Open Geospatial Consortium

Submission Date: 2015-10-22

Approval Date: 2017-03-23

Publication Date: 2017-09-15

External identifier of this OGC® document: <http://www.opengis.net/doc/IS/cis/1.1>

Internal reference number of this OGC® document: 09-146r6

Version: 1.1

Category: OGC® Implementation Standard

Editors: Peter Baumann, Eric Hirschorn, Joan Masó

OGC Coverage Implementation Schema

Warning

This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. This format is informative. The normative format is available at:

http://docs.opengeospatial.org/is/09-146r6/09-146r6.html

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Standard
Document subtype: Implementation Schema
Document stage: Approved
Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Contents

Page

v. Submitters 7
1. Scope 9
1.1 Overview 9
1.2 Compatibility 9
2. Conformance 11
3. References 15
4. Terms and definitions 16
4.1 Coverage 17
4.2 Regular grid 17
4.3 Irregular grid 17
4.4 Displaced grid 17
4.5 Mesh 17
4.6 Partition [of a coverage] 17
4.7 Sensor model 17
4.8 Transformation grid 17
5. Conventions 17
5.1 UML notation 17
5.2 Namespace prefix conventions 18
6. Class *coverage* 18
6.1 Overview 18
6.2 Coverages 19
6.3 CoverageFunction 21
6.4 Envelope and DomainSet 21
6.5 RangeType 25
6.6 RangeSet 27
6.7 Metadata 28
7. Class *grid-regular* 28
7.1 Overview 28
7.2 General grid coverages 28
8. Class *grid-irregular* 33
8.1 Overview 33
8.2 Irregular independent grid axes 33
8.3 Irregular correlated grid axes 33
9. Class *grid-transformation* 37
9.1 Overview 37
9.2 General 37
9.3 Transformation 37
9.4 SensorML grid 38
10. Class *discrete-pointcloud* 42
11. Class *discrete-mesh* 43
12. Class *gml-coverage* 43
12.1 Overview 43
12.2 Coverage representation 44
13. Class *json-coverage* 45
14. Class *rdf-coverage* 46
15. Class *other-format-coverage* 50
16. Class *multipart-coverage* 50
16.1 Overview 50

16.2 Root part 51
16.3 Further parts 51
17. Class *coverage-partitioning* 52
17.1 Overview 52
17.2 Partitioning 52
17.3 CRS and partition envelope constraints 55
17.4 Domain set constraints 56
17.5 Range type constraints 56
18. Class *container* 57
Annex A (normative) Abstract Test Suite 58
A.1 Conformance Test Class: *coverage* 58
A.2 Conformance Test Class: *grid-regular* 61
A.3 Conformance Test Class: *grid-irregular* 63
A.4 Conformance Test Class: *grid-transformation* 63
A.5 Conformance Test Class: *discrete-pointcloud* 64
A.6 Conformance Test Class: *discrete-mesh* 64
A.7 Conformance Test Class: *gml-coverage* 65
A.8 Conformance Test Class: *json-coverage* 66
A.9 Conformance Test Class: *rdf-coverage* 66
A.10 Conformance Test Class: *other-format-coverage* 67
A.11 Conformance Test Class: *multipart-coverage* 67
A.12 Conformance Test Class: *coverage-partitioning* 68
A.13 Conformance Test Class: *container* 71
Annex B (non-normative) Revision History 72
Annex C (non-normative) Complete `CIS::AbstractCoverage` UML diagram collection 73
Annex D (non-normative) Relation to Other Standards 75
D.1 Abstract Topic 6 / ISO 19123 75
D.2 GML 3.2.1 76
D.3 GML 3.3 77
D.4 SWE Common 77
D.5 Further Standards 77

Tables

Page

Table 1 Package URIs established in this standard 13
Table 2 Namespace mapping conventions 18
Table 3 `CIS::AbstractCoverage` data structure 20
Table 4 `CIS::EnvelopeByAxis` structure 22
Table 5 `CIS::AxisExtent` structure 24
Table 6 `CIS::RangeType` structure 25
Table 7 `CIS::InterpolationRestriction` structure 27
Table 8 `CIS::GeneralGridCoverage` structure 29
Table 9 `CIS::GeneralGrid` structure 30
Table 10 `CIS::Axis` structure 30
Table 11 `CIS::GridLimits` structure 30
Table 12 `CIS::IndexAxis` structure 31
Table 13 `CIS::RegularAxis` structure 31
Table 14 `CIS::IrregularAxis` structure 35
Table 15 `CIS::DisplacementAxisNest` structure 36
Table 16 `CIS::TransformationModel` structure 38
Table 17 `CIS::TransformationBySensorModel` structure 39
Table 18 `CIS::MultiPointCoverage` structure 42
Table 19 `CIS::CoverageByPartitioning` structure 54
Table 20 `CIS::PartitionSet` structure 54
Table 21 `CIS::Partition` structure 54
Table 22 `CIS::PositionValuePair` structure 54
Table 23 `CIS::RangeTypeComponentTranslation` structure 54
Table 24 Correspondence between ISO 19123 and CIS coverage types 76

Figures

Page

Figure 1: The `Coverage` class hierarchy as UML package diagram 13

Figure 2: `CIS::AbstractCoverage` structure (as per class *coverage*) 20

Figure 3: `CIS::DirectPosition` structure 21

Figure 4: `CIS::EnvelopeByAxis` structure 22

Figure 5: `CIS::GeneralGridCoverage` structure as per *grid-regular* 29

Figure 6: Some grid types: equidistant (far left), equidistant-skewed (left), irregular (right), displaced (far right) [2] 33

Figure 7: Sample grid combining regular and irregular axes (left) and irregular axes and "displaced" grids; time axis is drawn vertically 34

Figure 8: UML diagram of `CIS::GeneralGrid` structure as per *grid-irregular* 35

Figure 9: UML diagram of `CIS::GeneralGridCoverage` structure as per *grid-transformation* 38

Figure 10: UML diagram of `CIS::GeneralGridCoverage` structure as per *SensorML* 39

Figure 11: UML diagram of `CIS::MultiPointCoverage` structure 42

Figure 12: UML diagram of `CIS::CoverageByPartitioning` structure as per *coverage-partitioning* 53

Figure 13: UML diagram of `CIS::Object` structure as per *container* 57

Figure 14: Coverage types 73

Figure 15: Coverage structure 73

Figure 16: Grid coverages 74

i. Abstract

Coverages represent homogeneous collections of values located in space/time, such as spatio-temporal sensor, image, simulation, and statistics data. Common examples include 1-D timeseries, 2-D imagery, 3-D x/y/t image timeseries and x/y/z geophysical voxel models, as well as 4-D x/y/z/t climate and ocean data. Generally, coverages encompass multi-dimensional regular and irregular grids, point clouds, and general meshes.

This Coverage Implementation Schema (CIS) specifies the OGC coverage model by establishing a concrete, interoperable, conformance-testable coverage structure. It is based on the abstract concepts of OGC Abstract Topic 6 [1] (which is identical to ISO 19123) which specifies an abstract model which is not per se interoperable – in other words, many different and incompatible implementations of the abstract model are possible. CIS, on the other hand, is interoperable in the sense that coverages can be conformance tested, regardless of their data format encoding, down to the level of single "pixels" or "voxels."

Coverages can be encoded in any suitable format (such as GML, JSON, GeoTIFF, or NetCDF) and can be partitioned, e.g., for a time-interleaved representation. Coverages are independent from service definitions and, therefore, can be accessed through a variety of OGC services types, such as the Web Coverage Service (WCS) Standard [8]. The coverage structure can serve a wide range of coverage application domains, thereby contributing to harmonization and interoperability between and across these domains.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

Ogcdoc, coverage, gridded data, datacube, timeseries, sensor model, point cloud, mesh

iii. Preface

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

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iv. Submitting organizations

The following organizations submitted this Document to Open Geospatial Consortium Inc.:

- Jacobs University Bremen
- Envitia Ltd
- European Union Satellite Center

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	Organization
Peter Baumann	Jacobs University Bremen, rasdaman GmbH
Eric Hirschorn	KEYW Corp.
Joan Masó	CREAF

1. Scope

1.1 Overview

This document specifies the concrete, implementable, conformance-testable coverage structure to be used by OGC standards.

Coverages represent homogeneous collections of values located in space/time, such as spatio-temporal sensor, image, simulation, and statistics data. Common examples include 1-D timeseries, 2-D imagery, 3-D x/y/t image timeseries and x/y/z geophysical voxel models, as well as 4-D x/y/z/t climate and ocean data. Generally, coverages encompass multi-dimensional regular and irregular grids, point clouds, and general meshes.

This Coverage Implementation Schema (CIS) specifies the OGC coverage model by establishing a concrete, interoperable, conformance-testable coverage structure. It is based on the abstract concepts of OGC Abstract Topic 6 [1] (which is identical to ISO 19123) which specifies an abstract model which is not per se interoperable – in other words, many different and incompatible implementations of the abstract model are possible. CIS, on the other hand, is interoperable in the sense that coverages can be conformance tested, regardless of their data format encoding, down to the level of single "pixels" or "voxels."

Coverages can be encoded in any suitable data format, including formats as GML, JSON, GeoTIFF, and NetCDF. Further, coverages can be represented by a single document (stream or file) or by a hierarchically organized set of documents, each of which can be encoded individually – for example, the domain set, range type, and metadata may be encoded in easily parseable GML, JSON, or RDF while the range set is encoded in some compact binary format like NetCDF or JPEG2000. Such partitioning allows for coverages tiled in space, time, or mixed, thereby enabling mosaics, time-interleaved coverages, and efficiently subsettable datacubes.

Coverages are independent from service definitions and, therefore, can be accessed through a variety of OGC services types, such as the Web Coverage Service (WCS) Standard [8]. The coverage structure can serve a wide range of coverage application domains, thereby contributing to harmonization and interoperability between and across these domains.

1.2 Compatibility

1.2.1 OGC Abstract Topic 6 / ISO 19123

The OGC coverage model introduced with GMLCOV/CIS 1.0 and extended with CIS 1.1 is based on the abstract coverage specification of OGC Abstract Topic 6 (which is identical to ISO 19123) and harmonized with the GML coverage model [GML3.2.1] and the SWE sensor type description [SWE Common Data Model]. By way of normatively including GMLCOV/CIS 1.0 in CIS 1.1, every GMLCOV/CIS 1.0 coverage is a valid CIS 1.1 coverage. See Annex D.1 in CIS 1.1 for details.

1.2.2 GML

Like in GML, all coverage types in CIS 1.1 (as in GMLCOV/CIS 1.0) are derived from a common `AbstractCoverage` type. GMLCOV/CIS 1.0 is strictly derived from the corresponding GML type, so it is a GML Application Profile. CIS 1.1 is structurally equivalent, but embodies novel concepts which do not allow a formal derivation in all cases; further, modeling has been simplified over GML to make coverages easier to handle. Further,

having JSON and RDF next to GML had a design impact. As a consequence, CIS 1.1 formally speaking is not a GML Application Profile. See Annexes D.2 and D.3 for details.

1.2.3 SWE Common

The coverage `RangeType` component (see Clause 6) utilizes the SWE Common [4] `DataRecord`. Consequently, the semantics of sensor data acquired through SWE standards can be carried over into coverages without information loss. See also Annex D.4.

1.2.4 Extensions over previous version of this standard

This document is the successor version of *GML 3.2.1 Application Schema – Coverages* version 1.0.1 [5], abbreviated GMLCOV 1.0. This standard was renamed to *Coverage Implementation Schema* (CIS) in 2015 to remedy misunderstandings caused by the initial title, such as that only a GML encoding is defined (whereas, in fact, a format-independent implementable coverage model is established). Therefore, GMLCOV 1.0 is identical to CIS 1.0.

This document augments GMLCOV/CIS 1.0 as a backwards-compatible extension: any valid GMLCOV/CIS 1.0 coverage is also valid in CIS 1.1. This is accomplished through Requirement 1 which declares any valid GMLCOV/CIS 1.0 coverage to also be a valid CIS 1.1 coverage; on XML Schema level, the CIS 1.1 schema explicitly includes the GMLCOV/CIS 1.0 schema (although, given Requirement 1, this is not strictly necessary).

CIS 1.1 adds further coverage types over GMLCOV/CIS 1.0 – in particular, for more grids – and encoding options.

- CIS 1.1 adds comprehensive definitions for all possible types of irregular grids, which has been left unspecified in the previous version. As such, CIS 1.1 also incorporates and generalizes the grid coverage concepts established in GML 3.3 [3].

- CIS 1.1 extends the physical representation schema of gridded coverages by allowing an internal partitioning to accommodate different access patterns. One special case is time-interleaved where a coverage is represented by a list of pairs (timestamp, time slice). However, the partitioning schemes are not constrained and may include both spatial and temporal axes.

- CIS 1.1 complements the GML coverage representation with equivalent JSON and RDF representation.

To achieve this, CIS implements the following Change Requests on GMLCOV 1.0 [5]:

- Support for more general grid identifiers (with punctuation, national character sets, etc.) [OGC 15-086];

- Support for general non-regular grids [OGC 15-088];

- Clear regulation for interpolation methods associated with grid coverages, thereby also clarifying a long-standing confusion between discrete and continuous grid coverages [OGC 15-087];

- Introduction of `EnvelopeByAxis`, an envelope type which allows for a convenient handling of any type of coordinates together with a single CRS [OGC 15-093];

- Partitioned ("tiled") coverages, allowing – among others – "interleaved representations" of coverages [OGC 15-091] and datacubes tiled for efficient subsetting;

- Renaming from the confusing title "GML 3.2.1 Application Schema – Coverages" to "Coverage Information Schema" [OGC 15-094];

- Adding support for non-regularly gridded sensor models [OGC 15-092];

- Distinguish between grid dimension and the CRS dimension [OGC 15-089]; and

- Removal of a namespace ambiguity in `ReferenceableGridCoverage` [OGC 15-090] (resolved by introduction of `CIS::GeneralGridCoverage`).

Further, some GML 3.2.1 schema definitions whose generality complicates coverage understanding unnecessarily have been extracted and condensed into the pertaining CIS 1.1 GML schema. This remedies an often heard complaint about the complexity not of the coverage model, but the underlying GML. As a consequence, the GML encoding of CIS 1.1 is not a GML application schema any longer, but a compact, freestanding definition. Nevertheless, by way of integrating GMLCOV/CIS 1.0 it is possible for implementers to remain in the realm of a GML application schema.

Finally, as the new features make CIS substantially more expressive, not all implementers will want to support all functionality. Therefore, a further subdivision into separate requirements classes has been performed isolating, for example, discrete and grid coverages.

In summary, CIS 1.1 is a backwards compatible extension of GMLCOV/CIS 1.0, also merging in GML 3.3 grid types. Note that irregular grid types in both GMLCOV and GML in future may get deprecated in favor of the general grid type in CIS 1.1 which is more concise, better to analyze by applications, and support cases not addressed by the previous grid approaches.

2. Conformance

This standard defines: coverages.

Standardization target of this document are concrete **coverage instance documents**, as generated by some service and/or consumed by some client.

This document contains requirements for the following standardization target types (cf. Figure 1).

- The core class *coverage* (in red). This is the only abstract class – it establishes the basic framework, while the concrete conformance classes listed below define how concrete coverage instances can be built.
- The grid coverage classes (in green):
    - Class *grid-regular* establishes multi-dimensional unreferenced and regular referenced grids; in particular, `GridCoverage` and `RectifiedGrid-`

`Coverage` are provided here for backwards compatibility with version 1.0 of this standard;
    - Class *grid-irregular* establishes multi-dimensional irregular referenced grids; and
    - Class *grid-transformation* establishes multi-dimensional referenced grids defined by algorithmic transformations.
- The discrete coverage classes (in blue):
    - Class *discrete-pointcloud* establishes point clouds; and
    - Class *discrete-mesh* establishes general multi-dimensional meshes.
- The format encoding classes (in yellow):
    - Class *json-coverage* establishes JSON encoding of coverages;
    - Class *rdf-coverage* establishes RDF encoding of coverages;
    - Class *gml-coverage* establishes GML encoding of coverages;
    - Class *other-format-coverage* establishes further encodings of coverages; and
    - Class *multipart-coverage* establishes a multipart encoding of coverages.
- Class *coverage-partitioning* (in grey) establishes coverages composed from several sub-coverages.
- Class *container* (in white) establishes a general object capable of holding coverages and any other structure.

Note Classes *coverage*, *grid-regular*, *grid-irregular*, *grid-transformation*, *discrete-pointcloud*, and *discrete-mesh* together establish the conceptual coverage implementation model whereas classes *gml-coverage*, *json-coverage*, *rdf-coverage*, *other-format-coverage*, *multipart-coverage*, and *coverage-partitioning* establish encoding and representation schemes.

Figure 1 show the requirements class dependencies depicted as a UML package diagram: each package represents one class, the *depends-on* relationship represents the OGC requirements class dependency relationship.

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site[1].

In order to conform to this OGC™ CIS interface standard, a software implementation **shall** choose to implement:

- the core class *coverage* plus
- at least one of the discrete or grid coverage classes plus
- at least one of the encoding classes *json-coverage*, *gml-coverage* and *other-format-coverage*.

1 www.opengeospatial.org/cite

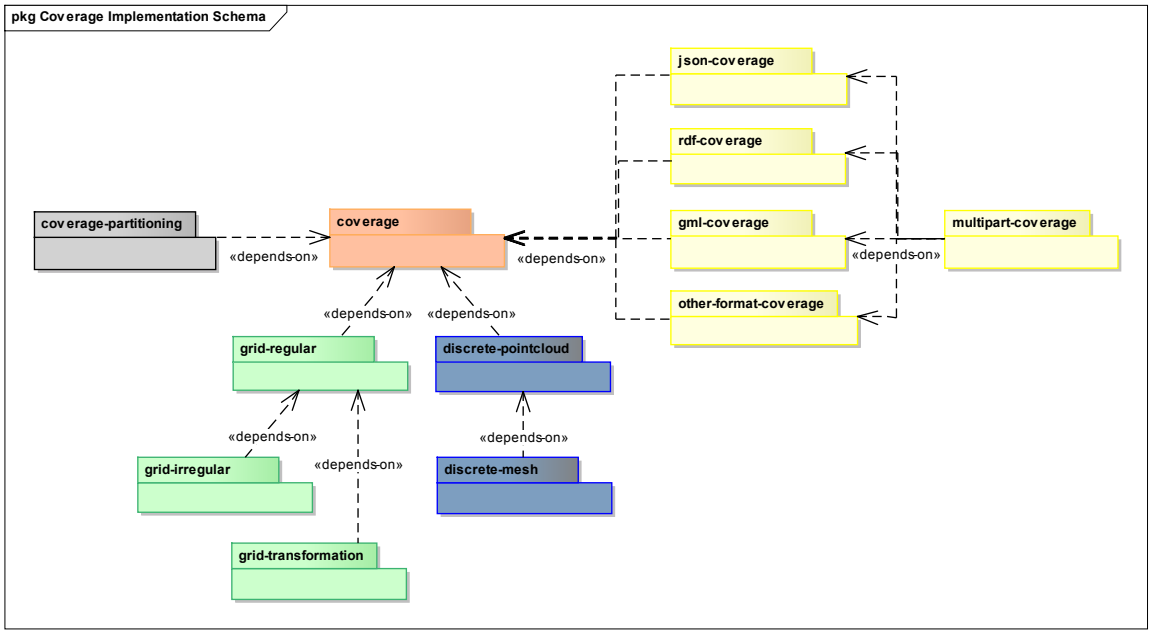

Figure 1: The `Coverage` class hierarchy as UML package diagram

Further classes can be implemented optionally as long as the dependencies set forth by this standard are respected.

