Towards a Model-Driven Datacube Analytics Language

Peter Baumann
Computer Science & Electrical Engineering
Jacobs University
Bremen, Germany
p.baumann@jacobs-university.de

***Abstract*—Datacubes form an accepted cornerstone for analysis (and visualization) ready spatio-temporal data offerings. Geo datacubes have been standardized since long under the umbrella concept of coverages, and such data structures are well understood in concept and practice. This, however, is not paired by a similar understanding of coverage analytics.**

We present a formal model for datacube analytics which is based on Linear Algebra, incorporates space and time semantics, and allows a wide range of common datacube operations, up to, say, the Discrete Fourier Transform. For convenience, the formalism is based on a language allowing expressions of any complexity.

The specification is currently in the avanced adoption process of ISO for becoming the future 19123-3 standard.

Keywords—datacube, analytics, language, coverage, ISO

I. MOTIVATION

Datacubes introduce function-rich services on spatio-temporally aligned, homogenized raster data assets typically coming as sensor, image (timeseries), simulation, and statistics data. Actionable datacubes support analytics through the paradigm of "any query, any time, on any size" [6] originally coined by research on Array Databases [7]. Today datacubes are an accepted cornerstone for providing data ready for analysis, fusion, and visualization. In particular this is due to the homogenization of the zillions of scenes and other data into spatio-temporal units where each one has its common coordinate system, pixel type, etc. Thanks to the flexibility of the OGC / ISO / EU INSPIRE coverage standards [8] there is still sufficient degree of freedom to accommodate missing data areas in datacubes, regular as well as irregular grids, and a wide variety of data formats while retaining interoperability to allow, for example, simple fusion of datacubes with different dimensions and coordinate systems.

Given the ever-increasing importance of coverage services going far beyond mere subsetting there is a strong need to standardize not only data representation and exchange, but also the capabilities of services in an interoperable manner – not the least for the emerging vision of interoperable automated service mashups.

Datacube query languages establish actionable datacubes enabling users to ask "any query, any time" without programming, and independent from internal storage and external ingest/delivery data formats. Following this philosophy, 19123-3 is independent from any concrete coverage encoding format (such as GeoTIFF, NetCDF, etc.) and concrete processing language (such as python, R, SQL/MDA, etc.). Likewise, it is not a service, but rather provides the basis for service concretizations such as those existing in OGC:WCS GET/KVP, WCS POST/XML, WCS SOAP, OAPI-Coverages, and OAPI-Processing. In future, ISO TC211 possibly might establish concrete coverage services in a forthcoming 19123-4.

However, due to the highly dynamic nature of the field it is not easy to understand terms, trends, and technologies, and how they relate or differ. It helps to look at standards where, centered around the notion of a "coverage", the main geo standardization bodies of OGC, ISO, and INSPIRE have established a mature, agreed, modular data and service model allowing implementations to differ in the extent of support while still remaining interoperable. For example, in the Web Coverage Service (WCS) suite WCS-Core just offers subsetting and format encoding while on the high end Web Coverage Processing Service (WCPS) offers a datacube analytics language.

In this paper, we report about our work on establishing an ISO standard for the foundations of datacube processing, based on a language for extraction, filtering, processing, analytics, and fusion of multi-dimensional geospatial datacubes representing, for example, spatio-temporal sensor, image, simulation, or statistics datacubes. Expressions in this language accept any number of input datacubes (together with further common inputs like numbers) to generate any number of output datacubes or scalar results.

The data model used for geo datacubes is given by the ISO coverage model as defined in the twin draft standard 19123-1 [5]. A coverage describes mathematical fields through several – practically induced – techniques, specifically: regular and irregular grids, point clouds, and general meshes.

The language is functionally defined and free of any side effects. It has a formal semantics foundation and is minimal: only two constructs establish all coverage processing: A coverage constructor to build (or derive) a coverage and an aggregation operator (called condenser) deriving summary information. Further convenience functions are derived from those.

The language does not define a service API – it is independent from any particular request and response encoding, as no concrete request/response protocol is assumed. Hence, this standard rather acts as the foundation for defining service standards functionality. Currently such concrete service definitions exist with OGC Web Coverage Service (WCS) [4] via GET/KVP, POST/XML, SOAP protocol bindings, as well as with emerging specifications like OAPI-Coverages [13] and OAPI-Processes [12].

