs:label>
  <skos:definition>Class A definition</skos:definition>
  <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">3</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:allValuesFrom rdf:resource="&exPk;ClassB"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

In the following two examples, the cardinality of the property *att1* of *ClassA* is set to "1" exactly, i.e. the minimum and the maximum cardinality is "1".

EXAMPLE 4

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "Class A definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:cardinality "1"^^xsd:nonNegativeInteger ;
        ] ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:allValuesFrom exPk:ClassB ;
        ] .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA">
    <rdfs:label>ClassA</rdfs:label>
    <skos:definition>Class A definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
```

```
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:allValuesFrom rdf:resource="&exPk;ClassB"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

EXAMPLE 5

RDF/Turtle serialization

```
exPk:ClassA a owl:Class ;
    rdfs:label "ClassA" ;
    skos:definition "Class A definition" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:minCardinality "1"^^xsd:nonNegativeInteger ;
        ] ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:maxCardinality "1"^^xsd:nonNegativeInteger ;
        ] ;
    rdfs:subClassOf
        [
            a owl:Restriction ;
            owl:onProperty exPk:ClassA.att1 ;
            owl:allValuesFrom exPk:ClassB ;
        ] .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA">
    <rdfs:label>ClassA</rdfs:label>
    <skos:definition>Class A definition</skos:definition>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&exPk;ClassA.att1"/>
            <owl:allValuesFrom rdf:resource="&exPk;ClassB"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```



## 6.10 Relationship

### 6.10.1 Generalization/inheritance

#### 6.10.1.1 UML notation

In UML, a GENERALIZATION is a classification relationship that exists from a more general to a more specific classifier. The specific classifier inherits the features of the more general classifier. Each instance of the specific classifier is de facto also an instance of the general classifier. The UML notation for a GENERALIZATION is shown in Figure 9.

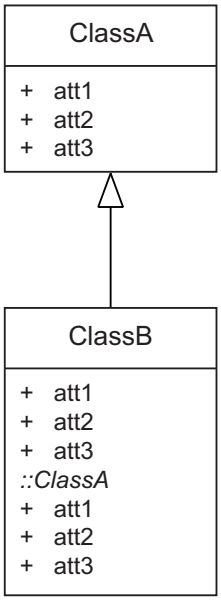


**Figure 9 — UML GENERALIZATION notation**

#### 6.10.1.2 OWL notation

OWL allows the generalization relation between classes by the way of the SubClassOf <RDFS> axiom. This axiom expresses that a class ClassB is a subclass of a class ClassA and as such ClassB is more specific than ClassA. Similarly to the UML GENERALIZATION, it allows the definition of a classification system or taxonomy of classes and subclasses.

#### 6.10.1.3 Rules

A UML GENERALIZATION is implemented as a SubClassOf <RDFS> using an *rdfs:subClassOf* declaration.

Table 20 sets the requirement for the description of GENERALIZATIONs in Ontology <OWL>.

**Table 20 — Relationship – generalization**

| **Requirement** |
|---|
| 19150-2package:relationship-generalization |
| A UML GENERALIZATION shall be implemented as a SubClassOf <RDFS> using an *rdfs:subClassOf* declaration. |

The following example illustrates the definition of a 'subclass' between the specialized 'class' ClassB and the generalized 'class' ClassA.

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA a owl:Class .
exPk:ClassB a owl:Class ;
    rdfs:subClassOf exPk:ClassA .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA"/>
```

```
<owl:Class rdf:about="&exPk;ClassB">
    <rdfs:subClassOf rdf:resource="&exPk;ClassA"/>
</owl:Class>
```

### 6.10.2 Association

#### 6.10.2.1 UML notation

An ASSOCIATION in UML specifies how classifiers are semantically related together. UML allows N-ary (i.e. binary, ternary, quaternary, etc.) associations that relate together multiple classifiers. ISO 19103:—[1)] requires to avoid N-ary association and to use binary association to limit the complexity of models. An association between two classes can be named and can be either unidirectional – i.e. navigable from one end to the other end, or bidirectional – i.e. navigable in both directions. Each end that is navigable must have a multiplicity and a role set as shown in Figure 10.

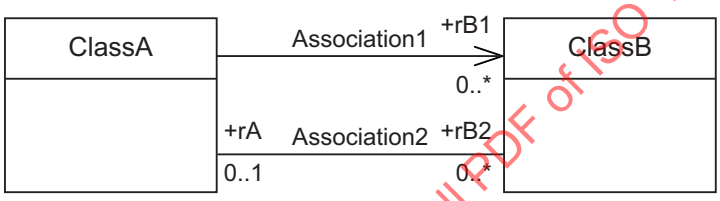


**Figure 10 — UML association notation**

#### 6.10.2.2 OWL notation

The manner OWL deals with ASSOCIATIONs, more specifically with ASSOCIATION ROLEs, is similar to ATTRIBUTEs. An ASSOCIATION ROLE takes the form of an Object Property <OWL>.

OWL has no specific mechanism for the description of an ASSOCIATION name. ASSOCIATION names are defined in UML mostly for documentation purposes.

#### 6.10.2.3 Rules

A ROLE in a UML ASSOCIATION corresponds to an Object Property <OWL>. The Object Property <OWL> connects the Classes <OWL> it associates by specifying its domain –i.e. the Class <OWL> that uses the role, and its range –i.e. the Class <OWL> that plays the role.

The Object Property <OWL> is annotated with a label and its source document. The label provides the name of the UML ASSOCIATION ROLE and uses a *rdfs:label* declaration. The source document identifies the resource defining this ASSOCIATION ROLE. It uses a *rdfs:isDefinedBy* declaration.

If the association is named, the Object Property <OWL> is annotated with the name of the ASSOCIATION. Consequently, this part of ISO 19150 defines an annotation property *associationName* formalized in the *base* ontology (Annex D).

**OWL Definition 3**

RDF/Turtle serialization

```
iso19150-2:associationName a owl:AnnotationProperty ;
    rdfs:domain owl:Class ;
    rdfs:range xsd:String .
```

RDF/XML serialization

```
<owl:AnnotationProperty rdf:about="&iso19150-2;associationName"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain rdf:resource="&owl;Class"/>
</owl:AnnotationProperty>
```

It provides the documentation mechanism for association name. Therefore, the Object Property <OWL> corresponding to ASSOCIATION ROLE is annotated with the name of the ASSOCIATION, using the annotation property *iso19150-2:associationName*.

In a bidirectional ASSOCIATION, one of the two Object Property <OWL> corresponding to ASSOCIATION ROLE is set as the inverse of the other using an *owl:inverseOf* declaration. As such, when an Object Property <OWL> connects an individual I1 to an individual I2, its inverse Object Property <OWL> connects also I2 to I1.

Table 21 sets the requirement for the description of ASSOCIATIONs in Ontology <OWL>.

**Table 21 — Relationship – association**

| **Requirement** |
|---|
| 19150-2package:relationship-association |
| A navigable ROLE in a UML ASSOCIATION shall correspond to an Object Property <OWL>, using an *owl:Object-Property* declaration. It shall be connected to the Class <OWL> resources it is associated with using an *rdfs:domain* declaration. The type of the Object Property <OWL> resources shall be specified using an *rdfs:range* declaration.<br>The Object Property <OWL> shall be annotated with:<br>— a label for the name of the ASSOCIATION ROLE, using *rdfs:label*, and<br>— a source document defining this ASSOCIATION ROLE, using *rdfs:isDefinedBy* to declare the IRI of the resource.<br>If the ASSOCIATION is named, the Object Property <OWL> shall be annotated with the name of the association, using *iso19150-2:associationName*.<br>In a bidirectional ASSOCIATION, one of the two Object Property <OWL> shall be declared as the inverse of the other, using an *owl:inverseOf* declaration. |

Additional annotation for the Object Property <OWL> corresponding to the ASSOCIATION ROLE may be provided such as:

— definition, using a *skos:definition* declaration to provides the semantics of the UML ATTRIBUTE, and

— deprecation, using an *owl:deprecated* declaration, which is set to *"true"^^xsd:boolean* if the Object Property <OWL> is deprecated.

The following example illustrates the OWL representation of a unidirectional association between ClassA and ClassB. It defines the classes ClassA and ClassB with the Object Property <OWL> rB1 where its domain is ClassA and its range is ClassB.

EXAMPLE 1

*RDF/Turtle serialization*

```
exPk:ClassA a owl:Class .
exPk:ClassB a owl:Class .
```



 

```
exPk:ClassA.rB1 a owl:ObjectProperty ;
    rdfs:label “rB1” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    iso19150-2:associationName “Association1” ;
    rdfs:domain exPk:ClassA ;
    rdfs:range exPk:ClassB .
```

RDF/XML serialization

```
<owl:Class rdf:about=”&exPk;ClassA”/>
<owl:Class rdf:about=”&exPk;ClassB”/>
<owl:ObjectProperty rdf:about=”&exPk;ClassA.rB1”>
    <rdfs:label>rB1</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <iso19150-2:associationName>Association1</iso19150-2:associationName>
    <rdfs:domain rdf:resource=”&exPk;ClassA”/>
    <rdfs:range rdf:resource=”&exPk;ClassB”/>
</owl:ObjectProperty>
```

The following example illustrates a bidirectional association in OWL.

EXAMPLE 2

RDF/Turtle serialization

```
exPk:ClassA a owl:Class .
exPk:ClassB a owl:Class .
exPk:ClassA.rB2 a owl:ObjectProperty ;
    rdfs:label “rB2” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    iso19150-2:associationName “Association2” ;
    rdfs:domain exPk:ClassA ;
    rdfs:range exPk:ClassB .
exPk:ClassB.rA> a owl:ObjectProperty ;
    rdfs:label “rA” ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    iso19150-2:associationName “Association2” ;
    rdfs:domain exPk:ClassB ;
    rdfs:range exPk:ClassA ;
```

```
    owl:inverseOf exPk:ClassA.rB2 .
```

RDF/XML serialization

```
<owl:Class rdf:about="&exPk;ClassA"/>
<owl:Class rdf:about="&exPk;ClassB"/>
<owl:ObjectProperty rdf:about="&exPk;ClassA.rB2">
    <rdfs:label>rB2</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <iso19150-2:associationName>Association2</19150-2:associationName>
    <rdfs:domain rdf:resource="&exPk;ClassA"/>
    <rdfs:range rdf:resource="&exPk;ClassB"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="&exPk;ClassB.rA">
    <rdfs:label>rA</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <iso19150-2:associationName>Association2</iso19150-2:associationName>
    <rdfs:domain rdf:resource="&exPk;ClassB"/>
    <rdfs:range rdf:resource="&exPk;ClassA"/>
    <owl:inverseOf rdf:resource="&exPk;ClassA.rB2"/>
</owl:ObjectProperty>
```

Multiplicities in associations are specified through property restrictions as described in 6.9.

