Spatio-Temporal Gridded Data Processing on the Semantic Web

Andrej Andrejev*, Dimitar Misev†, Peter Baumann† and Tore Risch*
*Department of Information Technology, Uppsala University, Uppsala, Sweden
†Computer Science & Electrical Engineering Department, Jacobs University, Bremen, Germany

***Abstract*—Multidimensional array data, such as remote-sensing imagery and timeseries, climate model simulations, telescope observations, and medical images, contribute massively to virtually all science and engineering domains, and hence play a key role in 'Big Data' challenges. Pure array storage management and analytics is relatively well understood today. However, arrays in practice never come alone, but are accompanied by metadata, including domain, range, provenance information, etc. The structure of this metadata is far less regular than arrays or tables, and may be incomplete or different from one array instance to another. Particularly in the field of the Semantic Web such integrated representations must convey a sufficiently complete and reasonable semantics for machine-machine communication. We show how the Resource Description Framework (RDF), the Semantic Web graph model for metadata, can be leveraged for such data/metadata integration specifically for representing spatio-temporal grid data. Based on the notion of a coverage as established by the Open Geospatial Consortium (OGC) we present a hybrid data store where efficiently represented arrays are incorporated as nodes into RDF graphs and connected to their metadata. We have extended the Semantic Web query language SPARQL to incorporate array query semantics and other functionality making it suitable for processing of large numeric arrays, including geo coverages.**

I. Introduction

Massive multi-dimensional arrays play a central role in science, engineering, and beyond. Consequently, a significant number of approaches for storage and retrieval on arrays have been proposed, both by scientific communities (such as [32], [1]) and in databases, where efforts have led to the new class of array database systems (such as [16], [10], [11], [36], [34], [18]) with impact on standardization [12][28].

In practice, though, arrays typically are forming part of some larger data structure. Metadata support in this context has been largely ignored, leading to a widely acknowledged *impedance mismatch* that is generally detrimental to the development of modern web-enabled, flexible and scalable scientific applications [20].

Let us look at a representative example. The notion of a *coverage* in geoinformatics describes a digital representation of some space-time-varying phenomenon, in practice: regular and irregular grids, point clouds, and meshes [24], [13]. Grid coverages consist of an array ornamented with metadata essentially describing the location of each grid point in space-time. Technically, coverages as per the respective OGC standard [13] additionally can hold any kind of domain and use case specific information. Earth Science communities have established their own metadata standards, such as EO-Metadata [19], and recently web service protocols [14], [12], which tentatively focus on allowing any kind of application specific metadata to be associated with the data.

Any conceptual modeling of a coverage, therefore, must be able to integrate arrays into the common framework. In particular in the realm of the Semantic Web and machine-to-machine communication it is indispensable to represent the full semantics in a coherent manner. On metadata level this is well under way – there is a strong trend towards metadata annotation of resources on the web within the Semantic Web paradigm; obviously, standardized, commonly understood metadata facilitate data integration and building federated datasets, thus making scientific data more accessible to the users. The relevant common vocabularies and ontologies include RDFS [21], VoiD[6], SKOS [3], RDF Data Cube [2].

As of today, coverage data are not yet accessible within the RDF framework, mainly due to lack of efficient representations and operations on arrays. To remedy this, an effort towards integration of coverages (and, hence, arrays) into RDF has started in the World-Wide Web Consortium (W3C) as part of the initiative on describing spatial image data (generally: gridded coverages) through RDF [4]. No results are available yet, and actually this W3C standardization work where we are engaged has motivated this research.

We propose the Semantic Web Ontology capturing all aspects of the metadata essential for understanding and querying Grid Coverages, while staying open and flexible enough for extensions with additional application-specific metadata. Arrays are incorporated as nodes into the RDF graph, and connected to all their metadata. We suggest using hybrid data stores, efficiently handling array data and RDF. Mediator technology accomplishes dispatching of sub-queries to dedicated database engines, while achieving an integrated logical representation. We encourage the users to combine array data and metadata within single queries, which brings the following benefits:

- Queries are more transparent and self-contained than array-only queries; there is no need to explicitly encode metadata values in queries, since the metadata can be retrieved from the same database.
- Metadata is used for both result selection and post-processing on the server, reducing the communication costs and improving scalability in contrast to client-based postprocessing.

- Clients can conveniently get answers to complex problems with just a single round trip, rather than through iterative communication with several servers employing different models and retrieval paradigms.
- The query optimizer has more freedom in building better access plans, when both metadata and data can be combined in the same query.

In our previous work [9], [7] we extended the Semantic Web query language, SPARQL, to incorporate array query semantics, together with additional functionality (second-order functions, closures, user-defined functions, storage extensibility) making it suitable for processing large numeric arrays, including grid coverages - this language we call SciSPARQL. Our implementation of SciSPARQL is capable of answering metadata-rich array queries in addition to the traditional array retrieval and processing requests. By modeling grid coverages as introduced above we show how to translate processing requests on grid coverages to equivalent SciSPARQL queries based on the formal Grid Coverage Ontology, thus opening raster geospatial data to the Semantic Web users. The query semantics is based on the OGC Web Coverage Processing Service (WCPS) standard [12], a geo query language grounding on the OGC coverage model. Our current work couples SciSPARQL with the rasdaman array database, where array-relevant SciSPARQL subqueries are pushed down into rasdaman to benefit from the array query optimization and parallelization available in rasdaman. For this purpose, we have further extended SciSPARQL with 2nd-order array functions `ARRAY()`, `MAP()`, `CONDENSE()`, as described in Section 3, which offer great flexibility to the user and correspond to rasdaman array processing primitives.