Supported by European Commission H2020 CENTURION.

The datacube analytics specification has been finalized and submitted for voting to the national delegations under its identifier 19123-3. This work is part of ISO plans on further populating the coverage ecosystem. Fig. 1 shows possible evolution paths and the position of 19123-3 in it.

	Geometry/data model	Function/processing model
abstract	**19123-1** Coverage Fundamentals = *OGC Abstract Topic 6*	**19123-3** Coverage Processing Fundamentals = *OGC WCPS*
concrete	**19123-2** Coverage Implementation Schema = *OGC CIS*	**19123-4** Coverage Services = *OGC WCS & OAPI-Coverages ?*

Fig. 1. Possible evolution of coverage standards in ISO

For the purpose of this paper we refer to the 19123-1 draft specification as *Coverage Fundamentals* (CF) and the 19123-3 draft text as *Coverage Processing Fundamentals* (CPF).

In its current version CPF supports grid coverages with index, regular, and irregular axes. In the future it is foreseen that the standard gets extended so as to address all CF types.

The remainder of this contribution is organised as follows. In the next section we provide an overview of the datacube part of the coverage model as a background. After that, the CPF processing language is introduced in Section 3. A comparison against the state of the art is provided in Section 4. Section 5 gives a summary and an outlook.

II. Coverages and Datacubes

For the reader's convenience this section gives a brief informal recap of the CF model, under adoption by ISO as 19123-1. A detailed description is provided in [5].

Following the mathematical notion of a function that maps elements of a domain (here: spatio-temporal coordinates) to a range (here: "pixel", "voxel", etc. values), a coverage consists of:

- an *identifier* which uniquely identifies a coverage in some context (here: the context of an expression);
- a *domain set* of coordinate points (expressed in a common Coordinate Reference System, CRS): "*where in the multi-dimensional space can I find values?*"
- a probing function which answers for each coverage coordinate in the domain set ("*direct position*"): "*what is the value here?*"
- a *range type*: "*what do those values mean?*"
- optional *metadata*: "*what else should I know about these data?*"

The coverage concept encompasses regular and irregular grids, point clouds, and meshes. In our context, we only consider multi-dimensional grid coverages which are used to represent datacubes. CF supports a very general grid concept where any kind of regular and irregular grids is supported; Fig. 2 shows some examples.

In an interface-oriented UML specification such a coverage, on abstract level can be described as in Fig. 3.

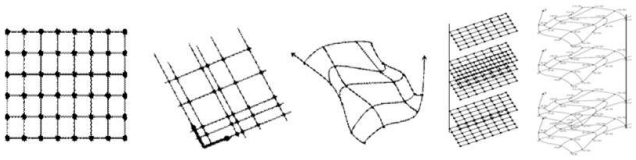

Fig. 2. Sample regular and irregular grid structures [5]

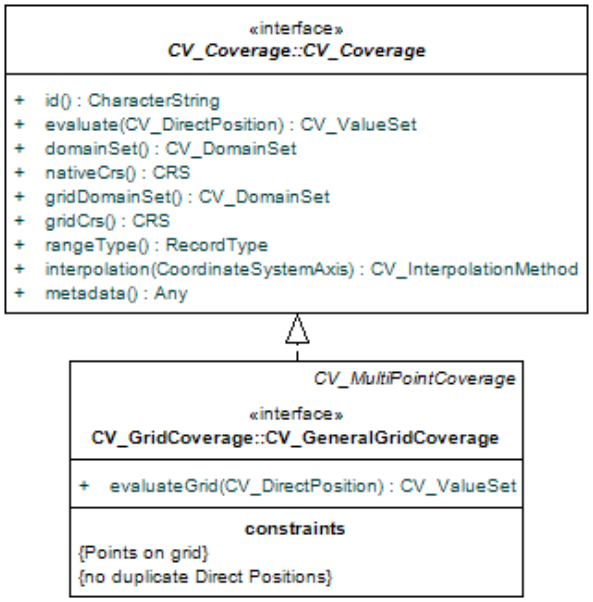

Fig. 3. Interface-oriented specification of coverages and grid coverages [10]

III. A Language for Coverage Analytics

Being on the same conceptual abstraction level as CF, the CPF likewise is not at implementation level (in particular: does not define a concrete service), but rather establishes coverage processing concepts that may serve to conceptualize concretizations, such as the OGC *Web Coverage Service* (WCS) API. Actually, core concepts draw on the OGC *Web Coverage Processing Service* (WCPS) language; however, CPF has been stripped from several OGC legacyand lifted to the abstract CF data model.