### 6.10.3 Aggregation

#### 6.10.3.1 UML notation

An AGGREGATION represents a part-whole association between two CLASSes. UML defines two kinds of AGGREGATION. The first one is the ordinary aggregation also called "shared" AGGREGATION. This kind of aggregation associates the various parts that make a whole feature. The second kind of aggregation is the COMPOSITION also called the "composite" AGGREGATION. This kind of aggregation is a stronger form of aggregation compared to an ordinary aggregation. It requires a part instance be included in at most one composite at a time. When a composite is deleted, all its parts are deleted at the same time.

The UML notation uses a hollow diamond for a "shared" AGGREGATION and a filled diamond for a "composite" AGGREGATION at the whole end of the association as illustrated in Figure 11.

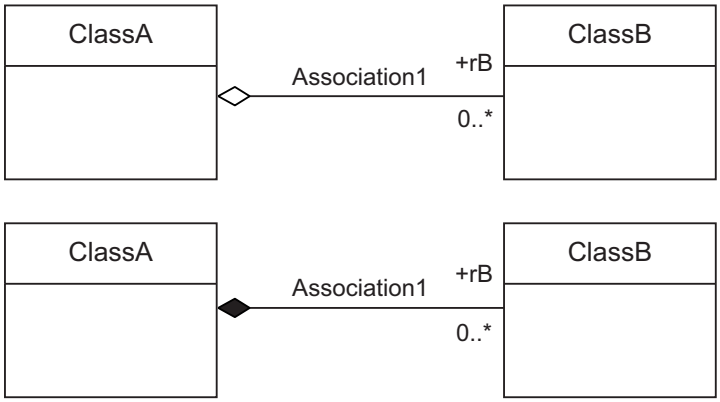


**Figure 11 — UML notation for shared (top) and composite (bottom) AGGREGATIONS**

#### 6.10.3.2 OWL notation

There is no specific mechanism in OWL for aggregation.

#### 6.10.3.3 Rules

AGGREGATIONS follow the same rules as for ASSOCIATIONS described in 6.10.2. However in the specific case of AGGREGATION, the Object Property <OWL> playing the part role is annotated with an annotation providing details on the kind of aggregation.

This part of ISO 19150 defines an annotation property *aggregationType* formalized in the *base* ontology (Annex D).

**OWL Definition 4**

RDF/Turtle serialization

```
iso19150-2:aggregationType a owl:AnnotationProperty ;
    rdfs:domain owl:Class;
    rdfs:range xsd:String .
```

RDF/XML serialization

```
<owl:AnnotationProperty rdf:about="&iso19150-2;aggregationType"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain rdf:resource="&owl;Class"/>
</owl:AnnotationProperty>
```

This annotation property provides the documentation mechanism for the kind of aggregation. The acceptable values for this annotation are:

— "partOfSharedAggregation" means shared aggregation;

— "partOfCompositeAggregation" means composite aggregation.

Table 22 sets the requirement for the description of AGGREGATIONs in Ontology <OWL>.

**Table 22 — Relationship – aggregation**

| Requirement |
|---|
| 19150-2package:relationship-aggregation |
| UML AGGREGATIONs shall follow the requirement for UML ASSOCIATIONs (requirement 19150-2package:relationship-association documented in Table 21).<br>Additionally, the Object Property <OWL> playing the part role shall be annotated using an *iso19150-2:aggregationType* according to the expression:<br>`iso19150-2:aggregationType = "partOfSharedAggregation" / "partOfCompositeAggregation"`<br>— "partOfSharedAggregation" means shared aggregation;<br>— "partOfCompositeAggregation" means composite aggregation. |

The following example illustrates an association role playing the parts in an aggregation association in OWL:

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA.rB a owl:ObjectProperty ;
    rdfs:label "rB" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    iso19150-2:aggregationType "partOfSharedAggregation" ;
    iso19150-2:associationName "Association1" ;
    rdfs:domain exPk:ClassA ;
    rdfs:range exPk:ClassB .
```

RDF/XML serialization

```
<owl:ObjectProperty rdf:about="&exPk;ClassA.rB">
    <rdfs:label>rB</rdfs:label>
    <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
    <iso19150-2:aggregationType>partOfSharedAggregation</iso19150-2:aggregationType>
    <iso19150-2:associationName>Association1</iso19150-2:associationName>
    <rdfs:domain rdf:resource="&exPk;ClassA"/>
    <rdfs:range rdf:resource="&exPk;ClassB"/>
</owl:ObjectProperty>
```

## 6.11 Constraint

### 6.11.1 UML notation

A UML CONSTRAINT is a condition or restriction for the purpose of declaring some of the semantics of an element. It can be expressed in natural language text or in a machine readable language. In a UML model, a constraint may appear as close as possible to the element it constrains in brace symbol ("{aConstraint}"). It may appear just after the element in a class symbol or as a note symbol attached to the element.

### 6.11.2 OWL notation

Although OWL has mechanisms for specifying constraints (e.g. functional, transitive, reflexive objectProperties and functional datatype properties), these constraints are somewhat different to or have a different purpose of UML CONSTRAINTs.

### 6.11.3 Rules

A UML CONSTRAINT on a UML PACKAGE, CLASS, ASSOCIATION, or ATTRIBUTE is expressed in an Ontology <OWL>, Class <OWL>, Data Property <OWL>, or Object Property <OWL> using an annotation.

This part of ISO 19150 defines an annotation property *constraint* formalized in the *base* ontology (Annex D).

**OWL Definition 5**

RDF/Turtle serialization

```
iso19150-2:constraint a owl:AnnotationProperty ;
    rdfs:domain owl:Class;
    rdfs:range xsd:String .
```

RDF/XML serialization

```
<owl:AnnotationProperty rdf:about="&iso19150-2;constraint"/>
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain rdf:resource="&owl;Class"/>
</owl:AnnotationProperty>
```

This annotation property provides a mechanism for documenting constraints for classes and properties.

Table 23 sets the requirement for the description of CONSTRAINTs in Ontology <OWL>.

**Table 23 — Constraint**

| **Requirement** |
|---|
| 19150-2package:constraint |
| A UML CONSTRAINT on a UML CLASS or ATTRIBUTE shall be expressed in a Class <OWL>, Data Property <OWL>, or Object Property <OWL> using an *iso19150-2:constraint* annotation. |

The following example illustrates a constraint on an object property in OWL.

EXAMPLE

RDF/Turtle serialization

```
exPk:ClassA.att1 a owl:ObjectProperty ;
    rdfs:label "att1" ;
    rdfs:isDefinedBy <http://sourceDefinitionIRI> ;
    iso19150-2:constraint "aConstraint" ;
    rdfs:domain exPk:ClassA ;
```

