Testbed-12 Javascript-JSON-JSON-LD Engineering Report

Table of Contents

Publication Date: 2017-05-12

Approval Date: 2016-12-07

Posted Date: 2016-11-15

Reference number of this document: OGC 16-051

Reference URL for this document: http://www.opengis.net/doc/PER/t12-A005-2

Category: Public Engineering Report

Editor: Joan Masó

Title: Testbed-12 Javascript-JSON-JSON-LD Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

This document is an OGC Public Engineering Report created as a deliverable of an initiative from the OGC Innovation Program (formerly OGC Interoperability Program). It is not an OGC standard and not an official position of the OGC membership.It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

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.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Abstract

The Testbed-11 deliverable OGC 15-053 *Implementing JSON/GeoJSON in an OGC Standard ER* enumerated strategies for implementing JSON in OGC services and OGC encodings. Previously, a mechanism to migrate XML into JSON was proposed by Pedro Gonçalves in 14-009r1 *OGC Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context*. In contrast, this engineering report (ER) proposes a mechanism to derive JSON and JSON-LD encodings from UML modeling without using XML as an intermediate step. The rules provided can be divided into rules for JSON instances and rules for JSON schemas.

These rules have been applied to the UML diagrams in OWS common 2.0 to derive JSON encodings for them. In practice this ER evaluates how to provide service metadata in the derived JSON. JSON schemas and @context documents for independent validation of the four main sections of the ServiceMetadata are provided. This activity is done in connection with the OGC 16-052 OWS Context / Capabilities ER. The rules are applied to WMS to provide a full JSON encoding for the WMS 1.4 standard candidate.

Finally, this ER discusses the applicability to data geospatial formats, both for coverage formats (compared to the CIS standard) and feature formats (compared to GeoJSON).

Readers unfamiliar with JSON, JSON-LD and JSON Schema should first read OGC 16-122 (Geo)JSON User Guide. OGC 16-122 includes guidelines and recommendations for the use of JSON and JSON-LD in OGC data encodings and services.

Business Value

This ER provides a way to transform UML class diagrams into JSON and JSON-LD. Since many OGC standards contain UML models, application of the rules described in this document should readily result in a JSON and JSON-LD encoding for these standards. Having a JSON encoding for OGC data formats and OWS Services standards lowers the barrier of dealing with them in web browsers, where a JSON is easier to implement than existing equivalent XML encodings.

In addition, use of JSON-LD will enable connection of OGC standards encodings and service descriptions to RDF databases.

By providing generic rules, OGC standards willing to adopt JSON encodings will

be able to do so in a consistent manner, improving interoperability between them.

What does this ER mean for the Working Group and OGC in general

This ER is relevant to the Architecture DWG because it proposes a way forward to adopt JSON as an alternative to the current XML, improving the applicability of the standards. JSON, complemented with REST, facilitates a lighter architecture that can be enable more rapid web application development.

How does this ER relate to the work of the Working Group

This ER relates to the Architecture DWG and the OGC in general because it provides generic rules that can form a base for a document produced by the Architecture DWG that will become guidance for the OGC standards willing to adopt JSON encodings in a consistent manner.

Keywords

ogcdocs, testbed-12, JSON, JSON-LD

Proposed OGC Working Group for Review and Approval

Architecture DWG

Chapter 1. Introduction

1.1. Scope

This OGC® ER provides a set of rules on how to transform a UML model describing the encoding of a format or service description into JSON and JSON-LD encodings and their corresponding JSON schemas. The rules have been applied and validated in a service metadata document based on OWS Common.

This ER is applicable to OGC data and metadata encodings and OGC services willing to provide a JSON encoding for rapid adoption in web browsers.

This ER is complemented by OGC 16-122 (Geo)JSON User Guide, which includes guidelines and recommendations for the use of JSON and JSON-LD in OGC data encodings and services.

1.2. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Table 1. Contacts

Name	Organization
Joan Masó	UAB-CREAF

1.3. Future Work

The proposed JSON encoding for OWS Common should result in a Change Request (CR) that eventually could lead to an extension of OWS Common for JSON. The proposed encoding for WMS 1.4 should results in a CR to WMS SWG that can be included in its work to define the new version of the WMS standard.

The rules for transforming UML into JSON could result in a best practice document delivered by the Architecture DWG. In addition, Testbed 13 could include implementing these rules in an application such as Shapechange.

A pending work item would be to precisely define the JSON encoding for all geometric properties that can appear in a UML model. This will require more discussion in the OGC community. In particular, a future initiative should consider experimenting with the "double JSON serialization" idea to overcome the difficulties with arrays of coordinates in JSON-LD.

1.4. Foreword

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.

Chapter 2. References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- ISO 19118:2010 Geographic information — Encoding.
- IETF RFC7159, The JavaScript Object Notation (JSON) Data Interchange Format.
- IETF RFC4627, The application/json Media Type for JavaScript Object Notation (JSON).
- W3C JSON-LD 1.0, A JSON-based Serialization for Linked Data.
- IETF draft draft-zyp-json-schema-03, A JSON Media Type for Describing the Structure and Meaning of JSON.
    - This ER demonstrates the usefulness of JSON Schema. Unfortunately, there has been no progress in the draft schema towards standardizing it since 2013.
- OGC 06-103r4 OpenGIS implementation Standard for Geographic Information — Simple feature access — Part 1: Common Architecture.
- OGC 06-121r9, OGC® Web Services Common Standard
    - This OWS Common Standard contains a list of normative references that are also applicable to this document.

Chapter 3. Terms and definitions

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

3.1. class

description of a set of objects that share the same attributes, operations, methods, relationships, and semantics [ISO/TS 19103, 4.7]

3.2. coverage

feature that acts as a function to return values from its range for any direct position within its spatial, temporal or spatiotemporal domain [ISO/TS 19123:2003, 4.1.7]

3.3. encoding

conversion of data into a series of codes [ISO 19118:2010, 4.13]

3.4. encoding rule

identifiable collection of conversion rules that define the encoding for a particular data structure [ISO 19118:2010, 4.14]

3.5. encoding service

NOTE: An encoding rule specifies the types of data to be converted as well as the syntax, structure and codes used in the resulting data structure.

software component that has an encoding rule implemented [ISO 19118:2010, 4.15]

3.6. feature

abstraction of real world phenomena [ISO 19101, 4.39]

3.7. identifier

linguistically independent sequence of characters capable of uniquely and permanently identifying that with which it is associated [ISO 19135:2005, 4.1.5]

3.8. instance

object that realizes a class [ISO 19107:2005, 4.53]

3.9. schema

formal description of a model [ISO 19118:2010, 4.28]

Chapter 4. Conventions

4.1. Abbreviated terms

- CIS: Coverage Implementation Schema
- CR: Change Request
- DWG: Domain Working Group
- ER: Engineering Report
- GFM: General Feature Model
- JSON: JavaScript Object Notation
- JSON-LD: JSON Linked Data
- OWS: OGC Web Services
- RDF: Resource Description Framework
- SWG: Standards Working Group
- UML: Unified Modeling Language
- WKT: Well-Known Text
- WMS: Web Map Service
- XMI: XML Metadata Interchange
- XML: eXtensible Markup Language

4.2. UML notation

Most diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r9].

Chapter 5. Overview

5.1. Rules for converting UML models into JSON instances, schema and JSON-LD @context

This document provides a set of rules for converting a UML model into a JSON schema file for JSON files validation and a set of @context fragments that guide the JSON conversion into RDF triples.

The *encoding rule* concept is described by the ISO 19118:2010, as follows: "An encoding rule allows geographic information defined by application schemas and standardized schemas to be coded into a system-independent data structure suitable for transport and storage. The encoding rule specifies the types of data to be coded and the syntax, structure and coding schemes used in the resulting data structure. The resulting data structure may be stored on digital media or transferred using transfer protocols. It is intended to be read and interpreted by computers, but may be in a form that is human readable."

Applying chapter 9 of ISO 19118 to the current case, the schema describing the input data structure is the *UML class diagram* and the input data structures are data instances consistent with the schema in UML class diagrams. UML class diagrams are formal enough to be expressed in a text encoding in an Enterprise Architect Project format or exported into XMI. In our case, the output data structure (also called *exchange format*) consists of JSON-LD instances. By selecting JSON-LD, we are also creating a JSON compatible encoding. The output schema describing the JSON is a combination of JSON Schema and/or @context JSON-LD documents.

ISO 19118:2010 explains in section 9.5 that two sets of *conversion rules* may exist:

- The first one is the set of schema conversion rules, which define a mapping from the UML class diagram to the schema of the output data structure. In our case, this set will be composed by rules for creating JSON schemas and @context documents. This set of rules will be formulated in a way that an automatic conversion tool can be created.
- The second is the set of instance conversion rules, which defines a mapping from instances in the instance model to instances in the resulting data model. These are rules that apply directly to the creation of JSON-LD instances.

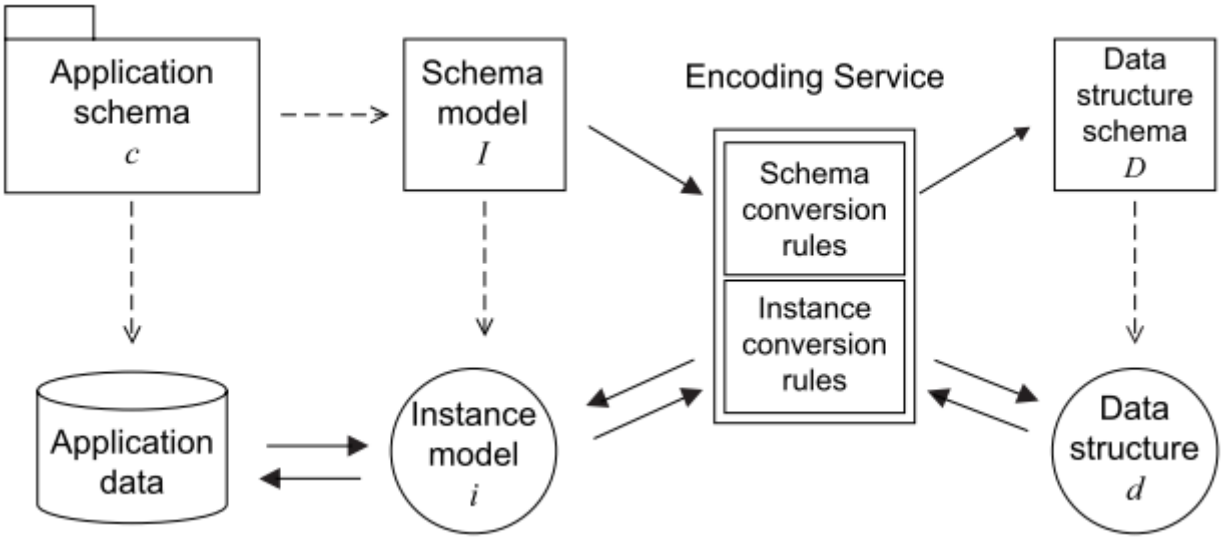

Figure 1. Schema model and Instance mode conversion rules (from ISO 19118)

The first set of *conversion rules* also imposes some rules on the instance creation (e.g. the conformity to a JSON Schema, the so called JSON validation). The two sets of rules must be consistent with each other.

The rules provided in this document have been elaborated with the intention of being self-sufficient. They can be translated into a code that can implement an *encoding service* that takes a collection of UML class diagrams and produces JSON schemas and JSON-LD @context automatically.

The rules focus on UML class diagrams showing classes that have a name (complex data type) with some attributes, associations and compositions. We apply three main assumptions:

- There is a need to group objects into classes that share the same data structure (the same complex data types), and the classes need to be transported to JSON.
- It is important that all objects have an identifier to be able to relate and refer to them from other objects in the model or from outside.
- A link from an object to another object will be done using a property in the first object where the property name expresses the semantics of the link (the reason to set the link) and the the value is an identifier of a target object.

In addition, we assume there is a need to be able to validate and document the meaning of the classes and attributes used in a JSON file with a combination of JSON and JSON-LD.

These three assumptions are well covered by JSON-LD. This means that they allow us to connect the object representation to a triple (RDF) representation as a bonus. The rules to convert JSON-LD into RDF are defined in the JSON-LD standards and do not need to be defined here.

Chapter 6. Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams

The rules provided in this clause and in the following one Rules for encoding JSON-LD instances conformant to the UML model have been created in a very formal way using normative language (i.e. using SHALL as a indication of obligation). Even if this style could be considered not appropriate for an ER, it has been done with the objective of facilitating the migration of the rules into normative document or a standard in the near future. In this sense, they are numbered and contain only ONE normative paragraph after the rule number. The normative text has been elaborated with the intention of being self-sufficient to implement an *encoding service*. However, the consequences or the reasoning behind a rule are sometimes difficult to extract from the rule itself. In these cases, after the normative text, informative (non normative) justifications, clarifications and examples can follow.

Rules have been classified in subsections for clarification purposes only. All rules form a single corpus and need to be applied together.

Also note that we target *JSON Schemas* and have @context documents to guide the JSON-LD to RDF conversions and give semantics of objects and attribute names. This JSON Schemas and @context documents are generic and will be applied to several JSON instances. Actually, next clause Rules for encoding JSON-LD instances conformant to the UML model targets the JSON instances.

In practice, these rules inform UML application schema conversion implementors. In other words, it can be applied in a future automatic UML application schema conversion tool (e.g. ShapeChange).

6.1. Root element

Rule 1.1: The JSON schema SHALL declare the first type as "object"

Example of the start of a JSON Schema file that defines the root property as an object.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Coverage objects",
    "description": "Schema for Coverage objects",
    "type": "object"
}
```

Even if a JSON-LD could also be of the array type (or a simple type) in encoding a root class coming from UML, this UML class is represented by a single the root element.

6.2. Namespaces

Rule 2.1: The *root @context document* SHALL contain a list of properties with the names of the abbreviated namespaces and value of the full namespace URI for the properties of the root element

of the JSON document.

Example of a fragment of the root "@context" document (coverage-context.json) containing definition of the abbreviated and full namespace.

```
{
    "@context":
    {
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "cis": "http://www.opengis.net/cis/1.1/",
        [...]
    }
}
```

Rule 2.2: The *root "@context" document* SHALL define a property "id" as "@id" and a property "type" as "@type".

Example of a fragment of the root "@context" document where id and type are defined.