In CPF, the following CF axis types are supported:

- A **Cartesian** ("index") axis just requires lower and upper bound (which are of type integer).
- A **regular** axis which can be described by lower and upper bounds together with a constant distance, the resolution.
- An **irregular** axis which has individual distances, described by a sequence of coordinates.

The coverage domain set with its axes has a single native CRS which may allow georeferencing. Additionally, the underlying grid structure is defined through a Cartesian grid CRS. Both CRSs have the same dimension, i.e.: number of axes. In CPF, CRSs are addressed by name in expressions. Both CF and CPF do not make any assumptions about the nature of identifying CRSs, but rather treat them as opaque identifiers.

These concepts are formalized in CPF through so-called *probing functions* which extract information from an otherwise abstract, encapsulated object following the theory of

Abstract Data Types [1]. Table I summarizes representative probing functions.

A. Processing Expressions

The CPF coverage processing language defines expressions on coverages which evaluate to ordered lists of either coverages or scalars (whereby "scalar" here is used as a summary term of all data structures that are not coverages). In the sequel, the terms *processing expression* and *query* are used interchangeably.

A CPF coverage processing expression consists the CPF primitives plus the nesting capabilities, altogether forming an expression language which is independent from any particular encoding and service protocol. The semantics of a CPF expression is defined recursively by indicating, for all admissible expressions, the semantics which is given by the probing function output when applied to a coverage-valued expression.

The basic shape of a CPF query is given by

```
for v1 in ( L1 ),
    v2 in ( L2 ),
     … ,
    vn in ( Ln )
[ let c1 := e1, …, cm := em ]
[ where P ]
return E
```

The L_i are lists in the for clause are coverages to which the corresponding variables v_1 are bound in sequence. This establishes an iteration over these coverages. Having several variables establishes a nested loop where any number of covverages can be combined for fusion.

Optional **let** clause is just for convenience – it allows for convenient abbreviations of sub-expressions.

Table I. — SELECTION OF COVERAGE PROBING FUNCTIONS

Coverage characteristic	Probing function for some coverage *C* as per CF	Comment
Coverage identifier	*id(C*) as per CF	Identifier of the coverage
Coverage native CRS	*nativeCrs*(*C*) as per CF	Native CRS of the coverage
Coverage grid CRS	*gridCrs*(*C*) as per CF	Grid CRS of the coverage
CRS axis list	*axisList*(*crs*) = (a_1,…,a_d) for some *d*-D CRS *crs* establishing this axis sequence	List of all axes of the CRS, in proper sequence
Domain extent of coverage, in its native CRS	*domainSet*(*C*) as per CF	Extent of the coverage in native CRS coordinates
Domain extent of coverage, in its grid CRS	*gridDomainSet*(*C*) as per CF	Extent of the coverage in Cartesian grid coordinates, relative to the coverage's grid CRS
Range type	*rangeType*(*C*) as per CF	The range type record is described by a list describing its components in sequence; for the purpose of 190123-3 only component name and its data type are considered.
Range field name list	*rangeFieldNames*(*C*) = (f_1, …, f_n) with field names f_i	Ordered list all of the coverage's range fields names and their data types; possible further constituents in a record component are ignored in this standard, their values are to be defined elsewhere (e.g., implementation dependent)
Range values	*value*(*C*,*p*) = $evaluate_C(p)$, $p \in domainSet(C)$ with *evaluate*() as per CF	Direct Positions can be expressed either in the native or the grid CRS. We use the standard array notation of *C*[*p*] alternatively.
Interpolation	*interpolation*(*C*) as per CF	List of the interpolation method allowed per axis, in axis order Interpolation associated with a particular axis
Metadata of coverage	*metadata*(*C*) as per CF	a string without further structure nor semantics, seen from a 19123-1 perspective

In the **return** clause, coverage expression *E* performs the analytics. These expressions may contain occurrences of the variables defined in the **for** and **let** clauses. If the result is scalar it will be returned as ASCII, coverage-valued results need to be encoded in some suitable data format.