In the next section we give an overview of work related to our effort presented here. Section III presents SciSPARQL, with an integration with rasdaman as a backend array database system. Section IV presents our RDF ontology on Grid Coverages, along with examples of representative queries. Finally Section V concludes the paper.

II. Related Work

Arrays play a core role in science, engineering, and beyond. Traditionally, file formats (such as netCDF [32]) have been established and subsequently been enhanced with processing interfaces, such as OPenDAP [1]. These are constrained in their functionality and lack joins between datasets, etc. Conversely, tools like R offer advanced array processing, but do not scale well beyond main memory sizes. Our approach combines expressiveness of an array query model and language with proven scalability [16].

In databases, arrays have found attention only gradually. Mapping arrays to BLOBs looses all semantics, and consequently adequate query semantics cannot be provided; additionally, this linearization destroys spatial proximity on disk. An initiative is under way, though, to extend the ISO SQL standard with arrays [28].

Array databases have been pioneered with rasdaman [10] and Array Algebra [11]. Declarative array primitives are embedded in expression languages which are optimizable and parallelizable. Today there is a number of array database systems in different stages of development, such as SciQL [36], SciDB [34], and Ophidia [18].

GeoSPARQL [29] is OGC's effort to add support for representing and querying geospatial data on the Semantic Web. Since GeoSPARQL is based on the Simple Feature Access ISO 19125 standard [25], [23], geospatial data in this case actually refers to *vector* data, rather than gridded, raster data. Modeling of raster data in RDF is explored in [35], although focusing on the representation of feature objects extracted from the data and their relationships. With our approach, gridded structure of the data is preserved in order to be queried.

Integration of arrays into overall models varies. While "pure" array databases like SciDB do not emphasize integration, others address such aspects, currently mainly for SQL [36][28]. An integration of arrays into RDF has started in W3C as part of the effort to describe spatial image data (generally: gridded coverages) through RDF [4]. No results are available yet, and this is where the work in this paper comes in. xWCPS [27] is a query language integrating WCPS [12] and XQuery [17]. As such it aims to achieve similar goals, but with XML as the model for semi-structured metadata associated with scientific data.

The Resource Description Framework (RDF) [26] graph data model was designed for expressing metadata about all kinds of resources on the web, including databases, raster images, etc. When it comes to storing multidimensional numeric data, pure RDF suggests two main approaches: nested collections and RDF Data Cube vocabulary [2], the latter being a Semantic Web adaptation of SDMX (Statistical Data and Metadata eXchange) [5], designed to exchange OLAP cubes. Both of these approaches create a number of graph nodes for each numerical element to store. Apart from inefficiency arising from this 'too general' graph-based storage and processing of arrays, this representation also fails to give important guarantees about the data structure, and the SPARQL queries lack the clarity of the array access. Our RDF with Arrays data model addresses all these issues, and SciSPARQL query language incorporates the array query semantics. Array-based storage of RDF collections and RDF Data Cubes is supported with our system, providing backwards-compatibility with original RDF representations.

III. Scientific SPARQL overview

In this work we show that our data model, RDF with Arrays, is well-suited for representing spatio-temporal gridded coverages, together with any metadata, including those currently used in WCS/WCPS requests. We model arrays as value nodes in RDF graph, and each array has associated element type and shape (i.e. sizes in each dimension).

The language: SciSPARQL [8], [9] is an extended version of the W3C query language SPARQL 1.1 [22]. SciSPARQL extends SPARQL with syntax and semantics for selecting and processing numeric array data along with the metadata

represented in RDF terms. The extensions (prior to this work) include:

- syntax for array *dereference*, *slicing*, and *range selection* (optionally with a *stride*). The syntax is borrowed from MATLAB, array subscripts are 1-based, and ranges are specified as *lo:stride:hi* (for each dimension independently), with *hi* value always in range[1].
- binding of free query variables to the sets of available array subscripts, for example an expression `?A[7, ?j]` would bind (otherwise unbound) `?j` variable to all column indices in `?A` matrix.
- functions to transform the arrays, permuting the array dimensions (generalized N-dimensional transposition), obtaining array shape and element type;
- library of array aggregate functions (operating across array elements), and SPARQL 1.1 standard bag-oriented aggregate functions like SUM, AVG, MIN, MAX, redefined to handle bags of arrays as well;
- extensibility with algorithms expressed in conventional algorithmic languages, like Python, Java, or C. It is possible to define both regular and aggregate functions, thus making use of any existing computational libraries;
- *functional views*, which are user-defined functions expressed in terms of SELECT queries. This allows for building up libraries of SciSPARQL subqueries defining standard computation formulas or common data retrieval tasks for further use.
- *second-order functions*, like ARGMIN and ARGMAX, taking *functional closures* as arguments. A *closure* is a function call with some of parameters specified and some free, marked by *. For example, an expression `ARGMAX(f(*, 5))` would return the value for the first argument of function *f()*, where it reaches its maximum, while the second argument is fixed to 5. Certainly, the set of allowed values for the free argument to *f()* should be finite, which is typically the case with functional views, for example:

```
DEFINE f(?x ?y) AS
SELECT ?value
WHERE { ?o a :Observation ;
           :x ?x ;
           :y ?y ;
           :value ?value }
```

