

# Chapter 15

## New Model for Geospatial Coverages in JSON: Coverage Implementation Schema and Its Implementation With JavaScript

**Joan Maso**  
*CREAF, Spain*

**Alaitz Zabala Torres**  
*Universitat Autònoma de Barcelona, Spain*

**Peter Baumann**  
*Jacobs University, Germany*

### **ABSTRACT**

*Map browsers currently in place present maps and geospatial information using common image formats such as JPEG or PNG, usually created from a service on demand. This is a clear approach for a simple visualization map browser but prevents the browser from modifying the visualization since the content of the image file represents the intensity of colors of each pixel. In a desktop GIS, a coverage dataset is an array of values quantifying a certain property in each pixel of a subdomain of the space. The standard used to describe and distribute coverages is called web coverage service (WCS). Traditionally, encoding of coverages was too complex for map browsers implemented in JavaScript, relegating the WCS to a data download, a process that creates a file that will be later used in a desktop GIS. The combination of a coverage implementation schema in JSON, binary arrays, and HTML5 canvas makes it possible that web map browsers can be directly implemented in JavaScript.*

## ***New Model for Geospatial Coverages in JSON***

### **INTRODUCTION**

In geospatial computer science geospatial information is mainly divided into two data models feature data (covering mainly vector data) and coverage data (covering mainly raster data). Feature data in JSON has been covered by the IETF RFC 7946 GeoJSON Format. This Chapter focuses on coverage data. Traditionally, aerial photographs, land cover maps, digital elevation models, etc have been encoded as a sequence of values in binary files, such as GeoTIFF. Geospatial Information Systems (GIS) required a better description for the spatio-temporal domain where these multidimensional arrays were situated (georeferenced) and about the exact meaning and restrictions of its values. The term “coverage” is defined in [ISO19123] as a family of data models based on a “feature that acts as a function to return values from its range for any direct position within its spatial, temporal or spatiotemporal domain” – in practice, spatio-temporal regular and irregular grids, point clouds, and general meshes. In other words, a coverage maps a distribution of space positions and time instants to a set of data values. For example, an UAV photograph can be modeled as a coverage that maps positions on the ground to the colors of the surface of the earth. A climate model maps a multidimensional grid (horizontal space coordinates, elevation, and time) to values of temperature, wind speed, humidity, etc.

Traditionally, encoding of coverages was too complex for web map browsers implemented in JavaScript, relegating the Open Geospatial Consortium (OGC) Web Coverage Service (WCS) standard to a data download; a process that creates a file that will be later used in a desktop GIS. To remedy this, the Chapter on hand presents an implementation that uses a combination of the Coverage Implementation Schema (CIS) in JSON, binary arrays, HTML5 canvas, and JSON styling rules, which makes possible that web map browsers can directly implement support to coverage visualization and analysis using only JavaScript code.

Based on the modern capabilities of the Web standards such as HTML5 and JavaScript, this Chapter describes the CIS standardized by the OGC which comes with a JSON encoding and compares it to a different recent initiative called CoverageJSON. Our modeling of JSON coverages, which is a main topic of this paper, is part of the OGC CIS version 1.1 standard.

### **BACKGROUND**

Coverages represent homogeneous collections of values located in space/time, such as spatio-temporal sensor, image, simulation, and statistics data. Common examples include measures by a static sensor of a variable over time (1-D time series), optical satellite imagery (2-D imagery), series of geometrically corrected satellite images (3-D x/y/t time series), a interpolation representing the temperature distribution of the atmosphere (x/y/z geophysical voxel models), and a simulation representing the evolution of the temperature of the atmosphere over time (4-D x/y/z/t models). Coverages encompass multi-dimensional regular and irregular grids, point clouds, and general meshes. [Baumann et al, 2017]

Often the word “coverage” is used a synonymous of “gridded data”, “raster data” or “imagery”. Even if the three expressions are examples of coverages, this is not the whole picture. For example, non-gridded data (like a river gauge time series) can also be modeled as a coverage. Generally, the concept of *coverages* encompasses spatio-temporal regular and irregular grids (both discrete and continuous), point clouds, and general meshes.

## ***New Model for Geospatial Coverages in JSON***

The OGC Geography Markup Language (GML) (Portele, 2007) mainly focuses on feature data but also provides foundational elements to describe coverages in XML. Following literally the common practices for feature data, it requires the creation of a GML Application Schema (and specialized form of XML Schema) for each coverage type. The definition of Application Schemas has some degrees of freedom and, if we combine this with the lack of easy to find GML Application Schemas catalogues, the risk is that descriptions of the same kind of information done by different organizations result in different data structures, preventing interoperability. It also makes the implementation of applications more difficult because software programmers need to anticipate all possible variations of the Application Schemas.

The Coverage Implementation Schema (CIS) (Baumann, et al. 2017) is an Open Geospatial Consortium (OGC) standard where a coverage model by establishing as a concrete, interoperable, conformance-testable single coverage structure. CIS 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”.

CIS allows for creating coverage descriptions of grid information encoded in common binary formats such as GeoTIFF, JPEG200 or NetCDF or in ASCII formats such as GML or JSON (Bray T., 2014). A coverage can be partitioned in more than one file. Coverages can be accessed through a variety of OGC services types, such as the Web Coverage Service (WCS) Standard suite (Baumann, 2017).

The services providing coverages were designed as downloading and server-side processing and analytics services capable of providing excerpts from coverages for analysis in your favorite GIS, RS or modeling tool. They were not originally designed for visualization and fast analysis in a web browser. In the last part of this Chapter we discuss how to use CIS encoded in JSON, combined with new functionalities of HTML5 such as raw binary arrays to communicate data in ways that can be consumed by modern web browsers directly (Maso et al 2018).

## **COVERAGE IMPLEMENTATION SCHEMA**

The CIS defines a coverage as the combination of different *elements* that together from the data model. For example, in a grid coverage, the most common elements are: *domainSet*, *rangeSet*, *rangeType* and metadata. A coverage is defined by the existence of one or more axes in space/time or with “abstract” axes such as spectral range. In a *Grid Coverage*, on top of these axes, we define a multidimensional grid by specifying a set of points (or cells) in this grid that defines a *domainSet* in multidimensional space. Some *Discrete Coverage* specialization has no grid but a list of geometrical objects presenting positions in a multidimensional space. In a *Grid Coverage*, each grid cell will receive a value, and in a generalized Discrete Coverage each geometric object will receive a value which can be atomic or a composite record. All values share some common characteristics and limitations that are documented in a *rangeType*; for example, one of the most useful characteristic is the data type (e.g. floating point numbers), possibly with limits on the interval of values occurring in the coverage. Finally, the actual list of values that populates the grid or the geometrical objects is defined in the *rangeSet*. Additional *metadata* might complete the set, holding arbitrary ancillary information. *DomainSet* and *rangeType* are commonly present in a CIS document (that might be encoded in XML or in JSON), while the *rangeSet* list of values can be stored within this encoding, or alternatively can be referenced by the CIS “root” document and stored in some other format, such as a well known binary file format such as a NetCDF, a GeoTIFF or a JPEG2000 or can be simply stored in a RAW format as a sequence of binary values of a fixed size. If that is the case, the CIS “root” document is normally a short text document while the binary file containing the *rangeSet*

### New Model for Geospatial Coverages in JSON

is commonly a big file. Following the CIS standard, these files are combined through some “container format” such as MIME attachments (CIS 1.0) or any other suitable format like ZIP, SAFE (<http://earth.esa.int/SAFE>), etc. (CIS 1.1).

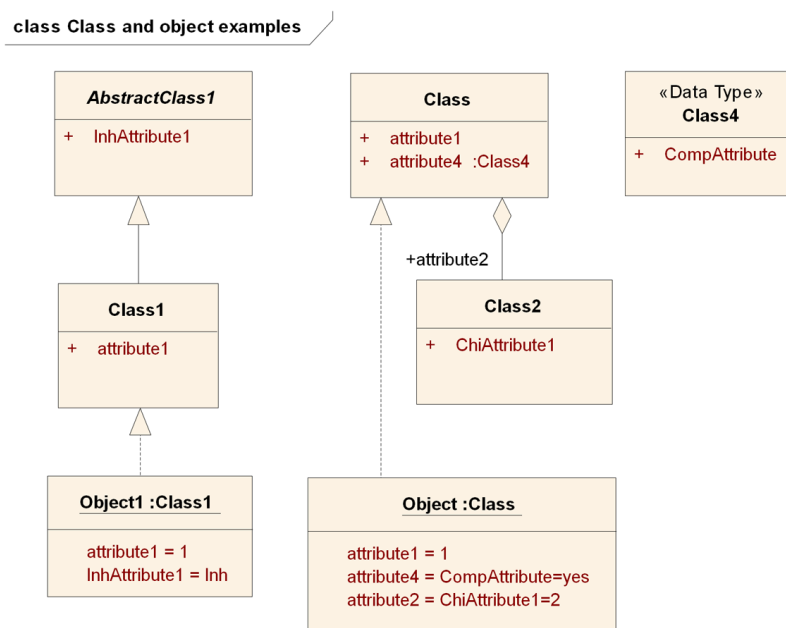
In this Chapter we will detail some examples of coverages and the corresponding JSON encoding. An XML and RDF encoding have also been defined by the OGC CIS 1.1 standard but its description will not be part of this book Chapter – see Bauman (2017b) for more details. The reader can refer to the OGC Schema repository (in <http://schemas.opengis.net/cis/json/examples-1.1>) to find the examples as JSON files (as well as the same examples in XML encoding in <http://schemas.opengis.net/cis/gml/examples-1.1>).

### UML Models Used in This Chapter

The CIS model is defined independently from the final encoding in UML class and object diagrams (Benson, 2016). Class diagrams are useful to describe the data types and relations among them and object diagrams are useful to show concrete examples. We limit to a set of characteristics in class and object diagrams exemplified in Figure 1.

Objects are instantiated from classes and values are assigned to their attributes. A class can be extended into another class by specifying new attributes. The extended class is related to the previous one by a *generalization* relation, represented by a continuous line with a triangle in the more general class extreme (see Class1 and AbstractClass1 in the diagram). Attribute types are specified as simple types or as complex types. Complex types can be defined as an independent class with *Data Type* stereotype and reused as many times as needed (see Class4 and attribute4 in the diagram). Complex types can also

Figure 1. Class, object and relation UML diagram



## New Model for Geospatial Coverages in JSON

be defined on the fly by an *association* or a *composition* relation where an attribute name is used as the source name of the association or the composition (see *attribute2* associated to *Class* and defined as *Class2* in the diagram). There is no practical difference between the two approaches but the later is uses as much as possible for clarity (see how the *Object* has both complex type approaches in the diagram).

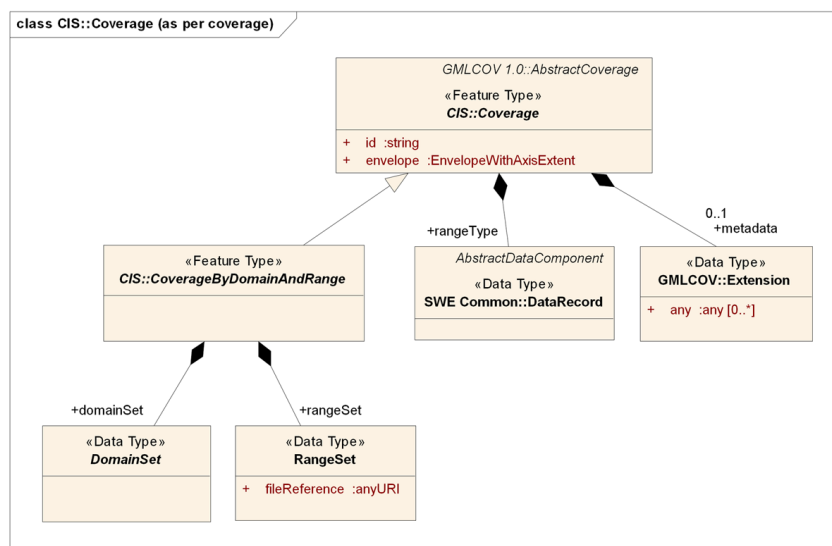
These diagrams follow the ISO TC211 Best Practices for UML diagram design (Jetlund K, 2015). The only exception is the *realization* relation. Realization is used between classes of different levels of abstraction or between a class and interfaces. In this chapter we are using the *realization* with a slightly different semantics to relate classes and their instantiation in objects (see *Object1* and *Class1* in the figure 1). We find this informative in diagrams showing concrete objects with attribute values that are implementing the related class.