Each requirements class in this standard corresponds to a single conformance class. Abstract conformance tests are listed in Annex A, whereby each test references back the requirement it assesses. Concrete implementations of these tests shall be exercised on any software artefact claiming to implement a conformance class of this standard.

Requirements and conformance tests are identified through URLs. Table 1 summarizes the respective URLs. As a rule, requirements and conformance class URLs defined in this document are relative to http://www.opengis.net/spec/CIS/1.1/.

All requirements-classes and conformance-classes described in this document are owned by the standard(s) identified.

Table 1 Package URIs established in this standard

Class	Description[2]
coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/coverage/req{req#}
discrete-pointcloud	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/discrete-pointcloud/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/discrete-

[2] {req#} denotes the requirement number in decimal notation, without leading zeroes.

	pointcloud/req{req#}
discrete-mesh	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/discrete-mesh/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/discrete-mesh/req{req#}
grid-regular	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/grid-regular/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/grid-regular/req{req#}
grid-irregular	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/grid-irregular/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/grid-irregular/req{req#}
grid-transformation	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/grid-transformation/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/grid-transformation/req{req#}
gml-coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/gml-coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/gml-coverage/req{req#}
json-coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/json-coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/json-coverage/req{req#}
rdf-coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/rdf-coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/rdf-coverage/req{req#}
other-format-coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/other-format-coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/other-format-coverage/req{req#}
multipart-coverage	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/multipart-coverage/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/multipart-coverage/req{req#}

coverage-partitioning	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/coverage-partitioning/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/coverage-partitioning/req{req#}
container	Requirements class URI: http://www.opengis.net/spec/CIS/1.1/req/container/req{req#} Conformance class URI: http://www.opengis.net/spec/CIS/1.1/conf/container/req{req#}

This OGC *Coverage Implementation Schema* consists of the UML diagrams and textual requirements classes established in this document as well as an external file bundle consisting of the corresponding XML Schema including Schematron constraints. The complete specification is identified by OGC URI http://www.opengis.net/spec/CIS/1.1, the document has OGC URI http://www.opengis.net/doc/AppSchema/CIS/1.1.

The complete standard is available at http://www.opengeospatial.net/standards/cis. The XML Schema is posted online at http://schemas.opengis.org/cis/1.1 as part of the OGC schema repository.

3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. The latest edition with the same major release number[3] as the document referred below applies.

[1] OGC: OGC 07-011, *Abstract Specification Topic 6: The Coverage Type and its Subtypes*, version 7.0 (identical to ISO 19123:2005), 2007

[2] OGC: OGC 07-036, *Geography Markup Language (GML) Encoding Standard*, version 3.2.1, 2007

[3] OGC: OGC 10-129r1, *OGC® Geography Markup Language (GML) – Extended schemas and encoding rules* (GML 3.3), version 3.3, 2012

[4] OGC: OGC 08-094, *OGC® SWE Common Data Model Encoding Standard*, version 2, 2011

[5] OGC: OGC 12-000, OGC® *SensorML: Model and XML Encoding Standard*, version 2, 2014

[6] OGC: OGC 09-146r2, *GML 3.2.1 Application Schema – Coverages*, version 1.0.1, 2012

[3] In the standards numbering scheme x.y.z, x is called major release number, y minor, and z corrigendum. Revisions of a standard where only the minor release number changes are backwards compatible. A major release number change signals possibly incompatible changes over the previous edition.

[7] OGC: OGC 16-083, *Coverage Implementation Schema – ReferenceableGridCoverage Extension*, version 1, 2017

[8] OGC: OGC 09-110r4, *Web Coverage Service (WCS) Core Interface Standard*, version 2, 2012

[9] OGC: OGC 13-102r2, *Name type specification – Time and index coordinate reference system definitions* (OGC Policy Document), version 1, 2014

[10] OGC: OGC 14-121, *Web Information Service (WIS)*, version 1 (unpublished)

[11] W3C: W3C Recommendation, *XML Path Language (XPath)*, version 2, 2007

[12] W3C: W3C Recommendation, *XML Linking Language (XLink)*, version 1, 2001

[13] W3C: W3C Working Draft, *The app: URI scheme*, 2013

[14] ISO/IEC: ISO/IEC 19757-3:2006 *Information technology – Document Schema Definition Languages (DSDL) – Part 3: Rule-based validation – Schematron*, 2006

[15] IETF: RFC 2183, 1997

[16] IETF: RFC 2387, 1998

[17] IETF: RFC 2392, 1998

[18] IETF: RFC 3986, 2005

[19] IETF: RFC 7159, The JavaScript Object Notation (JSON) Data Interchange Format. https://www.ietf.org/rfc/rfc7159.txt, 2014

[20] W3C: W3C JSON-LD 1.0, A JSON-based Serialization for Linked Data. http://www.w3.org/TR/json-ld/, 2014

[21] W3C: W3C JSON-LD 1.0 Processing Algorithms and API. http://www.w3.org/TR/json-ld-api, 2014

[22] W3C: W3C RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/, 2014

4. Terms and definitions

This document uses the specification terms defined in Subclause 5.3 of OGC Web Service Commons [OGC 06-121r9], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

For the purposes of this document, the terms and definitions given in the above references apply. In addition, the following terms and definitions apply.

4.1
Coverage
feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain, as defined in OGC Abstract Topic 6 [1]

4.2
Regular grid
grid whose grid lines have a constant distance along each grid axis

4.3
Irregular grid
Grid whose grid lines have individual distances along each grid axis

4.4
Displaced grid
grid whose direct positions are topologically aligned to a grid, but whose geometric positions can vary arbitrarily

4.5
Mesh
coverage consisting of a collection of curves, surfaces, or solids, respectively

4.6
Partition [of a coverage]
separately stored coverage acting, by being referenced in the coverage on hand, as one of its components

4.7
Sensor model
mathematical model for estimating geolocations from recorded sensor data such as digital imagery

4.8
Transformation grid
grid whose direct positions are given by some transformation algorithm not further specified in this standard

5. Conventions

5.1 UML notation

Diagrams using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of OGC Web Service Commons [OGC 06-121r9], adhere to the following conventions:

- UML elements having a package name of "`GML`" are those defined in the UML model of GML 3.2.1 [2];

- UML elements having a package name of "`SWE Common`" are those defined in the UML model of SWE Common 2.0 [4]; and

- UML elements not qualified with a package name, or with "`CIS`", are those defined in this standard.

Further, in any class where an attribute name or association role name is identical to a name in some superclass the local definition overrides the superclass definition.

5.2 Namespace prefix conventions

UML diagrams and XML code fragments adhere to the namespace conventions shown in Table 2. The namespace prefixes used in this document are **not** normative and are merely chosen for convenience. The namespaces to which the prefixes correspond are normative, however.

Whenever a data item from a CIS-external namespace is referenced this constitutes a **normative dependency** on the data structure imported together with all requirements defined in the namespace referenced.

Table 2 Namespace mapping conventions

UML prefix	GML prefix	Namespace URL	Description
CIS	cis	http://www.opengis.net/cis/1.1	Coverage Implementation Schema 1.1
CIS10	cis10	http://www.opengis.net/gmlcov/1.0	Coverage Implementation Schema 1.0
GML	gml	http://www.opengis.net/gml/3.2	GML 3.2.1
GML33	gml33	http://www.opengis.net/gml/3.3	GML 3.3
SWE Common	swe	http://www.opengis.net/swe/2.0	SWE Common 2.0
SML	sml	http://www.opengis.net/sensorml/2.0	SensorML 2.0

6. Class *coverage*

6.1 Overview

Class *coverage* lays the foundation for the coverage implementation schema. It is the core class of CIS, meaning that every coverage instance must conform to the requirements stated here. Class *coverage* does not allow creating coverage instances, but rather provides the fundament for the further classes (see next Clauses) which define various specializations of which instance documents can be created.

Note Clause 6 establishes a concrete conceptual model of a coverage which is independent from any particular encoding. While, in addition to UML, GML sometimes is used for establishing this (in particular when concepts and definitions from GML 3.2.1 [2] are used where a UML representation is not provided by that standard), CIS does not anticipate a GML encoding. Various encodings are established in Clauses 12 onwards.

This CIS 1.1 standard unifies OGC's coverage implementation model. It does so by extending CIS 1.0 (also known as GMLCOV 1.0) with further ways to model and represent coverages, and by integrating the GML 3.3 grid types.

Requirement 1 :
A coverage **shall** implement at least one of: this CIS 1.1 standard; the GMLCOV/CIS 1.0 standard; the GMLCOV/CIS 1.0 standard with the additional grid definitions provided with GML 3.3.

With the introduction of the CIS `GeneralGridCoverage` type and its unified modelling of all grid types, the gridded types of GMLCOV/CIS 1.0 [5], GML 3.3 [3], and ReferenceableGridCoverage Extension [7] may get deprecated in future.

6.2 Coverages

Coverages are represented by some binary or ASCII serialization, specified by some data (encoding) format. Coverage encoding is governed by specific standards. Some such encodings are defined as part of this standard in the classes *gml-coverage, json-coverage and rdf-coverage;* further formats are allowed through class *other-format-coverage*. In any case, for an instantiation of the general coverage definition given in this Clause 6 a concrete encoding needs to be available in the implementation on hand.

Requirement 2 :
A coverage instantiating class *coverage* **shall** implement at least one of *gml-coverage* , *json-coverage*, *rdf-coverage*, and *other-format-coverage*.

Note Not all encodings may be able to represent the full information making up a coverage, i.e.: not all encodings are informationally complete.

A coverage contains a `DomainSet` component describing the coverage's domain (the set of "direct positions", i.e., the locations for which values are stored in the coverage) and a `RangeSet` component containing these stored values (often referred to as "pixels", "voxels") of the coverage. Further, a coverage contains a `RangeType` element which describes the coverage's range set data structure (in the case of images usually called the "pixel data type"). Such a type often consists of one or more fields (also referred to as *bands* or *channels* or *variables*), however, much more general definitions are possible. For the description of the range value structure, SWE Common [OGC 08-094] `DataRecord` is used. The `metadata` component, finally, represents an extensible slot for metadata. The intended use is to hold any kind of application-specific metadata structures.

Note In this requirements class, *coverage*, a domain set invariably consists of a domain/range representation; requirements class *coverage-partitioning* (Clause 17) will add partitioning and position/value pair list as alternatives. This is why coverage subtype `CoverageByDomainAndRange` is introduced in Figure 2; while it may seem artificial in this requirements class, it will allow modelling the alternative representations in the future.

Requirement 3 :
A coverage instantiating class *coverage* **shall** conform with Figure 2, Figure 3, Table 3, and Table 7.

Note The `Envelope` item may be modelled differently in different encodings. In GML, for example, the `Envelope` element is enclosed in a `boundedBy` element.

The `id` attribute is the same as in GML and GMLCOV, but its type is extended from `NC-Name` to `string` to achieve a more human-readable style allowing for whitespace, special characters, globally unique naming schemes, etc.

Coverages make heavy use of n-dimensional coordinates in a space that may be made up from spatial and/or temporal and/or "abstract" (i.e., non-spatio/temporal) axes. For

representing direct positions of coverages, such n-dimensional coordinates are modelled through type `CIS::DirectPosition`. Each coordinate component is of the general type `anySimpleType` (in analogy to XML Schema) as it must accommodate data types as diverse as numbers (such as 1.23 degrees), dates (such as "2016-03-08"), and abstract categorical values (such as "orange", "apple"). The order of the coordinates is given by the axis order of the CRS defined in the context in which the direct position is used.

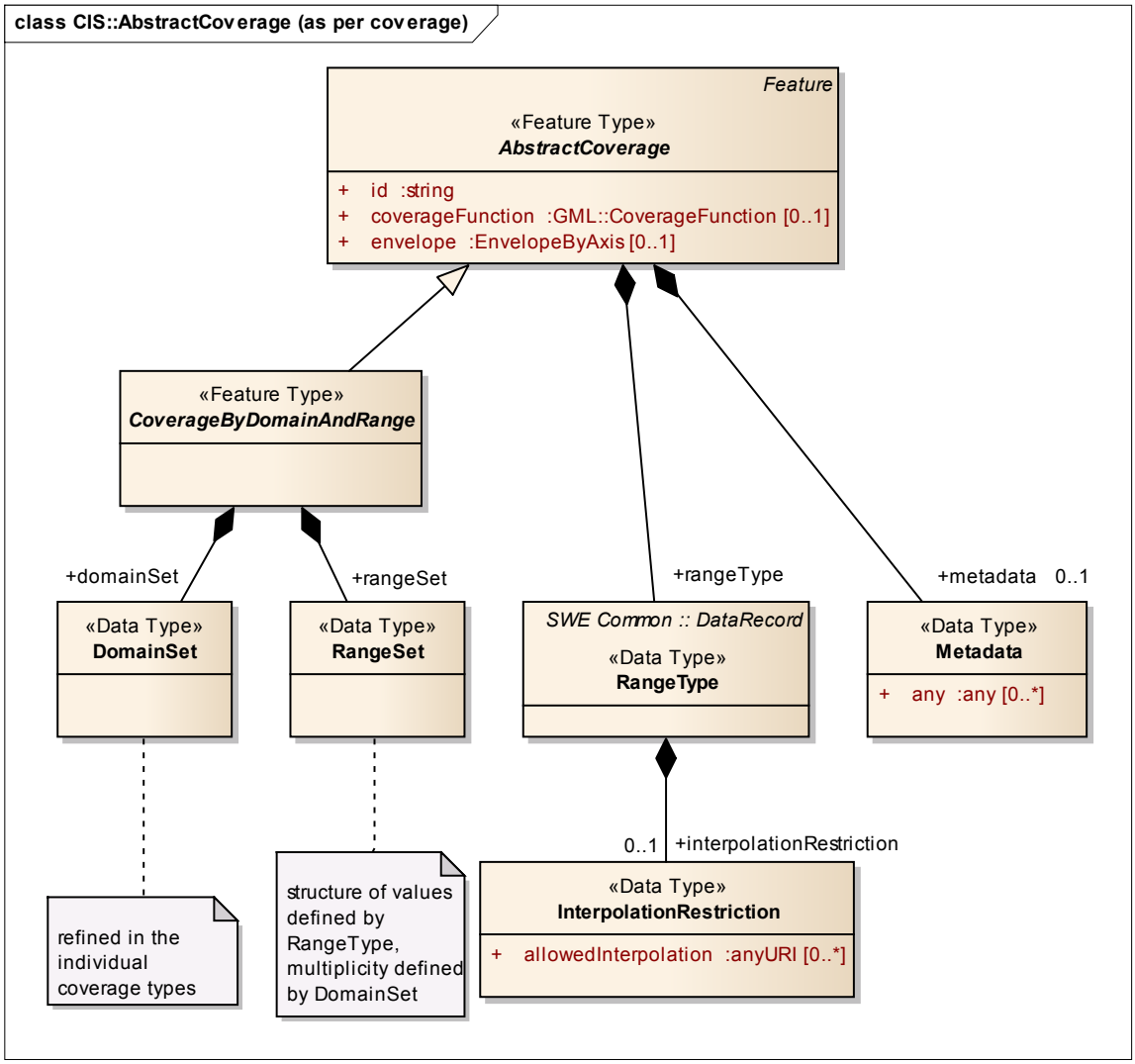

Figure 2: `CIS::AbstractCoverage` structure (as per class *coverage*)

Table 3 `CIS::AbstractCoverage` data structure

Name	Definition	Data type	Multiplicity
`id`	Identifier of the coverage	`string`	One (mandatory)
`coverage-Function`	Function describing how range values at the coverage's direct positions can be computed, as specified in GML 3.2.1 [2]	`GML:: Coverage-Function`	Zero or one (optional)

	Subclause 19.3.11		
envelope	Minimum bounding box of the coverage, as specified in GML 3.2.1 [2] Subclause 10.1.4.6	CIS:: Envelope-ByAxis	One (mandatory)
domainSet	Definition of coverage domain, i.e., its set of direct positions	CIS:: DomainSet	One (mandatory)
rangeSet	Coverage range values, each one associated with a direct position	CIS:: RangeSet	One (mandatory)
rangeType	Structure definition of the coverage range values, as specified in SWE Common 2.0 [4] Clause 7 and 8	SWE Commmon ::DataRecord	One (mandatory)
metadata	Application specific metadata, allowing for individual extensions	CIS:: Extension	Zero or one (optional)

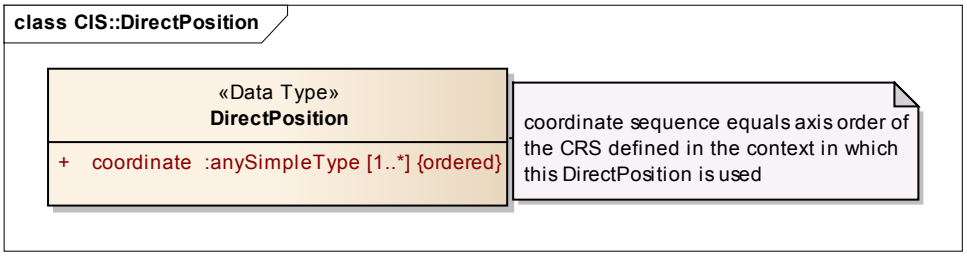

Figure 3: `CIS::DirectPosition` structure

6.3 CoverageFunction

The `coverageFunction` component is identical in its syntax and meaning to the corresponding element defined in GML 3.2.1 [2] Subclause 19.3.11. It describes the mapping function from the domain to the range of the coverage. For a grid coverage, it specifies the serialization of the multi-dimensional grid in the range set.

Note 1 This becomes particularly relevant when defining encoding formats, such as GML or JSON.