```
        rdfs:range exPk:ClassB .
```

RDF/XML serialization

```
    <owl:ObjectProperty rdf:about="&exPk;ClassA.att1">
        <rdfs:label>att1</rdfs:label>
        <rdfs:isDefinedBy>http://sourceDefinitionIRI</rdfs:isDefinedBy>
        <iso19150-2:constraint>aConstraint</iso19150-2:constraint>
        <rdfs:domain rdf:resource="&exPk;ClassA"/>
        <rdfs:range rdf:resource="&exPk;ClassB"/>
    </owl:ObjectProperty>
```

# 7 Rules for formalizing an application schema in OWL

## 7.1 General

An application schema provides a model for geographic data in a specific universe of discourse or application domain. It supports the access and the exchange of geographic data between suppliers and users, which can then be understood by human and computer systems. ISO 19109:—[2)] provides a meta model for application schemas (i.e. the General Feature Model), and a set of rules for formalizing a specific application schema using a UML profile defined in ISO 19103:—[1)] ISO 19103:—[1)] defines constraints on the UML profile used in conceptual modelling of geographic application schemas, and ISO 19109:—[2)] and ISO 19136:2007[8] provide additional elements used in application schemas. UML models that conform to ISO 19109:—[2)] are in packages stereotyped «applicationSchema». Taken together these allow definition of a specific UML-OWL encoding rule for geographic information models.

This may be contrasted with generic UML-OWL encoding rules, such as those developed by OMG, which support all the options implied by the UML meta model. Three important aspects of the profiles are:

— ISO 19103:—[1)] requires that every navigable association-end must have a role-name;

— ISO 19136:2007[8] defines the stereotype «featureType» for classes that instantiate the meta class FeatureType from ISO 19109:—[2)];

— ISO 19109:—[2)] defines the stereotype «applicationSchema» for packages that formalize an Application Schema.

OWL has been especially designed to enable data to be understood and processable by machines within the Semantic Web environment. It is therefore necessary to provide a parallel set of rules for the formalization of geographic information ontologies using OWL.

Clause 7 defines rules that shall be applied to develop ISO geographic information ontologies in OWL. The same basic meta model is used (i.e. the General Feature Model) and the rules described below follow the general structure of ISO 19109:—[2)], Clause 7. Annex E provides an example of an ISO geographic information ontologY in OWL derived from an application schema in UML.

The requirements for representing an application schema in OWL comprise a single requirements class (Table 24), identified as http://standards.iso.org/iso/19150-2/req/applicationSchema and abbreviated as 19150-2app.

**Table 24 — Requirements class for representing an application schema in OWL**

| **Requirements class** | |
|---|---|
| 19150-2app = http://standards.iso.org/iso/19150-2/req/applicationSchema | |
| **Target type** | Ontology |
| **Dependency** | http://www.w3.org/TR/owl2-syntax/ (OWL) |
| **Dependency** | http://tools.ietf.org/html/rfc3986 (URI Syntax) |
| **Dependency** | http://standards.iso.org/iso/19107/ed-1/en/ (Spatial schema) |
| **Dependency** | http://standards.iso.org/iso/19108/ed-1/en/ (Temporal schema) |
| **Dependency** | http://standards.iso.org/iso/19109/ed-2/en/ (Rules for application schema) |
| **Dependency** | http://standards.iso.org/iso/19112/ed-1/en/ (Spatial referencing by geographic identifiers) |
| **Dependency** | http://standards.iso.org/iso/19115-1/ed-1/en/ (Metadata - Fundamentals) |
| **Dependency** | http://standards.iso.org/iso/19123/ed-1/en/ (Schema for coverage geometry and functions) |
| **Dependency** | http://standards.iso.org/iso/19156/ed-1/en/ (Obeservation and measurement) |
| **Dependency** | http://standards.iso.org/iso/19157/ed-1/en/ (Data quality) |
| **Dependency** | 19150-2package:enumeration |
| **Dependency** | 19150-2package:codelist |
| **Dependency** | 19150-2package:union |
| **Dependency** | http://def.isotc211.org/iso19150/-2/2012/base |
| **Dependency** | http://def.isotc211.org/iso19103/2015/SchemaLanguage |
| **Dependency** | http://def.isotc211.org/iso19107/2003/SpatialSchema |
| **Dependency** | http://def.isotc211.org/iso19108/2002/TemporalSchema |
| **Dependency** | http://def.isotc211.org/iso19109/2015/ApplicationSchema |
| **Dependency** | http://def.isotc211.org/iso19112/2005/LocationByIdentifier |
| **Dependency** | http://def.isotc211.org/iso19115/-1/2014/ MetadataFundamentals |
| **Dependency** | http://def.isotc211.org/iso19123/2005/Coverages |
| **Dependency** | http://def.isotc211.org/iso19157/2013/DataQuality |
| **Requirement** | 19150-2app:identification |
| **Requirement** | 19150-2app:documentation-ontology |
| **Requirement** | 19150-2app:documentation-ontologyComponent |
| **Requirement** | 19150-2app:integration |
| **Requirement** | 19150-2app:featureType |
| **Requirement** | 19150-2app:attributeType |
| **Requirement** | 19150-2app:thematicAttributeType |
| **Requirement** | 19150-2app:coverageFunctionAttributeType |
| **Requirement** | 19150-2app:locationAttributeType |
| **Requirement** | 19150-2app:spatialAttributeType |
| **Requirement** | 19150-2app:temporalAttributeType |
| **Requirement** | 19150-2app:metadataAttributeType |
| **Requirement** | 19150-2app:qualityAttributeType |
| **Requirement** | 19150-2app:attributeOfAttribute |
| **Requirement** | 19150-2app:operation |
| **Requirement** | 19150-2app:featureAssociationRole |
| **Requirement** | 19150-2app:featureAssociationType |

**Table 24** *(continued)*

| **Requirements class** | |
|---|---|
| 19150-2app = http://standards.iso.org/iso/19150-2/req/applicationSchema | |
| **Requirement** | 19150-2app:featureAggregationType |
| **Requirement** | 19150-2app:featureCompositionType |
| **Requirement** | 19150-2app:spatialAssociationType |
| **Requirement** | 19150-2app:temporalAssociationType |
| **Requirement** | 19150-2app:inheritanceRelation |
| **Requirement** | 19150-2app:constraint |
| **Requirement** | 19150-2app:valueAssignment |

## 7.2 Rules for identification

In OWL, an application schema is described in an Ontology <OWL> using an *owl:Ontology* declaration. The Ontology <OWL> consists of a set of RDF triples that specifies the components of the application schema. The *owl:Ontology* declaration carries the name of the application schema and its version as required for the documentation of an application schema.

The name consists of two components:

1) an IRI that identifies the Ontology <OWL>;

2) the title of the application schema that provides a human readable identification.

The IRI is specified in the *owl:Ontology* declaration. The title of the application schema is specified with an annotation property using an *rdfs:label* declaration.

The version of the Ontology <OWL> has also two components:

1) a character string providing the version number or version date of the Ontology <OWL> using an *owl:versionInfo* declaration;

2) a version IRI that identifies the Ontology <OWL> including its version using an *owl:versionIRI* declaration.

NOTE The IRI of the Ontology <OWL> refers to the most recent version of the ontology whereas the version IRI refers to a specific version of the Ontology <OWL>.

Table 25 sets the requirement for the identification of an application schema in Ontology <OWL>.

**Table 25 — Identification**

| **Requirement** |
|---|
| 19150-2app:identification |
| An application schema shall be identified in an Ontology <OWL> using an *owl:Ontology* declaration.<br>The Ontology <OWL> shall be annotated with:<br>— its title, using an *rdfs:label* declaration,<br>— its version, using an *owl:versionInfo* declaration; and<br>— its version IRI, using an *owl:versionIRI* declaration. |

The following example illustrates the definition of an Ontology <OWL> corresponding to an application schema.

EXAMPLE

RDF/Turtle serialization



 

<http://my_organization.org/MyOntology> a owl:Ontology ;

rdfs:label “My application schema ontology name” ;

owl:versionInfo “2012” ;

owl:versionIRI http://my_organization.org/MyOntology/2012.

RDF/XML serialization

```
<owl:Ontology rdf:about="http://my_organization.org/MyOntology">
    <rdfs:label>My application schema ontology name</rdfs:label>
    <owl:versionInfo>2012</owl:versionInfo>
    <owl:versionIRI>http://my_organization.org/MyOntology/2012</owl:versionIRI>
</owl:Ontology>
```

## 7.3 Rules for ontology documentation

### 7.3.1 Ontology documentation

Table 26 sets the requirement for the documentation of application schema in Ontology <OWL>.

**Table 26 — Ontology documentation**

| **Requirement** |
|---|
| 19150-2app:documentation-ontology |
| The documentation of an application schema in Ontology <OWL> shall use, where applicable, the annotation properties listed hereafter:<br>— *rdfs:label* for a human-readable title of the application schema;<br>— *rdfs:isDefinedBy* for an IRI of the resource providing information about the definition of the application schema;<br>— *rdfs:comment* for additional human-readable information;<br>— *owl:deprecated*, when set to “true”^^xsd:boolean, to identify that the Ontology <OWL> is deprecated;<br>— *owl:versionInfo* for a version number or a version date;<br>— *owl:versionIRI* for a version IRI;<br>— *owl:priorVersion* for the identification of the previous version;<br>— *owl:backwardCompatibleWith* for the identification of the previous version of the Ontology <OWL> that is compatible with the current Ontology <OWL>;<br>— *owl:incompatibleWith* for the identification of the previous version of the Ontology <OWL> that is not compatible with the current Ontology <OWL>. |

### 7.3.2 Ontology component documentation

Table 27 sets the requirement for the documentation of application schema components in Ontology <OWL>.

**Table 27 — Ontology component documentation**

| Requirement |
|---|
| 19150-2app:documentation-ontologyComponent |
| The documentation of components of an application schema in Ontology <OWL> shall use, where applicable, the following annotation properties:<br>— *rdfs:label* for a human-readable name of an application schema component;<br>— *skos:definition* for a human-readable definition in natural language of an application schema component;<br>— *rdfs:isDefinedBy* for IRI of the source document providing information about the application schema component;<br>— *rdfs:seeAlso* for IRI providing additional information on the Ontology <OWL> component being described;<br>— *owl:deprecated*, when set to "true"^^xsd:boolean, to identify that the Ontology <OWL> component being described is deprecated;<br>— *owl:versionInfo* for a version of the Ontology <OWL> component being described;<br>— *iso19150-2:isAbstract*, when set to "true"^^xsd:boolean, to identify that the Ontology <OWL> component being described is abstract;<br>— *iso19150-2:associationName* for an identifier of the association in which an Object Property <OWL> participates;<br>— *iso19150-2:aggregationType* for the type of aggregation in which an Object Property <OWL> participates;<br>— *iso19150-2:constraint* for a description of a constraint associated with a Class <OWL>, a Data Property <OWL>, or an Object Property <OWL>. |

## 7.4 Rules for integration

An information model is frequently broken into several independent parts that are afterward integrated. As such, an application schema can refer to other application schemas or part of them as well as to ISO geographic information models.

In OWL, the dependency of an Ontology <OWL> with another is specified with an *owl:imports* declaration.

Table 28 sets the requirement for the integration of Ontologies <OWL> corresponding to different application schemas.

**Table 28 — Integration**

| Requirement |
|---|
| 19150-2app:integration |
| The dependency of an Ontology <OWL> with another shall be specified using an *owl:imports* declaration. |

## 7.5 Rules for FeatureType

As defined in ISO 19109:—[2)], a feature type of an application schema is an instance of FeatureType. A feature type is represented by a CLASS having the stereotype «FeatureType». The most general feature type is 'AnyFeature', which is the class of all features. Accordingly, it is the superclass of all feature types. 'AnyFeature' is stereotyped «FeatureType».

To support the implementation of feature type in OWL, the ontology corresponding to ISO 19109:—[2)] introduces a Class <OWL> AnyFeature, which is the most generic feature type.

In OWL, a feature type corresponds to a Class <OWL> using an *owl:Class* declaration. This Class <OWL> is a subclass of the Class <OWL> AnyFeature.

Table 29 sets the requirement for the documentation of instances of FeatureType (i.e. feature types) in Ontology <OWL>.

**Table 29 — FeatureType**

| Requirement |
|---|
| 19150-2app:featureType |
| An instance of FeatureType (i.e. a feature type) shall be implemented as a Class <OWL> using an *owl:Class* declaration. The Class <OWL> shall be:<br>— identified with a unique IRI,<br>— declared as a subclass of the Class <OWL> AnyFeature defined in the ontology corresponding to ISO 19109:—[2] using an *rdfs:subClassOf* declaration,<br>— annotated with a human readable name, using a *rdfs:label* declaration,<br>— if available, annotated with a human readable definition, using a *skos:definition* declaration, and<br>— if abstract, annotated with the *iso19150-2:isAbstract* annotation property set to true^^boolean. |

Other annotation properties from 7.3.2 may be used to document additional information on a feature type.

The following example illustrates the definition of a feature type in an Ontology <OWL>.

EXAMPLE

RDF/Turtle serialization

```
myapp:AFeatureType a owl:Class ;
    rdfs:subClassOf gfm:AnyFeature ;
    rdfs:label “a feature type” .
```

RDF/XML serialization

