Web Semantics: Science, Services and Agents on the World Wide Web 58 (2019) 100514

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Ontop-spatial: Ontop of geospatial databases

Konstantina Bereta [a], Guohui Xiao [b,*], Manolis Koubarakis [a]

[a] Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, Athens 15784, Greece
[b] KRDB Research Centre, Faculty of Computer Science, Free-University of Bozen-Bolzano, Bolzano, 39100, Italy

ARTICLE INFO

Article history:
Received 13 October 2018
Received in revised form 18 May 2019
Accepted 19 June 2019
Available online 1 July 2019

Keywords:
Spatial data
GeoSPARQL
Ontology-based data access
Ontop-spatial

ABSTRACT

In this paper, we propose an OBDA approach for accessing geospatial data stored in relational databases using the R2RML mappings and OGC standard GeoSPARQL. On the theoretical side, we provide a formalization of GeoSPARQL in terms of SPARQL entailment regime. For a practical query answering algorithm, we introduce an extension to the existing SPARQL-to-SQL translation method to support GeoSPARQL features. Our approach has been implemented in the system Ontop-spatial, an extension of the OBDA system Ontop for creating virtual geospatial RDF graphs on top of geospatial relational databases. We present an experimental evaluation of our system using and extending a state-of-the-art benchmark. To measure the performance of our system, we compare it to two state-of-the-art geospatial RDF stores –a free and open-source one, and a commercial one– and confirm its efficiency.

1. Introduction

Currently, there is an emerging interest of scientific communities from various domains that produce and process geospatial data (e.g., earth scientists) to publish this data as linked data and combine it with other data sources. Responding to this trend, the Semantic Web community has been very active in the geospatial domain, proposing data models, query languages, and systems for the representation and management of geospatial data. Notably, this research has led to the development of extensions of RDF and SPARQL, such as stRDF/stSPARQL [1] and GeoSPARQL [2], that handle geospatial data. Similarly, research on geospatial relational databases has been going on for a long time and has resulted in the implementation of several efficient geospatial DBMS.

Ontology-based data access (OBDA) [3,4] is a popular paradigm for providing a convenient and user-friendly access to data repositories. In OBDA, an OWL ontology describes the domain of interest, which is connected to a data source through a declarative R2RML mapping specification. Then the underlying data source is *virtualized* as an RDF graph using the vocabulary from the ontology, and the SPARQL queries over the ontologies are automatically rewritten by an OBDA engine into SQL queries expressed over the underlying database.

Despite the extensive research performed in the fields of relational databases and the Semantic Web on the development of solutions for handling geospatial data efficiently, to the best of our knowledge, there is no OBDA system that enables the creation of virtual, geospatial RDF graphs on top of geospatial databases. This would be very useful for scientists that produce and process geospatial data, as they mainly store this data in relational geospatial databases (e.g., PostGIS) or in other geospatial data formats that are easily imported into such databases (e.g., shapefiles). With the existing solutions in place, these scientists are forced to materialize their data as RDF in order to publish it as linked data and/or use it in combination with other data sources. However, this is often not practical and discourages users from using Semantic Web technologies. This issue applies to the OBDA paradigm in general, but it has more impact in the geospatial domain due to the reasons we have just described. We address these issues by extending the OBDA paradigm with geospatial support.

The contributions of this paper are the following:

- On the theoretical side, we provide a formalization of the OGC GeoSPARQL standard in terms of SPARQL entailment regime.
- For a practical query answering algorithm, we introduce an extension to the existing SPARQL-to-SQL translation method in order to support GeoSPARQL queries.
- We describe the implementation of our approach in the system Ontop-spatial, which to the best of our knowledge is the first OBDA system for GeoSPARQL.
- We present an experimental evaluation of our system by extending the benchmark Geographica [5], comparing the performance of Ontop-spatial with the state-of-the-art geospatial RDF store Strabon [6] and the free version of a state-of-the-art commercial triple store with GeoSPARQL support. Due to license reasons, in the context of this paper we will

* Corresponding author.
E-mail addresses: Konstantina.Bereta@di.uoa.gr (K. Bereta), xiao@inf.unibz.it (G. Xiao), koubarak@di.uoa.gr (M. Koubarakis).

https://doi.org/10.1016/j.websem.2019.100514
1570-8268/

refer to the commercial system using the alias "System-O". The results show that, in most cases, Ontop-spatial outperforms both of them.

Ontop-spatial is available as free and open source software at the following link: https://github.com/ConstantB/ontop-spatial.

The organization of the rest of the paper is as follows. In Section 2 we present background work. In Section 3 we provide a formalization of the GeoSPARQL standard. In Section 4 we explain the GeoSPARQL-to-SQL translation. In Section 5 we present the system Ontop-spatial and mention the real-world use cases in which it has been used. In Section 6 we present the experimental evaluation of our system. In Section 7 we survey the related work. Finally, in Section 8 we conclude the presentation of our approach discussing its advantages and limitations, as well as its possible extensions.

Delta from Previous Publications. This submission is a significant extension of the previous publication [7]. On the theoretical side, we have formalized the semantics of GeoSPARQL queries and the GeoSPARQL-to-SQL translation. On the practical side, we have studied how to access raster data and provided additional experiments comparing Ontop-spatial with the free version of a commercial state-of-the-art triple store with geospatial support.

2. Preliminaries

In this section, we recall the basic notions needed for the rest of the paper.

2.1. RDF And SPARQL

We consider a vocabulary of three pairwise disjoint and countably infinite sets of symbols: **I** for *IRIs*, **L** for *RDF literals*, and **V** for *variables*. In line with previous work on ontology-based data access, we do not consider blank nodes. Intuitively, an IRI represents an object, and a literal represents a typed value. A literal ℓ is of the form $value\hat{}\hat{}type$ where $value$ is the *lexical value* of the ℓ, and *type* is the *type* IRI of ℓ. The supported types defined in the RDF 1.1 Concepts document [8] is largely based on the XML Schema Definition Language (XSD) [9]. An RDF term is an element in $\mathbf{T} = \mathbf{I} \cup \mathbf{L}$. An (RDF) *triple* is an element in $\mathbf{T} \times \mathbf{I} \times \mathbf{T}$. An *(RDF) graph* is a set of triples.

SPARQL [10] is the W3C standard language designed to query RDF graphs. A *triple pattern* is an element of $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$. A *basic graph pattern* (*BGP*) is a finite set of triple patterns. A filter expression F is a Boolean function, which restricts on solutions over the whole group where the filter appears. We consider the fragment of SPARQL queries defined by Q in the following EBNF grammar[1]:

$$
\begin{aligned}
P &::= B \mid Q \mid P \texttt{ FILTER } F \mid P \texttt{ UNION } P \mid \\
&\quad\ P \texttt{ JOIN } P \mid P \texttt{ OPT } P \\
Q &::= \texttt{SELECT } \{ \mathbf{V} \texttt{ AS } \mathbf{V} \} \texttt{ WHERE } P
\end{aligned}
$$

where B is a BGP and F is a *filter expression* (we refer to [10] for details).

The semantics of SPARQL queries is given in terms of *solution mappings*, which are *partial* maps $s: \mathbf{V} \to \mathbf{T}$ with (possibly empty) domain $dom(s)$. Here, following [11,12], we use the set-based semantics for SPARQL (rather than the bag-based one, as in the W3C specification). More specifically, for a BGP B, the *answer* $[\![B]\!]_G$ to B over a graph G is $[\![B]\!]_G = \{s: var(B) \to \mathbf{T} \mid s(B) \subseteq G\}$, where $var(B)$ is the set of variables occurring in B and $s(B)$ is the result of substituting each variable u in B by $s(u)$. Then, the *answer* to a SPARQL query Q over a graph G is the set $[\![Q]\!]_G$ of solution mappings defined by induction using the SPARQL algebra operators (filter, join, union, optional, and projection) starting from BGPs; cf. [13]. This semantics is known as *simple entailment*.

[1] Recall that in EBNF "{ A }" means any number of repetitions of A.

2.2. SPARQL entailment regimes

SPARQL entailment regimes allow for querying RDF graphs with reasoning capabilities [14]. Specifically, an entailment regime **E** specifies how to obtain from an RDF graph G an entailed graph $eg^{\mathbf{E}}(G)$ [15]. Then, the answer $[\![B]\!]^{\mathbf{E}}_G$ to a BGP B under the entailment regime **E** is defined as $[\![B]\!]_{eg^{\mathbf{E}}(G)}$. Similarly, the answer $[\![Q]\!]^{\mathbf{E}}_G$ to a SPARQL query Q under the entailment regime **E** is defined as $[\![Q]\!]_{eg^{\mathbf{E}}(G)}$. In this way, entailment regimes only modify the evaluation of BGPs but not that of other SPARQL operators.

We present now the standard W3C semantics for SPARQL queries over OWL ontologies. Under the *OWL 2 direct semantics entailment regime*, one can query an RDF graph G that consists of two parts: the *intensional* sub-graph (i.e., TBox or ontology) $\mathcal{T}$ representing the background knowledge in terms of class and property axioms, and an *extensional* sub-graph (i.e., ABox) $\mathcal{A}$ representing the data as class and property assertions. We write such a graph G, which represents a *knowledge base*, as $(\mathcal{T}, \mathcal{A})$ to emphasize the partitioning when necessary. Moreover, for convenience, we use the triple notations $(s, \texttt{rdf:type}, C)$ and (s, p, o) and the ABox assertion notations $C(s)$ and $p(s, o)$ interchangeably. We are particularly interested in the OWL 2 QL profile [16] of OWL 2, which induces the *OWL 2 QL* entailment regime. For an OWL 2 QL knowledge base G we have $eg^{QL}(G) = \{t \mid G \models_{DL} t\}$, where $\models_{DL}$ denotes the standard OWL 2 entailment, defined in terms of description logics semantics, cf. [17].

