GeoSPARQL+: Syntax, Semantics and System for Integrated Querying of Graph, Raster and Vector Data

Timo Homburg[1(✉)], Steffen Staab[3,4], and Daniel Janke[2]

[1] Mainz University of Applied Sciences, DE, Mainz, Germany
timo.homburg@hs-mainz.de
[2] Universität Koblenz, DE, Mainz, Germany
danijank@uni-koblenz.de
[3] Universität Stuttgart, DE, Stuttgart, Germany
steffen.staab@ipvs.uni-stuttgart.de
[4] WAIS Research Group, University of Southampton, Southampton, UK

Abstract. We introduce an approach to semantically represent and query raster data in a Semantic Web graph. We extend the GeoSPARQL vocabulary and query language to support raster data as a new type of geospatial data. We define new filter functions and illustrate our approach using several use cases on real-world data sets. Finally, we describe a prototypical implementation and validate the feasibility of our approach.

Keywords: GeoSPARQL · Raster data · Geospatial semantics

1 Introduction

The Geospatial Semantic Web [9,16] has grown in size and importance in the last decade. It is estimated that about 80% of all data has a geospatial relation [19]. Therefore, GeoSPARQL [6] has been developed and became an OGC[1] and W3C[2] recommendation allowing for the representation and querying of geospatial data in the semantic web. GeoSPARQL and comparable approaches [22,24] only provide support for geospatial vector data. However, geospatial data may also take the shape of a raster. It may, e.g., be obtained from aerial imagery or from simulation data to support tasks such as city planning and risk assessment as shown by the examples depicted in Fig. 1.

Raster data must not be represented as vector geometries, because vector representations of raster data

1. are inefficient implying overconsumption of data storage. Raster data can be large and may be compressed efficiently.
2. are ineffective representations as they lack operations needed to query raster data e.g. raster algebra operations that transform raster data in ways not applicable to vector data.

[1] https://www.opengeospatial.org.
[2] https://www.w3.org.

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12506, pp. 258–275, 2020.
https://doi.org/10.1007/978-3-030-62419-4_15

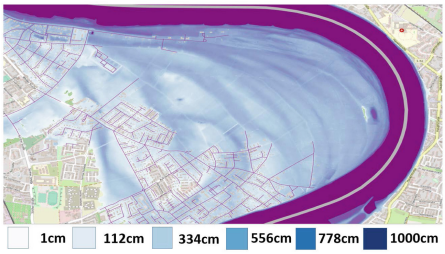
3. lack the semantics needed to appropriately represent raster data. Raster data is often visualized with RGB values, such as varying shades of blue for different flood altitudes. A semantic representation, however, should not represent color shades, but rather the underlying semantics, which should refer to data from the actual nominal, ordinal, interval or ratio scales and what they stand for.

We propose GeoSPARQL+, an extension of the GeoSPARQL query language, and the GeoSPARQL+ ontology in order to integrate geospatial raster data into the Semantic Web.

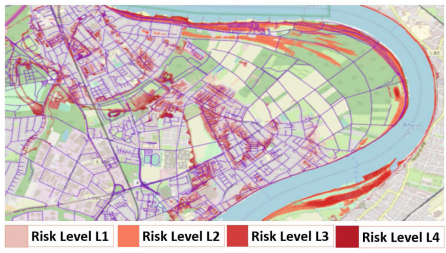
Let us consider the analysis of a flood as our running example. Our running example is depicted in Fig. 1a showing the overlay of two related datasets:

1. Vector data representing the roads of Cologne
2. Raster data representing the altitudes of a simulated flood

A query in one of our real-world use cases asks for all the road sections not covered by more than 10cm of water. This is only possible if the data model can represent raster data, vector data, semantics (road, water, depth, 10cm) and allows for joint querying of these representations. Existing geographical information systems lack the explicit representation of semantics and require the user to manually adapt his high-level information need into a query of the low-level representation. The GeoSPARQL standard [6] and systems that currently support geographic information in the Semantic Web [6, 10, 14, 15, 22, 24, 26] do not represent raster data, thus, they do not allow for asking such questions.



(a) Floodmap depicting the flood altitude and a road network. The map legend informally describes the semantics of colors in terms of a fractional scale of flood altitudes.



(b) Fire hazard risks displayed in different shades of red with darker shades implying higher risk levels. The map legend informally describes the risks using an ordinal scale.

Fig. 1. Visualizations of two sources of risk in Cologne

In the remainder of this paper, we will assume that there are data sources that contain vector data (e.g. roads in Fig. 1) and raster data (e.g. flood altitudes Fig. 1a or fire hazards Fig. 1b). We describe a GeoSPARQL+ ontology which allows a data engineer to integrate these data into an RDF graph. A user may issue a semantic query against the RDF graph using GeoSPARQL+. To allow for these capabilities, this paper makes the following contributions:

1. *Semantic Representation of Raster Data:* A data model that allows for representing raster data and its semantics (Sect. 4).
2. *GeoSPARQL Vocabulary Extension:* This vocabulary extension defines how to relate the raster data to semantic descriptions (Sect. 5.1).
3. *GeoSPARQL Query Language Extension:* A SPARQL extension which allows the interoperable use of semantic graph data, semantic vector geometries, and semantic raster data and uses map algebra [35] to combine and modify rasters (Sects. 5.2 and 5.3).
4. *Prototypical Implementation:* An open source implementation of the proposed approach for geospatial vector and raster data (Sect. 6).
5. *Requirements and Feasibility Check:* Deriving requirements of GeoSPARQL+ by discussing relevant use cases (Sect. 3), assessing their feasibility and conducting a performance check of the implemented system (Sect. 7).

The tasks of data integration and visualization of query results are beyond the focus of this paper. More technical details about the supported functions may be found in our companion technical report [18].

2 Foundations for Extending GeoSPARQL

In this publication we limit ourselves to 2D representations in order to remain concise. We see no major issue in extending our approach to higher dimensional representations. We assume that all geographical representations relate to coordinate reference systems (CRS), as postulated in [9]. For conciseness of illustration we discard these relations and transformations between CRSs.

2.1 Geometry

We formally define several OGC Simple Feature geometries [17], which we use in the remainder of this paper.

Definition 1 (*Geometry*)**.** *A geometry* $g \in \text{Geo}$*, with* Geo *representing the set of all geometries, is an instantiation of one of the following data structures:*

1. *A geometry g may be a* Point $p = (x, y), p \in \mathbb{R}^2$*, or*
2. *A* LineString *defined as a list of at least two different points denoted as* $g = (p_0, p_1, \ldots, p_n), p_i \in \mathbb{R}^2$*, or*
3. *A* Polygon *g represented as a LineString with* $g = (p_0, p_1, \ldots, p_n)$*,* $p_0 = p_n, p_i \in \mathbb{R}^2$ *and all other points being unique. We further restrict ourselves to valid Polygons. In valid Polygons lines do not cross each other. A Polygon includes the encompassed area.*
4. *A geometry g may also be a* Rectangle*, which is a special polygon comprised of four LineStrings with the angles between connected LineStrings being* 90°. $\text{Rect} \subset \text{Geo}$ *is the set of all rectangles.*
5. *Finally, a geometry may be a* GeometryCollection *g, which itself is a finite set of geometries* $g = \{g_1, \ldots, g_k\}, g_i \in \text{Geo}$.