```
<owl:Class rdf:about=”&myapp;AFeatureType”>
    <rdfs:subClassOf rdf:resource=“&gfm;AnyFeature“/>
    <rdfs:label>a feature type</rdfs:label>
</owl:Class>
```

## 7.6 PropertyType

### 7.6.1 Attribute

#### 7.6.1.1 Rules for AttributeType

An attribute of an application schema is an instance of AttributeType. It is represented by an ATTRIBUTE. In OWL, an attribute corresponds to a Data Property <OWL> if its value is represented as a literal, and to an Object Property <OWL> if its value is represented as an individual of a Class <OWL> defined by an ISO geographic information standard, an application schema, or another class.

Table 30 sets the requirement for the description of instances of AttributeType (i.e. attributes) in Ontology <OWL>.

**Table 30 — AttributeType**

| Requirement |
|---|
| 19150-2app:attributeType |
| An instance of AttributeType shall be implemented as:<br>— a Data Property <OWL>, if its value is represented as a literal, using an *owl:DatatypeProperty* declaration,<br>— an Object Property <OWL>, if its value is represented as an individual of a Class <OWL>, using an *owl:ObjectProperty* declaration,<br>Data Property <OWL> and Object Property <OWL> shall:<br>— have their type specified using an *rdfs:range* declaration, and<br>— be annotated with:<br>— a name, using a *rdfs:label* declaration, and<br>— a definition, using a *skos:definition* declaration.<br>Cardinalities of a property shall be specified with an *owl:minCardinality* and an *owl:maxCardinality* declaration together with *owl:allValuesFrom*. |

NOTE 1 When no cardinalities are declared the default values are "0" for the minimum cardinality and "unlimited" for the maximum cardinality.

Data Property <OWL> and Object Property <OWL> can be connected to the Class <OWL> it is associated with using an *rdfs:domain* declaration.

NOTE 2 Properties will be most re-usable if a generalized property domain is indicated. The best way to do this is to use the class corresponding to the UML stereotype as the domain – i.e. gfm:AnyFeature or iso19150-2:datatype. Cardinaility constraints can then be used to indicate usage patterns for specific properties in the context of a class.

Other annotation properties from 7.3.2 can be used to document additional information on an attribute such as:

— *rdfs:isDefinedBy* for the source document, and

— *owl:deprecated* set to *"true"^^xsd:boolean* for the identification as deprecated.

The following example illustrates the definition of an 'attribute' through 'data property' and 'object property.'

EXAMPLE

RDF/Turtle serialization

```
myapp:AFeatureType a owl:Class .
myapp:AnotherFeatureType a owl:Class .
myapp:attributeType1 a owl:DatatypeProperty ;
    rdfs:label "A first attribute type" ;
    skos:definition "Definition of a first attribute type" ;
    rdfs:domain gfm:AnyFeature ;
    rdfs:range <http://www.w3.org/2001/XMLSchema#string> .
myapp:AFeatureType.attributeType2 a owl:DatatypeProperty ;
    rdfs:label "A second attribute type" ;
    skos:definition "Definition of a second attribute type" ;
    rdfs:domain myapp:AFeatureType ;
```

```
    rdfs:range <http://www.w3.org/2001/XMLSchema#string> .
myapp:attributeType3 a owl:ObjectProperty ;
    rdfs:label “A third attribute type” ;
    skos:definition “Definition of a third attribute type” ;
    rdfs:domain gfm:AnyFeature ;
    rdfs:range myapp:AnotherFeatureType .
myapp:AFeatureType.attributeType4 a owl:ObjectProperty ;
    rdfs:label “A fourth attribute type” ;
    skos:definition “Definition of a fourth attribute type” ;
    rdfs:domain myapp:AFeatureType ;
    rdfs:range myapp:AnotherFeatureType .
```

RDF/XML serialization

```
<owl:Class rdf:about=”&myapp;AFeatureType “/>
<owl:Class rdf:about=”&myapp;AnotherFeatureType”/>
<owl:DatatypeProperty rdf:about=”&myapp;attributetype1”>
    <rdfs:label>A first attribute type</rdfs:label>
    <skos:definition>Definition of a first attribute type</skos:definition >
    <rdfs:domain rdf:resource=”&gfm;AnyFeature”/>
    <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about=”&myapp;AFeatureType.attributetype2”>
    <rdfs:label>A second attribute type</rdfs:label>
    <skos:definition>Definition of a second attribute type</skos:definition >
    <rdfs:domain rdf:resource=”&myapp;AFeatureType”/>
    <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about=”&myapp;attributetype3”>
    <rdfs:label>A third attribute type</rdfs:label>
    <skos:definition>Definition of a third attribute type</skos:definition >
    <rdfs:domain rdf:resource=”&gfm;AnyFeature”/>
    <rdfs:range rdf:resource=”&myapp;AnotherFeatureType”/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about=”&myapp;AFeatureType.attributetype4”>
    <rdfs:label>A fourth attribute type</rdfs:label>
```