Note 2 For the reader's convenience, the default is copied from GML 3.2.1: If the `gml:coverageFunction` property is omitted for a gridded coverage (including rectified gridded coverages) the `gml:startPoint` is assumed to be the value of the `gml:low` property in the `gml:Grid` geometry, and the `gml:sequenceRule` is assumed to be linear and the `gml:axisOrder` property is assumed to be "+1 +2".

6.4 Envelope and DomainSet

The domain set determines the exact locations of a coverage overall and its set of direct positions. The domain set is defined through an ordered list of axes whose lower and upper bounds establish the extent along each axis. The axis sequence and their meaning is defined by the CRS which is given by a `GML::SRSReferenceGroup` consisting of the URI identifying the CRS. This domain set CRS is called the coverage's *Native CRS*.

Additionally, some redundant information is present in the domain set for efficiency reasons: the number of dimensions, axis labels, and UoM (Unit of Measure) labels simplify parsing the coverage as the parser does not have to retrieve the CRS definition, such as from the OGC CRS resolver at http://www.opengis.net/def/crs and http://www.opengis.net/def/crs-compound.

The optional `CIS::Envelope` component helps applications in gaining a quick overview on the coverage's location. The location information does not need to use the same CRS as the domain set, therefore the bounding box may not always be the minimal.

Note Particularly in presence of displaced axes, transformation axes, and discrete coverages the domain set can quickly become hard to oversee.

Requirement 4 :
If present, the `envelope` of a coverage instantiating class *coverage* **shall** consist of a `CIS::EnvelopeByAxis` element conforming to Figure 4, Table 4, and Table 5.

Note As in GML 3.2.1, the `envelope` of a coverage, if present, encloses the entire coverage instance; it does not have to be minimal, though (for example, if the envelope is in a different – possibly easier to evaluate – CRS such as WGS84 a minimal bounding box normally cannot be expressed exactly).

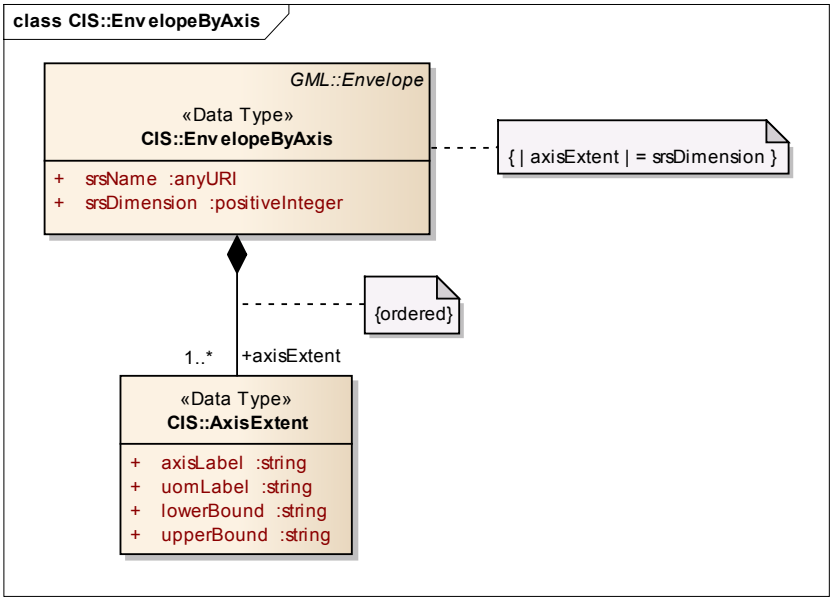

Figure 4: `CIS::EnvelopeByAxis` structure

Table 4 `CIS::EnvelopeByAxis` structure

Name	Definition	Data type	Multiplicity
`srsName`	URL identifying the CRS of the coordinates in this coverage	`anyURI`	One (mandatory)
`srsDimension`	Dimension (number of axes) of the grid	`positive-Integer`	One (mandatory)
`axisExtent`	Sequence of extents of the grid along a specific axis, exactly one	`CIS::`	One or more

	for each axis defined in the CRS referenced in `srsName`	`AxisExtent`	(mandatory)

As the envelope coordinate values refer to a CRS and its axes it is necessary to link to those references. To this end, a CRS identifier is provided through a URL referencing its definition. Axes used by the coverage are identified by their position in the (ordered) list of axes given in the CRS. In the `axisLabels` string, alias names are established for the axes used in the `axisExtent` components, matched with the axis through their position in the sequence. Additionally, the units of measure are indicated for each axis.

Requirement 5 :
In the `envelope` of a coverage instantiating class *coverage*, if present, the value of `srsName` **shall** be a URL which points to a CRS definition which fulfils the following conditions:
- `srsDimension` equals the dimension of the CRS (i.e., the number of axes);
- the number of `axisExtent` items is equal to `srsDimension`;
- the each `axis` in `envelope` there is exactly one matching CRS axis with `axisLabel` = CRS `axisAbbrev` for this axis and `uomLabel` = unit of measure for this axis;
- in each `axisExtent` the `uomLabel` value equals the unit of measure of the corresponding CRS axis.

Note This definition relaxes the `axisLabels` handling as per GMLCOV/CIS 1.0 where the identifiers referenced in `axisLabels` had to be identical to the corresponding `axisAbbrev` value in the CRS definition. In CIS 1.1, coverage `axisLabels` and CRS `axisAbbrev` are decoupled so that there is no such dependency any longer. This definition is backwards compatible, i.e., coverages can continue to use CRS `axisAbbrev` values; note, though, that `axisAbbrev` values in subsequent versions of a CRS may change without notice, so the correspondence may get lost over time.

Example The following envelope, encoded in XML, utilizes EPSG 4326 with two axis labels, “Lat” and “Long.” These labels correspond to the CRS axis abbreviations of EPSG v8.5, but not to EPSG v8.9.2 where the axis abbreviation for Longitude has been changed to “Lon”. In CIS 1.1, this is not an issue because (i) CRS axes are ordered and (ii) values in `axisLabels` are matched by position, so axis label “Long” is unambiguously associated with CRS axis abbreviated as “Lon.”

```
<Envelope srsName="http://www.opengis.net/def/crs/EPSG/0/4326"
   axisLabels="Lat Long" srsDimension="2">
   <AxisExtent axisLabel="Lat" uomLabel="deg" lowerBound="-80" upperBound="-70"/>
   <AxisExtent axisLabel="Long" uomLabel="deg" lowerBound="0" upperBound="10"/>
</Envelope>
```

Actually, a coverage is completely free to use any identifier whereby the syntax of identifiers is given by the encoding used; in GML, for example, it is `NCName`. The following version is semantically identical to the above:

```
<Envelope srsName="http://www.opengis.net/def/crs/EPSG/0/4326"
   axisLabels="a1 a2" srsDimension="2">
   <AxisExtent axisLabel="a1" uomLabel="deg" lowerBound="-80" upperBound="-70"/>
   <AxisExtent axisLabel="a2" uomLabel="deg" lowerBound="0" upperBound="10"/>
</Envelope>
```

This demonstrates that an axis label may be identical to the `axisAbbrev` value in CRS definition, but does not have to.

Table 5 `CIS::AxisExtent` structure

Name	Definition	Data type	Multiplicity
`axisLabel`	Shorthand axis identifier with scope given by the coverage document	`string`	One (mandatory)
`uomLabel`	Shorthand identifier of the Unit of Measure used on this axis (as indicated in the CRS definition for this axis)	`string`	One (mandatory)
`lowerBound`	Lowest coordinate along this axis	`string`	One (mandatory)
`upperBound`	Highest coordinate along this axis	`string`	One (mandatory)

Note At the time of this standard's writing the widely used EPSG database – which forms the basis also for the OGC CRS resolver, http://www.opengis.net/def/crs/ - does not have unit symbols, only non-normative names. Therefore, in general it is currently not possible to automatically deduce the unit of measure of an axis. Instead is recommended as a Best Practice to use the unit strings as defined by UCUM (http://unitsofmeasure.org). All examples used in this standard utilize UCUM.

Requirement 6 :
For each `axisExtent` in the `EnvelopeByAxis` element of a coverage the `lowerBound` **shall** be less than or equal to the `upperBound`.

Requirement 7 :
In a coverage instantiating class *coverage*, the extent of `CIS::Envelope` (if present) **shall** enclose `CIS::DomainSet` along all dimensions.

Note In other words: the bounding box given by the domain set must be fully enclosed in the bounding box as defined in the envelope. This requirement follows already from GML 3.2.1 Subclause 9.3.1, but is repeated here as GML does not have a uniform treatment of spatial, temporal, and other dimensions.

While the envelope can be approximate, the domain set is exact in its boundaries.

Requirement 8 :
In a coverage instantiating class *coverage*, for all axes in a `CIS::GeneralGrid` where axis coordinates of direct positions are given explicitly, the lowest and highest value of these coordinates **shall** be equal to the `lowerBound` and `upperBound` value, respectively.

Coverages are assumed to have a 1:1 correlation between the axis names given in `axisLabels` and `gridLabels`, i.e.: they shall relate pairwise, given by their sequence position. For example, `axisLabels`= "Lat Long h date" and `gridLabels`={i j k l} implies a correspondence of `Lat` with `i`, `Long` with `j`, `h` with `k`, and `date` with `l`. On coverage instance level, though, this cannot be conformance tested, therefore this is not a formal requirement.

6.5 RangeType

6.5.1 Overview

The `RangeType` component adds a structure description and technical metadata required for an appropriate (however, application independent) understanding of a coverage. For this structure description, the SWE Common `DataRecord` is used. Optionally, interpolation directives can be added.

Requirement 9 :
In a coverage instantiating class *coverage*, the RangeType component **shall** have a structure as given in Table 6.

Table 6 `CIS::RangeType` structure

Name	Definition	Data type	Multiplicity
`dataRecord`	Description of the common data type of all range values	`SWE Common :: DataRecord`	One (mandatory)
`interpolation Restriction`	Constraints on the interpolation methods meaningfully applicable to this coverage	`CIS::Inter-polation-Restriction`	Zero or one (optional)

6.5.2 Range data type specification

Specification of the common data type all range values share is done through the `Data-Record` part of the coverage's `RangeType` component. Atomic data types available for range values are those given by the SWE Common data type `AbstractSimpleCompon-ent`. As a range structure contains only structure definitions, but not the values themselves (these sit in the coverage range set component), the optional `AbstractSimpleComponent` component `value` is suppressed in coverages.

Requirement 10 :
In a coverage instantiating class *coverage*, for all `SWE Common :: AbstractSimpleComponent` items in a range type structure, instance multiplicity of the `value` component **shall** be zero.

Note Following [4], omission of the `value` component implies that in a `DataArray` there is no `encoding` component either.

Range values can be structured as records or arrays. Both structuring principles can be nested (and mixed) to any depth for a concrete coverage range structure definition.

Requirement 11 :
In a coverage instantiating class *coverage*, for all SWE Common `AbstractDataComponent` items in a coverage range type structure, the concrete subtype used **shall** be one of `DataRecord` and `DataArray`.

Note 1 These subtypes are not allowed: `DataChoice`, `Vector`, `Matrix`.

Note 2 As array-valued ranges (i.e., nested arrays) can always be represented in a "flat" way by a single-level array with extra dimension(s) the use of such array-valued range types is discouraged as it adds complexity without additional value. Effectively, only `DataRecord` should be used.

Within a `DataRecord` contained in a concrete range structure, each of its record components is locally uniquely identified by the record component's `field` attribute, in accordance with the "soft-typing" property introduced by SWE Common.

Example The following XML fragment represents a valid range structure; it models the red, green, and blue channel of a Landsat scene. Pixels are defined as unsigned 8-bit quantities where 0 and 255 denote null values, representing radiance values measured in W/cm^2:

```
<RangeType>
  <swe:DataRecord>
      <swe:field name="red">
        <swe:Quantity definition="http://opengis.net/def/property/OGC/0/Radiance">
          <swe:uom code="W/cm2"/>
        </swe:Quantity>
      </swe:field>
      <swe:field name="green">
        <swe:Quantity definition="http://opengis.net/def/property/OGC/0/Radiance">
          <swe:uom code="W/cm2"/>
        </swe:Quantity>
      </swe:field>
      <swe:field name="blue">
        <swe:Quantity definition="http://opengis.net/def/property/OGC/0/Radiance">
          <swe:uom code="W/cm2"/>
        </swe:Quantity>
      </swe:field>
  </swe:DataRecord>
</RangeType>
```

Note While SWE Common is confined to XML, a coverage can be encoded in any suitable format. Therefore, the GML examples are of informative nature only, but not constraining to this format.

6.5.3 Interpolation and continuous coverages

A continuous (grid) coverage as defined in Abstract Topic 6 [1] has values not only at the direct positions themselves, but also in between those positions – in other words, it is valid to apply interpolation to obtain values between direct positions.

Technically, a continuous grid coverage consists of a grid coverage with an interpolation method associated. Notably, often there is more than one interpolation method which can be applied meaningfully.

Example A satellite image can be interpolated by *nearest neighbor*, *linear*, *quadratic*, and several more methods. A land use map, on the other hand, can only be interpolated using *nearest-neighbor*.

In the `CIS::allowedInterpolation` element an application can specify which interpolation methods are meaningful (hence, allowed) on the coverage on hand. Without such an element, any interpolation is admissible on the coverage.

Table 7 `CIS::InterpolationRestriction` structure

Name	Definition	Data type	Multiplicity
`allowed-Interpolation`	Constraint on the interpolation methods meaningfully applicable to this coverage	`anyURI`	Zero or more (optional)

The `InterpolationRestriction` element is meant to be interpreted as follows:

- If no `interpolationRestriction` element is present, then any interpolation method is applicable to the coverage on hand; or
- In presence of an `interpolationRestriction` element, only those interpolation methods may be meaningfully applied whose identifiers appear in an `allowedInterpolation` element; in case of an empty list this means that no interpolation is applicable at all.

Note As selection of a particular interpolation method is at the discretion of the application processing a coverage, the interpolation behavior is not testable on the level of coverage definition and, therefore, cannot be put into a formal, testable requirement.

Example In a XML encoding, the following constitutes a valid interpolation restriction (using OGC-defined URLs for identifying interpolation methods as defined in ISO 19123) indicating that nearest-neighbor and linear interpolation are admissible on the coverage on hand:

```
<InterpolationRestriction>
   <AllowedInterpolation>
     http://www.opengis.net/def/interpolation/OGC/1/nearest-neighbor
   </AllowedInterpolation>
   <AllowedInterpolation>
     http://www.opengis.net/def/interpolation/OGC/1/linear
   </AllowedInterpolation>
</InterpolationRestriction>
```

6.6 RangeSet

The range set contains the actual values, each of which is associated with one direct position as defined in the domain set.

Both `DomainSet` and `RangeType` describe the coverage values given in the `RangeSet`. Hence, consistency must be enforced between them. The pertaining requirements are listed below.

There must be a 1:1 correspondence between direct positions and range values. Neither duplicates nor values omitted are allowed.

Note For range values not known null values can be used.

Requirement 12 :
In a coverage instantiating class *coverage*, for each coordinate position contained in the domain set description of a coverage there **shall** exist exactly one range value in the coverage's range set.

Note For each of the coverage subtypes the number of direct positions in the domain set is determined individually, as this varies greatly across the types.

Note This applies to `CIS::IrregularAxis`, the `CIS::Displacement`, and the `CIS::TransformationModel`.

Requirement 13 :
In a coverage instantiating class *coverage*, all range values contained in the range set of this coverage **shall** be consistent with the structure description provided in its range type.

The data type of all range values is the same, it is given by the range type defined through a `SWE::DataRecord`. In particular, in a coverage instantiating class *coverage*, atomic values inside a composite value shall be listed exactly in the same sequence as the range type components whereby arrays are treated like records, serialized in their natural ascending sequence.

Note This last sentence is not conformance testable on this standardization target (coverage instance), therefore not expressed as a requirement. However, at service level this requirement may be testable indeed.

6.7 Metadata

The `metadata` component is a carrier for any kind of application dependent metadata. Hence, no requirements are imposed here.

Note Implementations may impose restrictions on metadata stored (such as their sheer volume).

7. Class *grid-regular*

7.1 Overview

This class *grid-regular* establishes coverages with regular grid types, both referenced and non-referenced. For backwards compatibility, `CIS10::GridCoverage` and `CIS10::RectifiedGridCoverage` are kept from GMLCOV/CIS 1.0 [5]; additionally, a new structure `CIS::GeneralGridCoverage` is added.

7.2 General grid coverages

`CIS::GeneralGridCoverage` lays foundation for the modelling of all possible grid types in CIS. While in class *grid-regular* only regular grids are defined, classes *grid-irregular* and *grid-transformation* extend this framework successively with additional grid types.

Note Skewed and rotated grids are not modelled explicitly; they can be represented by making the grid's CRS a concatenation of any given CRS with an Engineering CRS describing, e.g., any affine transformation of the original grid.

Requirement 14 :
A coverage instantiating class *grid-regular* **shall** conform with class *coverage*.

Requirement 15 :
A coverage of type `CIS::GeneralGridCoverage` **shall** have a structure as given by Figure 5, Table 8, Table 9, Table 10, and Table 13.