2.3. Geospatial extensions of RDF and SPARQL

There are several research works on the spatial extensions of the data model RDF and the query language SPARQL. The data model *stRDF* and the query language *stSPARQL* are extensions of RDF and SPARQL 1.1 respectively, developed for the representation and querying of spatial [6] and temporal data (i.e., the valid time of triples [18]). Another framework that has been developed for the representation and querying of geospatial data on the Semantic Web is the OGC standard *GeoSPARQL* [2]. Although GeoSPARQL and stSPARQL were developed independently, they share a lot of features in common. They both adopt the OGC standards Well-known Text (WKT) and Geography Markup Language (GML) for representing geometries. Also both of them extend SPARQL with the topological functions defined in the OGC standard "OpenGIS Simple Feature Access for SQL" [19], and the Egenhofer [20] and the RCC-8 [21] topological relation families. The main difference between stSPARQL and GeoSPARQL is that stSPARQL also provides support spatial updates and spatial aggregates, and offer valid time support.

Since in the rest of the paper we will refer to the notation and concepts defined or followed by stSPARQL and GeoSPARQL, we briefly present them below for the convenience of the reader.

Spatial literal. A spatial literal represents the serialization of a geometry. In stSPARQL, it is a literal of type `strdf:geometry` or its subtypes `strdf:WKT` or `strdf:GML`, as defined in [6]. Similarly, in GeoSPARQL, it is a literal of type `geo:wktLiteral` or `geo:gmlLiteral`.

Spatial term. A spatial term is either a spatial literal or a variable that can be bound to a spatial literal.

Spatial filter. A spatial filter is a Boolean binary function $SF(t_1, t_2)$, where t_1, t_2 are spatial terms and SF is one of the Boolean functions of the Geometry extension of GeoSPARQL, e.g., `geof:sfEquals`.

Spatial selection. A spatial selection in GeoSPARQL/stSPARQL is a SELECT query with a FILTER which is a Boolean binary function with arguments *a variable and a constant*.

Spatial join. A spatial join in these languages is a query with a FILTER which is a Boolean binary function whose *all arguments are*

variables. The definition of the spatial join in SPARQL corresponds to the definition of the spatial join in the geospatial extensions of the relational model. In the rest of this paper, spatial joins will often be denoted as $\bowtie_{sf}$, where *sf* is a spatial filter.

2.4. Ontology-based data access

In ontology-based data access (OBDA) [3], we can view existing relational (geospatial) databases as RDF graphs with the help of mappings and ontologies. Formally, we start from an *OBDA specification* $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, consisting of a set $\mathcal{T}$ of OWL axioms (called the *TBox*), a relational database schema $\mathcal{S}$, and a set $\mathcal{M}$ of *mapping assertions*. An *OBDA instance* $(\mathcal{P}, \mathcal{D})$ is given by an OBDA specification $\mathcal{P}$ and a relational database instance $\mathcal{D}$ compliant with $\mathcal{S}$.

A mapping $\mathcal{M}$ is a declarative specification relating symbols in the ontology (classes and properties) to (SQL) views over the data. The W3C standard RDB2RDF Mapping Language (R2RML) was created with the goal of providing a language for such mappings. The ontology $\mathcal{T}$, together with the mapping $\mathcal{M}$, exposes a high-level conceptual view of the underlying data in terms of a virtual RDF graph, which users can query using the SPARQL query language. In this paper, we extend R2RML mapping with supports of datatypes defined in GeoSPARQL standard.

To ease the presentation, instead of the concrete Turtle serialization of an R2RML mapping, in the following we use an equivalent compact form. A mapping assertion takes the form $m : t \leftarrow sql$, where the source part *sql* is a SQL query and the target part t is an RDF triple template with placeholders inside.

By applying all mapping assertions in $\mathcal{M}$ to $\mathcal{D}$, one can derive a (*virtual*) RDF graph $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ [22]. Then, SPARQL query answering over an OBDA instance $(\mathcal{P}, \mathcal{D})$ is defined as query answering over $(\mathcal{T}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$.

The derived RDF graph can be materialized as RDF triples, or alternatively it can be kept virtual, in which case the user queries are translated by the OBDA system into queries over the data sources. In the virtual approach, one avoids the cost of materialization, and one can rely on the maturity of relational database systems for efficient query answering, with support for security, robust transactions, etc. Among the state-of-the-art systems supporting the virtual OBDA approach we mention Ontop [23], D2RQ,[2] Morph [24], Mastro [25], and Stardog.[3]

Example 1. Consider the table `crc` shown in Fig. 1 and the following mapping assertion.

```
clc:{gid} rdf:type clc:CorineLandCoverArea;
        geo:hasGeometry clc:geometry/{gid} .
clc:geometry/{gid} geo:asWKT {geom}^^geo:wktLiteral.
← SELECT gid, geom FROM clc
```

The source part of the first mapping assertion is a query over the table `clc`. Intuitively, for each row in the table `clc`, it generates three triples with the first two sharing the same subject s = `clc:{gid}`. The first triple declares that the type of s is an IRI `clc:CorineLandCoverArea` and the second triple declares that the geometry of s is an IRI o = `clc:geometry/{gid}`. The third triple declares that the WKT serialization of o is the literal `{geom}^^geo:wktLiteral`. Below is the resulting virtual RDF graph that is populated with information of the first row of table `clc`.

```
clc:20440 rdf:type clc:CorineLandCoverArea .
clc:20440 geo:hasGeometry clc:geometry/20440 .
clc:geometry/20440 geo:asWKT
            "POLYGON (...)"^^geo:wktLiteral .
```

[2] http://d2rq.org/.

[3] http://stardog.com/.

Consider another mapping assertion:

```
gag:{gid} rdf:type gag:AdministrativeDivision;
                hasGeometry gag:geometry/{gid} .
gag:geometry/{gid} geo:asWKT
                    {geom_4326}^^geo:wktLiteral.
← SELECT gid,ST_Transform(geom,'4326') AS geom_4326
   FROM gag
```

The source part is a query over the table `gag`, which contains information about administrative divisions. In a similar way to the above, for each row in the table `gag`, it firstly generates two triples sharing the same subject s = `gag:{gid}`. The first triple declares that the type of s is an IRI `gag:AdministrativeDivision` and the second triple declares that the geometry of s is an IRI o = `gag:geometry/{gid}`. The third triple declares that the WKT serialization of o the literal `{geom}^^geo:wktLiteral`. Notably, the source query contains a row SQL function in the select clause named `ST_Transform`. This function is widely used in the area of GIS and it transforms a geometry from its original Coordinate Reference System (CRS) to another. The geometries original shapefile which is imported to our database are expressed using the CRS 2100. Instead of transforming and materializing the geometries into the WGS84 (the universal CRS), we incorporated this function in the mappings to showcase the flexibility of our approach: Users can incorporate geospatial data manipulation functions in the source part of the mappings, so that the virtual semantic views that will be constructed on top of their data is an improved version of the original data.

3. A formalization of GeoSPARQL

In this section we present in detail the features of GeoSPARQL and provide a formalization in terms of the SPARQL entailment regime.

3.1. OGC GeoSPARQL standard

In the context of this paper, we will only consider GeoSPARQL (and, as a result, the geospatial part of stSPARQL). The main features of GeoSPARQL are specified in Clauses 6 to 10 of the standard [2] consisting of one core component and four extensions. Specifically, the core component, the topology vocabulary, and the geometry extension defines an OWL ontology[4] (denoted by $\mathcal{T}_{geo}$); the geometry topology extension defines SPARQL functions for spatial selection and spatial filter; and the query write extension defines additional rules for computing spatial relations. In the following, we summarize these features of GeoSPARQL following the structure of the specification.

3.1.1. Core component [2, Clause 6]

This component defines high-level RDFS/OWL classes for spatial objects. The main GeoSPARQL classes are shown in Fig. 2. Classes `Feature` and `Geometry` are subclasses of the general class `SpatialObject` (`SubclassOf` properties are represented using dotted arrows). Features have geometries, and this is expressed using the object property `geo:hasGeometry`.

3.1.2. Topology vocabulary extension [2, Clause 7]

This component defines RDF properties for asserting and querying topological relations (i.e., object properties in OWL) between spatial objects. Specifically, it covers different families of topological relations including *Simple Features Access* (e.g., `geo:sfOverlaps`), *RCC8* (e.g., `geo:rcc8po`), and *Egenhofer* (e.g., `geo:ehOverlap`).

[4] http://www.opengis.net/ont/geosparql

row	gid	code_00	id	area_ha	shape_length	shape_area	geom
1	20440	BroadLeavedForest	EU-1900387	371.56	53772.05	3715691.69	0103000020E...
2	20512	BroadLeavedForest	EU-1900769	948.98	37235.0022	9489833.31	0103000020E...
3	20543	BroadLeavedForest	EU-1900881	47.718	3246.17	477189.24	0103000020E...
4	20797	BroadLeavedForest	EU-1901587	63.21	3453.55	632107.82	0103000020E...
5	20904	BroadLeavedForest	EU-1901816	876.06	45397.04	8760618.99	0103000020E...

Fig. 1. Table `crc` that contains CORINE land cover data.

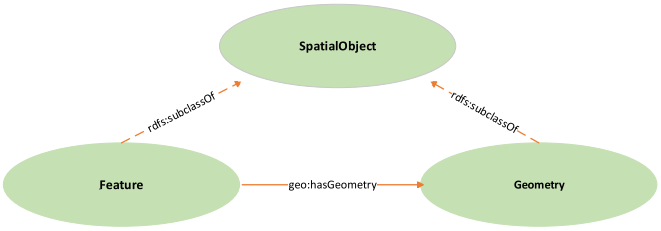


Fig. 2. GeoSPARQL class hierarchy.

3.1.3. Geometry extension [2, Clause 8]

The Geometry extension component defines RDFS data types for serializing geometry data, geometry-related RDF properties, and non-topological spatial query functions for geometry objects. More specifically, this component of GeoSPARQL defines that serializations of geometries are RDF literals, introducing the datatypes `geo:asWKT` and `geo:GML` that correspond to the respective OGC standards WKT and GML that are used to represent geometries as text. It also defines a set of properties that associate features with their geometries, such as the properties `geo:hasGeometry`, `geo:hasSerialization`, `geo:isSimple`, etc. The same component also defines a set of functions that perform non-topological spatial operations, such as the functions `geof:distance` and `geof:intersection`.