```
    <skos:definition>Definition of a fourth attribute type</skos:definition >
    <rdfs:domain rdf:resource="&myapp;AFeatureType"/>
    <rdfs:range rdf:resource="&myapp;AnotherFeatureType"/>
</owl:ObjectProperty>
```

There are subtypes of AttributeType: ThematicAttributeType, LocationAttributeType, SpatialAttributeType, TemporalAttributeType, and MetadataAttributeType. The above rule and requirement for AttributeType apply to all of them but additional attention is required and described in 7.6.1.2 to 7.6.1.8.

#### 7.6.1.2 Rules for ThematicAttributeType

An instance of ThematicAttributeType carries descriptive information other than those specified in 7.6.1.3 to 7.6.1.8.

Table 31 sets the requirement for the description of instances of ThematicAttributeType in Ontology <OWL>.

**Table 31 — ThematicAttributeType**

| **Requirement** |
|---|
| 19150-2app:thematicAttributeType |
| An instance of ThematicAttributeType shall be implemented as:<br>— a Data Property <OWL> using an owl:DatatypeProperty declaration when its type corresponds to an ISO 19103:—[1)] basic type as listed in Table 12; and<br>— an Object Property <OWL> using an owl:ObjectProperty declaration when its type corresponds to any other ISO 19103:—[1)] types than the ones listed in Table 12 or a user defined type.<br>The range of the Data Property <OWL> shall be associated with its corresponding OWL datatype as identified in Table 12 using an *rdfs:range* declaration. The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19103:—[1)] ontology corresponding to the type used or a Class <OWL> corresponding to the user defined type using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

#### 7.6.1.3 Rules for CoverageFunctionAttributeType

An instance of CoverageFunctionAttributeType carries thematic information whose value varies as a function of spatiotemporal position within the scope of a feature.

Table 32 sets the requirement for the description of instances of CoverageFunctionAttributeType in Ontology <OWL>.

**Table 32 — CoverageFunctionAttributeType**

| **Requirement** |
|---|
| 19150-2app:coverageFunctionAttributeType |
| An instance of CoverageFunctionAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> corresponding to CV_Coverage or its subclasses of the ISO 19123:2005 ontology, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

#### 7.6.1.4 Rules for LocationAttributeType

An instance of LocationAttributeType carries locational information.

Table 33 sets the requirement for the description of instances of LocationAttributeType in Ontology <OWL>.

**Table 33 — LocationAttributeType**

| **Requirement** |
|---|
| 19150-2app:locationAttributeType |
| An instance of LocationAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19112:2003 ontology corresponding to SI_LocationInstance, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

#### 7.6.1.5 Rules for SpatialAttributeType

An instance of SpatialAttributeType carries spatial information, either geometric or topologic.

Table 34 sets the requirement for the description of instances of SpatialAttributeType in Ontology <OWL>.

**Table 34 — SpatialAttributeType**

| **Requirement** |
|---|
| 19150-2app:spatialAttributeType |
| An instance of SpatialAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19107:2003 ontology corresponding to a spatial object of Table 35, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

**Table 35 — Valid spatial objects**

| **Geometric objects** | | | **Topological objects** | |
|---|---|---|---|---|
| **Geometric primitives** | **Geometric complexes** | **Geometric aggregates** | **Topological primitives** | **Topological complexes** |
| GM_Point<br>GM_Curve<br>GM_Surface<br>GM_Solid | GM_CompositePoint<br>GM_CompositeCurve<br>GM_CompositeSurface<br>GM_CompositeSolid<br>GM_Complex | GM_Aggregate<br>GM_MultiPoint<br>GM_MultiCurve<br>GM_MultiSurface<br>GM_MultiSolid | TP_Node<br>TP_Edge<br>TP_Face<br>TP_Solid<br>TP_DirectedNode<br>TP_DirectedEdge<br>TP_DirectedFace<br>TP_DirectedSolid | TP_Complex |
| NOTE The table lists only the highest level classes of spatial objects. Subtypes of these may also be used. | | | | |

#### 7.6.1.6 Rules for TemporalAttributeType

An instance of TemporalAttributeType carries temporal information, either temporal geometric or temporal topologic.

Table 36 sets the requirement for the description of instances of TemporalAttributeType in Ontology <OWL>.

**Table 36 — TemporalAttributeType**

| Requirement |
|---|
| 19150-2app:temporalAttributeType |
| An instance of TemporalAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19108:2002 ontology corresponding to a temporal object of Table 37, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

**Table 37 — Valid temporal objects**

| Temporal geometric primitives | Temporal topological primitives | Topological complexes |
|---|---|---|
| TM_Instant<br>TM_Period | TM_Node<br>TM_Edge | TM_TopologicalComplex |

#### 7.6.1.7 Rules for MetadataAttributeType

An instance of MetadataAttributeType carries metadata information.

Table 38 sets the requirement for the description of instances of MetadataAttributeType in Ontology <OWL>.

**Table 38 — MetadataAttributeType**

| Requirement |
|---|
| 19150-2app:metadataAttributeType |
| An instance of MetadataAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19115-1:2014 ontology, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

#### 7.6.1.8 Rules for QualityAttributeType

An instance of QualityAttributeType carries quality information.

Table 39 sets the requirement for the description of instances of QualityAttributeType in Ontology <OWL>.

**Table 39 — QualityAttributeType**

| Requirement |
|---|
| 19150-2app:qualityAttributeType |
| An instance of QualityAttributeType shall be implemented as an Object Property <OWL> using an *owl:ObjectProperty* declaration.<br>The range of the Object Property <OWL> shall be associated with a Class <OWL> of the ISO 19157:2013 ontology, using an *rdfs:range* declaration. |

The property may be connected to the Class <OWL> it describes using an *rdfs:domain* declaration.

#### 7.6.1.9 Rules for attribute of attribute

Table 40 sets the requirement for the description of attribute of attribute in Ontology <OWL>.

**Table 40 — Attribute of attribute**

| Requirement |
|---|
| 19150-2app:attributeOfAttribute |
| In the case where an attribute is characterized by other attributes, the following steps shall be taken:<br>a) define a new Class <OWL> and name it according to the attribute that is characterized by other attributes;<br>b) define a property (either Data Property <OWL> or Object Property <OWL>) for each attribute (i.e. the attribute that is characterized and the other attributes) with their respective name. The properties shall comply with the requirement 19150-2app:attributeType (see 7.6.1.1). The domain of these properties shall be associated with the new Class <OWL> defined in step a) using an *rdfs:domain* declaration;<br>c) define an Object Property <OWL>) for the attribute being characterized by these attributes with an appropriate name and the new Class <OWL> set as its range. The domain of this new Object Property <OWL> shall be set to the Class <OWL> corresponding to the CLASS owning the attribute. |

### 7.6.2 Rules for Operation

Instances of Operation are not implemented in Ontology <OWL>.

NOTE In the context of application schema, the implementation of operations is considered not mature enough within the OWL technology. Various frameworks and languages are under development and no single solution has been adopted.

Table 41 sets the requirement for the description of Operation in Ontology <OWL>.

**Table 41 — Operation**

| Requirement |
|---|
| 19150-2app:operation |
| Instances of Operation shall not be implemented in Ontology <OWL>. |

### 7.6.3 Rules for FeatureAssociationRole

A role of an application schema is an instance of FeatureAssociationRole. It is represented by a role name of an association end.

Table 42 sets the requirement for the description of instances of FeatureAssociationRole in Ontology <OWL>.

**Table 42 — FeatureAssociationRole**

| Requirement |
|---|
| 19150-2app:featureAssociationRole |
| An instance of FeatureAssociationRole (i.e. a role) shall be implemented as an Object Property <OWL> and shall comply with the requirement 19150-2app:attributeType (see 7.6.1.1). |

The Object Property <OWL> can be connected to the Class <OWL> it is associated with using an *rdfs:domain* declaration.

## 7.7 Rules for FeatureAssociationType

An association of an application schema is an instance of FeatureAssociationType. It is represented by an ASSOCIATION. ISO 19109:—[2)] identifies two cases of association. In the first case, an association links feature types. In the second case, an ASSOCIATION is characterized by instances of PropertyType additionally.

Table 43 sets the requirement for the description of instances of FeatureAssociationType in Ontology <OWL>.

**Table 43 — FeatureAssociationType**

| Requirement |
|---|
| 19150-2app:featureAssociationType |
| Case 1:<br>An instance of FeatureAssociationType (i.e. an association) linking instances of FeatureType shall be implemented by the way of the roles at its association ends.<br>Each role shall be implemented as an Object Property <OWL> and shall comply with the requirement 19150-2app:-featureAssociationRole (see 7.6.3). Each Object Property <OWL> shall have its range specified to the Class <OWL> that plays the role, using an *rdfs:range* declaration. Cardinalities for each Object Property <OWL> in the context of the associated Classes <OWL> shall be set using *owl:minCardinality*, *owl:maxCardinality*, and *owl:cardinality* declarations together with *owl:allValuesFrom*.<br>In a bidirectional association, the Object Property <OWL> corresponding to each end of the association shall be the inverse of each other, using *owl:inverseOf* declarations.<br>If the association is named, the name shall be documented as part of each Object Property <OWL> using the annotation property *iso19150-2:associationName*.<br>Case 2:<br>An association characterized by instances of PropertyType shall be implemented as a Class <OWL>, using an *owl:Class* declaration.<br>Each instance of PropertyType shall be implemented as either Data Property <OWL> or Object Property <OWL> (see 7.6).<br>An Object Property <OWL> shall be defined to link the Class <OWL> standing for the association to each Class <OWL> corresponding to an associated feature type. It shall be identified by a unique IRI and shall be annotated with a *rdfs:label* declaration to provide a human readable name. Also, its definition shall be documented with a *skos:definition* declaration. Range of each Object Property <OWL> must be set accordingly using *rdfs:range* declarations. Cardinalities for each Object Property <OWL> in the context of the created Class <OWL> and the associated Class <OWL> shall be set using *owl:minCardinality*, *owl:maxCardinality*, and *owl:cardinality* declarations together with *owl:allValuesFrom*.<br>In a bidirectional association, the Object Property <OWL> corresponding to each end of the association shall be the inverse of each other, using *owl:inverseOf* declarations.<br>If the association is named, the name shall be documented as part of each Object Property <OWL> using the annotation property *iso19150-2:associationName*. |

## 7.8 Rules for FeatureAggregationType

An aggregation of an application schema is an instance of FeatureAggregationType. It is represented by an AGGREGATION.

Table 44 sets the requirement for the description of instances of FeatureAggregationType in Ontology <OWL>.

**Table 44 — FeatureAggregationType**

| Requirement |
|---|
| 19150-2app:featureAggregationType |
| An instance of FeatureAggregationType shall be implemented similarly to an instance of FeatureAssociationType and shall comply with the requirement 19150-2app:featureAssociationType (see 7.7).<br>Additionally, the aggregation association shall be documented as part of the Object Property <OWL> playing the part role using an *iso19150-2:aggregationType* annotation property set to *partOfSharedAggregation* (see 6.10.3). |

## 7.9 Rules for FeatureCompositionType

A composition of an application schema is an instance of FeatureCompositionType. It is represented by a COMPOSITION.

Table 45 sets the requirement for the description of instances of FeatureCompositionType in Ontology <OWL>.

**Table 45 — FeatureCompositionType**

| Requirement |
|---|
| 19150-2app:featureCompositionType |
| An instance of FeatureCompositionType shall be implemented similarly to an instance of FeatureAssociationType and shall comply with the requirement 19150-2app:featureAssociationType (see 7.7).<br>Additionally, the composition association shall be documented as part of the Object Property <OWL> playing the part role using an *iso19150-2:aggregationType* annotation property set to *partOfCompositeAggregation* (see 6.10.3). |

## 7.10 Rules for SpatialAssociationType

A spatial or topological association of an application schema is an instance of SpatialAssociationType. It is represented by an ASSOCIATION.

Table 46 sets the requirement for the description of instances of SpatialAssociationType in Ontology <OWL>.

**Table 46 — SpatialAssociationType**

| Requirement |
|---|
| 19150-2app:spatialAssociationType |
| An instance of SpatialAssociationType shall be implemented similarly to an instance of FeatureAssociationType and shall comply with the requirement 19150-2app:featureAssociationType (see 7.7). |

## 7.11 Rules for TemporalAssociationType

A temporal association of an application schema is an instance of TemporalAssociationType. It is represented by an ASSOCIATION.

Table 47 sets the requirement for the description of instances of TemporalAssociationType in Ontology <OWL>.

**Table 47 — TemporalAssociationType**

| Requirement |
|---|
| 19150-2app:temporalAssociationType |
| An instance of TemporalAssociationType shall be implemented similarly to an instance of FeatureAssociationType and shall comply with the requirement 19150-2app:featureAssociationType (see 7.7). |

## 7.12 Rules for InheritanceRelation

A generalization relation of an application schema is an instance of InheritanceRelation. It is represented by a GENERALIZATION relation.

Table 48 sets the requirement for the description of instances of InheritanceRelation in Ontology <OWL>.

**Table 48 — InheritanceRelation**

| Requirement |
|---|
| 19150-2app:inheritanceRelation |
| An instance of InheritanceRelation shall be implemented as a subclass between a specialized Class <OWL> and a generalized Class <OWL> using an *rdfs:subClassOf* declaration as part of the specialized Class <OWL> (see 6.10.1). |

## 7.13 Rules for constraints

Table 49 sets the requirement for the description of constraint in Ontology <OWL>.

**Table 49 — Constraint**

| Requirement |
|---|
| 19150-2app:constraint |
| A constraint shall be documented using an annotation property, using *iso19150-2:constraint*, as part of the Class <OWL>, Data Property <OWL>, or Object Property <OWL> it constrains (see 6.11). |

## 7.14 Rules for ValueAssignment

### 7.14.1 Role of Association class

ISO 19109:—[2] defines a meta-class ValueAssignment, instances of which carry metadata describing a property value instance. The primary application of this is to provide information about the way that a specific property value was assigned, for example information about an observation that provided an estimate of the property value. This is expected to be available when the attribute or association role has the stereotype «estimated». Other value assignment mechanisms might include an algorithm for derivation from other properties, or a rule for assignment by a competent authority. The intention is to allow this extra information to be provided in a standard way rather than using an application-specific mechanism, so all applications can provide value metadata in the same way.

### 7.14.2 ValueAssignment property

To support the ValueAssignment requirement the property *iso19150-2:valueAssignment* is formalized in the *base* ontology (Annex D):

**OWL Definition 6**

RDF/Turtle serialization

```
iso19150-2:valueAssignment a owl:ObjectProperty ;
    rdfs:label "value assignment" ;
  skos:definition "property that can be attached to any statement that describes a feature property
    instance, to support a link to the evidence for the value" ;
    rdfs:isDefinedBy <http://standards.iso.org/iso/19109/ed-2/en> ;
    rdfs:domain rdf:Statement .
```

RDF/XML serialization

```
<owl:ObjectProperty rdf:about="&iso19150-2;valueAssignment">
    <rdfs:label>value assignment</rdfs:label>
  <skos:definition>property that can be attached to a statement that describes a feature property
```



 

instance, to support a link to the evidence for the value</skos:definition>

<rdfs:isDefinedBy>http://standards.iso.org/iso/19109/ed-2/en</rdfs:isDefinedBy>

<rdfs:domain rdf:resource="&rdf;Statement"/>

</owl:ObjectProperty>

### 7.14.3 RDF reification pattern

Additional information about a property can be provided using RDF Reification.[16] The reified statement name is a URI, which can be the subject of a statement linking to the value assignment information.

EXAMPLE

A person may be described as a feature using a simple type ns:Person, with two properties, height and IQ:

RDF/Turtle serialization

```
john:Doe a ns:Person ;
    ns:height [
    a basic:Measure ;
    ns:value "1.7"^^xsd:decimal ;
    ns:uom unit:m .
] ;
ns:IQ "150"^^xsd:positiveInteger .
```

RDF/XML serialization

```
<ns:Person rdf:about="&john;Doe">
    <basic:Measure rdf:about="&ns;height">
        <ns:value rdf:datatype="&xsd;decimal">1.7</ns:value>
        <ns:uom rdf:resource="&unit;m"/>
    </basic:Measure>
    <ns:IQ rdf:datatype="&xsd; positiveInteger">150</ns:IQ>
</ns:Person>
```

The property values were each determined by some observation procedure: height through a measurement designated http://example.org/measurement/abc123, and IQ through a test designated http://example.org/test/zyx987. These are indicated using RDF reification as follows:

RDF/Turtle serialization