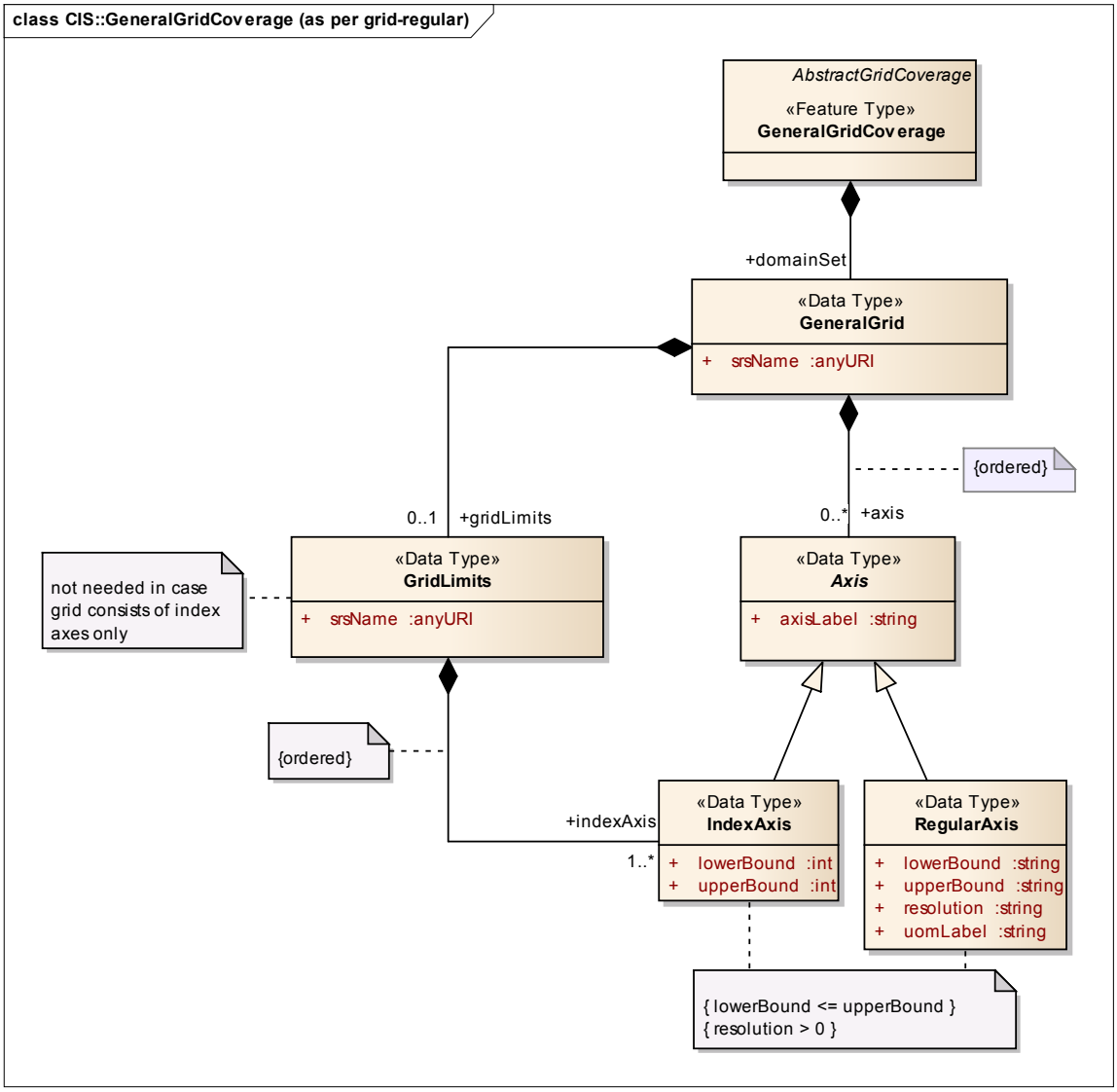

Figure 5: `CIS::GeneralGridCoverage` structure as per *grid-regular*

Table 8 `CIS::GeneralGridCoverage` structure

Name	Definition	Data type	Multiplicity
`DomainSet`	grid defining the coverage's direct positions, specializing the general `DomainSet` of `CIS::AbstractCoverage`	`CIS::General-Grid`	One (mandatory)
(all other components inherited unchanged from `CIS::AbstractCoverage`)			

7.2.1 General Grid

7.2.1.1 Overview

Gridded coverages have a grid as their domain set describing the direct positions in multi-dimensional coordinate space, depending on the type of grid. In this class *grid-regular*, simple equidistant grids are established.

Requirement 16 :
A `CIS::GeneralGrid` **shall** have a structure as given by Figure 5, Table 9, Table 10, Table 11, Table 12, and Table 13.

Table 9 `CIS::GeneralGrid` structure

Name	Definition	Data type	Multiplicity
`srsName`	URL identifying the CRS of the coordinates in this coverage	`anyURI`	One (mandatory)
`axis`	grid axis identifiers, all distinct within a grid	`CIS::Axis`	One or more (mandatory)

Note Such a General Grid does not contain global offset vectors because these are available with the axis subtypes where appropriate. It does not contain a rotation vector as this can be modelled by concatenating the CRS with an appropriate Engineering CRS for general affine transformations.

A `CIS::Axis` item contains information about a particular axis: its axis name, unit of measure along the axis, and further information depending on the axis type.

Table 10 `CIS::Axis` structure

Name	Definition	Data type	Multiplicity
`axisLabel`	identifier of this axis	`string`	One (mandatory)

Except for an index axis (which is a bare array grid), coordinates in an axis are expressed in some geodetic CRS or similar. Correspondingly, the grid limits in the `CIS::Axis` structure contains information about the grid boundaries in the coverage's CRS.

In addition, the limits of the underlying array are given by the `CIS::gridLimits` component. This structure is optional because it is not needed when all coverage axes are of type `CIS::indexAxis`, in which case the boundary information is redundant.

Table 11 `CIS::GridLimits` structure

Name	Definition	Data type	Multiplicity
`srsName`	URL identifying the Index CRS of the domain set grid array in this coverage	`anyURI`	One (mandatory)

indexAxis	all axes of the Index CRS referenced in srsName, in proper sequence	CIS::IndexAxis	One or more (mandatory)

Example The Index CRS for a 2-D grid is http://www.opengis.net/def/crs/OGC/0/Index2D. It defines axis names *i* and *j*.

In this *regular-grid* class, two subtypes of axes are defined, characterized by their axis type and CRS used: index and regular axis.

7.2.1.2 Index Axis

Axis type CIS::IndexAxis requires an Index CRS as its CRS, as defined in the OGC Name Type Specification for Index CRSs [9]. An Index CRS allows only integer coordinates with spacing ("resolution") of 1, hence resembling Cartesian coordinates; therefore, there is no resolution value.

Table 12 CIS::IndexAxis structure

Name	Definition	Data type	Multiplicity
lowerBound	Lowest array coordinate along this axis	integer	One (mandatory)
upperBound	Highest array coordinate along this axis	integer	One (mandatory)

Note A grid coverage containing exclusively axes of type IndexAxis technically corresponds to a CIS10::GridCoverage, however, with a slightly differing schema.

7.2.1.3 Regular Axis

Axis type CIS::RegularAxis has no restriction on the CRS used; as it is regularly spaced it contains the common distance, i.e.: resolution, as a part of the axis definition.

Table 13 CIS::RegularAxis structure

Name	Definition	Data type	Multiplicity
lowerBound	Lowest coordinate along this grid axis	string	One (mandatory)
upperBound	Highest coordinate along this axis	string	One (mandatory)
resolution	grid resolution along this axis	string	One (mandatory)
uomLabel	unit of measure in which values along this axis are expressed	string	One (mandatory)

Note The type is string to accommodate any potential resolution specification, such as "100" for degrees or meters, "2015-07-30T23Z" for a 1-hour duration in Gregorian calendar, and potential future calendar types.

Requirement 17
In a coverage using the *grid-regular* scheme, the `resolution` value in a `CIS::RegularAxis` **shall** be a nonzero, positive value expressed in the units of measure of this axis as defined in the CRS identified in the `srsName` item of the `envelope`.

The set of direct positions in a grid is given by the number of grid points. In the simplest case of a grid with index axes only, this is the product of the axis extents. For more complex grid types this computation gets more involved.

For some `CIS::GeneralGrid` g, let n_x be the number of `CIS::IndexAxis` elements, n_r the number of `CIS::RegularAxis` elements, n_i the number of `CIS::Irregular` axis elements, n_d the number of `CIS::DisplacementAxisNest` elements associated with any of the `CIS::DisplacementAxis` items, and n_t be the number of `CIS::TransformationModel` elements associated with any of the `CIS::TransformationAxis` items.

Let the following positive integer numbers be defined for the number of direct position coordinates along axes or axis combinations:

- IndexAxis:
  px_a := $g.a$. `upperBound` − $g.a$.`lowerBound` + 1 for $a \in g$.`CIS::IndexAxis`;

- RegularAxis:
  pr_a := $\lfloor (g.a$. `upperBound`−$g.a$.`lowerBound+1`)/resolution$\rfloor$ (i.e., rounded down) for $a \in g$.`CIS::RegularAxis`;

- IrregularAxis:
  pi_a := card($g.a$. `directPositions`) for $a \in g$.`CIS::IrregularAxis`;

- DisplacementAxis:
  pd_d := card(g . d . `directPositions`) for $d \in g$.`displacement`;

- TransformationAxis:
  pt_m := card($f(g)$) for $m \in g$.`model` where f is a function on g delivering all direct positions (such as a sensor model);

Then, the number n_p of direct positions in g is given by the product of all the above items:

$$n_p := \prod_a px_a * \prod_a pr_a * \prod_a pi_a * \prod_d pd_d * \prod_m pt_m$$

where a partial product is 1 if no such item exists.

Requirement 18 :
The `RangeSet` of a coverage containing the above `CIS::GeneralGrid` g **shall** contain exactly n_p value items.

8. Class *grid-irregular*

8.1 Overview

This class *grid-irregular* adds coverages of irregular axis types to the `GeneralGridCoverage` introduced with class *grid-regular*. Figure 6 shows some common 2-D grid types tractable with class *grid-irregular*.

The concept builds upon axis types with individual characteristics, such as non-referenced, referenced-equidistant, referenced-nonequidistant, etc. from which CRSs and, hence, grids are assembled. All axis types can be combined freely in a grid. This model includes the GML 3.3 [3] grid types `ReferenceableGridByVector` and `ReferenceableGridByArray` as special cases and allows representing all grid types.

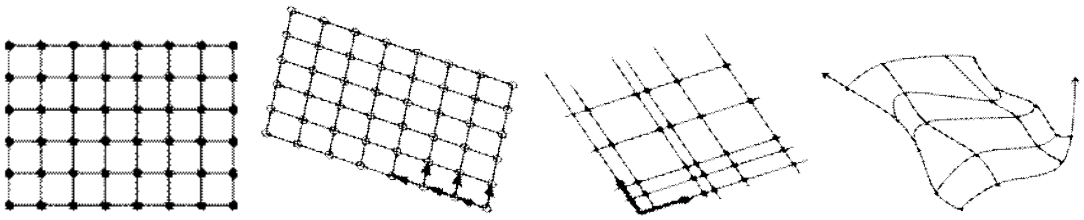

Figure 6: Some grid types: equidistant (far left), equidistant-skewed (left), irregular (right), displaced (far right) [2]

Note Skewed and rotated grids such as shown in Figure 6 can be represented by making the grid's CRS a concatenation of any given CRS with an Engineering CRS describing, e.g., any affine transformation of the original grid.

8.2 Irregular independent grid axes

The first extension over regular axes consists of irregular axes where spacing along an axis can have any positive increment. Graphically, this can be represented by straight lines (but consider that existence of values between direct positions is possibly guided by interpolation restrictions). Such axes are modelled by type `CIS::IrregularAxis`.

Example This allows grid representations like swath data, but also mixes like *Lat/Long/t* datacubes over orthorectified imagery where *Lat* and *Long* are equidistant while acquisition time, hence *t*, is irregular. This is schematically shown in Figure 7 (left).

8.3 Irregular correlated grid axes

The second extension consists of building axis groups, informally called "nests", within which the coordinates of direct positions are not tied to the crossing points of "straight" grid lines. Instead, coordinates can vary freely; however, the topological neighborhood relationship is retained. This leads to "displaced grids" as shown in Figure 6 far right (but consider that the curves drawn suggest a particular interpolation scheme which may or may not be allowed as per interpolation restrictions).

Not all axes in a grid need to participate in a nest, and a grid may contain several disjoint nests (although this case is unlikely).

Example A grid displaced in *Lat/Long* may also contain a time axis not involved in this nest. This situation is shown in Figure 7 where the vertical axis is not involved in the displacement field. Further, a grid may contain several nests, which, however, need to be disjoint in their participating axis sets.

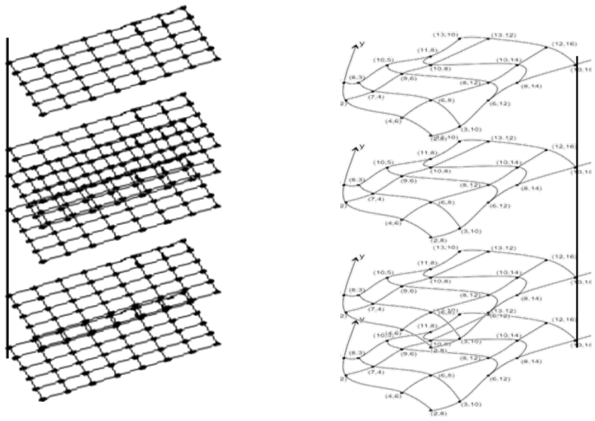

Figure 7: Sample grid combining regular and irregular axes (left) and irregular axes and "displaced" grids; time axis is drawn vertically

Class *grid-irregular* extends class *grid-regular* with further axis types, hence it requires implementation of that class.

Requirement 19 :
A coverage instantantiating class *grid-irregular* **shall** conform with class *grid-regular*.

The new axis types require storage of additional information. While for a regular axis a single resolution value is sufficient per axis, irregular grids require a sequence of direct positions along the axis (axis type `CIS::IrregularAxis`).

Nests require an n-D tensor, i.e., an array which stores the coordinates of each direct position for the axes participating in the nest (cf. `CIS::DisplacementAxisNest`).

Requirement 20 :
A coverage using the *grid-irregular* scheme **shall** conform with Figure 8, Table 14, and Table 15.

An irregular axis abandons the equidistant spacing of a regular axis. Therefore, all direct positions along such an axis must be enumerated explicitly which is achieved by replacing the lower bound / resolution / upper bound scheme by an ordered list of direct positions.

Note GML 3.3 type `ReferenceableGridByVector` resembles the special case that all axes are irregular, but independent. In CIS, this is modelled through a `CIS::GeneralGrid` that has only axes of type `CIS::IrregularAxis`.

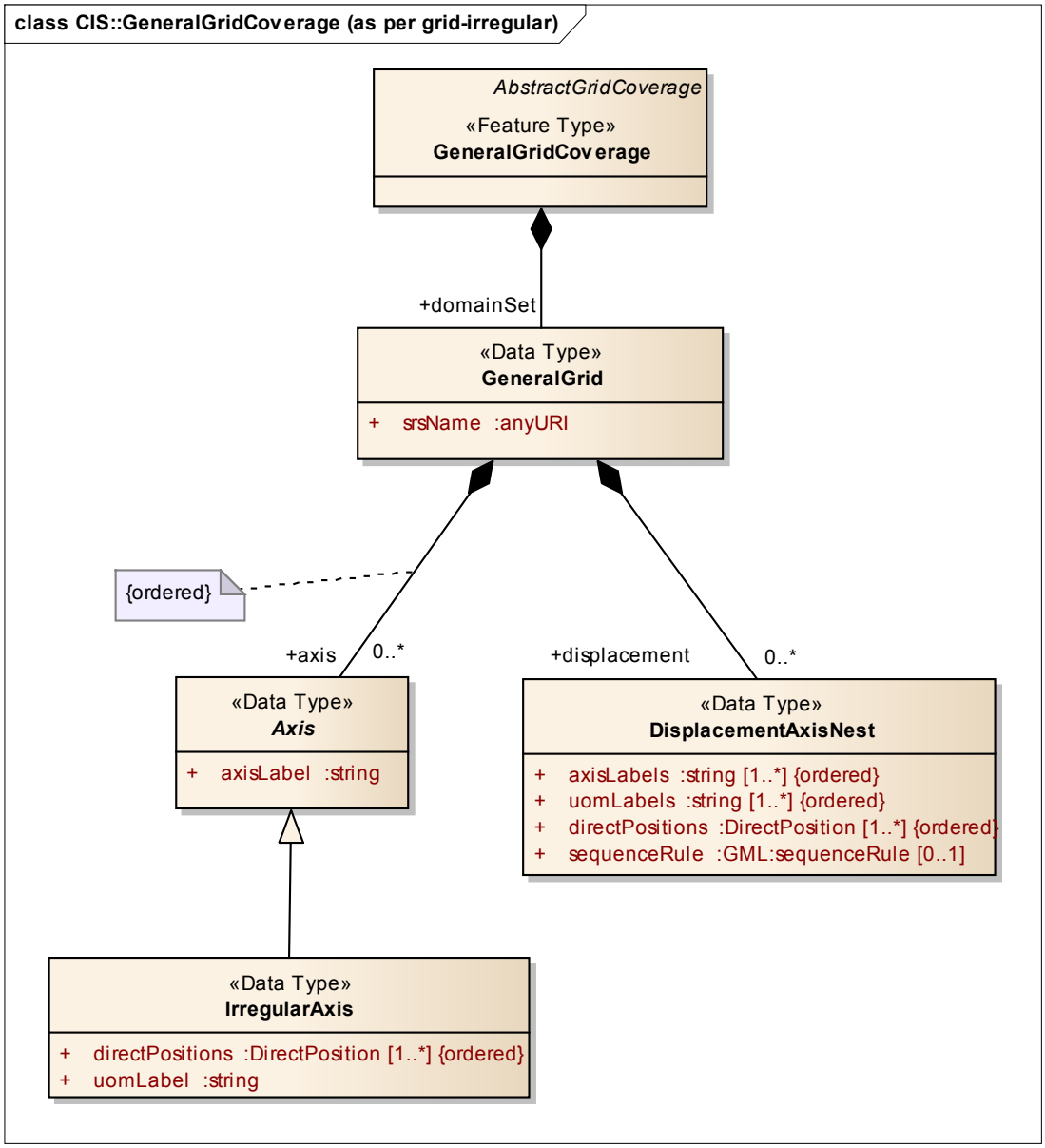

Figure 8: UML diagram of `CIS::GeneralGrid` structure as per *grid-irregular*

Table 14 `CIS::IrregularAxis` structure

Name	Definition	Data type	Multiplicity
`direct-Positions`	Ordered sequence of direct positions along this axis	`CIS::Direct-PositionType`	One or more (mandatory)
`uomLabel`	unit of measure in which values along this axis are expressed	`String`	One (mandatory)

An axis being part of a displacement grouping generalizes irregular axes further. Several axes together represent a grid where the individual direct positions of range values are situated

arbitrary in space/time. The `CIS::DisplacementAxisNest` combines several axes to a single "nest" where the coordinates are enumerated individually for each direct position.

Therefore, the direct positions are no longer associated with individual axes, but collectively form an array (tensor) which is stored in the `CIS::DisplacementAxisNest` structure, associated with the axes involved. The linearization scheme of this array is stated in the `sequenceRule` the same way as the linearization is described for the range set array.

Table 15 `CIS::DisplacementAxisNest` structure

Name	Definition	Data type	Multiplicity
`axisLabels`	Axes involved in the "nest" of displaced direct positions; these axes **shall** form a subset of the `CIS::GeneralGrid axisLabels`	`string`	One or more (mandatory)
`uomLabels`	units of measure in which values along the axes are expressed	`string`	One or more (mandatory)
`direct-Positions`	Array of direct positions along this axis, linearized according to the sequence rule or, if missing, along the GML 3.2.1 [2] default	`string`	One or more (mandatory)
`sequenceRule`	Description of the array linearization in `directPos-itions`, according to the GML 3.2.1 [2] sequence rule	`GML:: sequenceRule`	Zero or one (optional)

Note 1 Not all axes of a coverage need to participate in such a displacement "nest". For example, Lat and Long may form a surface in 3-D space whereas time axis is irregular. This is the case described in Figure 7 (right).