3.1.4. Geometry Topology extension [2, Clause 9]

This component defines topological query functions that take two geometry literal and return a boolean value. Topological query functions can be used for spatial selections and spatial joins. The standard defines functions in the families of Simple Features Access, RCC8, and Egenhofer. These functions belong to the namespace `geof` and have the same names as the topological predicates of the Topology extension. For example, `geof:sfOverlaps` is the topological query function corresponding to the relation `geo:sfOverlaps`.

3.1.5. Query rewrite extension [2, Clause 10]

This clause defines query a set of transformation rules, denoted $\mathcal{R}_{geo}$, for computing spatial relations between spatial objects based on their associated geometries. The rules $\mathcal{R}_{geo}$ use the topological extension functions defined in Clause 9 to establish the existence of topological predicates defined in Clause 7.

For example, the rules $R_{overlaps}$ in Fig. 3 specifies how to compute `geo:sfOverlaps` relations of two spatial objects `?f1` and `?f2`. The first feature–feature rule deals with the situation where `?f1` and `?f2` are features, and the overlaps relation can be computed by calling the `geof:sfOverlaps` function over the WKT serializations `?g1WKT` and `?g2WKT` of the geometries `?g1` and `?g2` of the objects `?f1` and `?f2`. We use $\mathcal{R}_{geo}(G)$ to denote the minimal model obtained by applying rules $\mathcal{R}_{geo}$ to a set of facts G.

The rules $\mathcal{R}_{geo}$ are also used for transforming *qualitative* spatial queries into equivalent *quantitative* queries. When using $\mathcal{R}_{geo}$ for query rewriting, it transforms the triple patterns that contain the topological relation (e.g., `ogc:sfOverlaps`) into an equivalent (union) query that describes the same relation using the respective function (e.g., `geof:overlaps`) in the filter clause of the query. Given a SPARQL query q, we denote by $rew_{geo}(q)$ the SPARQL query rewritten by $\mathcal{R}_{geo}$.

Example 2. The following GeoSPARQL query q retrieves all pairs of Corine Land Cover areas ($s1$) and the administrative divisions ($s2$) such that $s1$ overlaps with $s2$.

```
SELECT ?s1 ?s2 WHERE {
    ?s1 a clc:CorineLandCoverArea .
    ?s2 a gag:AdministrativeDivision .
    ?s1 geo:sfOverlaps ?s2.
}
```

The topological relation `overlaps` in q is expressed using the predicate `geo:sfOverlaps`. Query q can be transformed to the following query q' using the equivalent quantitative function in the example transformation rules of $\mathcal{R}_{overlaps}$.

```
SELECT ?s1 ?s2 WHERE {
  ?s1 a clc:CorineLandCoverArea .
  ?s2 a gag:AdministrativeDivision .
  {
    { ?s1 geo:sfOverlaps ?s2.  }
    UNION
    # feature - feature
    { ?s1 geo:hasDefaultGeometry ?g1 .
      ?g1 geo:asWKT ?g1WKT .
      ?s2 geo:hasDefaultGeometry ?g2 .
      ?g2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    UNION
    # feature - geometry
    { ?s1 geo:hasDefaultGeometry ?g1 .
      ?g1 geo:asWKT ?g1WKT .
      ?s2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    # geometry - feature
    UNION
    { ?s1 geo:asWKT ?g1WKT .
      ?s2 geo:hasDefaultGeometry ?g2 .
      ?g2 geo:asWKT ?g2WKT .
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
    UNION
    # geometry - geometry
    { ?s1 geo:asWKT ?g1WKT. ?s2 geo:asWKT ?g2WKT.
      FILTER(goef:sfOverlaps(?g1WKT, ?g2WKT)).
    }
  }
}
```

3.2. GeoSPARQL entailment regime

Now we develop a formal framework capturing the semantics of the family of GeoSPARQL query languages in terms of SPARQL entailment regime [12,15,26]. Here we introduce a more general version of the formal semantics of GeoSPARQL. Given an entailment regime **E**, we can augment it with the GeoSPARQL capabilities, to derive a new entailment regime (geo-**E**). Intuitively, the geo-**E**-entailment regime captures the core component, the

(1) feature – feature rule

```
geo:sfOverlaps(?f1, ?f2) ← geo:hasDefaultGeometry(?f1, ?g1), geo:asWKT(?g1, ?g1WKT),
                           geo:hasDefaultGeometry(?f2, ?g2), geo:asWKT(?g2, ?g2WKT),
                           geof:sfOverlaps(?g1WKT, ?g2WKT).
```

(2) feature — geometry rule

```
geo:sfOverlaps(?f1, ?f2) ← geo:hasDefaultGeometry(?f1, ?g1), geo:asWKT(?g1, ?g1WKT),
                           geo:asWKT(?f2, ?g2WKT),
                           geof:sfOverlaps(?g1WKT, ?g2WKT).
```

(3) geometry – feature rule

```
geo:sfOverlaps(?f1, ?f2) ← geo:asWKT(?f1, ?g1WKT),
                           geo:hasDefaultGeometry(?f2, ?g2), geo:asWKT(?g2, ?g2WKT),
                           geof:sfOverlaps(?g1WKT, ?g2WKT).
```

(4) geometry – geometry rule

```
geo:sfOverlaps(?f1, ?f2) ← geo:asWKT(?f1, ?g1WKT),
                           geo:asWKT(?f2, ?g2WKT),
                           geof:sfOverlaps(?g1WKT, ?g2WKT)
```

Fig. 3. The Rules $\mathcal{R}_{overlaps}$ for computing overlaps relations.

topology vocabulary and the geometry extension by the built-in ontology $\mathcal{T}_{geo}$, the geometry topology extension by corresponding SPARQL functions, and the query rewrite extent by the rules $\mathcal{R}_{geo}$.

Definition 1. Let $\mathcal{T}_{geo}$ be the GeoSPARQL ontology, $(\mathcal{T}, \mathcal{A})$ a DL ontology, **E** an entailment regime, then the geo-**E** entailment is defined as

$$eg^{\text{geo-}\mathbf{E}}(\mathcal{T}, \mathcal{A}) = \mathcal{R}_{geo}(eg^{\mathbf{E}}(\mathcal{T} \cup \mathcal{T}_{geo}, \mathcal{A})).$$

It is not difficult to show that the "query write extension" can be indeed realized by query rewriting.

Proposition 1. *Let $\mathcal{T}_{geo}$ be the GeoSPARQL ontology, $(\mathcal{T}, \mathcal{A})$ a DL ontology, q a BGP, then*

$$[\![q]\!]^{\text{geo-}\mathbf{E}}_{\mathcal{T},\ \mathcal{A}} = [\![rew_{geo}(q)]\!]^{\mathbf{E}}_{\mathcal{T}\cup\mathcal{T}_{geo},\ \mathcal{A}}.$$

We note that OGC GeoSPARQL standard does not explicitly specify which level of reasoning is required. The reasoning capabilities listed in the requirements essentially only require RDFS. Therefore, GeoSPARQL roughly corresponds the geo-RDFS entailment regime. However, the $\mathcal{T}_{geo}$ ontology is actually classified as $\mathcal{SHIF}$, which is much more expressive than RDFS.

Proposition 1 can be readily applied to the OBDA setting. Given an OBDA instance $(\mathcal{P}, \mathcal{D})$ where $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{S})$, the answers of a SPARQL query q under the geo-**E**-entailment regime is $[\![q]\!]^{\text{geo-}\mathbf{E}}_{\mathcal{T},\mathcal{A},\mathcal{M},D}$ and consequently $[\![rew_{geo}(q)]\!]^{\mathbf{E}}_{\mathcal{T}\cup\mathcal{T}_{geo},\ \mathcal{A}}$. When **E** is first-order rewritable (e.g. OWL 2 QL), the geo-**E** entailment regime can be implemented in an OBDA system by modifying the query translation workflow. Details of such techniques are discussed in the next section.

4. GeoSPARQL-to-SQL

In this section, we present the techniques of answering GeoSPARQL queries in OBDA by translating to SQL queries, which is based on the SPARQL-to-SQL algorithm used in Ontop [12,23]. The pseudo code of algorithm is outlined in Fig. 4. As in the classical case, the algorithm takes as inputs a (Geo)SPARQL query, an ontology $\mathcal{T}$, and a mapping $\mathcal{M}$, and returns a SQL query. The algorithm consists of (1) an offline step, which is query-independent and preprocesses the mapping and ontology and generates the so-called saturated mapping or T-mapping, and (2) an online step, which translates the input SPARQL query into an SQL query. We refer the readers to [23] for more details of the workflow. In the following, we discuss the GeoSPARQL specific steps, which are underlined in the pseudo code.

Input: GeoSPARQL query q, Ontology $\mathcal{T}$, Mapping $\mathcal{M}$
Output: An SQL expression

1: // offline phase
2: $\mathcal{T}_{classified}$ = classify($\mathcal{T} \cup \mathcal{T}_{geo}$) ▷ ontology classification
3: $\mathcal{M}_{\mathcal{T}} \leftarrow saturate(\mathcal{M}, \mathcal{T}_{classified})$ ▷ mapping saturation
4: // online phase
5: $Q \leftarrow rew_{geo}(q)$ ▷ GeoSPARQL query write
6: $S \leftarrow$ list of nodes in Q in a bottom-up topological order
7: sql ← empty map from nodes to SQL expressions
8: **for** node $n \in S$ **do**
9: **if** n is triple pattern **then** ▷ translating leaves
10: sql[n] ← replace-Tmap-def(n, $\mathcal{M}_{\mathcal{T}}$)
11: **else** ▷ translating non-leaf nodes
12: **if** n = JOIN(n_1,n_2) **then**
13: sql[n] ← InnerJoin(sql[n_1], sql[n_2])
14: **else if** n = OPTIONAL(n_1, n_2, e) **then**
15: sql[n] ← LeftJoin(sql[n_1], sql[n_2], e)
16: **else if** n = UNION(n_1, n_2) **then**
17: sql[n] ← Union(sql[n_1], sql[n_2])
18: **else if** n = FILTER(n_1, e) **then**
19: sql[n] ← Filter(sql[n_1], e)
20: **else if** n = PROJECT(n_1, p) **then**
21: sql[n] ← Project(sql[n_1], p)
22: **end if**
23: **end if**
24: **end for**
25: **return** sql[S.last()]

Fig. 4. Algorithm of GeoSPARQL-to-SQL translation.