```
john:height a rdf:Statement ;
    rdf:subject john:Doe ;
    rdf:predicate ns:height ;
```

```
    rdf:object [ a basic:Measure ;
        ns:value “1.7”^^xsd:decimal ;
        ns:uom unit:m .
    ] .
    john:height iso19150-2:valueAssignment <http://example.org/measurement/abc123> .
```

RDF/XML serialization

```
<rdf:Description rdf:about=”&john;height”>
    <rdf:subject resource=”&john;Doe”/>
    <rdf:predicate resource=”&ns;height”/>
    <rdf:object>
        <ns:value rdf:datatype=”&xsd;decimal”>1.7</ns:value>
        <ns:uom rdf:resource=”&unit;m”/>
    </rdf:object>
    <rdf:type resource=”http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement”/>
            <iso19150-2:valueAssignment>http://example.org/measurement/abc123</iso19150-
    2:valueAssignment>
</rdf:Description>
```

RDF/Turtle serialization

```
john:IQ a rdf:Statement ;
    rdf:subject john:Doe ;
    rdf:predicate ns:IQ ;
    rdf:object “150”^^xsd:positiveInteger .
john:IQ iso19150-2:valueAssignment <http://example.org/test/zyx987> .
```

RDF/XML serialization

```
<rdf:Description rdf:about=”&john;IQ”>
    <rdf:subject resource=”&john;Doe”/>
    <rdf:predicate resource=”&ns;IQ”/>
    <rdf:object rdf:datatype=”&xsd;positiveInteger >150</rdf:object>
    <rdf:type resource=”http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement”/>
  <iso19150-2:valueAssignment>http://example.org/test/zyx987</iso19150-2:valueAssignment>
</rdf:Description>
```

NOTE RDF reification is only supported in OWL Full.

### 7.14.4 SPARQL named-graph pattern

SPARQL[15] named-graphs are an alternative method to give identity to a set of RDF statements, for which TriG[17] provides a serialization syntax. The graph name is a URI, which can be the subject of a statement linking to the value assignment information.

EXAMPLE

Using the same data as in the previous (reification) example the value assignment statements are indicated using named graphs as follows:

TriG serialization

```
john:height
{
    john:Doe ns:height [
        a basic:Measure ;
        ns:value “1.7”^^xsd:decimal ;
        ns:uom unit:m .
    ] .
}
john:height iso19150-2:valueAssignment <http://example.org/measurement/abc123> .