During query processing the functional views and certain second-order functional expressions are expanded similarly to SQL views, in order to give the query optimizer greater freedom for finding the optimal order of execution. As a query language, SciSPARQL is declarative, optimizable, and terse. This means that most kinds of conditions and constraints on the data retrieved from a database can be expressed directly as algebraic equations. It is the responsibility of the DBMS to come up with a good execution plan, taking into account storage statistics, distribution, communication and computation time estimates. The user-defined foreign functions can be provided cost and cardinality models for improved query optimization.

Array storage: SciSPARQL is implemented with Scientific SPARQL Database Manager, which includes query processor, functional extensibility mechanisms, and an in-memory database, designed for efficient storage of RDF with Arrays. In order to provide scalability beyond the memory bounds, a highly flexible storage backend interface is included. This flexibility comes from utilizing different capabilities of storage back-ends, including selection of array subsets, computing aggregate functions etc. Currently supported storage back-ends include RDBMS (arrays are stored in binary chunks), specialized file formats (e.g. `.mat` files used together with MATLAB integration in [7]), and specialized array stores [9].

Whenever an array is stored in a back-end system, it is represented by an *array proxy* object in the RDF graph. Array dereference, slicing and range selection operations are stacked by deriving one proxy object from another during the query execution (which is very cheap), so the array data is retrieved lazily - only when it is needed for computations. This helps greatly reducing disk access and communication overheads, compared to 'eager' data retrieval.

Integration with rasdaman and new features: The integration with the rasdaman array database [31] is an important step forward, since the latter system has a rich array algebra [10], [11] implementation, which can handle most of the array processing workload expressed in SciSPARQL queries. While SciSPARQL is handling query processing and communication with the client, an extensive set of array operations is propagated to rasdaman together with data retrieval requests. This should result in practically the same performance, as if rasdaman was addressed directly, while providing the full power of RDF metadata querying available with SciSPARQL.

For the purpose of this integration, SciSPARQL was extended with three more second-order functions, which translate clearly into MARRAY operator calls in rasdaman:

- `ARRAY(type, shape, mapper)` array constructor, where `mapper` function or closure is called to compute the value of each cell, given a 1-dimensional array of logical cell subscripts as an argument;
- `MAP(type, mapper, v1, ..., vn)` array mapper, constructing a new array filled with results of `mapper` function or closure, applied to the respective elements of `v1`, ... `vn` aligned arrays;
- `CONDENSE(op, v, filter)` generalized array aggregation, where the aggregate operation `op` (any SciSPARQL aggregate function) is applied to elements of the array `v` for which the `filter` function or closure returns *true*.

The element-wise array operations, including arithmetic `.+`, `.-`, `.*`, `./`, `.^`, comparison `.=`, `.>`, `.>=`, `.<`, `.<=`, and logical `.&`, `.|` are defined for operand types (array, array), (array, number) and (number, array) as simple yet polymorphic shortcuts for the respective MAP operations. For example, the expression `?a .* ?b` can be also expressed as

```
MAP(xsd:double, times(*, *), ?a, ?b)
```

[1]We also support a Python-compliant dialect of SciSPARQL, with 0-based subscripts and a NumPy range syntax

if both `?a` and `?b` are arrays, and their widest numeric type is *xsd:double*, or

```
MAP(xsd:integer, times(*, ?b), ?a)
```

if `?b` is integer and `?a` is an array of integer. Both examples are using built-in function `times()`, an alias for `*` multiplication operator.

The intergated solution utilizes the back-end interface of SciSPARQL Database Manager, translating query predicates to rasdaman API calls whenever possible, thus pushing the computations closer to the array storage, and retrieving smaller amounts of data from rasdaman.

IV. Grid Coverages in RDF

Multidimensional grids, both regular and irregular, come up as the natural representation of various space/time varying data in the geospatial domain, such as 1D time-series, 2D remote sensing imagery, 3D x/y/t image time-series and x/y/z geophysical data, as well as 4D x/y/z/t atmospheric and ocean data. Dealing with any of these types of data requires taking into account more than just the array data itself. The location information of the array contents is needed to properly relate such values to physical positions in the world. Frequently, elevation / depth and time coordinates have to be considered as well.

In geoinformatics, gridded data form a special case of coverages with its standards mainly maintained by the Open Geospatial Consortium (OGC). According to the abstract model of ISO 19123 [24] and its OGC implementation model [13], a *coverage* represents some space/time varying phenomenon, with subtypes available for regular and irregular grids, point clouds, and meshes. Formally, a coverage establishes a function mapping from a given multidimensional domain (the set of *direct positions* altogether forming the *domain set*) to some value set (referred to as the *range set*, described by the coverage's *range type*). We call a location in the domain set together with the value it is associated to a *cell*. For a 7-band Landsat satellite image, for example, the domain set is 2D with point coordinates expressed in some horizontal Coordinate Reference System (CRS) such as WGS84; the range type is a 7-component structure of unsigned integers. For image timeseries, the domain set is 3D with the additional third axis describing image acquisition times.