Note 2 The GML 3.3 type `ReferenceableGridByArray` resembles the special case that all axes form one nest – in other words, for each range value its direct position is explicitly listed in the domain set. This case is reflected in CIS through a `CIS::GeneralGrid` which has only axes of type `CIS:: DisplacementAxis` with one `CIS::DisplacementAxisNest` array (holding the direct position coordinates) associated with all these axes.

Requirement 21 :
In a coverage using the *grid-irregular* scheme, the `directPosition` values in any `CIS::IrregularAxis` **shall** be listed in strictly monotonic order, expressed in the units of measure of this axis as defined in the CRS identified in the `srsName` item of the `envelope`.

Note "Strictly monotonic" means that the sequence of position values is either completely in increasing order, or decreasing. Neither are changes in direction is allowed, nor equality of any two positions. This is to ensure that applications will not run into singularities causing, e.g., a division by zero. There is no corresponding monotonicity requirement on displaced axes (in the way Requirement 21 states for irregular axes). In practice, coverage generators should avoid grids that may lead to issues in coverage consumers - for example, singularities like neighboring points sharing the same coordinate could lead to a

division by zero. Conversely, applications reading coverages should be ruggedized to cope with borderline cases in an appropriate way.

Requirement 22 :
In a coverage using the *grid-irregular* scheme, for any two `CIS::DisplacementAxis-Nest` elements their set of `axis` names **shall** be disjoint.

All combinations of axis types index and regular (from class *grid-regular*) as well as irregular and displaced (from class *grid-irregular*) are permitted. However, no two axes may have the same name (i.e., axis label).

Example In a *Lat/Long/t* timeseries datacube, axes *Lat* and *Long* form a nest represented by two axes with axis name *Lat* and *Long*, resp., of type `CIS::RegularAxis` and one axis named *t* of type `CIS::IrregularAxis` storing all the image acquisition timestamps.

9. Class *grid-transformation*

9.1 Overview

Class *grid-transformation* establishes coverages with algorithmically defined grids. Currently one such transformation is defined which is based on SensorML 2.0 [5].

9.2 General

Requirement 23 :
A coverage using the *grid-transformation* scheme **shall** implement class *grid-regular*.

Requirement 24 :
A coverage using the *grid-transformation* scheme **shall** conform with Figure 9 and Table 16.

The cases currently supported by this standard – algorithmic transformation and specifically SensorML model – are defined in the Subclauses below.

9.3 Transformation

Grid definitions in the previous Clauses of this standard are defined through some well-known principle and (comparatively simple) computation methods. In the most general case, however, this is not the case, and only some special-built code – here called a "transformation" – with some particular variable instantiation can determine the direct positions of the grid. A special case of a transformation is provided by SensorML 2.0 [5], in CIS modelled through coverage type `CIS::SensorModelCoverage`.

Note It is recommended to ensure that transformations are invertible (i.e., an inverse transformation exists) in order to support the determination of the associated grid location of a given direct position.

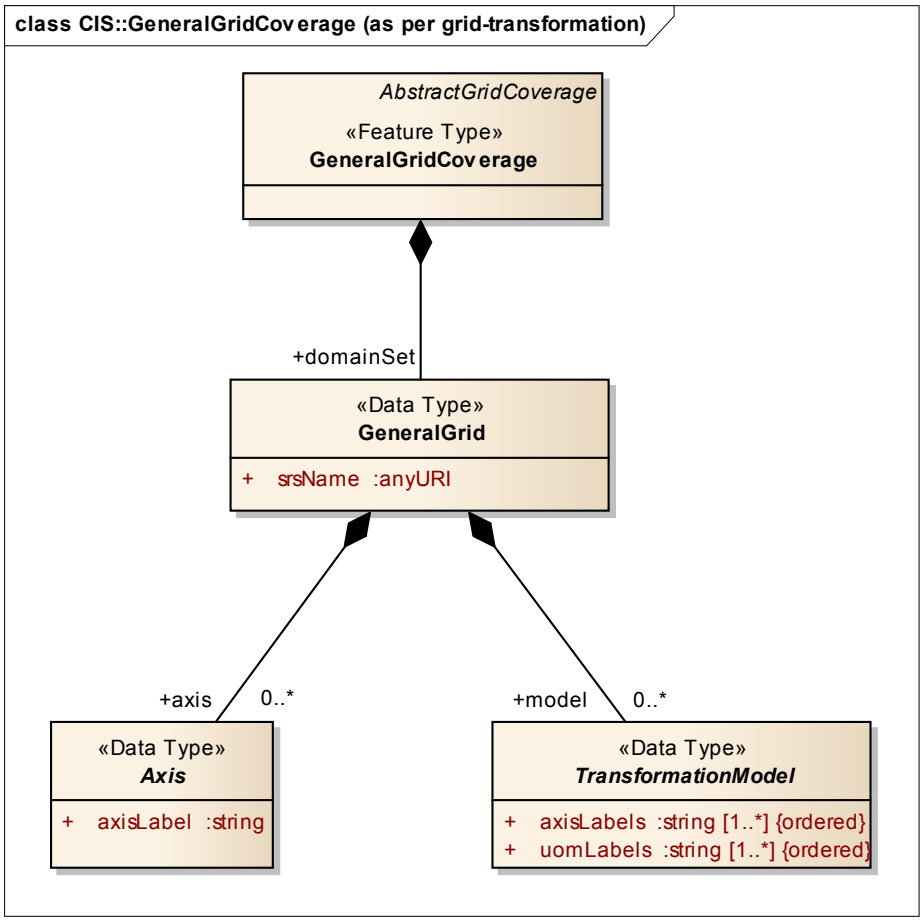

Figure 9: UML diagram of `CIS::GeneralGridCoverage` structure as per *grid-transformation*

Table 16 `CIS::TransformationModel` structure

Name	Definition	Data type	Multiplicity
`axisLabels`	List of axes involved in the transformation model	`string`	One or more (mandatory)
`uomLabels`	units of measure in which values along the axes are expressed	`string`	One or more (mandatory)

9.4 SensorML grid

Aside from the general definition, this standard supports one special case of such a transformation as defined by SensorML 2.0 [5]. Such a sensor model involves two inputs: a sensor model description containing free variables plus a separate set of variable instantiations (Table 17). As the sensor model defines the grid and its direct positions, this transformation effectively represents the coverage domain set.

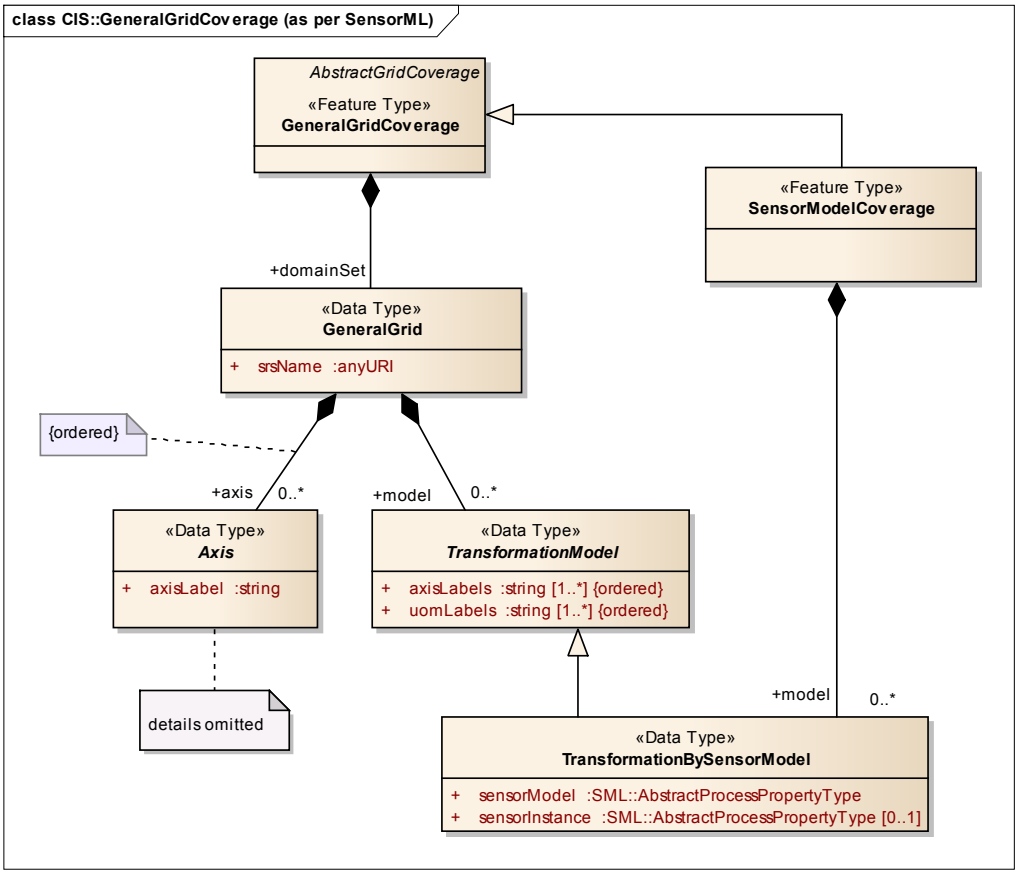

Figure 10: UML diagram of `CIS::GeneralGridCoverage` structure as per *SensorML*

Requirement 25:
In coverage of type `CIS::SensorModelCoverage` every `CIS::TransformationModel` **shall** be of type `CIS::TransformationBySensorModel` as specified in Figure 10 and Table 17.

Table 17 `CIS::TransformationBySensorModel` structure

Name	Definition	Data type	Multiplicity
`sensorModel`	SensorML model yielding the direct positions of the grid	`SML::Abstract-Process-PropertyType`	One (mandatory)
`sensor-Instance`	Parameter values for the sensor model	`SML::Abstract-Process-PropertyType`	Zero or one (optional)

The CIS::TransformationBySensorModel of the SensorML grid inherits attributes uomLabels and axisLabels that will be a directive to the sensor model software for the computed output geo locations. In general, these attributes will have no effect whatever on sensor model calculations except for the last stage when the output geo locations will be transformed from the native units and CRS of the software to the specified units and CRS of the CIS::TransformationBySensorModel.

Example 1 The following XML fragment defines the DomainSet of a frame camera sensor image modelled as a CIS::TransformationBySensorModel.

```
<DomainSet>
    <GeneralGrid srsName="http://www.opengis.net/def/crs/EPSG/0/4326"
            axisLabels="Lat Long">
        <GridLimits srsName=
                "http://www.opengis.net/def/crs/OGC/0/Index2D"
                axisLabels="i j">
            <IndexAxis axisLabel="i" lowerBound="0" upperBound="1919"/>
            <IndexAxis axisLabel="j" lowerBound="0" upperBound="1079"/>
        </GridLimits>
        <TransformationBySensorModel
                uomLabels="deg deg" axisLabels="Lat Long">
            <SensorModel xlink:href=
                "http://www.sensorml.com/csmFrame.html"/>
            <SensorInstance xlink:href=
                "http://www.sensorml.com/myHDCamera.html"/>
        </TransformationBySensorModel>
    </GeneralGrid>
</DomainSet>
```

Example 2 The following SensorML 2.0 defines parameters of a 2D electro-optical grid of a frame camera sensor, as part of a sensor model description referenced in the SensorModel subelement.

```
<SensorModel>
    <swe:field name="pixelGrid">
        <swe:DataRecord>
            <swe:label>Pixel Grid Characteristics</swe:label>
            <swe:field name="numberOfRows">
                <swe:Count definition=
                      "http://sensorml.com/ont/csm/property/NROWS">
                    <swe:label>Number of Rows</swe:label>
                </swe:Count>
            </swe:field>
            <swe:field name="numberOfColumns">
                <swe:Count definition=
                      "http://sensorml.com/ont/csm/property/NCOLS">
                    <swe:label>Number of Columns</swe:label>
                </swe:Count>
            </swe:field>
            <swe:field name="rowSpacing">
                <swe:Quantity definition=
                     "http://sensorml.com/ont/csm/property/ROW_SPACING">
                    <swe:label>Row Spacing</swe:label>
                    <swe:uom code="mm"/>
                </swe:Quantity>
            </swe:field>
            <swe:field name="columnSpacing">
```

```
                <swe:Quantity definition=
                     "http://sensorml.com/ont/csm/property/COL_SPACING">
                    <swe:label>Column Spacing</swe:label>
                    <swe:uom code="mm"/>
                </swe:Quantity>
            </swe:field>
        </swe:DataRecord>
    </swe:field>
</sensorModel>
```

Example 3 The following SensorML 2.0 fragment sets parameters of a 2D electro-optical grid of a frame camera sensor, as part of a sensor instance description referenced in the `sensorInstance` sub-element of `CIS::TransformationBySensorModel`, coherent with the parameter definitions of the previous example.

```
<sensorInstance>
    <sml:configuration>
        <sml:Settings>
            <sml:setValue ref="parameters/csm/pixelGrid/numberOfRows">
                1080
            </sml:setValue>
            <sml:setValue ref="parameters/csm/pixelGrid/numberOfColumns">
                1920
            </sml:setValue>
            <sml:setValue ref="parameters/csm/pixelGrid/rowSpacing">
                0.0074
            </sml:setValue>
            <sml:setValue ref="parameters/csm/pixelGrid/columnSpacing">
                0.0074
            </sml:setValue>
        </sml:Settings>
    </sml:configuration>
<sensorInstance>
```

Example 4 The following SensorML 2.0 snippet defines a 2D grid of a sensor model image through a list of inputs consistent with the `sensorModel` and `sensorInstance` subelements above.

```
<sml:inputs>
  <sml:InputList>
    <sml:input name="pixelGridCoordinates">
      <swe:Vector referenceFrame=
            "http://www.opengis.net/def/crs/OGC/0/IndexCRS2D">
        <swe:coordinate name="r">
          <swe:Quantity definition=
              "http://sensorml.com/def/property/ImageRowPosition">
            <swe:label>Row Position</swe:label>
              <swe:uom xlink:href=
                  "http://sensorml.com/def/property/pixel"/>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="c">
          <swe:Quantity definition=
              "http://sensorml.com/def/property/ImageColumnPosition">
            <swe:label>Column Position</swe:label>
              <swe:uom xlink:href=
                  "http://sensorml.com/def/property/pixel"/>
          </swe:Quantity>
        </swe:coordinate>
```

```
      </swe:Vector>
    </sml:input>
  </sml:InputList>
</sml:inputs>
```

10. Class *discrete-pointcloud*

Class *discrete-pointcloud* defines coverages which represent sets of multi-dimensional points at arbitrary positions.

The domain set of a discrete coverage consists of spatial and/or temporal objects, finite in number. The range set is comprised of a finite number of attribute values each of which is associated to every direct position within any single spatiotemporal object in the domain. In other words, the range values are constant on each spatiotemporal object in the domain. This coverage function maps each element from the coverage domain to an element in its range.

Requirement 26 :
A coverage instantiating class *discrete-pointcloud* **shall** conform with class *coverage*.

Requirement 27 :
A coverage using the *discrete-pointcloud* scheme **shall** conform with Figure 11 and Table 18.

Note While this definition is based on GML it does not preclude a GML encoding (through class *gml-coverage*); the same structures may be represented in any other suitable format (using class *other-format-coverage*).

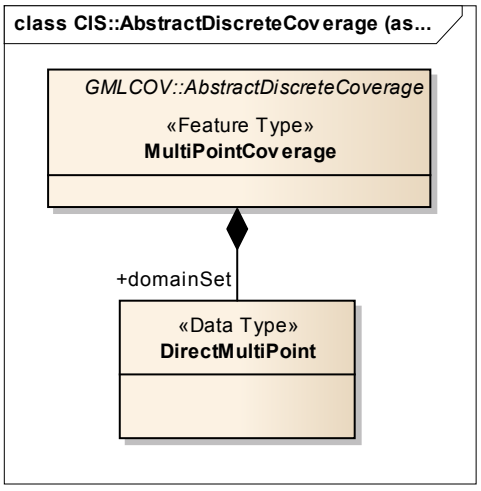

Figure 11: UML diagram of `CIS::MultiPointCoverage` structure

In a `MultiPointCoverage` the domain set is a `GM_MultiPoint`, that is a collection of arbitrarily distributed geometric points.

Table 18 `CIS::MultiPointCoverage` structure

Name	Definition	Data type	Multiplicity
`DomainSet`	Direct positions of coverage, describing points	`CIS::Direct-MultiPoint`	One (mandatory)

11. Class *discrete-mesh*

This class *discrete-mesh* establishes those discrete coverages which have a non-zero topological dimension, thereby extending class *discrete-pointcloud*. As such, it defines coverages consisting of curve, surface, and solid bundles, resp.

Requirement 28 :
A coverage using the *discrete-mesh* scheme **shall** implement class *discrete-pointcloud*.

Requirement 29 :
A coverage using the *discrete-mesh* scheme **shall** implement GMLCOV/CIS 1.0 coverage types `CIS10::MultiCurveCoverage`, `CIS10::MultiSurfaceCoverage`, and `CIS10::MultiSolidCoverage`.

Note While this definition is based on the *conceptual* model of GML it does not preclude a GML *encoding* (through class *gml-coverage*); the same structures may be represented in any other suitable format (using class *other-format-coverage*).

12. Class *gml-coverage*

12.1 Overview

Class *gml-coverage* establishes how coverages, as defined in this standard, are represented in the GML encoding format.

Note To make the GML schema of CIS more lightweight and self-contained, several GML definitions have been migrated into the CIS schema, at the same time simplifying these very general definitions for the particular use with coverages. Further, highly repetitive elements have been given particularly short to keep file size low. Therefore, strictly speaking the GML conformance class of CIS 1.1 is not a GML Application Profile anymore in the sense as defined in the GML standard.

The following convention has been adopted throughout CIS 1.1 for *gml-coverage*:

- Element and type names are in camel case with first letter capitalized
- Attribute names are in camel case with first letter lowercase.

Note This is a change over the corresponding schema definitions in GML 3.2.1 and GMLCOV/CIS 1.0 (which adheres to GML 3.2.1) where both lower and upper case can appear in element names, depending on their role in the schema. The reason for this change is to achieve coherent upper/lower case conventions across the XML, JSON, and RDF encoding of CIS as well as to simplify XML handling towards common XML Schema practices.

Requirement 30 :
A coverage using the *gml-coverage* scheme **shall** implement class *coverage*.

Requirement 31 :
In a coverage encoded in GML, the coverage document represented **shall** conform to the XML Schema definitions and Schematron rules being part of this standard.

Note 1 The XML Schema contained in this standard does not copy the abstract class definitions of Figure 2; rather, it deviates by not defining namespaces for GMLCOV/CIS 1.0 and GML 3.3. This allows applications which utilize only CIS 1.1 coverages to avoid pulling massive additional GML Schema files during validation.