MultiPolygons *and* MultiLineStrings *are examples of GeometryCollections.*

We assume that the function geom2pset : $Geo \rightarrow 2^{\mathbb{R}^2}$ exists which converts a geometry to a PointSet representation.

2.2 RDF, SPARQL and GeoSPARQL

In order to semantically describe and query raster data we build upon the following standard definitions of SPARQL 1.1 [12,27]. We provide the first formal definitions of the operators and filter functions which GeoSPARQL [6] adds to the SPARQL query language and describe the resulting modified definitions of SPARQL 1.1 in the following. In order to keep the definitions concise enough for this paper, we formalize syntax and semantics with 3 exemplary operators and 2 exemplary filter functions. We pick GeoSPARQL specific elements such that they are representative for the most common types of signatures. The differences between SPARQL 1.1 and the GeoSPARQL extensions are marked in blue fonts.

Definition 2 (*RDF Triple and RDF Graph*)**.** *Let I, B and L be disjoint sets of IRIs, blank nodes and literals, respectively. An element of the set $(I \cup B) \times I \times (I \cup B \cup L)$ is called a* triple $t \in T$ *with T denoting the set of all triples. $G \in 2^{(I \cup B) \times I \times (I \cup B \cup L)}$ is called an* RDF graph. *$GL \subset L$ is the set of all geometry literals.*

In an RDF triple (s, p, o), s, p and o are called *subject*, *predicate* and *object*, respectively. Geometry literals (GL) are serialized according to the GeoSPARQL standard either as Well-Known-Text (WKT) [36] literals or as Geography Markup Language (GML) [29] literals.

Definition 3 (*Triple Pattern*)**.** *Let V be a set of variables that is disjoint to I, B and L. An element of $(I \cup B \cup L \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$ is called a* triple pattern.

The set of variables occurring in a triple pattern tp is abbreviated as $\text{var}(tp)$.

Definition 4 (*Expression*)**.** *An* expression *is*

$Expression ::=$ `?X`	*with* `?X` $\in V$
$\mid c$	*with constant* $c \in L \cup I$.
$\mid E_1 \cap E_2$	*with E_1, E_2 being expressions.*
$\mid$ geof : buffer(E_1, E_2, E_3)	*with E_1, E_2, E_3 being expressions*
$\mid$ geof : distance(E_1, E_2)	*with E_1, E_2 being expressions*

Definition 5 (*Filter Condition*)**.** *A* filter condition *is*

$FilterCondition ::=$ `?X = c`	*with* `?X` $\in V$ *and* $c \in I \cup L$
$\mid$ `?X = ?Y`	*with* `?X`, `?Y` $\in V$
$\mid \neg F$	*with filter condition F*
$\mid F_1 \vee F_2$	*with filter conditions F_1 and F_2*
$\mid F_1 \wedge F_2$	*with filter conditions F_1 and F_2*
$\mid E_1 (=) E_2$	*with E_1, E_2 being expressions*
$\mid E_1 (\cap) E_2$	*with E_1, E_2 being expressions*

$\cap$, $\overset{=}{\cap}$, $\overset{\cap}{\cap}$ correspond to the GeoSPARQL operators geof : intersection, geof : equals and geof : intersects respectively [11]. We provide complete list of all GeoSPARQL functions in our technical report that extends this paper [18].

Definition 6 (*Basic Graph Pattern*)**.** *A* basic graph pattern *(BGP) is*

$BGP ::=$ tp	*a triple pattern* tp
$\mid \{B\}$	*a block of a basic graph pattern B*
$\mid B_1.B_2$	*a conjunction of two basic graph patterns* B_1 *and* B_2
$\mid B$ `FILTER` F	*a filter pattern with BGP B and filter condition F*
$\mid B$ `BIND` $(E$ `AS ?X`$)$	*a bind with BGP B, expression E and variable* `?X`.

Definition 7 (*Select Query*)**.** *A* select query *is defined as* `SELECT` W `WHERE` B *with* $W \subseteq V$ *and basic graph pattern B.*

Definition 8 (*Variable Binding*)**.** *A* variable binding *is a partial function* $\mu : V \nrightarrow I \cup B \cup L$. *The set of all variable bindings is* Φ.

The abbreviated notation $\mu(tp)$ means that variables in triple pattern *tp* are substituted according to μ.

Definition 9 (*Compatible Variable Binding*)**.** *Two variable bindings* μ_1 *and* μ_2 *are* compatible*, denoted by* $\mu_1 \sim \mu_2$*, if*

$$\forall \texttt{?X} \in \text{dom}(\mu_1) \cup \text{dom}(\mu_2) : \mu_1(\texttt{?X}) = \mu_2(\texttt{?X})$$

Thereby $\text{dom}(\mu)$ *refers to the set of variables of variable binding* μ.

Definition 10 (*Join*)**.** *The* join *of two sets of variable bindings* Φ_1, Φ_2 *is defined as*

$$\Phi_1 \bowtie \Phi_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Phi_1 \wedge \mu_2 \in \Phi_2 \wedge \mu_1 \sim \mu_2\}$$

Definition 11 (*Expression Evaluation*)**.** *The* evaluation *of an expression E over a variable binding* μ*, denoted by* $[\![E]\!]_\mu$*, is defined recursively as follows:*

$[\![\texttt{?X}]\!]_\mu := \mu(\texttt{?X})$	*with* $\texttt{?X} \in V$.
$[\![c]\!]_\mu := c$	*with c being a constant, literal or IRI.*
$[\![E_1 \cap E_2]\!]_\mu := [\![E_1]\!]_\mu \cap [\![E_2]\!]_\mu$	*retrieves a geometry* $g \in$ Geo *that represents all Points in the intersection of* $[\![E_1]\!]_\mu, [\![E_2]\!]_\mu \in$ Geo *[6]*
$[\![\text{geof} : \text{buffer}(E_1, E_2, E_3)]\!]_\mu := g$	*retrieves a bounding box* $g \in$ Rect *of radius* $[\![E_2]\!]_\mu \in \mathbb{R}$ *around* $[\![E_1]\!]_\mu \in$ Geo *using the unit given in* $[\![E_3]\!]_\mu \in I$ *[6]*
$[\![\text{geof} : \text{distance}(E_1, E_2)]\!]_\mu := c$	*returns the minimum distance* $c \in \mathbb{R}$ *between* $[\![E_1]\!]_\mu \in$ Geo *and* $[\![E_2]\!]_\mu \in$ Geo *[6]*