john:IQ
{
    john:Doe ns:IQ “150”^^xsd:positiveInteger .
}
john:IQ iso19150-2:valueAssignment <http://example.org/test/zyx987> .
```

NOTE SPARQL named-graphs and TriG syntax are not yet supported by all RDF/OWL tools.

### 7.14.5 Rules for ValueAssignment in OWL tern

Either of these patterns provides a URI for a value assignment statement, allowing a link to be attached to the value assignment information, while also preserving a simple path from a feature instance to the value within the RDF graph.

Table 50 sets the requirement for the description of value assignment in Ontology <OWL>.

**Table 50 — ValueAssignment**

| Requirement |
|---|
| 19150-2app:valueAssignment |
| An instance of ValueAssignment shall be implemented by creating either<br>— a RDF resource that reifies the property instance, or<br>— a named graph containing the property instance.<br>The reified property or graph shall have a single property *iso19150-2:valueAssignment*, whose value identifies a description of the value assignment process used for this value. |

# Annex A
(normative)

# Abstract test suite

## A.1 Conformance classes

Annex A describes tests corresponding to each requirement. They are packaged in two conformance classes:

— Conformance class for conversion of a UML package from the ISO/TC 211 Harmonized Model to OWL, 19150-2package-conf = http://standards.iso.org/iso/19150-2/conf/package

— Conformance class for formalization of an application schema in OWL: 19150-2app-conf = http://standards.iso.org/iso/19150-2/conf/applicationSchema

The conformance class 19150-2package-conf includes the following tests:

— Name;

— Ontology name;

— RDF namespace for ontology;

— Class name;

— Datatype name;

— Property name;

— Names for code lists and their members;

— UML package;

— UML class;

— UML abstract class;

— Data property;

— Object property;

— UML enumeration;

— UML code list;

— Union class;

— UML multiplicity;

— UML generalization/inheritance;

— UML association;

— UML aggregation;

— UML constraint.

The conformance class 19150-2app-conf includes the following tests:

— Application schema, rules for identification;
— Application schema, rules for ontology documentation;
— Application schema, rules for ontology component documentation;
— Application schema, rules for integration;
— Application schema, rules for FeatureType;
— Application schema, rules for AttributeType;
— Application schema, rules for ThematicAttributeType;
— Application schema, rules for CoverageFunctionAttributeType;
— Application schema, rules for LocationAttributeType;
— Application schema, rules for SpatialAttributeType;
— Application schema, rules for TemporalAttributeType;
— Application schema, rules for MetadataAttributeType;
— Application schema, rules for QualityAttributeType;
— Application schema, rules for attribute of attribute;
— Application schema, rules for Operation;
— Application schema, rules for FeatureAssociationRole;
— Application schema, rules for FeatureAssociationType linking instances of FeatureType;
— Application schema, rules for FeatureAssociationType characterized by instances of PropertyType;
— Application schema, rules for FeatureAggregationType;
— Application schema, rules for FeatureCompositionType;
— Application schema, rules for SpatialAssociationType;
— Application schema, rules for TemporalAssociationType;
— Application schema, rules for InheritanceRelation;
— Application schema, rules for constraints;
— Application schema, rules for ValueAssignment.



## A.2 Naming

### A.2.1 Name

The test 19150-2package-conf:name for " name" is as follows:

a) Test purpose Verify that all names of the ontology don't use space characters and that punctuation characters different than dash and underscore are replaced by underscore characters.

b) Test method — Inspect the *owl:Ontology*, *owl:Class*, *rdfs:Datatype*, *owl:DatatypeProperty*, *owl:ObjectProperty*, *skos:ConceptScheme*, *skos:Concept*, and *skos:Collection* declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:name (6.2.1).

d) Test type — Capability test.

### A.2.2 Ontology name

The test 19150-2package-conf:ontologyName for "ontology name" is as follows:

a) Test purpose — Verify that the ontology name URI consists of a base URI and the UML package separated by a "/" character.

b) Test method — Inspect the *owl:Ontology* declaration of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:ontologyName (6.2.2).

d) Test type — Capability test.

### A.2.3 RDF namespace for ontology

The test 19150-2package-conf:rdfNamespace for "RDF namespace for ontology" is as follows:

a) Test purpose — Verify that the RDF namespace of the ontology consists of the ontology name with the "#" character appended at the end.

b) Test method — Inspect the namespace declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:rdfNamespace (6.2.3).

d) Test type — Capability test.

### A.2.4 Class name

The test 19150-2package-conf:className for "class name" is as follows:

a) Test purpose — Verify that each class name consists of the RDF namespace of the ontology followed by the UML class name.

b) Test method — Inspect the *owl:Class* declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:className (6.2.4).

d) Test type — Capability test.

### A.2.5 Datatype name

The test 19150-2package-conf:datatypeName for "datatype name" is as follows:

a) Test purpose — Verify that each datatype name consists of the RDF namespace of the ontology followed by the local name of the datatype.

b) Test method — Inspect the *rdfs:Datatype* declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:datatypeName (6.2.5).

d) Test type — Capability test.

### A.2.6 Property name

The test 19150-2package-conf:propertyName for "property name" is as follows:

a) Test purpose — Verify that each property name consists of the RDF namespace of the ontology followed by the UML ATTRIBUTE localName if the attribute is scoped to the UML package or the UML CLASS name followed in order by a "." (full stop) and the UML ATTRIBUTE localName if the ATTRIBUTE is scoped to the UML CLASS.

b) Test method — Inspect the *owl:DatatypeProperty* and owl:*ObjectProperty* declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:propertyName (6.2.6).

d) Test type — Capability test.

### A.2.7 Names for code lists and their members

The test 19150-2package-conf:codeName for "names for code lists and their members" is as follows:

a) Test purpose — Verify that each *skos:ConceptScheme* corresponding to a codeList class consists of the ontology URI followed by the string "/code/" followed by the UML CLASS name. Verify that each *skos:Concept* corresponding to a codeList item consists of the concept-scheme URI followed by the string "/" followed by the UML ATTRIBUTE localName.

b) Test method — Inspect the *skos:ConceptScheme* and *skos:Concept* declarations of the Ontology <OWL>.

c) Reference — Requirement 19150-2package:codeName (6.2.7).

d) Test type — Capability test.

## A.3 UML package

The test 19150-2package-conf:package for "UML package" is as follows:

a) Test purpose — Verify the existence of one or more OWL ontologies for each ISO geographic information standard UML model and included PACKAGEs and their respective *owl:Ontology* declaration. Verify the annotation property declarations for the ontologies: *rdfs:label* for the full name of the ontology, *dct:source* for the source for the ontology, *owl:versionInfo* for the version date of the reference document or the ontology. Verify that each sub package corresponds to specific ontology and that the dependency is set in the parent ontology with *owl:imports*.

b) Test method — Inspect:

— existence of one or more Ontology <OWL> for each ISO geographic information standard UML model,

— *owl:Ontology*, *rdfs:label*, *dct:source*, and *owl:versionInfo* declarations,

— *owl:imports* declaration in the parent ontology, and

— owl:imports of the 19150-2 base ontology.

c) Reference — Requirement 19150-2package:package (6.3).

d) Test type — Capability test.

## A.4 UML class

The test 19150-2package-conf:class for "UML class" is as follows:

a) Test purpose — Verify the existence of a Class <OWL> for each UML CLASS definition and their respective *owl:Class* declaration. Verify the annotation property declarations for the class: *rdfs:label* for its label name and *rdfs:isDefinedBy* for the source document.

b) Test method — Inspect the Ontology <OWL> for:

— Inspect the existence of an Class <OWL> for each UML CLASS, and

— owl:Class, rdfs:label and rdfs:isDefinedBy declarations.

c) Reference — Requirement 19150-2package:class (6.4).

d) Test type — Capability test.

## A.5 UML abstract class

The test 19150-2package-conf:abstractClass for "UML abstract class" is as follows:

a) Test purpose — Verify the existence of a Class <OWL> for each UML ABSTRACT CLASS definition and their respective *owl:Class* declaration. Verify the compliancy with the A.4 abstract test for UML CLASS. Verify the annotation property declaration for the abstract class: *iso19150-2:isAbstract* for its identification as abstract class.

b) Test method — Inspect the Ontology <OWL> for:

— the existence of a Class <OWL> for each UML CLASS, and

— owl:Class, rdfs:label, skos:definition, rdfs:isDefinedBy, and iso19150-2:isAbstract declarations.

c) Reference — Requirement 19150-2package:abstractClass (6.5).

d) Test type — Capability test.

## A.6 UML attribute

### A.6.1 Data property

The test 19150-2package-conf:attribute-dataProperty for "data property" is as follows:

a) Test purpose — Verify the existence of a Data Property <OWL> for each UML ATTRIBUTE described by data values and their respective *owl:DatatypeProperty* declaration. Verify the linkage for each Data Property <OWL> to its related Class <OWL> with the *rdfs:domain* declaration. Verify the data value type for each Data Property <OWL> with the *rdfs:range* declaration. Verify the annotation property declarations for the Data Property <OWL>: *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source document.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Data Property <OWL> for each UML ATTRIBUTE described by data values, and

— owl:DatatypeProperty, rdfs:domain, rdfs:range, rdfs:label, and rdfs:isDefinedBy declarations.

c) Reference — Requirement 19150-2package:attribute-dataProperty (6.6.3.1).

d) Test type — Capability test.

### A.6.2 Object property

The test 19150-2package-conf:attribute-objectProperty for "object property" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each UML ATTRIBUTE described by a UML CLASS and their respective *owl:ObjectProperty* declaration. Verify the linkage for each Object Property <OWL> to its related Class <OWL> with the *rdfs:domain* declaration. Verify the type for each Object Property <OWL> with the *rdfs:range* declaration. Verify the annotation property declarations for the Object Property <OWL>: *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source document.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each UML ATTRIBUTE described by a UML CLASS, and

— owl:ObjectProperty, rdfs:domain, rdfs:range, rdfs:label, and rdfs:isDefinedBy source.

c) Reference — Requirement 19150-2package:attribute-objectProperty (6.6.3.2).

d) Test type — Capability test.

## A.7 Enumeration

The test 19150-2package-conf:enumeration for "enumeration" is as follows:

a) Test purpose — Verify the existence of a Datatype <RDFS> for each UML ENUMERATION and their respective *rdfs:Datatype* declaration. Verify the enumeration of values with the *owl:oneOf* and *rdf:List* declarations. Verify the annotation property declarations for the datatype: *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source document.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Datatype <RDFS> for each UML ENUMERATION, and

— owl:oneOf, rdfs:range, rdfs:label, and rdfs:isDefinedBy declarations.

c) Reference — Requirement 19150-2package:enumeration (6.7.1).

d) Test type — Capability test.

## A.8 Code list

The test 19150-2package-conf:codelist for "code list" is as follows:

a) Test purpose — Verify the existence of a Class <OWL>, a ConceptScheme <SKOS>, and Collection <SKOS>- for each UML code list and assertions of the relationship between them. Verify the existence of a Concept <SKOS> for each UML code list item and their membership in the concept scheme and collection. Verify the annotation property declarations for the datatype: *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source for the definition.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Class <OWL> for each code list,

— existence of a ConceptScheme <SKOS> for each codelist, with the property *dct:isFormatOf* [class implementing the code list],

— existence of a Concept <SKOS> for each item on each codelist, with a property *rdf:type* [class implementing the code list] and a property *skos:inScheme* [concept scheme implementing the code list],

— existence of a Collection <SKOS> for each codelist, and

— rdfs:label, rdfs:isDefinedBy and rdfs:isDefinedBy properties.

c) Reference — Requirement 19150-2package:codelist (6.7.2).

d) Test type — Capability test.

## A.9 Union class

The test 19150-2package-conf:union for "union class" is as follows:

a) Test purpose — Verify the existence of a Class <OWL> for each UML class with the stereotype «Union», with membership corresponding to the class extension of the UNION. Verify the annotation property declarations *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source for the definition.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Class <OWL> for each UNION CLASS, with the property owl:unionOf whose value is a collection of classes corresonding with the class extension of the UNION CLASS, and

— rdfs:label and rdfs:isDefinedBy declarations.

c) Reference — Requirement 19150-2package:union (6.8).

d) Test type — Capability test.

## A.10 UML multiplicity

The test 19150-2package-conf:multiplicity for "multiplicity" is as follows:

a) Test purpose — Verify the Restriction <OWL> for cardinality corresponding to UML MULTIPLICITY and their respective *rdfs:Restriction*, *owl:cardinality*, *owl:minCardinality*, *owl:maxCardinality* and *owl:allValuesFrom* declarations.

b) Test method — Inspect the Ontology <OWL> for:

— default cardinality and explicit cardinality specifications, and

— owl:Restriction, owl:cardinality, owl:minCardinality, owl:maxCardinality and owl:allValuesFrom declarations.

c) Reference — Requirement 19150-2package:multiplicity (6.9).

d) Test type — Capability test.

## A.11 UML generalization/inheritance

The test 19150-2package-conf:relationship-generalization for "generalization/inheritance" is as follows:

a) Test purpose — Verify the existence of SubClassOf <RDFS> assertion for each Class <OWL> corresponding to a UML specialized CLASS in a UML generalization/inheritance relation and the respective *rdfs:subClassOf* declaration.

b) Test method — Inspect the Ontology <OWL> for:

— existence of SubClassOf <RDFS> assertion for each UML specialized CLASS, and

— rdfs:subClassOf declaration.

c) Reference — Requirement 19150-2package:relationship-generalization (6.10.1).

d) Test type — Capability test.

## A.12 UML association

The test 19150-2package-conf:relationship-association for "association" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each UML ROLE in an ASSOCIATION and its respective *owl:ObjectProperty* declaration. Verify the linkage for each Object Property <OWL> to its related Class <OWL> with the *rdfs:domain* declaration. Verify the Class <OWL> playing the role for each Object Property <OWL> with the *rdfs:range* declaration. Verify the annotation property declarations for the object property: *rdfs:label* for its label name, and *rdfs:isDefinedBy* for the source document.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each UML ASSOCIATION ROLE, and

— owl:ObjectProperty, rdfs:domain, rdfs:range, rdfs:label, and rdfs:isDefinedBy declarations.

c) Reference — Requirement 19150-2package:relationship-association (6.10.2).

d) Test type — Capability test.

## A.13 UML aggregation

The test 19150-2package-conf:relationship-aggregation for "aggregation" is as follows:

a) Test purpose — Verify the compliancy with the A.12 abstract test for UML ASSOCIATION. Verify the existence of the annotation property *iso19150-2:aggregationType* in the Object Property <OWL> playing the part role and its value.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each UML ROLE in an AGGREGATION or COMPOSITION association, and

— owl:ObjectProperty, rdfs:domain, rdfs:range, rdfs:label, rdfs:isDefinedBy, and iso19150-2:aggregationType declarations.