The domain set is described in varying complexity, depending on the grid type (see Figure 1). Regular grids (Figure 1 left) require only an origin vector plus one single offset vector describing the (uniform) stepping in each direction. Irregular grids (Figure 1 middle right), where axes are straight but at individual distances along each axis, require a list of offset vectors per axis indicating the grid distances at each direct position. Warped grids (Figure 1 right), finally, abandon any fixed location and allow arbitrary placement of the direct positions (as long as the grid topology is maintained); therefore, for each grid point its individual location has to be maintained, effectively requiring a second array whose values are coordinates. Additionally, grids can be skewed (Figure 1 middle left).

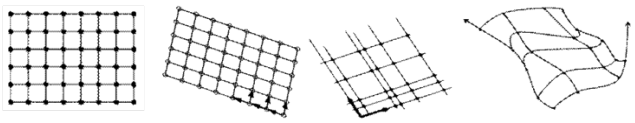

Fig. 1. Grid types [30].

The OGC complements this coverage concept with the Web Coverage Service (WCS) standards suite [14], [12], a concrete, interoperable and modular coverage implementation model [13]. Our focus is on WCPS as it establishes a declarative coverage query language. As XML today is the most widely used metadata format in geo services, WCPS has been crafted compatible with XQuery syntax [12]. Based on a `for` loop over coverage objects the `where` and `return` clauses may contain coverage expressions. In these expressions, coverages can be created, combined, and aggregated. The basic constructs are the coverage constructor, which delivers a new coverage possibly derived from others, and the condenser (aka aggregator), which summarizes over a coverage by iterating over its direct positions. Shorthand operations are available for all usual arithmetic, exponential, trigonometric, etc. functions. The following example shows the flavour of the language: *"From MODIS scenes M1, M2, M3 retrieve the pixel-wise difference between the red and nir channels, encoded in TIFF – but only those where nir exceeds 127 somewhere"*.

```
for $c in ( M1, M2, M3 )
where some( $c.nir > 127 )
return encode( $c.red - $c.nir, "image/tiff" )
```

A. RDF Grid Coverage Ontology

Here we establish an RDF ontology, of the OGC coverage model, that would allow native integration of coverages within the Semantic Web ecosystem, metadata modelling and coverage analytics with SciSPARQL queries. The native components of gridded coverages are:

1) **coverage identifier**
2) **domain set** describing the spatio-temporal region of interest, within which the coverage values are defined. One of the below for grid coverages:
   - a geometric **grid** of equidistant points, described with `integer` *dimension* d, *axis names* (`string vector` of size d) and the *coordinates* of diagonally opposed corners of a rectangular region (low and high limits as `integer vectors` of size d).
   - a **rectified grid** for which there is an affine conversion between the grid coordinates and the coordinates of an external coordinate reference system (CRS). It is defined by the coordinate (in some CRS) of the grid *origin* (a `tuple` of size d, which can be a mixture of `float`, `integer`, or other user-defined type of values, depending on the CRS), and d *offset vectors* of size d (same type as the origin coordinate) that determine the grid spacing in each direction in terms of the CRS.

- a **referenceable grid** which explicitly specifies the coordinates of each position individually (as a complete list).

3) **range type**, a structure description and technical metadata required for an appropriate (however, application independent) understanding of a coverage; modeled on the definition of 'Record' from [33]. It is composed of one or more **fields**, each of which having its own:
   - **name** identifier
   - human readable **description**
   - **type definition** referring with a URI to an ontology for example
   - a list of **allowed values**, or an interval within which the range values must fit in
   - a list of **nil values** used to denote that a value is not available with a URI referencing a human readable reason
   - **unit of measure** specified as a `string` according to the Unified Code for Units of Measure[2], or a URI referring to an externally defined UoM
   - whether the field is **optional**
   - whether the data is **updatable**
4) **coverage function**, mapping domain locations to range attribute values, which can be:
   - **Grid function** provides an explicit mapping rule for grid geometries, consisting of a *start point*, the index position of a point in the grid that is mapped to the first point in the range, a *sequence rule*, the method for sequential enumeration of the grid points, that can be one of Linear, Boustrophedonic, Cantor-diagonal, Spiral, Morton or Hilbert, as defined in [24], and an *axis order* as a list of axes (the incrementation order to be used on an axis of the grid, positive or negative).
   - **Coverage mapping rule** which can be a formal (in MathML for example), informal as text, or a reference to an external description of the coverage function.

Ontology: One benefit of representing the metadata with RDF in general, is that the set of properties for each instance of each class becomes open: applications are free to enrich the metadata without necessarily changing the schema. So the schema (formally defined in RDFS format by means of classes, properties and relationships) remains very simple, and captures only the most common and essential features. (In contrast, relational schemas tend to address every tiny need of every conceived application, thus becoming overly complex.