Table 1
GeoSPARQL simple feature functions to SQL functions.

GeoSPARQL function	OGC SFS SQL function
`geof:sfEquals`	`ST_Equals`
`geof:sfDisjoint`	`ST_Disjoint`
`geof:stIntersects`	`ST_Intersects`
`geof:sfTouches`	`ST_Touches`
`geof:sfCrosses`	`ST_Crosses`
`geof:sfWithin`	`ST_Within`
`geof:sfContains`	`ST_Contains`
`geof:sfOverlaps`	`ST_Overlaps`

Ontology classification. At line 2, the algorithm classifies the input ontology $\mathcal{T}$ union with the GeoSPARQL ontology $\mathcal{T}_{geo}$, and construct an explicit hierarchy of classes and properties. By assuming the built-in ontology $\mathcal{T}_{geo}$, the algorithm is able to support the Clauses 6–8 of the GeoSPARQL standard.

Geosparql query-rewrite. At line 5, the algorithm expands the input GeoSPARQL query q using the rules $\mathcal{R}_{geo}$ as in Example 2. We note that the resulting query is of polynomial size of the input query.

Spatial filter expressions. At line 19, the algorithm transforms the SPARQL filters to its SQL equivalences. Now it also translates GeoSPARQL functions to the corresponding functions in the spatial extension of SQL. In Table 1, we provide a list of SPARQL Simple Feature functions defined in GeoSPARQL and their equivalences in SQL functions defined in OpenGIS SQL standard [27].

5. The ontop-spatial system

We implemented the GeoSPARQL-to-SQL translation framework discussed in Section 4 as an extension of the system Ontop with geospatial features focusing on *spatial selections* and *spatial joins*. We chose to extend Ontop instead of systems offering similar functionality because (i) it is open source, robust and extensible, (ii) it offers a wide range of functionalities that are useful for geospatial applications (reasoning, multiple APIs), and (iii) it implements significant SPARQL-to-SQL optimizations, producing queries that can be executed efficiently by the underlying DBMS as reported in [28]. Ontop-spatial is available as free and open source software at the following link: https://github.com/ConstantB/ontop-spatial.

5.1. System overview

An abstract overview of the system as well as a high-level architecture diagram can be seen in Figs. 5(a) and 5(b) respectively. In the following, we highlight the components of Ontop that we have extended as they are placed in the query processing workflow:

- The virtual Ontop repository takes as input an ontology and a mapping file. Mappings can be either R2RML or the Ontop native OBDA mapping language.
- Once Ontop-spatial receives a GeoSPARQL query, the query gets parsed. We modified the Sesame (now known as rdf4j) parser used by Ontop (and the javacc parser that the respective Sesame library uses), in order to extend its syntax to support geospatial operations in the filter clause of the query. Additionally, qualitative geospatial queries, (i.e., queries containing geospatial triple patterns such as `ex:feauture1 geo:overlaps ex:feature2`) are also supported as standard SPARQL triple patterns, and get transformed into their quantitative equivalents (i.e., queries with spatial filters) in the following step.
- Conventionally, the next step in Ontop is to translate the SPARQL query and the R2RML mappings into a Datalog program so that the query can be represented formally and optimized following a series of optimization steps described in detail in [28]. Ontop-spatial inherits these optimizations and extends the SPARQL-to-Datalog translation module. As explained in the previous section, the geospatial filters are transformed into Datalog using distinguished geospatial predicates. The same distinguished geospatial predicates are used in the case of the qualitative geospatial queries as well using the Query Rewrite extension of GeoSPARQL. As a result, both quantitative and qualitative representations of a GeoSPARQL query are transformed into the same SQL query in the following step.
- The optimized version of the Datalog query, as derived from the previous step, gets translated into SQL. Every geospatial Datalog predicate is mapped to the respective geospatial SQL operator, following the syntax of the underlying DBMS. The DBMS adapter has been extended in order to be able to identify geospatial columns in the database of the user. The PostgreSQL adapter of Ontop has been modified to support the PostGIS extension and an adapter of the open source DBMS Spatialite has been added.
- The SQL query gets eventually executed in the underlying DBMS. Currently, the spatially-enabled databases that ontop supports are the geospatial extensions of PostgreSQL and Sqlite, namely PostGIS,[5] Spatialite,[6] and Oracle Spatial[7] respectively. More geospatial databases will be supported in the future.
- After the evaluation of the spatial SQL query in the DBMS, Ontop-spatial gets the results and sends them to the user. If geometries need to be projected, the SQL query that is produced returns the result as WKT. This enables Ontop-spatial to be used as a GeoSPARQL endpoint, that could serve as input endpoint for applications like linked geospatial data visualizers [29] to display the geometries that are returned as a result of a GeoSPARQL query.

Like the default version of Ontop, Ontop-spatial can be used as a web application (using Sesame workbench), as a Sesame library, as a Protege plugin, or it can be executed from the command line. The virtual geospatial graphs created by Ontop can also be materialized, creating an RDF dump, so that it can then be imported in a geospatial RDF store.

5.2. Compliance with GeoSPARQL

In the following, we explain the parts of GeoSPARQL that are supported in Ontop-spatial.

Core component. Ontop-spatial supports SPARQL and the RDFS classes of the GeoSPARQL ontology.

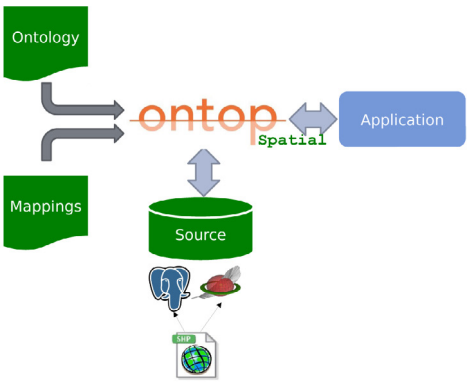
Topology vocabulary extension. Ontop-spatial supports the properties `geo:sfEquals`, `geo:sfDisjoint`, `geo:sfIntersects`, `geo:sfTouches`, `geo:sfCrosses`, `geo:sfWithin`, `geo:sfContains`, `geo:sfOverlaps` and the respective Egenhofer and RCC relations to be used in SPARQL graph patterns.

Geometry extension. Ontop-spatial supports the Geometry classes and properties defined in the Geometry topology component of the GeoSPARQL specification. It also supports the serializations of geometries as literals of the datatypes `geo:wktLiteral` and `geo:gmlLiteral` respectively, as well as the serialization properties `geo:asWKT` and `geo:asGML`. Furthermore, Ontop-spatial supports the non-topological query functions `geof:distance`, `geof:buffer`, `geof:convexHull`, `geof:intersection`, `geof:union`, `geof:difference`, `geof:symDifference`, `geof:envelope` and `geof:boundary` as SPARQL extension functions.

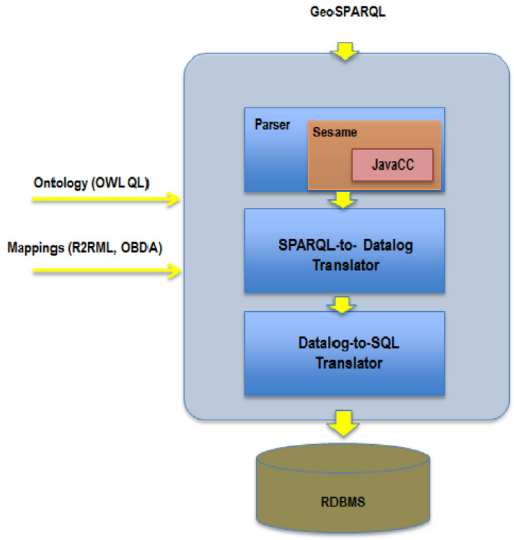
Geometry topology extension. Ontop-spatial supports all of the topological relation functions defined in the Geometry topology extension.

Query rewrite extension. To the best of our knowledge, Ontop-spatial is the first GeoSPARQL implementation that supports this extension of GeoSPARQL.

[5] http://www.postgis.net
[6] http://www.gaia-gis.it/gaia-sins/
[7] https://www.oracle.com/database/spatial/index.html



(a) Abstract overview



(b) Ontop-spatial architecture

Fig. 5. Ontop-spatial.

5.3. Beyond GeoSPARQL: raster data support

In the raster data model, the geospatial data are represented in a different way than in the vector data model. Essentially, they are represented as pixels, with each pixel containing a set of values. This data format is more compact and it is very common in scientific data. For example, a value of a raster cell could indicate a measurement value, such as temperature, moisture level, etc. Due to the popularity of this data model, the geospatial databases incorporated the implementation of adapters for Raster data, introducing specialized data types for representing raster data formats and a set of extension functions for their processing and manipulation, handling them in a similar way to how they handle vector data.

However, none of the geospatial extensions of the framework of RDF and SPARQL, such as stRDF and stSPARQL and GeoSPARQL have considered support for raster data. The main challenge that lies behind this is twofold: First, a raster file is associated with a geometry only as a whole. It is not straight-forward to associate separate raster cells to a geometry, they have to be vectorized first (i.e., translated into polygons). Second, every raster cell is associated with one or more values. In order to convert all information contained in a raster file into RDF, each raster cell should be described by multiple triples, and thus the conversion produces a large amount of triples for the whole raster file. However, not all of this information is needed. In most of the use cases, only the information that derives from a raster file and satisfies certain criteria (e.g., value constraints) is all that is needed to be converted into RDF. This means that the raster file needs to be processed and then the results of this processing are useful as RDF, while any other information is redundant. These challenges have discouraged the scientific community from converting and materializing raster data to RDF. Recently, OGC and W3C have established a working group on Spatial Data on the Web.[8] One of the working notes published by the working group recently is called "Coverages in linked data" and, among other, this discusses these challenges of raster data.