The optional **where** clause performs filtering: only those coverages where the filter predicate *P* evaluates to *true* are forwarded for processing the **return** clause instantiation.

For example, assume availability of coverages *A*, *B*, and *C*. Then, the following CPF query

```
for $c in ( A, B, C )
return encode( $c, "image/tiff" )
```

will produce a result list containing three TIFF-encoded coverages. In the next example, assume availability of satellite images *A*, *B*, and *C* and a coverage *M* acting as a mask (i.e., with range values of 0 and 1 and the same extent as *A*, *B*, and *C*). Then, masking each satellite image can be performed with this query:

```
for $s in ( A, B, C ),
    $m in ( M )
return encode( $s * $m, "image/tiff" )
```

The operations available for building filter and processing expressions are based on only two primitives, a coverage constructor and an aggregation operation, called condenser.

B. Coverage Constructor

A *coverage constructor* creates a d-dimensional grid coverage for some $d \geq 1$ by defining the coverage's domain set, range type, range set, and metadata through expressions. This allows deriving entirely new shapes, dimensions, and values.

The coverage domain set is built from a CRS defining the multi-dimensional axes and the meaning of coordinates, including units of measure; indicating the coordinates of the direct positions, i.e., the points where values sit.

Axis names can be chosen according to the rules of CF; it is recommended to keep native and grid CRS axis names disjoint.

A range type expression optionally creates the coverage range type. In the scope of the embedding CPF condensers this expression defines the range component names as known (immutable) variables. Values derived for some such range component will automatically be cast to the target type of that range component.

A range set expression creates the coverage range set. A corresponding scalar expression is evaluated at every direct position of the coverage's domain set.

An optional metadata expression creates the coverage metadata component. As such metadata are not interpreted by the coverage they are represented as a string which may contain any character, depending on the character set supported.

Syntactically, a coverage is built as

```
coverage id [ D ] [ T ] R [ M ]
```

with the following constituents. The *id* parameter is the new name of the coverage. In a concrete service, the name may be required to not exist in the service data pool yet. On this level of abstraction, however, no such requirement is in order.

The *D* parameter defines the domain set, through its constituents CRS, each axis with its type – regular or irregular – and its constituents such as lower and upper bound, resolution (if regular), etc. For example, the following clause defines a 2-D grid with axes *Lat* and *Long*. Both axes are regular with the indicated extent and resolution. The common CRS defining them is WGS84 which has the EPSG code 4326. Further, interpolation along both axes is set to linear:

```
domain set
crs "EPSG:4326" with
  Lat  regular (10:30) resolution 0.01
    interpolation linear,
  Long regular (10:30) resolution 0.01
    interpolation linear
```

A typical range type definition *T* for 3-band RGB images is expressed as follows:

```
range type
  red  :integer,
  green:integer,
  blue :integer
```

The range set definition provides an expression which is evaluated for each cell („direct position" in CF terminology) for determining the value of each cell in the datacube. An example for such an expression is

```
range set (integer) $c / 2
```

Every cell value of the output coverage is given by taking the corresponding value of the $c coverage, dividing it by 2, and casting the result to an integer value.

This implicit iteration over all elements defined in the domain set, i.e., its direct positions, allows for operations that sometimes are called „embarrassingly parallel", or local operations in Tomlin's Map Algebra [14]. However, the mechanics of the constructor provides not only the iteration over all the direct positions, but also the coordinate iteration variables, available through the axis names. In the following example the result value for each direct position is determined from subtracting two adjacent time slices in coverage *$c*. In a 3-D x/y/t timeseries datacube this would effectively determine change:

```
range set
  $c[ date(date-1) ] - $c[ date(date) ]
```

Note that the axis name serves to identify the axis addressed and also the variable contents. The syntactic position guarantees differentiating both positions. Beyond local operations this allows expressing also Tomlin's zonal, focal, and global operations.

