Doctoral Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

OnGIS: Ontology-Based Geospatial Data Integration and Retrieval

Marek Šmíd

Supervisor: Doc. Ing. Zdeněk Kouba, CSc.
Supervisor–study guarantor: Ing. Petr Křemen, Ph.D.
Field of study: Electrical Engineering and Information Technology
Subfield: Artificial Intelligence and Biocybernetics
August 2016

Acknowledgements

My thanks go to both my supservisor Doc. Ing. Zdeněk Kouba, CSc. and study guarantor Ing. Petr Křemen, Ph.D., for their patience and willingness to help. My gratitude also belongs to my wife Eliška, whose endurance and support helped me during almost all of my long doctoral studies. Last, but not least, my thanks go to all colleagues and staff of our Department of Cybernetics, who made it a pleasant place to work at during the time of a transition.

My research has been partially supported by the Grant Agency of the Czech Technical University in Prague, the grant no. SGS10/276/OHK3/3T/13 and the grant no. SGS13/204/OHK3/3T/13.

Declaration

I declare that my doctoral thesis was prepared personally and bibliography used was duly cited. This thesis and the results presented were created without any violation of copyright of third parties.

Abstract

Querying geospatial data from multiple heterogeneous sources backed by different management technologies poses an interesting problem in the data integration and the subsequent result interpretation. I propose two ways of entering complex spatial queries in a user-friendly way and broker techniques for answering a query by finding relevant data sources (from a catalogue) capable of answering the query, eventually splitting the query and finding relevant data sources for the query parts, when no single source suffices.

For expressing data source capabilities, I describe each source with a set of prototypical queries that are algorithmically arranged into a lattice, which makes searching efficient. The proposed algorithms leverage GeoSPARQL query containment enhanced with OWL 2 QL semantics. Using GeoSPARQL as both the querying language and the language for expressing capabilities gives the flexibility to incorporate many heterogeneous geospatial data sources, no matter what storage, data model, or terminology they use.

All parts of the design are presented in a prototype called OnGIS, which implements the proposed methods of entering a user query, searching for the relevant data sources, and translating the query to requests the underlying data source technologies understand.

Keywords: geospatial semantics, OWL 2 QL, data integration, query containment, query broker, heterogeneous data, lattice, OnGIS

Supervisor: Doc. Ing. Zdeněk Kouba, CSc.
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

Abstrakt

Dotazování geografických dat z mnoha heterogenních zdrojů založených na různých technických platformách představuje zajímavý problém jak pro integraci dat, tak pro jejich následující interpretaci. V této práci navrhuji dva způsoby přívětivého zadávání složitých prostorových dotazů a vyhledávacího zprostředkovatele pro zodpovídání těchto dotazů hledáním vhodných zdrojů z katalogu, které by tyto dotazy zodpověděly. V případě, že nějaký dotaz nelze zodpovědět jedním zdrojem, je dotaz rozdělen a hledají se zdroje vhodné pro jednotlivé části dotazu.

Pro vyjádření, jaké dotazy je zdroj schopen zodpovědět, charakterizuji každý zdroj sadou prototypových dotazů, které jsou uspořádány do svazu, což usnadňuje vyhledávání. Navržené algoritmy používají porovnávání dotazů vyjádřených v jazyce GeoSPARQL s využitím OWL 2 QL sémantiky. Použití GeoSPARQL pro vyjádření jak uživatelského dotazu, tak zodpovídacích schopností zdrojů dává možnost zapojit mnoho rozličných geografických datových zdrojů, bez ohledu na techniku jejich úložiště, jejich datový model či terminologii.

Všechny části návrhu jsou ukázány na prototypu nazvaném OnGIS, který implementuje navržené metody uživatelského zadávání dotazů, hledání relevantních datových zdrojů a překlad dotazů do požadavků, kterým příslušné datové zdroje rozumí.

Klíčová slova: sémantika geografických dat, OWL 2 QL, integrace dat, porovonávání dotazů, zprostředkovatel dotazů, různorodá data, svaz, OnGIS

Překlad názvu: OnGIS: Propojování a zpřístupňování geografických dat pomocí ontologií

Contents

Figures

Tables

Chapter 1

Introduction

This thesis focuses on searching geospatial data, which have some specifics compared to general data on the Internet. General search engines, when dealing with large amounts of mostly unstructured data, usually written in natural languages, have to cope with impreciseness. Such search engines usually use statistics to find relevant sources, word similarities, etc.

But in the GIS (geographical information systems) world, it is a little different. Publishing geospatial data requires more technical effort, which usually also calls for describing what the data are about (such description is often called metadata). There is also a long tradition of GIS systems with rigid data models, frequently tightly coupled with relational databases.

As GIS data are important for many aspects of human living, also governments spend a lot of money to build high-quality geospatial sources for the country needs – cadastral maps, city and postal maps, maps analyzing various aspects of nature and meteorology, and many more. Governments not only initiate creating and publishing such sources but also dictate how public data should be published – they give technical requirements and provide a guide how the data should be annotated by metadata. This is a case of INSPIRE [3] (see Section 2.5.2), an initiative of European Commission to standardize public data among European Union countries. There are also open initiatives of international map projects, for example OpenStreetMap (see Section 2.5.1), which gained very high quantity and good quality of data.

The structured character of GIS data makes it feasible to describe them with semantic techniques, e.g. all spatial objects of a map layer can become instances of a class, an attribute of objects in a layer can become a data property (for the explanation of the semantic terms, see Chapter 3). An example can be OpenStreetMap, which is semantically described by LinkedGeoData initiative (see Section 3.2.1).

The semantic layer over GIS systems brings the advantage of adding domain knowledge over the structure of the data (e.g. hierarchies over feature classes, with an example of *Pizzeria* being a subclass of *Alimentation*) and of linking the GIS systems together (e.g. that a feature class in one GIS is equal to or a subclass of a feature class in another GIS, with an example of *Forest* in one GIS being a subclass of *NatureLandscape* in another).

This thesis focuses on using GIS data sources described by semantic techniques for a user-friendly searching, being agnostic to the underlying technologies of the sources. It consists of three main parts:

- A proposal of two user-friendly ways of entering a complex spatial query based on semantic entities (for explanation see Chapter 3), so the query has a precise meaning,
- translation of the semantic queries to requests for different sources providing geospatial data, e.g. the support for relational databases, SPARQL endpoints (see Section 3.4.1), and WMS and ArcGIS services (see Section 2.4), and
- describing source capabilities (both in terms of data it has and operations it provides) by semantic techniques, which allows searching for relevant sources for a user's query. The proposed algorithms support the case when no single source can answer the query, i.e. when the query has to be split into parts, and for each part, a relevant source has to be found.

A potential application of the proposed system is in the area of urban design, as city planning is a complex task, requiring the use of various map data, potentially from various sources. This is reflected in my experiments, where one of the data sources is the geoportal of IPR Praha (see Section 2.5.2), offering many interesting map layers.

One of the characteristics of the proposed system is that it uses one of the ontology languages (OWL 2 QL, see Section 3.3.4), which has lower expressive power compared to some other widely used languages, but is tractable (i.e. it is of polynomial time complexity), which makes it possible to translate queries with OWL 2 QL semantics to SQL queries. This makes it possible to connect the proposed system to many existing GIS data sources, as many use relational databases. For representing queries, a recent standard GeoSPARQL is used (see Section 3.4.2).

1.1 Thesis Contributions

The contribution of this thesis is proposing a complete system for intuitive query answering in the field of geospatial data. As the underlying technologies, the GeoSPARQL query language with OWL 2 QL semantics (see Chapter 3) is used. Specifically, the novel contributions are:

- Designing two user-friendly ways of entering a geospatial query,
- proposing annotations for mapping domain ontologies to several GIS service technologies, which are tailored for spatial requests,
- designing a method of geospatial query containment, i.e. adapting the techniques proposed in [4] and [5] to OWL 2 QL language and extending it by geospatial reasoning, and

- designing the algorithms for arranging a set of geospatial queries to a lattice using the query containment and for searching the lattice. The searching algorithm supports the case a query is not found in the lattice and has to be split into parts. The algorithms are useful for searching a source, eventually sources, capable of answering a given user's query.

These contributions are presented in two articles in conference proceedings and a reviewed journal paper (and one more in the review process) listed in the *Publications of the Author Relevant to the Thesis* section at the end of this thesis.

1.2 Thesis Outline

The thesis is structured as follows: Chapter 2 gives an introduction to the GIS domain, and a comparison to other GIS integration techniques in Section 2.6. Chapter 3 presents the ideas and techniques of semantic web relevant to the purpose of this thesis, with Section 3.5 giving an overview of similar work based on semantic techniques. The design of the proposed system, including two methods of user-friendly query input, spatial query containment design, and data source searching algorithms, is presented in Chapter 4. The implementation aspects of the OnGIS prototypes are briefly described in Chapter 5. Chapter 6 presents a list of successful examples of testing the implementation, proving the proposed design is feasible. Finally, Chapter 7 gives a conclusion.

Chapter 2

GIS and Geospatial Integration

GIS (Geographic information system) is a broad term referring to a wide family of standards, technologies, and software tools to store, edit, analyze, display, and integrate spatial data. In this thesis, I focus on the search and integration aspects.

This chapter presents some basic concepts in Section 2.1, common GIS operations in Section 2.2, ways how to express spatial data in Section 2.3, standards and software tools for providing services in Section 2.4, some relevant data sources in Section 2.5, and finally the existing work related to this thesis is presented in Section 2.6.

2.1 GIS Basics

The basic division of GIS data is a geometry of an object and attributes of an object. A geometry of a spatial object can be a point, a line, a polygon (optionally with holes), or a collection of any of the aforementioned. Attributes of an object are non-spatial data values, e.g. a name, opening hours (in the case of a shop), road number, population, etc.

Traditionally, the main output of GIS systems is a map. A map can contain a legend (explaining what all the used symbols mean), a compass rose (to show the orientation), a map scale (to show the scale of the map), and when the map is interactive (e.g. a map on a website, a computer application displaying a map), the tools to operate the map (tools for zooming, panning, selecting layers, searching, highlighting objects, etc.). The map can be presented in a vector, or more often in a raster format. A map can consist of multiple layers, which are rendered in a given order, one on top of another. The first layer (rendered first) is usually called a base layer, the rest overlay layers. Layers can be raster images (e.g. satellite imagery, aerial photography), vector objects (e.g. roads, land parcels, house outlines), and texts (e.g. city names, street names, house numbers, statistic values).

Other types of GIS outputs can be vector geometries, which can be used for further processing, and data tables containing some statistics computed from the GIS data (e.g. country names with the areas they cover and the lengths of their borders, or average precipitations of continents).