In this paper, we address these challenges by following the OBDA paradigm:

- Ontop-spatial can connect to a geospatial relational database with a raster adapter.
- The raster datatype is internally handled in the same way as its vector counterpart (e.g., the Geometry datatype).
- The following GeoSPARQL operators are overloaded for supporting the respective operations having raster data as arguments in addition to vector data: `ST_Contains`, `ST_Covers`, `ST_Within`, `ST_Overlaps`, `ST_Intersects`, `ST_Touches`.
- PostGIS operators can be added in the mappings in order to process the raster data and create virtual geospatial RDF views above them. For example, certain operators can be used in the SQL query of a mapping in order to refine the results, refining the information from the original raster file that will be virtually translated into RDF. An example is given below.

Example 3. In this example, we combine both vector and raster sources. First, we import a shapefile containing USA second-level administrative divisions to a PostGIS database. The table that contains this information is named `USA_ADM2` and it has the following columns: the `gid` column that stores the id of the respective entries of the shapefile, the `id_0` column that stores an identifier of the administrative division, and the `geom` column that stores the boundaries of the administrative divisions as vector geometries in Well-Known-Binary format (WKB). Then, we import a raster file that is a GeoTIFF image of Chicago. With the raster extension of PostGIS enabled in the database, the raster file is imported into a table named `CHICAGO`, in which the column `rid` is the identifier and the column `rast` of the raster datatype contains the raster cells of the GeoTIFF image. To summarize, the schema of these two tables are: `USA_ADM2(gid, id_0, geom)` and `CHICAGO(rid, rast)`.

The mapping below encodes how data stored in these two tables can be mapped into (virtual) RDF triples.

```
:r/{rid} rdf:type :RasterCell ;
         geo:asWKT {geom}^^geo:WKTLiteral.
← SELECT rid, ST_DumpAsPolygons(rast).geom AS geom
  FROM CHICAGO

:adm/{id_0} rdf:type :AdministrativeDivision ;
:adm/{id_0} geo:hasGeometry :geom/{gid} .
:geom/{gid} geo:asWKT {geom}^^geo:WKTLiteral .
← SELECT gid, id_0, geom FROM USA_ADM2
```

The first mapping assertion shows how the geometries (that are of the raster datatype in the database) are mapped to the WKT format, after they are vectorized, using the PostGIS `ST_DumpAsPolygons` function in the source part. This procedure allows domain experts to use all geometries that they may have in a database uniformly, and execute spatial operations involving

[8] https://www.w3.org/2015/spatial/wiki/Main_Page.

vector and raster geometries, for example. Domain experts usually perform this vectorization step as part of pre-processing. The mapping assertion described above shows how this can be done on-the-fly using Ontop-spatial.

The following GeoSPARQL query involves the combination of vector and raster data sources. Specifically, it retrieves administrative divisions that intersect with raster cells of the GeoTIFF image of Chicago:

```
SELECT ?adm WHERE {
  ?r rdf:type :RasterCell .
  ?r geo:hasGeometry ?rast .
  ?adm rdf:type :AdministrativeDivision .
  ?adm geo:hasGeometry ?g .
  ?g geo:asWKT ?geom .
  FILTER(geof:sfIntersects(?geom,?rast))
}
```

In this query, we retrieve the geometries that correspond to the boundaries of second level administrative divisions whose serializations get bound to the variable `?geom`. We also retrieve the geometries of raster cells that get bound to the variable `?rast`. In Ontop-spatial, the GeoSPARQL function `geof:sfIntersects` is overloaded so that it can evaluate the condition that checks the spatial intersection of vector and raster geometries. As a result, we retrieve the administrative divisions whose boundaries intersect with the GeoTIFF image.

In this way, a user can handle geospatial data sources regardless of their original formats. The query described above is identical to a query that we would pose if only vector data sources were involved. Notably, there is no other system that is able to perform spatial joins between vector and raster geometries in such transparent way, even in the case of specific software solutions that specialize in raster data management like Rasdaman.[9]

5.4. Ontop-spatial in use

The motivation behind the development of Ontop-spatial was the Statoil use case of the project Optique, in order to address the issue of creating virtual RDF graphs on top of large databases that contain geometries and get frequently updated [30]. Ontop-spatial has been used in the urban development, land management and crisis mapping services of the EU FP7 project MELODIES.[10] [31] Finally, Ontop-spatial has recently be used in the Maritime security domain, in collaboration with Airbus [32].

6. Evaluation

We conduct an empirical evaluation of Ontop-spatial based on the philosophy of Geographica,[11] a benchmark for testing the performance of geospatial RDF stores [5]. Geographica consists of a *micro benchmark* and *a macro benchmark*. The *micro benchmark* is designed for testing basic geospatial operators, such as spatial selections and spatial joins. The *macro benchmark* tests the performance of the evaluated systems using queries that correspond to real application scenarios. As our aim is not to test geospatial RDF stores as done in [5], we use a modified benchmark based on the micro benchmark of Geographica as we explain later in this section. The work described in [33] describes an experimental evaluation of triple stores with geospatial functionality for smart cities use cases. Although this benchmark covers a wide variety of smart cities scenarios and queries, only a small subset of spatial operators are used but in a large number of scenarios (e.g., intersection, distance). This could be due to the fact that one of the systems in comparison (Virtuoso) does not provide rich geospatial functionalities yet. The queries are also not designed to stress the geospatial capabilities of the systems in comparison (indexes, spatial optimizations), as the focus of the work is more general, i.e., to perform evaluation of systems in specific use cases. Moreover, [33] does not include the evaluation of Strabon, which, as reported in [5], is the most efficient geospatial RDF store. Hence, we have chosen to use Geographica as the basis of our evaluation.

Since there was no alternative OBDA system that allow for posing GeoSPARQL queries over geospatial relational databases, we decided to evaluate Ontop-spatial in comparison with geospatial RDF stores. We consider that the RDF store Strabon [6] is a good representative of the family of the geospatial RDF stores to compare with as (i) it is a state-of-the-art geospatial RDF store both in terms of functionality and performance [5,6], (ii) it supports a big subset of GeoSPARQL (apart from stSPARQL), and (iii) it uses a spatially-enabled DBMS as back-end, performing a SPARQL-to-SQL translation following a specific storage scheme as explained in [6]. This enables us to use the same DBMS (PostGIS with the same configuration and tuning) and perform a comprehensive comparison. We also consider the commercial system System-X (the free version) which is one of the most efficient triple stores and recently added support for GeoSPARQL. Since System-X is more recent than the study described in [5], we included it in our experimental evaluation to find out how it compares to both Strabon and Ontop-spatial.

6.1. Datasets

Geospatial data come, in most cases, in native geospatial data formats. In a real-world scenario, a user that works with geospatial data obtains it as files in a geospatial data format (e.g., a shapefile) and stores it either in a GIS or a spatially-enabled relational database. Later on, he may convert the data into RDF and store it in a geospatial RDF store in order to combine it with other linked data.

The benchmark Geographica is based on such real-world geospatial application scenarios and for the experimental evaluation of Ontop-spatial we will also follow this approach: We will import real geospatial datasets in a spatially-enabled relational database and use it as the back-end of Ontop-spatial.

We chose to use the datasets of Geographica that are available in their original format (shapefiles). These datasets are the Corine Land Cover dataset of Greece, which is provided by the European Environment Agency (EEA), the Greek Administrative Geometry (GAG), and the Hotspots dataset provided by the National Observatory of Athens. We complemented these data sources with the original raw files of OpenStreetMap data about Greece which are available as shapefiles.[12] Geographica uses the RDF versions of the same subset of the OSM datasets created by the project LinkedGeoData.[13] For the rest of this paper, we will refer to this dataset using the acronym LGD of the resulting RDF version. We added more OSM categories to our workload (e.g., buldings, waterways, etc.), as we will exploit the fact that each one is contained in a different shapefile (so it will be imported into a different table), to stress our system as we explain later on in this section.

For the evaluation of Ontop-spatial, we imported the shapefiles in a PostGIS database using the `shp2pgsql` command as described here: https://github.com/ConstantB/Ontop-spatial/wiki/Shapefiles. In this way, each shapefile is loaded into a separate

9 http://www.rasdaman.com/.

10 http://www.melodiesproject.eu/software-tools.

11 http://geographica.di.uoa.gr/.

12 http://download.geofabrik.de/europe/greece.html.

13 http://linkedgeodata.org/.

table in the database. Each one of these tables contains a column where geometries are stored in binary format (WKB) and an index has been built on that column. Then, we created the minimum set of mappings in order to pose the queries of the benchmark. We used PostgreSQL version 9.1.13 and PostGIS 2.0.3, performing the fine tuning configurations suggested here: http://geographica.di.uoa.gr.

Table 4 shows information about the datasets described above, such as the disk size that each of these tables occupy, the number of tuples and the average number of points per geometry. Notice that the LGD dataset consists of 7 shapefiles/tables which is important in the OBDA setting as we will explain later on. Also, LGD-Places and LGD-Points contain only point geometries.

In order to compare the performance of Ontop-spatial with Strabon and System-X, we materialized the geospatial RDF graph produced by Ontop-spatial and stored it in Strabon and System-X, so that both the virtual RDF graph produced by Ontop-spatial and the graphs stored in other systems contain exactly the same information. The produced RDF dump consists of 5,620,482 triples and contains 855,502 geometries. The total PostGIS database size (in terms of disk usage) of Ontop-spatial is 700 MB. The respective size of the PostGIS database that was produced after loading the RDF dump to Strabon is 1665 MB, which is more than twice the disk space compared to the original database produced by importing the shapefiles directly. The reason is that in the first case the database stores the data, while in the second case the database stores the equivalent set of triples. This kind of overhead is common in RDF stores that use a relational database as backend. Also, Strabon inherits the *per_predicate* storage scheme of the Sesame RDBMS package, so every predicate is stored in a different table and additional tables are used for dictionary encoding. According to this storage scheme, all geometries are stored in a table called *geo_values* in WKB format and the respective column is indexed using an R-tree-over-GiST index, as described in [6].

6.2. Queries

The GeoSPARQL queries that we used for the experimental evaluation of our system are a set of *spatial selections* and a set of *spatial joins*. We used some of the queries of Geographica, and some queries that are appropriate in the OBDA setting as we will explain in the rest of this section. The queries used in our evaluation are presented in Tables 2 and 3. Each query has a numeric identifier, a mnemonic label, a number that shows how many BGPs it consists of and a number that shows how many results it returns.