The metadata element *M*, finally, allows associating a metadata string. As the syntax and semantics of the metadata remains opaque and unknown in this framework there are no particular rules on this string – it might be XML, JSON, or any other structure defined by some concrete standard based on CPF. In CPF, this might simply be written as

```
metadata "any metadata contents"
```

In a fully fledged coverage constructor all these clauses are present. However, in particular where other coverages are referenced some details often can be inferred – for example,

if for some given datacube all values simply are halved then obviously the domain set of the derived cube is that of the original cube. In many cases this yields very compact expressions, such as

```
coverage LogOfCube
range set log( $c )
```

This gives rise to induced expressions: For cell-wise operations the result coverage is trivially defined through the input coverage constituents. We simply write the range set expression and define its semantics through the corresponding coverage constructor. All common arithmetic, comparison, trigonometric, and exponential become immediately available this way, and additionally „if" statements, records access, etc. become defined naturally.

We illustrate these unary, binary, and n-ary induced operations by a binary addition of two coverages *$c* and *$d* which both need to agree in their domain set – say, axes *x* and *y* – and need to have compatible range types. Then,

```
$c + $d
```

is equivalent to

```
coverage Sum
domain set domainSet($c)
range set
  $c[ x(x), y(y) ] + $d[ x(x), y(y) ]
```

Conversely, by manipulating the domain set while retaining the original coverage values we can define extraction of sub-coverages. Following the WCS standard [4], we differentiate subsetting into *trimming* (which reduces the domain footprint, but keeps the dimension) and *slicing* (which reduces the dimension). Then, subsetting can be expressed by taking the range values of the input coverage at the reduced domain set of coordinates:

```
coverage Subset
domain set
  crs crs($c) with
  x index ( x0 : x1 ),
  y index ( y0 : y1 )
range set
  $c[ x(x), y(y) ]
```

This can be simplified to

```
$c[ x(x0:x1), y(y0:y1) ]
```

For some coverage *$c* with unknown dimension, but with an axis *date* a time slice at a particular date can be written as

```
$c[ date( "2021-08-28" ) ]
```

In practice, combinations of constructor and condenser are common.

C. Coverage Condenser

The second fundamental operator is the condenser. It collapses a coverage into a scalar. On principle, any binary operation forming a monoid applies; CPF defines addition, multiplication, max, min, and, and or.

The general shape of the condenser is:

```
condense op
over name1 axis1( I1 ),
     …,
     named axisd( Id )
[where P]
using V
```

The following expression iterates over a 5000x5000 extent of image *$c* delivering the sum of all values encountered at the Direct Positions.

```
condense +
over x ( 0 : 4999 ), y ( 0 : 4999 )
using $c[ x(x), y(y) ]
```

Again, common shorthands can be defined for simplification, as summarized in Table II.

A timeline diagram can be obtained through a 1-D expression which aggregates over space while iterating over time:

```
coverage AverageTemperature
domain set
  crs "OGC:DateTime" with
  t ( domainSet( $tCube, Date ) )
range type f: float
range set
  condense +
  over  lat( domainSet( $tCube, Lat ) ),
        lon( domainSet( $tCube, Lon ) )
  using $tCube[ Lat(lat),
                Lon(lon),
                Date( t ) ]
```

Further, the practically important class of zonal operations can be expressed. To this end, we first introduce a shorthand notation for coverages with only Cartesian coordinates. 3x3 matrices can be expressed like this:

```
coverage Sobel3x3
domain set
  crs "OGC:Index2D" with
  i ( -1 : 1 ), j ( -1 : 1 )
range type s: integer
range set
  <  1;  2;  1;
     0;  0;  0;
    -1; -2; -1
  >
```

This matrix can now be used as a filter kernel for edge detection as used in a Sobel filter:

```
coverage FilteredImage
domain set
  crs "OGC:Index2D" with
  x index ( 0 : 5000 ),
  y index ( 0 : 5000 )
range set
  condense +
  over  i ( -1 : +1 ), j ( -1 : +1 )
  using   $c.blue[ x(x+i), y(y+j) ]
        * Sobel3x3[ i(i), j(j) ]
```

D. Further Functions

In addition, common convenience functions relevant for geo applications are available in the language, such as scaling and reprojection.

The encode function specifies encoding of a coverage-valued query result by means of a data format and possible extra encoding parameters. The decode function, conversely, evaluates a byte stream passed as parameter to a coverage by decoding the byte stream. This byte stream must represent a coverage encoding following CIS 1.1 [8] and its coverage encoding profiles.