2.2 Spatial Relations and Operations

Relations of two geometries can be simplified to topological relations, which neglect the exact geometry coordinates, mutual orientation, or distance. They just reflect the mutual relationship of how their interiors and boundaries overlap. The topological relation of two geometries can be described by DE-9IM (dimensionally extended nine-intersection model) [6], which is represented by a 3×3 matrix:

$$\text{DE-9IM}(a,b) = \begin{pmatrix} \dim(I(a) \cap I(b)) & \dim(I(a) \cap B(b)) & \dim(I(a) \cap E(b)) \\ \dim(B(a) \cap I(b)) & \dim(B(a) \cap B(b)) & \dim(B(a) \cap E(b)) \\ \dim(E(a) \cap I(b)) & \dim(E(a) \cap B(b)) & \dim(E(a) \cap E(b)) \end{pmatrix}, \tag{2.1}$$

where dim is a dimension of an intersection, and I, B, and E are interior, boundary, and exterior, respectively, of a geometry. The dimension can be -1 (often denoted as “F”) when there is the empty intersection, 0 (the intersection is a point), 1 (a line), and 2 (an area).

Then a DE-9IM relation can be represented as a string consisting of nine characters (read from the matrix by lines from the top-left corner). An example in Fig. 2.1 has the DE-9IM string “1020F1102”. To allow representing groups of DE-9IM relations, the dimension “T” represents any nonempty intersection (i.e. dim = 0, 1, or 2), and the dimension “*” stands for any dimension.

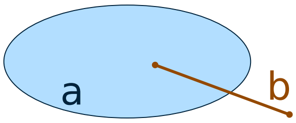


Figure 2.1: Two geometries with DE-9IM relation 1020F1102.

To simplify it (reduce the complexity of expressing a relation), there are three sets (families) of named topological relations, each containing eight easily understandable relations. The three families are a simple feature family, Egenhofer family, and RCC8 (Region Connection Calculus) [1] family. They all are used in GeoSPARQL described in Section 3.4.2, where also definitions and brief descriptions of the relations are given. See simple examples of RCC8 relations in Fig. 2.2. For more details, see Section 3.4.2.

Spatial operations can be used to create new geometries out of existing, for various analytical tasks, and for many other purposes. For example, the complete list of the spatial functions of PostGIS (presented in Section 2.4.2) is in [7].

Some typical operations for modifying geometries are:

- **buffer**, which returns a geometry covering all points within a given distance from all points of an input geometry,

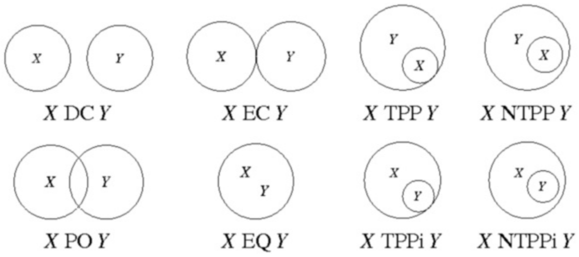


Figure 2.2: RCC8 relations examples using simple geometries, from [1].

- **convex** and **concave hull**,
- **difference**, **intesection**, and **union**.

Some examples of most common analytical functions for measuring geometries and checking their relation are:

- **area**, for calculating the area of the surface of a polygon,
- **distance**, for obtaining the minimum distance between two geometries,
- **length**, for getting the length of a line,
- a set of boolean functions based on topological relations, returning true when two geometries are related the given way (e.g. **contains**, **covers**, **crosses**, **intersects**, **overlaps**), and
- **relate** function, which checks or obtains the topological relation of two geometries in the DE-9IM format.

2.3 Spatial Data Representation

There are some standardized and commonly used formats and serializations for spatial data:

- **WKT, WKB – Well Known Text, Well Known Binary** are classical textual and binary geometry representation. Some examples of WKT are `POINT(0 0)`, `LINESTRING(0 0,1 1,1 2)`, `POLYGON( (0 0,4 0,4 4,0 4,0 0), (1 1, 2 1, 2 2, 1 2,1 1) )`.
- **GeoJSON** [8] is JSON encoding geometric information including additional attributes, e.g.

```
{ "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
        [100.0, 1.0], [100.0, 0.0] ]
      ]
  },
  "properties": {
    "prop0": "value0"
  }
}
```

- **GML – Geography Markup Language** [9] is an XML encoded geometry format. It is richer than the previous formats, as it supports more features, e.g. expressing used coordinate reference systems, units of measure, time, and more geometry types. A simple example[1] is

```
<Feature   fid="142" featureType="school" >
  <Description>Balmoral Middle School</Description>>
  <Property Name="NumFloors" type="Integer" value="3"/>
  <Property Name="NumStudents" type="Integer" value="987"/>
    <Polygon  name="extent" srsName="epsg:27354">
      <LineString  name="extent" srsName="epsg:27354">
        <CData>
          491889,5458046 491905,5458045
          491909,5458065 491925,5458065
          491926,5458080 491978,5458121
          491954,5458018
        </CData>
    </LineString>
  </Polygon>
</Feature>
```

2.4 GIS Services

This section gives some often used standards and systems for GIS servers and services.

2.4.1 OGC Standards for GIS Services

OGC[2] (Open Geospatial Consortium, an international organization for making open geospatial standards) defines several standards of GIS services for different purposes:

- **WMS (Web Map Service)** – a protocol for serving raster maps. A service provides a map image given the requested resolution, CRS, layer, and the bounding box.

[1] From `https://www.w3.org/Mobile/posdep/GMLIntroduction.html`, cit. 2016-06-19.

[2] `http://www.opengeospatial.org/`, cit. 2016-08-28.

- **WCS (Web Coverage Service)** [10] – a protocol for providing coverage data, that can be used either for rendering or for further processing. As opposed to WMS, it provides data with given semantics and scale. It can serve e.g. GML, GeoTIFF.
- **WFS (Web Feature Service)** [11] – a protocol for providing source feature vector data, also allowing data manipulation. It has some querying capabilities.
- **WPS (Web Processing Service)** – a protocol for providing geospatial processing services.

These standards are widely accepted, as most GIS server technologies support them (e.g. ArcGIS, GeoMedia – see Section 2.4.3, and many open source platforms), as well as many client libraries can use them (e.g. Javascript web map viewers, desktop applications, and software libraries).

2.4.2 PostgreSQL, PostGIS

PostgreSQL [12] is an open source DBMS (database management system), originally developed at the University of California at Berkeley. It offers features like foreign keys, triggers, views, transactional integrity, and multi version concurrency control. It is a well-established DBMS. It is used as a classical relational database conforming to SQL.

PostGIS [7] adds support for spatial data to PostgreSQL. It allows efficient storage and retrieval of geographical data in database tables. It provides a binary column type suitable for storing geographical coordinates and many functions to manipulate them. Such geographical columns can be indexed, thus providing fast access to the data while performing a complex search.

PostGIS offers two modes when handling geographical data:

- Geometrical – operates on a plane with Cartesian coordinates. The shortest join of two points is a straight line. It makes calculations a lot easier, e.g. calculations of distances, areas, and intersections.
- Geographical – uses spherical coordinates for points (in angular units) – latitude and longitude, and operates on a sphere or spheroid. The shortest join of two points is an arc. It makes calculations more complex. For calculations being more exact (as the Earth is not a sphere), a spheroid has to be taken into account, which makes the calculations even more complicated.

In most cases, as well in my prototype described in Chapter 5, PostGIS is used in the geometrical mode, since the calculations are faster in this mode (and the accuracy after projection is sufficient for the needs of most cases), and more functions are available in geometrical mode.

Basic spatial objects supported by PostGIS are points, line strings, and polygons, which can be represented in WKB (well-known binary), WKT (well-known text) and other formats (see Section 2.3).

There is a wide variety of functions to manipulate and process spatial data:

- input/output functions – for converting WKT, WKB, GML, KML and other formats to and from internal spatial objects,
- conversion functions – for joining points into a line, constructing a polygon from its borders, etc.,
- accessor functions – for retrieving properties of spatial objects,
- editor functions – for modifying spatial objects (scaling, rotating, transforming, . . .),
- measurement functions – for calculating various facts about spatial objects (distance between two objects, whether one object lies within a given radius of another, whether two objects overlap, etc.),
- and others.

For details and complete reference of all functions and properties of PostGIS, see [7].

2.4.3 Commercial Systems

GeoMedia[3] is a set of commercial GIS tools developed by Intergraph, now Hexagon Geospatial. In the Czech Republic, their systems are used e.g. by State Administration of Land Surveying, South Moravian Region Offices, and Woods of the Czech Republic (Lesy ČR) [13].

ArcGIS[4] developed by Esri is a commercial wide-spread GIS platform used e.g. by IPR Praha (see Section 2.5.2), which publishes its data using this platform via various services [14]. ArcGIS servers can publish its data and services by many interfaces[5], e.g. WMS, WCS, WFS, and WPS services (see Section 2.4.1), a geocoding service (an address locator), mobile device map extracts, raster and mosaic datasets, and web services using REST and SOAP APIs. My prototype presented in Section 5.1 uses the ArcGIS server of IPR Praha.

[3] Available at `http://www.hexagongeospatial.com/products/producer-suite/geomedia`, cit. 2016-07-07.

[4] Available at `http://www.esri.com/software/arcgis`, cit. 2016-07-07.

[5] From `http://server.arcgis.com/en/server/latest/publish-services/linux/what-types-of-services-can-you-publish.htm`, cit. 2016-08-28.

2.5 Interesting Data Sources

Here are some interesting publicly accessible GIS data sources, some of which are used for the prototype testing in Chapter 6.

2.5.1 OpenStreetMap

OpenStreetMap, available at `http://www.openstreetmap.org/`, is a publicly available, extensible, and editable geographical data of the World with complete software infrastructure for rendering raster maps. Everybody can contribute to the data, but the main sources are accessible data from various organizations (governmental, etc.), which conform to the OpenStreetMap license. For details, see [15]. For an idea of the data coverage, see generated raster maps in Fig. 2.3, obtained from the OpenStreetMap website.

The OpenStreetMap data are available in two formats: a custom binary (PBF), and an XML file. The basic OpenStreetMap objects are nodes (spatially represented by a point), ways (ordered lists of points connected by lines), and relations (consisting e.g. of a set of ways, forming a polygon). Every object can be labeled with tags, which are “key=value” pairs. A tag can define the type of an object (e.g. a restaurant, a semaphore, a water surface), or give it an attribute (e.g. a name, an author).