A coverage can have several attributes including an id, an envelope, a domainSet, a rangeSet a rangeType an 0 or more metadata (Figure 2). In the following subsections we will review these and other options that the CIS model can offer.

### Rules for Generating JSON Encoding From UML Based on the CIS Schema

A set of generic rules to transform UML class models into JSON Schemas and encodings was proposed the OGC Testbed 12 (Maso, 2017). One of the innovations presented in this Chapter is to apply them to generate JSON encodings and JSON Schemas based on the UML class model for the CIS. This work has become the JSON encoding part of CIS 1.1 standard recently approved by the OGC. A coverage is the root of the JSON files presented in this chapter. In JSON there is no name for the root element but the first attribute of the root object is the type of object: “CoverageByDomainAndRangeType”. The general rules for transforming UML Class diagrams into JSON Schemas and objects can be summarized as follows: Each attribute in the UML model becomes an extra attribute of a JSON object. Each *association* or *composition* in the UML model becomes an attribute in the JSON model (that will be of

Figure 2. AbstractCoverage structure in CIS  
 Source: OGC 09-146r6



## ***New Model for Geospatial Coverages in JSON***

JSON “object” type). Simple attributes types in the model are mapped to the closest simple type in JSON (e.g. a *floating point* is encoded as a *number*). An element with multiplicity more than one becomes a JSON array. Attributes in the UML classes that are complex classes are also encoded as objects with the name of the class as a value of the “type” attribute. In case a class is a generalization of another, the most concrete one is used as “type” attribute value. For convenience, each object receives a unique identifier. The inclusion of id and type as properties of all JSON objects rules were designed to ensure proper compatibility with JSON-LD (Sporny, et al. 2014). This book Chapter presents these rules into practice in the examples for the CIS 1.1 developed in the next subsections. It is worth mentioning that other authors have made efforts to go in the opposite direction: the creation UML models for JSON documents (Izquierdo, 2016).

### **Grid Coverages**

One of the most common coverages is the gridded coverage type. Gridded coverages have a grid as their domain set describing the direct positions in multi-dimensional coordinate space. The geometrical distribution of the values in a coverage encoded in a domainSet can have a regular or an irregular distribution. A raw image that has been captured from a satellite (sometimes called “swath data”) is described by an irregular grid, because the values recorded are subjected to geometrical deformations of the optical system and to the effects of the variations on the elevation of the terrain. By executing a process called geometrical correction, deformations in the structure of the irregular grid can be determined and it can be compensated and data can be remapped into a regular grid. Spatial agencies and cartographic institutes distributing imagery provide regular grids that have already been rectified.

### **Regular Grid**

Gridded coverages have a grid as their domain set describing the direct positions in multi-dimensional coordinate space, depending on the type of the grid. Here we describe the regular grids described by the axes they are composed of. Positions are defined by the intersections of the spaces defined in each axis. Axes can be defined as index axes or regular axes. An index axis allows only integer coordinates with spacing (“resolution”) of 1. In contrast, regular axes can express a sampling common distance, i.e.: resolution, as a part of the axis definition (Figure 3).

In figure 4 we see a representation of a regular grid formed by indexAxis defining 9 cells by the intersection of 3 spaces in each axis.

Figure 5 illustrates the case of an image that is defined in a domain of 2 index axes both setting 3 steps, configuring grid of  $3 \times 3 = 9$  cells. The use of index axes makes sense in an image has no associated any Coordinate Reference System (CRS) that places the image in the Earth, and cells are addressed by counting cells in the grid in both dimensions.

JSON is not a class based encoding because it does not natively incorporate the definition of concrete complex data *types*. Nevertheless, it permits construction of complex *objects* on-the-fly. This means that only the lower part of the model in figure 5 can be directly encoded in JSON. We defined objects that contain a property “type” that reproduces the class name they are based on. The following code is the transcription in JSON of the Figure 4. The rangeType part is included for completeness but its content is not reproduced because it will be described later in this chapter.

**New Model for Geospatial Coverages in JSON**

Figure 3. GeneralGridCoverage structure as per grid-regular  
 Source: OGC 09-146r6

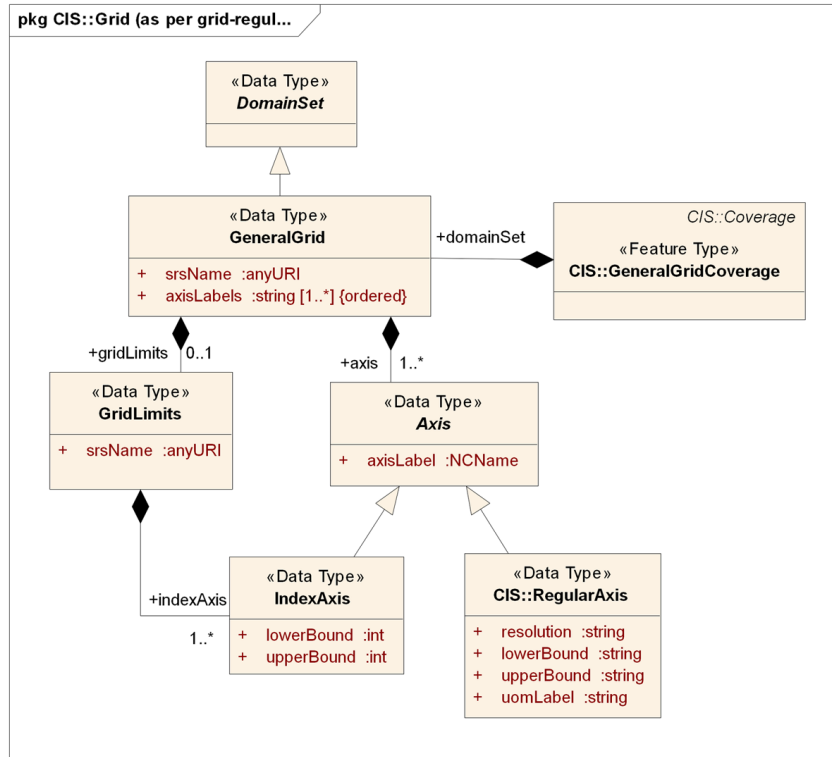
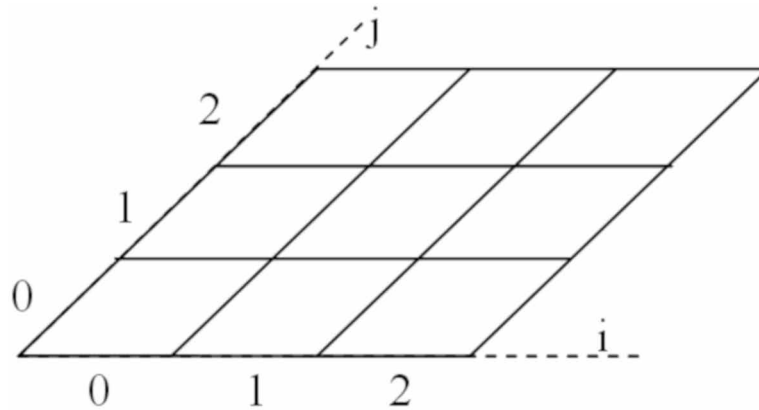


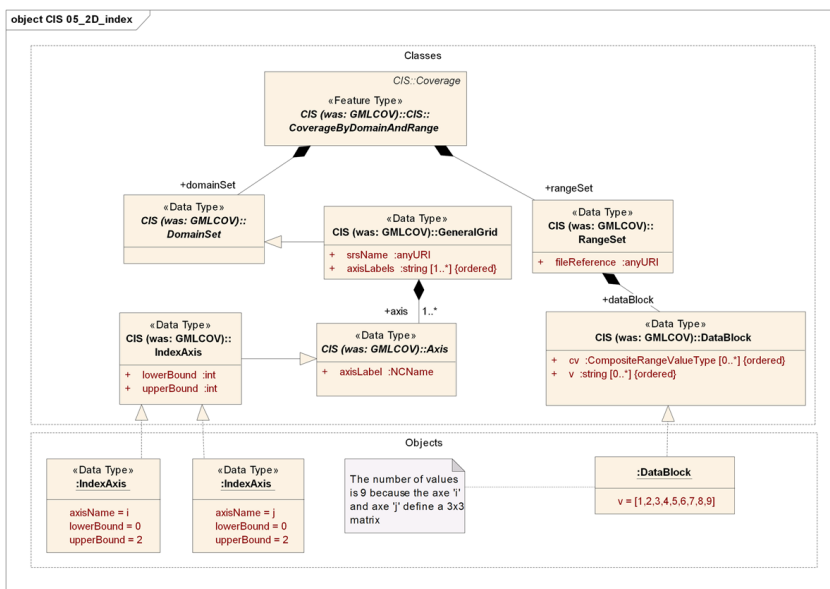
Figure 4. Representation of a 2D coverage with IndexAxis



```
{
  "type": "CoverageByDomainAndRangeType",
  "id": "examples:CIS_05_2D",
  "domainSet": {
    "type": "GeneralGridType",
```

## New Model for Geospatial Coverages in JSON

Figure 5. Description of a coverage with IndexAxis



```

    "id": "examples:CIS_DS_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": {
    "type": "RangeSetType",
    "id": "examples:CIS_RS_05_2D",
    "dataBlock": {
      "id": "examples:CIS_RS_DB_05_2D",
      "type": "VDataBlockType",
      "values": [1,2,3,4,5,6,7,8,9]
    }
  }
}

```



### New Model for Geospatial Coverages in JSON

```

        }
    },
    "rangeType": {
        ...
    }
}
    
```

Regular grids can also be described by regular axis where cells are defined as regularly spaced by a common distance, called resolution. In practice, this means that by specifying the start value, the end value and the resolution, all positions on the axes are defined.

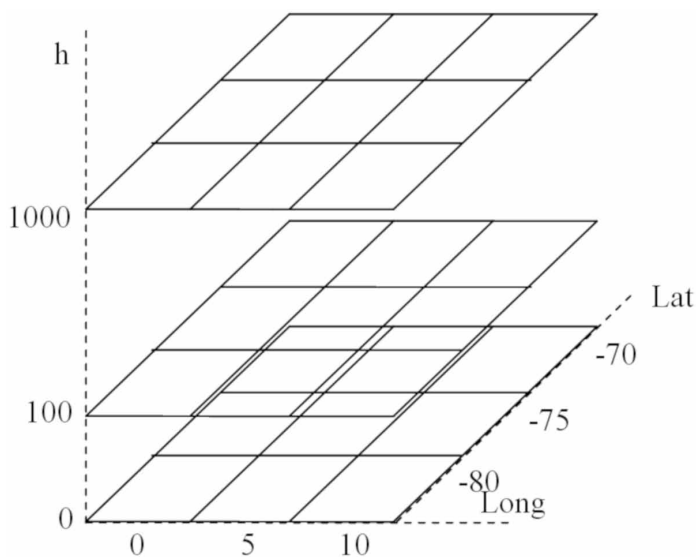
### Irregular Grids

An irregular grid can be defined through other types of axes. We will start by introducing the irregular axis that 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 (h axis in figure 6).

Figure 7 illustrates an image that is defined in a domain of 3 dimension represented by 2 regular axes and one irregular axis, configuring grid of  $3 \times 3 \times 3 = 27$  cells. The domain is associated to a CRS that places the “cube” in the Earth. To map the actual cells with the domain set, a grid limits 3D data structured of index axes is also defined. The 27 values are provided in the rangeSet.