Definition 12 (*Filter Condition Satisfaction*)**.** *Whether variable binding* μ satisfies a filter condition *F, denoted by* $\mu \models F$*, is defined recursively as follows:*

$\mu \models ?X = c$	*holds if* $?X \in \text{dom}(\mu)$ *and* $\mu(?X) = c$.
$\mu \models ?X = ?Y$	*holds if* $?X \in \text{dom}(\mu)$, $?Y \in \text{dom}(\mu)$ *and* $\mu(?X) = \mu(?Y)$.
$\mu \models \neg F$	*holds if it is not the case that* $\mu \models F$.
$\mu \models F_1 \vee F_2$	*holds if* $\mu \models F_1$ *or* $\mu \models F_2$.
$\mu \models F_1 \wedge F_2$	*holds if* $\mu \models F_1$ *and* $\mu \models F_2$
$\mu \models E_1 \textcircled{=} E_2$	*holds if* $[\![E_1]\!]_\mu \in$ Geo, $[\![E_2]\!]_\mu \in$ Geo *and* $\text{geom2pset}([\![E_1]\!]_\mu) = \text{geom2pset}([\![E_2]\!]_\mu)$
$\mu \models E_1 \textcircled{\cap} E_2$	*holds if* $[\![E_1]\!]_\mu \in$ Geo, $[\![E_2]\!]_\mu \in$ Geo *and* $\text{geom2pset}([\![E_1]\!]_\mu) \cap \text{geom2pset}([\![E_2]\!]_\mu) \neq \varnothing$.

Definition 13 (*SPARQL Evaluation*)**.** *The* evaluation *of a SPARQL query Q over an RDF graph G, denoted by* $[\![Q]\!]_G$*, is defined recursively as follows:*

$[\![\text{tp}]\!]_G := \{\mu \mid \text{dom}(\mu) = \text{var}(\text{tp}) \wedge \mu(\text{tp}) \in G\}$	*with triple pattern* tp.
$[\![\{B\}]\!]_G := [\![B]\!]_G$	*with basic graph pattern B.*
$[\![B_1.B_2]\!]_G := [\![B_1]\!]_G \bowtie [\![B_2]\!]_G$	*with basic graph patterns* B_1 *and* B_2.
$[\![B \texttt{ FILTER } F]\!]_G := \{\mu \mid \mu \in [\![B]\!]_G \wedge \mu \models F\}$	*with basic graph pattern B and filter condition F.*
$[\![B \texttt{ BIND } (E \texttt{ AS } ?X)]\!]_G :=$ $\{\mu \cup \{?X \mapsto [\![E]\!]_\mu\} \mid \mu \in [\![B]\!]_G \wedge ?X \notin \text{dom}(\mu)\}$	*with basic graph pattern B, expression E and variable* $?X$.
$[\![\texttt{SELECT } W \texttt{ WHERE } B]\!]_G := \{\mu_{\mid W} \mid \mu \in [\![B]\!]_G\}$	*with basic graph pattern B and* $W \subseteq V$

Thereby $\mu_{\mid W}$ *means that the domain of* μ *is restricted to the variables in W.*

3 Use Case Requirements

We now define requirements for use cases we have encountered when collaborating with companies developing geographical information systems.

U1 *Client: Rescue Forces; Use case: Emergency rescue routing*
Rescue vehicles and routing algorithms guiding them need to know which roads are passable in case of flooding.
Example query: *"Give me all roads which are not flooded by more than 10 cm"*

U2 *Client: Insurance; Use case: Risk assessment*
Insurances evaluate the hazard risk for str'eets and buildings in order to calculate the insurance premium.
Example query: *"Assess the combined risk of fire and flood hazards for all buildings in the knowledge base"*

U3 *Client: Disaster Management Agency; Use case: Rescue capacity planning*
In case of disasters, the number of people present at a specified time and place needs to be estimated to prepare hospitals for casualties.
' Example query: *"Give me the roads which contain elements at risk which are open to the public at 23rd May 2019 10.20am"* Note: An *element at risk* is a term in disaster management describing a class of buildings affected by certain disasters [18].

U4 *Client: City Planning Agency; Use case: Rescue facility location planning*
Rescue forces should be stationed in a way that they can react fast to possible hazards and city planners should position rescue stations accordingly.
Example query: *"Give me the percentage of served hazardous areas, i.e. areas within a bounding box of 10km around a to-be-built rescue station at a given geo-coordinate"*

These example queries can currently not be expressed using GeoSPARQL. Abstracting from the given natural language query examples we have defined a graph data model for raster data and the syntax and semantics of GeoSPARQL+ that allow us to respond to these queries.

4 Modeling Raster Data

We have analyzed the requirements for representing raster data using examples like the ones depicted in Fig. 1 and use cases in Sect. 3. These examples show that we need to transform the following visual elements into semantic representations:

1. Raster geometry: A raster covers a geometrical area. In this paper, we limit ourselves to rectangular areas though other geometries might be supported in the future.
2. Atomic values: In visualizations of raster data, atomic values are mapped onto pixel values. In simple cases this is a one-to-one mapping. Depending on the resolution of the raster and the rendered picture, it may also be a n:1 or 1:n or even an n:m mapping.
3. Atomic value geometry: Each atomic value represents the situation in a geometry area, typically in a rectangular area.
4. Raster legend: A raster legend is a description of the semantic interpretation of the raster's atomic values. This description includes a categorical, ordinal, interval or fractional scale.

We formally define a raster following [20] as:

Definition 14 (*Raster*)**.** *Let R be the set of all rasters and* $\mathbb{S}$ *the set of all scales. A* Raster $r \in R$ *is a partial function* $r : \mathbb{R}^2 \nrightarrow S$ *which maps positions onto a scale* $S \in \mathbb{S}$*. We define a scale as a partially ordered set. In addition, every scale defines a NODATA value, a unique value which is not to be used elsewhere in the scale definition. The domain of a raster* $dom(r)$ *is the closed, rectangular region represented by its raster geometry for which its atomic values are defined.*

dom(r) can be represented by a rectangle defined by its four corners (p_l,p_b,p_r,p_t), where $p_i = (x_i, y_i)$ and $x_l \leq x_r, x_l \leq x_t, x_l \leq x_b, x_r \geq x_t, x_r \geq x_b$ and $y_l \geq y_b, y_t \geq y_r, y_l \leq y_t, y_b \leq y_r$.

Figure 2 shows an example of a raster. In order to execute geometric operations on raster data and geometries we assume a function raster2geom(r) returning dom(r) as a geometric object. In order to compare rasters to other rasters we assume an equality function. rastervaleq(r,r) compares the rasters atomic values and its domains.