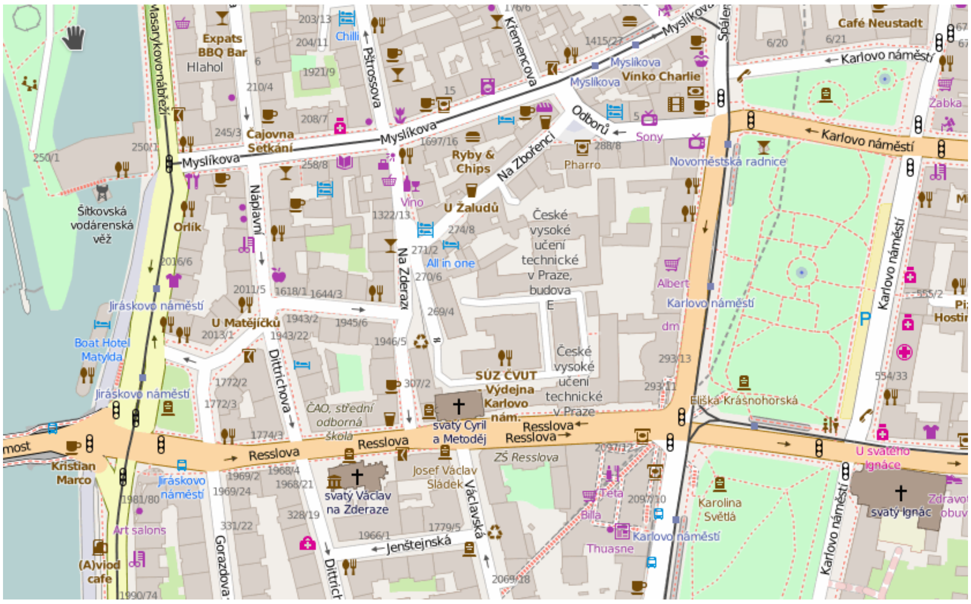


Figure 2.3: OpenStreetMap raster map detail of Karlovo náměstí, Prague, the Czech Republic, where a part of CTU, FEE resides.

Thanks to OpenStreetMap being open, anybody can download source vector data of the map, which most of other publicly available maps (e.g. Google Maps, Mapy.cz) do not offer. There is a utility *osm2pgsql*, which performs the import of OpenStreetMap data into PostGIS database. The data are available e.g. at http://planet.openstreetmap.org/, the full planet in XML format compressed in BZIP2 has approximately 50 GB. As of 2016-06-26, the number of the elements of OpenStreetMap is approximately 3.4 billion nodes and 350 million ways.

2.5.2 Official National Data Sources

Official geographical data sources supported by local governments are becoming more and more freely available, though it has not always been the case. In the United States of America, it has been a tradition to provide all data collected by official agencies for public use [16, 17].

In the European Union, the INSPIRE [3] directive (Infrastructure for Spatial Information in the European Community) started in 2007 is important, as it regulates the EU countries, specifying how and which spatial data should be provided to public. It deals mostly with environmental data – agriculture, geology, atmosphere, land use, etc.

In the Czech Republic, Czech Environmental Information Agency (CENIA) is responsible for compliance with INSPIRE. It runs a geoportal with all the required data available at https://geoportal.gov.cz/. It provides INSPIRE compliant metadata of the data available, and the data itself, both as an interactive web-based map and as a WMS service (see Section 2.4.1). However, vector data are not freely publicly available, they can be purchased.

There is also ČÚZK, State Administration of Land Surveying and Cadastre, which provides cadastral maps of the Czech Republic as a web-based interactive map, a WMS service, and also as a WFS service providing vector data, most of them for free [18].

On the other hand, Czech Hydrometeorological Institute offers its historical data about the weather for sale only.

In this thesis, I focused on another data source – Prague Institute of Planning and Development ("Institut plánování a rozvoje hlavního města Prahy" in Czech, IPR Praha; formerly known as Útvar rozvoje hlavního města Prahy, ÚRM). The institute runs the Prague geoportal available at http://www.geoportalpraha.cz/, providing a lot of interesting data, available both as rasters via a WMS service, and in the vector form via an ArcGIS service, as it runs on an ArcGIS server (see Section 2.4.3).

2.6 Existing GIS Integration Techniques

There are some techniques for defining and searching GIS catalogues. For example, the Catalogue Service (CSW) [19], a standard by OGC, is an interface to discover, browse, and query metadata about GIS data and services. It uses Dublin Core[6] vocabulary to describe web resources. It can be searched by metadata – keywords, author, date, etc. However, it does not allow for more complex queries or semantic search. For spatial querying, only bounding box is supported, and the language for describing the source capabilities is very limited.

A traditional GIS method of using multiple sources is the manual connection to the required GIS services from a desktop PC application, e.g. ArcGIS for Desktop[7], loading the relevant layers from the sources, and performing the integration and interpretation of the results by hand.

For integrating relational databases, database federation techniques [20] are known for a long time. However, they are not suitable for GIS data integration because a GIS integration requires special functions for spatially constraining the data (e.g. by distance to a set of objects, by falling into a specified rectangle, or by other spatial properties), which are not standardized yet in the world of relation databases, and are specific to the used GIS storage implementation, therefore cannot be supported by general data integration tools.

The author of [21] gives a theoretical background for data integration. The paper explains and compares two integration approaches: global as view and local as view. The paper operates with a set of local source schemas, and a global schema (which is used for user queries). In the global as view approach, mappings of elements from the global schema to the source schemas are provided. In the case of the local as view approach, it is the opposite.

In [22], the authors summarize the past in data integration and see the future in the semantic integration.

For existing semantic integration techniques, focusing on geospatial integration, see Section 3.5.

[6] `http://dublincore.org/`, cit. 2014-05-22.

[7] `http://www.esri.com/software/arcgis/arcgis-for-desktop`, cit. 2016-08-28.

Chapter 3

Semantic Web

The idea of the semantic web is motivated by the desire to explore the web more automatically and to obtain more focused results.

The classical approach of searching the web is based on general text and word similarities and synonyms, and using statistics. It is difficult to narrow down the results a user wants, to express the meaning of what is expected. This way, a user usually needs to follow an iterative process of finding the proper set of search keywords to obtain relevant results, which still may contain a lot of irrelevant results, which need to be filtered out manually.

In the case of the semantic web, all provided information is not in the form of a natural human language, but in the form of axioms, which have strictly defined semantics so computers can understand them. The basic form of an axiom is a triple "subject, predicate, object", where each member can be a reference to a symbol (usually represented by a URI), having a well-defined meaning; an object can also be some data value.

Some example pieces of information can be represented by following triples:

- `ex:people/Joe` `dbp:ontology/residence` `dbp:resource/Prague`.
- `dbp:resource/Prague` `dbp:ontology/populationTotal` "1259079".
- `dbp:resource/Prague` `dbp:ontology/country` `dbp:resource/Czech_Republic`.

The prefix `ex:` stands for `http://example.org/`, and `dbp:` stands for `http://dbpedia.org/` (see Section 3.2.3 for details of this data source). Intuitively, they represent the facts that a person having the URI `http://example.org/people/Joe` has the residence in Prague, and that Prague has the total population of 1259079, and it is in the Czech Republic.

The triples form a large graph, where subjects and objects are the nodes, and predicates are the edges. Therefore, when data are represented by such triples, they are usually searched by a graph pattern (see Section 3.4.1).

3.1 Linked Data

Linked Data [23] is a method how technically publish data, along with the principles of the semantic web. There are four basic principles stated in [24]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs so that they can discover more things.

Item 2 is often called "dereferencing." This could be done according to [25]: the URI itself should identify real-world objects, which should not be confused with the documents that describe them. Therefore HTTP content negotiation can be used, with two strategies to follow: either using HTTP response 303 to redirect or using hash URIs.

To give an example of the former strategy, HTTP 303 redirecting, let us get some information about URI `http://example.org/people/Joe`. First, perform an HTTP GET request to exactly that URL, with the appropriate requested content type in the HTTP `Accept:` header, e.g. `application/rdf+xml` or `text/html`. The server responds with the HTTP response code 303 "See Other" with a redirect according to the requested content type, e.g. to `http://example.org/people/Joe.rdf`, respectively `http://example.org/people/Joe.html`, where another GET request would retrieve the description of Joe in the requested format (RDF/XML, see below in this chapter, or HTML).

Note that URI stands for Uniform Resource Identifier, and it is a superset of

- URL (Uniform Resource Locator), the widely used web addresses, e.g. `https://en.wikipedia.org/`, and
- URN (Uniform Resource Name), names of web resources, not very widely used these days, e.g. `urn:isbn:0-521-78176-0`.

In the Czech Republic, for example, is publishing open data, including publishing in linked data form, supported by the action plan for open government [26] from 2014. Open data are also supported e.g. by Fond Otakara Motejla, which produced a guide about open data publishing in [27].

The key feature of Linked Data is that an author of one data set can link its objects to objects in another data set. In my example, the data publisher, who is using the domain `example.org`, who describes Joe, used for stating his birthplace an object of another data publisher, the city Prague published on domain `dbpedia.org` by DBpedia (which produces its data by extracting Wikipedia).

Many interesting data sets in many areas are already publicly available; it is always a good idea to link a new data set to the relevant ones. The LOD cloud[1] (Linking Open Data cloud diagram) in Fig. 3.1 depicts, what well-known datasets are published as Linked Data and which way are they linked.

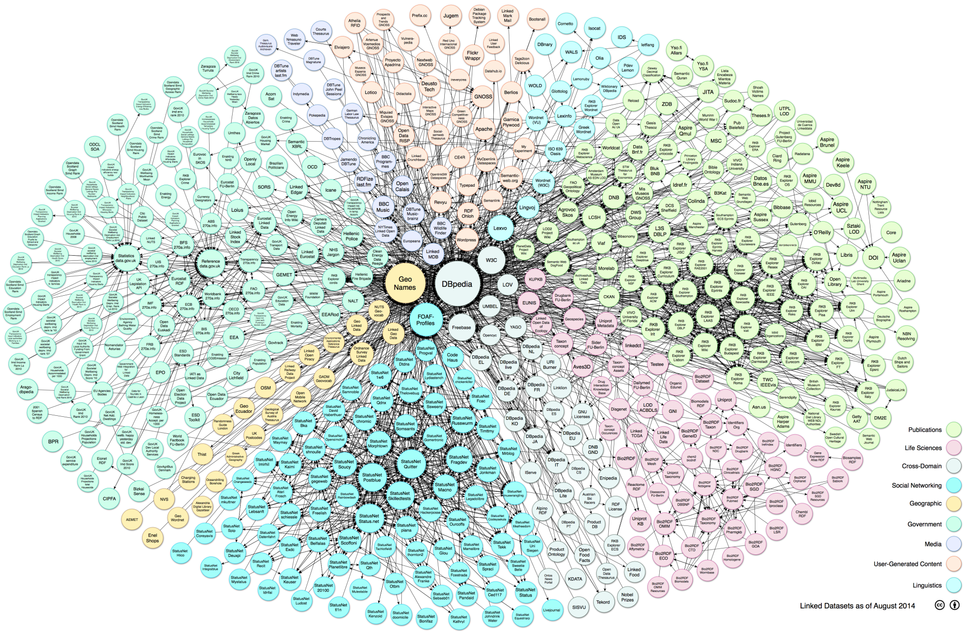


Figure 3.1: LOD cloud – datasets published in Linked Data format, updated on 2014-08-30.