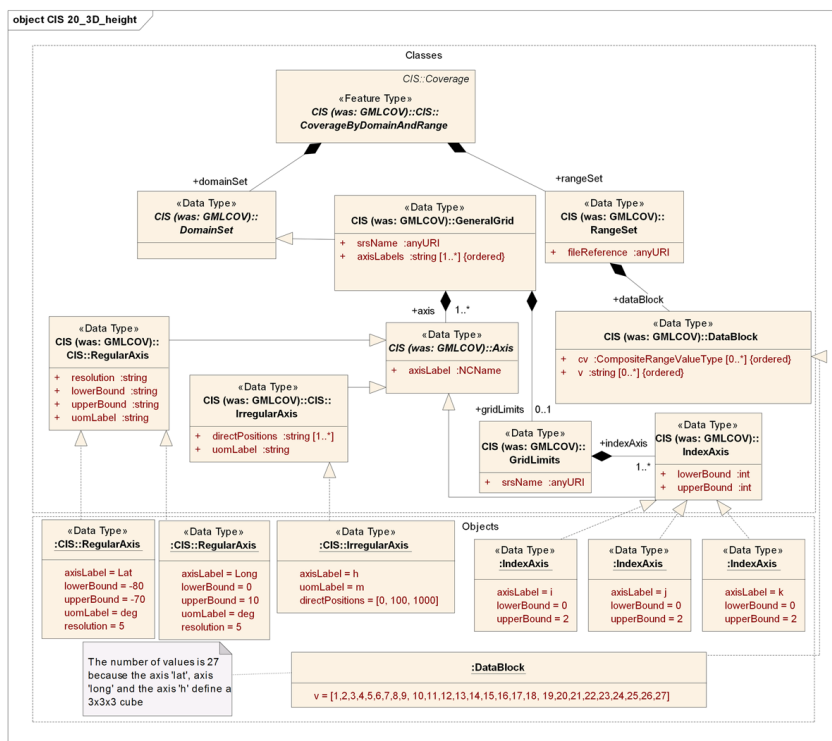
The following code is the transcription in JSON of the Figure 7.

Figure 6. A 3D coverage with 2 regularAxis defining cells in latitude and longitude and 1 irregularAxis representing 3 levels of elevation.



## New Model for Geospatial Coverages in JSON

Figure 7. Description of a 3D coverage with two RegularAxis and one IrregularAxis



```

{
    "type": "CoverageByDomainAndRangeType",
    "examples:CIS_20_3D",
    "GeneralGridType",
    "DS_20_3D",
    "crs/EPSG/0/4979",
    "h"],
    "RegularAxisType",
    "examples:CIS_DS_GG_LAT_20_3D",
    "isLabel": "Lat",
    "-80",
    "Label": "deg",
    5
}, {
    "type": "RegularAxisType",
    "examples:CIS_DS_GG_LONG_20_3D",
    "label": "Long",
    0,
    "Label": "deg",
    "resolution": 5
}, {
    "type": "IrregularAxisType",
    "examples:CIS_DS_GG_H_20_3D",
    "axisLabel":

```

### New Model for Geospatial Coverages in JSON

```

    "h",
    ordinate": [0, 100, 1000]
    Limits": {
        "GridLimitsType",
        DS_GG_GL_20_3D",
        opengis.net/def/crs/OGC/0/Index3D",
        isLabels": ["i", "j", "k"],
        dexAxis": [{
            dexAxisType",
            "examples:CIS_DS_GG_GL_I_20_3D",
            isLabel": "i",
            Bound": 0,
            2
        }, {
            "type": "IndexAxisType",
            "examples:CIS_DS_GG_GL_J_20_3D",
            isLabel": "j",
            Bound": 0,
            2
        }, {
            "type": "IndexAxisType",
            "examples:CIS_DS_GG_GL_K_20_3D",
            isLabel": "k",
            Bound": 0,
            2
        }
    ]
    eSet": {
        "type": "RangeSetType",
        "examples:CIS_RS_20_3D",
        Block": {
            "id": "examples:CIS_RS_DB_20_3D",
            "type": "VDataBlockType",
            "values": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
        }
    }
    "rangeType":
    {
        ...
    }
    "uomLabel": "m",
    "grid-
    "co-
    "type":
    "id": "examples:CIS_
    "srsName": "http://www.
    "ax-
    "in-
    "type": "In-
    "id":
    "ax-
    "lower-
    "upperBound":
    "id":
    "ax-
    "lower-
    "upperBound":
    "id":
    "ax-
    "lower-
    "upperBound":
    "rang-
    "id":
    "data-
    "id": "examples:CIS_RS_DB
    "type": "VDataBlock-
    "values": [1,2,3,4,5,6,7,8,9,
    17,18,
    }
    },
    "rangeType":
    {
        ...
    }
    
```

### Displaced Grids

A displaced grid 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 neighbourhood relationship is retained. can be defined as by some other types of axis (Figure 8).

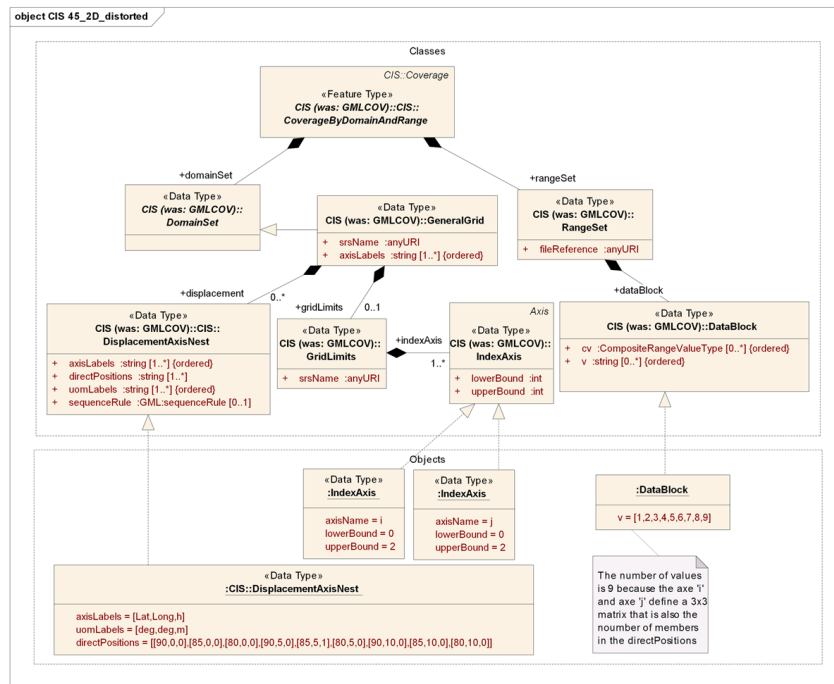
Figure 9 illustrates the case of an image that is defined in a domain of 3 dimension represented by a displaced nest, configuring grid of 9 cells defined by direct positions in the space. This nest is mapped to a 2D grid. The 3D grid is mapped into the cells of grid of 2D index axes with 3 positions each. The 9 values are provided in the rangeSet.

### New Model for Geospatial Coverages in JSON

Figure 8. Representation of a 2D coverage defining a displaced grids of 2 axes.



Figure 9. Description a 3D coverage in a displaced grid nest



The following code is the transcription in JSON of the Figure 9.

```

{
  "type": "CoverageByDomainAndRangeType",
  "id": "examples:CIS_45_2D",
  "domainSet": {
    "type": "GeneralGridType",
    "id": "examples:CIS_DS_45_2D",
    "srsName": "http://www.opengis.net/def/crs/EPSG/0/4979",

```

### New Model for Geospatial Coverages in JSON

```
    "axisLabels": ["Lat", "Long", "h"],
    "displacement": {
      "type": "DisplacementAxisNestType",
      "id": "examples:CIS_DS_GG_D_45_2D",
      "axisLabels": ["Lat", "Long", "h"],
      "uomLabels": ["deg", "deg", "m"],
      "coordinates": [[90, 0, 0], [85, 0, 0], [80, 0, 0],
[90, 5, 0], [85, 5, 1], [80, 5, 0], [90, 10, 0], [85, 10, 0], [80, 10, 0]]
    },
    "gridLimits": {
      "type": "GridLimitsType",
      "id": "examples:CIS_DS_GG_GL_45_2D",
      "srsName": "http://www.opengis.net/def/crs/OGC/0/In-
dex1D",
      "axisLabels": ["i", "j"],
      "axis": [{
        "type": "IndexAxisType",
        "id": "examples:CIS_DS_GG_GL_I_45_2D",
        "axisLabel": "i",
        "lowerBound": 0,
        "upperBound": 2
      }, {
        "type": "IndexAxisType",
        "id": "examples:CIS_DS_GG_GL_J_45_2D",
        "axisLabel": "j",
        "lowerBound": 0,
        "upperBound": 2
      }
    ]
  },
  "rangeSet": {
    "type": "RangeSetType",
    "id": "examples:CIS_RS_45_2D",
    "dataBlock": {
      "id": "examples:CIS_RS_DB_45_2D",
      "type": "VDataBlockType",
      "values": [1,2,3,4,5,6,7,8,9]
    }
  },
  "rangeType": {
    ...
  }
}
```

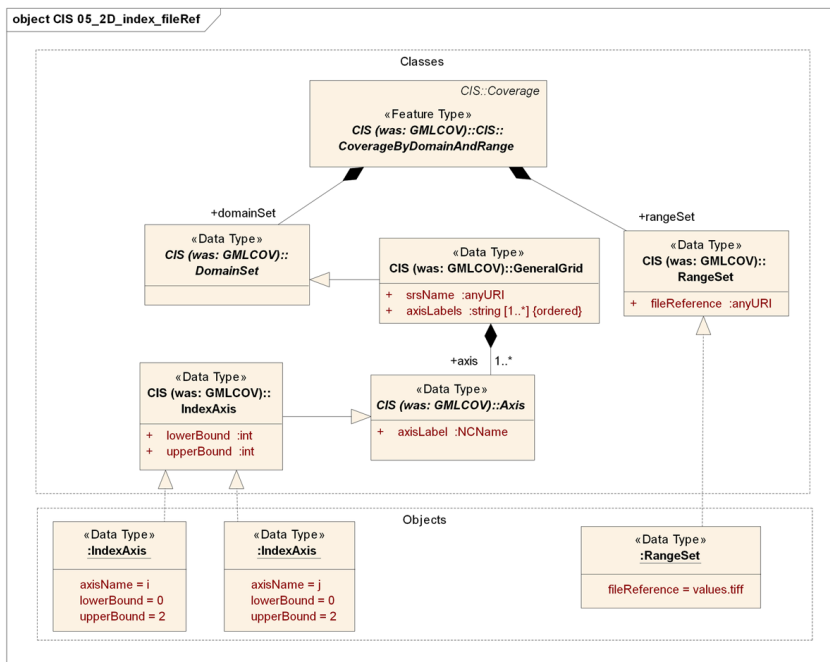
## New Model for Geospatial Coverages in JSON

### RangeSet

The range set contains the actual values associated with one direct position as defined in the domainSet. There are as many values as direct positions in the domainSet and in the same order that they have been defined. In the examples provided so far, the values in the rangeSet are included directly in the JSON encoding as an array of values in the dataBlock object but this is not a common in practice. Coverages that describe a certain area of the space at a reasonable level of detail require a large array in the rangeSet. If we take the example of the Shuttle Radar Topographical Mission (SRTM), the elevations of the Earth were measured every 3 seconds of arc (approximately every 100m) almost for the entire planet. Considering that there are 20 values every minute of arc, to cover the Earth 77760000 values as needed. Having this list of values as a text array will make it too long. Commonly you would like to use only a part of it but in text encoding low numbers are encoded in less characters that large numbers, making random access to a part of the data impossible without sequentially having to read all previous values (Witayangkurn, 2012). For these reasons, coverages are traditionally stored in binary files that are more compact and can be randomly addressed. In addition, these files have powerful compression algorithms that make them even more compact. The CIS encoding provide an alternative way to link to external binary files.

Figure 10 reproduces the same example illustrated in Figure 5 but replacing the list of values in the rangeSet by a reference to a external file; in this case in tiff format.

Figure 10. RangeSet as a reference to a file.



**New Model for Geospatial Coverages in JSON**

**Coverage by Partitioning**

Sometimes, to distribute or maintain the coverage in a single file is impractical. Examples are coverages that are tiled into smaller pieces (figure 11 shows one 1x1 degree fragment of the SRTM) or coverages representing temporal series where each time slide is stored in an independent file.

CIS provides a way to encode coverages that has its values arranged in more than one piece (more than one file) called coverageByPartitioning. Each partition is a fragment of a coverage that has the definition of the domainSet (the axes) replicated for each partition (Figure 12).

Figure 13 illustrates the case of an image that is defined in a domain of 3 dimension represented by 2 regular axes and one irregular axis. Each tile slide is defined by a partition that is stored in a different file.