We will be using `GridDomain` and `RangeType` classes to capture all the information about the domains and ranges of a grid coverage. As shown on Figure 2, a GridCoverage instance would have an `id` and a references to `GridDoman` and `RangeType` instances, so that it is clear when two coverage instances are *'aligned'* - they share the same `GridDomain`, or *'uniform'* - they share the same `RangeType`. Besides that, GridCoverage instances will have array-valued properties, defined as instances of `Field` class.

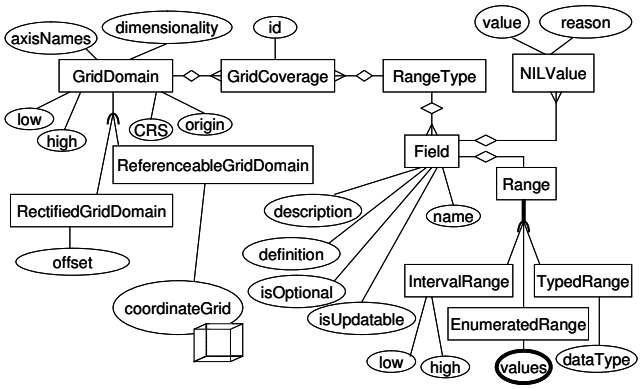

Fig. 2. Grid Coverage Ontology.

The variety of domain types is captured by `GridDomain` subclasses. Any `GridDomain` has dimensionality, a list of axis names, `low` and `high` vectors containing the bounding values for each dimension. All domain types refer to a particular Coordinate Reference system (grid's own coordinate axes in the simplest case), identified by a URI pointing to a GML document [15], and the *origin* vector. Rectified grid domains will contain an *offset* vector, used in a simple coordinate transformation. `ReferenceableGridDomain` instances require a geo-coordinate vector for every point in the grid, together represented by `coordinateGrid` array attribute. This would be a problem in any storage system that rigidly separates data from metadata but *RDF with Arrays* is designed to handle big arrays as part of RDF graph, and there are no restrictions on how small the "metadata" should be.

A `RangeType` is simply a collection of fields, jointly specified to by GridCoverage instances. The coverage fields, as specified by SWE [33] standard, have textual *name* and *description*, URI *definition*, specifications of reserved *NIL values*, and, most importantly, `Range`. There is a number of ways to define the range of a field, modelled by `Range` subclasses. One option is to define a range as a basic RDF/XSD type, e.g. `xsd:integer`. Alternatively, one could provide an interval of values (`IntervalRange`), or a full list of possible values.

Example coverage: To illustrate the use of our RDF Grid Coverage Ontology, we make an RDF description of a coverage "myCoverage", containing 3-dimensional data along x/y/t axes, without binding to any particular CRS (so the example is self-contained). First, we are going to specify the domain, including `low` and `high` vectors:

```
@prefix : <http://udbl.uu.se/GCO/> .
@prefix ex: <http://udbl.uu.se/GCO/example> .
ex:myGridDomain a :GridDomain ;
                :dimensionality 3 ;
                :low  (-100, -100, 0) ;
                :high (100, 100, 365) ;
                :axisNames ("x" "y" "t") .
```

and the range, including the definitions of each field:

[2]http://unitsofmeasure.org/ucum.html

```
ex:myRangeType a :RangeType ;
               :hasField ex:nir ;
               :hasField ex:red .
ex:nir a :Field ;
       :name "nir" ;
       :description "Near infra-red part of the spectrum" ;
       :definition <http://opengis.net/def/
                    property/OGC/0/Radiance> ;
       :hasRange ex:greyScale ;
       :isUpdatable true ;
       :hasNILValue ex:SensorFaultNIL ;
       :unitOfMeasure "N/A" .
```

(We omit the definition of the similar `ex:red` field) The interval range and the allowed NIL values are specified as follows:

```
ex:greyScale a :IntervalRange ;
               :low 0 ;
               :high 255 .
ex:SensorFaultNIL a :NILValue ;
                  :reason <http://www.opengis.net/def/
                           property/OGC/0/
                           BelowDetectionRange> ;
                  :value 0 .
```

And finally, comes an instance of our coverage, referring to the `nir` and `red` variables storing array data in a NetCDF file:

```
ex:myCoverage a :GridCoverage ;
              :id "myCoverage" ;
              :hasDomain ex:myGridDomain ;
              :hasRange ex:myRangeType ;
              ex:nir <file://myCoverage.nc#nir> ;
              ex:red <file://myCoverage.nc#red> .
```

This is a complete example, and a queryable *RDF with Arrays* dataset. Below, we show how SciSPARQL queries can be used instead of WCPS requests (which do not handle any metadata beyond what is described in this section), and xWCPS queries, which address an open set of metadata found in GML documents.