RDF [28] (Resource Description Framework) is a model for data interchange suitable for the semantic web. It is the primary semantic language for Linked Data. Along with RDFS [29] (RDF Schema), it allows basic deduction via reasoning inference. RDF and RDFS define concepts “resource”, “class”, “property”, and “literal”. A resource is anything represented by a URI. The class of all resources is denoted `rdfs:Resource`. Similarly, the class of all literal values (strings, integers, ...) is `rdfs:Literal`. A class is a group of resources. The class of all classes is `rdfs:Class`. A property is a binary relation between resources and another resources or literals. The class of all properties is `rdf:Property`. The property `rdf:type` states that a resource is an instance of a class. See Listing 3.1 for examples, how these concepts can be used.

Note that lines 1 and 2 are redundant: 1 can be inferred from 3 (domain of `rdf:type` is `rdfs:Resource`), and also 2 can be inferred from 3 (range of `rdf:type` is `rdfs:Class`).

[1] Attribution: Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. `http://lod-cloud.net/`

Listing 3.1: An example of RDF triples.

```
ex:people/Joe rdf:type rdfs:Resource.
dbp:ontology/Engineer rdf:type rdfs:Class.
ex:people/Joe rdf:type dbp:ontology/Engineer.
dbp:ontology/residence rdf:type rdf:Property.
ex:people/Joe dbp:ontology/residence dbp:resource/Prague.
```

Now comes the fun part – the RDFS properties, that allow inferring new interesting facts:

- `rdfs:subClassOf` states that all instances of one class are automatically instances of another class,
- `rdfs:subPropertyOf` states that all pairs of resources related by one property are automatically related by another property,
- `rdfs:domain` states that all subjects of a property are automatically instances of a class, and
- `rdfs:range` states that all objects of a property are automatically instance of a class.

Thus, I can add the triples in Listing 3.2 to those in Listing 3.1.

Listing 3.2: Another example of more RDFS triples.

```
dbp:ontology/Engineer rdfs:subClassOf dbp:ontology/Person.
dbp:ontology/residence rdfs:subPropertyOf dul:hasLocation.
dbp:ontology/residence rdfs:range dbp:ontology/Place.
```

They state that (1) all engineers are persons, (2) when something has a residence somewhere, it is located there (a property defined in another data source), and (3) when something is a residence of somebody, it is a place. So thanks to the triples in Listing 3.2, it can be inferred from Listing 3.1 that: Joe is a person, Joe is located in Prague, and Prague is a place.

Another widely used RDFS property is `rdfs:label`, giving a human-readable label to a resource.

Data in RDF can be serialized in a few formats:

- N-Triples [30], a simple triple-per-line format, suitable for large data files for parallel processing,
- Turtle [31], a superset of N-Triples, which is more compact, and more human readable,
- Notation3 (N3) [32], a superset of Turtle, going even beyond RDF model,
- JSON-LD [33], a JSON-based format for Linked Data, capable of serializing any RDF graph, and

- RDF/XML [34], a widely used XML syntax; see Listing 3.3 for an example.

Listing 3.3: An RDF/XML example.

```
<rdf:RDF xmlns="http://example.org/"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xmlns:dbo="http://dbpedia.org/ontology/">
  <rdf:Description rdf:about="http://example.org/people/Joe">
    <rdf:type rdf:resource="http://dbpedia.org/ontology/Engineer"/>
    <dbo:residence rdf:resource="http://dbpedia.org/resource/Prague"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/ontology/Engineer">
    <rdfs:subClassOf rdf:resource="http://dbpedia.org/ontology/Person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/ontology/residence">
    <rdfs:subPropertyOf rdf:resource=
      "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#hasLocation"/>
    <rdfs:range rdf:resource="http://dbpedia.org/ontology/Place"/>
  </rdf:Description>
</rdf:RDF>
```

3.2 Interesting Semantic Spatial Data Sources

There are many data sources conforming to Linked Data, see Fig. 3.1. I will pick and describe a few of them, which are relevant to this thesis – the ones containing some sort of spatial information. They are used for my prototype in Chapter 5.

3.2.1 LinkedGeoData

LinkedGeoData [35], available at `http://linkedgeodata.org/`, are the data of OpenStreetMap (see Section 2.5.1) presented as Linked Data. The original data are for example in XML format, tagged with a defined set of labels, categorizing each element into a type. LinkedGeoData effort uses the tags to produce RDF classes and properties to describe the data and also links the classes and properties to other knowledge bases in the Linking Open Data initiative.

LinkedGeoData contains many classes in good hierarchical organization, with some erroneous entries, a small amount of object properties (not in any hierarchy), and quite many data properties, which contain some errors as well (it is because of the nature how OpenStreetMap works, and that LinkedGeoData is an automated extract of what OpenStreetMap contains; users of OpenStreetMap upload nodes and ways and label them with tags of their will; they are only advised to use the predefined set), and not in any hierarchy, too. Unluckily it does not contain many label annotations (just a few in a few languages).

The LinkedGeoData ontology (the hierarchy of its classes and properties) is unfortunately not updated recently (also [36] is not accessible recently). The last version available[2] is from 2014-09-09. Besides class and property definitions, it contains some of large volumes of actual OpenStreetMap data with its geometries in RDF in N-Triples format (see Section 3.1). The geometries are encoded in WKT literals (Section 2.3) referenced to by a GeoSPARQL property (Section 3.4.2).

It has a small amount of links to DBpedia (see Section 3.2.3), but it is quite a popular destination of links from other Linked Data sources. I will refer to entities from the LinkedGeoData ontology by the prefix `lgd:`.

3.2.2 GeoNames

GeoNames [37], available at `http://www.geonames.org/` is another source published as Linked Data. It is a geographical database of points, having about 9 million features. These features are categorized into nine classes and marked with one of 645 feature codes. Besides downloading complete data in tab-separated text file, they can be accessed using restful web services and viewed and edited on a web page with Google Maps.

One of the key advantages of GeoNames is it has names of the features typically in many languages. Its sources of data are mostly already existing data from various governments:

- National Geospatial-Intelligence Agency's (NGA), U.S. Geological Survey Geographic Names Information System, www.geobase.ca, gtopo30, ...
- In the Czech Republic: FreeGeodataCZ – project based on GRASS GIS at Czech Technical University, Faculty of Civil Engineering.

However, any user can add features using the GeoNames web page.

A part of GeoNames is its ontology[3], containing all classes and feature codes, which are very well defined and labeled in a few languages. There are also the links from the classes and feature codes to other data sources, e.g. to DBpedia and LinkedGeoData. It is also a popular target to link to. Besides the definition of feature classes and codes, the ontology contains several properties organized in simple but useful hierarchies, e.g. `neighbours`, `nearby`, `name`, `postalCode`, and `population`.

GeoNames use FOAF[4] and SKOS[5] vocabularies. For stating a point coordinates, it uses the W3C Basic Geo Vocabulary [38], a simple vocabulary for expressing WGS 84 coordinates (latitude and longitude; widely used by GPS) of points only. It consists of classes `SpatialThing`, `Point`, and data properties `lat`, `long`, `alt`, `location`, `lat_long`. For an example of

[2] Available at `http://downloads.linkedgeodata.org/releases/2014-09-09/2014-09-09-ontology.sorted.nt.bz2`, accessed on 2016-07-04

[3] Available at `http://www.geonames.org/ontology/documentation.html`, accessed on 2016-07-04.

[4] `http://www.foaf-project.org/`, accessed on 2016-07-04.

[5] `http://www.w3.org/2004/02/skos/`, accessed on 2016-07-04.

description of Prague see Listing 3.4, where many alternate names in different languages and XML namespaces are skipped for compactness. In the rest of thesis, the `gn:` prefix will refer to the GeoNames ontology entities.

Listing 3.4: A snippet of GeoNames description of Prague RDF/XML.

```
<rdf:RDF>
  <gn:Feature rdf:about="http://sws.geonames.org/3067696/">
    <rdfs:isDefinedBy
      rdf:resource="http://sws.geonames.org/3067696/about.rdf"/>
    <gn:name>Prague</gn:name>
    <gn:officialName>Praha</gn:officialName>
    <gn:officialName xml:lang="de">Prag</gn:officialName>
    <gn:officialName xml:lang="en">Prague</gn:officialName>
    <gn:officialName xml:lang="fr">Prague</gn:officialName>
    <gn:officialName xml:lang="cs">Praha</gn:officialName>
    <gn:alternateName xml:lang="hu">Prga</gn:alternateName>
    <gn:alternateName xml:lang="eo">Prago</gn:alternateName>
    <gn:featureClass rdf:resource="http://www.geonames.org/ontology#P"/>
    <gn:featureCode rdf:resource="http://www.geonames.org/ontology#P.PPLC"/>
    <gn:countryCode>CZ</gn:countryCode>
    <gn:population>1165581</gn:population>
    <wgs84_pos:lat>50.08804</wgs84_pos:lat>
    <wgs84_pos:long>14.42076</wgs84_pos:long>
    <gn:parentFeature rdf:resource="http://sws.geonames.org/3067695/"/>
    <gn:parentCountry rdf:resource="http://sws.geonames.org/3077311/"/>
    <gn:parentADM1 rdf:resource="http://sws.geonames.org/3067695/"/>
    <gn:nearbyFeatures
      rdf:resource="http://sws.geonames.org/3067696/nearby.rdf"/>
    <gn:locationMap
      rdf:resource="http://www.geonames.org/3067696/prague.html"/>
    <gn:wikipediaArticle
      rdf:resource="http://en.wikipedia.org/wiki/Prague"/>
    <rdfs:seeAlso rdf:resource="http://dbpedia.org/resource/Prague"/>
  </gn:Feature>
  <foaf:Document rdf:about="http://sws.geonames.org/3067696/about.rdf">
    <foaf:primaryTopic rdf:resource="http://sws.geonames.org/3067696/"/>
    <cc:license rdf:resource="http://creativecommons.org/licenses/by/3.0/"/>
    <cc:attributionURL rdf:resource="http://sws.geonames.org/3067696/"/>
    <cc:attributionName>GeoNames</cc:attributionName>
    <dcterms:created>2006-01-15</dcterms:created>
    <dcterms:modified>2013-11-25</dcterms:modified>
  </foaf:Document>
</rdf:RDF>
```

3.2.3 DBpedia

DBpedia [39], available at `http://wiki.dbpedia.org/`, is the center point of LOD Cloud in Fig. 3.1, as it is a large data source on a very wide range of topics since it is an extract of Wikipedia. The English version of it currently describes cca. 4.5 million things, including cca. 1.4 million persons and 730 thousand places. There are also versions in different languages, mutually interlinked. It is linked to other Linked Data sources via approx. 50 million links. In 2014, it contained around 3 billion triples in total.