The following code is the transcription in JSON of the Figure 13.

```
{
  "type": "CoverageByPartitioningType",
  "id": "examples:CIS_50_3D",
  "partitionSet": {
    "type": "PartitionSetType",
    "id": "examples:CIS_PS_50_3D",
    "partition": [{
      "type": "PartitionRefType",
      "id": "examples:CIS_PS_P1_50_3D",
```

Figure 11. Example of a 1x1 degree fragment of a SRTM v3 over Spain distributed as a single file  
 Source: <https://earthexplorer.usgs.gov/>



### New Model for Geospatial Coverages in JSON

Figure 12. Representation of a 3D coverage with 2 regularAxis defining cells in latitude and longitude and 1 irregularAxis representing 2 levels of time. Each time slide is defined in a different file

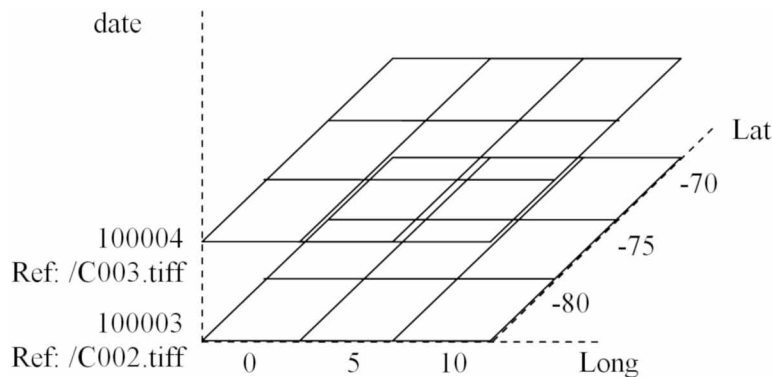
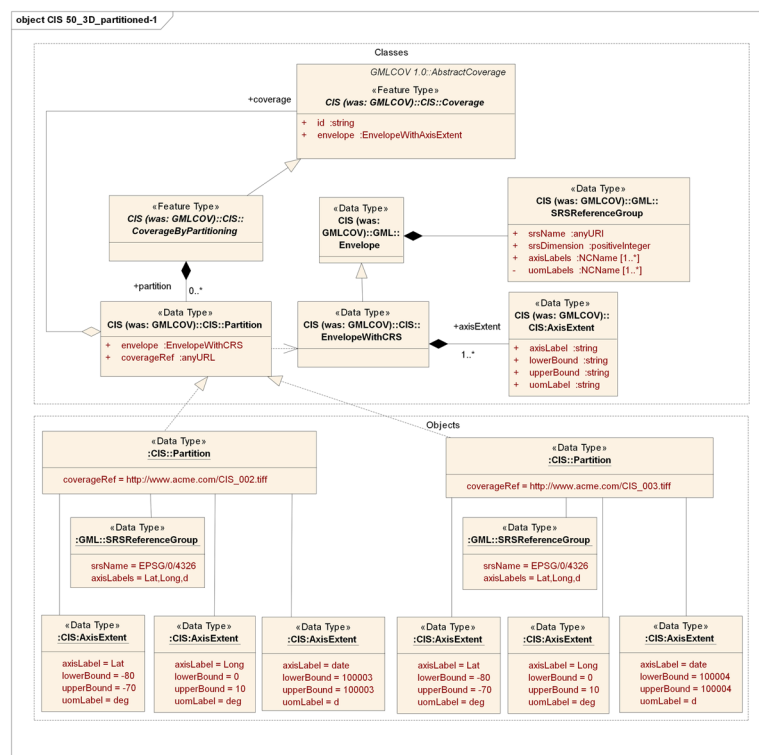


Figure 13. Description of a 3D coverage that is partitioned in 2 partitions stored in different files each one representing a different instant in time.



```

"envelope": {
    "type": "EnvelopeByAxisType",
    "id": "examples:CIS_PS_P1_E_50_3D",
    "srsName": "http://www.opengis.net/def/crs/
    EPSG/0/4326",

```



### New Model for Geospatial Coverages in JSON

```
    "axisLabels": ["Lat", "Long", "d"],
    "axis": [{
      "type": "AxisExtentType",
      "id": "examples:CIS_PS_P1_LAT_50_3D",
      "axisLabel": "Lat",
      "lowerBound": -80,
      "upperBound": -70,
      "uomLabel": "deg"
    }, {
      "type": "AxisExtentType",
      "id": "examples:CIS_PS_P1_
LONG_50_3D",
      "axisLabel": "Long",
      "lowerBound": 0,
      "upperBound": 10,
      "uomLabel": "deg"
    }, {
      "type": "AxisExtentType",
      "id": "examples:CIS_PS_P1_D_50_3D",
      "axisLabel": "date",
      "lowerBound": 100003,
      "upperBound": 100003,
      "uomLabel": "d"
    }
  ]
}, {
  "coverageRef": "http://www.acme.com/CIS_002.tiff"
}, {
  "type": "PartitionRefType",
  "id": "examples:CIS_PS_P2_50_3D",
  "envelope": {
    "type": "EnvelopeByAxisType",
    "id": "examples:CIS_PS_P2_E_50_3D",
    "srsName": "http://www.opengis.net/def/crs/
EPSG/0/4326",
    "axisLabels": ["Lat", "Long", "d"],
    "axis": [{
      "type": "AxisExtentType",
      "id": "examples:CIS_PS_P2_LAT_50_3D",
      "axisLabel": "Lat",
      "lowerBound": -80,
      "upperBound": -70,
      "uomLabel": "deg"
    }, {
      "type": "AxisExtentType",
```

### New Model for Geospatial Coverages in JSON

```

LONG_50_3D",
    "id": "examples:CIS_PS_P2_
    "axisLabel": "Long",
    "lowerBound": 0,
    "upperBound": 10,
    "uomLabel": "deg"
  }, {
    "type": "AxisExtentType",
    "id": "examples:CIS_PS_P2_D_50_3D",
    "axisLabel": "date",
    "lowerBound": 100004,
    "upperBound": 100004,
    "uomLabel": "d"
  }
],
"coverageRef": "http://www.acme.com/CIS_003.tiff"
}
},
"rangeType": {
  ...
}
}

```

### Discrete Coverages

The OGC coverage concept recognizes that beyond grids there are further kinds of coverages. The coverages that are not described by a grid are called *discrete coverages*; instead of containing points sitting on some grid they are composed of a bundle of some common vector geographical primitive of a specific dimension. In order of increasing dimensions, the domainSet can consist of a set of points (MultiPointCoverage), of curves (MultiCurveCoverage), of surfaces (MultiSurfaceCoverage), or of solids (MultiSolidCoverage). One recent application of this type of coverages that has become popular in recent years is the representation of a LIDAR campaign as a point cloud (Figure 14). While point clouds consist of a set of points like gridded data, the former do not have to be aligned to some grid as the latter do.

Figure 15 illustrates the case of a point cloud that is defined by a direct multi point of 3 dimensions. Each point has 3 values associated in the rangeSet.

The following code is the transcription in JSON of the Figure 15.

```

{
  "type": "MultiPointCoverageType",
  "id": "examples:CIS_90",
  "domainSet": {
    "type": "DirectMultiPointType",
    "id": "examples:CIS_DS_DMP_90",

```

**New Model for Geospatial Coverages in JSON**

Figure 14. Representation of a 3D coverage with a point cloud. The points are not arranged in a grid.

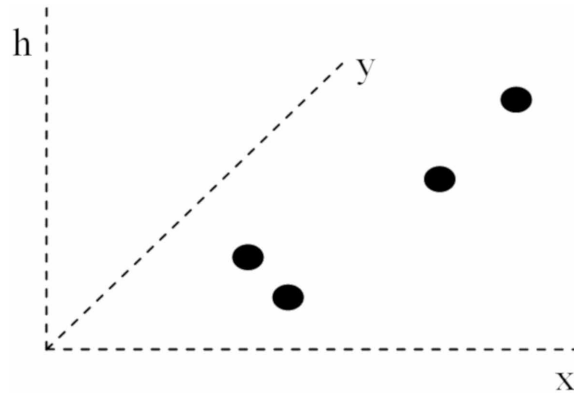
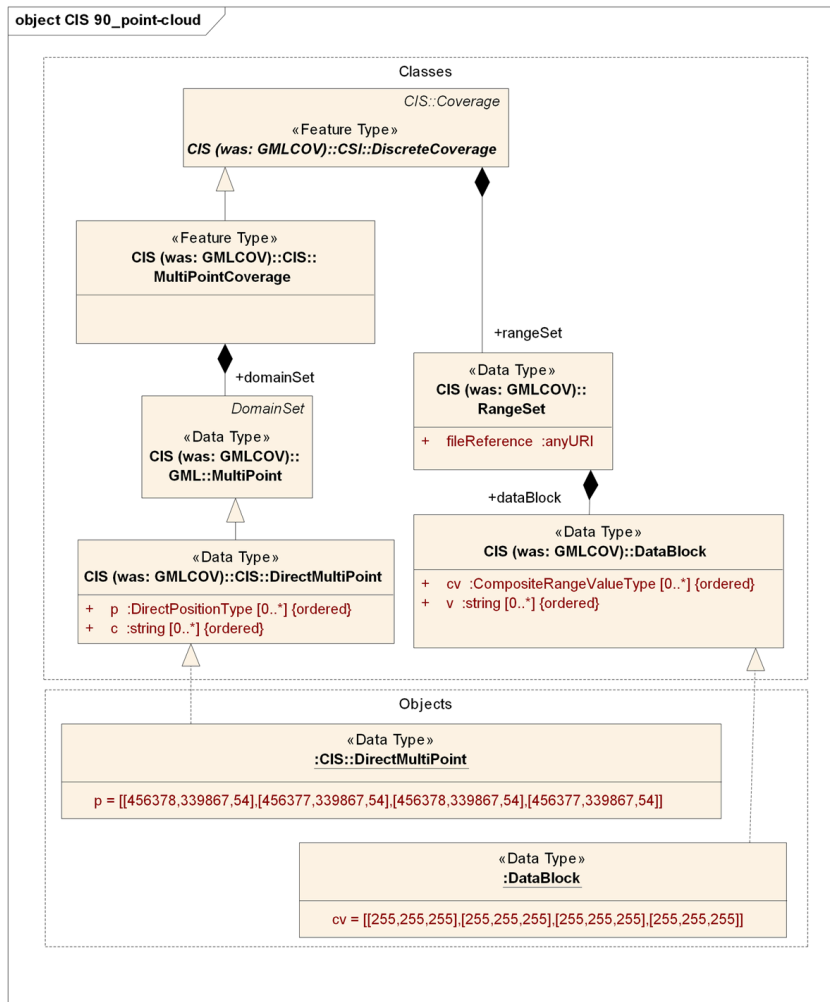


Figure 15. Description of a 3D point cloud that is defined by a direct multi point of 3 dimensions. Each point has 3 values associated in the rangeSet.



### New Model for Geospatial Coverages in JSON

```

        "coordinates": [[456377.56, 339867.25, 53.95],
                        [456377.50, 339867.22,
53.96],
                        [456377.56, 339867.22,
53.95],
                        [456377.50, 339867.22,
53.96]]
    },
    "rangeSet": {
        "type": "RangeSetType",
        "id": "examples:CIS_RS_90",
        "dataBlock": {
            "id": "examples:CIS_RS_DB_90",
            "type": "VDataBlockType",
            "values": [[255,255,255], [255,255,255], [255,255,255],
[255,255,255]]
        }
    },
    "rangeType": {
        ...
    }
}

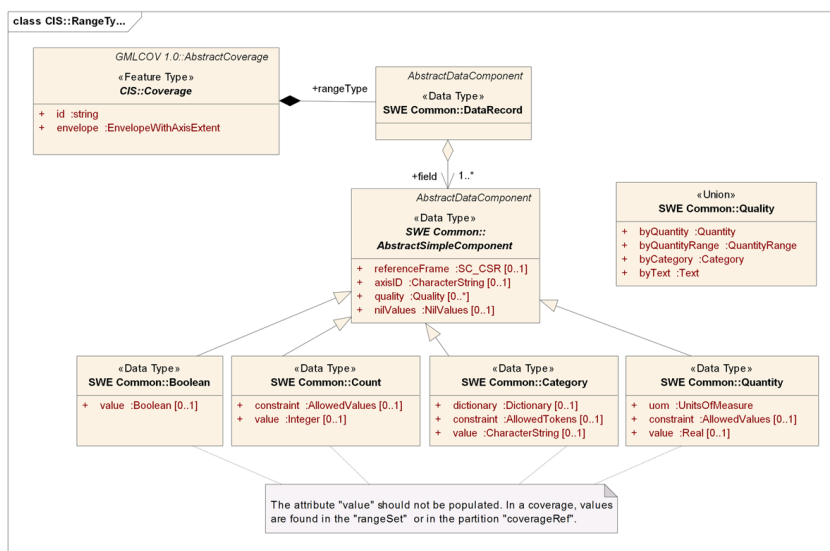
```

### RangeType

Continuing with the example of satellite imagery, the values associated with each cell of the grid are related to the intensity of light received by the satellite at different frequencies. They do not directly represent any magnitude of the Earth surface or the atmosphere because reflections and absorptions of the light in its path has distorted them. Fortunately, there are ways to compensate for does effects. In the part, this compensation was done by the final user but now remote sensing agencies are doing the correction by themselves distributing what is called “analysis ready data”. Whatever the process the new values gain semantics and this semantics needs to be recorded in the CIS model. The rangeType element of a coverage describes the semantics and type restrictions of the coverage’s range set data structure. In the case of imagery, one expected information included is 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. The rangeType structure is based on the SWE Common DataRecord (Robin, 2011) allowing mapping sensor observations to coverages without loss of semantics. Figure 16 describes the values associated to the grid or the features. This way we expect that all cells of the grid or features to be of the same data type and have the same restrictions. The descriptions include the name of the properties, the null values (also called *nilValues*; that are values reserved to represented cells that were not measured or had a out of range value), the allowed values (*allowedValues*), the data quality and the value units (*uom*). The original data types (e.g. SWE Common Quantity), as imported from the SWE Common standard, also contain an attribute for specifying a constant *value*. This *value*

### New Model for Geospatial Coverages in JSON

Figure 16. Description of the rangeType part of the CIS model.



is not going to be used by the CIS; as we have reiteratively explained, values are expected to be found in the coverage rangeSet.