Both spatial selection and spatial join queries contain a spatial filter that checks if a spatial relation holds between two geometries that are given as arguments to the respective GeoSPARQL function. In the case of spatial selections, one of the arguments is a variable and the other one is a constant, which can be either a line (queries suffixed with "L" in the query label) or a polygon (using "P" suffix). In spatial join queries, both arguments of the respective spatial binary operator are variables. The first set of queries that we consider contains simple geospatial queries, i.e., queries consisting of a single triple pattern to retrieve the geometries of a dataset and a spatial filter (spatial selections 00-14 and spatial joins 00-03). Note that spatial joins require at least two triple patterns to retrieve the geometries that will be bound to the variables that are involved in the spatial filter. This kind of queries test the response time of the compared systems to perform "pure" geospatial queries (i.e., with the minimum amount of non-spatial triple patterns involved, focusing as much as possible on the evaluation of the spatial condition).

The next set of queries that we consider tackles an important issue that is crucial in OBDA systems: the generation of `UNION` operators, deriving from the ontology and the schema of the database in the SPARQL-to-SQL translation phase. For example, the LGD dataset consists of 7 shapefiles, each one containing a column where geometries are stored. In contrast, according to the ontology, the data property `lgd:asWKT` that connects a spatial object with its geometry is universal for all spatial objects in the dataset. We present the mapping for two of these tables/shapefiles in Fig. 6.

```
lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral.
← SELECT gid, geom FROM buildings

lgd:{gid} lgd:asWKT {geom}^^geo:wktLiteral.
← SELECT gid, geom FROM landuse
```

Fig. 6. Examples of geospatial mappings for two LGD tables.

```
SELECT ?s1 ?o1 WHERE {
  ?s1 lgd:asWKT ?o1 .
  FILTER(geof:FUNCTION(SPATIAL_CONSTANT,?o1))
}
```

Fig. 7. Template for spatial selection queries.

Listing 1: Spatial selection query 19

```
SELECT distinct ?s1 WHERE {
  ?s1 lgd:asWKT ?o1 .
  { {?s1 rdf:type lgd:Road} UNION
    {?s1 rdf:type lgd:Waterway} }
  FILTER(geof:sfIntersects(GEOMETRY,?o1))
}
```

Listing 2: Spatial join query 6

```
SELECT ?s1 ?s2 WHERE {
  ?s1 lgd:asWKT ?o1 .
  ?s2 lgd:asWKT ?o2 .
  FILTER(geof:sfIntersects(?o1,?o2))
}
```

Let us now consider the template for spatial selection queries in Fig. 7. The translated SQL query corresponding to a GeoSPARQL query following this template would create unions in order to fetch results deriving from all the tables it has been mapped to, that is, all seven LGD tables, and then apply the spatial selection to this union. This is the case for spatial selection queries 15–19. In order to test how our system responds by increasing/decreasing the number of unions produced in the translated query, we add an additional, thematic filter that selects a different number of LGD categories each time, thus affecting a different number of tables, and producing different number of unions, respectively. For example, consider query 19 shown in Listing 1, which contains a union to retrieve both waterways and roads (coming from different shapefiles, thus different tables in the database).

The queries 15, 16, 17, and 18 produce 6, 4, 3, and 4 unions respectively. The presence of unions has a negative impact on the query response time, but things get even worse when unions appear in spatial joins (e.g., spatial join query 6). Since variables appear in the spatial filters that serve as the conditions of the spatial joins, all combinations of the respective tables that are involved in the corresponding mappings should be spatially joined pairwise.

For example, consider the spatial join query 6 which is given in Listing 2. This query performs a spatial join with the condition `intersects` in *all* LGD tables that are involved in the mappings containing the predicate *lgd:asWKT*. This join is translated into the corresponding relational algebra expression as follows:

$$(L_{buildings} \cup L_{luse} \cup \cdots \cup L_{waterways}) \bowtie_{sf} (L_{buildings} \cup L_{luse} \cup \cdots \cup L_{waterways})$$

Table 2
Spatial selections description.

No	Query	#BGPs	Results
00	Equals_GADM_P	1	0
01	Contains_GADM_P	1	9
02	Contains_GADM_P	1	0
03	Equals_GADM_L	1	1
04	Overlaps_GADM_L	1	0
05	Contains_GADM_L	1	0
06	Intersects_CLC_L	1	5
07	Contains_CLC_L	1	0
08	Equals_CLC_L	1	5
09	Overlaps_CLC_L	1	0
10	Overlaps_CLC_P	1	132
11	Intersects_CLC_P	1	533
12	Contains_CLC_P	1	401
13	Equals_CLC_P	1	0
14	Intersects_LGD_P	2	2749
15	Intersects_LGD_B	2	2749
16	Intersects_LGD_PL	2	2626
17	Intersects_LGD_P	2	2522
18	Intersects_LGD_LU	2	2722
19	Intersects_LGD_ROA	2	2387
20	Intersects_LGD_bigP	1	729189
21	Intersects_LGD_P2	3	5

Table 3
Spatial joins description.

No	Query	#BGPs	Results
00	Within_CLC_GADM	2	34114
01	Intersects_GADM_GADM	2	1556
02	Overlaps_GADM_CLC	2	17035
03	Intersects_LGD_GADM	3	154725
04	Intersects_LGD_LGD_Mus	4	2
05	Intersects_LGD_GADM	2	819319
06	Intersects_LGD_LGD	1	3686229
07	Crosses_LGD_LGD_Roads	4	178602

Table 4
Workload characteristics.

Dataset	Size	Tuples	$Avg\ \frac{\#points}{geometry}$
CLC	283MB	44834	187.84
Hotspots	35 MB	37048	5
GAG	24 MB	326	3020.14
LGD-Buildings	42 MB	155474	6.5
LGD-Landuse	20 MB	40220	19.4
LGD-Places	2.4 MB	13043	1
LGD-Points	12 MB	61664	1
LGD-Railways	2 MB	4996	13.3
LGD-Roads	250 MB	514403	19
LGD-Waterways	16 MB	20565	39.84

where $L_{buildings}, L_{luse}, \ldots, L_{waterways}$, etc. are LGD tables and sf is spatial operator corresponding to `geof:sfIntersects` from the query. The query engine evaluates this relational algebra expression as unions of joins and all involved tables get spatially joined pairwise.

Last, in order to measure how the selectivity of the queries affect the performance of the systems, we included the spatial selection queries 20 and 21 involve the computation of the intersection of all kinds of LGD areas with a specific polygon. This polygon is large in the case of spatial selection query 20 so that many geometries will be returned, while in spatial selection query 21 this polygon is small enough so that very few LGD areas intersect with it.

6.3. Results

Experimental set up. The experiments were carried out on a server with the following specifications: Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz, 12MB L3, RAID 5, 32GB RAM and OS: Ubuntu 12.04. All experiments were carried out with both cold and warm cache. Queries are first executed in cold cache and then in warm cache. The queries for which the system under test times out (the time out threshold is set to 40 min) are not executed in warm cache. All queries and code we used to execute the experiments in both systems, can be found in the "experiments" branch of the github repository of Ontop-spatial (folder "benchmark") at https://github.com/ConstantB/Ontop-spatial.

Query response time. The results of our experimental evaluation can be seen in Figs. 8 and 9. Response time is measured in milliseconds and presented in logarithmic scale. A general observation is that the query response time of Ontop-spatial is better than the one of Strabon and System-X, especially when big datasets are involved, both for spatial selections and spatial joins. Strabon times out after 40 min in spatial join queries 6 and 7. System-X times out after 40 min in spatial join queries 0, 1, 2, 3, 6 and 7. In spatial selection queries 2–5, although Ontop-spatial achieves better response time than Strabon in cold cache, it gets outperformed in warm cache, as intermediate results (which are not many as the dataset involved in this query is relatively small), are more likely to be found in the cache, increasing the hit rate of the cache and decreasing I/O requests. However, such differences between executions in warm and cold cache are eliminated in larger datasets. System-X performs worse than Ontop-spatial and Strabon in both cases.

In what follows we explain the reason why Ontop-spatial outperforms Strabon in most cases, as well as why it does not in other cases.

Listing 3: Spatial join query 2

```
SELECT ?s1 ?s2 WHERE {
  ?s1 clc:asWKT ?o1 .
  ?s2 gag:asWKT ?o2 .
  FILTER(geof:sfWithin(?o1, ?o2))
}
```

Listing 4: Spatial join query 4

```
SELECT ?s1 ?s2 WHERE {
  ?s1 lgd:asWKT ?o1 .
  ?s1 rdf:type lgd:Building .
  ?s1 lgd:type "Museum" .
  ?s2 lgd:asWKT ?o2 .
  ?s2 rdf:type lgd:Landuse .
  FILTER(geof:sfIntersects(?o1,?o2))
}
```

Listing 5: Ontop-spatial SQL query

```
SELECT
 1 AS "s1QuestType", NULL AS "s1Lang",
 ('http://geo.linkedopendata.gr/clc/'
   || clc.gid || '/') AS "s1",
 1 AS "s2QuestType", NULL AS "s2Lang",
 ('http://geo.linkedopendata.gr/gag/ont/'
   || gag.gid || '/') AS "s2"
FROM
  clc QVIEW1, gag QVIEW2
WHERE
  QVIEW1."gid" IS NOT NULL AND
  QVIEW1."geom" IS NOT NULL AND
  QVIEW2."gid" IS NOT NULL AND
  QVIEW2."geometry" IS NOT NULL AND
  ST_Within(QVIEW1."geom",QVIEW2."geometry")
```

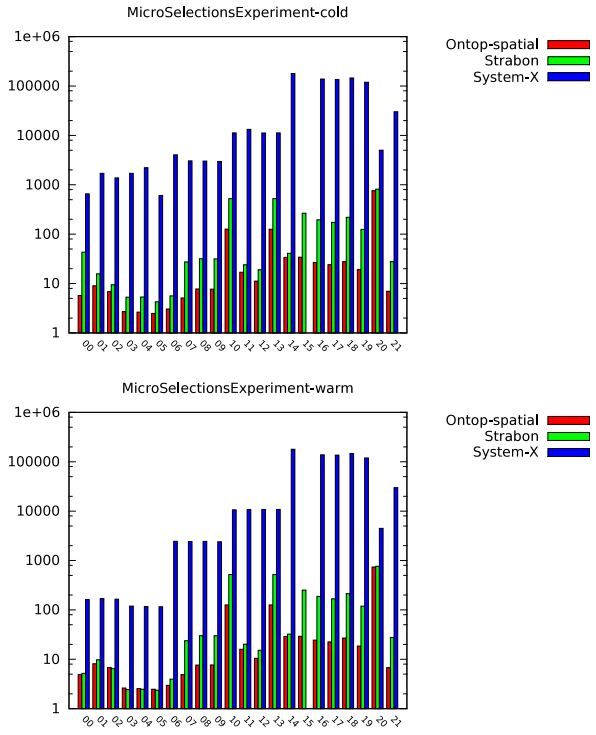


Fig. 8. Spatial Selections experiment (cold and warm cache).