IV. Related Work

A. Standardization

Geo Web services for raster data started with 2D map rendering with OGC *Web Map Service* (WMS). This delivers images suitable for human interpretation, but not data results that may be perused, e.g., by a GIS analysis tool. Further, no processing is supported, just selecting predefined colirng styles.

The emerging OGC *Environmental Data Retrieval* (EDR) standard [9] defines some fixed functionality, in the spirit of (but not to the extent) of OGC WCS. There is only static functionality, but no flexibility for composing requests of arbitrary complexity as in CPF.

OGC WCS, conversely, is a data-oriented service extension, however with requests of fixed, static functionality. One extension, WCPS, offers a datacube analytics language based on the concrete coverage data model CIS; CPF is derived from WCPS and reshaped to be the processing counterpart of the abstract CF data model of ISO 19123-1.

Table II. — CONDENSER SHORTHANDS

reduceExprdefinition[1]	Description

`add($a) =`
`condense +`
`over $p1 (gridDomainSet($a,D1)),`
`…,`
`$pd (gridDomainSet($a,D1)),`
`using $a[ $p1 , …, $pd ]`
	sum over all points in *$a*

`avg($a) =` `add($a) / | gridDomainSet($a) |`	average of all points in *$a*

`min($a) =`
`condense min`
`over $p1 (gridDomainSet($a,D1)),`
`…,`
`$pd (gridDomainSet($a,D1))`
`using $a[ $p1 , …, $pd ]`
	minimum of all points in *$a*

`max($a) =`
`condense max`
`over $p1 (gridDomainSet($a,D1)),`
`…,`
`$pd (gridDomainSet($a,D1))`
`using $a[ $p1 , …, $pd ]`
	maximum of all points in *$a*

`count($b) =`
`condense +`
`over $p1 (gridDomainSet($b,D1)),`
`…,`
`$pd (gridDomainSet($b,D1))`
`where $b[ $p1 , …, $pd ]`
`using 1`
	number of points in *$b*

`some($b) =`
`condense or`
`over $p1 (gridDomainSet($b,D1)),`
`…,`
`$pd (gridDomainSet($b,D1))`
`using $b[ $p1 , …, $pd ]`
	is there any point in *$b* with value *true*?

`all($b) =`
`condense and`
`over $p1 D1(gridDomainSet($b,D1)),…,$pd Dd(gridDomainSet($b,D1))`
`using $b[ $p1 , …, $pd ]`
	do all points of *$b* have value *true*?

[1] *$a* is assumed to evaluate to a coverage with a single numeric range field, *$b* to a coverage with a single Boolean range field.

OGC Web Processing Service (WPS) defines a Web API for remote function invocation, i.e., Remote Procedure Calls (RPC) [11]. This principle exists with C since the 1980s and lateron with SOAP. Any process to be invoked in the server must be defined by the server administrator. The invocation syntax in WPS is described through XML documents; the actual code executed remains opaque to the invoker.

Hence, WPS realizes *syntactic interoperability*: the invocation syntax (function name, parameter number and types) is defined whereas the execution semantics is not. CPF, on the other hand, establishes *semantic interoperability*: clients and services based on CPF share the same understanding of the filtering and processing.

B. Technology

Image processing has a strong history. After using programming languages natively libraries emerged encapsulaing advanced imaging functionality. With the recent proliferation of python libraries in this language have become popular, such as xarray for n-D arrays. These are usually limited to main memory processing and are not directly usable for Web services. Likewise, they require concrete programming and, additionally, do not support space and time semantics directly. In several language specific support has been added for built-in array handling, from APL over Matlab to R. CPF is suitable for describing the datacube-related parts and define interoperability, up to possibly automatic translation across languages and services.

V. Conclusion

We presented a language for expressing geo datacube operations , specifically tailored in its operations to the ISO abstract grid coverage model. This allows manipulating datacubes of any dimension and with space, time, and other axes in a uniform manner, including combination of heterogeneous objects for data fusion.

The first innovation is that, to the best of our knowledge, it is the currently only formalized processing model that strictly relies on the coverage standards.

Further, this approach is novel as it abstracts away from the usual procedural APIs, but rather offers a high-level, declarative language allowing open-ended complexity in the requests while focusing on the „what“ rather than on the „how“. Similar languages are known on vector data, such as SQL Simple Features, so our proposal can be seen as closing a gap, thereby making datacubes first-class citizens in the conceptual framework world of geographic data.