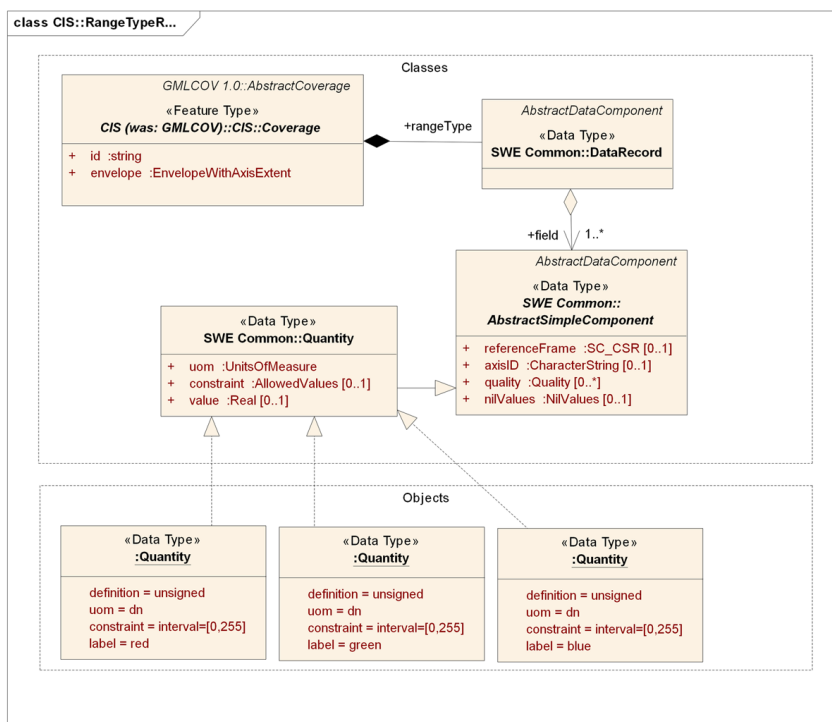
Figure 17 illustrates the case of a coverage where each grid cell or discrete coverage feature receives 3 values. The coverage is defining read, green and blue intensities in *digital numbers* that has a depth of 256 possibilities of brightness.

The following code is the transcription in JSON of the Figure 17. The rangeSet part was already exemplified previously and it is included only for completeness.

```
{
  "domainSet": {
    ...
  },
  "rangeSet": {
    "type": "RangeSetType",
    "dataBlock": {
      "type": "VDataBlockType",
      "values": [[255,255,255], [255,255,255], [255,255,255],
[255,255,255]]
    }
  },
  "rangeType": {
    "type": "DataRecordType",
    "field": [{
      "type": "QuantityType",
      "label": "red",
```

### New Model for Geospatial Coverages in JSON

Figure 17. Description of the rangeType formed by 3 channels labelled red, green and blue.



```

        "definition": "ogcType:unsigned",
        "uom": {
            "type": "UnitReference",
            "code": "dn"
        },
        "constraint": {
            "type": "AllowedValues",
            "interval": [0, 255]
        }
    }, {
        "type": "QuantityType",
        "label": "green",
        "definition": "ogcType:unsigned",
        "uom": {
            "type": "UnitReference",
            "code": "dn"
        },
        "constraint": {
            "type": "AllowedValues",
            "interval": [0, 255]
        }
    }
    ]
}
    
```

## New Model for Geospatial Coverages in JSON

```

    }, {
      "type": "QuantityType",
      "label": "blue",
      "definition": "ogcType:unsigned",
      "uom": {
        "type": "UnitReference",
        "code": "dn"
      },
      "constraint": {
        "type": "AllowedValues",
        "interval": [0, 255]
      }
    }
  ]
}

```

## Coverage Implementation Schema in JSON

We have seen several examples of coverages encoded in JSON using CIS. JSON was defined as an object based encoding, as opposed to defining classes. CIS provides a JSON Schema validation designed to ensure that. All examples shown here can be validated against the JSON Schema that can be found in <http://schemas.opengis.net/cis/1.1/json>.

To understand the basics of the presented JSON Schema, we include a UML diagram (Figure 18) not showing anything new but is summarizing the different arrangements of the CIS model.

The equivalent JSON Schema describes the main structure of the JSON document. One of the characteristics that we immediately see is the number of times we are using a “oneOf” property to show different possibilities that a coverage has. Actually, each generalization forces a “oneOf” in the JSON Schema. This way, we can see that a coverage can be described by domainSet, rangeSet and rangeType (in the CoverageByDomainAndRange and in DiscreteCoverage) or by a envelop, partitionSet and rangeType (in the CoverageByPartitioning). To be able to recycle the definitions of these elements, they are referenced to the definitions part of the Schema. Again, in the definition of a rangeSet we see another “oneOf” that present a choice between a dataBlock or a fileReference.

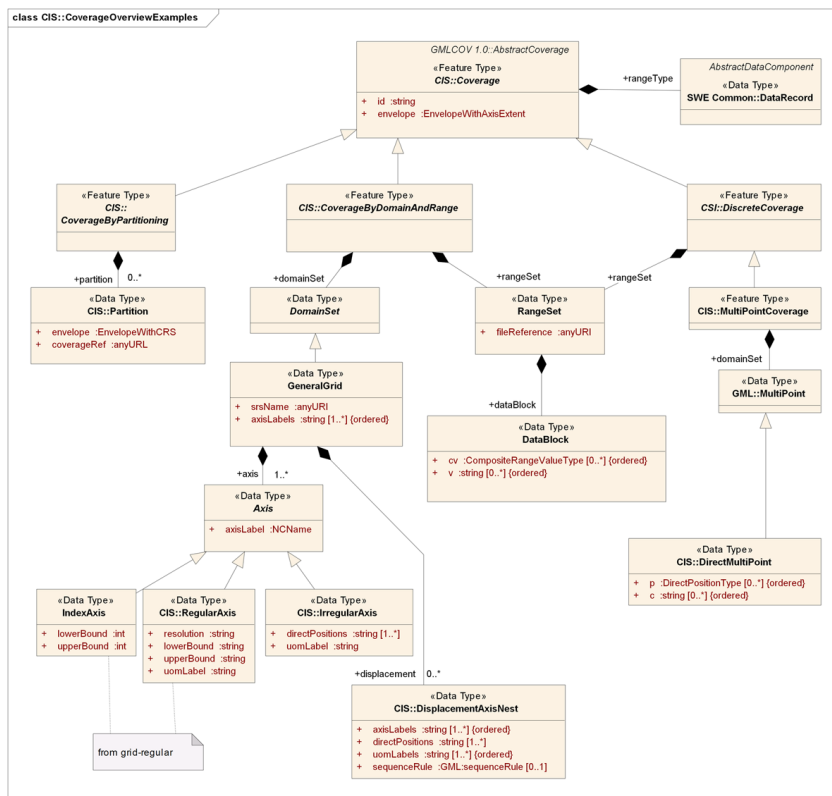
```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Coverage object",
  "description": "Component of OGC Coverage Implementation Schema 1.1.  
Last updated: 2016-may-18. Copyright (c) 2016 Open Geospatial Consortium, Inc.  
All Rights Reserved. To obtain additional rights of use, visit http://www.opengeospatial.org/legal/.",
  "type": "object",
  "oneOf": [
    {
      "required": [ "type", "domainSet", "rangeSet", "rangeType" ],

```

### New Model for Geospatial Coverages in JSON

Figure 18. Summary of all classes used in the previous examples.



```

    "properties": {
        "id": { "type": "string"},
        "type": { "enum": [ "CoverageByDomainAndRangeType",
            "MultiPointCoverageType" ] },
        "envelope": { "$ref": "#/definitions/envelope" },
        "domainSet": { "$ref": "#/definitions/domainSet" },
        "rangeSet": { "$ref": "#/definitions/rangeSet" },
        "rangeType": { "$ref": "#/definitions/rangeType" },
        "metadata": { "$ref": "#/definitions/metadata" }
    }
}, {
    "required": [ "type", "partitionSet", "rangeType"],
    "properties": {
        "id": { "type": "string"},
        "type": { "enum": [ "CoverageByPartitioningType" ] },
        "envelope": { "$ref": "#/definitions/envelope" },
        "partitionSet": { "$ref": "#/definitions/partition-
            Set" },
        "rangeType": { "$ref": "#/definitions/rangeType" },
    }
}

```



### ***New Model for Geospatial Coverages in JSON***

```
        "metadata": { "$ref": "#/definitions/metadata" }
    },
    "definitions": {
        "envelope": {
            ...
        },
        "domainSet": {
            "title": "domainSet",
            "description": "The domainSet describes the *direct
positions* of the coverage, i.e., the locations for which values are avail-
able.",
            "oneOf": [
                {
                    "description": "A general n-D grid is defined
through a sequence of axes, each of which can be of a particular axis type.
Use only if the parent object is type: 'CoverageByDomainAndRangeType'.",
                    ...
                }, {
                    "title": "domainSetMultiPoint",
                    "description": "The domainSet describes the
*direct positions* of the coverage, i.e., the locations for which values are
available. Use only if the parent object is type: 'MultiPointCoverageType'.",
                    "oneOf": [
                        {
                            "required": [ "type", "coordinates"
],
                            ...
                        }, {
                            "required": [ "type", "fileReference"],
                            ...
                        }
                    ]
                },
                "rangeSet": {
                    "title": "rangeSet",
                    "description": "The rangeSet lists a value ",
                    "type": "object",
                    "oneOf": [{
                        "required": [ "type", "dataBlock"],
                        ...
                    },
                    {
                        "required": [ "type", "fileReference"],
                        "properties": {
```

### **New Model for Geospatial Coverages in JSON**

```

        "type": { "enum": [ "RangeSetRe-
fType" ] },
        "fileReference": { "type": "string",
"format": "uri" }
    }
}
},
"partitionSet": {
    "title": "Partitioning Set",
    "type": "object",
    ...
},
"rangeType": {
    "title": "rangeType",
    "description": "The rangeType element describes the
structure and semantics of a coverage's range values, including (optionally)
restrictions on the interpolation allowed on such values.",
    "type": "object",
    "oneOf": [{
        "required": [ "type", "field" ],
        "properties": {
            "type": { "enum": [ "DataRecordType" ]
},
            ...
        }
    }
], {
    "required": [ "type", "fileReference" ],
    ...
}
}
},
"metadata": {
    "title": "Metadata",
    "type": "object"
}
}
}

```

In addition to the Schema, in the OGC Schemas repository we can also find JSON Context documents that are used in CIS instances to describe the JSON files in a way that a JSON-LD engine can automatically transform the JSON CIS documents to a RDF encoding.