```
{
    "@context":
    {
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "cis": "http://www.opengis.net/cis/1.1/",

        "id": "@id",
        "type": "@type"
        [...]
    }
}
```

The properties starting with "@" are a bit more difficult to access in JavaScript than the others because they can NOT be called using "coverage.@id" but using the array notation that is less *natural* "coverage[@id]". The existence of this rule allows for using "context.id" and "context.type" instead for "@id" and "@type" in JavaScript.

Rule 2.3: For all abbreviated namespaces, the correspondence to full namespace URI SHALL be listed in a @context document. If the document has more than one "@context", and the abbreviated namespace will be used only inside a particular JSON object that has a "@context" property, the definition SHALL be done in this "@context" (instead of using the root "@context").

6.3. Objects

Rule 3.1: In the JSON schema, each JSON object (including the root) SHALL define "id" property with *type* "string" and *format* "uri". This property SHALL be defined as mandatory (*required*).

Example of the addition of the necessary id's

```
{
    [...]
    "type": "object",
    "required": [ "id", ... ],
    "properties": {
        "id": { "type": "string", "format": "uri"},
        [...]
    }
}
```

Rule 3.2: In a JSON schema, each JSON object (including the root) SHALL define a "type" property with *type enum*. The *enum* will enumerated the UML class type name. The value of the *type* property of each JSON object in a JSON instance SHALL not contain an abbreviated namespace prefix. UML class type names are usually in UpperCamelCase and the values of the *enum* will be too.

Example of the addition of the necessary id's

```
{
    "seviceIdentification": {
        "title": "Sevice identification",
        "type": "object",
        "required": [ "id", "type", ... ],
        "properties": {
            "type": { "enum": ["SeviceIdentification"] },
        }
    }
}
```

Rule 3.3: The possible values of the *type* property SHALL be listed as a property name of a @context document with a value of the abbreviated namespace, a ":" character and the name again.

Example of the definition of types in JSON-LD.

```
{
    "@context":
    {
        "md": "http://www.opengis.net/md",
        "examples": "http://www.opengis.net/cis/1.1/examples/"

        "id": "@id",
        "type": "@type",
        "MD_LegalConstraints": "md:MD_LegalConstraints"
    },
    "type": "MD_LegalConstraints",
    "id": "examples:CIS_05_2D",
    ...
}
```

Using the types values without namespace in the JSON object definitions allows for creating a JSON schema that is able to correctly enumerate the possible values of a type property without the abbreviated namespace. This way the abbreviated namespace may vary from one instance without affecting the common JSON schema. In contrast, "id" values are expected to be different in each instance (and their values are not verified by the JSON schema) so they should contain the abbreviated namespace.

If an object can present more that one type (e.g. there is a one or more generalized classes of a more generic class) the all the alternative types are listed in the appropriate @context section.

Example of more that one type definition for a class.

```
{
    "@context":
    {
        "MD_Constraints": "md:MD_Constraints",
        "MD_LegalConstraints": "md:MD_LegalConstraints",
        "MD_SecurityConstraints": "md:MD_SecurityConstraints"
    }
}
```

Again, there is no need to list the Abstract types due to they cannot be instantiated in JSON-LD instances.

In a JSON-LD instance, when UML generalization is used to derive more specific classes from a generic class, the value of the property "type" will be the specialized class type name, instead of the generic class name. This behavior will favor both old and the new implementations. Old implementations will be able to ignore the "type" value, identify the object name and read the common properties from the generic class. New implementation will recognize the "type" value and be prepared to find the specialized properties.

Rule 3.4: A UML *Union* SHALL be expressed as a JSON object defined as an *anyOf* array. Each anyOf item SHALL be defined as one property (plus the *type* and *id* properties) that are defined as required. All *type* properties SHALL be identically defined.

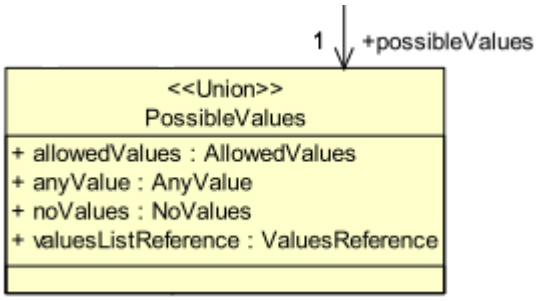

Figure 2. UML model for a generalized class ServiceIdentification.

JSON Schema fragment example of a UML Union encoded in JSON.

```
"possibleValues": {
    "title": "Possible Values list",
    "type": "object",
    "anyOf": [
        {
            "required": [ "type", "allowedValues"],
            "properties": {
                "type": { "enum": ["PossibleValues"] },
                "allowedValues": { "$ref": "#/definitions/AllowedValues" }
            }
        },{
            "required": [ "type", "anyValue"],
            "properties": {
                "type": { "enum": ["PossibleValues"] },
                "anyValue": { "$ref": "#/definitions/AnyValue" }
            }
        },{
            "required": [ "type", "noValues"],
            "properties": {
                "type": { "enum": ["PossibleValues"] },
                "noValues": { "$ref": "#/definitions/NoValues" }
            }
        },{
            "required": [ "type", "valuesListReference"],
            "properties": {
                "type": { "enum": ["PossibleValues"] },
                "valuesListReference": { "$ref": "#/definitions/ValuesReference" }
            }
        }
    ]
}
```

Rule 3.5: A UML class that is a *aggregation* or a *composition* of a parent class, SHALL be considered equivalent to a JSON complex property of the parent object (considering it as the same as a UML complex attribute). The name of the source extreme of the relation (the tip of the arrow) will be considered the name of the complex *attribute* and the *class name* will be considered the *type* of the property.

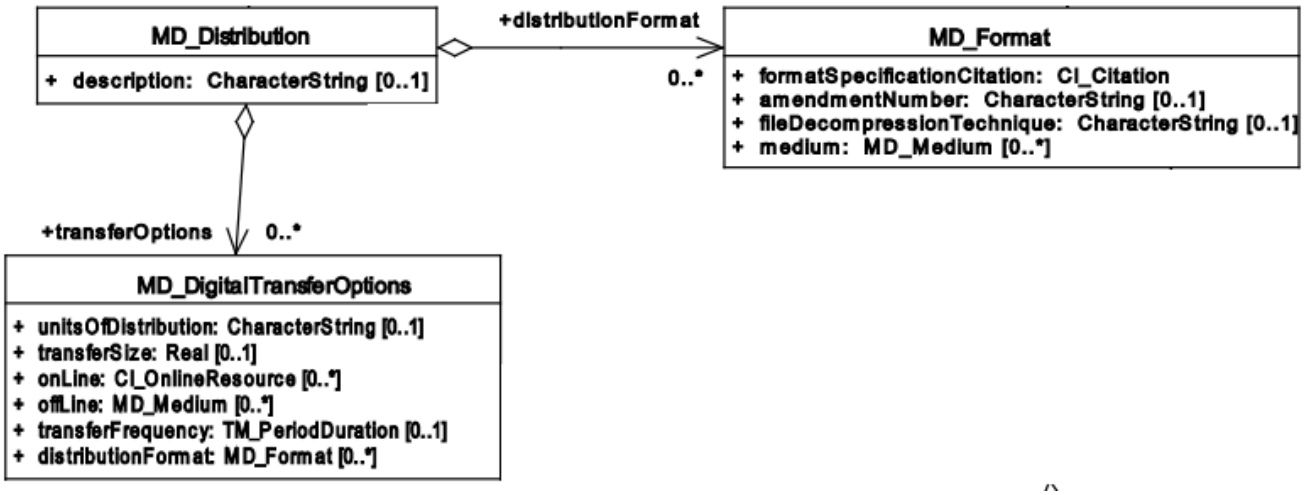

Figure 3. UML model showing a class that is defined as an aggregation of MD_Distribution and also as a complex data type MD_DigitalTransferOptinos

JSON Schema fragment example of a class used as an aggregation and as a data type (the example ignores the tranferOption multiplicity for simplicity)

```
{
    ...
    "distributionInfo": {
        "description": {"type": "string"},
        "distributionFormat": {"$ref": "#/definitions/MD_format"},
        "transferOptions": {
            ...
            "distributionFormat": {"$ref": "#/definitions/MD_format"}
        }
    }
}
```

Please consider the rules for complex attributes and properties below.

6.4. Object attributes: names

Rule 4.1: All UML class attributes SHALL be listed in a @context object. If the document has more than one "@context", the property SHALL be defined in the "@context" that is closer to it (with the exception of "id" and "type" that are defined in the root "@context" document and always have global scope). The value of the listed properties has to be the abbreviated namespace, the ":" character and the name of the property again.

Defining the properties in the "@context" that is closer to it makes the definition local to the object where the "@context" belongs.

Example of the addition of the abbreviate namespace to properties

```
{
    "domainSet":{
        "@context":
        {
            "generalGrid": "cis:generalGrid",
            "axis": "cis:axis",
            "axisLabel": "cis:axisLabel",
            "lowerBound": "cis:lowerBound",
            "upperBound": "cis:upperBound"
        },
        "generalGrid":{
            "axis": [{
                "type": "IndexAxisType",
                "axisLabel": "i",
                "lowerBound": 0,
                "upperBound": 2
            },{
                "type": "IndexAxisType",
                "axisLabel": "j",
                "lowerBound": 0,
                "upperBound": 2
            }]
        }
    }
}
```

Rule 4.2: All UML class attributes (as well as composition and aggregations) SHALL be listed as *properties* of the object in the JSON Schema.

Example of axisLabel, lowerBound and upperBound properties definition.

```
{
    "type": "object",
    "properties": {
        "type": { "enum": [ "IndexAxisType"] },
        "axisLabel":  { "type": "string" },
        "lowerBound":  { "type": "number" },
        "upperBound": { "type": "number" }
    }
}
```

6.5. Object attributes: multiplicity

Rule 5.1: Attributes, aggregations and compositions of an object with multiplicity 0 or 1 in the UML will be listed as JSON properties in the JSON schema.

Example of axisLabel, lowerBound and upperBound properties definition.

```
    "axis": {
        "type": "object",
        "properties": {
            "type": { "enum": [ "IndexAxisType"] },
            "axisLabel":  { "type": "string" },
            "lowerBound":  { "type": "number" },
            "upperBound": { "type": "number" }
        }
    }
```

Rule 5.2: Attributes, aggregations and compositions of an object with multiplicity more than 0 in the UML will have their names listed in the array of "required" properties in the JSON schema

Example of the "required" list of property.

```
    "axis": {
        "type": "object",
        "required": [ "type", "axisLabel", "lowerBound", "upperBound" ],
        "properties": {
            "type": { "enum": [ "IndexAxisType"] },
            "axisLabel":  { "type": "string" },
            "lowerBound":  { "type": "number" },
            "upperBound": { "type": "number" }
        }
    }
```

Rule 5.3: Attributes, aggregations and compositions with multiplicity more that 1 in the UML will be encoded as JSON properties of the *type* "array" in the JSON schema.

In JSON instances the arrays of JSON objects will defined the type for each member of the array. The "type" can be different from the other members of the array (JavaScript allows arrays that are heterogenous in types) but all "type" values will be a generalization of the same generic UML class.

Example of JSON schema for numeric properties with multiplicity more than 1.

```
{
    "coordinates": {
        "type": "array",
        "items": { "type": "number" }
    }
}
```

Rule 5.4: A property defined in the UML as *ordered* and with multiplicity more than 1, SHALL be defined as "@container": "@list" in a @context document.

There is an important singularity in JSON-LD about this. When a JSON document is converted into JavaScript all JSON arrays become automatically with order. However in JSON-LD the situation is

the opposite, and by default arrays are NOT considered as ordered when converting JSON-LD arrays to RDF. There is a technical reason behind this: ordered arrays require much more RDF code and the conversion does *not* result in a *nice* RDF code. For that reason if the a property is made as *ordered* in the UML, we have to explicitly indicate this in a @context. Please limit the use this parameter only when order is really important.

6.6. Object attributes: data types

Rule 6.1: Numeric attributes in the UML SHALL have "type": "number" in the JSON Schema.

Example of JSON schema for anyURL properties with multiplicity 0 or 1.

```
{
    "lowerBound":  { "type": "number" },
    "upperBound": { "type": "number" }
}
```

In JSON there is no able to distinction between different numeric data types: E.g. Integer, Float, Double etc and they all became *number*.

Numeric attributes with multiplicity more than 1 will have "type": "array" and "items": {"type": "number"} in the JSON Schema.

Example of JSON schema for numeric properties with multiplicity more than 1.

```
{
    "coordinates": {
        "type": "array",
        "items": { "type": "number" }
    }
}
```

Rule 6.2: Boolean attributes SHALL have "type": "boolean" in the JSON Schema.

Boolean attributes with multiplicity more than 1 will have "type": "array" and "items": {"type": "boolean"} in the JSON Schema.

Rule 6.3: String attributes SHALL have "type": "string" in the JSON Schema.

Example of JSON schema for anyURL properties with multiplicity 0 or 1.

```
{
    "axisLabel":  { "type": "string" }
}
```

If there is a reason to believe that the attribute has been defined as a UML string to allow both numbers or strings (depending of the case), defining the type as an array of "number" and "string" is recommended.

Example of JSON schema for a string that can be also instantiated as a number.

```
{
    "value":  { "type": ["number", "string"] },
}
```

String attributes with multiplicity more than 1 will have "type": "array" and "items": {"type": "string"} in the JSON Schema.

Example of JSON schema for string properties with multiplicity more than 1.

```
{
    "axisLabels":
    {
        "type": "array",
        "items": { "type": "string" }
    }
}
```

Rule 6.4: anyURL attributes with multiplicity 0 or 1 SHALL have "type": "string" and "format": "uri" in the JSON Schema.

Example of JSON schema for anyURL properties with multiplicity 0 or 1.

```
{
    "srsName": { "type": "string", "format": "uri"}
}
```

anyURL attributes with multiplicity more that 1 will have "type": "array" and "items": {"type": "string", "format": "uri"} in the JSON Schema.

Example of JSON schema for anyURL properties with multiplicity more than 1.

```
{
    "srsNames":
    {
        "type": "array",
        "items": { "type": "string", "format": "uri"}
    }
}
```

Rule 6.5: A property that has the value anyURI SHALL be described as "@type": "@id" in a JSON @context document

Example of the definition of a value anyURI in JSON-LD

```
{
    "@context":
    {
        "srsName": {"@id":"swe:srsName", "@type": "@id"}
    }
}
```

The reasoning behind this is that property values defined without "@type": "@id" are considered literals when converted to RDF. Property values defined with "@type": "@id" considered as URIs when converted to RDF.

Rule 6.6: Complex type attributes SHALL have "type": "object" in the JSON Schema.

Example of JSON schema for complex properties with multiplicity 0 or 1.

```
{
    "generalGrid":{
        "title": "General Grid",
        "description": "General Grid",
        "type": "object"
    }
}
```

Complex type attributes with multiplicity more than 1 will have "type": "array" and "items": {"type": "object"} in the JSON Schema

Example of JSON schema for complex properties with multiplicity more than 1.