B. WCPS/xWCPS requests in SciSPARQL

Slicing: Get a slice from the coverage with id 'myocean', at t=25 in png format

```
for $c in ( myocean )
return
  encode( $c[t(25)], "png")
```

In SciSPARQL it would correspond to array slicing expression

```
SELECT (?v[:,:,25] AS ?result)
 WHERE { ?c :id "myocean" ;
            ex:value ?v }
```

NDVI: Derive Normalized Difference Vegetation Index NDVI on the fly. The NDVI is a measure of the probability of vegetation in remote sensing: The closer to +1 a pixel value is, the more likely it is plants.

```
for $c in ( mycoverage )
return
  encode(((unsigned char)
    (((((float)$c.nir - $c.red) /
       ((float)$c.nir + $c.red)) > $x) * 255),
  "png")
```

In SciSPARQL, we expect the RDF graph to contain two aligned arrays storing the NIR and RED components as the properties `:nir` and `:red` of the node with id `"mycoverage"`.

```
SELECT (MAP(xsd:integer, NDVI(*, *, $x),
            ?nir, ?red) AS ?result)
 WHERE { ?c :id "mycoverage" ;
            ex:nir ?nir ;
            ex:red ?red }
```

where function NDVI is mapped to the corresponding pairs of array elements

```
DEFINE FUNCTION NDVI(?nir ?red ?x)
 AS SELECT (255 * xsd:integer(
   ((?nir - ?red) / (?nir + ?red)) > ?x) AS ?result)
```

Note that this translation separates the standard computation involved from the data retrieval per se. A query can hard-code a particular data instance (identifying it by unique :id value in this case), while the computation function NDVI can be used in other queries, both for post-processing and filtering.

Average chlorophyll concentration: Return the average chlorophyll concentration for the whole area for a given time step for the 'myocean' coverage:

```
for $c in ( myocean )
return
  avg(($c[t(29)] > 0.0) * $c[t(29)] )
```

This request constructs a Boolean array for the selected slice first, and than uses it as a mask when computing the average for that slice. SciSPARQL also has element-wise comparison and multiplication operations, so formulating an equivalent query is straightforward:

```
SELECT (AVG((?v[:,:,29] .> 0.0) .* ?v[:,:,29]) ?result)
 WHERE { ?c :id "myocean" ;
            :value ?v }
```

The alternative expression of the same query is using a *condenser*, which would skip allocating an intermediate Boolean array:

```
for $c in ( myocean )
return
  (condense +
   over $i x(imageCrsDomain($c, x)),
        $j y(imageCrsDomain($c, y))
   using $c[x($i), y($j), t(29)]
   where $c[x($i), y($j), t(29)] > 0.0) /
  (condense +
   over $i x(imageCrsDomain($c, x)),
        $j y(imageCrsDomain($c, y))
   using 1
   where $c[x($i), y($j), t(29)] > 0.0)
```

Here, we directly use a SciSPARQL condenser, which allows to use built-in AVG aggregate function:

```
SELECT (CONDENSE(AVG, ?v, gt(*, 0)) AS ?result)
 WHERE { ?c :id "myocean" ;
            :grid ?v }
```

The built-in function `gt()` used for condenser condition returns *true* when first argument is greater than second argument, and *false* otherwise. Next example uses a similar `eq()` built-in function for equality.

Histogram computation: Compute histogram for the values between 0 and 255:

```
for $c in ( myocean )
return
  encode(coverage histogram over $n x(0:255)
         values count( $c = $n ), "csv" )
```

In SciSPARQL, we would use `ARRAY()` constructor, which maps every coordinate vector with the provided functional argument. The first two arguments are the array type and shape, the latter being specified using the literal array constructor `A()`.

```
SELECT (ARRAY(xsd:integer, A(256),
              histogramEq(?c, *)) AS ?result)
 WHERE { :myocean :grid ?c }
```

where the function `histogramEq()` counts the number of elements in the given array `?a`, which are equal to the first component in `?coords`:

```
DEFINE FUNCTION histogramEq(?a ?coords)
 AS SELECT (CONDENSE(COUNT,
            ?a, eq(*, ?coords[0])) AS ?result)
```

The interesting aspect about the above query is the superposition of two second-order array functions: constructor over condenser.

Search metadata, return data: This is a xWCPS request addressing the web service at http://acme.com, that has information about a number of coverages and their associated metadata. It returns the difference between red and near-infrared channels of each coverage of Austria, in which some near-infrared cell has a value greater than 127:

```
for $c in doc("http://acme.com")//coverage
where
  some( $c.nir > 127 ) and
  $c/metadata/@region = "Austria"
return
  encode( $c.red - $c.nir, "image/tiff" )
```

If we use SciSPARQL, we would send the following query to the SciSPARQL endpoint with access to an *RDF with Arrays* graph, that contains nodes of type `:Coverage` with their `:nir` and `:red` array properties, and string `:region` properties.