## New Model for Geospatial Coverages in JSON

### CoverageJSON

CoverageJSON is an alternative initiative for expressing coverages in JSON developed in a joint W3C-OGC effort that takes the work within the MELODIES project as a starting point. The objective is to develop a well-structured, consistent and easy-to-use JSON format for coverages that is able to encode many different kinds of coverage, including multidimensional grids, time series, vertical profiles, polygon-based coverages, point clouds and more. (Blower, et al. 2016).

In CoverageJSON, a Coverage consists of the following objects:

- One *domain* which encodes the set of points in space and time for which we have data values (a grid or a discrete series of geometrical features); equivalent to the domainSet in CIS.
- An array of *ranges* (one per scalar quantity or field), holding an array of actual data values; one per grid cell or geometrical feature; equivalent to the rangeSet in CIS.
- An array of *parameters* objects (one per scalar quantity), describing the data value type; equivalent to the rangeType in CIS.
- An optional array of *parameterGroup* objects, which describe the semantic associations between *parameters*.

The following example encodes a 4 dimensional gridded coverage with two fields (Sea Ice concentration and Land Cover) To elaborate it we have merged of two examples in the CoverageJSON playground: <https://covjson.org/playground/>)

```
{
  "type": "Coverage",
  "domain": {
    "type": "Domain",
    "domainType": "Grid",
    "axes": {
      "x": { "values": [-10,-5,0] },
      "y": { "values": [40,50] },
      "z": { "values": [ 5] },
      "t": { "values": ["2010-01-01T00:12:20Z"] }
    },
  },
  "parameters": [{
    "ICEC": {
      "type": "Parameter",
      "description": {
        "en": "Sea Ice concentration (ice=1;no ice=0)"
      },
      "unit": {
        "label": {
          "en": "Ratio"
        },
      },
    },
  ]
}
```

### **New Model for Geospatial Coverages in JSON**

```
    "symbol": {
      "value": "1",
      "type": "http://www.opengis.net/def/uom/UCUM/"
    }
  },
  "observedProperty": {
    "id": "http://vocab.nerc.ac.uk/standard_name/sea_ice_area_fraction/",
    "label": {
      "en": "Sea Ice Concentration"
    }
  }
}, {
  "LC": {
    "type": "Parameter",
    "description": {
      "en": "Land Cover according to xyz classification"
    },
    "observedProperty": {
      "id": "http://example.com/landcover",
      "label": {
        "en": "XYZ Land Cover"
      }
    },
    "categories": [{
      "id": "http://example.com/landcover/categories/grass",
      "label": {
        "en": "Grass"
      },
      "description": {
        "en": "Very green grass."
      },
      "preferredColor": "#01A611"
    }, {
      "id": "http://example.com/landcover/categories/rocks",
      "label": {
        "en": "Rock"
      },
      "description": {
        "en": "Just rocks."
      },
      "preferredColor": "#A0A0A0"
    }
  ]
}, {
  "categoryEncoding": {
    "http://example.com/landcover/categories/grass": 1,
```

## New Model for Geospatial Coverages in JSON

```
    "http://example.com/landcover/categories/rocks": 2
  }
}
}],
"ranges": [{
  "ICEC": {
    "type": "NdArray",
    "dataType": "float",
    "axisNames": ["t", "z", "y", "x"],
    "shape": [1, 1, 2, 3],
    "values": [ 0.5, 0.6, 0.4, 0.6, 0.2, null ]
  }
}, {
  "LC": {
    "type": "NdArray",
    "dataType": "integer",
    "axisNames": ["t", "y", "x"],
    "shape": [1, 2, 3],
    "values": [ 1, 1, null, 2, 1, 2 ]
  }
}
}]
}
```

## Comparing CIS JSON With CoverageJSON

The W3C Spatial Data on the Web Working Group has established CoverageJSON is an alternative JSON encoding of a concept named coverages, although it is structurally incompatible with the one accepted by the OGC coverages group; both should not be confused. We briefly discuss CoverageJSON in the sequel.

Both encodings look similar in the concepts and in the structure of the root element. Many differences are only superficial consisting in a different naming and grouping of properties. Nevertheless there are some fundamental difference between CIS JSON and CoverageJSON that are worth mentioning.

- CIS model uses a different object for grid coverages and discrete coverages (non-gridded coverages), whereas CoverageJSON uses the same object with internal differences.
- CIS allows each coverage to have exactly one CRS, whereas CoverageJSON allows different CRSs to be applied to different sets of coordinates in the domain (e.g. one CRS for x and y, and another CRS for z). The CIS approach is to define a composite CRS which encompasses all axes in a uniform manner, both they space or time.
- Version 1.1 of CIS defines a JSON encoding that uses a near-literal translation of the UML model that has its origins in GML structures (that were originally inspired by ISO 19123). CoverageJSON does not use GML as starting point and does not define any conceptual model. Instead it takes ISO 19123 as starting point and directly suggests a JSON encoding.

## New Model for Geospatial Coverages in JSON

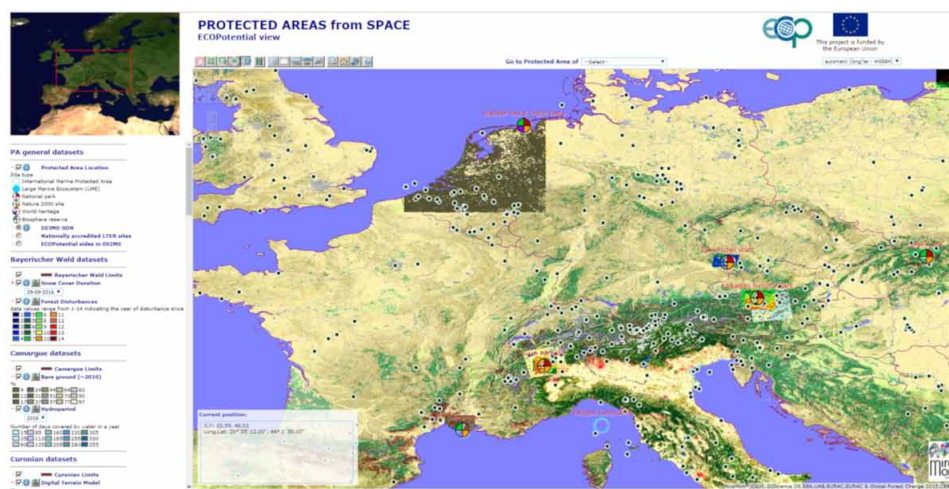
- CIS uses SWE Common to encode value types directly allowing for documenting data quality at the coverage field level and indirectly grid cell level quality. CoverageJSON focus on including grid cell statistical uncertainties.
- CoverageJSON includes default symbolization and category descriptions (for example for a land cover map classes) that are not available in the CIS encoding in JSON. CIS is engineered more specific and expects domain specific items to be added through extensions or application profiles (such as EO-WCS for satellite imagery and MetOcean-WCS for atmospheric data).

## VISUALIZING AND ANALYZING COVERAGES ON THE WEB

The CIS coverage encoding and coverage services can be used to build a smart geospatial visualization web client replacing the traditional way of using a WMS service to send pictorial representations of maps to the client. This innovative approach is used in the H2020 ECOPotential web map browser, Protected Areas from Space: <http://maps.ecopotential-project.eu/> (Figure 19) but it is also offered as a generic open source project in the GitHub (<https://github.com/joanma747/MiraMonMapBrowser>).

This web map client requests several coverage datasets from a server as arrays of gridded values for each coverage in the client. In the client side, the user can interact with a JavaScript based interface generating a on the fly visual representations based on CIS rangeSet data values. The advantage is that once the client has the CIS rangeSet data values it can do much more than simple data visualization, for example statistical calculations on what is in the screen or performing pixel based operations among coverages from different sources (Maso et al 2018). The generic web map browser software can be adapted to a particular application (e.g. ECOPotential) by encoding a configuration file config.json. The config specifies generic properties of the map browser as well as the list of geospatial layers to be seen, the server to retrieve the data and the symbolization information needed to represent each layer on the screen.

Figure 19. Protected Areas from Space Map Browser, general overview.



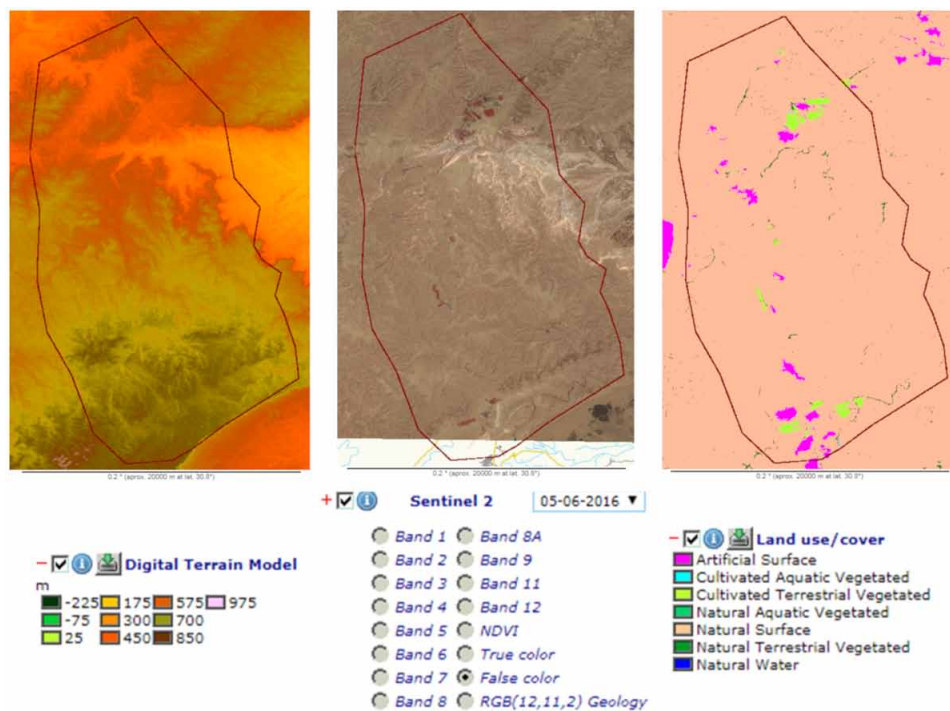
### New Model for Geospatial Coverages in JSON

The web map client deals with several coverage types, such as quantitative, multiband and categorical coverages. In the area of Har Ha Negev Protected Area, an example of quantitative coverage is the Digital Terrain Model (DTM): a dataset that has values recording the elevation of each pixel. On the other hand, an example of multiband (as well as multitemporal) coverage is an image obtained by the Sentinel 2 satellite (from the ESA Copernicus program) which shows values for several available bands, covering different regions of the electromagnetic spectrum (13 bands for OLI sensor). An example of categorical coverage is the Land Use where each integer numerical value corresponds to a land use class (e.g. Natural Surface) (Figure 20).

### Coverage Request and Storage

A map in the screen area has a stack of several overlaying transparent canvases of the exact same size, each one representing data coming from a coverage from a server. The JavaScript code uses the domain-Set information (mainly the bounding box coordinates and the fixed number of columns and rows of the canvas) to elaborate a request to a service that responses with a rangeSet with values corresponding one-to-one to the pixels of a map area in the screen. Web service response is sent in an asynchronous mode using the XMLHttpRequest() function configured for binary array responses:

Figure 20. Several coverage types and their legends for the Har Ha Negev Protected Area. Left: DTM (quantitative coverage); Center: Sentinel 2 image (multiband - coverage); Right: Land use (categorical coverage).