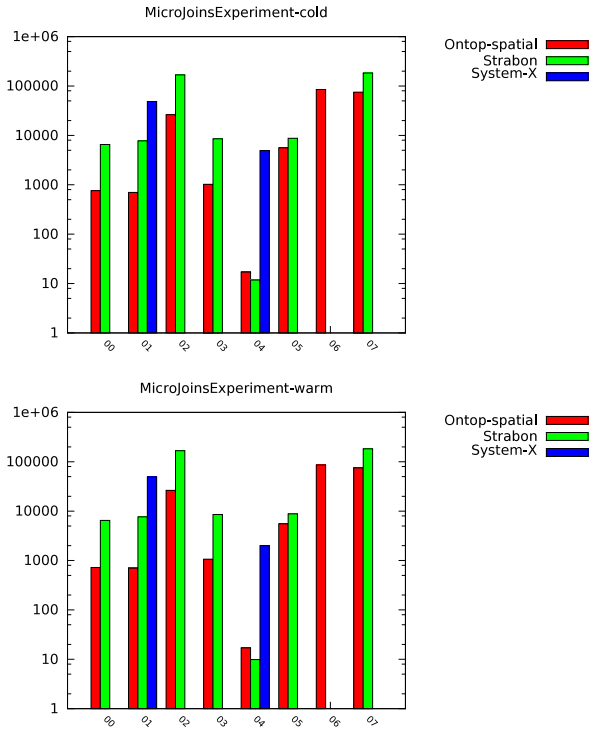


Fig. 9. Spatial Joins experiment (cold and warm cache).

Listing 6: Strabon SQL query

```
SELECT a0.subj, u_s2.value, a2.subj, u_s1.value
FROM aswkt_855211 a0
  INNER JOIN geo_values l_o2 ON (l_o2.id = a0.obj)
  INNER JOIN geo_values l_o1
    ON ST_Within(l_o1.strdfgeo, l_o2.strdfgeo)
  INNER JOIN aswkt_135992 a2 ON (a2.obj = l_o1.id)
  LEFT JOIN uri_values u_s2 ON (u_s2.id = a0.subj)
  LEFT JOIN uri_values u_s1 ON (u_s1.id = a2.subj)
```

The queries provided in Listings 5 and 6 are the SQL translations of the GeoSPARQL spatial join query 2, which is provided in Listing 3. One can observe that Ontop-spatial produces the same query as one would have written by hand in a geospatial relational database. Strabon produces some extra joins, as a result of the star schema that it follows in the database (and has been inherited from the Sesame RDBMS that Strabon is built on), i.e., each predicate is stored in a different table and there are some additional tables used for dictionary encoding (tables storing URIs, one table for each different datatype, etc.). This has a negative impact on performance when many intermediate results are produced. In Strabon, geometries are stored in a single table, named *geo_values*, and are indexed on the geometry column using an R-tree-over-GiST index. On the other hand, Ontop-spatial stores each shapefile in a different table, and geometries are stored in a separate column for each table, and a separate R-tree-over-GiST index is constructed for the geometries of each shapefile/table. As Table 4 shows, there are cases where geometries of a shapefile/table are of the same type (e.g., all contain points/linestrings/polygons), allowing Ontop-spatial to build smaller and more efficient indices.

Nevertheless, in spatial join query 4, Strabon outperforms Ontop-spatial. The query is provided in Listing 4. Using this query, we want to retrieve the land use of areas that intersect with Museums. This is a very selective query with respect to the thematic condition, so the PostgreSQL optimizer correctly chooses to perform the thematic conditions first so that only the geometries of Museums will be checked in the spatial condition that follows, and the R-tree index will be used. Both systems execute the query very fast, with Strabon achieving nearly 4 times better performance than Ontop-spatial, as the overhead of the extra joins it performs, as described above, is reduced because very few intermediate results are produced. Also, dictionary decoding helps Strabon to perform string comparison (for value "Museum") only once, in order to retrieve the id of that value and then perform thematic joins efficiently using the id (numeric) value.

Queries 15–19 have filters that select different kinds of LGD categories. Query response time increases every time many LGD categories are involved (Query 15 asks for all categories), producing the respective number of unions in the case of Ontop-spatial and more intermediate results for Strabon, forcing more geometries to be checked in the spatial filter. On the contrary, query response time decreases when less LGD categories need to be selected.

The results of union-queries are more interesting in the case of spatial joins, shown in Fig. 9. One would expect that unions with spatial joins, as in the case of the spatial join query 6, would dramatically decrease the performance of Ontop-spatial. Indeed, query response time increases in the case of queries like query 6, but Ontop-spatial still performs better than Strabon. The explanation for this lies in the fact that each time a spatial join is performed between two different LGD tables, the optimizer chooses the one having the smaller index (and usually smaller geometries, in this case) to be nested inside the inner branch of the nested loop, where it performs an index scan. This has greater impact on the execution time of geospatial queries, as the evaluation of spatial joins is more expensive due to the cost of the evaluation of the spatial conditions.

In spatial selection query 20, the performance of the two systems is very close, while in the more selective version of the same query, i.e., spatial selection query 21, the gap in the execution times between Ontop-spatial and Strabon increases again. This happens because nearly every geometry in the workload is included in the results, so spatial indices are not useful in this case.

System-X performs worse than the other two systems in all cases mainly because of the fact that its geospatial index relies on Apache Lucene,[14] that does not support R-trees but simpler forms of spatial indices, such as quad-trees. The spatially-enabled RDBMS (i.e., PostgreSQL with PostGIS extension enabled) that serves as the back-end for Ontop-spatial and Strabon, on the other hand, incorporates more advanced and mature techniques for efficient geospatial query processing which are considered standard for the relational database community, such as support for R-tree indices and spatially-enabled optimizer.

Overall, we observe that importing the shapefiles to a database and then using an OBDA approach is very efficient, as in most cases, the information that is contained in a shapefile is compact and homogeneous, as we often have one shapefile per data source. So, the SQL queries that are produced based on such a schema contain reduced amount of joins and can be executed efficiently. It is also evident from the experiments that forwarding geospatial query processing to a spatially-enabled DBMS as back-end can improve performance significantly, as they incorporate well-established optimization techniques in the area of geospatial query processing that have not yet been incorporated in native geospatial triple stores up to date.

7. Related work

The work on extending RDF and SPARQL with geospatial functionality also gave rise to the implementation of geospatial RDF stores such as Parliament, uSeekM and Virtuoso, that implement a subset of GeoSPARQL, and Strabon [6] that implements both GeoSPARQL and stSPARQL. Meanwhile, the work of Ontop-temporal [34,35] extends the OBDA framework with temporal reasoning.

There have also been systems that enable the translation of geospatial data from their native formats to RDF. GeoTriples [36] is a tool for the conversion of geospatial data from a variety of source formats (shapefiles, relational databases, XML files, etc.) to RDF using GeoSPARQL and stSPARQL vocabularies and R2RML mappings.

Another category of systems that are related to our work is SPARQL-to-SQL systems such as Ontop [28], Ultrawrap [37], D2RQ[15] and Morph [24]. These systems offer no geospatial functionality.

In the area of description logics, the work described in [38] extends DL-Lite with spatial primitives and presents a rewriting mechanism to standard DL-Lite, preserving FOL-rewritability. The work described in [39] examines the FOL rewritability of spatial calculi (e.g., RCC8, RCC2) combined with DL-Lite. PelletSpatial [40] is a qualitative spatial reasoner implemented on top of Pellet.

8. Discussion and conclusions

In this paper, we describe how we extended the techniques of [28] to develop the first geospatially-enabled OBDA system, named Ontop-spatial. By extending the OBDA system Ontop, Ontop-spatial inherits the advantages of using RDB2RDF systems in real use cases: (i) RDB-to-RDF workflow becomes less complicated, without having to use different tools for converting data into RDF and then storing it in RDF stores, (ii) no data needs to be transferred, as existing databases are used as input to the system, and (iii) mappings provide a layer of abstraction between the data manipulation/database experts and the end users.

These advantages have even greater impact when dealing with geospatial data. The domains where geospatial data are produced and used are dominated by geospatial databases and other tabular file formats that could easily be imported to a database (e.g., shapefiles). GIS practitioners use geospatial relational databases in their day-to-day tasks, either directly or as the back-end of applications to store and manipulate data (e.g., GIS have connectors for geospatial relational databases). Ontop-spatial provides a solution for combining the advantages of geospatial relational databases, for example, the wide variety of geospatial data operators and the performance achieved by the use of spatial indices, with the data modeling advantages of the RDF data model. Moreover, Ontop-spatial allows for encapsulating geospatial data manipulation functions offered by geospatial extensions to SQL (e.g., functions for transforming geometries to a different coordinate reference system) in the mappings.

