Lorenzo Bigagli, CNR 17 May 2016 comments John Herring, 17 May 2016

Notes on The Specification Model – A Standard for Modular Specification (08-131r3)

Introduction

A netCDF files is self-describing. In particular, it contains the definition of its own content, in terms of multi-dimensional arrays (termed "variables") indexed by limited and unlimited dimensions (also declared in the file).

In the basic flavor of the format, a variable defined on more than one unlimited dimension is illegal. Hence, the OGC Network Common Data Form (NetCDF) Core Encoding Standard version 1.0 (OGC 10-090r3) includes:

> ***Requirement 7***
> ***http://www.opengis.net/spec/netcdf/1.0/req/core/recorddimension:***
> *At most one dimension (the record dimension) shall have unlimited length*

And the related test case (A.1.7):

> *Open the dataset and check that at most one dimension (the record dimension) has unlimited length.*

This requirement is relaxed by some flavors of the format (in fact, there may be multiple unlimited dimensions in a dataset; it suffices that every variable is defined on at most one).

However, as the above test case is in the core class, the current semantics of requirements extension implies that it *must* be observed by all extensions. In other words, the above requirement implies that some flavors of netCDF cannot be standardized (in OGC).

Similar issues arose in the context of WCS 2.0 specification.[1]

To mitigate this problem, 08-131r3 includes Req 25, which forbids requirements like the one above:

Req 25 - A specification conformant to this standard shall never restrict in any manner future, logically-valid extensions of its standardization targets.

This is not a violation of *Req 25*. One would assume that the limit in *Requirement 7* above is a semantic issue on the meaning of the "dimension" in the base standard causes this limit. This is a limitation on implementations (the datasets referenced in A.1.7) not on extensions. Extensions must maintain the semantics of the Root Standard, and apparently *Requirement 7* is part of those semantics.

Unfortunately, restrictive requirements are very common, e.g. in:

[1] See for example: P. Baumann, "Core & Extension Model: Lessons Learned While Drafting WCS 2.0", 72st OGC Technical Committee Meeting, Frascati, Italy, 10 March 2010. See also L. Bigagli, "Core and Extensions PubSub SWG lessons learned", 96th OGC Technical Committee Meeting, Nottingham, UK, 14-17 September 2015.

- Application schemas
  - *"Contents restriction is expected to be frequently used to restrict contents, in order to increase interoperability and reduce ambiguity when greater flexibility is not needed for applications"* (OWS Common, §11.6.6)
- *NOTE: in* GeoJSON, *a type is distinguished by the value of the "type" key and there 9 valid ones (FeatureCollection, Feature, Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection). This is perfectly in line with OWS Common, although prohibited by Req 25.*

GeoJSON is in violation of *Req 25*, because it limits known valid extensions of both geometry and of coordinate reference systems. OWS Common is not, especially since there is a "get out of jail free card" in the phrase "*when greater flexibility is not needed for applications". Once the applications reach the point that the "flexibility is needed for applications" the "excess restrictions" can be dropped.*

- Profiles, enumerations, anything with an upper bound, etc.
- UML specialization, cardinality, etc.

Noticeably, almost all requirements of 08-131r3 are restrictions. Req 25 itself is a restriction: it forbids restrictions.

All valid requirements are restriction of some sort, albeit useful and necessary, but they against the <u>implementation</u>, not the <u>extensions</u>. Restrictions and requirements on implementations are what standards are all about. But once the "core" implemetnaiton is dealt with, there is no reason to limit its implementations from going further as long as the stated implementation restrictions are not violated. If the restrictions are not logically needed, then the "get out of jail free card" like OWS common provides side steps the "simple" implementations for the base standard.

Given that 08-131r3 is *"a standard for writing OGC standards"* (ib., [frontispiece]) and that:

- 08-131r3 is *"considered one of its own standardization targets and thus a subject of its own requirements"* (§6.1);

Yes, the ModSpec is a standard on standards, i.e. a meta-standard. Any of its extensions would therefore also be meta-standards.

- Standardization targets for 08131r3 = OGC standards;
- Any extension of a target type is an instance of the core target; It follows that, if 08-131r3 is coherent, Req 25 can be restated as:

Req 25 – 08-131r3 shall never restrict in any manner future, logically-valid OGC standards.

No. Applied to 08-131r3, the requirement would read (with some parentheticals):

> ***Req 25 (applied to the ModSpec) – 08-131r3 (an OGC meta-standard conformant to itself) shall never restrict in any manner future, logically-valid extensions of OGC standards.***

This means that 08-131r3 shall enable all logically valid extensions of OGC standards, which is does because it contains *Req 25*.