## **New Model for Geospatial Coverages in JSON**

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (xhr.readyState ===
      XMLHttpRequest.DONE &&
      xhr.status === 200)
  {
    // xhr.response contains the
    // binary array
  }
}
xhr.open("GET", path, true);
xhr.responseType = "arraybuffer";
xhr.send();
```

The web server generates a raw format with no headers but just values as a little-endian binary sequence. We avoid using a GeoTIFF file or similar because it will require considerably more work in the client side. Using the new JavaScript array buffer object, the data can be stored in a variable and accessed by creating a new `DataView()` object. Then, depending on the data type (contained in the `CIS rangeType`) of the binary values, values are extracted:

```
var dv=DataView(arrayBuffer);
var i_byte=(ncol*ifil+icol)
if (datatype=="uint16")
    getUint16(i_byte*2, true);
else if (datatype=="float32")
    getFloat32(i_byte*4, true);
else if ...
```

Next step is to represent the data in the map area of the screen. The binary arrays returned cannot be send directly to the canvas and are adapted to the appropriate format required by `createImageData()` function.

## **Coverage Encoding in the Client**

### **Coverage Values Description**

Each coverage in the client is described as an entry in the `config.json` file. For the sake of brevity we limit its description to the parameters needed to understand how to convert the `rangeSet` into RGB colors for the canvas. The information on the full composition of this file can be found in the `config-schema.json` document that is used not only for validation purposes but also as documentation.

When the image format is set to *application/x-img*, the client is aware that a coverage will be used and that certain other information (such as color palettes) will be described in the client. A *values* array contains a simplification of the information extracted from the `rangeType` information, consisting on the



### New Model for Geospatial Coverages in JSON

*data type*, *nodata* values and the compression format (currently limited to *none* or simple Run-Length-Encoding; RLE). See a DTM example:

```
"ImageFormat": "application/x-img",
"values": [{
  "compression": "RLE",
  "datatype": "int16",
  "nodata": [-9999]
}]
```

For multiband coverages (Sentinel 2 example) the *values* arrays has some extra parameters that describe the dimensions of the domainSet (in this example only one extra dimension with the bands of the coverage as an index axis).

```
"ImageFormat": "application/x-img",
"values": [{
  "param": [{"key": {"name": "DIM_BAND", "descr": "Band"}, "value": {"name": "B01", "descr": "1 Coastal aerosol (0.443µm)"}}],
  "compression": "RLE",
  "datatype": "uint16",
  "nodata": [65535, 0]
},
{
  "param": [{"key": {"name": "DIM_BAND", "descr": "Band"}, "value": {"name": "B02", "descr": "2 Blue (0.490µm)"}}],
  "compression": "RLE",
  "datatype": "uint16",
  "nodata": [65535, 0]
},
{
  "param": [{"key": {"name": "DIM_BAND", "descr": "Band"}, "value": {"name": "B03", "descr": "3 Green (0.560µm)"}}],
  "compression": "RLE",
  "datatype": "uint16",
  "nodata": [65535, 0]
}]
```

Each index axis is described in a key (the axis name) and value (the index) structure. This short example only describes the first three bands of the OLI sensor in the Sentinel 2 satellite, i.e. "1 Coastal aerosol (0.443µm)", "2 Blue (0.490µm)" and "3 Green (0.560µm)".

## New Model for Geospatial Coverages in JSON

### Styles Description

Styles are a set of rules that should be implemented in the JavaScript client to transform the rangeSet into the proper RGBA values compatible with the canvas. There are two main ways of doing that: using color palettes or creating RGB compositions.

In the first mode a single rangeSet of values is used to generate the visualization by mapping the values to a color map consisting on a predefined list of RGB values. If the value is described by a category text, the value of the cell is interpreted as a numerical index into the list of RGB values. In case that the value represents a continuous value, a linear transformation is applied to map the minimum possible value in the rangeSet to the first color of the color map list and the maximum possible value in the rangeSet to the last color of the color map list.

The second mode requires that 3 bands are requested to the server (The client does that in three separated asynchronous request). The first returned rangeSet is rescaled to represent the R color channel intensity between 0 and 255; the second will be rescaled to represent the G color channel and the third to represent the B color channel. This mode is useful to represent band combinations of satellite imagery (e.g. obtaining a natural color composition from a Sentinel 2 image by requesting the bands B04, B03 and B02).

For the first mode a single component is described and the minimum and maximum values to rescale the rangeSet to a color palette indices, as well as the units for the value are given.

```
"style": [{
  "descr":          "DTM",
  "ItemsDescr":    "m",
  "component":     [{
    "i_value":      0,
    "PaletteRescale": {
      "maxValue":   1050,
      "minValue":   -300
    }
  ]
}],
```

In the second mode, for an RGB composition, an array with three elements is used in the *component* property of the style to describe each of the three bands used for the RGB composition.

```
"style": [{
  "descr":          "False color",
  "component":     [{
    "i_value":      7,
    "PaletteRescale": {
      "maxValue":   8191,
      "minValue":   0
    }
  }, {
    "i_value":      3,
```

## New Model for Geospatial Coverages in JSON

```

        "PaletteRescale": {
            "maxValue": 8191,
            "minValue": 0
        }
    }, {
        "i_value": 2,
        "PaletteRescale": {
            "maxValue": 8191,
            "minValue": 0
        }
    }
}],
    
```

### Palette Encoding and Usage

When a color palette is needed, an extra property *palette* is included in the symbolization description. It contains a single property called *colors* which is an array containing a list of colors. The colors are encoded using hexadecimal notation. For example, for the Land Use coverage, the following JSON fragment is used:

```

"style": [{
    "descr": "LULC",
    (...)
    "palette": {
        "colors": [null, "#FF00FF", "#00FFFF", "#C0FF00",
"#00C07A", "#FFC0A0", "#008000", "#0000FF"]
    }
}]
    
```

This simple example for land use a few colors are used, as shown in the encoding above and in Table 1.

The color palette is created in a way that the number of colors matches the rank of the rangeSet (starting from the lowest value). In this example a cell in the rangeSet with a value “2” will be colored with Cyan.

Usually quantitative coverages (such as DTM) are equipped with a color palette that defines a higher number of colors to give an impression of some continuous gradient. A common approach is to use a 256 color list to portray the coverage (even if in the client legend only a selection of them is shown for simplicity). As explained before, in the “component” property of the style description, the minimum and maximum values to rescale the color palette are included.








The encoding of the component and the color palette for the DTM example is (some encoded properties and some values in the colors’ palette array are omitted and substituted by “(...)”):

```

"style": [{
    "descr": "DTM",
    "ItemsDescr": "m",
    "component": [{
    
```

### New Model for Geospatial Coverages in JSON

Table 1. Land Use colors used adapt the data to the requirements of the canvas

HEX color	RGB color	Value	Textual color
null	null	0	<i>Not in the legend</i>
#FF00FF	rgb(255, 0, 255)	1	Pink 
#00FFFF	rgb(0, 255, 255)	2	Cyan 
#C0FF00	rgb(192, 255, 0)	3	Light green 
#00C07A	rgb(0, 192, 122)	4	Medium green 
#FFC0A0	rgb(255, 192, 160)	5	Salmon 
#008000	rgb(0, 128, 0)	6	Dark green 
#0000FF	rgb(0, 0, 255)	7	Blue 

```

        "i_value": 0,
        "PaletteRescale": {
            "maxValue": 1050,
            "minValue": -300
        }
    }],
    (...),
    "palette": {
        "colors": ["#000000", (...), "#003300", (...),
"#00CC00", (...), "#CCFF00", (...), "#FFCC00", (...), "#FF9900", (...),
"#FF6600", (...), "#CC6600", (...), "#999900", (...), "#663300", (...), "#FFC-
CFF", (...), "#FFB6FF"]
    }
}]]

```

In this example, 256 colors are described but only a few are selected and shown in the legend as represented in Table 2.

### Categories Description Encoding

The rangeTypes of coverages should be numerical even if for coverage representing textual categories. This is encoded in the style description by adding an *attributes* and a *categories* properties:











```

"style": [{
    "descr": "LULC",
    "categories": [
        null,
        {"COVER": "Artificial Surface"},
        {"COVER": "Cultivated Aquatic Vegetated"},
    ]
}]

```

**New Model for Geospatial Coverages in JSON**

Table 2. Some representative colors used in the Digital Terrain Model color map

HEX color	RGB color	Values rank	Textual color
#000000	rgb(0, 0, 0)	[-300,-295]	Black <i>(not in the legend)</i>
#003300	rgb(0, 51, 0)	[-225,-221]	Dark green 
#00CC00	rgb(0, 204, 0)	[-77,-73]	Medium green 
#CCFF00	rgb(204, 255, 0)	[23,28]	Light green 
#FFCC00	rgb(255, 204, 0)	[172,176]	Light orange 
#FF9900	rgb(255, 153, 0)	[299,303]	Medium orange 
#FF6600	rgb(255, 102, 0)	[447,451]	Dark orange 
#CC6600	rgb(204, 102, 0)	[574,578]	Light brown 
#999900	rgb(153, 153, 0)	[696,700]	Olive 
#663300	rgb(102, 51, 0)	[849,854]	Dark brown 
#FFCCFF	rgb(255, 204, 255)	[971,975]	Pink 
#FFB6FF	rgb(255, 182, 255)	[1045,1050]	Dark pink <i>(not in the legend)</i>