```
{
    "axis": {
        "type": "array",
        "items": {
            "type": "object"
        }
    }
}
```

Rule 6.7: A UML complex type attribute SHALL be encoded as JSON Object properties or as an Array of JSON Object type properties in a JSON-LD instance.

6.7. Object attributes: null values

Rule 7.1: For a attribute that can have null values, an array of types combining the variable data type and "null" SHALL be used.

Example of a nullable value.

```
{
    "lowerBound":  { "type": ["number", "null"] },
}
```

6.8. Object attributes: enumerations and codelists

Rule 8.1: A UML "enumeration" SHALL be encoded as an "enum" in a JSON schema. Enumerations SHALL be listed in the "definitions" section of the JSON schema to be able to reuse them as needed.

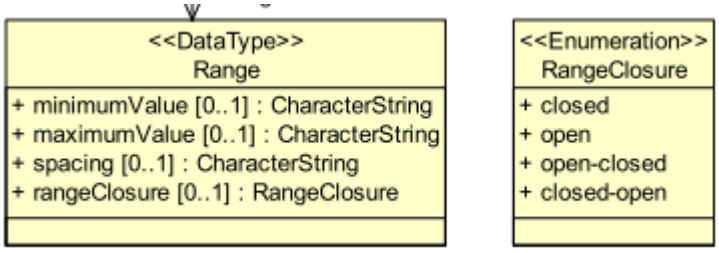

Figure 4. UML model for a enumeration.

Example of enumerations

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "rangeClosure": {"$ref": "#/definitions/RangeClosure"},
        ...
    }
    "definitions": {
        "RangeClosure": {
            "title": "Values of RangeClosure enumeration",
            "enum": ["closed", "open", "open-closed", "closed-open"]
        }
    }
}
```

Rule 8.2: A UML "codelist" SHALL be encoded as an oneOf "enum" or "string" in a JSON schema. Codelists SHALL be listed in the *definitions* section of the JSON schema to be able to reuse them as needed.

Example of the addition of the necessary id's

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "required": [ "codelist_a"],
    "properties": {
        "codelist_a": { "$ref": "#/definitions/codelist1" },
        ...
    },
    "definitions": {
        "codelist": {
            "oneOf": [
                {
                    "enum": [ "a2", "b2", "c2" ]
                },{
                    "type": "string"
                }
            ]
        }
    }
}
```

The reason for this is that codelists are considered extendable, and in practice they should support any value. See a good discussion on how to encode enumeration a codelist in JSON Schema here: http://grokbase.com/t/gg/json-schema/14b79eqgqq/code-list-enum-extension.

6.9. Objects: Data types and inheritance

Rule 9.1: If a UML class is defined as *DataType* (and potentially used in more that one place in the UML model) it SHALL be defined in the *definitions* section of the JSON schema and referenced by each attribute that is declared of this *DataType*

Actually, it is highly recommended that all UML classes are defined in the *definitions* sections. Only the objects defined in the *definitions* and the root object can be referenced from another JSON schema.

JSON Schema fragment example of the use of definitions section.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "seviceIdentification": { "$ref": "#/definitions/SeviceIdentification" },
        ...
    },
    "definitions": {
        "ServiceIdentification": {
            "title": "Service identification",
            "type": "object",
            "required": [ "type", "serviceType", "serviceTypeVersion" ],
            "properties": {
                "type": { "enum": ["ServiceIdentification"] },
                "serviceType": { "$ref": "#/definitions/Code"},
                "serviceTypeVersion":  {
                    "type": "array",
                    "items": {"type": "string" }
                },
                "profile":  {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "fees": {"type": "string"},
                "accessConstraints": {"type": "string"}
            }
        }
  }
}
```

Example of using "definitions" section to define an object that can be used in more that one place emulating the UML data type behavior.

```
{
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "profile" : { "$ref": "#/definitions/links" },
        "links" : {
            "type": "array",
            "items": { "$ref": "#/definitions/links" }
        }
    }
    "definitions": {
        "links": {
            "title": "links",
            "description": "Properties that all types of links have. It mimics the
Atom link",
            "required": [ "href" ],
            "properties": {
            "href": {"type": "string", "format": "uri" },
            "type" : { "type": "string" },
            "title" : { "type": "string" },
            "lang" : { "type": "string" }
            }
            }
        }
    }
```

Rule 9.2: If a class is generalized into other classes in the UML, the JSON schema SHALL define the main class properties in the definitions section with the UML class name and the word "Properties" (not including the "type"). If the main class is not abstract, it SHALL then be defined with the UML call name by combining the previous "properties" and the "type" property with *allOf*. The generalized class will do the same and combine the main properties, the "type" and its own properties with *allOf*

If the main class is empty (it has no attributes) this rule does not apply.

Unnecessary duplication of the definition of common elements coming from the abstract class is avoided by using *$ref* and pointing to a *definitions* element in the JSON Schema. There is no mechanism to inherit properties from a previous object in JSON schema, but the suggested mechanism achieves an equivalent results.

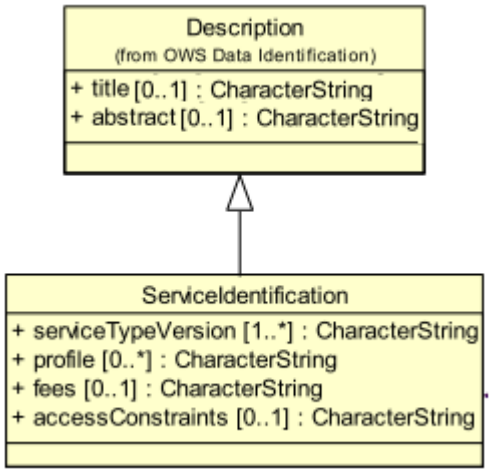

Figure 5. UML model for a generalized class ServiceIdentification.

JSON Schema fragment example of generalization of ServiceIdentification inheriting a group of properties DescriptionProperties. Note that "Description" is defined for completeness but it is not used.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "serviceIdentification": { "$ref": "#/definitions/ServiceIdentification" }
    },
    "definitions": {
        "ServiceIdentification": {
            "required": ["type"],
            "allOf": [
                { "$ref": "#/definitions/DescriptionProperties" },
                {
                    "properties":{
                        "type": {"enum": ["ServiceIdentification"]},
                        "serviceTypeVersion": {"type": "string"},
                        "profile": {"type": "string"},
                        "fees": {"type": "string"},
                        "accessConstraints": {"type": "string"}
                    }
                }
            ]
        },
        "Description": {
            "required": ["type"],
            "allOf": [
                { "$ref": "#/definitions/DescriptionProperties" },
                {
                    "properties":{
                        "type": {"enum": ["Description"]}
                    }
                }
            ]
        },
        "DescriptionProperties": {
            "properties": {
                "id": {"type": "string", "format": "uri"},
                "title": {"type": "string"},
                "abstract": {"type": "string"}
            }
        }
    }
}
```

JSON fragment that validates with he previous example

```
{
    "serviceIdentification": {
        "type": "ServiceIdentification",
        "title": "My WMS server",
        "abstract": "This WMS server is mine",
        "serviceTypeVersion": "1.1.1",
        "profile": "http://www.opengis.net/profiles/nga"
    }
}
```

Rule 9.3: If a class is generalized into more than one classes in the UML (which forces the instance to choose one among of them), the JSON Schema SHALL define an object that offers the different options using the *oneOf* property.

When the main class is *abstract*, there is no need to define it as a data type due to it cannot be instantiated in JSON-LD instances.

In the following example, the class MD_Constraints is generalized into 2 different classes: MD_LegalConstraints and MD_SecurityConstraints. The pattern used in the example is a bit different for the one in the example of Rule 3.4. Testing the us of *oneOf* to select between object definitions that contains *allOf* including the same $ref fails to work. To solve this, we define *MD_LegalConstraintsAdditions* and *MD_SecurityConstraintsAdditions* that only defines the non common properties. Then A new object *Alternatives_MD_Constraints* offers all the alternatives available with *oneOf* and add the common properties *MD_ConstraintsProperties* at the end with *allOf*.

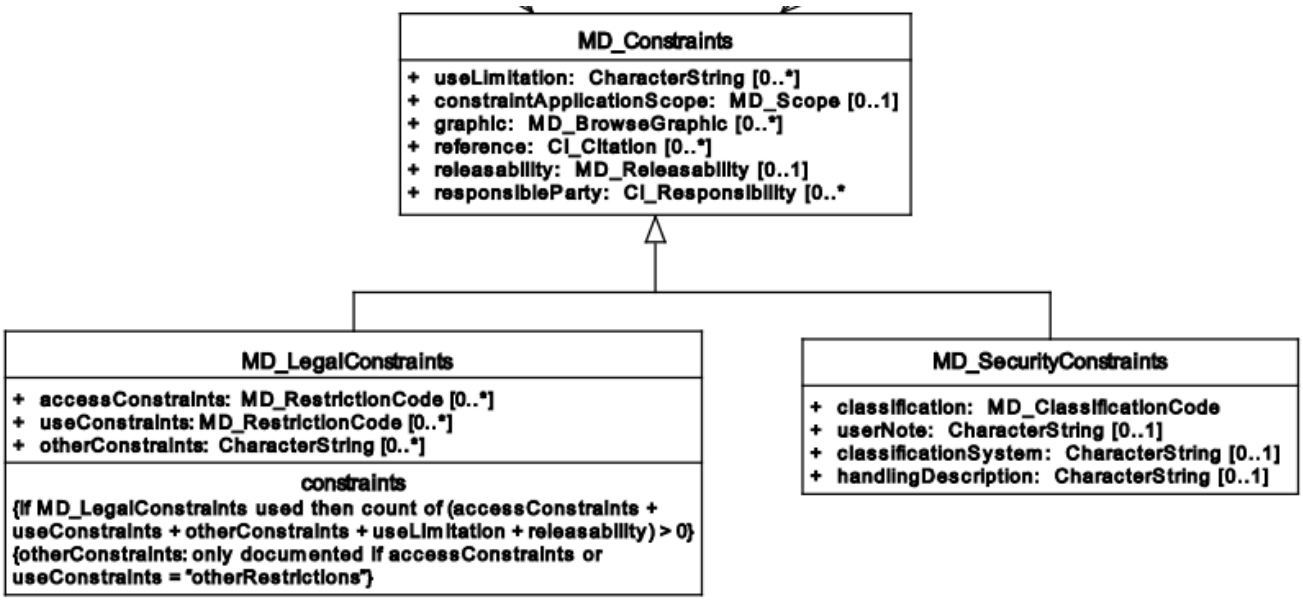

Figure 6. UML model for a generalized class from MD_Constraints.

Example of multiple generalization

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
```

```
    "type": "object",
    "properties":
    {
        "resourceConstraints": {
            "$ref": "#/definitions/Alterantives_MD_Constraints"
        }
    },
    "definitions": {
        "Alterantives_MD_Constraints": {
            "type": "object",
            "allOf": [
                { "$ref": "#/definitions/MD_ConstraintsProperties" },
                { "oneOf": [
                        {"$ref": "#/definitions/MD_SecurityConstraintsAdditions"},
                        {"$ref": "#/definitions/MD_LegalConstraintsAdditions"},
                        {"$ref": "#/definitions/MD_ConstraintsAdditions"}
                    ]
                }
            ]
        },
        "MD_LegalConstraintsAdditions": {
            "required": ["type"],
            "properties":{
                "type": {"enum": ["MD_LegalConstraints"]},
                "accessConstraints": {"type": "object"},
                "otherConstraints": {"type": "string"}
            }
        },
        "MD_SecurityConstraintsAdditions": {
            "required": ["type"],
            "properties":{
                "type": {"enum": ["MD_SecurityConstraints"]},
                "useNote": {"type": "string"},
                "classificationSystem": {"type": "string"},
                "handlingDescription": {"type": "string"}
            }
        },
        "MD_ConstraintsAdditions": {
            "required": ["type"],
            "properties":{
                "type": {"enum": ["MD_Constraints"]}
            }
        },
        "MD_ConstraintsProperties": {
            "properties": {
                "id": {"type": "string", "format": "uri"},
                "useLimitation": {"type": "string"},
                "constraintApplicationScope": {"type": "object"}
            }
        }
    }
```

```
}
```

6.10. Object libraries and multiple schemas

UML classes can be structured in a modular way in packages. In this case there is a need to use more than one schema file. The core schema offers a set of *datatypes* in the *definitions* section. This core schemas can be reused by other schemas pointing the definition of the right objects in the core schemas using a full path to them.

Rule 10.1: Each UML class packages SHALL be described in the *definitions* in a JSON schema. Schemas that reuse other UML classes in other packages SHALL point to them using a full path.

Example of a core JSON schema (called ServiceMetadata_schema.json) for a common class in OWS common package

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "definitions": {
        "ServiceIdentification": {
            "title": "Service identification",
            "description": "Metadata about this specific server. The contents and
organization of this section should be the same for all OWSs. ",
            "type": "object",
            "properties": {
                "type": { "enum": ["ServiceIdentification"] },
                "serviceType": { "$ref": "#/definitions/Code"},
                "serviceTypeVersion":  {
                    "type": "array",
                    "items": {"type": "string" }
                },
                "profile":  {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "fees": {"type": "string"},
                "accessConstraints": {"type": "string"}
            }
        }
    }
}
```

Example of a another JSON schema (called WMSServiceMetadata_schema.json) for a common class in OWS common package

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "WMS Service Metadata root object",
    "required": [ "type", "version"],
    "properties": {
        "type": { "enum": [ "WMSServiceMetadata"] },
        "id":  { "type": "string" },
        "version": { "type": "string"},
        "serviceIdentification": { "$ref":
"ServiceMetadata_schema.json/#/definitions/ServiceIdentification" },
    }
}
```

Example of a JSON instance (called WMSServiceMetadata.json) validated with the WMSServiceMetadata_schema.json file.

```
{
    "type": "WMSServiceMetadata",
    "version": "1.4",
    "serviceIdentification": {
        "type": "ServiceIdentification",
        "serviceType": {
            "type": "Code",
            "code": "WMS"
        },
        "serviceTypeVersion": ["1.4"],
        ...
    }
}
```

6.11. Other considerations

6.11.1. Large arrays

A property containing a large array of values (or coordinates), where the order is important, should not be defined in @context objects if an alternative representation of the same information can be provided by other means. Not defining the long ordered arrays properties in @context avoids them to be converted into RDF. Ordered arrays are nasty and very verbose, double ordered arrays cannot even be converted. *Alternative representations* can be "string literals" (encoding the arrays in a text that is more compact; e.g. coordinates encoded as WKT) or links to URLs in a non RDF format containing this information (such as NetCDF or GeoTIFF files). This is further discussed in the section Data formats encoding with specific focus on coordinate arrays but not only.