This is clearly inconsistent.

Actually it is recursively consistent. It prevents its compliant standards to never restrict their extension or their implementations in a logically invalid manner.

Suggested amendments to 08-131r3

In general, the rationale for Req 25 involves informal and subjective notions (e.g., wisdom, goodness) and the intended scope is very general (any future, logically-valid extension, above and beyond the current design requirements).

No, the statement is quite formal and precise without subjective content. The only "rationale" given for it is a pair of examples. Any extension would be future since the standard must exist before it is extended. "Logically valid" means that the extension is consistent with its base standard and its implementations, i.e. the standard that already exists. The requirement is in fact protection for valid extensions. GeoJSON used as a "avatar" above is really an example of the havoc that violation of Req 25 can engender.

The crux of the issue seems related to the semantics of requirements class extension, in relation to the semantics of target type extension. The former is a dependency relationship (extensions can only add requirements), whereas standardization target types *"inherit from one another in the same way that UML classes do"* (§4.24).

Inheritance in UML classes creates the strong substitutability of instances. Adding requirements preserves the requirements of the core, which means implementations of extensions must also satisfy any of its ancestors. The analogy is perfectly consistent.

Possible amendments:

- Requirements class extension should be clarified
  - see generalization/specialization in UML;
  - see the extension/restriction mechanism of the XML Schema Language;
- Restrictive constraints should be allowed and inheritable just like any other
  - The OOPL notion of "final" (not-extensible) classes may be introduced;

In this case, it is hard to see that any specification could be in such a state of no logically extension possible.

- Req 25 should be relaxed/clarified: may be made into a recommendation, or the second part of the section may be rephrased to allow negative requirements and exclusions. Another option is to restrict the scope of the extension to well identified "extension points" (see below).

The only example of a standard that OGC would never be interested in adopting as a core standard is the one example of a violation of Req 25, i.e. GeoJSON.

More in details, the following page edits are proposed.

P. 7, §4.18 – the definition of “Requirements” should explicitly include the notion of negative requirements (i.e. constraints, closure statements, restrictions, restrictive requirements, etc.) This is in line with ISO 19105 – Conformance and Testing:

- *“[...] conformance requirements may be stated*
  - *A)* ***positively****: they state what is required to be done;*
  - *B)* ***negatively****: they state what is required not to be done” (ib., §6.3)*
- *“A conforming implementation may support additional capabilities not described in the standard, providing those capabilities are not explicitly prohibited in the standard” (ib., §6.5).*

P. 20, §6.5.4 – the final NOTE exemplifying “GML for Simple Features” as a possible candidate GML core is confusing, since any “Simple Feature” profile is likely to restrict the allowed property types. E.g., 06-049r1 currently states: *“Spatial properties are limited to being of type: point, linearly interpolated curve, planar surface, or aggregates thereof”* (ib., §2.1). The example should be removed or clarified.

P. 20, §6.5.5 – the wording of the second part of the section (trying to formalize how a standard is supposed to feature evolvable requirements) forbids restrictions and exclusions (e.g.: *“a specification may require a standardization target to accept GML as a feature specification language, but cannot require a standardization target to not accept an alternative, such as KML, or GeoJSON”*). It should be expanded or clarified, as elaborated above.

Other editorials and typos

P. 10, item 10 – punctuation;

P. 11, paragraph 4, row 3 – punctuation;

P. 12, note at the top – the reference to §4.11 is probably misplaced (should be after “B extends A”), since §4.11 defines “extension” and not “profile”; the sentence “A is a profile of B” seems not coherent with the definition of “profile” in §4.16; P. 14, §6.2 – the least 2 rows are repeated two paragraphs above;

P. 20, §6.5.5 – punctuation of the last paragraph, row 3 (‘;’).

Extended solution

We may allow the definition of “extension points” (e.g. functions returning appropriate assessment values to be used in test cases) and change the semantics of requirement class extension allowing a sub-class to override the parents’ requirements, limitedly to their defined extension points.

The above test case on NetCDF Core may be rephrased like:

○ Verify that every variable is defined on at most <getUnlimitedDims()> unlimited dimension. Fail otherwise.

Extension points:

getUnlimitedDims() = 1

Sub-classes may redefine getUnlimitedDims() as suitable.

This mechanism for test case overriding could be easily implemented in any Object-Oriented language, or in CTL, extending its grammar and resorting to XSLT transformations (AspectOriented extensions may be investigated).

The same mechanisms could be extended to allow for the definition of abstract extension points and support the generalized dynamic composition of ETS's, enabling the assessment of transversal capabilities in arbitrary combinations of conformance classes.