```

        {"COVER": "Cultivated Terrestrial Vegetated"},
        {"COVER": "Natural Aquatic Vegetated"},
        {"COVER": "Natural Surface"},
        {"COVER": "Natural Terrestrial Vegetated"},
        {"COVER": "Natural Water"}],
    "attributes": [{
        "name": "COVER",
        "descr": "Land use/cover",
        "visible": true
    }],
    },

```

In the *attributes* property, several rangeType fields can be defined for each coverage. In this case, only one attribute is described, as only one numerical field is available on this coverage. The *categories* array is linked to the range in the rangeSet; thus, here value 0 is linked to a null description (no pixel is having this value), value 1 is linked to “Artificial Surface”, and so on.

Taking this into consideration, the table summarizing Land Use rangeSet values in the previous section can be extended with a new column including the category description as shown in Table 3.

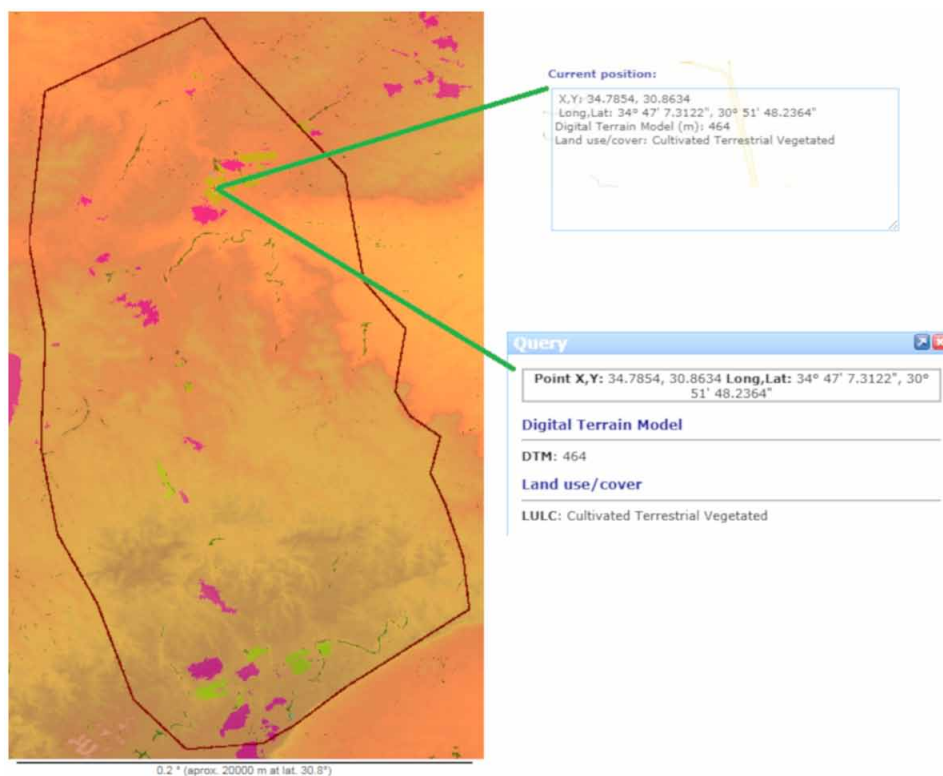
The use of those textual descriptors instead of numerical values is extended not only in the client legend but in query results and statistical pie charts that show textual descriptors of coverages. As shown in Figure 21 and 22 below, for a quantitative coverage such as the DTM, the numeric values of the coverage (and its units of measure, if available) are shown and for a categorical coverage, the numerical value of the coverage is hidden and category descriptions are shown instead.

### New Model for Geospatial Coverages in JSON

Table 3. Land Use color meaning used in the legend, data queries and statistical summaries

HEX color	RGB color	Value	Textual color	Category description
null	null	0	Black <i>(not in the legend)</i>	<i>null (no pixel has this value)</i>
#FF00FF	rgb(255, 0, 255)	1	Pink 	Artificial Surface
#00FFFF	rgb(0, 255, 255)	2	Cyan 	Cultivated Aquatic Vegetated
#C0FF00	rgb(192, 255, 0)	3	Light green 	Cultivated Terrestrial Vegetated
#00C07A	rgb(0, 192, 122)	4	Medium green 	Natural Aquatic Vegetated
#FFC0A0	rgb(255, 192, 160)	5	Salmon 	Natural Surface
#008000	rgb(0, 128, 0)	6	Dark green 	Natural Terrestrial Vegetated
#0000FF	rgb(0, 0, 255)	7	Blue 	Natural Water

Figure 21. Quantitative and categorical coverages in queries for the Har Ha Negev Protected Area.

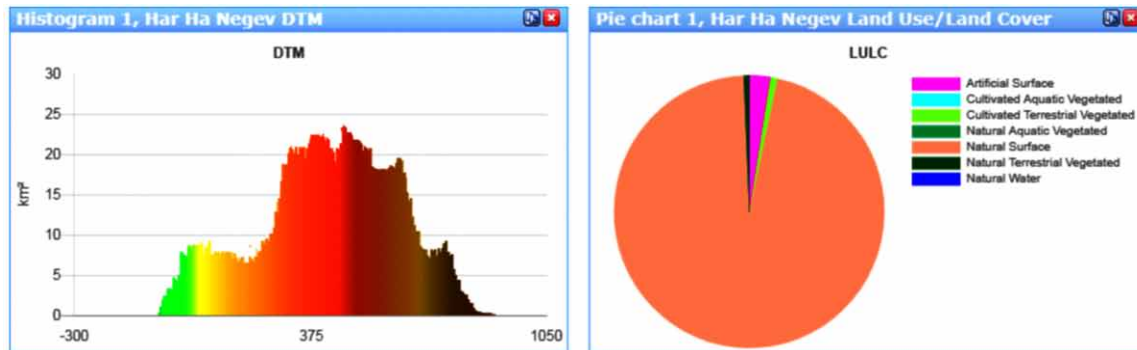


### FUTURE RESEARCH DIRECTIONS

We have shown the similarities and the differences of the CIS encoding in JSON and the CoverageJSON. CoverageJSON proponents claim that some CIS structures are too complex and that CoverageJSON provides some extra functionalities missing in CIS. However, there are two points to be made from

**New Model for Geospatial Coverages in JSON**

Figure 22. Quantitative and categorical coverages in histograms and pie charts for the Har Ha Negev Protected Area.



the OGC coverage standards perspective, on technical and interoperability aspects. Technically, OGC coverages form a piece of a complete ecosystem of data and service models, with a rich landscape of encoding formats, simple access formats, modular service model components, up to declarative datacube analytics; therefore facets appear in the coverage definition which CoverageJSON, due to its narrower focus, does not need to consider. From an interoperability perspective, it is disadvantageous that the W3C model has chosen to piggyback on the name “coverages”, although incompatible with the OGC coverage standards established since many years. A different naming would allow proper distinction and, thereby, reduce confusion in the communities.

That said, it is desirable to achieve a harmonization. Similarities need to be exploited and lessons need to be learned from both models (in particular, the Petascale experience gained with OGC coverages and WCS) and a single harmonized model should emerge. One option under discussion is to establish CoverageJSON as an additional JSON encoding of OGC coverages, following the CoverageJSON adjustments required for this. Another option could be to define a simple profile for CIS that satisfies the needs of CoverageJSON community. OGC should take the lead in this process to avoid duplication of efforts and confusing implementers.

We have illustrated that, with a set of rules, the CIS UML class model can be converted in conformant JSON instances and JSON Schemas. It seems possible to generalize these practices to many other UML class model in the geospatial domain developed by OGC and ISO. In this respect, the Sensor Web Enablement family of standards should advance in the definition of JSON encodings using this approach. The JSON encoding might not result in the most compact possible but the advantage of having several encodings derived from the common UML model needs to be emphasized as a better good.

The encoding of coverages in JSON lowers the bar for implementing web map browsers and, at the same time, opens interactive possibilities for the user interfaces where users are able to play with the data. The possibilities and limitations opened by the presented implementation need to be exploited further. The application of canvas has demonstrated fast and easy to use but it is limited to 2D visualization. Presenting multidimensional information in a 2D computer screen is still a challenge that needs to be overcome.

## ***New Model for Geospatial Coverages in JSON***

The capability to have binary arrays at screen resolution has proved effective to do analytical calculations among datasets and present virtually computed layers that can be browsed because they are calculated on-the-fly. Statistical summaries of these visualizations can be calculated and shown but users have to take into account the implications of doing statistical assessments at screen resolution. More work needs to be done to assess the uncertainties introduced. Real time visual inspection needs to be complemented with slower but precise calculations done in the server side using for example the WCPS standard.

The client presented has revealed the feasibility of the approach. The initial set of tools (e.g. layer calculator, histogram, animation of time series, filter capabilities) has to be extended with a complete set of analytical capabilities.

## **CONCLUSION**

The fundamentals of what a coverage is and the data model behind it were defined many years ago in ISO19123. One of the first implementations was done in GML but it required an GML Application Schema to be created for each type of coverage. This level of abstraction is difficult to master what restricted the usability of the approach. The Coverage Implementation Schema elaborated by the OGC coverages group provides a generic coverage model in UML and a fixed schema that can be reused in all coverages and across all encodings. In parallel, in an OGC testbed generic rules to transform UML models into JSON instances and JSON schemas.

The work presented in this Chapter demonstrates how JSON can be used to represent and transmit coverage information in standard way. The Chapter presents and extends the work done in the elaboration of the version 1.1 of the CIS that was released as a conceptual model and three encodings, one of them in JSON. For the first time a JSON encoding was standardized by the OGC for describing coverages. The authors of this Chapter presents a successful stress test that demonstrates the applicability of the generic rules for directly derive JSON from UML class diagrams proposed in the Testbed. The JSON encoding for CIS is able to represent gridded and discrete as well as continuous coverages as defined by CIS. The data model covers the definition of the array of values (rangeSet) together with the description of the rules to generate the domainSet from axes or directly to individual geometries, and the semantics of the values (rangeType), plus any additional metadata. The JSON objects are instances of the class model.

Due to the flexibility of JSON, other alternative encodings for coverages can emerge that are perfectly valid from the JSON structure and validation point view such as the CoverageJSON done by a group in W3C, They produce different alternatives that are based on similar conceptual schemas that generate other JSON objects that are incompatible with the one presented in this chapter and represent a risk for interoperability, forcing implementations to include extra parser efforts to, in the end, achieve very similar functionalities. Even if they are correct JSON encodings, they are not embedded in the OGC and ISO standards ecosystem; and may be incomplete. For example, in the case of CoverageJSON they lack a service model such as OGC WCS.

The incorporation of JSON will ease the implementation of OGC standards in map browsers. The current web browsers that implement the new HTML5 and the enhanced JavaScript engines make possible to use coverages directly in the web browser. In HTML5 it is possible to create rich map representations that go beyond the mass marked approach of having only imagery and road maps. In the past, map browsers were only able to present JPEG images at screen resolution needing constant web service com-



## ***New Model for Geospatial Coverages in JSON***

munication that results in delays and limited user experience. The client presented in this Chapter uses internal binary arrays embedded in JSON encoded coverage descriptions. For example and in addition to personalized visualization as interactive maps, the information can be presented to the user with statistics, graphics, dynamic time series, and all this increasing user experience. Expert users can combine OGC coverage data to do geospatial analysis directly in the web browser that will present virtual layer results. In the end, the server-side fusion capabilities of the OGC WCPS language will provide full access to the analytical results. This opens up additional degrees of freedom in system design.

## **ACKNOWLEDGMENT**

This research was supported by the European Union's Horizon 2020 Programme [ECOPotential (641762-2)]; This work has been partially funded by the Spanish MCIU Ministry through the NEWFORESTS research project (RTI2018-099397-B-C21/C22 MCIU/AEI/FEDER, UE).

## **REFERENCES**

- Baumann, P. (2017). *OGC Web Coverage Service (WCS) 2.1 Interface Standard – Core v.2.1*. Open Geospatial Consortium (OGC 17-089r1). Retrieved from: <http://docs.opengeospatial.org/is/17-089r1/17-089r1.html>
- Baumann, P., Hirschorn, E., & Maso, J. (2017). *OGC Coverage Implementation Schema v.1.1*. Open Geospatial Consortium (OGC 09-146r6). Retrieved from: <http://docs.opengeospatial.org/is/09-146r6/09-146r6.html>
- Baumann, P., Hirschorn, E., Maso, J., Merticariu, V., & Misev, D. (2017). All in One: Encoding Spatio-Temporal Big Data in XML, JSON, and RDF without Information Loss. *Proceedings: IEEE International Conference on Big Data*, 3406-3415.
- Benson, T., & Grieve, G. (2016). UML, BPMN, XML and JSON. In *Principles of Health Interoperability* (pp. 55–81). Cham: Springer. doi:10.1007/978-3-319-30370-3\_4
- Blower, J., & Riechert, M. (2016). *Coverages, JSONLD and RDF data cubes*. In Workshop on Spatial Data on the Web (SDW 2016), Montreal, Canada. Retrieved from: <http://centaur.reading.ac.uk/74395/1/paper2.pdf>
- Bray, T. (2014). *IETF RFC7159, The JavaScript Object Notation (JSON) Data Interchange Format*. Retrieved from: <https://www.ietf.org/rfc/rfc7159.txt>
- Izquierdo, J. L. C., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. *Knowledge-Based Systems*, 103, 52–55. doi:10.1016/j.knosys.2016.03.020
- Jetlund, K. (2015). *Best practices for diagram design*. Retrieved from: <https://github.com/ISO-TC211/UML-Best-Practices/wiki/Best-practices-for-diagram-design>

## **New Model for Geospatial Coverages in JSON**

Maso, J. (2017). *Testbed-12 JavaScript-JSON-JSON-LD Engineering Report (OGC 16-051)*. Retrieved from: <http://docs.opengeospatial.org/per/16-051.html>

Maso, J., Zabala, A., Serral, I., & Pons, X. (2018). Remote Sensing Analytical Geospatial Operations Directly in the Web Browser. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42(4), 403–410. doi:10.5194/isprs-archives-XLII-4-403-2018

OGC 07-011. (2015). *Abstract Specification Topic 6: Schema for coverage geometry and functions, version 7.0*. Retrieved from: [http://portal.opengeospatial.org/files/?artifact\\_id=19820](http://portal.opengeospatial.org/files/?artifact_id=19820)

Portele, C. (2007). *OpenGIS Geography Markup Language (GML) Encoding Standard. Ver 3.2.1, (OGC 07-036)*. Retrieved from: [http://portal.opengeospatial.org/files/?artifact\\_id=20509](http://portal.opengeospatial.org/files/?artifact_id=20509)

Robin, A. (2011). *OGC 08-094, OGC® SWE Common Data Model Encoding Standard, version 2.0*. Retrieved from: [http://portal.opengeospatial.org/files/?artifact\\_id=41157](http://portal.opengeospatial.org/files/?artifact_id=41157)

Sporny, M., Kellogg, G., & Lanthaler, M. (2014). *W3C JSON-LD 1.0, A JSON-based Serialization for Linked Data*. Retrieved from: <http://www.w3.org/TR/json-ld/>

Witayangkurn, A., Horanont, T., & Shibasaki, R. (2012). Performance comparisons of spatial data processing techniques for a large scale mobile phone dataset. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications* (p. 25). ACM. 10.1145/2345316.2345346

## **KEY TERMS AND DEFINITIONS**

**Client:** Software component that can invoke an operation from a server.

**Coordinate Reference System:** Coordinate system that is related to the real world by a datum.

**Coordinate System:** Set of mathematical rules for specifying how coordinates are to be assigned to points.

**Coverage:** A feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain.

**Displaced Grid:** A grid whose direct positions are topologically aligned to a grid, but whose geometric positions can vary arbitrarily.

**Geographic Information:** Information concerning phenomena implicitly or explicitly associated with a location relative to the Earth.

**Irregular Grid:** A grid whose grid lines have individual distances along each grid axis.

**Mesh:** A coverage consisting of a collection of curves, surfaces, or solids, respectively.

**Partition (of a Coverage):** A separately stored coverage acting, by being referenced in the coverage on hand, as one of its components.

**Regular Grid:** A grid whose grid lines have a constant distance along each grid axis.

**Service:** Distinct part of the functionality that is provided by an entity through interfaces.