Definition 15 (*Raster Literal*). *The set* RL $\subset L$ *with* GL $\cap$ RL $= \emptyset$ *represents the set of all raster literals.*

We use the CoverageJSON format [8] to serialize rasters to raster literals, but many other textual serializations or even binary serializations are possible. These representations assume that the raster geometry is divided uniformly into rectangular cell geometries (atomic value geometries in our previous definition). A cell c is a pair $(g,s) \in \text{Rect} \times S$. We relate a cell c to a raster r via a pair of indexes (i,j). $r_{i,j}$ refers to a specific cell indexed by (i,j) in a raster r. $r_{i,j}(x,y)$ is undefined for values outside of the cell and has the identical value for all positions within the cell. Thus, given x,y such that $r_{i,j}(x,y)$ is defined, c may be defined as $(raster2geom(r_{i,j}), r_{i,j}(x,y))$.

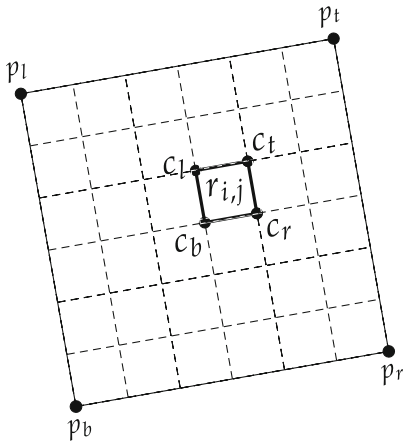


Fig. 2. Raster representation: The raster r is represented using a raster geometry $dom(r)$, a subdivision in cells and a scale $S \in \mathbb{S}$.

The function cellval : $R \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ retrieves the atomic value of a given raster cell. The function cellval2 : $R \to \{\mathbb{R}\}$ retrieves atomic values of all raster cells.

Raster Algebra or map algebra is a set based algebra to manipulate raster data. Following [35] we assume the definition of scale-dependent raster algebras with operations $\ominus$, $\oplus$ and $\oslash$ defined for the following signatures:

$$(1)\ \ominus : R \to R,\ (2)\ \oplus : R \times R \to R.\ (3)\ \oslash : R \times \mathbb{R} \to R$$

The three operations we indicate here, their formal definitions given in [35], are examples for a broader set of possible raster algebra operations. Most other algebraic operators exhibit the same signatures as one of these three example operations. Hence, syntax and semantics of other operators can be integrated into GeoSPARQL+ taking the integration of example operators as templates.

The $\ominus$ function converts each atomic value different from 0 to 0, all 0 values to 1 and does not change NODATA values. The $\oplus$ function creates a new raster with the domain of the first raster. The resulting raster contains all values of the first raster which have no correspondence with the atomic values of the second raster (i.e. not map to the same position). All values with a correspondence are added together or ignored if one of the input values is the NODATA value of either of the two rasters. This function can be used

to combine risks of fire and flood hazards given in two different rasters representing the same area.

The $\lessdot$ function takes one raster and one constant. It returns a new raster with the domain of the given raster. Atomic values smaller than the given constant are kept, all other values become the NODATA value. One application of this function is to only keep the flood altitude values displayed in Fig. 1a which signify an altitude value smaller than a given constant.

Implementations like PostGIS [31] and JAI [32] provide 26 and 108 raster functions respectively. Out of those we have implemented 14 in our system which we describe in [18].

5 GeoSPARQL+

In order to describe raster data semantically,we must define (i) their geometries, (ii) their atomic values, (iii) the atomic value geometries, and (iv) the semantic meaning of raster's atomic values. The latter is specified in this section. When the raster's contents have been described, new functions are needed to filter, relate or modify the raster's atomic values in order to be useful in the application cases we would like to solve. Therefore we extend the GeoSPARQL query language to include such functions in Sects. 5.2 and 5.3

5.1 The GeoSPARQL+ Vocabulary

We define the new GeoSPARQL+ vocabulary (cf. Fig. 3).

A raster is described by its semantic class (*geo2:Raster*), and a scale which describes the semantic content of its atomic values. In Fig. 3, we depict the example of a semantic class *ex:FloodArea* which is assigned an instance of *geo2:Raster* with a CoverageJSON literal (Listing 1.1) including the raster's type, a CRS, the raster's atomic values and their description. In order to re-use the representations of the CoverageJSON format, we model rasters in a concept hierarchy of OGC coverage types. By the OGC definition, a raster is a special type of coverage which is rectangular, i.e. a grid, and is georeferenced. This definition is reflected in Fig. 3 in the given concept hierarchy. The instance of *geo:Raster* connects to an instance of *om:Scale* describing its legend and unit of measurement derived from the units of measurements ontology (UOM) as well as the scales NODATA value.

```
{"type" : "Coverage","domain" : { "type" : "Domain", "domainType" : "Grid",
"axes": { "x" : { "values": [−10,−5,0] },"y" : { "values": [40,50] }
"referencing": [{"coordinates": ["y","x"],"system": {
"type": "GeographicCRS","id": "http://www.opengis.net/def/crs/EPSG/0/4979"}}]},
"observedProperty" : {
"ranges" : { "FloodAT" : { "type" : "NdArray", "dataType": "float",
"axisNames": ["y","x"], "shape": [2, 2], "values" : [ 0.5, 0.6, 0.4, 0.6 ]}}}
```

Listing 1.1. Coverage JSON Literal example

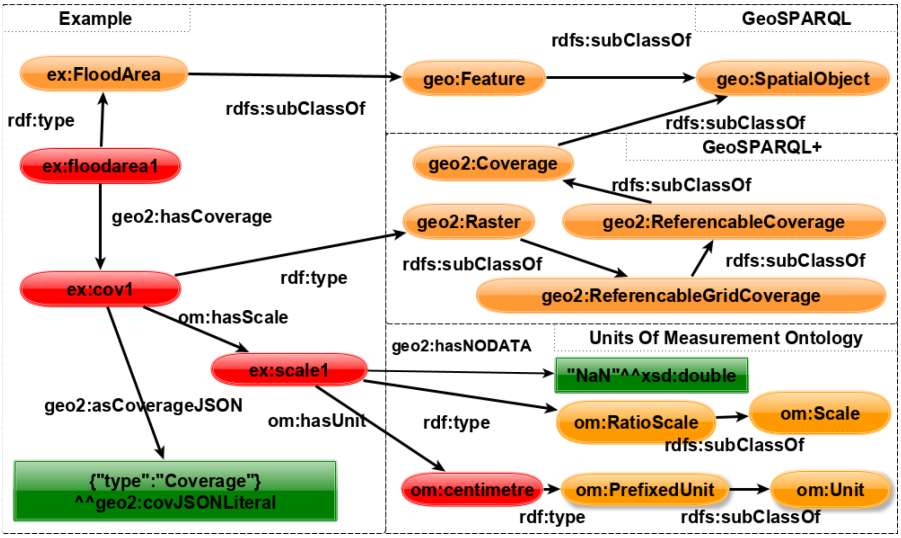


Fig. 3. We use vocabularies of three different ontologies: The GeoSPARQL ontology describes the concepts *geo:SpatialObject* and *geo:Feature*, the OGC coverage hierarchy describes the abstract concepts of coverages and the unit of measurement vocabulary describes legends of raster data.