Chapter 7. Rules for encoding JSON-LD instances conformant to the UML model

The rules provided in this section and in the previous one Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams have been created in a very formal way using normative language (i.e. using SHALL as a indication of obligation). Even if this style could be considered not appropriate for an ER, it has been done with the objective of facilitating the migration of the rules into normative document or a standard in the near future. In this sense, they are numbered and contain only ONE normative paragraph after the rule number. The normative text has been elaborated with the intention of being self-sufficient to implement an *encoding service*. However, the consequences or the reasoning behind a rule are sometimes difficult to extract from the rule itself. In these cases, after the normative text, informative (non normative) justifications, clarifications and examples can follow.

Rules have been classified in subsections for clarification purposes only. All rules form a single corpus and need to be applied together.

Also note that we target *JSON-LD instances* that are validated using JSON Schema and have @context documents to guide the JSON-LD to RDF conversions and give semantics of objects and attribute names. Actually, the previous clause Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams targets the JSON schemas and @context documents

In practice, these rules inform JSON instance writers.

7.1. General

Rule 1.1: A JSON-LD instance SHALL follow the JSON syntax specified in the IETF RFC7159 The JavaScript Object Notation (JSON) Data Interchange Format.

This *RFC7159 conformance rule* actually implies several other rules stated in RFC7159 and that will not be repeated here but need also considered (e.g. JSON Strings values will be encoded in UTF8 in a JSON-LD instance)

Rule 1.2: A JSON-LD instance SHALL validate with the JSON schema derived from the corresponding UML class diagram model.

This rule imposes automatically all the JSON schema specific rules for property values encoding (e.g. Numeric attributes will be encoded as JSON Numbers or Boolean attributes will have the *true* or *false* values or Arrays of them in a JSON-LD instance). Most of this rules will not be included here because they are imposed by the JSON schema validation process. Some others are included here are rules for clarity.

Several tools exist to validate JSON instances with a JSON schema automatically (such as XML Validator Buddy).

7.2. Root element

Rule 2.1: The root element in the JSON-LD instance shall have no name.

The JSON file is supposed to be loaded in a JavaScript object that will have a name identifying the object.

Example of code that load the JSON objects in the coverage variable

```
var coverage=JSON.parse(json_serialized_text);
```

Rule 2.2: The first property of the root element of a JSON-LD instance SHALL have the name "@context". The root "@context" property SHAL have a value consisting in an array formed two parts: a reference to a @context document (called *root @context document*) and a embedded context (called *root embedded context*).

Rule 2.3: A JSON-LD instance *root embedded context* SHALL contain at least a property with the name of the abbreviated namespace and value of the full namespace URI of the identifiers present on this JSON document.

Example of a fragment of the first part of the root object of a JSON document. If references the @context document in the last example and also embeds the definition of the id's namespace.

```
{
    "@context": [
        "http://localhost/json-ld/coverage-context.json",
        {
            "examples": "http://www.opengis.net/cis/1.1/examples/"
        }]
}
```

It is important to note that it can be other "@context" objects in the interior of any object (as the first property). This is particularly useful to define properties with local scope.

7.3. Objects

Rule 3.1: In a JSON-LD instance, each JSON object (including the root) SHALL contain a "id" property that uniquely identifies it (this is enforced by the JSON Schema validation). The value of this property shall be composed by a abbreviated namespace of the identifiers, a ":" character and an identifier name.

Example of the addition of the necessary id's

```
{
    "@context": [
        "http://localhost/json-ld/coverage-context.json",
        {
            "examples": "http://www.opengis.net/cis/1.1/examples/"
        }]  "id": "examples:CIS_05_2D",
    [...]
    "domainSet":{
        "id": "examples:CIS_DS_05_2D",
        "generalGrid":{
            "id": "examples:CIS_DS_GG_05_2D",
            [...]
        }
    }
}
```

Rule 3.2: In a JSON-LD instance, each JSON object (including the root) SHALL include a "type" property (this is enforced by the JSON Schema validation)

Parsers of the JSON encoding are not required to read and understand the "type" property but they can use it to redirect the code to the right parser of a code fragment or to identify a specialized type among all possible ones of a generic one (among the sub-classified types).

Rule 3.3: The value of the *type* property of each JSON object in a JSON instance shall not contain an abbreviated namespace (actually, valid nomes are enforced by the JSON Schema validation).

Using the types values without namespace in the JSON object definitions allows for using a JSON schema that is able to correctly enumerate the possible values of a type property without the abbreviated namespace. This way the abbreviated namespace may vary without affecting the common JSON schema. In contrast, "id" values are expected to be different in each instance (and their values are not verified by the JSON schema) so they should contain the abbreviated namespace.

Chapter 8. Example on applying JSON to OWS Common UML models.

OWS Common specifies many of the aspects that are, or should be, common to all or multiple OGC Web Service (OWS) interface Implementation Standards. These common aspects are primarily some of the parameters and data structures used in operation requests and responses. In the current version of OWS Common, XML is the only proposed encoding. This clause proposes a JSON encoding that is the result of applying the JSON and JSON-LD rules described in previous clauses to OWS Common 2.0 (OGC 06-121r9).

This clause represents a complete Change Request for OWS Common 2.0. To make the adoption of this JSON encoding as a standard, an encoding extension in an independent document is suggested containing only the JSON/JSON-LD encoding to complement the current XML encoding.

8.1. ServiceMetadata document in JSON

A service metadata document is the normal response to a client from performing the GetCapabilities operation, and should contain the metadata appropriate to the specific server for the specific OWS. In addition to defining the service as a whole, the service exposes *resources* of some sort that will also be described. For example, if the server is serving *features*, the service metadata document will include metadata about *features types*. OWS Common specifies a minimum set of metadata for the *resources* too.

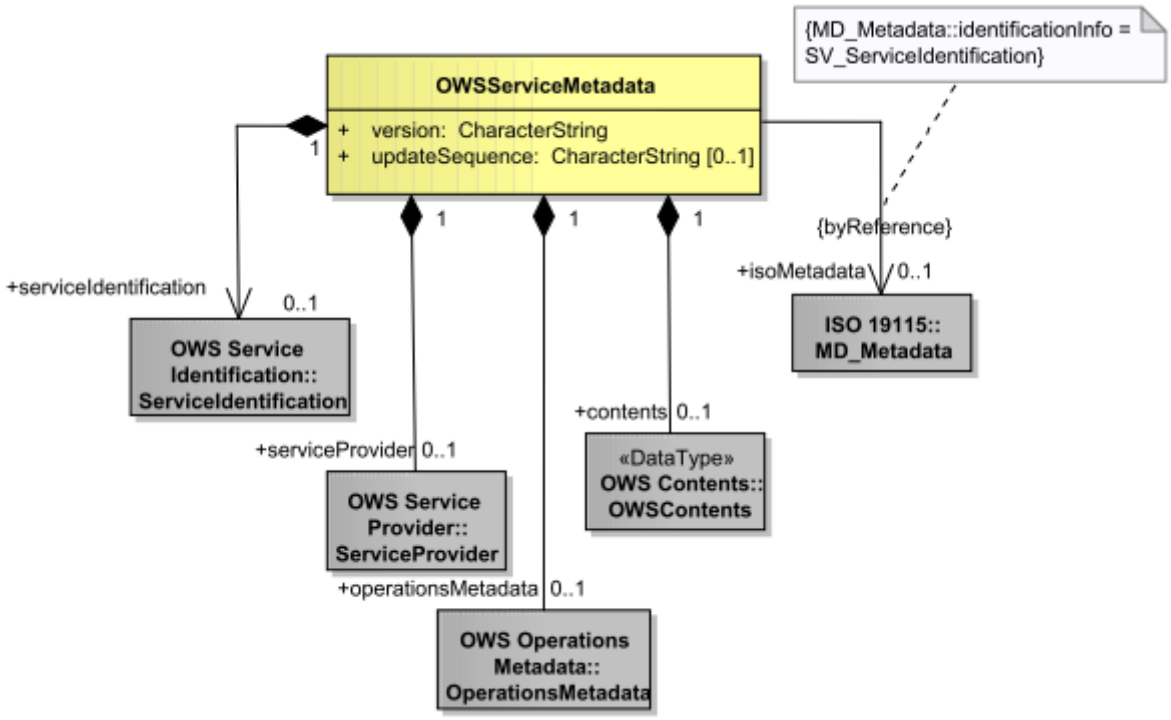

Figure 7. GetCapabilities operation response UML class diagram

This is the JSON Schema that has been elaborated following the rules in Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams manually and applying them to the OWS Common UML Class diagrams.

8.2. OWS Common JSON Schema

Full OWS Common JSON Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Service Metadata root object",
    "description": "A service metadata document is the normal response to a client
from performing the GetCapabilities operation, and contains metadata appropriate to
the specific server for the specific OWS.",
    "type": "object",
    "required": [ "type", "version"],
    "properties": {
        "type": { "enum": [ "ServiceMetadata"] },
        "id":  { "type": "string" },
        "version": { "type": "string"},
        "updateSequence": { "type": "string"},
        "serviceIdentification": { "$ref": "#/definitions/ServiceIdentification" },
        "serviceProvider": { "$ref": "#/definitions/ServiceProvider"},
        "operationsMetadata": { "$ref": "#/definitions/OperationsMetadata"},
        "contents": { "$ref": "#/definitions/OWSContents"},
        "isoMetadata": { "$ref": "#/definitions/IsoMetadata"},
        "languages": { "$ref": "#/definitions/Languages"}
    },
    "definitions": {
        "DescriptionProperties": {
            "title": "Description properties",
            "type": "object",
            "properties": {
                "title": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/LanguageString"}
                },
                "abstract": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/LanguageString"}
                },
                "keywords": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/Keywords"}
                }
            }
        },
        "LanguageString": {
            "title": "Language string",
            "type": "object",
            "required": [ "type", "value" ],
            "properties": {
                "type": { "enum": ["LanguageString"] },
                "value": {"type": "string"},
                "lang": {"type": "string"}
```

```
            }
        },
        "Keywords": {
            "title": "Keywords",
            "type": "object",
            "required": [ "type", "keyword" ],
            "properties": {
                "type": { "enum": ["Keywords"] },
                "keyword": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/LanguageString"}
                },
                "keywordsType": { "$ref": "#/definitions/code"}
            }
        },
        "ServiceIdentification": {
            "title": "Service identification",
            "description": "Metadata about this specific server. The contents and
organization of this section should be the same for all OWSs. ",
            "type": "object",
            "required": [ "type", "serviceType", "serviceTypeVersion" ],
            "allOf": [
                { "$ref": "#/definitions/DescriptionProperties" },
                {
                    "properties": {
                        "type": { "enum": ["ServiceIdentification"] },
                        "serviceType": { "$ref": "#/definitions/Code"},
                        "serviceTypeVersion":  {
                            "type": "array",
                            "items": {"type": "string" }
                        },
                        "profile":  {
                            "type": "array",
                            "items": {"type": "string"}
                        },
                        "fees": {"type": "string"},
                        "accessConstraints": {"type": "string"}
                    }
                }
            ]
        },
        "ServiceProvider": {
            "title": "Service provider",
            "description": "Metadata about the organization operating this server. The
contents and organization of this section should be the same for all OWSs.",
            "type": "object",
            "required": [ "type", "providerName" ],
            "properties": {
                "type": { "enum": ["ServiceProvider"] },
                "providerName": {"type": "string"},
                "serviceContact": { "$ref": "#/definitions/ResponsibleParty"},
```

```
                "providerSite": { "$ref": "#/definitions/OnlineResouce"}
            }
        },
        "OperationsMetadata": {
            "title": "Operations metadata",
            "description": "Metadata about the operations specified by this service
and implemented by this server, including the URLs for operation requests. The basic
contents and organization of this section shall be the same for all OWSs, but
individual services may add elements and/or change the optionality of optional
elements",
            "type": "object",
            "required": [ "type", "operation" ],
            "properties": {
                "type": { "enum": ["OperationsMetadata"] },
                "operation": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/Operation"}
                },
                "parameter": {
                    "type": "array",
                    "items": { "$ref": "#/definitions/Domain"}
                },
                "constraint": {
                    "type": "array",
                    "items": { "$ref": "#/definitions/Domain"}
                },
                "extendedCapabilities": {"type": "object"}
            }
        },
        "IsoMetadata": {
            "title": "ISO metadata",
            "description": "Metadata about this specific server. The contents and
organization of this section should be the same for all OWSs.",
            "type": "object"
        },
        "Code": {
            "title": "code",
            "type": "object",
            "required": [ "type", "code" ],
            "properties": {
                "type": { "enum": ["Code"] },
                "code": {"type": "string"},
                "codeSpace": { "type": "string", "format": "uri"}
            }
        },
        "Languages": {
            "title": "languages",
            "type": "object",
            "required": [ "type", "language" ],
            "properties": {
                "type": { "enum": ["Languages"] },
```

```
                "language": {
                    "type": "array",
                    "items": {"type": "string", "description": "This language
identifier shall be as specified in IETF RFC 4646."}
                }
            }
        },
        "OnlineResouce": {
            "type": "object",
            "required": [ "type", "linkage" ],
            "allOf": [
                { "$ref": "#/definitions/OnlineResouceProperties" },
                { "properties": {"type": { "enum": ["OnlineResource"] } }}
            ]
        },
        "OnlineResouceProperties": {
            "type": "object",
            "properties": {
                "linkage": { "type": "string", "format": "uri"}
            }
        },
        "Domain": {
            "type": "object",
            "required": [ "type", "name" ],
            "allOf": [
                { "$ref": "#/definitions/UnNamedDomainProperties" },
                { "properties": {
                    "type": { "enum": ["Domain"] },
                    "name": { "type": "string"}
                }}
            ]
        },
        "UnNamedDomain": {
            "type": "object",
            "required": [ "type", "name" ],
            "allOf": [
                { "$ref": "#/definitions/UnNamedDomainProperties" },
                { "properties": {"type": { "enum": ["UnNamedDomain"] } }}
            ]
        },
        "UnNamedDomainProperties": {
            "title": "UnNamed domain properties",
            "type": "object",
            "properties": {
                "defaultValue": {"type": "string"},
                "possibleValues": { "$ref": "#/definitions/PossibleValues" },
                "meaning": { "$ref": "#/definitions/Meaning" },
                "dataType": { "$ref": "#/definitions/DataType" },
                "valuesUnit": {"$ref": "#/definitions/ValuesUnit" },
                "metadata": { "$ref": "#/definitions/Metadata" }
            }
```