Notice that `some` is a shortcut for a condenser with OR (whereas `all` would be a shortcut for a condenser with AND)

```
SELECT (MAP(xsd:integer, minus(*, *),
            ?red, ?nir) AS ?result)
 WHERE { ?c a :Coverage ;
            :region "Austria" ;
            :nir ?nir ;
            :red ?red .
         FILTER ( CONDENSE(OR, ?nir, gt(*, 127)) ) }
```

Discover anomalies: Count the exceedances of fractional snow cover with respect to a given threshold of 30.

```
for $c in ( SnowCover )
return
  encode(
    coverage count_cov
    over $px x(imageCrsDomain($c, Long)),
         $py y(imageCrsDomain($c, Lat))
    values count($c[Long($px), Lat($py), *] > 30),
  "csv")
```

Figure 3 visualizes the result. The function `imageCrsDomain()` returns the `low` and `high` components of grid bounding box along the specified axis. In WCPS, the axis is specified by either grid axis name, or corresponding (by order of enumeration) CRS axis name, as `Long` and `Lat` in this example. In SciSPARQL, since all we need is the grid size defined by low and high properties of `GridDomain`, we simply use vector .- operation to get the new image size.

```
DEFINE FUNCTION gridDomainSize(?coverage)
 AS SELECT (?high .- ?low AS ?result)
     WHERE { ?coverage :hasDomain [ :low ?low ;
                                    :high ?high ] }
```

Now the query can be expressed as an array constructor, creating the array according to the first two components of the coverage size:

```
SELECT (ARRAY(xsd:integer,
              gridDomainSize(?coverage)[1:2],
              ex:snowHistogram(?gray, *, 30))
        AS ?result)
 WHERE { ?coverage :id "SnowCover" ;
                   ex:gray ?gray }
```

where we define our function `ex:snowHistogram()` to count the elements of the given array along the third dimension, with coordinates in the first two dimensions specified in the argument:

```
DEFINE FUNCTION ex:snowHistogram(?gray, ?coords,
                                 ?threshold)
 AS SELECT (CONDENSE(COUNT,
                     ?gray[?coords[1], ?coords[2], :],
                     gt(*, ?threshold))
            AS ?result)
```

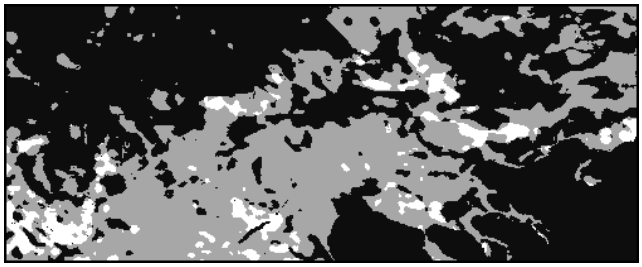

Fig. 3. Count the number of fractional snow cover exceedances over a region of interest.

V. Conclusion

Providing a common and flexible framework for storage and machine-readable annotation of spatio-temporal grids has been in the focus of Earth Science communities' efforts for a long time. This is becoming more important and challenging, as the diversity of the datasets and the applications increases. Obviously, one size does not fit all, so a model for *extensible* metadata is certainly needed. RDF is one such model, widely adopted by the Semantic Web community, in order to describe all kinds of resources on the web. Any instance of any RDF class can be completed with properties specified in any vocabulary, thus forming a graph of globally-identifiable class and instance nodes, which any application is free to extend.

In this work we bring together the best from the two worlds - a mature and flexible array store, efficiently handling a wide class of array retrieval and processing requests; and a powerful query language, so that queries combining raster/grid/array data and graph-structured metadata can be optimized and processed entirely on the server. Useful Array Algebra abstractions have been defined as part of the query

language, including second-order functions and closures. And, since our integrated solution is extensible both storage-wise and computation-wise, there is practically no limit to the kinds of functionality a user can invoke from a query.

Certainly, the extensions to SciSPARQL, as well as the integrated storage solution, combining it with rasdaman, are useful well beyond the geospatial applications, as far as RDF and array processing are combined. With respect to spatio-temporal gridded data processing, we have shown that even though WCPS and SciSPARQL have slightly different array models, it is easy to formulate WCPS requests in terms of SciSPARQL queries. This provides flexibility and potential efficiency benefits to WCPS users, and opens geospatial data to the Semantic Web users. We designed the ontology to represent the common Grid Coverages data and metadata as RDF with Arrays, so that it can be extended with any additional metadata. As an ongoing work, we are completing and testing our integrated solution, and collecting more use cases from the geo-spatioal gridded data domain.

REFERENCES

[1] OPeNDAP. http://www.opendap.org/. Accessed online on October 15, 2015.

[2] RDF Data Cube. http://www.w3.org/TR/vocab-data-cube/. Accessed online on October 15, 2015.

[3] SKOS Vocabulary. http://www.w3.org/2004/02/skos/. Accessed online on October 15, 2015.

[4] Spatial Data on the Web Working Group. https://www.w3.org/2015/spatial/wiki/Main_Page. Accessed online on October 15, 2015.

[5] Statistical Data and Metadata eXchange. http://sdmx.org/. Accessed online on October 15, 2015.

[6] VoiD Vocabulary. http://www.w3.org/TR/void/. Accessed online on October 15, 2015.

[7] A. Andrejev, X. He, and T. Risch. Scientific Data as RDF with Arrays: Tight Integration of SciSPARQL Queries Into MATLAB. In M. Horridge, M. Rospocher, and J. van Ossenbruggen, editors, *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, volume 1272 of *CEUR Workshop Proceedings*, pages 221–224. CEUR-WS.org, 2014.

[8] A. Andrejev and T. Risch. Scientific SPARQL: Semantic Web Queries over Scientific Data. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*, ICDEW '12, pages 5–10. IEEE Computer Society, 2012.