5.2 GeoSPARQL+ Syntax

We added several new operators to the GeoSPARQL+ query language that allow to filter, modify and combine rasters as well as polygons. Due to space limitations, we present only one example for each of the three possibilites. A full list of the implemented functions is provided in [18]. geometryIntersection calculates intersections between arbitrary combinations of Geometries and Rasters, returning a Geometry. To get a raster as result instead, the rasterIntersection can be used. $\boxed{+}$ and $\boxed{<}$ provide two examples of raster algebra expressions.
GeoSPARQL+ defines the following new expressions to replace Definition 4:

Definition 16 (*GeoSPARQL+ Expression*)**.**

Expression ::= ?X	*with* $?X \in V$
| c	*with constant* $c \in L \cup I$.
| geometryIntersection(E_1, E_2)	*with* E_1, E_2 *being expressions*
| rasterIntersection(E_1, E_2)	*with* E_1, E_2 *being expressions*
| $E_1 \boxed{+} E_2$	*with* E_1, E_2 *being expressions*
| $E_1 \boxed{<} E_2$	*with* E_1, E_2 *being expressions*
| $\boxed{\neg} E$	*with E being an expression*
| raster2geom(E)	*with E being an expression*
| rastervaleq(E_1, E_2)	*with* E_1, E_2 *being expressions*
| geom2raster(E_1, E_2)	*with* E_1, E_2 *being expressions*

GeoSPARQL+ does not introduce new filter conditions in comparison to GeoSPARQL. However, the semantics of the previously defined filter conditions $\textcircled{=}$ and $\textcircled{\cap}$ are extended to also include raster literals.

5.3 GeoSPARQL+ Semantics

We define the semantics of a GeoSPARQL+ expression in Definition 17. In order to specify the intersection we map geometries and rasters to the corresponding PointSets. The result is a Geometry or Raster based on the selection of the user. In the special case of the intersection of two geometries, when a raster should be returned, we require a default value represented by parameter E_3 to which the atomic values of the created raster are mapped. The raster algebra functions geo2 : rasterPlus and geo2 : rasterSmaller are mapped to their respective raster algebra expression defined in Sect. 4.

GeoSPARQL+ adds the following evaluations of expressions to Definition 11:

Definition 17 (*GeoSPARQL+ Expression Evaluation*)**.**

$$
\begin{aligned}
[\![\text{geometryIntersection}(E_1,E_2)]\!]_\mu &:= [\![E_1]\!]_\mu \cap [\![E_2]\!]_\mu \\
&\quad \textit{if } [\![E_1]\!]_\mu \textit{ and } [\![E_2]\!]_\mu \in \textit{Geo} \\
[\![\text{geometryIntersection}(E_1,E_2)]\!]_\mu &:= [\![E_1]\!]_\mu \cap \text{raster2geom}([\![E_2]\!]_\mu) \\
&\quad \textit{if } [\![E_1]\!]_\mu \in \textit{Geo and } [\![E_2]\!]_\mu \in R \\
[\![\text{geometryIntersection}(E_1,E_2)]\!]_\mu &:= [\![\text{geometryIntersection}(E_2,E_1)]\!]_\mu \\
&\quad \textit{if } [\![E_1]\!]_\mu \in R \textit{ and } [\![E_2]\!]_\mu \in \text{Geo} \\
[\![\text{rasterIntersection}(E_1,E_2)]\!]_\mu &:= r \in R \textit{ with } \forall i,j : r_{i,j} = r1_{i,j} \\
&\quad \textit{if } [\![E_1]\!]_\mu = r1, [\![E_2]\!]_\mu = r2 \in R \\
&\quad \textit{and } \text{dom}(r1_{i,j}) \cap \text{dom}(r2_{i,j}) \neq \emptyset \\
[\![\text{rasterIntersection}(E_1,E_2)]\!]_\mu &:= r \in R \textit{ with } \forall i,j : r_{i,j} = r1_{i,j} \\
&\quad \textit{if } [\![E_1]\!]_\mu = r1 \in R \textit{ and } [\![E_2]\!]_\mu = g \in \text{Geo} \\
&\quad \textit{and } \text{dom}(r1_{i,j}) \cap g \neq \emptyset \\
[\![\text{rasterIntersection}(E_1,E_2)]\!]_\mu &:= [\![\text{rasterIntersection}(E_2,E_1)]\!]_\mu \\
&\quad \textit{if } [\![E_1]\!]_\mu = r1 \in R \textit{ and } [\![E_2]\!]_\mu = g \in \text{Geo} \\
[\![\text{rastervaleq}(E_1,E_2)]\!]_\mu &:= r \in R \quad \textit{with } \forall i,j : \text{dom}(r1_{i,j}) \cap \text{dom}(r2_{i,j}) \neq \emptyset \\
&\quad \textit{and } \text{cellval}(r1_{i,j}) == \text{cellval}(r2_{i,j}) \\
&\quad \textit{if } [\![E_1]\!]_\mu = r1, [\![E_2]\!]_\mu = r2 \in R \\
[\![\boxed{\neg} E]\!]_\mu &:= r \in R \quad \textit{with } \forall i,j : r_{i,j} = \textcircled{\neg} r1_{i,j} \textit{ if } [\![E]\!]_\mu = r1 \in R \\
[\![E_1 \boxed{+} E_2]\!]_\mu &:= [\![E_1]\!]_\mu \textcircled{+} [\![E_2]\!]_\mu \quad \textit{if } [\![E_1]\!]_\mu,\ [\![E_2]\!]_\mu \in R \\
[\![E_1 \boxed{<} E_2]\!]_\mu &:= [\![E_1]\!]_\mu \textcircled{<} [\![E_2]\!]_\mu \quad \textit{if } [\![E_1]\!]_\mu, [\![E_2]\!]_\mu \in R \\
[\![\text{geom2raster}(E_1,E_2,E_3,E_4)]\!]_\mu &:= r \in R \textit{ with} \\
&\quad \forall (x,y) \in \text{geof} : \text{buffer}([\![E_1]\!]_\mu, 1, \text{uom} : \text{meter}) \\
&\quad r(x,y) = [\![E_2]\!]_\mu \\
&\quad \textit{if } [\![E_1]\!]_\mu \in \textit{Geo}, [\![E_2]\!]_\mu, [\![E_3]\!]_\mu, [\![E_4]\!]_\mu \in \mathbb{R} \\
&\quad \textit{with } [\![E_3]\!]_\mu \cdot [\![E_4]\!]_\mu \textit{ indicating the number of cells}
\end{aligned}
$$

We define the semantics of a GeoSPARQL+ filter condition in Definition 18. The geo2:equals method returns true if two Raster or two Geometries are identical. The

geo2 : intersects method returns true if the PointSets of two Raster or Geometries overlap. GeoSPARQL+ replaces the evaluation of the filter condition from Definition 12 as follows:

Definition 18 (*GeoSPARQL+ Filter Condition Satisfaction*)**.**

$\mu \models E_1 \textcircled{=} E_2$ *holds if* $[\![E_1]\!]_\mu, [\![E_2]\!]_\mu \in$ Geo *and* $\text{geom2pset}([\![E_1]\!]_\mu) = \text{geom2pset}([\![E_2]\!]_\mu)$.