c) Reference — Requirement 19150-2package:relationship-aggregation (6.10.3).

d) Test type — Capability test.

## A.14 UML constraint

The test 19150-2package-conf:constraint for "constraint" is as follows:

a) Test purpose — Verify the existence of an annotation property corresponding to each UML CONSTRAINT associated either to a UML CLASS or a UML ATTRIBUTE and its *iso19150-2:constraint* declaration. Verify the annotation property is part of its corresponding Class <OWL>, Data Property <OWL>, or Object Property <OWL>.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an annotation property for each UML CONSTRAINT,

— iso19150-2:constraint declaration, and

— corresponding owl:Class, owl:DatatypeProperty, and owl:ObjectProperty declaration.

c) Reference — Requirement 19150-2package:constraint (6.11).

d) Test type — Capability test.

## A.15 Application schema, rules for identification

The test 19150-2app-conf:identification for "application schema, rules for identification" is as follows:

a) Test purpose — Verify the existence of an Ontology <OWL> for the application schema and its *owl:Ontology* declaration. Verify the annotation property declarations for the ontology: *rdfs:label* for the full name of the Ontology <OWL>, *owl:versionInfo* for its version number or the version date, and *owl:versionIRI* for its version IRI.

b) Test method — Inspect the existence of an Ontology <OWL> for the application schema, and *owl:Ontology*, *rdfs:label*, *owl:versionInfo*, and *owl:versionIRI* declarations.

c) Reference — Requirement 19150-2app:identification (7.2).

d) Test type — Capability test.

## A.16 Application schema, rules for documentation

### A.16.1 Application schema, rules for ontology documentation

The test 19150-2app-conf:documentation-ontology for "ontology documentation" is as follows:

a) Test purpose — Verify the Ontology <OWL> documentation corresponding to the application schema uses the following annotation property: *rdfs:label*, *rdfs:isDefinedBy*, *rdfs:comment*, *owl:deprecated*, *owl:versionInfo*, *owl:versionIRI*, *owl:priorVersion*, *owl:backwardCompatibleWith*, and *owl:incompatibleWith.*

b) Test method — Inspect the Ontology <OWL> annotation properties.

c) Reference — Requirement 19150-2app:documentation-ontology (7.3.1).

d) Test type — Capability test.

### A.16.2 Application schema, rules for ontology component documentation

The test 19150-2app-conf:documentation-ontologyComponent for "ontology component documentation" is as follows:

a) Test purpose — Verify the Ontology <OWL> component documentation corresponding to the application schema uses the following annotation property: *rdfs:label*, *skos:definition*, *rdfs:isDefinedBy*, *rdfs:seeAlso*, *owl:deprecated*, *owl:versionInfo*, *iso19150-2:isAbstract*, *iso19150-2:associationName*, *iso19150-2:aggregationType*, and *iso19150-2:constraint*.

b) Test method — Inspect the Ontology <OWL> components annotation properties.

c) Reference — Requirement 19150-2app:documentation-ontologyComponent (7.3.2).

d) Test type — Capability test.

## A.17 Application schema, rules for integration

The test 19150-2app-conf:integration for "application schema, rules for integration" is as follows:

a) Test purpose — Verify the existence of dependencies in the Ontology <OWL> with others and their correspondence with the dependencies that the application schema has with other application schemas. Verify the *owl:imports* declarations in the Ontology <OWL>.

b) Test method — Inspect the Ontology <OWL> for:

— existence of dependencies in the Ontology <OWL> for the application schema, and

— owl:imports declarations.

c) Reference — Requirement 19150-2app:integration (7.4).

d) Test type — Capability test.

## A.18 Application schema, rules for FeatureType

The test 19150-2app-conf:featureType for "application schema, rules for FeatureType" is as follows:

a) Test purpose — Verify the existence of a Class <OWL> for each instance of FeatureType of the application schema and their respective *owl:Class* declaration. Verify the annotation property declarations: *rdfs:label* for its label name, *skos:definition* for its definition, and *iso19150-2:isAbstract*. Verify that the Class <OWL> is a subclass of the Class <OWL> AnyFeature from the ontology corresponding to ISO 19109:—[2)]

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Class <OWL> for each instance of FeatureType the application schema, and

— owl:Class, rdfs:label, skos:definition, rdfs:subClassOf, and iso19150-2:isAbstract declarations.

c) Reference — Requirement 19150-2app:featureType (7.5).

d) Test type — Capability test.

## A.19 Application schema, rules for AttributeType

The test 19150-2app-conf:attributeType for "application schema, rules for AttributeType" is as follows:

a) Test purpose — Verify the existence of a Data Property <OWL> or an Object Property <OWL> for each instance of AttributeType of the application schema and their respective *owl:DatatypeProperty* or *owl:ObjectProperty* declaration. Verify the annotation property declarations for the Data Property <OWL> or an Object Property <OWL>: *rdfs:label* for its *label name* and *skos:definition* for its definition. Verify the *rdfs:range* declaration.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Data Property <OWL> or an Object Property <OWL> for each instance of AttributeType of the application schema, and

— owl:DatatypeProperty, owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:attributeType (7.6.1.1).

d) Test type — Capability test.

## A.20 Application schema, rules for ThematicAttributeType

The test 19150-2app-conf:thematicAttributeType for "application schema, rules for ThematicAttributeType" is as follows:

a) Test purpose — Verify the existence of a Data Property <OWL> or an Object Property <OWL> for each instance of ThematicAttributeType of the application schema and their respective *owl:DatatypeProperty* or *owl:ObjectProperty* declaration. Verify the annotation property declarations for the Data Property <OWL> or an Object Property <OWL>: *rdfs:label* for its *label name* and *skos:definition* for its definition. Verify the *rdfs:range* declaration.

b) Test method — Inspect the Ontology <OWL> for:

— existence of a Data Property <OWL> or an Object Property <OWL> for each instance of ThematicAttributeType of the application schema, and

— owl:DatatypeProperty, owl:ObjectProperty, rdfs:label, skos:definition, rdfs:range declarations.

c) Reference — Requirement 19150-2app:thematicAttributeType (7.6.1.2).

d) Test type — Capability test.

## A.21 Application schema, rules for CoverageFunctionAttributeType

The test 19150-2app-conf:coverageFunctionAttributeType for "application schema, rules for CoverageFunctionAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of CoverageGeometryAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.19 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19123:2005 ontology.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of CoverageFunctionAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:coverageGeometryAttributeType (7.6.1.3).

d) Test type — Capability test.

## A.22 Application schema, rules for LocationAttributeType

The test 19150-2app-conf:locationAttributeType for "application schema, rules for LocationAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of LocationAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.20 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19112:2003 SI_LocationInstance UML class.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of LocationAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, rdfs:subPropertyOf, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:locationAttributeType (7.6.1.4).

d) Test type — Capability test.

## A.23 Application schema, rules for SpatialAttributeType

The test 19150-2app-conf:spatialAttributeType for "application schema, rules for SpatialAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of SpatialAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.19 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19107:2003 ontology.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of SpatialAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:spatialAttributeType (7.6.1.5).

d) Test type — Capability test.

## A.24 Application schema, rules for TemporalAttributeType

The test 19150-2app-conf:temporalAttributeType for "application schema, rules for TemporalAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of TemporalAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.19 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19108:2002 ontology.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of TemporalAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:temporalAttributeType (7.6.1.6).

d) Test type — Capability test.

## A.25 Application schema, rules for MetadataAttributeType

The test 19150-2app-conf:metadataAttributeType for "application schema, rules for MetadataAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of MetadataAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.19 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19115-1:2014 ontology.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of MetadataAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:metadataAttributeType (7.6.1.7).

d) Test type — Capability test.

## A.26 Application schema, rules for QualityAttributeType

The test 19150-2app-conf:qualityAttributeType for "application schema, rules for QualityAttributeType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of QualityAttributeType of the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with the A.19 abstract test "application schema, rules for AttributeType." Verify that the range of the object property declaration corresponds to a Class <OWL> of the ISO 19157:2013 ontology.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of MetadataAttributeType of the application schema, and

— owl:ObjectProperty, rdfs:label, skos:definition, and rdfs:range declarations.

c) Reference — Requirement 19150-2app:qualityAttributeType (7.6.1.8).

d) Test type — Capability test.

## A.27 Application schema, rules for attribute of attribute

The test 19150-2app-conf:attributeOfAttribute for "application schema, rules for attribute of attribute" is as follows:

a) Test purpose — Verify the existence of Data Property <OWL> or Object Property <OWL> for each ATTRIBUTE of CLASSes of the application schema described by other ATTRIBUTEs. Verify the existence of Data Property <OWL> or Object Property <OWL> for each ATTRIBUTE of CLASSes of the application schema characterizing described ATTRIBUTEs. Verify the existence of domain declarations for each Data Property <OWL> and Object Property <OWL> set to a Class <OWL> that owns them. Verify the existence of an Object Property <OWL>, its range declaration sets to one of the above Class <OWL>, and its domain declaration sets to the Class <OWL> correponding to the CLASS of the application schema that owns the respective ATTRIBUTE.

b) Test method — Inspect the Ontology <OWL> for:

— existence of Data Property <OWL> or Object Property <OWL> for each ATTRIBUTE of CLASSes of the application schema described by other ATTRIBUTEs, and their owl:DatatypeProperty or owl:ObjectProperty declarations,

— existence of Data Property <OWL> or Object Property <OWL> for each ATTRIBUTE of CLASSes of the application schema characterizing described ATTRIBUTEs, and their owl:DatatypeProperty or owl:ObjectProperty declarations,

— existence of domain declarations for each Data Property <OWL> and Object Property <OWL> set to a Class <OWL> that owns it, and its respective owl:Class and rdfs:domain declarations,

— existence of Object Property <OWL>, its range declaration sets to one of the above Class <OWL>, and its domain declaration sets to the Class <OWL> correponding to the CLASS of the application schema that owns the respective ATTRIBUTE, and its rdfs:range, rdfs:domain, and owl:Class declarations.

c) Reference — Requirement 19150-2app:attributeOfAttribute (7.6.1.9).

d) Test type — Capability test.

## A.28 Application schema, rules for Operation

The test 19150-2app-conf:Operation for "application schema, rules for Operation" is as follows:

a) Test purpose — Verify that instances of Operation are non implemented in any manner in the Ontology <OWL>.

b) Test method — Inspect the Ontology <OWL> for any instances of implementation of Operation.

c) Reference — Requirement 19150-2app:operation (7.6.2).

d) Test type — Basic/Capability test.

## A.29 Application schema, rules for FeatureAssociationRole

The test 19150-2app-conf:featureAssociationRole for "application schema, rules for FeatureAssociationRole" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each instance of FeatureAssociationRole of the application schema and their respective *owl:ObjectProperty declaration*. Verify the compliancy with A.19 abstract test "application schema, rules for AttributeType."

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each instance of FeatureAssociationRole of the application schema, and

— compliancy of the owl:ObjectProperty declaration with A.19 abstract test "application schema, rules for AttributeType."

c) Reference — Requirement 19150-2app:featureAssociationRole (7.6.3).

d) Test type — Capability test.

## A.30 Application schema, rules for FeatureAssociationType

### A.30.1 Application schema, rules for FeatureAssociationType linking instances of FeatureType

The test 19150-2app-conf:featureAssociationType-LinkingFeatureTypes for "FeatureAssociationType linking instances of FeatureType" is as follows:

a) Test purpose — Verify the existence of an Object Property <OWL> for each role of an instance of a FeatureAssociationType in the application schema and their respective *owl:ObjectProperty* declaration. Verify the compliancy with A.29 abstract test "application schema, FeatureAssociationRole." Verify the domain of the Object Property <OWL>, its *rdfs:domain* declaration. Verify the range of the Object Property <OWL>, its *rdfs:range* declaration. Verify the cardinalities according to the cardinalities of the application schema, with their *rdfs:Restriction*, *owl:cardinality*, *owl:minCardinality*, and *owl:maxCardinality* declarations together with *owl:allValuesFrom*. Verify for bidirectional associations that Object Properties <OWL> are inverse of each other, and the *owl:inverseOf* declaration. Verify the annotation property *iso19150-2:associationName* of the Object Properties <OWL> for named associations.

b) Test method — Inspect the Ontology <OWL> for:

— existence of an Object Property <OWL> for each role of a FeatureAssociationType, and

— owl:ObjectProperty, rdfs:domain, rdfs:range, rdfs:label, skos:definition, rdfs:Restriction, owl:cardinality, owl:minCardinality, owl:maxCardinality, owl:allValuesFrom, owl:inverseOf, and iso19150-2:associationName declarations.

c) Reference — Requirement 19150-2app:featureAssociationType (7.7).

d) Test type — Capability test.

### A.30.2 Application schema, rules for FeatureAssociationType characterized by instances of PropertyType

The test 19150-2app-conf:featureAssociationType-CharacteizedByProperties for "FeatureAssociationType characterized by instances of PropertyType" is as follows:

a) Test purpose — Verify the existence of a Class <OWL> for each instance of FeatureAssociationType characterized by instances of PropertyType in the application schema and its respective *owl:Class* declaration. Verify the existence of a Data Property <OWL> or an Object Property <OWL> for each instance of PropertyType characterizing this instance of FeatureAssociationType with their *owl:DatatypeProperty* or *owl:ObjectProperty* declaration.

Verify the existence of Object Properties <OWL> that link this Class <OWL>, corresponding to the instance of FeatureAssociationType, to the other Classes <OWL>, corresponding to the associated feature types with their *owl:ObjectProperty* declaration. Verify the annotation properties *rdfs:label* for human readable name and *skos:definition* for definition. Verify the domain and range of the Object Properties <OWL> with their *rdfs:domain* and *rdfs:range* declarations. Verify the cardinalities according to the cardinalities of the application schema, with their *rdfs:Restriction*, *owl:cardinality*, *owl:minCardinality*, and *owl:maxCardinality* declarations together with *owl:allValuesFrom*. Verify for bidirectional associations that Object Properties <OWL> are inverse of each other, and the *owl:inverseOf* declaration. Verify the annotation property *iso19150-2:associationName* of the Object Properties <OWL> for named associations.