The examples in Chapter 3 and Section 3.1 use some of the DBpedia properties. It is a good idea to link new data sources to DBpedia classes, properties, and resources, as also many other sources do so, thus providing linkage to many sources with minimal effort.

It contains simple geospatial data about places – points expressed in W3C Basic Geo Vocabulary [38].

3.3 Web Ontology Languages

When describing more complex knowledge than simply stating there exist some objects, somehow related, RDF and RDFS presented in Section 3.1 might not be enough. Imagine you want to state that when someone has a residence somewhere, the place is related to the person also via "has citizen" relation (the inverse relation). So if some data source contains the fact that Joe has residence in Prague, when someone asks it "Who are the citizens of Prague?", the source would correctly respond with Joe, even though this fact is not explicitly stated. Moreover, many more queries, more complex than this one, can be useful.

Allowing to state more complex pieces of knowledge and allowing to perform more complex searches has given the motivation to develop a new family of languages: Web Ontology Languages abbreviated as OWL. They take some techniques from RDF and RDS, but they have richer semantics, having the background in description logics, which are presented in Section 3.3.1. The set of W3C standards forming OWL is presented in Section 3.3.2.

As this thesis focuses on a profile of OWL, OWL 2 QL, I will also present the description logic $DL\!-\!Lite$ in Section 3.3.3, as this one gives semantic background to OWL 2 QL. Finally, OWL 2 QL is presented in Section 3.3.4.

3.3.1 Description Logics

Description logics is a family of languages based on formal logics used for knowledge representation. Most of them are subsets of first-order predicate logic. The key feature for their usability is the balance between expressive power and decidability together with lower computational complexity.

A good initial resource about description logics is [40].

There are some $n-$ary logics (using predicates of arbitrary arity), but the most widely used description logics are restricted to binary predicates. No functions are allowed, except constants (zero-arity functions).

The basic description logic terminology is:

- **concept** – a unary predicate (a class in RDF),
- **role** – a binary predicate (a property in RDF),
- **individual** – a constant.

A specific description logic can be characterized by the operators allowed, which affects both its expressive power and the computational complexity.

There are letter symbols to denote allowed operators. First, there are three base logics, on top of which one can build more complex logics:

- $\mathcal{AL}$ – attributive language, allowing atomic concept negation, concept intersection, universal value restriction, and limited existential quantification,
- $\mathcal{FL}$ – frame-based language, allowing concept intersection, universal value restriction, limited existential quantification, and role restriction,
- $\mathcal{EL}$ – allowing concept intersection, and full existential quantification.

These logics can be enriched by the following features:

- $\mathcal{F}$ – functional properties,
- $\mathcal{U}$ – union of concepts,
- $\mathcal{E}$ – full existential quantification,
- $\mathcal{C}$ – full concept negation,
- $\mathcal{N}$ – unqualified number restrictions,
- $\mathcal{Q}$ – qualified number restrictions,
- $\mathcal{O}$ – nominals,
- $\mathcal{I}$ – role inverse,
- $\mathcal{H}$ – role hierarchy,
- $\mathcal{R}$ – complex role inclusion, and others.

Then there is the alias $\mathcal{S}$ for $\mathcal{ALC}$ with transitive roles. A very nice summary of the computational complexity of selected description logics is in [41]. The description of the semantics of all the mentioned constructs is out of the scope of this thesis; for more see [40].

A single fact in description logics is called an axiom. Axioms can be divided into two groups:

- **TBox** (terminological axioms) – relationships and restrictions on concepts and roles; the axioms can use available operators.
- **ABox** (assertional axioms) – concept assertions (an individual is a member of a concept) and role assertions (two individuals are related by a role).

A TBox and an ABox together are denoted as a knowledge base.

3.3.2 OWL – Web Ontology Language

Web Ontology Language (OWL), currently in version OWL 2 [42], is a group of ontology languages for the Semantic Web having the semantic background in description logics (see Section 3.3.1).

It uses different basic vocabulary than description logics (DL), being closer to the one used in RDF:

- **classes** for DL concepts,
- **properties** for DL roles,
- **individuals**, and
- **data values** for RDF literals.

Instead of URIs used in RDF, OWL uses IRIs – Internationalized Resource Identifiers, which allow international characters from Universal Character Set. OWL has several standard serializations:

- **RDF/XML** – OWL can be serialized as RDF/XML, this format is often used for interchange,
- **OWL/XML** – there is also a tailored XML format specifically for OWL,
- **Turtle** – also Turtle can be used for serializing OWL,
- **Manchester syntax** – more human-readable, compact form of OWL serialization,
- **Functional syntax** – a syntax more following the structure of ontologies.

There are two alternative ways how to assign meaning to OWL 2 ontologies:

- **The Direct Semantics** assigns meaning directly to axioms. It is compatible with the model theoretic semantics of the $\mathcal{SROIQ}$ description logic. Some restrictions must be applied to the syntax to fit the $\mathcal{SROIQ}$ DL, in which case it is called an OWL 2 DL ontology.
- **RDF-Based Semantics** assigns meaning to an RDF graph produced by an OWL 2 ontology via a mapping. This semantics is fully compatible with the RDF semantics. No restrictions on the ontology are necessary.

The computational complexity of OWL 2 is rather high[6]:

- **RDF-Based Semantics** is undecidable in all aspects,
- consistency checks and instance checking using the **Direct Semantics** is N2ExpTime-complete (nondeterministic, double exponential time complexity) in taxonomic complexity (complexity measured with respect to the size of a TBox), and decidable, but complexity unknown (known to be at least NP-hard), in data complexity (complexity measured with respect to the size of an ABox), and

[6] `https://www.w3.org/TR/owl2-profiles/#Computational_Properties`, accessed on 2016-07-05

- complexity of conjunctive query answering using the **Direct Semantics** is an open problem (even decidability is not known).

Therefore ontologies using the full expressive power of OWL 2 cannot be too large, to allow reasonable query response times. It is useful in case of highly specialized, compact, and complex knowledge; particularly it found usage in the medical domain. Using it for large data, like Linked Data and geographical sources is practically impossible. Partially because the complex reasoning algorithms for OWL query answering usually require having the ontology stored in computer memory, which might not be possible.

OWL properties are split into two groups: object properties and data properties. An object property links an individual to another individual, a data property gives an individual a data value. A property name cannot be used for both an object property assertion and a data property assertion, as it would complicate reasoning. However, using the same name (IRI) for two different types of entities, called punning, is allowed in OWL in some cases.

There are also annotation properties, which can be used much more freely. They can be used for annotating classes, properties, individuals, and even axioms. The difference of annotation properties from standard properties is that a simpler reasoning is performed over them, i.e. only the sub-property, domain, and range axioms are supported for annotation properties. For details, see [42].

3.3.3 $DL\!-\!Lite$

$DL\!-\!Lite$ is a member of description logics family, being one of the least expressive ones. This brings the advantage of that it can be stored in an RDBMS (relational database management system), where also queries can be evaluated using standard SQL language. This is a very modern approach, because most of the other languages, though providing higher expressive power, are hardly usable for large datasets. This is a typical case on the Internet, where an ontology is about to store vast amounts of facts.

$DL\!-\!Lite$ family was originally defined in [43, 44, 45], then extended in [2]. $DL\!-\!Lite$ is a family of a few variants, one of which – $DL\!-\!Lite^{\mathcal{H}}_{core}$ – is standardized by W3C as OWL 2 QL [46] (see Section 3.3.4), the widely accepted tractable profile of OWL 2.

Because of the application for the Internet, I need feasible worst-case complexities. In order to have an efficient reasoning, the time complexity of queries with respect to ABox should be as efficient as possible, preferably polynomial in time (polytime), because the sizes of ABoxes are usually very big on the Internet. Also, the space complexity should be logarithmic (logspace). Note that for these complexity classes the following is valid: logspace $\subseteq$ polytime. These restrictions make it possible to reformulate a query into the SQL language. Other languages that have higher query answering complexity are practically unusable for domains with many instances the Internet offers.

Another advantage of this approach becomes obvious when I compare memory management during query answering. $DL\!-\!Lite$ reformulates a query from e.g. SPARQL (see Section 3.4.1) into an SQL query, which is evaluated by an underlying RDBMS, keeping the ABox in a persistent storage (e.g. a hard drive), only operating on indices of database tables (typically B-trees); just the individuals featuring in the final answer need to be read from the DB. Another key advantage is that SQL query answering is a well-settled problem, with many optimizations implemented in most modern DBMS's, making it very robust and efficient. Whereas most of the reasoners for OWL load the whole ABox into memory, where they perform the whole reasoning process. The efficiency of the process is very dependent on the optimizations the reasoner's authors used, and hence it may take some time for the reasoner to "evolve" into a state suitable for practical applications. A bigger issue of this approach is the memory requirements since it is usually very demanding to store complete ABox of possibly many gigabytes into a computer RAM memory.

The query to SQL translation in query answering problem is possible by query reformulation described in [43]: a query can be translated using a TBox to an ABox-only query of polynomial complexity (therefore e.g. an SQL query).

To summarize, if we succeed to fit an ontology into a DBMS where it could also be queried, we gain a few advantages. The system would efficiently permanently store large ABoxes in a DB, which is a well-established way of storing strictly organized data. Query answering keeps an ABox in the DB – no need to transfer it into the memory, which would be infeasible because of the memory size restriction anyway. Query answering is inherently of polytime complexity, which is another "must" for large datasets. Let us see, what the languages $DL{-}Lite$ satisfying all these conditions can express, as presented in [45, 44, 43].

The basic version of $DL{-}Lite$ is $DL{-}Lite_{core}$, which satisfies all the aforementioned requirements, is already more expressive than RDF and RDFS, but still can be expanded, while preserving tractability. $DL{-}Lite_{core}$ constructs for defining concepts and roles are:

- $B ::= A|\exists R$
- $C ::= B|\neg B$
- $R ::= P|P^{-}$

where A denotes a concept name, B a basic concept, and C a general concept. Symbol P denotes a role name and R a complex role.

The semantics is defined by an interpretation $\mathcal{I}$:

$$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), \tag{3.1}$$

where $\Delta^{\mathcal{I}}$ is a nonempty interpretation domain and $\cdot^{\mathcal{I}}$ is an interpretation function, that assigns each individual to an element of $\Delta^{\mathcal{I}}$, each named concept to a subset of $\Delta^{\mathcal{I}}$, and each named role to a binary relation over $\Delta^{\mathcal{I}}$.

The semantics of the used constructs are defined in Table 3.1.

Table 3.1: Constructs used in $DL\text{--}Lite$ and their semantics.