Note 2 Coverage identifiers, as per GML are represented as `gml:id` attributes of XML type `NCName` which has constraints in the characters allowed. Therefore, naming of coverages is constrained, too, to such identifiers when using GML encoding.

This GML encoding is prepared for split representations where different parts of a coverage reside in different objects (such as files or databases), individually encoded. For example, domain set, range type, and range set each can independently be given by a URL; the same is possible for metadata – although it does not contain a file reference explicitly, its <any> definition allows for a URL as well.

Each range value is either atomic or composed from atomic values, each individually enclosed in an element.

Requirement 32 :
In a coverage encoded in GML, each atomic range value (i.e., `cis:v` element) **shall** contain exactly one value.

Note Such values will normally be numbers, encoded dates (as per ISO 8601), etc. The exact type definition for each range value component is governed by the range type.

Example The XML Schema being part of this specification contains several examples for different coverages encoded in XML.

References in GML are indicated through type `xs:anyURI` which specifies general syntax and semantics of URIs up to, and excluding, resolution of the fragment part (i.e., the URI part starting with a number sign, "#"). Fragment resolution is specified analogously to HTML:

Requirement 33 :
In a URI reference to a coverage component instantiating class *gml-coverage* the URI fragment component, if present, **shall** identify the value of a `gml:id` attribute in the target XML resource.

Example The following XML snippet demonstrates a possible way to incorporate a CRS definition within the coverage document:

```
<GeneralGridCoverage>
  <DomainSet>
    <GeneralGrid srsName="#myCrs"/>
    ...
  </DomainSet>
  ...
  <Metadata>
    <myLocalCrs gml:id="myCrs">
      here goes my CRS definition in GML, WKT, or otherwise
    </myLocalCrs>
  </Metadata>
</GeneralGridCoverage>
```

12.2 Coverage representation

Coverages can be encoded in any suitable format. One such format is established in GML 3.2.1 [2] stating that domain set items are mapped to range set items in XML document order or file sequence order, respectively.

Note As this statement above is not conformance testable no corresponding normative requirement is established.

13. Class *json-coverage*

Class *json-coverage* establishes how coverages, as defined in this standard, are represented in the JSON encoding format.

Requirement 34 :
A coverage using the *json-coverage* scheme **shall** implement class *coverage*.

Requirement 35 :
A coverage encoded in JSON test **shall** conform to IETF RFC7159.

Requirement 36 :
In a coverage encoded in JSON, the coverage document represented **shall** conform to the JSON Schema definitions being part of this standard.

Example The following JSON snippet is an example of a JSON encoded coverage.

```
{
    "type": "CoverageByDomainAndRangeType",
    "DomainSet":{
        "type": "DomainSetType",
        "generalGrid":{
            "type": "GeneralGridCoverageType",
            "srsName":
               "http://www.opengis.net/def/crs/OGC/0/Index2D",
            "axisLabels": ["i", "j"],
            "axis": [{
                "type": "IndexAxisType",
                "axisLabel": "i",
                "lowerBound": 0,
                "upperBound": 2
            },{
                "type": "IndexAxisType",
                "axisLabel": "j",
                "lowerBound": 0,
                "upperBound": 2
            }]
        }
    },
    "RangeSet": {
        "type": "RangeSetType",
        "dataBlock": {
            "type": "VDataBlockType",
            "values": [1,2,3,4,5,6,7,8,9]
        }
    },
    "RangeType": {
        "type": "DataRecordType",
        "field":[{
            "type": "QuantityType",
            "definition": "ogcType:unsignedInt",
            "uom": {
                "type": "UnitReference",
                "code": "10^0"
            }
        }]
    }
```

```
}
```

Note The JSON Schema being part of this specification has been used to validate the examples for different coverages encoded in JSON also provided.

14. Class *rdf-coverage*

Class *rdf-coverage* establishes how to represent coverages as Linked Data in RDF. This is done by providing a mapping between the JSON encoding and the RDF triples model using JSON-LD which allows that a JSON file with some additional content, defined in the W3C JSON-LD syntax [20], can be converted into RDF notation automatically using the W3C JSON-LD API [21].

Note One implementation of this API is provided in the JSON-LD Playground (http://json-ld.org/playground/).

Requirement 37 :
A coverage encoded in RDF **shall** conform to W3C RDF 1.1 Concepts and Abstract Syntax [22] and shall be constructed as if derived from a JSON encoded coverage which additionally conforms to W3C JSON-LD version 1 [20].

Note This conformance class has a dependency on the json-coverage only if the RDF encoding is derived from JSON-LD. The dependency on this class is not normative as coverage instances of this class can be RDF encoded without any previous use of JSON or JSON-LD to derive the class.
Although this conformance class refers to class *json-coverage* it is not normatively dependent on this class as coverage instances of this class do not implement the JSON encoding, but RDF. Subsequent requirements detail the structure of a hypothetical JSON-LD coverage leading to the RDF coverage defined.

Requirement 38 :
A coverage encoded in JSON-LD **shall** include a reference to a JSON-LD `@context` document for the coverage's root object and other JSON-LD `@context` documents for the objects `DomainSet`, `RangeSet`, `RangeType`, `envelope` and `partitionSet` when these objects are present.

Note Coverage components which are not in the above list of objects require personalized JSON-LD `@context` objects embedded or linked to allow mapping to the RDF models. One example for this is the `metadata` object.

Note The JSON-LD `@context` documents being part of this specification have been used to validate that examples of the different coverages encoded in JSON-LD also provided can be successfully converted to RDF.

Example The sample JSON code being part of this specification contains the necessary `@context` objects that can be linked or embedded in other JSON instances wanting to be conformant to this standard.

Requirement 39 :
A coverage encoded in JSON-LD **shall** embed or include a reference to a `@context` object defining the abbreviated and full namespace of the object identifiers in the way defined by the W3C JSON-LD standard.

Note This `@context` object is not included as a separated JSON-LD `@context` document because id namespaces are commonly responsibility of the data provider and should be provided by them. The provides can decide to provide a JSON-LD `@context` document to include by reference to several coverages or can embed this definition directly in the coverage.

Requirement 40 :
In a coverage encoded in JSON-LD, each object **shall** contain an `id` and `type` property where `id` values **shall** be composed by the abbreviated namespace for `ids`, a ":" (colon) character and the `id` value, and the `type` property **shall** be the name of the object's data type without namespace.

Note Large lists of values or coordinates embedded in the document are likely to produce excessively large RDF encodings. Therefore, instead of including them in the JSON file directly it can be advantageous to store such parts in separate files and reference these instead.

Example 1 Some of the sample JSON files being part of this specification have the values embedded (in places where potentially large lists will be used in practice) instead of being factored out into separated files. This is for didactic purpose only, these values are not be mapped to RDF when using the JSON-LD `@context` documents provided.

Example 2 The following JSON snippet illustrates an example of a JSON-LD encoded coverage with links to the `@context` document provided by this standard.

```
{
    "@context": ["http://schemas.opengis.net/cis/1.1/json/coverage-
context.json",
        {"examples": "http://www.opengis.net/cis/1.1/examples/"}],
    "type": "CoverageByDomainAndRangeType",
    "id": "examples:CIS_05_2D",
    "DomainSet":{
        "@context": "http://schemas.opengis.net/cis/1.1/json/domainset-
context.json",
        "type": "DomainSetType",
        "id": "examples:CIS_DS_05_2D",
        "generalGrid":{
            "type": "GeneralGridCoverageType",
            "id": "examples:CIS_DS_GG_05_2D",
            "srsName": "http://www.opengis.net/def/crs/OGC/0/Index2D",
            "axisLabels": ["i", "j"],
            "axis": [{
                "type": "IndexAxisType",
                "id": "examples:CIS_DS_GG_I_05_2D",
                "axisLabel": "i",
                "lowerBound": 0,
                "upperBound": 2
            },{
                "type": "IndexAxisType",
                "id": "examples:CIS_DS_GG_J_05_2D",
                "axisLabel": "j",
                "lowerBound": 0,
                "upperBound": 2
            }]
        }
    },
    "RangeSet": {
        "@context": "http://schemas.opengis.net/cis/1.1/json/rangeset-
context.json",
        "type": "RangeSetType",
        "id": "examples:CIS_RS_05_2D",
        "fileReference": "http://myserver.com/fileref.tiff"
    },
    "RangeType": {
```

```
      "@context": "http://schemas.opengis.net/cis/1.1/json/rangetype-
context.json",
      "type": "DataRecordType",
      "id": "examples:CIS_RT_05_2D",
      "field":[{
         "type": "QuantityType",
         "id": "examples:CIS_RT_F_05_2D",
         "definition": "ogcType:unsignedInt",
         "uom": {
            "type": "UnitReference",
            "id": "examples:CIS_RT_F_UOM_05_2D",
            "code": "10^0"
         }
      }]
   }
}
```

Example 3 The following RDF triples representation corresponds to the JSON-LD encoded coverage listed above:

```
<http://www.opengis.net/cis/1.1/examples/CIS_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/CoverageByDomainAndRangeType> .

<http://www.opengis.net/cis/1.1/examples/CIS_05_2D>
     <http://www.opengis.net/cis/1.1/DomainSet>
     <http://www.opengis.net/cis/1.1/examples/CIS_DS_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_05_2D>
     <http://www.opengis.net/cis/1.1/generalGrid>
     <http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/DomainSetType> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D>
     <http://www.opengis.net/cis/1.1/axis>
     <http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_I_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D>
     <http://www.opengis.net/cis/1.1/axis>
     <http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_J_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D>
     <http://www.opengis.net/cis/1.1/axisLabels>
     <http://www.opengis.net/cis/1.1/axisLabels0> .
<http://www.opengis.net/cis/1.1/axisLabels0>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "i" .
<http://www.opengis.net/cis/1.1/axisLabels0>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
     <http://www.opengis.net/cis/1.1/axisLabels1> .
<http://www.opengis.net/cis/1.1/axisLabels1>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "j" .
<http://www.opengis.net/cis/1.1/axisLabels1>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D>
     <http://www.opengis.net/cis/1.1/srsName>
     <http://www.opengis.net/def/crs/OGC/0/Index2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/GeneralGridCoverageType> .
```

```
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_I_05_2D>
     <http://www.opengis.net/cis/1.1/axisLabel> "i" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_I_05_2D>
     <http://www.opengis.net/cis/1.1/lowerBound>
     "0"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_I_05_2D>
     <http://www.opengis.net/cis/1.1/upperBound>
     "2"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_I_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/IndexAxisType> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_J_05_2D>
     <http://www.opengis.net/cis/1.1/axisLabel> "j" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_J_05_2D>
     <http://www.opengis.net/cis/1.1/lowerBound>
     "0"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_J_05_2D>
     <http://www.opengis.net/cis/1.1/upperBound>
     "2"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_J_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/IndexAxisType> .

<http://www.opengis.net/cis/1.1/examples/CIS_05_2D>
     <http://www.opengis.net/cis/1.1/RangeSet>
     <http://www.opengis.net/cis/1.1/examples/CIS_RS_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RS_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/cis/1.1/RangeSetRefType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RS_DB_05_2D>
     <http://www.opengis.net/cis/1.1/fileReference>
     <http://myserver.com/fileref.tiff> .

<http://www.opengis.net/cis/1.1/examples/CIS_05_2D>
     <http://www.opengis.net/cis/1.1/RangeType>
     <http://www.opengis.net/cis/1.1/examples/CIS_RT_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_05_2D>
     <http://www.opengis.net/swe/2.0/field>
     <http://www.opengis.net/cis/1.1/examples/CIS_RT_F_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/swe/2.0/DataRecordType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_05_2D>
     <http://www.opengis.net/swe/2.0/definition>
     <http://www.opengis.net/def/dataType/OGC/0/unsignedInt> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_05_2D>
     <http://www.opengis.net/swe/2.0/uom>
     <http://www.opengis.net/cis/1.1/examples/CIS_RT_F_UOM_05_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/swe/2.0/QuantityType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_UOM_05_2D>
     <http://www.opengis.net/swe/2.0/code> "10^0" .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_UOM_05_2D>
     <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.opengis.net/swe/2.0/UnitReference> .
```

15. Class *other-format-coverage*

Class *other-format-coverage* establishes how coverages are represented in encoding formats other than those defined in this standard.

Note Such formats may be able to encode only parts of a coverage (i.e., they are "informationally incomplete"), and they may be able to encode only specific categories of coverages (such as raster images, but not point clouds).

Requirement 41 :
A coverage using the *other-format-coverage* scheme **shall** implement class *coverage*.

16. Class *multipart-coverage*

16.1 Overview

Class *multipart-coverage* establishes how coverages can be packaged into multiple files, meaning that the coverage document (henceforth referred to as the "first part") has one or more components shifted out into separate documents (henceforth called "further parts"). To maintain connection between the parts, the first part references all other parts through URLs (which may be local). Packaging can be done through any appropriate container format. Additionally, parts can be stored outside the package, referenced by URLs.

Note Among the suitable container formats are multipart MIME [4], GMLJP2, zip, and tar. Out of those, MIME is normatively defined here.

Such a splitting is particularly useful for the range set so as to allow a different, possibly more efficient encoding of this (typically) bulk of information. However, with the same argument other parts of the coverage (such as a large domain set with displaced axes) can be shifted into further parts as well.

To achieve a complete representation of the coverage, the encoding used in the first part must be "informationally complete," i.e.: able to hold the complete coverage information. Further, it must be allow expression of references (which replace the substructure – such as the range set – to be shifted into a separate part). Notably, the format used in the further parts does not need to be informationally complete with respect to coverage metadata; however, it must be able to represent the values factored out of the first-part document.

Note Among the list of suitable formats for the first part are GML and JSON. Image/data formats like GeoTIFF and NetCDF are suitable formats for the further parts.

Requirement 42 :
A coverage using the *multipart-coverage* scheme **shall** implement class *coverage*.

Requirement 43 :
A coverage encoded as a multipart MIME message **shall** adhere to IETF RFC 2387 [16] in that it consists of a multipart MIME document with a `Content-Type` parameter of value "`Multipart/Related`" and a `Type` parameter containing a MIME type identifier matching the encoding of the first ("root") part; references to further parts located in the same container as the first-part coverage shall use a local "cid" (Content-ID) URL as specified by IETF RFC 2392 [17].

Note 1 The MIME type identifier of GML, for example, is "application/gml+xml".

Note 2 In GMLCOV/CIS 1.0 a `ContentDisposition` parameter of value "`inline`" was required. This is not required any more in CIS 1.1.

References used in coverage parts follow common URI standards for syntax [18] and semantics [12].

16.2 Root part

The *root part* of a multipart coverage consists of the top-level structure of the coverage. Each container format needs to individually determine how this root part is represented.

Example In Multipart / MIME, this is the first item in the stream. In a zip file, it might be a manifest file. Each format needs establish unambiguous conventions, such as a particular file name in a zip archive.

Requirement 44 :
In a coverage encoded as per class *multipart-coverage*, the root part **shall** be a complete coverage as per this standard, but with one or more components replaced by a reference to the further parts of the multipart message where these components replaced get manifested.

Example In a GML encoded coverage, a reference can be expressed through a `fileReference` element.

Note Each part of the message can be encoded in different formats individually and independently.

Requirement 45 :
In a coverage encoded as per class *multipart-coverage*, references from the first message part (containing the coverage root part) to subsequent parts **shall** use the method foreseen by the container format to achieve an unambiguous identification of the further parts located in the same container as the first-part coverage.

Note 1 Generally, syntax and semantics of the reference may depend on the environments in which the coverage containing the reference, on the one hand, and the item referenced, on the other hand, reside: in a multipart MIME message, this will be cid identifiers; in a zip file, identification will be done through file names and paths relative to the zip directory root; this hierarchical scheme would allow relative references. In a GMLJP2 file, identification will be done through XML identifiers, i.e., locally unique gml:id attributes. If keeping a sandboxed environment is important, e.g., for security reasons, the W3C `app:` URI scheme [13] might be used.

Note 2 A reference may be temporarily or permanently unresolvable. In case of an unresolvable reference, the coverage may still be reconstructable through other means – for example, treatment of CRSs given by some well-known URI may be hardwired in an application handling coverages.

16.3 Further parts

The root part may, instead of containing coverage constituents verbatim, shift such constituents into subsequent parts of the multipart document and reference them.

Requirement 46 :
In a coverage encoded as per class *multipart-coverage*, any part referenced from the root part **shall** contain the complete information required to substitute the reference and obtain a complete coverage as per class *coverage*.

Note In GMLCOV/CIS 1.0, only one extra part was foreseen exclusively for the range set. Starting with CIS 1.1 more than one coverage component can be extracted into a separate part. Besides the (often large) range set, another candidate for a separate part is the domain set in a coverage with displaced axes, as such a domain set may become just as large as the range set. In a Discrete Coverage, the domain set typically is even larger than the range set.

Example The following MIME message represents a valid multipart coverage structure with the root part encoded in GML and the second part encoded in TIFF (assuming all "..." substituted by proper XML and with a proper TIFF stream instead of "`...binary TIFF data...`"):

```
Content-Type: Multipart/Related; boundary=cis;
        start="GML-Part"
        type="application/gml+xml"

--cis
Content-type: application/gml+xml
Content-ID: GML-Part

<?xml version="1.0" encoding="UTF-8"?>
...GML data...
--cis
Content-Type: image/tiff
Content-Description: coverage data
Content-Transfer-Encoding: binary
Content-ID: grey.tif
Content-Disposition: inline

...binary TIFF data...
--cis--
```

17. Class *coverage-partitioning*

17.1 Overview

This class *coverage-partitioning* establishes an alternative representation for coverages through partitioning into sub-coverages or direct enumeration of position/value pairs.

17.2 Partitioning

With the coverage extensions provided by this class coverages can be composed from other coverages which are either copied in directly ("domain-and-range" variant), or referenced by coverage id ("partitioning" variant), or can contain single values per direct position ("position/value pair" variant, sometimes also called "geometry/value pair" or "interleaved").

Coverages embedded ("sub-coverages") can be of the same or lower dimension than the coverage embedding them ("super-coverage"). The `partition` element in the super-coverage, acting as a connection between sub- and super-coverage, contains an `envelope` element determining the sub-coverage's position relative to the super-coverage. A coverage can be part of several partitioned coverages simultaneously, thereby allowing shared regions. A partitioned coverage can itself be part of another partitioned coverage, thereby allowing trees of coverages to be built recursively.

In the position/value pair approach, single range values (which may be composite, such as RGB pixel values) are listed together with their direct position.