```
        },
        "PossibleValues": {
            "title": "Possible Values list",
            "type": "object",
            "anyOf": [
                {
                    "required": [ "type", "allowedValues"],
                    "properties": {
                        "type": { "enum": ["PossibleValues"] },
                        "allowedValues": { "$ref": "#/definitions/AllowedValues" }
                    }
                },{
                    "required": [ "type", "anyValue"],
                    "properties": {
                        "type": { "enum": ["PossibleValues"] },
                        "anyValue": { "$ref": "#/definitions/AnyValue" }
                    }
                },{
                    "required": [ "type", "noValues"],
                    "properties": {
                        "type": { "enum": ["PossibleValues"] },
                        "noValues": { "$ref": "#/definitions/NoValues" }
                    }
                },{
                    "required": [ "type", "valuesListReference"],
                    "properties": {
                        "type": { "enum": ["PossibleValues"] },
                        "valuesListReference": { "$ref":
"#/definitions/ValuesReference" }
                    }
                }
            ]
        },
        "AllowedValues": {
            "title": "Possible Values list",
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["AllowedValues"] },
                "value": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "range": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/Range"}
                }
            }
        },
        "Range": {
            "title": "Range of values",
```

```
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["Range"] },
                "minimumValue": {"type": "string", "description": "Minimum value of
this range of this numeric parameter. Default is negative infinity"},
                "maximumValue": {"type": "string", "description": "Maximum value of
this range of this numeric parameter. Default is positive infinity"},
                "spacing": {"type": "string", "description": "Regular distance or
spacing between allowed values in this range"},
                "rangeClosure": {"$ref": "#/definitions/RangeClosure", "description":
"Specifies which of minimum and maximum values are included in this range"}
            }
        },
        "RangeClosure": {
            "title": "Values of RangeClosure enumeration",
            "enum": ["closed", "open", "open-closed", "closed-open"]
        },
        "AnyValue": {
            "title": "Any value empty object",
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["AnyValue"] }
            }
        },
        "NoValues": {
            "title": "No values empty object",
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["NoValues"] }
            }
        },
        "ValuesUnit": {
            "title": "Values unit",
            "type": "object",
            "oneOf": [
                {
                    "required": [ "type", "referenceSystem" ],
                    "properties": {
                        "type": { "enum": ["ValuesUnit"] },
                        "referenceSystem": {
                            "type": "array",
                            "items": {"$ref": "#/definitions/ReferenceSystem"}
                        }
                    }
                },{
                    "required": [ "type", "uom" ],
                    "properties": {
                        "type": { "enum": ["ValuesUnit"] },
```

```
                    "uom": {
                        "type": "array",
                        "items": {"$ref": "#/definitions/UOM"}
                    }
                }
            }
        ]
    },
    "ValuesReference": {
        "title": "Possible Values list",
        "type": "object",
        "required": [ "type", "name", "reference"],
        "properties": {
            "type": { "enum": ["ValuesReference"] },
            "name": {"type": "string"},
            "reference": {"type": "string", "format": "uri"}
        }
    },
    "Meaning": {
        "title": "Meaning",
        "type": "object",
        "required": [ "type", "name"],
        "allOf": [
            { "$ref": "#/definitions/DomainMetadataProperties" },
            { "properties": {"type": { "enum": ["Meaning"] }}}
        ]
    },
    "DataType": {
        "title": "Meaning",
        "type": "object",
        "required": [ "type", "name"],
        "allOf": [
            { "$ref": "#/definitions/DomainMetadataProperties" },
            { "properties": {"type": { "enum": ["DataType"] }}}
        ]
    },
    "UOM": {
        "title": "Units Of Measure",
        "type": "object",
        "required": [ "type", "name"],
        "allOf": [
            { "$ref": "#/definitions/UOM" },
            { "properties": {"type": { "enum": ["UOM"] }}}
        ]
    },
    "ReferenceSystem": {
        "title": "Reference system",
        "type": "object",
        "required": [ "type", "name"],
        "allOf": [
            { "$ref": "#/definitions/ReferenceSystem" },
```

```
            { "properties": {"type": { "enum": ["ReferenceSystem"] }}}
        ]
    },
    "DomainMetadata": {
        "title": "Domain metadata",
        "type": "object",
        "required": [ "type", "name"],
        "allOf": [
            { "$ref": "#/definitions/DomainMetadataProperties" },
            { "properties": {"type": { "enum": ["DomainMetadata"] }}}
        ]
    },
    "DomainMetadataProperties": {
        "title": "Domain metadata",
        "type": "object",
        "properties": {
            "name": {"type": "string"},
            "reference": {"type": "string", "format": "uri"}
        }
    },
    "RequestMethod": {
        "type": "object",
        "required": [ "type", "linkage" ],
        "allOf": [
            { "$ref": "#/definitions/OnlineResouceProperties" },
            {
                "properties": {
                    "type": { "enum": ["RequestMethod"] },
                    "constraint": { "$ref": "#/definitions/Domain"}
                }
            }
        ]
    },
    "Operation":
    {
        "type": "object",
        "required": [ "type", "DCP" ],
        "properties": {
            "type": { "enum": ["Operation"] },
            "name": { "type": "string" },
            "parameter": {
                "type": "array",
                "items": {"$ref": "#/definitions/Domain"}
            },
            "constraint": {
                "type": "array",
                "items": {"$ref": "#/definitions/Domain"}
            },
            "DCP": {
                "type": "array",
                "items": {
```

```
                    "type": "object",
                    "required": [ "type", "HTTP"],
                    "properties": {
                        "type": { "enum": ["DCP"] },
                        "HTTP": {"$ref": "#/definitions/HTTP"}
                    }
                }
            },
            "metadata": { "$ref": "#/definitions/Metadata"}
        }
    },
    "Metadata": {
        "type": "object",
        "required": [ "type" ],
        "properties": {
            "type": { "enum": ["Metadata"] },
            "metadata": { "type": ["string", "object"] },
            "link": { "type": "string", "format": "uri"},
            "about": { "type": "string", "format": "uri"}
        }
    },
    "BoundingBox": {
        "type": "object",
        "required": [ "type", "lowerCorner", "upperCorner"],
        "properties": {
            "type": { "enum": ["Metadata"] },
            "lowerCorner": {
                "type": "array",
                "items": {"type": "number"}
            },
            "upperCorner": {
                "type": "array",
                "items": {"type": "number"}
            },
            "crs": { "type": "string", "format": "uri" },
            "dimensions": { "type": "number" }
        }
    },
    "WGS84BoundingBox": {
        "type": "object",
        "required": [ "type", "lowerCorner", "upperCorner"],
        "properties": {
            "type": { "enum": ["Metadata"] },
            "lowerCorner": {
                "type": "array",
                "items": {"type": "number"}
            },
            "upperCorner": {
                "type": "array",
                "items": {"type": "number"}
            },
```

```
            "crs": { "type": "string", "format": "uri" },
            "dimensions": { "type": "number" }
        }
    },
    "ResponsibleParty": {
        "type": "object",
        "required": [ "type" ],
        "properties": {
            "type": { "enum": ["ResponsibleParty"] },
            "individualName": {"type": "string"},
            "positionName": {"type": "string"},
            "role": { "$ref": "#/definitions/Code"},
            "contactInfo": { "$ref": "#/definitions/Contact"}
        }
    },
    "Contact": {
        "type": "object",
        "required": [ "type" ],
        "properties": {
            "type": { "enum": ["Contact"] },
            "hoursOfService": {"type": "string"},
            "contactInstructions": {"type": "string"},
            "onlineResource": { "$ref": "#/definitions/OnlineResouce"},
            "address": { "$ref": "#/definitions/Address"},
            "phone": { "$ref": "#/definitions/Phone"}
        }
    },
    "Address": {
        "type": "object",
        "required": [ "type" ],
        "properties": {
            "type": { "enum": ["Adress"] },
            "deliveryPoint": {
                "type": "array",
                "items": {"type": "string"}
            },
            "city": {"type": "string"},
            "administrativeArea": {"type": "string"},
            "postalCode": {
                "type": "array",
                "items": {"type": "string"}
            },
            "country": {
                "type": "array",
                "items": {"type": "string"}
            },
            "electronicMailAddress": {
                "type": "array",
                "items": {"type": "string"}
            }
        }
```

```
        },
        "Phone": {
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["Phone"] },
                "voice": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "facsimile": {
                    "type": "array",
                    "items": {"type": "string"}
                }
            }
        },
        "HTTP": {
            "type": "object",
            "required": [ "type" ],
            "properties": {
                "type": { "enum": ["HTTP"] },
                "get": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/RequestMethod"}
                },
                "post": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/RequestMethod"}
                }
            }
        },
        "DatasetSummaryProperties": {
            "title": "Dataset summary",
            "description": "Metadata describing a dataset available from this
server.",
            "type": "object",
            "allOf": [
                { "$ref": "#/definitions/DescriptionProperties" },
                {
                    "properties": {
                        "identifier": { "$ref": "#/definitions/Code" },
                        "metadata": { "$ref": "#/definitions/Metadata" },
                        "boundingBox":  {
                            "type": "array",
                            "items": {"$ref": "#/definitions/BoundingBox" }
                        },
                        "WGS84BoundingBox":  {
                            "type": "array",
                            "items": {"$ref": "#/definitions/WGS84BoundingBox" }
                        }
                    }
```

```
                }
            ]
        },
        "ContentsProperties": {
            "title": "contents",
            "description": "",
            "type": "object",
            "properties": {
                "otherSource": { "type": "string", "format": "uri"}
            }
        },
        "OWSContents": {
            "title": "Contents",
            "description": "Metadata about the data served by this server. The
contents and organization of this section are specific to each OWS type, as defined by
that Implementation Specification. Whenever applicable, this section shall contain a
set of dataset descriptions, which should each be based on the MD_DataIdentification
class specified in ISO 19115 and used in ISO 19119.",
            "type": "object",
            "allOf": [
                { "$ref": "#/definitions/DatasetSummaryProperties" },
                { "$ref": "#/definitions/ContentsProperties" }
            ]
        }
    }
}
```

Chapter 9. Example on applying JSON to WMS to the WMS 1.4 UML models.

This clause shows the result of applying the JSON and JSON-LD rules to the WMS 1.4 proposed standard based on OWS Common.

9.1. Service metadata document

The following JSON Schema can be used to validate a WMS ServiceMetadata instance.

It defines the "root" of a WMS ServiceMetadata as composed by serviceIdentifications, serviceProvider, operationsMetadata, contents, isoMetadata and languages. The complex types of serviceIdentifications, serviceProvider, operationsMetadata, isoMetadata and languages are imported (actually they are "referenced" with "$ref") from the OWS common schemas while the contents part is defined by the WMS schema.

Fragment of WMS JSON Schema defining the "root" element of the service metadata document

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "WMS Service Metadata root object",
    "description": "A service metadata document is the normal response to a client
from performing the GetCapabilities operation, and contains metadata appropriate to
the specific server for the specific OWS.",
    "type": "object",
    "required": [ "type", "version"],
    "properties": {
        "type": { "enum": [ "WMSServiceMetadata"] },
        "id":  { "type": "string" },
        "version": { "type": "string"},
        "updateSequence": { "type": "string"},
        "serviceIdentification": { "$ref":
"ServiceMetadata_schema.json/#/definitions/ServiceIdentification" },
        "serviceProvider": { "$ref":
"ServiceMetadata_schema.json/#/definitions/ServiceProvider"},
        "operationsMetadata": { "$ref":
"ServiceMetadata_schema.json/#/definitions/OperationsMetadata"},
        "contents": { "$ref": "#/definitions/Contents"},
        "isoMetadata": { "$ref":
"ServiceMetadata_schema.json/#/definitions/IsoMetadata"},
        "languages": { "$ref": "ServiceMetadata_schema.json/#/definitions/Languages"}
    }
}
```

The "contents" element is defined as generalization of the "Contents" OWS Common section by the use of "allOf". This way, it inherits the capability to reference to otherSources that comes from OWS Common while is able to define a specific "layer".

Fragment of WMS JSON Schema defining the "content section" element of the service metadata document

```
    "definitions": {
        "Contents": {
            "title": "Contents",
            "description": "Metadata about the data served by this server.",
            "type": "object",
            "allOf": [
                {
                    "properties": {
                        "layer": {"$ref": "#/definitions/Layer" }
                    }
                },
                { "$ref":
"ServiceMetadata_schema.json/#/definitions/ContentsProperties" }
            ]
        }
    }
```

The "layer" element is defined as generalization of the "DatasetSummary" OWS Common section by the use of "allOf". This way, it inherits the many properties that are of common use such as title, abstract, identifiers, etc while being able to add more properties that are specific from map layer such as the minScaleDenominator and minScaleDenominator. Please note that WMS Layer has more parameters that have not been defined here.

Fragment of WMS JSON Schema defining the "layer section" element of the service metadata document

```
        "Layer": {
            "title": "Layer",
            "description": "Metadata about a layer served by this server.",
            "type": "object",
            "required": ["type"],
            "allOf": [
                { "$ref":
"ServiceMetadata_schema.json/#/definitions/DatasetSummaryProperties" },
                {
                    "properties": {
                        "type": { "enum": ["Layer"] },
                        "minScaleDenominator": { "type": "number" },
                        "maxScaleDenominator": { "type": "number" },
                        "layer": { "$ref": "#/definitions/Layer" }
                    }
                }
            ]
        }
```

Now we present an example of JSON instance that has been validated with the previous JSON Schema for WMS (that imports the OWS Common Schema too).