Syntax	Semantics	Comment
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	concept name
P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	role name
P^{-}	$(P^{-})^{\mathcal{I}} = \{(b,a) \vert (a,b) \in P^{\mathcal{I}}\}$	inverse of a role
$\exists R$	$(\exists R)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \vert \exists b : (a,b) \in R^{\mathcal{I}}\}$	existential quantification
$\neg B$	$(\neg B)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$	negation of a basic concept
$\neg R$	$(\neg R)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$	negation of a basic role

Then a TBox can be defined by inclusion axioms of the form:

- $B \sqsubseteq C$, interpreted by $\mathcal{I}$ as $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$.

An ABox consists of the following assertion axioms:

- concept assertion $A(a)$, interpreted by $\mathcal{I}$ as $a^{\mathcal{I}} \in A^{\mathcal{I}}$, and
- property assertion $P(a,b)$, interpreted by $\mathcal{I}$ as $\left(a^{\mathcal{I}}, b^{\mathcal{I}}\right) \in P^{\mathcal{I}}$,

where a, b are individuals interpreted by $\mathcal{I}$ as $a^{\mathcal{I}}, b^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

In the following text, the abbreviation UNA stands for Unique Name Assumption, which is the following condition:

$$a^{\mathcal{I}} \neq b^{\mathcal{I}} \text{ for all } a \neq b, \tag{3.2}$$

where a, b are object names.

The $DL\text{--}Lite_{core}$ can be extended in three different directions. One of them is allowing more complex axioms of TBox. The extension of TBox axioms denoted as *Krom* allows for negation to be on the left-hand side of an inclusion axiom, thus in addition to $DL\text{--}Lite_{core}$ axioms there can be:

$$C \sqsubseteq B. \tag{3.3}$$

Another extension of TBox axioms is denoted as *Horn*, which allows conjunction of basic concepts to appear on the left-hand side of an inclusion axiom, which is:

$$\bigsqcap_{k} B_k \sqsubseteq B. \tag{3.4}$$

Finally the most expressive extension in this direction is denoted as *bool*, which allows for inclusion axioms with general concepts on both sides:

$$C_1 \sqsubseteq C_2, \tag{3.5}$$

where general concepts C_1, C_2 can be constructed quite generally – in addition to $DL\text{--}Lite_{core}$, also as a negation of another general concept, or conjunction of two general concepts. For details, see [2].

Another direction of extensions is allowing role hierarchies. This is denoted as letter $\mathcal{H}$, i.e. $DL\text{--}Lite_{\alpha}^{\mathcal{H}}$, where $\alpha \in \{\text{core}, \text{krom}, \text{horn}, \text{bool}\}$. Thus the TBox axioms can be extended with

$$R_1 \sqsubseteq R_2. \tag{3.6}$$

The third direction of extensions concerns number restrictions. One option is the functional restriction denoted as $DL{-}Lite^{\mathcal{F}}_{\alpha}$, which allows a basic concept to be defined as the cardinality restriction $\geq qR$, but where either $q = 1$, or for $q = 2$ the cardinality restriction must be of the form $\geq 2R \sqsubseteq \bot$; no other versions are possible. The most expressive version in this direction is allowing all cardinality restriction, denoted as $DL{-}Lite^{\mathcal{N}}_{\alpha}$.

A simple combination of cardinality restrictions and role hierarchies, $DL{-}Lite^{\mathcal{HF}}_{\alpha}$ or $DL{-}Lite^{\mathcal{HN}}_{\alpha}$, even for $\alpha =$ core, increases complexity. However, the combination can be restricted, which gives the version $DL{-}Lite^{(\mathcal{HN})}_{\alpha}$, which along with the restrictions adds some new features (e.g. role disjointness, (a)symmetry and (ir)reflexivity constraints), without affecting complexity. Another extension is $DL{-}Lite^{(\mathcal{HN})+}_{\alpha}$, which adds transitivity constraint. For details see [2].

Here are some examples, of what $DL{-}Lite_{core}$ can express:

- is-a relationship: concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$,
- disjoint concepts: $A_1 \sqsubseteq \neg A_2$,
- typed roles: $\exists P \sqsubseteq A_1$, $\exists P^- \sqsubseteq A_2$,
- mandatory participation in a role (all instances of a concept participates in a role): $A \sqsubseteq \exists P$, $A \sqsubseteq \exists P^-$,
- prohibited participation in a role: $A \sqsubseteq \neg\exists P$, $A \sqsubseteq \exists P^-$,
- functional restriction of roles: $(\geq 2R \sqsubseteq \bot)$, $(\geq 2R^- \sqsubseteq \bot)$ (only in $DL{-}Lite^{\mathcal{F}}_{\alpha}$ and $DL{-}Lite^{\mathcal{N}}_{\alpha}$).

Fig. 3.2 [2] depicts relations between various versions of $DL{-}Lite$ languages, with the arrows meaning “is extended by”.

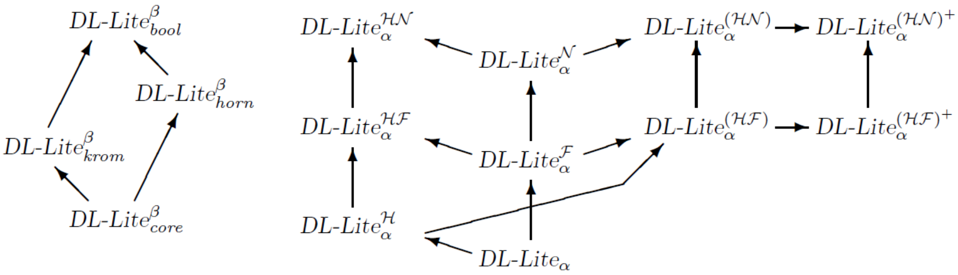

DL-Lite$^{\beta}_{bool}$
DL-Lite$^{\beta}_{horn}$
DL-Lite$^{\beta}_{krom}$
DL-Lite$^{\beta}_{core}$
DL-Lite$^{\mathcal{HN}}_{\alpha}$
DL-Lite$^{\mathcal{N}}_{\alpha}$
DL-Lite$^{(\mathcal{HN})}_{\alpha}$
DL-Lite$^{(\mathcal{HN})+}_{\alpha}$
DL-Lite$^{\mathcal{HF}}_{\alpha}$
DL-Lite$^{\mathcal{F}}_{\alpha}$
DL-Lite$^{(\mathcal{HF})}_{\alpha}$
DL-Lite$^{(\mathcal{HF})+}_{\alpha}$
DL-Lite$^{\mathcal{H}}_{\alpha}$
DL-Lite$_{\alpha}$

Figure 3.2: Language inclusions in the extended $DL{-}Lite$ family, from [2].

In Fig. 3.3 [2], data complexities of query answering under the UNA of different versions of $DL{-}Lite$ languages are displayed. Note that AC^0 is a complexity class defined by digital circuits, consisting of all families of circuits of depth $O(1)$ and polynomial size, with unlimited-fanin (number of

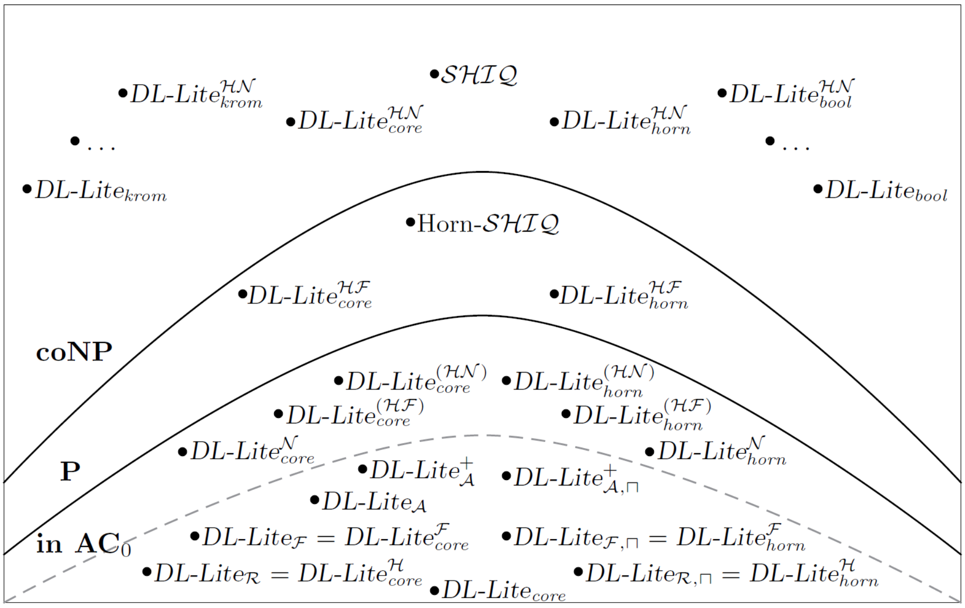

$\bullet\mathcal{SHIQ}$
$\bullet$*DL-Lite*$^{\mathcal{HN}}_{krom}$
$\bullet$*DL-Lite*$^{\mathcal{HN}}_{core}$
$\bullet$*DL-Lite*$^{\mathcal{HN}}_{horn}$
$\bullet$*DL-Lite*$^{\mathcal{HN}}_{bool}$
$\bullet\dots$
$\bullet\dots$
$\bullet$*DL-Lite*$_{krom}$
$\bullet$*DL-Lite*$_{bool}$
$\bullet$Horn-$\mathcal{SHIQ}$
$\bullet$*DL-Lite*$^{\mathcal{HF}}_{core}$
$\bullet$*DL-Lite*$^{\mathcal{HF}}_{horn}$
coNP
$\bullet$*DL-Lite*$^{(\mathcal{HN})}_{core}$
$\bullet$*DL-Lite*$^{(\mathcal{HN})}_{horn}$
$\bullet$*DL-Lite*$^{(\mathcal{HF})}_{core}$
$\bullet$*DL-Lite*$^{(\mathcal{HF})}_{horn}$
$\bullet$*DL-Lite*$^{\mathcal{N}}_{core}$
$\bullet$*DL-Lite*$^{\mathcal{N}}_{horn}$
P
$\bullet$*DL-Lite*$^{+}_{\mathcal{A}}$
$\bullet$*DL-Lite*$^{+}_{\mathcal{A},\sqcap}$
$\bullet$*DL-Lite*$_{\mathcal{A}}$
in AC$_0$
$\bullet$*DL-Lite*$_{\mathcal{F}}$ = *DL-Lite*$^{\mathcal{F}}_{core}$
$\bullet$*DL-Lite*$_{\mathcal{F},\sqcap}$ = *DL-Lite*$^{\mathcal{F}}_{horn}$
$\bullet$*DL-Lite*$_{\mathcal{R}}$ = *DL-Lite*$^{\mathcal{H}}_{core}$
$\bullet$*DL-Lite*$_{core}$
$\bullet$*DL-Lite*$_{\mathcal{R},\sqcap}$ = *DL-Lite*$^{\mathcal{H}}_{horn}$

Figure 3.3: Landscape of data complexity of *DL−Lite*-related logics, from [2].

inputs) *AND* gates and *OR* gates, allowing *NOT* gates only at the inputs. The hierarchy of some used complexity classes is as follows:

$$\mathrm{AC}^0 \subsetneq \mathrm{LogSpace} \subseteq \mathrm{NLogSpace} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{ExpTime}. \tag{3.7}$$

The versions corresponding to the original *DL−Lite* family, as defined in [43, 44, 45], are concentrated below the dashed line.