All of the above variants can be combined freely within a single coverage as per this standard. However, an implementation may constrain the partitioning choices available, such as to "partitioning only along time axis" or "only equi-sized sub-coverages". Further, it may support selection of partitioned and "geometry/value pair" representation.

Requirement 47 :
A coverage using the *coverage-partitioning* scheme **shall** conform to class *coverage*.

Requirement 48 :
A coverage using the *coverage-partitioning* scheme **shall** conform to Figure 12, Table 19, Table 20, Table 21, Table 22, and Table 23.

The partitioning mechanism effectively establishes a nesting of coverages. This nesting must be acyclic, i.e., a coverage cannot contain itself.

Requirement 49 :
A coverage **shall** not reference itself through a `partition` element, neither directly nor indirectly.

All "sub-coverages" participating in a partitioned coverage must lie inside the super-coverage and additionally must fulfill homogeneity criteria to ensure that the resulting structure adheres to the definition of a coverage, as specified in the following Subclauses.

A coverage can act as sub-coverage in more than one coverages.

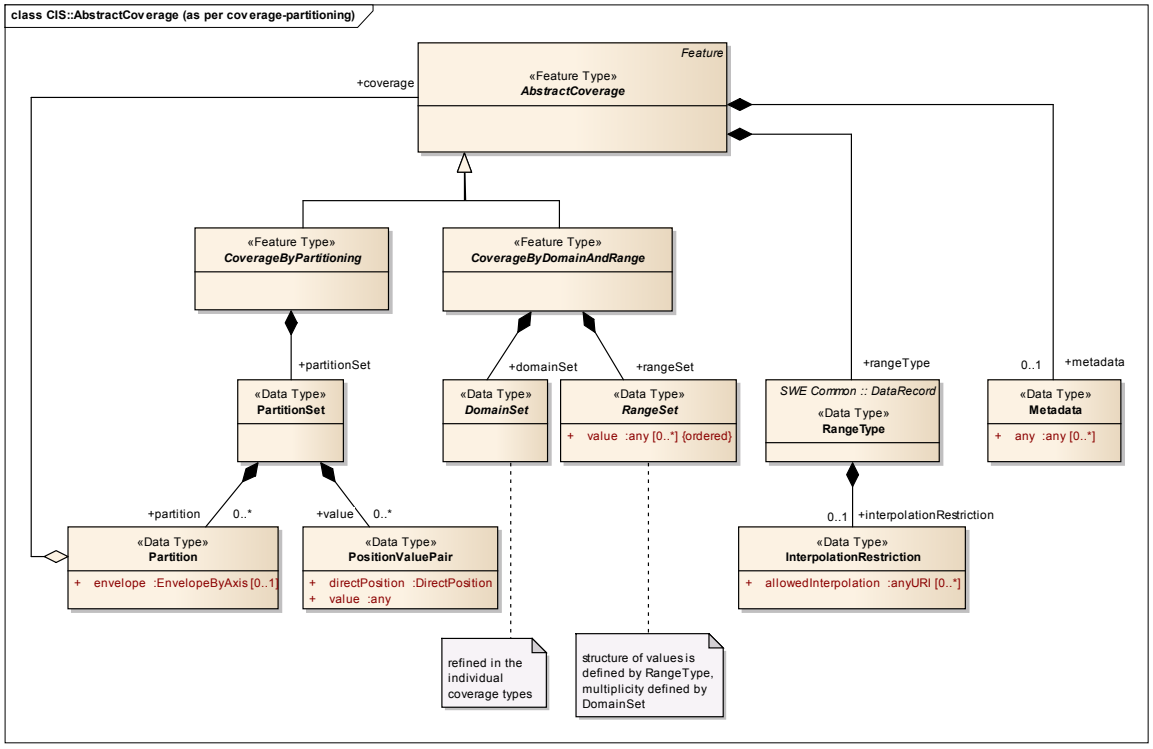

Figure 12: UML diagram of `CIS::CoverageByPartitioning` structure as per *coverage-partitioning*

Table 19 `CIS::CoverageByPartitioning` structure

Name	Definition	Data type	Multiplicity
`partition-Set`	Set of coverages or single positioned values which together make up the coverage on hand	`CIS::PartitionSet`	one (mandatory)

Table 20 `CIS::PartitionSet` structure

Name	Definition	Data type	Multiplicity
`partition`	Sub-coverage being part of the coverage on hand, together with positioning information	`CIS::Partition`	Zero or one (optional)
`value`	Range value being part of the coverage on hand, together with positioning information	`CIS::PositionValuePair`	Zero or one (optional)

Table 21 `CIS::Partition` structure

Name	Definition	Data type	Multiplicity
`envelope`	Envelope of sub-coverage making up this partition; default: envelope of the coverage referenced	`CIS::EnvelopeByAxis`	Zero or one (optional)
`coverage`	Coverage acting as partition (directly stored here or through some resolvable reference, such as coverage id or a URL)	`CIS::AbstractCoverage`	One (mandatory)

Table 22 `CIS::PositionValuePair` structure

Name	Definition	Data type	Multiplicity
`direct-Position`	Direct position of the coverage to which `value` is assigned	`string`	One (mandatory)
`value`	Coverage value to be associated with `directPosition`	`any`	One (mandatory)

Table 23 `CIS::RangeTypeComponentTranslation` structure

Name	Definition	Data type	Multiplicity
`super-Coverage-Component-Name`	Name of range type component as defined in the super-coverage range type	`string`	One (mandatory)
`sub-Coverage-`	Name of corresponding range type component as defined in the sub-	`string`	One (mandatory)

`Component-Name`	coverage range type		

Sub-coverages can be stored directly as the value of `coverage`, or they can be given as some reference, such as coverage id or a URL.

Note Support for these alternatives may vary across data format encodings. Further, as this is a normative requirement which a server must fulfill an implementation possibly will restrict the options for referencing coverages to those ones where it can control this acyclicity requirement.

17.3 CRS and partition envelope constraints

The sub-coverage CRS must allow that the coverage data can be embedded in the super-coverage referencing it.

Requirement 50 :
For any coverage *s* with domain set CRS *cs* being a partition of some coverage *c* with domain set CRS *cc*, the following **shall** hold: *cs* is obtained from *cc* by deleting zero or more axes from *cc*.

Note This definition enforces an identical axis order among those axes present in both the sub- and super-coverage CRSs.

Example A timeseries datacube with CRS axes *Lat/Long/t* can contain sub-coverages whose CRS axes are given by *Lat/Long*, but not by *Long/Lat*. A datacube with axis order *t/Lat/Long* likewise can contain sub-coverages with a *Lat/Long* CRS.

Lower-dimensional sub-coverages are embedded as slices of thickness one into the super-coverage.

Requirement 51 :
For any axis not occurring in the domain set CRS *cp* of coverage *p* but listed as a partition of some coverage *c* with domain set CRS *cc*, `lowerBound` = `upperBound` **shall** hold in the `envelope` of the *p* partition referencing *s*.

Note This allows to "lift"coverage parts into higher-dimensional spaces in the super-coverage, such as embedding a 2-D Lat/Long timeslice into a 3-D Lat/Long/time datacube.

The `CIS::partitionEnvelope` element does not need to repeat coordinate axis values of the sub-coverage if they are identical in the context of the super-coverage.

Requirement 52 :
For any axis of the domain set CRS *cc* of some coverage *c* containing some coverage *p* as a partition, any axis not listed in *c*'s `partitionEnvelope` within *p* the default `lowerBound` and `upperBound` of this axis in the `partitionEnvelope` **shall** be given by the corresponding values in the `DomainSet` of *p*.

Note Axis identification and sequence is unambiguous even when axes are left out because `partitionEnvelope` coordinates are expressed in terms of the super-coverages CRS which defines all axes and their sequence.

17.4 Domain set constraints

The sub-coverage domain sets, as well as single direct positions, must be non-overlapping (considering all axes plus the range components) and properly contained in the super-coverage; missing boundary values are represented as a null value.

Note Such null values can be used whenever the actual extent of the super-coverage is not known in the super-coverage itself, such as in timeseries where further timeslices can be appended at any time. The representation of such a null value is defined in the concrete encodings.

Requirement 53 :
For any coverage *p* referenced as `partition` in a coverage *c*, the `envelope` of *p* **shall** be a subset of the domain set of *c*, obtained by ignoring all values of `lowerBound` and `UpperBound` in the `envelope` of *c* which have a null value.

Requirement 54 :
For any coverage *c* of type `CIS:CoverageByPartitioning`, all `partition` and `value` components **shall** have pairwise disjoint extents across any of its range components.

Example Band-interleaved (BIL) representation can be achieved through multiple sub-coverages all registered to the same extent, but each one adding an individual band.

Requirement 55 :
In a coverage containing at least one direct position for which no value is stored there **shall** be at least one null (i.e., nil) value defined in its range type.

Note 1 Such "undefined areas" can only occur with coverages containing partitions (in a domain / range representation there must always exist a value for each direct position). This rule makes sure that "null values" exist when needed.

Note 2 Such "default" null values can differ among direct positions, an implementation is free to choose values non-deterministically. It is good practice, though, to use a single value whenever possible.

17.5 Range type constraints

Sub- and super-coverage must have compatible range types – either identical ones, or partitions contribute parts of the full super-coverage range component record.

Requirement 56 :
For any coverage *p* with range type *rp* referenced as a `partition` in a coverage *c* with range type *rc*, the following **shall** hold: *rp* is obtained from *rc* by deleting zero or more range components from *rc*.

Note Sub-coverage bands are visible in the super-coverage under the name indicated in the range type translation list, which obviously must not lead to name clashes in the super-coverage (i.e., range component names still have to be pairwise distinct). Further, from the super-coverage perspective, all range components "imported" must adhere to the same range type definition to not violate the basic definition of range type coherence in a coverage.

Example Band-interleaved storage of satellite imagery, as well as variables in climate model output can be accomplished this way: single bands, or combinations of bands, can go into separate sub-coverages which are linked together through a super-coverage.

If the partitions altogether are not commensurate to the complete range type structure then the range components not covered are equivalent to some null value (which must be defined in this case).

Requirement 57 :
In any coverage containing at least one range component for which no value is stored there **shall** be at least one null (i.e., nil) value defined in the corresponding range type component.

Example 1 Consider an RGB coverage where the color bands are factored out into partitions. Assume that there are only partitions for the red and green, but not for the blue band. In this case, the range type definition of the RGB coverage must provide a null value for the blue band so that an equivalent "flat" coverage can be constructed which contains null values in all direct positions for the missing blue band.

Example 2 Band interleaving combined with spatial partitioning (such as in mosaics) may lead to small islands of null values. For each of them, a proper null value definition must exist allowing an implementation to interpret the missing value as one of these null values.

18. Class *container*

This class *container*, which is free-standing and not dependent on class *coverage*, establishes a general data type and format independent information unit. Such units are particularly useful when aggregating homogeneous information (such as several coverages) or heterogeneous information (such as coverages annotated with other coverages, features, and metadata).

Note Container objects can be conveniently queried by XPath when encoded in XML, and by similar existing techniques when encoded in some other format like JSON. This notwithstanding, there is no restriction on the encoding – individual components of an object may be encoded individually in different formats.

The definition of the target structure, `CIS::Object`, is tentatively as general as ever possible. Applications will derive bespoke instantiable subclasses from this abstract class.

Requirement 58 :
An object using the *container* scheme **shall** conform to Figure 13.

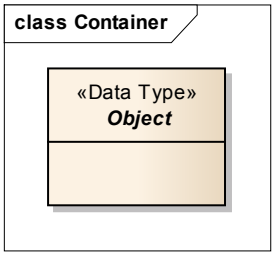

Figure 13: UML diagram of `CIS::Object` structure as per *container*

Note This container approach is intended to align with related standards on heterogeneous data and services on them. Information from such objects can be extracted, for example, through the XPath-based retrieval defined in the OGC Web Information Service (WIS) [10].

Annex A
(normative)

Abstract Test Suite

This Annex specifies an Abstract Test Suite which shall be passed in completeness by any implementation claiming conformance with this Application Schema.

The test approach conceptually consists of two steps:

- Transcode the coverage from its original format into one of the formats directly addressed by this standard[4], following the mapping rules defined for the particular original format on hand[5].

- Perform all conformance tests on this transcoded coverage representation. Tests fail/succeed if they fail/succeed, resp., on this transcoded representation.

A concrete test implementation may choose a different strategy (may be for efficiency reasons) as long as the tests behave as indicated in this Abstract Test Suite.

A.1 Conformance Test Class: *coverage*

Test Purpose: Requirement 1

Test method: Test the coverage under test:

- If the coverage passes the tests of CIS 1.1 core conformance class *coverage* (disregarding this Requirement 2), pass test.

- Otherwise, if the coverage passes the tests of GMLCOV/CIS 1.0 core conformance class *gml-coverage*, pass test.

- Otherwise, if the coverage is a gridded coverage and it passes the tests of GMLCOV/CIS 1.0 core conformance class *gml-coverage* with a grid structure as defined in GML 3.3, pass test.

- Otherwise, fail test.

Test Purpose: Requirement 2

Test method: Determine the encoding of the coverage under test:

- If the encoding is GML, perform the conformance test defined for

[4] Currently, this is GML; in future, JSON will be added.

[5] At the time of this writing, such OGC coverage mapping standards exist for GeoTIFF, GMLJP2, and NetCDF; GRIB is under construction.

class *gml-coverage*.

- Otherwise, if the encoding is in some other format:
    - Convert the coverage into one of the formats directly addressed by this CIS standard, according to the coverage mapping defined for the corresponding encoding standard;
    - perform the conformance test defined of the resp. format;
    - perform the conformance test defined for class *other-format-coverage*.
- Otherwise, fail test.

Test passes overall if all detail checks pass.

Test Purpose: Requirement 3

Test method: Verify that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Verify that all necessary elements are present (with the exception described in class *other-format-coverage*).

Test passes if all detail checks pass.

Test Purpose: Requirement 4

Test method: From the coverage under test extract the `envelope`, if present.

- If none present: pass test.
- If present: verify that it consists of a `CIS::EnvelopeByAxis` element with the required structure.

Test passes if all constraints evaluate to true.

Test Purpose: Requirement 5

Test method: From the coverage under test extract the `envelope`, if present.

- If none present: pass test.
- If present: verify that all constraints are fulfilled.

Test passes if all detail checks pass.

Test Purpose: Requirement 6

Test method: From the coverage under test extract the `envelope`, if present.

- If none present: pass test.
- If present: verify constraint for all occurrences of `axisExtent`.

Test passes if all constraints evaluate to true.

Test Purpose: Requirement 7

Test method: From the coverage under test extract the `envelope`, if present.

- If none present: pass test.
- If present: If the `envelope` uses a CRS different from the `DomainSet` then first transform the `envelope` CRS coordinates into the `DomainSet` CRS. Check that the `envelope` describes a bounding box around the `DomainSet`, taking into account all axes of the `DomainSet` CRS.

Test passes if all detail checks pass.

Test Purpose: Requirement 8

Test method: In the coverage under test, verify that for each axis in the domain set the coordinates of all direct positions are within the closed interval [`lower-Bound`, `upperBound`] indicated in the corresponding axis extent.

Test passes if all detail checks pass.

Test Purpose: Requirement 9

Test method: In the coverage under test, inspect the RangeType component and verify that the structure is as required.

Test passes if all detail checks pass.

Test Purpose: Requirement 10

Test method: In the coverage under test, inspect all SWE Common `AbstractSimpleComponent` subtypes in a range type structure and verify that no `value` component is present[6].

Test passes if all detail checks pass.

Test Purpose: Requirement 11

Test method: In the coverage under test, inspect the range type structure and verify that each SWE Common `AbstractSimpleComponent` item is of the allowed subtypes listed.

Test passes if all detail checks pass.

Test Purpose: Requirement 12

Test method: In the coverage under test, verify that for each location defined in the domain set there is exactly one corresponding value in the range set.

Test passes if all detail checks pass.

Test Purpose: Requirement 13

Test method: In the coverage under test, verify for each range value tuple:

- Number of tuple components adheres to range structure definition.
- Data type (including unit of measure, where indicated) of each range value conforms to the corresponding data type specification in the range structure definition.

Test passes if all detail checks pass.

A.2 Conformance Test Class: *grid-regular*

Test Purpose: Requirement 14

Test method: The coverage under test must pass all tests of class *coverage*.

[6] In case of a GML encoding, the corresponding Schematron rule provided with the XML Schema checks this.

Test passes if all detail checks pass.

Test Purpose: Requirement 15

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

Test Purpose: Requirement 16

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

Test Purpose: Requirement 17

Test method: In the coverage under test, verify that the requirement is met by each regular axis.

Test passes if all detail checks pass.

Test Purpose: Requirement 18

Test method: In the coverage under test, verify:

- if the coverage's domain set contains a `CIS::GeneralGrid` then verify whether the equation for the number of direct positions in the grid is fulfilled.
- Otherwise, pass test.

Test passes if all detail checks pass.

A.3 Conformance Test Class: *grid-irregular*

Test Purpose: Requirement 19

Test method: The coverage under test must pass all tests of class *grid-regular*.

Test passes if all detail checks pass.

Test Purpose: Requirement 20

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

Test Purpose: Requirement 21

Test method: In the coverage under test, verify monotonicity for every axis of type `CIS::IrregularAxis` in the domain set.

Test passes if all detail checks pass.

Test Purpose: Requirement 22

Test method: In the coverage under test, verify that all displacement axes have pairwise different names.

Test passes if all detail checks pass.

A.4 Conformance Test Class: *grid-transformation*

Test Purpose: Requirement 23

Test method: The coverage under test must pass all tests of class *grid-regular*.

Test passes if all detail checks pass.

Test Purpose: Requirement 24

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

Test Purpose: Requirement 25

Test method: In the coverage under test, verify:

- If its type is `CIS::SensorModelCoverage`, verify that each axis in the domain set is of type `CIS::TransformationAxis` and that there is exactly one `CIS::TransformationModel`.
- Otherwise, pass test.

Test passes if all detail checks pass.

A.5 Conformance Test Class: *discrete-pointcloud*

Test Purpose: Requirement 26

Test method: The coverage under test must pass all tests of class *coverage*.

Test passes if all detail checks pass.

Test Purpose: Requirement 27

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

A.6 Conformance Test Class: *discrete-mesh*

Test Purpose: Requirement 28

Test method: The coverage under test must pass all tests of class *discrete-pointcloud*.

Test passes if all detail checks pass.

Test Purpose: Requirement 29

Test method: Check that the coverage under test conforms with one of the coverage types listed.

Test passes if all detail checks pass.

A.7 Conformance Test Class: *gml-coverage*

Test Purpose: Requirement 30

Test method: The coverage under test must pass all tests of class *coverage*.

Test passes if all detail checks pass.

Test Purpose: Requirement 31

Test method: In the coverage under test, if it is encoded in XML then verify that the document body validates against the schema and the Schematron rules being part of this standard.