The syntax tentatively is shaped along the XQuery language – the vision is to integrate data and metadata analytics, and many of today's metadata are in XML. Even when changing to JSON, or any other structured metadata description model, XQuery still works. Given this generality of XQuery we have shaped the CPF syntax to prepare for an integration which ultimately should overcome the data / metadata divide.

At the time of this writing the specification is sent out to the voting delegations of the participating nations for Draft International Standard (DIS) ballot. Should it be accepted then only editorial changes will be further possible any more.

An implementation of CPF is possible on principle. In a slightly different syntax One mapping to a concrete standard is exemplified in the 19123-3 specification: the OGC WCS Core and some extensions are described through the CPF language, demonstrating how it can be used to unambiguously describe functionality.

Our hope is that the concepts of this language will help to better communicate algorithms and ideas. Different coverage processing standards might define their semantics through 19123-3 making them comparable, possibly even enabling cross-translation between them. Further, the systematics of coverage processing might guide software implementers in the design of their functionality, using whatever interface style like function libraries, different languages, etc.

Future work includes extending the datacube analytics expressiveness with AI methods, based on the common basis of tensor algebra. Another research direction is to extend support for further coverage types, specifically: point clouds and meshes.

Acknowledgment

The author is grateful for the many discussions with Graham Wilkes, Emmanuel Devys, Kathi Schleidt, and many more experts which has helped shaping the concepts and clarifying many corner cases and use cases.

References

[1] F.L. Bauer, H. Wössner, "Algorithmic Language and Program Development". Springer 1982

[2] P. Baumann, "The OGC Web Coverage Processing Service (WCPS) Standard". Geoinformatica, 14(4)2010, pp 447-479

[3] P. Baumann, "OGC Web Coverage Processing Service (WCPS) Language Interface Standard". OGC document 08-068r3, https://docs.ogc.org/is/08-068r3/08-068r3.html

[4] P. Baumann, "OGC Web Coverage Service (WCS) Interface Standard – Core". OGC document 17-089r1, http://docs.opengeospatial.org/is/17-089r1/17-089r1.html

[5] P. Baumann, "A General Conceptual Framework for Multi-Dimensional Spatio-Temporal Data Sets". Environmental Modelling and Software (2021), https://doi.org/10.1016/j.envsoft.2021.105096

[6] P. Baumann, D. Misev, V. Merticariu, B. Pham Huu, "Datacubes: Towards Space / Time Analysis-Ready Data". In: J. Doellner, M. Jobst, P. Schmitz (eds.): Service Oriented Mapping - Changing Paradigm in Map Production and Geoinformation Management, Springer Lecture Notes, 2018, DOI 10.1007/978-3-319-72434-8_14

[7] P. Baumann, "Language Support for Raster Image Manipulation in Databases". Int. Workshop on Graphics Modeling, Visualization in Science & Technology, Darmstadt/Germany 1992, pp. 236 – 245, DOI 10.1007/978-3-642-77811-7_19

[8] P. Baumann, E. Hirschorn, J. Maso: *Coverage Implementation Schema (CIS), version 1.1.1.* OGC document 09-146r6, docs.opengeospatial.org/is/09-146r8/09-146r8.html

[9] M. Burgoyne, D. Blodgett, C. Heazel, C. Little, "OGC API - Environmental Data Retrieval Standard". http://docs.ogc.org/DRAFTS/19-086.html

[10] Hidden ISO, "Coverage Processing Fundamentals", https://external.ogc.org/twiki_public/pub/CoveragesDWG/CoveragesBigPicture/ISO_19123-3_2021-09-01.docx

[11] M. Mueller, B. Pross, "OGC® WPS 2.0.2 Interface Standard Corrigendum 2". OGC document 14-065, http://docs.opengeospatial.org/is/14-065/14-065.html

[12] N..n., "OGC API - Processes". https://github.com/opengeospatial/ogcapi-processes

[13] N.n., "OGC API - Coverages". https://github.com/opengeospatial/ogcapi-coverages

[14] C. Tomlin, "Map algebra: one perspective". Landscape and Urban Planning, Volume 30, Issues 1–2, 1994, pp. 3-12

[15] W3C: "XQuery 1.0: An XML Query Language (Second Edition)". https://www.w3.org/TR/2010/REC-xquery-20101214