On the other hand, Ontop-spatial inherits the disadvantages of the OBDA systems as well. First, in order to combine information coming from different geospatial sources, the data should be imported in databases. Second, as the database is given as input to the system, it is read-only and Ontop-spatial does not support SPARQL store or update operations; all updates should be done directly on the database level. Third, the performance of the system is heavily dependent on the ontology, the schema of the database, and the mapping, as we explained in the previous sections, which applies for OBDA approaches in general. However, our experiments showed that in many cases, our geospatially enhanced OBDA approach achieves significantly better performance than the state-of-the-art geospatial RDF store Strabon. The main reasons for this are summarized as follows:

- The database schema that is produced simply by importing the shapefiles to the database is in most cases suitable for OBDA approaches, as shapefiles contain compact and homogeneous information per dataset.
- The database produced by storing the materialized RDF dump that Ontop exports in Strabon is bigger than the database that results from importing the shapefiles, even though only the RDF triples that were involved in the OBDA mapping (i.e., the *virtual* RDF triples) were exported. This happens because of (i) the normalization imposed by the RDF data model itself (i.e., triples) and (ii) the additional tables used for dictionary encoding.
- The additional joins that are created in the translated SQL queries of Strabon and the fact that geometries are stored in a single table where geospatial operators are performed increase even by more than an order of magnitude in very large workloads with many and complicated geometries, when many intermediate results are produced in queries.

In future work, we plan to continue the development of Ontop-spatial in the directions of (i) fully supporting GeoSPARQL and stSPARQL (i.e., adding also valid time support), and (ii) improving the performance by using cost-based query planner [41] or by creating a distributed version of our extension exploiting the fact that the union-all spatial queries are parallelizable, and (iii) carrying out more scalability evaluation with a data scaler [42].

Acknowledgments

This work is partially supported by the EU projects Optique (318338), MELODIES (603525) and Copernicus App Lab (730124), unibz projects OBATS and QUADRO. We thank the Ontop development team for their support.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

14 https://lucene.apache.org/core/.

15 http://d2rq.org/.

References

[1] M. Koubarakis, K. Kyzirakos, Modeling and querying metadata in the semantic sensor web: the model strdf and the query language stSPARQL, in: L. Aroyo, et al. (Eds.), ESWC, in: LNCS, vol. 6088, Springer, 2010, pp. 425–439.

[2] Open Geospatial Consortium. GeoSPARQL - A geographic query language for RDF data, OGC Candidate Implementation Standard, 2012.

[3] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. Zakharyaschev, Ontology-based data access: A survey, in: IJCAI-ECAI-18–July 13–19 2018, Stockholm, Sweden, 2018.

[4] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual knowledge graphs: an overview of systems and use cases, Data Intelligence 1 (2019) 201–223.

[5] G. Garbis, K. Kyzirakos, M. Koubarakis, Geographica: a benchmark for geospatial RDF stores (long version), in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), ISWC, in: Lecture Notes in Computer Science, vol. 8219, Springer, 2013, pp. 343–359.

[6] K. Kyzirakos, M. Karpathiotakis, M. Koubarakis, Strabon: a semantic geospatial DBMS, in: ISWC, in: LNCS, vol. 7649, Springer, 2012, pp. 295–311.

[7] K. Bereta, M. Koubarakis, Ontop of geospatial databases, in: P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, Y. Gil (Eds.), The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I, Springer International Publishing, Cham, 2016, pp. 37–52.

[8] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 concepts and abstract syntax, in: W3C Recommendation, W3C, 2014. Available at https://www.w3.org/TR/rdf11-concepts/.

[9] D. Peterson, S.S. Gao, A. Malhotra, C.M. Sperberg-McQueen, H.S. Thompson, W3C XML schema definition language (XSD) 1.1 part 2: datatypes, W3C Recommendation, W3C, 2012. Available at https://www.w3.org/TR/xmlschema11-2/.

[10] S. Harris, A. Seaborne, SPARQL 1.1 query language, W3C recommendation, W3C, 2013. Available at http://www.w3.org/TR/sparql11-query.

[11] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of SPARQL, ACM Trans. Database Syst. 34 (3) (2009) 16:1–16:45, http://dx.doi.org/10.1145/1567274.1567278.

[12] R. Kontchakov, M. Rezk, M. Rodriguez-Muro, G. Xiao, M. Zakharyaschev, Answering SPARQL queries over databases under OWL 2 QL entailment regime, in: Proc. of International Semantic Web Conference (ISWC 2014), in: LNCS, vol. 8796, Springer, 2014, pp. 552–567, http://dx.doi.org/10.1007/978-3-319-11964-9_35.

[13] M. Kaminski, E.V. Kostylev, B. Cuenca Grau, Query nesting, assignment, and aggregation in SPARQL 1.1, ACM Trans. Database Syst. 42 (3) (2017) 17:1–17:46.

[14] B. Glimm, C. Ogbuji, SPARQL 1.1 entailment regimes, W3c recommendation, W3C, 2013. Available at http://www.w3.org/TR/sparql11-entailment/.

[15] G. Xiao, D. Hovland, D. Bilidas, M. Rezk, M. Giese, D. Calvanese, Efficient ontology-based data integration with canonical IRIs, in: ESWC, in: Lecture Notes in Computer Science, vol. 10843, Springer, 2018, pp. 697–713.

[16] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 web ontology language profiles, in: W3C Recommendation, W3C, Second ed., 2012.

[17] W3C OWL Working Group, OWL 2 Web Ontology Language document overview, second ed., W3C Recommendation, W3C, 2012.

[18] K. Bereta, P. Smeros, M. Koubarakis, Representation and querying of valid time of triples in linked geospatial data, in: Extended Semantic Web Conference 2013, vol. 7882, Springer Berlin Heidelberg, 2013, pp. 259–274.

[19] Open Geospatial Consortium. OpenGIS Simple Features Specification For SQL, OGC Implementation Standard, 1999.

[20] M. Egenhofer, A formal definition of binary topological relationships, in: Foundations of Data Organization and Algorithms, in: Lecture Notes in Computer Science, vol. 367, Springer Berlin Heidelberg, 1989, pp. 457–472.

[21] D.A. Randell, Z. Cui, A.G. Cohn, A spatial logic based on regions and connection, in: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25–29, 1992, pp. 165–176.

[22] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semantics 10 (2008) 133–173.

[23] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: answering SPARQL queries over relational databases, Semantic Web J. 8 (3) (2017) 471–487.

[24] F. Priyatna, O. Corcho, J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proc. of the 23rd International Conference on World Wide Web, ACM, NY, USA, 2014, pp. 479–490.

[25] D. Calvanese, G.D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D.F. Savo, The mastro system for ontology-based data access, Semantic Web 2 (1) (2011) 43–53.

[26] B. Glimm, C. Ogbuji, SPARQL 1.1 entailment regimes, W3C recommendation, W3C, 2013.

[27] J.R. Herring, OpenGIS implementation specification for geographic information - simple feature access - Part 2: SQL option, OpenGIS Implementation Standard 06-104r4, Open Geospatial Consortium Inc., 2010.

[28] M. Rodriguez-Muro, M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, J. Web Semant. 33 (1) (2015).

[29] C. Nikolaou, K. Dogani, K. Bereta, G. Garbis, M. Karpathiotakis, K. Kyzirakos, M. Koubarakis, Sextant: visualizing time-evolving linked geospatial data, J. Web Sem. 35 (2015) 35–52.

[30] E. Kharlamov, D. Hovland, M.G. Skjæveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao, A. Soylu, D. Lanti, M. Rezk, D. Zheleznyakov, M. Giese, H. Lie, Y. Ioannidis, Y. Kotidis, M. Koubarakis, A. Waaler, Ontology based data access in statoil, J. Web Semant. 44 (2017) 3–36.

[31] K. Bereta, G. Xiao, M. Koubarakis, M. Hodrius, C. Bielski, G. Zeug, Ontop-spatial: geospatial data integration using GeoSPARQL-to-SQL translation, in: Proceedings of the ISWC 2016 Posters & Demonstrations Track. Co-located with the 15th International Semantic Web Conference (ISWC 2016), CEUR Electronic Workshop Proceedings, vol. 1690, 2016.

[32] S. Bruggemann, K. Bereta, G. Xiao, M. Koubarakis, Ontology-based data access for maritime security, in: The Semantic Web. Latest Advances and New Domains: 13th Extended Semantic Web Conference, ESWC 2016, Heraklion, Crete, Greece, May 29–June 2, 2016, Proceedings, Springer International Publishing, 2016, pp. 741–757.

[33] P. Bellini, P. Nesi, Performance assessment of RDF graph databases for smart city services, J. Vis. Lang. Comput. 45 (2018).

[34] S. Brandt, E. Güzel Kalaycı, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Querying log data with metric temporal logic, J. Artificial Intelligence Res. 62 (2018) 829–877.

[35] E.G. Kalayci, G. Xiao, V. Ryzhikov, T.E. Kalayci, D. Calvanese, Ontop-temporal: a tool for ontology-based query answering over temporal data, in: CIKM, ACM, 2018, pp. 1927–1930.

[36] K. Kyzirakos, I. Vlachopoulos, D. Savva, S. Manegold, M. Koubarakis, GeoTriples: a tool for publishing geospatial data as RDF graphs using R2RML mappings, in: Proceedings of the ISWC 2014 Posters & Demonstrations Track, Riva del Garda, Italy, October 21, 2014, pp. 393–396.

[37] J. Sequeda, D. P.M.iranker, Ultrawrap: SPARQL execution on relational data, J. Web Semant. 22 (2013).

[38] T. Eiter, T. Krennwallner, P. Schneider, Lightweight spatial conjunctive query answering using keywords, in: The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26–30, 2013. Proceedings, 2013, pp. 243–258.

[39] Ö.L. Özçep, R. Möller, Scalable geo-thematic query answering. in: The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11–15, 2012, Proceedings, Part I, 2012, pp. 658–673.

[40] M. Stocker, E. Sirin, PelletSpatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine, in: Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23–24, 2009.

[41] D. Lanti, G. Xiao, D. Calvanese, Cost-driven ontology-based data access, in: International Semantic Web Conference (1), in: LNCS, vol. 10587, Springer, 2017, pp. 452–470.

[42] D. Lanti, G. Xiao, D. Calvanese, VIG: data scaling for OBDA benchmarks, Semantic Web 10 (2) (2019) 413–433.