$\mu \models E_1 \textcircled{=} E_2$ *holds if* $[\![E_1]\!]_\mu \in R$ *and* $[\![E_2]\!]_\mu \in$ Geo *and* $\text{geom2pset}(\text{raster2geom}([\![E_1]\!]_\mu)) = \text{geom2pset}([\![E_2]\!]_\mu)$

$\mu \models E_1 \textcircled{=} E_2$ *holds if* $[\![E_1]\!]_\mu \in$ Geo *and* $[\![E_2]\!]_\mu \in R$ *and* $\mu \models E_2 \textcircled{=} E_1$

$\mu \models E_1 \textcircled{=} E_2$ *holds if* $[\![E_1]\!]_\mu, [\![E_2]\!]_\mu \in R$ *and* $\text{geom2pset}(\text{raster2geom}([\![E_1]\!]_\mu)) = \text{geom2pset}(\text{raster2geom}([\![E_2]\!]_\mu))$

$\mu \models E_1 \textcircled{\cap} E_2$ *holds if* $[\![E_1]\!]_\mu \in R$, $[\![E_2]\!]_\mu \in R$ *and* $\text{geom2pset}(\text{raster2geom}([\![E_1]\!]_\mu)) \cap \text{geom2pset}(\text{raster2geom}([\![E_2]\!]_\mu)) \neq \varnothing$

$\mu \models E_1 \textcircled{\cap} E_2$ *holds if* $[\![E_1]\!]_\mu \in$ Geo, $[\![E_2]\!]_\mu \in R$ *and* $\text{geom2pset}([\![E_1]\!]_\mu) \cap \text{geom2pset}(\text{raster2geom}([\![E_2]\!]_\mu)) \neq \varnothing$

$\mu \models E_1 \textcircled{\cap} E_2$ *holds if* $[\![E_1]\!]_\mu \in R$, $[\![E_2]\!]_\mu \in$ Geo *and* $\mu \models E_2 \textcircled{\cap} E_1$

Further Functions. We have provided a couple of example functions and their signatures in order to show the principles of working with raster data. In practice, one needs a much larger set of functions and signatures. In particular the signatures *geo:area*: *Geo* $\rightarrow \mathbb{R}$, *geo2:max*: $R \rightarrow \mathbb{R}$ are used. *geo:area* is a GeoSPARQL function calculating the area of a Geometry, *geo2:max* calculates the maximum atomic value of a raster. We also use the additional raster algebra functions *geo2:isGreater*: $Rx\mathbb{R} \rightarrow \mathbb{R}$ and *geo2:rasterUnion* $RxR \rightarrow \mathbb{R}$. The first one returns a raster which only includes atomic values greater than a given constant and the second one is the complement of the *geo2:rasterIntersection* function.

6 Implementation

The implementation[3] is built on Apache Jena [22] and geosparql-jena [3] and extends the ARQ query processor of Apache Jena with the GeoSPARQL+ functions defined in Sect. 5. ARQ registers functions in an internal function registry which maps URIs to function implementations. The implementations were done in Java and used the Java Topology Suite library to implement vector geometry related functions, Apache SIS[4] to represent rasters in Java and the Java Advanced Imaging Library (JAI) [21] to implement raster algebra operations. In addition, new literal types needed to be implemented in ARQ. geosparql-jena already provides support for vector literals (WKT and GML). To represent rasters we implemented CoverageJSON and Well-Known-Binary (WKB)

[3] https://github.com/i3mainz/jena-geo.
[4] http://sis.apache.org.

literals with appropriate parsers for (de)serialization. In addition we implemented further functions defined in the SQL/MM standard [34]. These functions help to prepare/-modify vector geometries before they are compared or combined with rasters. Finally, we combined our implementation to work with a Apache Jena Fuseki triple store used for the feasibility study in Sect. 7.

7 Feasibility

We work with the following datasets:

1. A vector dataset (GeoJSON): Road network of Cologne from OpenStreetMap
2. A vector dataset (GeoJSON) of elements at risk extracted from OpenStreetMap
3. Two rasters (flood altitude and fire hazards) of Cologne provided by a company simulating hazards

The RDF graph contains the classes *ex:Road*, classes for elements at risk and the classes *ex:FloodRiskArea*, *ex:FireRiskArea* for the rasters described in Sect. 5.

7.1 GeoSPARQL+ Queries

The feasibility check includes the four use cases defined in Sect. 3 and defines two queries per application case in GeoSPARQL+ and an equivalent query in SQL/MM [34]. The GeoSPARQL+ query is executed on our prototypical implementation, the second query is executed on a POSTGIS implementation. For brevity we only illustrate the GeoSPARQL+ queries in Listings 1.2 to 1.5.

The first query (Listing 1.2) solves usecase U1 and uses the raster algebra function *geo:rasterSmaller* ($\boxed{<}$) (line 5) to filter those parts of a flood raster where roads that are still passable.

```
1  SELECT ?road WHERE {
   ?road a ex:Road ; geo:hasGeometry ?roadseg . ?roadseg geo:asWKT ?roadseg_wkt .
3  ?floodarea a ex:FloodRiskArea ; geo2:asCoverage ?floodarea_cov .
   ?floodarea_cov geo2:asCoverageJSON ?floodarea_covjson .
5  BIND(geo2:rasterSmaller(?floodarea_covjson,10) AS ?relfloodarea)
   FILTER(geo2:intersects(?roadseg_wkt,?relfloodarea))}
```

Listing 1.2. Use Case 1: Flood Altitude

The second query (Listing 1.3) solving use case U2 adds the values of two different rasters (fire and floodhazard) of the same area together (*geo2:rasterPlus* ($\boxed{+}$) line 8) and extracts atomic values of the combined raster to assign a risk value to each given building. The maximum risk value per building is returned.

```
1  SELECT ?building (MAX(?riskvalue) AS ?riskmax) WHERE {
   ?building a ex:Building ; geo:hasGeometry ?building_geom .
3  ?building_geom geo:asWKT ?building_wkt .
   ?floodarea a ex:FloodRiskArea ; geo2:hasCoverage ?floodcov.
5  ?floodcov geo2:asCoverageJSON ?floodcov_covjson .
   ?firearea rdf:type ex:FireRiskArea ; geo2:hasCoverage ?firecov.
7  ?firecov geo2:asCoverageJSON ?firecov_covjson .
   BIND (geo2:rasterPlus(?firecov_covjson,?floodcov_covjson) AS ?riskarea)
9  BIND (geo2:cellval2(geo2:rasterIntersection(?building_wkt,?riskarea)) AS ?riskvalue)
   FILTER(geo2:intersects(?building_wkt,?riskarea))}
```