Test passes if all detail checks pass.

Test Purpose: Requirement 32

Test method: In the coverage under test, verify for each that each element contains exactly one value conforming to the coverage's range type definition.

Test passes if all detail checks pass.

Test Purpose: Requirement 33

Test method: In the coverage under test, verify for each reference targeting an XML document that the fragment, if present, identifies a `gml:id` attribute in the target document.

Test passes if all detail checks pass.

A.8 Conformance Test Class: *json-coverage*

Test Purpose: Requirement 34

Test method: The coverage under test must pass all tests of class *coverage*.

Test passes if all detail checks pass.

Test Purpose: Requirement 35

Test method: In the coverage under test, if it is encoded in JSON then verify that the document conforms to IETF RFC7159.

Test passes if all detail checks pass.

Test Purpose: Requirement 36

Test method: In the coverage under test, if it is encoded in JSON then verify that the document body validates against the schema being part of this standard.

Test passes if all detail checks pass.

A.9 Conformance Test Class: *rdf-coverage*

Test Purpose: Requirement 37

Test method: In the coverage under test, if it is encoded in RDF then verify that the document conforms to W3C RDF 1.1 and can be derived from a JSON-LD encoded coverage as defined in this conformance class and W3C JSON-LD version 1.

Test passes if all detail checks pass.

Test Purpose: Requirement 38

Test method: In the coverage under test, if it is encoded in JSON-LD then verify that the document links to the `@context` documents being part of this standard for the root object and the objects `DomainSet`, `RangeSet`, `RangeType`, `envelope` and `partitionSet` if these objects are present.

Test passes if all links required are present.

Test Purpose: Requirement 39

Test method: In the coverage under test, if it is encoded in JSON-LD then verify that all abbreviated namespaces for identifiers are defined in a `@context` section

Test passes if all detail checks pass.

Test Purpose: Requirement 40

Test method: In the coverage under test, if it is encoded in JSON-LD then verify that all objects in the JSON document have two properties with the name "`id`" and "`type`". In addition, verify that the "`id`" values use an abbreviated namespace and "`type`" values do not.

Test passes if all detail checks pass.

A.10 Conformance Test Class: *other-format-coverage*

Test Purpose: Requirement 41

Test method: The coverage under test must pass all tests of class *coverage*.

Test passes if all detail checks pass.

A.11 Conformance Test Class: *multipart-coverage*

Test Purpose: Requirement 42

Test method: The coverage under test must pass all tests of class *coverage*.

Test Purpose: Requirement 43

Test method: In the coverage under test, verify:

- If it is encoded as a multipart message, verify all MIME conditions. Test passes if all partial tests pass.
- Otherwise, pass test.

Test passes if all detail checks pass.

Test Purpose: Requirement 44

Test method: In the coverage under test, verify:

- If it is encoded in a multipart message, extract the first part. Substitute all references from this part into subsequent parts of the same message by the resp. message contents. Verify that there are no dangling references and that the resulting document is a valid coverage by applying all tests required by this conformance class *multipart-coverage*.

- Otherwise, pass test.

Test passes if all detail checks pass.

Test Purpose: Requirement 45

Test method: In the coverage under test, verify:

- If it is encoded in a multipart message, verify that all references into subsequent parts are valid (i.e., no dangling links) in accordance with the container format used.

- Otherwise, pass test.

Test passes if all detail checks pass.

Test Purpose: Requirement 46

Test method: In the coverage under test, replace all references by the reference target (while decoding the target format appropriately). If no error occurs, perform tests of class *coverage* on the resulting coverage.

Test passes if all detail checks pass.

A.12 Conformance Test Class: *coverage-partitioning*

Test Purpose: Requirement 47

Test method: The coverage under test must pass all tests of class *coverage*.

Test passes if all detail checks pass.

Test Purpose: Requirement 48

Test method: Check that the coverage under test contains the information structures defined by this requirement. This involves checks against the complete UML model, including classes, attributes and their values, associations, multiplicities, and further constraints. Check that all necessary elements are present.

Test passes if all detail checks pass.

Test Purpose: Requirement 49

Test method: In the coverage under test, verify all `partition` references do not form a circle, neither through directly referencing itself nor indirectly through a circular reference chain.

Test passes if all detail checks pass.

Test Purpose: Requirement 50

Test method: In the coverage under test, verify for each sub-coverage referenced in a `partition`, that the super/sub-coverage CRS condition holds.

Test passes if all detail checks pass.

Test Purpose: Requirement 51

Test method: In the coverage under test, verify for each partition that all axes fulfil the constraint required.

Test passes if all detail checks pass.

Test Purpose: Requirement 52

Test method: In the coverage under test, verify for each partition that all axes fulfil the constraint required.

Test passes if all detail checks pass.

Test Purpose: Requirement 53

Test method: In the coverage under test, verify for each partition that the constraint required holds.

Test passes if all detail checks pass.

Test Purpose: Requirement 54

Test method: In the coverage under test, determine the set of all partition and value components. Verify that for any two components in this set their extent is disjoint for each range component.

Test passes if all detail checks pass.

Test Purpose: Requirement 55

Test method: In the coverage under test, verify:

- If there is at least one direct position in the domain set of the coverage for which no range value is stored: verify that a least one null value is defined in the range set.
- Otherwise, pass test.

Test passes if all detail checks pass.

Test Purpose: Requirement 56

Test method: In the coverage under test, verify that each partition's range type is a subset of the coverage under test, with any eventual range component name translation duly applied.

Test passes if all detail checks pass.

Test Purpose: Requirement 57

Test method: In the coverage under test, check whether there is a value missing for any range type component. If such a gap exists, verify that the range type has at least one null value defined for the range component in which this gap occurs.

Test passes if all detail checks pass.

A.13 Conformance Test Class: *container*

Test Purpose: Requirement 58

Test method: On the object under test, no tests are defined in this standard (structural constraints will be added by applications instantiating this scheme).

Test passes always.

Annex B
(non-normative)
Revision History

Date	Release	Author	Paragraph modified	Description
2015-07-23	1.1.0	Peter Baumann	All	Reworked for 1.1, based on 1.0
2015-11-22	1.1.0	Peter Baumann	Annex A	Added test suite
2016-05-24	1.1.0	Peter Baumann, Eric Hirschorn, Joan Maso	All	Reflected RFC comments and further stakeholder input; added JSON and JSON-LD/RDF
2016-11-27	1.1.0	Peter Baumann	Intro, Annex B	More background explanations, resolution of TC vote comments

Annex C
(non-normative)
Complete `CIS::AbstractCoverage` UML diagram collection

This Annex summarizes the UML diagrams presented in the normative part. For the reader's convenience, they are split into coverage types, coverage structure, and grid coverages.

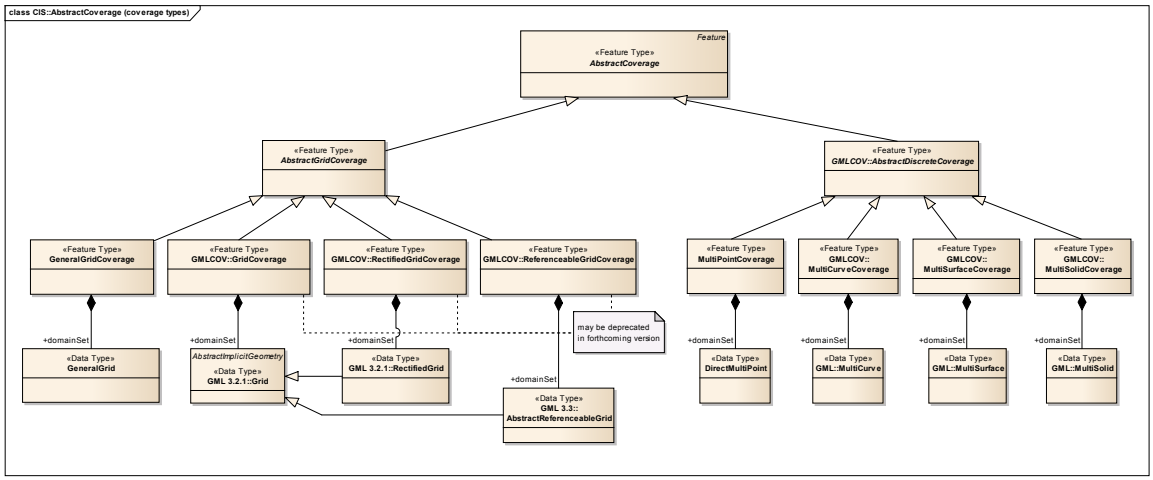

Figure 14: Coverage types

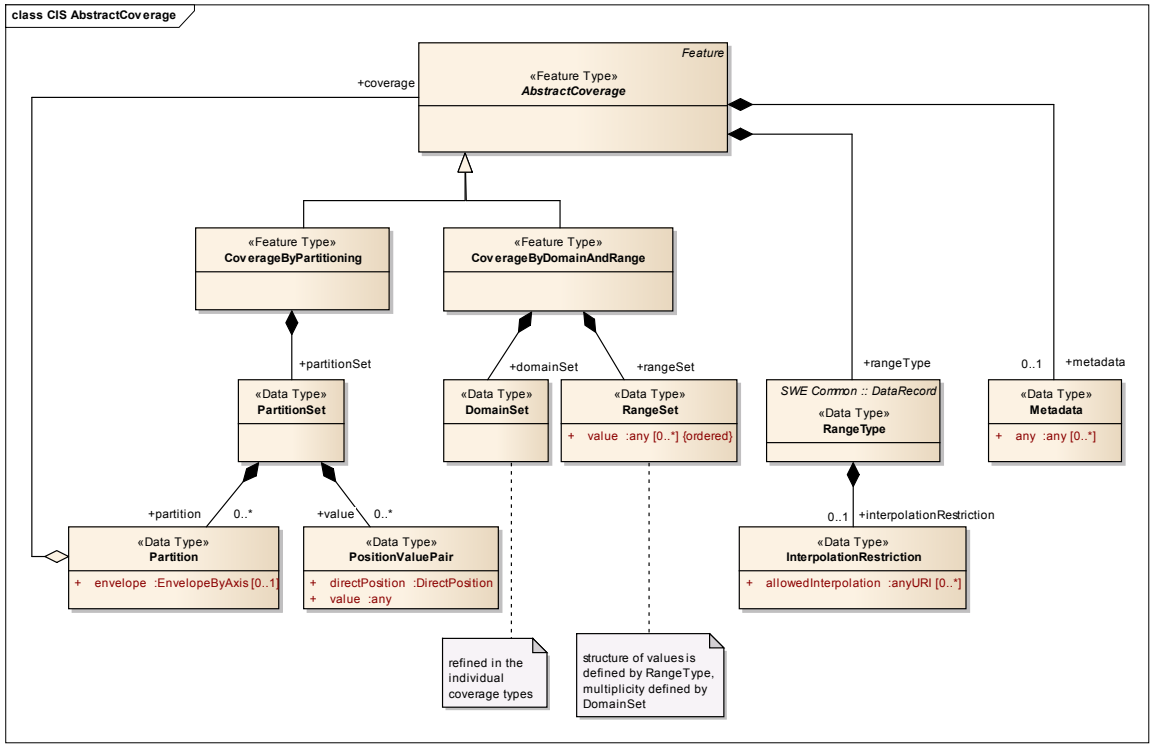

Figure 15: Coverage structure

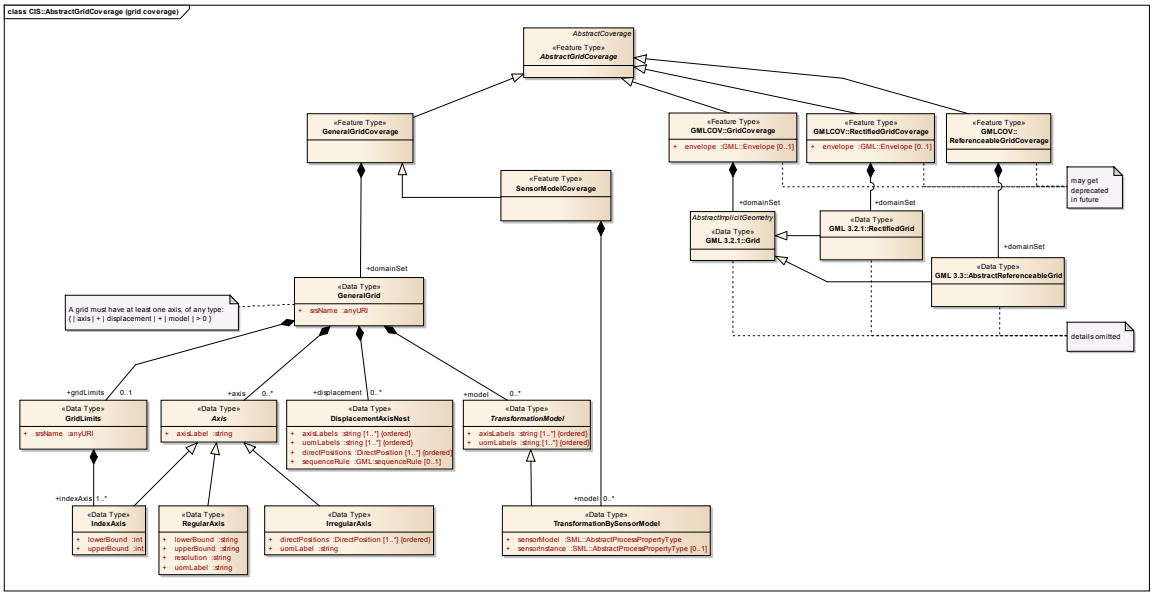

Figure 16: Grid coverages

Annex D
(non-normative)
Relation to Other Standards

D.1 Abstract Topic 6 / ISO 19123

ISO 19123 (which is identical to OGC Abstract Topic 6 [1]) defines an abstract coverage model. This model tentatively is general and abstract; as a consequence, different and incompatible coverage implementations are possible. The OGC Coverage Implementation Schema, therefore, complements it with a concrete coverage structure definition which can be conformance tested and allows for interoperable implementations.

The following table correlates ISO 19123 and GMLCOV/CIS 1.0 and CIS 1.1 coverage types. Note that continuous coverages are modelled separately in ISO 19123 whereas in CIS they consist of discrete coverages together with some interpolation method; typically, this will be specified in the interpolation method associated with the range type (starting CIS 1.1); alternatively, the coverage function can express interpolation (starting GMLCOV/CIS 1.0).

Those coverage types which represent point clouds and general meshes (i.e., all non-gridded coverages) are consistent with the modelling introduced by GML 3.2.1, Consequently, all corresponding ISO 19123 types are implemented by CIS types MultiPointCoverage, MultiCurveCoverage, MultiSurfaceCoverage, and MultiSolidCoverage.

ISO 19123:2003 coverage type	CIS coverage type
CV_Coverage	Coverage (CIS 1.0 or 1.1)
CV_DiscreteCoverage	Coverage (CIS 1.0 or 1.1)
CV_DiscretePointCoverage	MultiPointCoverage (CIS 1.0 or 1.1 with no interpolation method)
CV_DiscreteGridPointCoverage	GeneralGridCoverage (CIS 1.1 with no interpolation method) or GridCoverage / RectifiedGridCoverage / ReferenceableGridCoverage (CIS 1.0)
CV_DiscreteCurveCoverage	MultiCurveCoverage (CIS 1.0 or 1.1) with no interpolation method
CV_DiscreteSurfaceCoverage	MultiSurfaceCoverage (CIS 1.0 or 1.1) with no interpolation method
CV_DiscreteSolidCoverage	MultiSolidCoverage (CIS 1.0 or 1.1) with no interpolation method
CV_ContinuousCoverage	Coverage (CIS 1.0 or 1.1) with at least one interpolation method

CV_ContinuousQuadrilateralGridCoverage	GeneralGridCoverage (CIS 1.1) with at least one interpolation method
CV_ThiessenPolygonCoverage	MultiSurfaceCoverage (CIS 1.0 or 1.1) with at least one interpolation method
CV_HexagonalGridCoverage	GeneralGridCoverage (CIS 1.1) with at least one interpolation method
CV_SegmentedCurveCoverage	MultiCurveCoverage (CIS 1.0 or 1.1) with at least one interpolation method
CV_TINCoverage	MultiSurfaceCoverage (CIS 1.0 or 1.1) with at least one interpolation method

Table 24 Correspondence between ISO 19123 and CIS coverage types

D.2 GML 3.2.1

In GML 3.2.1 [2], all coverage types are derived from the abstract `Coverage` data type containing a `DomainSet` and a `RangeSet` component. The OGC coverage implementation schema, CIS, extends this with two additional components, a mandatory `RangeType` and optional `metadata`, an extensible slot for individual, application-specific metadata structures.

The GMLCOV/CIS 1.0 changes which apply over GML 3.2.1 are detailed in [5].

The following CIS 1.1 changes apply over GML 3.2.1 [2]:

- There are several extra concepts not present in GML 3.2.1, ranging from model (grid definition by axis rather than by grid type, SensorML domains, etc.) over representation (partitioning and geometry/value pairs) to encoding (addition of JSON and RDF).

- Coordinates are not required to be numeric only, but can also contain strings such as ISO 8601 date/timestamps or categorical values. This is instrumental for general multi-dimensional coverages.

- A point cloud coverage type, `MultiPointCoverage`, is provided which semantically is equivalent to GML 3.2.1 and GMLCOV/CIS 1.0, but allows string coordinates as described above.

  Note GMLCOV/CIS 1.0 coverage types `MultiCurveCoverage`, `MultiSurface-Coverage`, and `MultiSolidCoverage` are not addressed by CIS 1.1, the original GMLCOV/CIS 1.0 definitions remain valid.

D.3 GML 3.3

GML 3.3 [3] adds several grid types to GML 3.2.1. However, given the OGC modular specification rules these are not automatically available for GMLCOV/CIS 1.0. Further, these grid types resemble only special cases omitting, for example, combinations of regular and irregular axes in the same datacube. The CIS 1.1 model encompasses and generalizes GML 3.3. In the CIS 1.1 XML encoding, the GML 3.3 schema is included.

D.4 SWE Common

The `RangeType` element of a coverage describes the coverage's range set data structure (see Clause 6). This range value structure description is adopting the SWE Common [4] `DataRecord`.

D.5 Further Standards

The OGC standards WaterML 2 [OGC 10-126r4], TimeseriesML 1 [OGC 15-043rX], and OM-JSON [OGC 15-100r1] represent domain-specific standards for which the OGC Coverage Implementation Schema establishes a domain-neutral basic data structure which can be used whenever a coverage-like structure occurs; such standards, while retaining interoperability by using the common coverage model, will likely extend coverages with domain specific metadata, such as done in TimerseriesML.