Fig. 3.4 [2] summarizes complexities for all versions of extended *DL−Lite* family, both for combined complexity of satisfiability and data complexity of instance checking.

3.3.4 OWL 2 QL

OWL 2 QL [46] is a profile of the Web Ontology Language (OWL). The key feature is its tractability (along with other OWL 2 profiles) traded for expressiveness, which is lower compared e.g. to OWL 2 DL with the Direct Semantics. The tractability brings the advantage that queries can be reformulated into SQL and thus RDBMSs can be used as OWL 2 QL storage.

There are other OWL 2 profiles, which are polynomial for ABox – OWL 2 RL and OWL 2 EL. OWL 2 RL is based on rule-based reasoning, which requires many syntactic restrictions to be sound and complete. However, the semantics is not the same as for OWL. The restrictions are designed for example to avoid the need to infer the existence of individuals not explicitly stated. OWL 2 EL data complexity class is higher than the complexity AC^0 of OWL 2 QL, and it does not support e.g. concept negation (which is, in limited form, supported by OWL 2 QL). As my design and prototype need

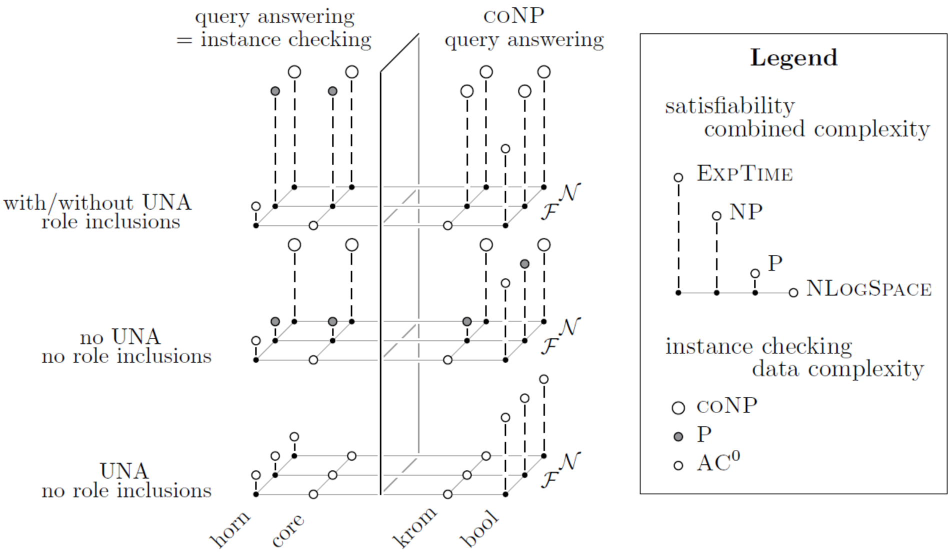


Figure 3.4: Complexity of basic *DL–Lite* logics, from [2].

logical negation, both OWL 2 EL and OWL 2 RL were not used (rule-based inference does not allow logical negation, only negation-on-failure).

Consistently with OWL, OWL 2 QL uses the terms "class" for DL concept, and "property" for DL role. I will use the OWL terminology in the rest of this thesis.

OWL 2 QL is based on $DL\text{–}Lite^{\mathcal{H}}_{core}$. Therefore the supported axioms are:

$$B \sqsubseteq C\,, \qquad R_1 \sqsubseteq R_2\,, \qquad A(a)\,, \qquad P(a,b)$$

subject to

$$B ::= A \mid \exists R\,, \qquad C ::= B \mid \neg B\,, \qquad R ::= P \mid P^{-}\,,$$

where A denotes a class name, B a basic class, and C a general class. Symbol P denotes a named property, and R (optionally subscripted) a complex property. Symbols a and b represent individuals. For the semantics of the axioms, see Section 3.3.3.

OWL 2 QL extends *DL–Lite* with various features not affecting its tractability, e.g. data properties and annotations.

3.4 Querying Languages

This section introduces two semantic query languages: SPARQL is a widely used standard for querying RDF data, while GeoSPARQL is an extension of SPARQL supporting geospatial operations.

3.4.1 SPARQL

SPARQL, currently in version 1.1 [47], superseding SPARQL 1.0 [48], is a query language for RDF, capable of querying required and optional graph patterns along with their conjunctions and disjunctions.

The structure of the language resembles SQL. The basic query commands are:

- SELECT: returns all possible bindings of a set of variables,
- CONSTRUCT: returns a single RDF graph specified by a graph template,
- ASK: answers, whether or not a query pattern has a solution,
- DESCRIBE: returns a single result RDF graph containing RDF data about resources.

The results of a query are constrained by a WHERE section, which provides the basic graph pattern to match against the data graph. The graph pattern consists of a set of triples, which form a graph, where an element of a triple can be either a resource URI, a data value (in the case of the triple object), or a variable. A variable is prefixed with the question mark, and can appear in the SELECT section (the output variable). The WHERE section can also contain FILTER clauses, which filter e.g. string and numeric values using various functions (predicates).

SPARQL 1.1 also supports optional parts of query graph with the OPTIONAL keyword; they may be or may be not satisfied, and therefore the variables contained in them may be bound or not. It also supports the GROUP BY and HAVING keyword, which aggregates and filters groups of results, similarly to SQL.

A simple example of a SELECT query is in Listing 3.5, with the corresponding response in Table 3.2, using the example data presented earlier in this chapter. Notice that the "a" in line 5 in Listing 3.5 is an alias for `rdf:type`. A response to a SPARQL query can be formatted in XML, JSON, CSV, and TSV.

Listing 3.5: An example of SAPRQL query.

```
1 PREFIX dbo: <http://dbpedia.org/ontology/>
2 SELECT ?eng ?bigcity
3 WHERE {
4   ?eng dbo:residence ?bigcity.
5   ?eng a dbo:Engineer.
6   ?bigcity dbo:populationTotal ?population.
7   FILTER (?population > 1000000).
8 }
```

It supports federated queries by explicitly delegating a certain subquery to a different SPARQL endpoint via the SERVICE keyword. Similarly to SQL, SPARQL 1.1 also supports updating data in a data source by the INSERT

Table 3.2: The response to the example SPARQL query.

eng	bigcity
`<http://example.org/people/Joe>`	`<http://dbpedia.org/resource/Prague>`

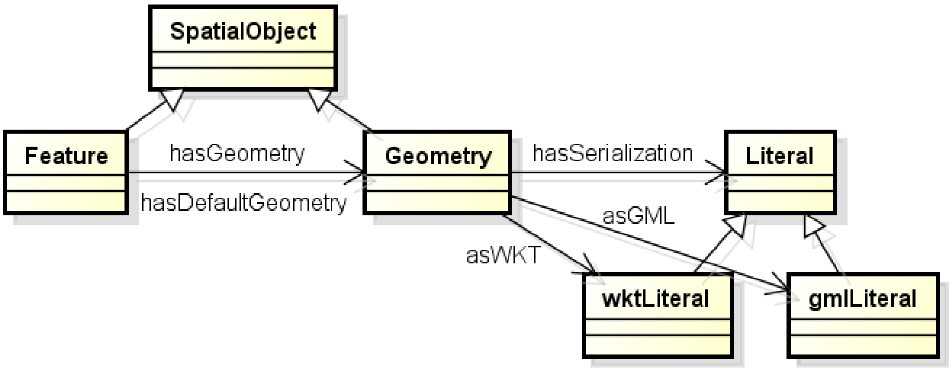


Figure 3.5: GeoSPARQL basic classes and properties.

DATA, DELETE DATA, LOAD, CLEAR, etc. keywords. It also supports different entailment regimes. For more details, see [47].

SPARQL is well settled and widely used language for querying RDF ontologies, but it lacks necessary constructs for querying spatial data. Luckily, during the work on this thesis, a new standard gradually emerged: GeoSPARQL.

3.4.2 GeoSPARQL

GeoSPARQL is an initiative of OGC, which released the GeoSPARQL standard in [49], to define spatial extensions to the W3C's SPARQL protocol and RDF query language. At the beginning of the work on this thesis, only little information was available; the major sources were the presentation [50] and a demo available at `http://www.geosparql.org/`.

Basic GeoSPARQL concepts are in Fig. 3.5: three basic classes: `SpatialObject`, which has two disjoint sub-classes `Feature` and `Geometry`. The object property `hasDefaultGeometry` is a sub-property of `hasGeometry`, which links the two sub-classes.

GeoSPARQL's vocabulary and definitions are contained in an ontology provided by OGC[7]. In the rest of this thesis, I will refer to the main GeoSPARQL namespace[8] with the prefix `geo:`.

`Geometry` instances can have various data properties. The most important one is for the data themselves, given in a form of serialization. All serialization data properties are sub-properties of `hasSerialization`, and the predefined ones are

[7] Available at `http://schemas.opengis.net/geosparql/`, cit. 2014-04-16, along with the imported ontologies for Simple Feature and GML geometries.

[8] `http://www.opengis.net/ont/geosparql#`

- `asWKT` for WKT strings, having the range of custom datatype `wktLiteral`, and
- `asGML` for GML data, having the range of custom datatype `gmlLiteral`.

Other datatype properties are

- `dimension` (topological dimension),
- `coordinateDimension` (dimension of direct positions),
- `spatialDimension` (dimension of the spatial portion of the direct positions),
- `isEmpty` (has no points), and
- `isSimple` (contains no self-intersections except its boundary).

Then it defines object properties for topological relations, there are three definition families of the relations, each containing eight relations. For their list and brief explanations, see Table 3.3. Every such object property has its domain and range equal to `SpatialObject`.

All the three families divide all possible spatial relations between two objects into eight basic topological relations, but not exactly the same way. The precise meaning of each topological relation can be described by the DE-9IM model, which uses 3×3 matrices. Table 3.4 gives the DE-9IM definitions serialized in 9-character strings, where the notation X/Y restricts usage of a relation of two geometries X and Y to certain types only. The X and Y can be "P" (points), "L" (lines), and "A" (polygons). For the details of the DE-9IM model see Section 2.2.