Listing 1.3. Use case 2: Risk assessment

The third query (Listing 1.4) solving use case U3 combines the assessment of properties of vector geometries (line 10) with assessments gained from rasters (line 7) and GeoSPARQL functions like geo:buffer and geo:intersects (line 11–12) to evaluate roads with a higher priority to be evacuated.

```
1  SELECT ?road WHERE{
   ?road a ex:Road ; geo:hasGeometry ?roadgeom . ?roadgeom geo:asWKT ?road_wkt .
3  ?ear a ear:ElementAtRisk ; geo:hasGeometry ?eargeom ; ex:openTime ?earopen ; ex:closeTime ?earclose .
   ?eargeom geo:asWKT ?ear_wkt .
5  ?floodarea a ex:FloodRiskArea ; geo2:hasCoverage ?floodcov. ?floodcov geo2:asCoverageJSON ?floodcov_covjson
       .
   ?firearea rdf:type ex:FireRiskArea ; geo2:hasCoverage ?firecov. ?firecov geo2:asCoverageJSON ?firecov_covjson .
7  BIND (geo2:rasterPlus(?firecov_covjson,?floodcov_covjson) AS ?riskarea)
   BIND("2019−05−23T10:20:13+05:30"^^xsd:dateTime AS ?givendate)
9  FILTER(?givendate>?earopen AND ?givendate<?earclose)
   FILTER(geo:intersects(geo:buffer(?road_wkt,2,uom:meter),?ear))
11 FILTER(!geo:intersects(?road_wkt,?riskarea))}
```

Listing 1.4. Use case 3: Rescue Capacity Planning

Roads with a higher priority are near elements at risk for which we provide an ontology model in the appended technical report. The element at risk definition simplifies this query in comparison to an equivalent POSTGIS query, as the semantics are already explicitly stated.

Finally, the query for use case U4 (Listing 1.5) combines the GeoSPARQL functions *geo:area* (line 8) and *geo:buffer* (line 7) with GeoSPARQL+ functions to intersect geometries and rasters (line 7–8) and to return a rasters geometry (line 8).

```
   SELECT ?hazardcoveragepercentage WHERE {
2  ?floodarea a ex:FloodRiskArea; geo2:hasCoverage ?floodcov.
   ?floodcov geo2:asCoverageJSON ?floodcov_covjson .
4  ?firearea rdf:type ex:FireRiskArea ; geo2:hasCoverage ?firecov.
   ?firecov geo2:asCoverageJSON ?firecov_covjson .
6  BIND(geo2:rasterUnion(?firecov_covjson,?floodcov_covjson) AS ?hazardriskarea)
   BIND(geo2:geometryIntersection(?hazardriskarea,geo:buffer(?locationtocheck,10,uom:km)) AS ?intersectarea)
8  BIND(geo:area(?intersectarea)/geo2:raster2geom(?hazardriskarea) AS ?hazardcoveragepercentage)
   BIND("POINT(49.2,36.2)"^^geo:wktLiteral AS ?locationtocheck)}
```

Listing 1.5. Use case 4: City Planning

7.2 Results

We measured the execution times of the introduced GeoSPARQL+ queries in comparison to equivalent SQL/MM [36] queries run on a POSTGIS implementation. The results are shown in Table 1.
Table 1 shows that the execution time of our prototype is significantly longer than that of the native POSTGIS implementation.

7.3 Discussion

In Sect. 5 have shown that the query solutions for use cases U1-U4 exploit different elements of GeoSPARQL+. Use case U1 relates a raster to a vector data set, use case U2 showcases the need of raster algebra operators to solve questions of combined risks, use

Table 1. Execution times of the given queries in the GeoSPARQL+ prototype vs. the comparison implementation in POSTGIS.

Use case	GeoSPARQL+	POSTGIS
Use case 1	112,423 ms	86,817 ms
Use case 2	164,865 ms	108,357 ms
Use case 3	134,865 ms	112,817 ms
Use case 4	184,865 ms	140,357 ms

case U3 combines values gained from rasters with attributes gained from vector data at the same geographic location. Both use case U2 and U3 make use of raster-aware filter functions. Finally, the query to solve use case U4 utilizes the raster to geometry function to create intersections between rasters with certain characteristics. We therefore illustrated the usefulness of GeoSPARQL+. Our prototypical implementation exhibits a slight performance decay between 23% and 34% for various example queries. We speculate that this degradation comes from overhead of dealing with semantics, lack of geospatial indices for rasters and further caches as well as a lack of technical optimizations that POSTGIS as a mature well-used system comes with. Considering that our implementation merely constitutes a proof of concepts, we consider this a graceful degradation and an acceptable result. Future work may consider an improvement of its performance.

8 Related Work

[23] and [28] proposed stSPARQL and SPARQL-ST, which extend SPARQL with spatiotemporal query capabilities for vector data. Spatiotemporal aspects for raster data and vector data are not considered by our approach but we see no major issues to combine the ideas of stSPARQL with our work. This is relevant as not only rasters with spatiotemporal aspects exist, but the content of raster data may also change over time.

Some approaches like LinkedGeoData [5] convert SPARQL queries to SQL queries in order to execute them on a native geospatial-aware SQL database. Similarly, hybrid systems such as Virtuoso [15] add a semantic layer on top of a relational database such as POSTGIS [31]. In principle, this would allow for accessing raster data, but has only been used to store and distribute vector data (cf. [5]). We attribute this to a lack of semantic description of raster data which we address in this publication. Furthermore, we provide a solution independent of SQL datatabases and independent of the need for query conversions from SPARQL to SQL.

Relational spatial databases like POSTGIS [31] or OGC geospatial webservices [25] along with software suites such as QGIS[5] and their accompanying libraries can handle, import, modify and query raster data, in particular with raster algebra. None of the aforementioned systems combines the advantages of linked data with the ability to semantically describe or access raster data information.

[5] https://qgis.org/de/site/.

In addition to the previously mentioned work, there is a line of work that represents raster data as linked data ([13,30,33]). These works do not consider how to query raster data. Hence, they lack the expressiveness required to cover our use cases. Similarly, [7] wrap raster data from a POSTGIS database and make it available as vector data that can be queried with GeoSPARQL. Because GeoSPARQL has no means for asking raster-specific queries (e.g. raster algebra), this work also lacks the expressiveness that our approach provides.

Another line of work includes representing and querying multi-dimensional arrays, SciSPARQL [4]. While there is an overlap between managing raster data and arrays, raster data has geometric aspects that our approach supports (e.g. raster cell geometries, intersections and conversions between rasters and polygons, semantic descriptions of scales) that are not available when the underlying data model is restricted to arrays of real numbers. Hence, [4] can not support our use cases, e.g. lacking intersecting street data and flooding data as we illustrate in Fig. 1a.