Partial WMS ServiceMetadata JSON instance

```
{
    "type": "WMSServiceMetadata",
    "version": "1.4",
    "updateSequence": "a",
    "serviceIdentification": {
        "type": "ServiceIdentification",
        "serviceType": {
            "type": "Code",
            "code": "WMS"
        },
        "serviceTypeVersion": ["1.4"],
        "title": [{"type": "LanguageString", "value": "WMS service", "lang": "en-
en"}],
        "keywords": [{"type": "Keywords", "keyword": [{"type": "LanguageString",
"value": "service", "lang": "en-en"}]}]
    },
    "serviceProvider": {
        "type": "ServiceProvider",
        "providerName": "CREAF",
        "serviceContact": {
            "type": "ResponsibleParty",
            "individualName": "Joan Masó"
        }
    },
    "operationsMetadata": {
        "type": "OperationsMetadata",
        "operation": [
            {
                "type": "Operation",
                "name": "GetCapabilities",
                "DCP": [{
                    "type": "DCP",
                    "HTTP":
                    {
                        "type": "HTTP",
                        "get": [{"type": "RequestMethod", "linkage":
"http://www.myserver.org/myserver.cgi"}]
                    }
                }]
            },{
                "type": "Operation",
                "name": "GetMap",
                "DCP": [{
                    "type": "DCP",
                    "HTTP":
                    {
                        "type": "HTTP",
                        "get": [{"type": "RequestMethod", "linkage":
"http://www.myserver.org/myserver.cgi"}]
```

```
                    }
                }]
            },{
                "type": "Operation",
                "name": "GetFeatureInfo",
                "DCP": [{
                    "type": "DCP",
                    "HTTP":
                    {
                        "type": "HTTP",
                        "get": [{"type": "RequestMethod", "linkage":
"http://www.myserver.org/myserver.cgi"}]
                    }
                }]
            }
        ],
        "constraint": [
            {
                "type": "Domain",
                "name": "layerLimit",
                "possibleValues": {
                    "type": "PossibleValues",
                    "allowedValues": {
                        "type": "AllowedValues",
                        "value": ["10"]
                    }
                }
            },
            {
                "type": "Domain",
                "name": "maxWidth",
                "possibleValues": {
                    "type": "PossibleValues",
                    "allowedValues": {
                        "type": "AllowedValues",
                        "value": ["1024"]
                    }
                }
            },
            {
                "type": "Domain",
                "name": "maxWidth",
                "possibleValues": {
                    "type": "PossibleValues",
                    "allowedValues": {
                        "type": "AllowedValues",
                        "value": ["1024"]
                    }
                }
            }
        ]
```

```
    },
    "contents": {
        "layer": {
            "type": "Layer",
            "title": [{"type": "LanguageString", "value": "The first layer", "lang":
"en-en"}],
            "abstract": [{"type": "LanguageString", "value": "This is the first layer
in the service layer list. For example; maximum temperature in 2015", "lang": "en-
en"}],
            "keywords": [{"type": "Keywords", "keyword": [{"type": "LanguageString",
"value": "climate", "lang": "en-en"}]}],
            "minScaleDenominator": 10,
            "maxScaleDenominator": 10000
        }
    },
    "isoMetadata": {},
    "languages": {
        "type": "Languages",
        "language": ["en-en"]
    }
}
```

The work on the WMS JSON schema presented here demonstrates how the WMS Schema can import the OWS Common schema and instances can be validated using both. The work to define a full JSON Schema for WMS 1.4 is not completed and requires to complete the Layer definition. We hope that work can be continued by the WMS SWG.

Chapter 10. Data formats encoding

In Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams of this document, a set of rules to encode in JSON a UML model is provided. For the moment, this document has not discussed how to encode geometric properties. This clause discusses this aspect and considers some alternatives.

The first problem we face is the overlap with GeoJSON. Unfortunately several considerations prevent use of GeoJSON directly when converting UML class diagrams into JSON:

- GeoJSON imposes an artificial separation between a single geometric property and the rest of properties.
- GeoJSON proposes that geometries are encoded as multidimensional arrays of coordinates that can not be processed by JSON-LD engines.
- IETF GeoJSON limits the CRS to only CRS84.
- The number of extra dimensions is limited to 3 (height).
- The number of geometric types defined is limited to bbox, point, line and polygon and aggregations of them. IETF explicitly disallows extending these types creating an artificial limitation.

Example of a polygon encoded in GeoJSON. Note that coordinates is a array of an array of an array.

```
{
   "type": "Feature",
   "geometry": {
       "type": "Polygon",
       "coordinates": [
           [
               [100.0, 0.0],
               [101.0, 0.0],
               [101.0, 1.0],
               [100.0, 1.0],
               [100.0, 0.0]
           ]
       ]
   },
   "properties": {
       "prop0": "value0",
       "prop1": {
           "this": "that"
       }
   }
}
```

The first limitation alone is incompatible with the set of rules proposed in Rules for encoding JSON Schema and JSON-LD context documents from UML class diagrams and with the General Feature Model (GFM). What it is needed is a JSON encoding for geometrical properties that can easily be

combined with other kind of properties.

10.1. Requirements for encoding geometries in JSON

This are the requirements that a good JSON encoding has to implement:

- Easy to use in JavaScript
- Easy transformation from JSON-LD to RDF
- Support to multiple dimensions
- Support to a variety of CRSs
- Support to a variety of geometric types beyond simple features.

Unfortunately, the two first requirements seems to go in opposite directions. An easy to use JavaScript encoding should be based on arrays of coordinates that are numbers that can be easily manipulated and transformed. An easy transformation in JSON-LD suggests the use other alternatives (e.g. a string) to avoid the multidimensional array problem and to avoid that each coordinate ends up in a different triple.

10.2. Use WKT as an alternative for encoding coordinates

Several possibilities have been suggested in the past to support a textual encoding that is "JSON-LD friendly", being the use of WKT encodings the most accepted in the discussions. Unfortunately, WKT encoding deals with coordinates as strings of text, which is quite convenient for inserting them into SQL sentences (the original purpose of the encoding) or SPARQL, but it is not very friendly with the spirit of JSON and JavaScript applications that can operate easily with arrays of numbers but need to parse WKT strings with specific pieces of code. This difficulty in adopting WKT can be overcome by a open source .js library capable of converting WKT into multidimensional arrays when needed.

Example of a polygon encoded in WKT. Note that coordinates are part of a string.

```
"POLYGON ((100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0, 100.0 0.0))"
```

10.3. Extending JSON-LD to support direct conversion from coordinates arrays into WKT

This sub-clause explores the possibility to extend the JSON-LD standard to allow for a transformation between potentially large arrays into WKT string literals during the RDF transformation. More experimentation is required to demonstrate the feasibility of this possibility.

A property that contains a potentially long array of geospatial coordinates or values associated to positions (where the order is important) should be converted to RDF as a WKT literal. To indicate this an extension of JSON-LD shall be used defining the property as "@container": "asWKT:WKT_Type"

Example of a property that should be converted to WKT during the conversion to RDF.

```
{
    "@context":
    {
        "coordinates": {"@id":"cis:coordinates", "@container": "asWKT:Multipoint Z"}
    }
}
```

Example of a 3 dimensional coordinates array before the transformation:

```
{
    "displacement": {
        @context: {
            "coordinates": {"@id":"cis:coordinates", "@container": "asWKT:Multipoint
Z"}
        }
        "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]]
    }
}
```

Example of a 3 dimensional coordinates array after the transformation to RDF is applied:

```
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_D_45_2D">
<http://www.opengis.net/cis/1.1/coordinates> "Multipoint Z ((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))" .
```

10.4. Extending WKT

Unfortunately, the proposal described in the previous section is not perfect because WKT is much more limited than the GFM. An extension for the WKT is required. WKT is defined in the OGC 06-103r4 OpenGIS implementation Standard for Geographic Information — Simple feature access — Part 1: Common Architecture.

10.4.1. Extending WKT to support more than 4 dimensions

WKT does not support numbers of dimensions greater than 4 and assumes that the 3rd dimension is Z. It could be good that WKT includes a variant encoding to indicate the number of dimensions

Example of 5 dimensional multipoint:

```
"Multipoint 5 ((90 0 0 0 0), (85 0 0 0 0))"
```

10.4.2. Extending WKT to support multidimensional arrays of values

WKT only supports coordinates and it does not support describing lists of values in predefined positions (such as the ones that a grid coverage needs in the *rangeSet*). It could be good that WKT includes a variant encoding to indicate list of measurements. It could be also useful to support several measurements for a single point such as the ones that are required for a multiband coverage.

Example of a 3 dimensional coordinates array before the transformation:

```
{
    "dataBlock": {
        @context: {
            "values": {"@id":"cis:coordinates", "@container": "asWKT:Multimeasure 5"}
        }
        "id": "cis:CIS_RS_DB_05_2D",
        "values": [[1, 2, 3, 4, 5], [11, 22, 33, 44, 55]]
    }
}
```

Example of 5 measurements in two predefined places converted to RDF using and extension of WKT.

```
<http://www.opengis.net/cis/1.1/examples/CIS_RS_DB_05_2D>
<http://www.opengis.net/cis/1.1/values> "Multimeasure 5 ((1 2 3 4 5), (11 22 33 44
55))" .
```

10.5. Extending double JSON serialization

There is an alternative that is more close to the JSON spirit that was not discussed so far in previous testbeds that it is worth to consider: Encode the multidimensional arrays as the serialization of a multidimensional array in JSON. In practice, this means to encode coordinates as multidimensional arrays written as string (i.e. in between ""). We can call this practice *double JSON serialization*. This alternative solves the requirements in a simple way:

- the string of coordinates will be considered as a text block in JSON-LD conversions to RDF
- the string can be immediately converted to multidimensional array using the JSON functions for parsing and serializing JSON.

Example of a polygon geometry double serialized in JSON. Please note the use of "" in the coordinates array.

```
{
    "type": "Polygon",
    "crs": "http://www.opengis.net/def/crs/ogc/1.3/crs84",
    "dimensions": 2,
    "coordinates": "[[
               [100.0, 0.0],
               [101.0, 0.0],
               [101.0, 1.0],
               [100.0, 1.0],
               [100.0, 0.0]
           ]]"
}
```

Example of JavaScript code to get the coordinates of a polygon.

```
    var field=JSON.parse('{
                            "type": "Polygon",
                            "crs": "http://www.opengis.net/def/crs/ogc/1.3/crs84",
                            "dimensions": 4,
                            "coordinates": "[[
                                       [100.0, 0.0],
                                       [101.0, 0.0],
                                       [101.0, 1.0],
                                       [100.0, 1.0],
                                       [100.0, 0.0]
                                  ]]"
                            }');
    var coordinates=JSON.parse(field.coordinates);
```

The advantage of this approach is that the code that currently supports GeoJSON can be easily modified to support the suggested encoding.

A pending work item is to precisely define the JSON encoding for all the necessary geometries including the ones beyond what is defined in GeoJSON.

Chapter 11. Coverages encoded in JSON

A particular case of geometry is a coverage. Coverages are defined by OGC using the GMLCOV 1.0. This specification defines only a XML encoding based on GML and SWE common namespaces. In 2016 the GMLCOV 1.0 is being re-branded as Coverage Implementation Schema (CIS) 1.1 and, in the context of this ER, a full encoding for coverages has been developed. The encoding was developed before completing the general UML to JSON rules clause, and this means that it does not strictly follows it even if most of the encoding respect some of the same suggested patterns.

The CIS combines the old GMLCOV 1.0 approach with new encodings for coverages. The requirements for the JSON encoding included in CIS 1.1 are not directly enough to define the encoding. The implementor is expected to follow the JSON examples and the JSON schema document also provided with the CIS 1.1 standard candidate.

11.1. GMLCov 1.0 coverage style

The CIS 1.1 includes the GMLCOV 1.0 UML model where a coverage is defined as a Domain, a RangeSet and a RangeType. We have provides this also in the JSON encoding.

Example of the 3 main element of the GMLCOV 1.0 coverage "style" in JSON.

```
{
    "type": "CoverageByDomainAndRangeType",
    "id": "examples:CIS_10_2D",
    "domainSet":{},
    "rangeSet": {},
    "rangeType": {}
}
```

The coverage range can define a regular or an irregular grid for 1 or more dimensions.

Example of the 3 main element of the GMLCOV 1.0 coverage "style" in JSON.

```
{
    "domainSet":{
        "type": "DomainSetType",
        "id": "examples:CIS_DS_10_2D",
        "generalGrid":{
            "type": "GeneralGridCoverageType",
            "id": "examples:CIS_DS_GG_10_2D",
            "srsName": "http://www.opengis.net/def/crs/EPSG/0/4326",
            "axisLabels": ["Lat", "Long"],
            "axis": [{
                "type": "RegularAxisType",
                "id": "examples:CIS_DS_GG_LAT_10_2D",
                "axisLabel": "Lat",
                "lowerBound": -80,
                "upperBound": -70,
                "uomLabel": "deg",
```

```
                "resolution": 5
            },{
                "type": "RegularAxisType",
                "id": "examples:CIS_DS_GG_LONG_10_2D",
                "axisLabel": "Long",
                "lowerBound": 0,
                "upperBound": 10,
                "uomLabel": "deg",
                "resolution": 5
            }],
            "gridLimits": {
                "type": "GridLimitsType",
                "id": "examples:CIS_DS_GG_GL_10_2D",
                "srsName": "http://www.opengis.net/def/crs/OGC/0/Index2D",
                "axisLabels": ["i", "j"],
                "axis": [{
                    "type": "IndexAxisType",
                    "id": "examples:CIS_DS_GG_GL_I_10_2D",
                    "axisLabel": "i",
                    "lowerBound": 0,
                    "upperBound": 2
                },{
                    "type": "IndexAxisType",
                    "id": "examples:CIS_DS_GG_GL_J_10_2D",
                    "axisLabel": "j",
                    "lowerBound": 0,
                    "upperBound": 2
                }]
            }
        }
    },
    "rangeSet": {
        "type": "RangeSetType",
        "id": "examples:CIS_RS_10_2D",
        "dataBlock": {
            "id": "examples:CIS_RS_DB_10_2D",
            "type": "VDataBlockType",
            "values": [1,2,3,4,5,6,7,8,9]
        }
    },
    "rangeType": {
        "type": "DataRecordType",
        "id": "examples:CIS_RT_10_2D",
        "field":[{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F_10_2D",
            "definition": "ogcType:unsignedInt",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F_UOM_10_2D",
                "code": "10^0"
```

```
            }
        }]
    }
}
```

One important thing that we wanted to include is the capacity to define the rangeSet directly embedded in the JSON encoding as an array of numbers or, as an alternative, provide a link to it.

11.2. CIS 1.1 distorted grid example

The CIS 1.1 includes new types of grids that were not possible to describe in GMLCOV 1.0 such as a distorted grid.

Example of CIS 1.1 distorted grid example in JSON.

```
{
    "type": "CoverageByDomainAndRangeType",
    "id": "examples:CIS_45_2D",
    "domainSet":{
        "type": "DomainSetType",
        "id": "examples:CIS_DS_45_2D",
        "generalGrid":{
            "type": "GeneralGridCoverageType",
            "id": "examples:CIS_DS_GG_45_2D",
            "srsName": "http://www.opengis.net/def/crs/EPSG/0/4979",
            "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/Index1D",
                "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": {
        "type": "DataRecordType",
        "id": "examples:CIS_RT_45_2D",
        "field":[{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F_45_2D",
            "definition": "ogcType:unsignedInt",
            "name": "singleBand",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F_UOM_45_2D",
                "code": "W/cm2"
            }
        }]
    }
}
```

11.3. CIS 1.1 Time series

The CIS 1.1 includes new types of coverages that are not defined as grids but as interleaved position and value pairs, that in this case is actually used to represent a time series.