[9] A. Andrejev, S. Toor, A. Hellander, S. Holmgren, and T. Risch. Scientific Analysis by Queries in Extended SPARQL over a Scalable e-Science Data Store. In *IEEE 9th International Conference on eScience*, pages 98–106, Oct 2013.

[10] P. Baumann. Management of Multidimensional Discrete Data. *The VLDB Journal*, 3(4):401–444, Oct. 1994.

[11] P. Baumann. A database array algebra for spatio-temporal data and beyond. In *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems*, NGITS '99, pages 76–93, London, UK, 1999. Springer-Verlag.

[12] P. Baumann. The OGC Web Coverage Processing Service (WCPS) standard. *Geoinformatica*, 14(4):447–479, Oct. 2010.

[13] P. Baumann. GML application schema for coverages. *OGC 09–146r2*, 2012.

[14] P. Baumann. OGC Web Coverage Service (WCS) – Core. *OGC 09–110r4*, 2012.

[15] P. Baumann, P. Campalani, J. Yu, and D. Misev. Finding my CRS: a systematic way of identifying CRSs. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 71–78, New York, NY, USA, 2012. ACM.

[16] P. Baumann, P. Mazzetti, J. Ungar, R. Barbera, D. Barboni, A. Beccati, L. Bigagli, E. Boldrini, R. Bruno, A. Calanducci, P. Campalani, O. Clement, A. Dumitru, M. Grant, P. Herzig, K. Kakaletris, L. Laxton, P. Koltsida, K. Lipskoch, A. Mahdiraji, S. Mantovani, V. Merticariu, A. Messina, D. Misev, S. Natali, S. Nativi, J. Oosthoek, J. Passmore, M. Pappalardo, A. Rossi, F. Rundo, M. Sen, V. Sorbera, D. Sullivan, M. Torrisi, L. Trovato, M. Veratelli, and S. Wagner. Big Data Analytics for Earth Sciences: the EarthServer approach. *International Journal of Digital Earth*, 0(0):1–27, 0.

[17] M. F. Fernández, D. Florescu, S. Boag, J. Robie, D. Chamberlin, and J. Siméon. XQuery 1.0: An XML query language (second edition). W3C proposed edited recommendation, W3C, Apr. 2009. http://www.w3.org/TR/2009/PER-xquery-20090421/.

[18] S. Fiore, A. D'Anca, C. Palazzo, I. Foster, D. Williams, and G. Aloisio. Ophidia: Toward big data analytics for escience. *Procedia Computer Science*, 18(0):2376 – 2385, 2013. 2013 International Conference on Computational Science.

[19] J. Gasperi, F. Houbie, A. Woolf, and S. Smolders. Earth observation metadata profile of observations and measurements. *OGC 10–157r3*, 2012.

[20] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, G. Heber, and D. DeWitt. Scientific Data Management in the Coming Decade. *ACM SIGMOD Record*, 34(4):35–41, January 2005. also as MSR-TR-2005-10.

[21] R. Guha and D. Brickley. RDF schema 1.1. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[22] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[23] J. R. Herring. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture. OpenGIS Implementation Standard 06–103r4, Open Geospatial Consortium Inc, 2011.

[24] ISO 19123:2005: Geographic information – Schema for coverage geometry and functions, 2005.

[25] ISO 19125-1:2004 Geographic information – Simple feature access – Part 1: Common architecture, 2004.

[26] G. Klyne and J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[27] P. Liakos, P. Koltsida, G. Kakaletris, and P. Baumann. xwcps: Bridging the gap between array and semi-structured data. In *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.*, pages 120–123, 2014.

[28] D. Misev and P. Baumann. Extending the sql array concept to support scientific analytics. In *Proc. Intl. Conf. on Scientific and Statistical Database Management (SSDBM'2014)*, page paper no. 10, June 30 - July 2, 2014.

[29] M. Perry and J. Herring. OGC GeoSPARQL – A Geographic Query Language for RDF Data. *OGC 11–052r4*, 2012.

[30] C. Portele. OGC Geography Markup Language (GML) Encoding Standard. *OGC 07–036*, 2007.

[31] Rasdaman. The rasdaman Raster Array Database. http://rasdaman.org. Accessed: 2015-feb-28.

[32] R. Rew, G. Davis, S. Emmerson, H. Davies, E. Hartnett, and D. Heimbigner. The NetCDF Users Guide – Data Model, Programming Interfaces, and Format for Self-Describing, Portable Data - NetCDF Version 4.1, March 2010.

[33] A. Robin. OGC SWE Common Data Model Encoding Standard. *OGC 08–094r1*, 2011.

[34] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The Architecture of SciDB. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, SSDBM'11, pages 1–16, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] E. L. Usery and D. Varanka. Design and Development of Linked Data from The National Map. *Semantic Web – On linked spatiotemporal data and geo-ontologies*, 3(4):371–384, Oct. 2012.

[36] Y. Zhang, M. L. Kersten, M. Ivanova, and N. Nes. SciQL, Bridging the Gap between Science and Relational DBMS. In B. C. Desai, I. F. Cruz, and J. Bernardino, editors, *IDEAS*, pages 124–133. ACM, 2011.