9 Conclusion

We presented GeoSPARQL+ a novel approach that allows for the semantic description and querying of raster data in the semantic web. We expect these new capabilities to make publishing geospatial data in the geospatial semantic web more attractive and consider contributing this work to the currently discussed revision of GeoSPARQL [1, 2]. Future work could explore the semantic description of further OGC coverage types such as trajectories or even point clouds. Also, non-grid-based raster types should be investigated, as well as the representation of 3D rasters.

Acknowledgements. Work by Steffen Staab was partially supported by DFG through the project LA 2672/1, Language-integrated Semantic Querying (LISeQ).

References

1. Abhayaratna, J., et al.: OGC benefits of representing spatial data using semantic and graph technologies (2020). https://github.com/opengeospatial/geosemantics-dwg/raw/master/white_paper/wp.pdf
2. Abhayaratna, J., van den Brink, L., Car, N., Homburg, T., Knibbe, F.: OGC GeoSPARQL SWG charter (2020). https://github.com/opengeospatial/geosemantics-dwg/tree/master/geosparql_2.0_swg_charter
3. Albiston, G.L., Osman, T., Chen, H.: GeoSPARQL-Jena: Implementation and benchmarking of a GeoSPARQL graphstore. Semant. Web J. (2019)
4. Andrejev, A., Misev, D., Baumann, P., Risch, T.: Spatio-temporal gridded data processing on the semantic web. In: 2015 IEEE International Conference on Data Science and Data Intensive Systems, pp. 38–45. IEEE (2015)
5. Auer, S., Lehmann, J., Hellmann, S.: LinkedGeoData: adding a spatial dimension to the web of data. In: Bernstein, A., et al. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 731–746. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_46
6. Battle, R., Kolas, D.: Enabling the geospatial semantic web with parliament and GeoSPARQL. Semant. Web **3**(4), 355–370 (2012)

7. Bereta, K., Stamoulis, G., Koubarakis, M.: Ontology-based data access and visualization of big vector and raster data. In: 2018 IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2018, pp. 407–410. IEEE (2018)
8. Blower, J., Riechert, M., Roberts, B.: Overview of the CoverageJSON format (2017)
9. Van den Brink, L., Barnaghi, P., et al.: Best practices for publishing, retrieving, and using spatial data on the web. Semant. Web **10**(1), 95–114 (2019)
10. Cerans, K., Barzdins, G., et al.: Graphical schema editing for StarDog OWL/RDF databases using OWLGrEd/S In: OWLED, vol. 849 (2012)
11. Consortium, O.G., et al.: OGC GeoSPARQL-a geographic query language for RDF data. OGC Candidate Implementation Standard (2012)
12. World Wide Web Consortium: Sparql 1.1 overview (2013)
13. World Wide Web Consortium: The RDF data cube vocabulary (2014)
14. Eclipse Foundation Contributor: Rdf4j (2020). rdf4j.org
15. Erling, O.: Virtuoso, a hybrid RDBMS/graph store. IEEE Data Eng. **35**(1), 3–8 (2012)
16. Fonseca, F.: Geospatial semantic web. In: Shekhar, S., Xiong, H. (eds.) Encyclopedia of GIS, pp. 388–391. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-35973-1_513
17. Herring, J., et al.: Opengis® implementation standard for geographic information-simple feature access-part 1: Common architecture [corrigendum] (2011)
18. Homburg, T., Staab, S., Janke, D.: GeoSPARQL+: syntax, semantics and system for integrated querying of graph, raster and vector data. extended version. technical report (2020) (at arXiv.org). Technical report, Mainz University of Applied Sciences (2020)
19. Huxhold, W.E., et al.: An introduction to urban geographic information systems. OUP Catalogue (1991)
20. ISO 19123:2005: Geographic information–schema for coverage geometry and functions. The International Organization for Standardization, Geneva, Switzerland (2005)
21. Jaiswal, D., Dey, S., Dasgupta, R., Mukherjee, A.: Spatial query handling in semantic web application: an experience report. In: 2015 Applications and Innovations in Mobile Computing (AIMoC), pp. 170–175. IEEE (2015)
22. Jena, A.: A free and open source java framework for building semantic web and linked data applications (2019)
23. Koubarakis, M., Kyzirakos, K.: Modeling and querying metadata in the semantic sensor web: the model stRDF and the query language stSPARQL. In: Aroyo, L., et al. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 425–439. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13486-9_29
24. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: a Semantic Geospatial DBMS. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7649. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35176-1_19
25. Nogueras-Iso, J., Zarazaga-Soria, F.J., Béjar, R., Álvarez, P., Muro-Medrano, P.R.: OGC catalog services: a key element for the development of spatial data infrastructures. Comput. Geosci. **31**(2), 199–209 (2005)
26. Ontotext: Graphdb (2020). graphdb.ontotext.com
27. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. In: Cruz, I., et al. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_3
28. Perry, M., Jain, P., Sheth, A.P.: SPARQL-ST: extending SPARQL to support spatiotemporal queries. In: Ashish, N., Sheth, A. (eds.) Geospatial Semantics and the Semantic Web. ADSW, vol. 12, pp. 61–86. Springer, Boston (2011). https://doi.org/10.1007/978-1-4419-9446-2_3
29. Portele, C.: OpenGIS® geography markup language (GML) encoding standard. Open Geospatial Consortium (2007)

30. Quintero, R., Torres, M., Moreno, M., Guzmán, G.: Towards a semantic representation of raster spatial data. In: Janowicz, K., Raubal, M., Levashkin, S. (eds.) GeoS 2009. LNCS, vol. 5892, pp. 63–82. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10436-7_5
31. Ramsey, P., et al.: PostGIS Manual, p. 17. Refractions Research Inc. (2005)
32. Santos, R.: Java advanced imaging API: a tutorial. Revista de Informática Teórica e Aplicada **11**(1), 93–124 (2004)
33. Scharrenbach, T., Bischof, S., Fleischli, S., Weibel, R.: Linked raster data. In: Xiao, N., Kwan, M.P., Goodchild, M.F., Shekhar, S. (eds.) Geographic Information Science. LNCS, vol. 7478. Springer, Heidelberg (2012)
34. Stolze, K.: SQL/MM spatial: The standard to manage spatial data in a relational database system. In: BTW 2003-Datenbanksysteme für Business, Technologie und Web, Tagungsband der 10. BTW Konferenz. Gesellschaft für Informatik eV (2003)
35. Tomlin, C.D.: Map algebra: one perspective. Landsc. Urban Plan. **30**, 3–12 (1994)
36. Wirz, D.: OGC Simple Features (for SQL and XML/GML). University of Zurich, Department Geography, Zurich (2004)