Example of CIS 1.1 distorted grid example in JSON.

```
{
    "type": "CoverageByPartitioningType",
    "id": "examples:CIS_61_1D",
    "partitionSet":{
        "type": "PartitionSetType",
        "id": "examples:CIS_PS_61_1D",
        "positionValuePair":[
            { "type": "PVPType", "id": "examples:CIS_PVP1_61_1D", "coordinate":
["2015-09-20"], "value": [0]},
            { "type": "PVPType", "id": "examples:CIS_PVP2_61_1D", "coordinate":
["2015-09-21"], "value": [1]},
            { "type": "PVPType", "id": "examples:CIS_PVP3_61_1D", "coordinate":
["2015-09-22"], "value": [2]},
            { "type": "PVPType", "id": "examples:CIS_PVP4_61_1D", "coordinate":
["2015-09-23"], "value": [3]}
        ]
    },
    "rangeType": {
        "type": "DataRecordType",
        "id": "examples:CIS_RT_61_1D",
        "field":[{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F_61_1D",
            "name": "singleBand",
            "definition": "ogcType:unsignedInt",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F_UOM__61_1D",
                "code": "10^0"
            }
        }]
    }
}
```

11.4. CIS 1.1 Point cloud

This is another example of the CIS 1.1 new types of coverages that are not defined as grids but as direct multipoints, that in this case is actually used to represent a point cloud.

Example of CIS 1.1 distorted grid example in JSON.

```
{
    "type": "CoverageByDomainAndRangeType",
    "id": "examples:CIS_90",
    "envelope": {
        "type": "EnvelopeByAxisType",
        "id": "examples:CIS_EV_90",
        "srsName": "http://www.opengis.net/def/crs/EPSG/0/4979",
```

```
        "axisLabels": ["x", "y", "h"],
        "axis": [{
            "type": "AxisExtentType",
            "id": "examples:CIS_EV_X_90",
            "axisLabel": "x",
            "lowerBound": 456377.5,
            "upperBound": 456377.7,
            "uomLabel": "m"
        },{
            "type": "AxisExtentType",
            "id": "examples:CIS_EV_Y_90",
            "axisLabel": "y",
            "lowerBound": 339866.8,
            "upperBound": 339867.2,
            "uomLabel": "m"
        },{
            "type": "AxisExtentType",
            "id": "examples:CIS_EV_H_90",
            "axisLabel": "h",
            "lowerBound": 52,
            "upperBound": 53,
            "uomLabel": "m"
        }]
    },
    "domainSet":{
        "type": "DomainSetType",
        "id": "examples:CIS_DS_90",
        "directMultiPoint":{
            "type": "DirectMultiPointType",
            "id": "examples:CIS_DS_DMP_90",
            "coordinates": [[456377.56257493998, 339867.24995001999,
53.953899579999998],
                            [456377.50007493998, 339867.21865002002,
53.960899580000003],
                            [456377.56257493998, 339867.21865002002,
53.958999579999997],
                            [456377.50007493998, 339867.21865002002,
53.960899580000003]]
        }
    },
    "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]
        }
    },
    "rangeType": {
        "type": "DataRecordType",
```

```
        "id": "examples:CIS_RT_90",
        "field": [{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F1_90",
            "name": "red",
            "definition": "ogcType:unsigned",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F1_UOM_90",
                "code": "10^0"
            },
            "constraint": {
                "type": "AllowedValues",
                "id": "examples:CIS_RT_C1_UOM_90",
                "interval": [0, 255]
            }
        },{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F2_90",
            "name": "green",
            "definition": "ogcType:unsigned",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F2_UOM_90",
                "code": "10^0"
            },
            "constraint": {
                "type": "AllowedValues",
                "id": "examples:CIS_RT_C2_UOM_90",
                "interval": [0, 255]
            }
        },{
            "type": "QuantityType",
            "id": "examples:CIS_RT_F3_90",
            "name": "blue",
            "definition": "ogcType:unsigned",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F3_UOM_90",
                "code": "10^0"
            },
            "constraint": {
                "type": "AllowedValues",
                "id": "examples:CIS_RT_C3_UOM_90",
                "interval": [0, 255]
            }
        }]
    }
}
```

11.5. CIS 1.1 JSON Schema

This is the CIS 1.2 JSON Schema that can be used to validate the previous examples.

JSON schema document coverage-schema.json

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Coverage object",
    "description": "The CIS 1.1 coverage structures.",
    "type": "object",
    "oneOf" : [{
        "required": [ "type", "domainSet", "rangeSet", "rangeType"],
        "properties": {
            "id": { "type": "string"},
            "type": { "enum": [ "CoverageByDomainAndRangeType" ] },
            "envelope": { "$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/partitionSet" },
            "rangeType": { "$ref": "#/definitions/rangeType" },
            "metadata": { "$ref": "#/definitions/metadata" }
        }
    }],
    "definitions": {
        "envelope": {
            "title": "envelope",
            "description": "The envelope around a coverage is defined by the lower and
upper bound of each axis, respectively. The purpose of the axisLabels attribute, which
lists the axis labels of all axisExtent elements in proper sequence, is to enforce
axis sequence also in XML systems which do not preserve document order.",
            "type": "object",
            "required": [ "type", "srsName", "axisLabels", "axis"],
            "properties": {
                "type": { "enum": [ "EnvelopeByAxisType"] },
                "id":  { "type": "string" },
                "srsName": { "type": "string", "format": "uri"},
                "axisLabels":
                {
                    "type": "array",
                    "items": { "type": "string" }
                },
```

```
                "axis": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "required": [ "type", "lowerBound", "upperBound", "uomLabel"],
                        "additionalProperties": false,
                        "properties": {
                            "type": { "enum": [ "AxisExtentType"] },
                            "id":  { "type": "string" },
                            "axisLabel":  { "type": "string" },
                            "lowerBound":  { "type": ["number", "string", "null",
"boolean"] },
                            "upperBound": { "type": ["number", "string", "null",
"boolean"] },
                            "uomLabel": { "type": "string" }
                        }
                    }
                }
            }
        },
        "domainSet": {
            "title": "domainSet",
            "description": "The domainSet describes the *direct positions* of the
coverage, i.e., the locations for which values are available.",
            "type": "object",
            "oneOf" : [
            {
                "required": [ "type", "generalGrid"],
                "properties": {
                    "type": { "enum": [ "DomainSetType"] },
                    "generalGrid":{
                        "title": "General Grid",
                        "description": "A general n-D grid is defined through a
sequence of axes, each of which can be of a particular axis type.",
                        "type": "object",
                        "required": [ "type"],
                        "additionalProperties": false,
                        "properties": {
                            "type": { "enum": [ "GeneralGridCoverageType"] },
                            "id":  { "type": "string" },
                            "srsName": { "type": "string", "format": "uri"},
                            "axisLabels":
                            {
                                "type": "array",
                                "items": { "type": "string" }
                            },
                            "axis": {
                                "type": "array",
                                "items": {
                                    "type": "object",
                                    "oneOf" : [
```

```
                                        {
                                            "title": "Index Axis",
                                            "description": "An Index Axis is an axis
with only integer positions allowed.",
                                            "required": [ "type", "axisLabel",
"lowerBound", "upperBound"],
                                            "additionalProperties": false,
                                            "properties": {
                                                "type": { "enum": [ "IndexAxisType"]
},
                                                "id":  { "type": "string" },
                                                "axisLabel":  { "type": "string" },
                                                "lowerBound":  { "type": "number" },
                                                "upperBound": { "type": "number" }
                                            }
                                        },{
                                            "title": "Regular Axis",
                                            "description": "A Regular Axis is an axis
where all direct coordinates are at a common distance from its immediate neighbors.",
                                            "required": [ "type", "axisLabel",
"lowerBound", "upperBound", "resolution", "uomLabel"],
                                            "additionalProperties": false,
                                            "properties": {
                                                "type": { "enum": [ "RegularAxisType"]
},
                                                "id":  { "type": "string" },
                                                "axisLabel":  { "type": "string" },
                                                "lowerBound":  { "type": ["number",
"string", "null", "boolean"] },
                                                "upperBound": { "type": ["number",
"string", "null", "boolean"] },
                                                "uomLabel": { "type": "string" },
                                                "resolution": { "type": "number" }
                                            }
                                        },{
                                            "title": "Irregular Axis",
                                            "description": "An irregular axis
enumerates all possible direct position coordinates.",
                                            "required": [ "type", "axisLabel",
"uomLabel", "coordinate"],
                                            "additionalProperties": false,
                                            "properties": {
                                                "type": { "enum": [
"IrregularAxisType"] },
                                                "id":  { "type": "string" },
                                                "axisLabel":  { "type": "string" },
                                                "uomLabel": { "type": "string" },
                                                "coordinate": {
                                                    "type": "array",
                                                    "items": { "type": ["number",
"string", "boolean"] }
```

```
                                        }
                                    }
                                }
                            ]
                        }
                    },
                    "displacement": {
                        "title": "Displacement",
                        "description": "A Displacement is a warped axis nest
where points on the grid all have their individual direct position coordinates. The
sequenceRule element describes linearization order.",
                        "type": "object",
                        "oneOf": [
                        {
                            "required": ["type", "axisLabels", "uomLabels",
"coordinates" ],
                            "properties": {
                                "type": { "enum": [
"DisplacementAxisNestType"] },
                                "id":  { "type": "string" },
                                "axisLabel":  { "type": "string" },
                                "srsName": { "type": "string", "format":
"uri"},
                                "axisLabels":
                                {
                                    "type": "array",
                                    "items": { "type": "string" }
                                },
                                "uomLabels":
                                {
                                    "type": "array",
                                    "items": { "type": "string" }
                                },
                                "coordinates": {
                                    "type": "array",
                                    "items": {
                                        "type": "array",
                                        "items": { "type": ["number",
"string", "boolean"] }
                                    }
                                }
                            }
                        },{
                            "required": ["type", "axisLabels", "uomLabels",
"coordinatesRef"],
                            "properties": {
                                "type": { "enum": [
"DisplacementAxisNestTypeRef"] },
                                "id":  { "type": "string" },
                                "axisLabel":  { "type": "string" },
                                "srsName": { "type": "string", "format":
```

```
"uri"},
                                        "axisLabels":
                                        {
                                            "type": "array",
                                            "items": { "type": "string" }
                                        },
                                        "uomLabels":
                                        {
                                            "type": "array",
                                            "items": { "type": "string" }
                                        },
                                        "coordinatesRef": { "type": "string",
"format": "uri"}
                                    }
                                }]
                            },
                            "model": {
                                "title": "Sensor model",
                                "description": "A Transformation By Sensor Model is a
transformation definition which is given by a SensorML 2.0 transformation
specification.",
                                "type": "object",
                                "required": [ "type", "sensorModelRef" ],
                                "properties": {
                                    "type": { "enum": [
"TransformationBySensorModelType"] },
                                    "id":  { "type": "string" },
                                    "axisLabels":
                                    {
                                        "type": "array",
                                        "items": { "type": "string" }
                                    },
                                    "uomLabels":
                                    {
                                        "type": "array",
                                        "items": { "type": "string" }
                                    },
                                    "sensorModelRef": { "type": "string", "format":
"uri"},
                                    "sensorInstanceRef": { "type": "string", "format":
"uri"}
                                }
                            },
                            "gridLimits": {
                                "title": "Grid limits",
                                "description": "This is the boundary of the array
underlying the grid, given by its diagonal corner points in integer _60_3D. The grid
limits can be omitted in case all axes are of type index axis, because then it repeats
the grid information in a redundant way. The purpose of the axisLabels attribute,
which lists the axis labels of all axisExtent elements in proper sequence, is to
enforce axis sequence also in XML systems which do not preserve document order.",
```

```
                                "type": "object",
                                "required": [ "type"],
                                "properties": {
                                    "indexAxis": {
                                        "title": "Index Axis",
                                        "description": "An Index Axis is an axis with
only integer positions allowed.",
                                        "type": "object",
                                        "required": [ "type", "lowerBound",
"upperBound"],
                                        "additionalProperties": false,
                                        "properties": {
                                            "type": { "enum": [ "IndexAxisType"] },
                                            "id":  { "type": "string" },
                                            "axisLabel":  { "type": "string" },
                                            "lowerBound":  { "type": "number" },
                                            "upperBound": { "type": "number" }
                                        }
                                    },
                                    "srsName": { "type": "string", "format": "uri"},
                                    "axisLabels":
                                    {
                                        "type": "array",
                                        "items": { "type": "string" }
                                    }
                                }
                            }
                        }
                    }
                }
            },{
                "required": [ "type", "directMultiPoint"],
                "properties": {
                    "type": { "enum": [ "DomainSetType"] },
                    "directMultiPoint":{
                        "oneOf" : [{
                            "required": [ "type", "coordinates" ],
                            "properties": {
                                "type": { "enum": [ "DirectMultiPointType"] },
                                "coordinates": {
                                    "type": "array",
                                    "items": {
                                        "type": "array",
                                        "items": { "type": ["number", "string",
"boolean"] }
                                    }
                                }
                            }
                        },{
                            "required": [ "type", "coordinatesRef" ],
                            "properties": {
```

```
                                "type": { "enum": [ "DirectMultiPointTypeRef"] },
                                "coordinatesRef": { "type": "string", "format": "uri"}
                            }
                        }]
                    }
                }
            }]
        },
        "rangeSet": {
            "title": "rangeSet",
            "description": "The rangeSet lists a value for each of the coverage's
direct positions. Values resemble the *payload* information of some particular direct
positions. Values can be composite (with a single nesting level, i.e.: composites
always consist of atomics), or atomic (emulated through single-component composites)
whereby the sequence, structure, and meaning of every value is defined through the
rangeType. Values can be represented in-line or by reference to an external file which
may have any suitable encoding.",
            "type": "object",
            "oneOf" : [{
                "required": [ "type", "dataBlock"],
                "properties": {
                    "type": { "enum": [ "RangeSetType"] },
                    "dataBlock":{
                        "title": "dataBlock",
                        "description": "Data block objects",
                        "type": "object",
                        "required": [ "type", "values"],
                        "properties": {
                            "type": { "enum": [ "VDataBlockType", "CVDataBlockType"]
},
                            "values": {
                                "type": "array",
                                "items": { "type": ["number", "string", "null",
"boolean"] }
                            }
                        }
                    }
                }
            },
            {
                "required": [ "type", "fileReference"],
                "properties": {
                    "type": { "enum": [ "RangeSetRefType"] },
                    "fileReference": { "type": "string", "format": "uri"}
                }
            }]
        },
        "partitionSet": {
            "title": "Partitioning Set",
            "description": "A partition describes how a coverage (*sub-coverage*)
referenced is located within referencing coverage (*super-coverage*). The sub-coverage
```

```
can be represented by referencing a coverage id or a URL pointing to a coverage.  Such
sub-coverages referenced may be grouped into the super-coverage document, or reside
remote, or mixed. As an additional alternative, single range values can be indicated
verbatimg, together with their direct position. All values must share an identical
structure and conform to the rangeType definition.",
            "type": "object",
            "required": [ "type"],
            "properties": {
                "type": { "enum": [ "PartitionSetType"] },
                "partition":{
                    "type": "array",
                    "items": {
                        "type": "object",
                        "oneOf" : [{
                            "required": [ "type", "coverageRef"],
                            "properties": {
                                "id": { "type": "string"},
                                "type": { "enum": [ "PartitionRefType" ] },
                                "envelope": { "$ref": "#/definitions/envelope" },
                                "coverageRef": { "type": "string", "format": "uri"}
                            }
                        },{
                            "required": [ "type", "coverage"],
                            "properties": {
                                "id": { "type": "string"},
                                "type": { "enum": [ "PartitionType" ] },
                                "envelope": { "$ref": "#/definitions/envelope" },
                                "coverage": { "$ref": "#" }
                            }
                        }]
                    }
                },
                "positionValuePair": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "required": [ "type", "coordinate", "value" ],
                        "properties": {
                            "id": { "type": "string"},
                            "type": { "enum": [ "PVPType" ] },
                            "coordinate": {
                                "type": "array",
                                "items": { "type": ["number", "string", "boolean"] }
                            },
                            "value": {
                                "type": "array",
                                "items": { "type": ["number", "string", "null",
"boolean"] }
                            }
                        }
                    }
```

```
                }
            }
        },
        "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",
            "required": [ "type", "field"],
            "properties": {
                "type": { "enum": [ "DataRecordType"] },
                "field":{
                    "type": "array",
                    "items": {
                        "title": "quantity",
                        "description": "quantiy",
                        "type": "object",
                        "required": [ "type"],
                        "properties": {
                            "type": { "enum": [ "QuantityType"] },
                            "id": { "type": "string" , "format":"uri"},
                            "name": { "type": "string" },
                            "definition": { "type": "string", "format":"uri"},
                            "uom": {
                                "title": "units of measure",
                                "description": "units of measure",
                                "type": "object",
                                "required": [ "type", "code"],
                                "properties": {
                                    "type": { "enum": [ "UnitReference"] },
                                    "id": { "type": "string" , "format":"uri"},
                                    "code":{ "type": "string" }
                                }
                            },
                            "constraint": {
                                "title": "Constraint",
                                "description": "Constraint",
                                "type": "object",
                                "required": [ "type" ],
                                "properties": {
                                    "type": { "enum": [ "AllowedValues"] },
                                    "id": { "type": "string" , "format":"uri"},
                                    "interval":{
                                        "type": "array",
                                        "items": { "type": ["number", "string",
"boolean"] }
                                    }
                                }
                            }
                        }
```

```
                    }
                },
                "interpolationRestriction": {
                    "title": "interpolationRestriction",
                    "description": "Interpolation restriction",
                    "type": "object",
                    "required": [ "type"],
                    "properties": {
                        "type": { "enum": [ "InterpolationRestrictionType"] },
                        "id": { "type": "string" , "format":"uri"},
                        "allowedInterpolation": {
                            "type": "array",
                            "items": { "type": "string", "format": "uri"}
                        }
                    }
                }
            }
        },
        "metadata": {
            "title": "Metadata",
            "description": "The metadata element is a container of any (not further
specified) information which should be transported along with the coverage on hand,
such as domain-specific metadata.",
            "type": "object"
        }
    }
}
```

11.6. CIS 1.1 @context documents

The following @context documents can be used to transform the JSON instances into JSON-LD ones.

JSON-LD @context document coverage-context.json

```
{
    "@context": {
        "cis": "http://www.opengis.net/cis/1.1/",
        "id": "@id",
        "type": "@type",
        "CoverageByDomainAndRangeType": "cis:CoverageByDomainAndRangeType",
        "CoverageByPartitioningType": "cis:CoverageByPartitioningType",
        "envelope": "cis:envelope",
        "domainSet": "cis:domainSet",
        "rangeSet": "cis:rangeSet",
        "partitionSet": "cis:partitionSet",
        "rangeType": "cis:rangeType",
        "metadata": "cis:metadata"
    }
}
```

JSON-LD @context document envelope-context.json

```
{
    "@context":
    {
        "EnvelopeByAxisType": "cis:EnvelopeByAxisType",
        "srsName": {"@id":"cis:srsName", "@type": "@id"},
        "axisLabels": {"@id":"cis:axisLabels", "@container": "@list"},
        "axis": "cis:axis",
        "AxisExtendType": "cis:AxisExtendType",
        "axisLabel": "cis:axisLabel",
        "lowerBound": "cis:lowerBound",
        "upperBound": "cis:upperBound",
        "uomLabel": "cis:uomLabel"
    }
}
```

JSON-LD @context document domainset-context.json

```
{
    "@context":
    {
        "sml": "http://www.sensorml.com/sensorML-2.0/",
        "DomainSetType": "cis:DomainSetType",
        "GeneralGridCoverageType": "cis:GeneralGridCoverageType",
        "generalGrid": "cis:generalGrid",
        "srsName": {"@id":"cis:srsName", "@type": "@id"},
        "axisLabel": "cis:axisLabel",
        "axisLabels": {"@id":"cis:axisLabels", "@container": "@list"},
        "uomLabel": "cis:uomLabel",
        "uomLabels": {"@id":"cis:uomLabels", "@container": "@list"},
        "axis": "cis:axis",
        "IndexAxisType": "cis:IndexAxisType",
        "RegularAxisType": "cis:RegularAxisType",
        "IrregularAxisType": "cis:IrregularAxisType",
        "DisplacementAxisNestType": "cis:DisplacementAxisNestType",
        "lowerBound": "cis:lowerBound",
        "upperBound": "cis:upperBound",
        "resolution": "cis:resolution",
        "GridLimitsType": "cis:GridLimitsType",
        "gridLimits": "cis:gridLimits",
        "coordinate": {"@id":"cis:coordinate", "@container": "@list"},
        "displacement": "cis:displacement",
        "model": "cis:model",
        "TransformationBySensorModelType": "cis:TransformationBySensorModelType",
        "sensorModelRef": "sml:sensorModelRef",
        "sensorInstanceRef": "sml:sensorInstanceRef"
    }
}
```

JSON-LD @context document partitionset-context.json

```
{
    "@context":
    {
        "PartitionSetType": "cis:PartitionSetType",
        "partition": "cis:partition",
        "coverage": "cis:coverage",
        "coverageRef": {"@id":"cis:coverageRef", "@type": "@id"},
        "positionValuePair": "cis:positionValuePair",
        "PVPType": "cis:PVPType",
        "coordinate": "cis:coordinate",
        "value": "cis:value"
    }
}
```

JSON-LD @context document rangeset-context.json

```
{
     "@context": {
        "dataBlock": "cis:dataBlock",
        "VDataBlockType": "cis:VDataBlockType",
        "fileReference": {"@id":"cis:fileReference", "@type": "@id"}
    }
}
```

JSON-LD @context document rangetype-context.json

```
{
     "@context": {
        "swe": "http://www.opengis.net/swe/2.0/",
        "sml": "http://www.sensorml.com/sensorML-2.0/",
        "ogcType": "http://www.opengis.net/def/dataType/OGC/0/",
        "ogcInterpolation": "http://www.opengis.net/def/interpolation/OGC/0/",

        "DataRecordType": "swe:DataRecordType",
        "field": "swe:field",
        "names": {"@id":"swe:names", "@container": "@list"},
        "name": "swe:name",
        "QuantityType": "swe:QuantityType",
        "uom": "swe:uom",
        "code": "swe:code",
        "UnitReference": "swe:UnitReference",
        "definition": {"@id":"swe:definition", "@type": "@id"},
        "interpolationRestriction": "cis:interpolationRestriction",
        "allowedInterpolation": {"@id":"cis:allowedInterpolation", "@type": "@id"},

        "constraint": "swe:constraint",
        "AllowedValues": "swe:AllowedValues",
        "interval": "swe:interval"
    }
}
```

11.7. CIS 1.1 transformation into RDF

The following CIS 1.1 JSON-LD example was transformed to RDF using The JSON-LD playground that can be found here: http://json-ld.org/playground/

Examples of a CIS 1.1 examples: 10_2D_regular.json-ld

```
{
    "@context": ["http://localhost/json-ld/coverage-context.json", {"examples":
"http://www.opengis.net/cis/1.1/examples/"}],
    "type": "CoverageByDomainAndRangeType",
    "id": "examples:CIS_10_2D",
    "domainSet":{
        "@context": "http://localhost/json-ld/domainset-context.json",
        "type": "DomainSetType",
        "id": "examples:CIS_DS_10_2D",
        "generalGrid":{
            "type": "GeneralGridCoverageType",
            "id": "examples:CIS_DS_GG_10_2D",
            "srsName": "http://www.opengis.net/def/crs/EPSG/0/4326",
            "axisLabels": ["Lat", "Long"],
            "axis": [{
                "type": "RegularAxisType",
```

```
                "id": "examples:CIS_DS_GG_LAT_10_2D",
                "axisLabel": "Lat",
                "lowerBound": -80,
                "upperBound": -70,
                "uomLabel": "deg",
                "resolution": 5
            },{
                "type": "RegularAxisType",
                "id": "examples:CIS_DS_GG_LONG_10_2D",
                "axisLabel": "Long",
                "lowerBound": 0,
                "upperBound": 10,
                "uomLabel": "deg",
                "resolution": 5
            }],
            "gridLimits": {
                "type": "GridLimitsType",
                "id": "examples:CIS_DS_GG_GL_10_2D",
                "srsName": "http://www.opengis.net/def/crs/OGC/0/Index2D",
                "axisLabels": ["i", "j"],
                "axis": [{
                    "type": "IndexAxisType",
                    "id": "examples:CIS_DS_GG_GL_I_10_2D",
                    "axisLabel": "i",
                    "lowerBound": 0,
                    "upperBound": 2
                },{
                    "type": "IndexAxisType",
                    "id": "examples:CIS_DS_GG_GL_J_10_2D",
                    "axisLabel": "j",
                    "lowerBound": 0,
                    "upperBound": 2
                }]
            }
        }
    },
    "rangeSet": {
        "@context": "http://localhost/json-ld/rangeset-context.json",
        "type": "RangeSetType",
        "id": "examples:CIS_RS_10_2D",
        "dataBlock": {
            "id": "examples:CIS_RS_DB_10_2D",
            "type": "VDataBlockType",
            "values": [1,2,3,4,5,6,7,8,9]
        }
    },
    "rangeType": {
        "@context": "http://localhost/json-ld/rangetype-context.json",
        "type": "DataRecordType",
        "id": "examples:CIS_RT_10_2D",
        "field":[{
```

```
            "type": "QuantityType",
            "id": "examples:CIS_RT_F_10_2D",
            "definition": "ogcType:unsignedInt",
            "uom": {
                "type": "UnitReference",
                "id": "examples:CIS_RT_F_UOM_10_2D",
                "code": "10^0"
            }
        }]
    }
}
```

Transformation to RDF result.

Examples of a CIS 1.1 examples: 10_2D_regular.n3

```
<http://www.opengis.net/cis/1.1/examples/CIS_10_2D>
<http://www.opengis.net/cis/1.1/domainSet>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_10_2D>
<http://www.opengis.net/cis/1.1/rangeSet>
<http://www.opengis.net/cis/1.1/examples/CIS_RS_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_10_2D>
<http://www.opengis.net/cis/1.1/rangeType>
<http://www.opengis.net/cis/1.1/examples/CIS_RT_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_10_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_DS_10_2D>
<http://www.opengis.net/cis/1.1/generalGrid>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_10_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_10_2D>
<http://www.opengis.net/cis/1.1/axis>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_2D>
<http://www.opengis.net/cis/1.1/axis>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_2D>
<http://www.opengis.net/cis/1.1/axisLabels> _:b0 .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_2D>
<http://www.opengis.net/cis/1.1/gridLimits>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_2D>
<http://www.opengis.net/cis/1.1/srsName> <http://www.opengis.net/def/crs/EPSG/0/4326>
.
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_10_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_GL_10_2D>
<http://www.opengis.net/cis/1.1/axis>
```

```
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_I_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_10_2D>
<http://www.opengis.net/cis/1.1/axis>
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_J_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_10_2D>
<http://www.opengis.net/cis/1.1/axisLabels> _:b2 .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_10_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_GL_10_2D>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/cis/1.1/GridLimitsType> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_I_10_2D>
<http://www.opengis.net/cis/1.1/axisLabel> "i" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_I_10_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_GL_I_10_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_GL_I_10_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_GL_J_10_2D>
<http://www.opengis.net/cis/1.1/axisLabel> "j" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_GL_J_10_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_GL_J_10_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_GL_J_10_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_LAT_10_2D>
<http://www.opengis.net/cis/1.1/axisLabel> "Lat" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D>
<http://www.opengis.net/cis/1.1/lowerBound> "-
80"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D>
<http://www.opengis.net/cis/1.1/resolution>
"5"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D>
<http://www.opengis.net/cis/1.1/uomLabel> "deg" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D>
<http://www.opengis.net/cis/1.1/upperBound> "-
70"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LAT_10_2D>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/cis/1.1/RegularAxisType> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_2D>
```

```
<http://www.opengis.net/cis/1.1/axisLabel> "Long" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_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_LONG_10_2D>
<http://www.opengis.net/cis/1.1/resolution>
"5"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_2D>
<http://www.opengis.net/cis/1.1/uomLabel> "deg" .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_2D>
<http://www.opengis.net/cis/1.1/upperBound>
"10"^^<http://www.w3.org/2001/XMLSchema#integer> .
<http://www.opengis.net/cis/1.1/examples/CIS_DS_GG_LONG_10_2D>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/cis/1.1/RegularAxisType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RS_10_2D>
<http://www.opengis.net/cis/1.1/dataBlock>
<http://www.opengis.net/cis/1.1/examples/CIS_RS_DB_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RS_10_2D> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://www.opengis.net/cis/1.1/RangeSetType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RS_DB_10_2D>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/cis/1.1/VDataBlockType> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_10_2D>
<http://www.opengis.net/swe/2.0/field>
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_10_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_10_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_10_2D>
<http://www.opengis.net/swe/2.0/uom>
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_UOM_10_2D> .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_10_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_10_2D>
<http://www.opengis.net/swe/2.0/code> "10^0" .
<http://www.opengis.net/cis/1.1/examples/CIS_RT_F_UOM_10_2D>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.opengis.net/swe/2.0/UnitReference> .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "Lat" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:b1 .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "Long" .
_:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#nil> .
_:b2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "i" .
_:b2 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> _:b3 .
_:b3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> "j" .
_:b3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-
```

```
rdf-syntax-ns#nil> .
```

Appendix A: Revision History

Table 2. Revision History

Date	Release	Editor	Primary clauses modified	Descriptions
Sep 20, 2016	Joan Maso	.1	all	first full version
Nov 15, 2016	Joan Maso	1.0	all	comments integrated; text finalized and ready for TC review.

Appendix B: Bibliography

OGC 14-009r1 Testbed-10 Rules for JSON and GeoJSON Adoption: Focus on OWS-Context.

OGC 15-053 Implementing JSON/GeoJSON in an OGC Standard ER