There are also some functions for comparing and manipulating geometries, which can be used in the FILTER section, see Table 3.5. These functions include alternates of all topological relation properties, to be applied as functions on geometry literals. GeoSPARQL also contains some RIF (Rule Interchange Format [51]) rules.

Table 3.3: Topological relations with their meanings, divided into families.

object property	**meaning**
	Simple Feature family
`sfEquals`	spatially equal
`sfDisjoint`	disjoint (cannot touch)
`sfIntersects`	share at least a point
`sfTouches`	externally touch
`sfWithin`	inside (can touch boundary)
`sfContains`	the inverse of `sfWithin`
`sfOverlaps`	some points common, same dimension
`sfCrosses`	e.g. line crosses area
	Egenhofer family
`ehEquals`	spatially equal
`ehDisjoint`	disjoint (cannot touch)
`ehMeet`	externally touch
`ehOverlap`	overlap
`ehCovers`	the inverse of `ehCoveredBy`
`ehCoveredBy`	inside (can touch boundary)
`ehInside`	inside (cannot touch boundary)
`ehContains`	the inverse of `ehInside`
	RCC8 – Region Connection Calculus family
`rcc8eq`	spatially equal
`rcc8dc`	disconnected
`rcc8ec`	externally connected
`rcc8po`	partially overlapping
`rcc8tppi`	tangential proper part inverse
`rcc8tpp`	tangential proper part
`rcc8ntpp`	non-tangential proper part
`rcc8ntppi`	non-tangential proper part inverse

Table 3.4: Topological relations with their DE-9IM definitions.

object property	**DE-9IM definition**
Simple Feature family	
sfEquals	TFFFTFFFT
sfDisjoint	FF*FF****
sfIntersects	T********, *T*******, ***T*****, ****T****
sfTouches	FT*******, F**T*****, F***T**** (all not P/P)
sfWithin	T*F**F***
sfContains	T*****FF*
sfOverlaps	T*T***T** (only A/A, P/P), 1*T***T** (only L/L)
sfCrosses	T*T***T** (only P/L, P/A, L/A), 0******** (only L/L)
Egenhofer family	
ehEquals	TFFFTFFFT
ehDisjoint	FF*FF****
ehMeet	FT*******, F**T*****, F***T**** (all not P/P)
ehOverlap	T*T***T**
ehCovers	T*TFT*FF* (only A/A, A/L, L/L)
ehCoveredBy	TFF*TFT** (only A/A, L/A, L/L)
ehInside	TFF*FFT**
ehContains	T*TFF*FF*
RCC8 – Region Connection Calculus family (all only A/A)	
rcc8eq	TFFFTFFFT
rcc8dc	FFTFFTTTT
rcc8ec	FFTFTTTTT
rcc8po	TTTTTTTTT
rcc8tppi	TTTFTTFFT
rcc8tpp	TFFTTFTTT
rcc8ntpp	TFFTFFTTT
rcc8ntppi	TTTFFTFFT

Table 3.5: Various GeoSPARQL functions for comparing and manipulating geometries.

Function	Description of what the function returns
distance	the distance of two geometry literals measured in given units
buffer	geometry literal as an input literal with a buffer added, given the radius and units of the buffer
convexHull	the convex hull of a geometry literal
intersection	the intersection of two geometry literals
union	union of two geometry literals
difference	the difference of two geometry literals
symDifference	set symmetric difference of two geometry literals
envelope	the bounding box of a geometry literal
boundary	the boundary of a geometry literal
getsrid	spatial reference system URI of a geometry literal

3.5 Existing Semantic Geospatial Search Systems

Let us have a look at some another system with a competing language behind it first. There have been many works on semantic integration ([52] gives an overview). An example is CARIN [53]. The CARIN family of languages, which is more general than AL-Log language [54], combines Horn rules and description logics. It deals with designing a sound and complete inference procedure for answering queries, which cannot be obtained by combining standard Horn rule inference procedures with intermediate terminological reasoning steps. It shows that when it takes description logic $\mathcal{ALCNR}$ and combines it with Horn rules, the non-recursive version is decidable, and the sound and complete algorithm for reasoning is presented; for the recursive version, it shows the sources of undecidability and the ways how to restrict it to make it obtain sound and complete reasoning.

An application of CARIN, Information Manifold, as presented in [55], serves a similar purpose as OnGIS – information gathering system providing uniform access to multiple structured information sources. It uses materialized database views that guarantee accessing only relevant sources. However, Information Manifold does not deal with spatial data sources. C-OWL [56] is an interesting approach for linking semantic data using contextualized ontologies based on OWL.

Owlgres [57], a system that one of the prototypes built as part of this thesis bases upon, is one of the $DL{-}Lite$ implementations, which connects to a database. The limitations of its original version compared to my prototype are that it cannot use multiple databases, and a user cannot provide a custom mapping of an ontology to the database.

Authors of the description logic family $DL{-}Lite$ (as stated in the articles [43, 44, 45] at Sapienza, a university in Rome) developed their own system. First, they developed the $DL{-}Lite$ reasoner QuOnto [58], which is based on $DL{-}Lite^{\mathcal{F}}_{core}$. It is a Java implementation, but not open-source. The reasoner uses H2[9] as a DBMS, which is a simple Java SQL database, in the embedded mode, using memory storage only. This system is of no use for my purposes at all, since the data input is in functional-style syntax [59] using its GUI only.

The authors also released ROWLKit [60] – another, slightly different implementation of $DL{-}Lite$ reasoner, based on QuOnto. The main difference is that ROWLKit uses OWL API for parsing its input and it extends the embedded QuOnto reasoner to meet OWL 2 QL ($DL{-}Lite^{\mathcal{H}}_{core}$). Still, it is not an open-source and uses H2 database. Hence it cannot be connected to an external database, nor a custom database mapping can be provided.

The last activity of the authors is the system Mastro, first introduced in [61], and later expanded in [62]. It is again based on QuOnto reasoner. However, this time, Mastro can connect to external databases (via JDBC), or can be coupled with a relational data federation tool. It also provides a way of defining ontology-to-database mapping – by typing the mapping

[9] `http://www.h2database.com/html/main.html`, cit. 2016-07-07.

assertions and data-to-ontology mapping assertions. These mappings are defined as views into the database, assigning instances to concepts and properties. Mastro is also available as a Protégé plugin. It is written in Java, but again it is not open-source.

Mastro aims for similar goals as this proposal. It uses a $DL{-}Lite$ ontology as the domain description, it connects to an external database, and it can be provided with a complex mapping of the ontology to the database. It can even use multiple data sources with the help of a data federation tool. However, it is not tailored for spatial data, and the use of a general data federation tool prevents implementing own optimizations on querying multiple data sources using an ontology.

Clark & Parsia, now Complexia Inc., a well-settled company in the field of Semantic Web, the developer (though originally created at the University of Maryland) of widely used OWL reasoner Pellet, developed Stardog[10]. Stardog is a commercial RDF database with OWL reasoning support (it supports RDFS, OWL 2 QL, RL, EL, and DL reasoning and SWRL rules). As its data are indexed, it supports full-text searching. Stardog database can contain a virtual graph, which is mapped to a relational database by R2RML. It supports querying via SPARQL and a proprietary protocol SNARL. It supports spatial data, GeoSPARQL query answering, and geospatial indexes. It would be interesting to use Stardog as an OnGIS data source with GeoSPARQL capabilities.

OWLDB [63] is a storage system for OWL language. As opposed to triple stores for RDF(S), which are not optimal for OWL, even though they can be used for its storage, OWLDB stores natively OWL constructs as objects (inspired by OWL API) by object-relational mapping into a relational database. Though it provides effective OWL storage, suitable for OWL axioms retrieval and editing, it is not optimized for in-database reasoning. This makes it not very usable for query answering over large data sources.

Another system using $DL{-}Lite$ is OWLIM[11] (OWLIM-SE has spatial support according to [64], but very limited).

For mapping ontology entities to a database, I have chosen a set of my own simple OWL annotations. It was difficult to reuse existing mapping approaches, e.g. the D2RQ mapping language [65, 66], R2RML [67], or its non-relational extension xR2RML [68], because they are not tailored for retrieving also spatial data. Another practical reason is that at the time the OnGIS mapping was being designed, D2RQ missed some properties to easily allow some necessary optimization techniques (e.g. query containment), and R2RML and xR2RML were not yet published.

In [69], an ontology-based information system is implemented, focusing on ontology-based spatio-thematic query answering for city maps. It bases on description logics reasoner RacerPro [70], implementing more expressive logic (compared to what I use) $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$. The system implements its own custom storage, which directly includes the inference algorithms and

[10] `http://stardog.com/`, cit. 2016-07-06.

[11] `http://www.ontotext.com/owlim`, cit. 2012-07-24

the query evaluation engine. A custom query language, SuQL (the substrate query language, also in [69]), is used. Spatial data in an ABox are represented e.g. as RCC relations only, or by using a special spatial ABox. However, it does not solve the problem of integrating multiple data sources.

The authors in [64] use Parliament triple store, supporting geospatial indexes, for storing spatial data and for making complex spatial queries via GeoSPARQL over them. However, they use a precomputed data set and do not directly support data integration.

The system in [71] links an RDF ontology to databases and WFS [11]. It uses custom rules and algorithms for query rewriting, but it does not provide the standard OWL semantics. However, it supports query answering from multiple data sources, specifically WFS servers for spatial data and databases (via the D2R interface) for attributes.

The spatial decision support system in [72] integrates various data sources (OGC standards WMS, WFS, WCS, WPS) and links them with ontologies. It also uses catalogue services via ontologies and automatic web service discovery. However, it focuses more on geospatial analysis and ontology alignment than spatial search.

Similar goal as OnGIS have the authors of [73], where they propose an interesting system also based on semantic technologies. But instead of open-world OWL semantics, they use rules (specifically SWRL rules), both for integrating sources and for answering queries. Also, the whole problem of data integration is summarized there. The authors state that for a spatial data infrastructure (SDI) to be easily accessible and useful, three problems have to be solved: (a) discovery – currently, SDIs usually support only simple string-based queries on metadata, (b) interpreting schemas – without semantic description, a source schema needs to be manually examined, to see, whether the source provides relevant information, and (c) search strategies and inferences – how to relate pieces of information from different sources, when there is no single one to satisfy a query entirely. The authors also define a data integration system as a uniform interface to a multitude of data sources, using a global schema (i.e. a shared domain vocabulary), where two approaches are available: global as view, and local as view.

Buster [74, 75] is a complex system dealing with terminological, spatial, and temporal query answering. It provides a common interface to heterogeneous information sources in terms of an intelligent information broker. It represents its terminology using the language OIL [76] and the description logic $\mathcal{SHIQ}$ (it is utilizing the FaCT reasoner), and uses Dublin Core as the vocabulary for modeling metadata. It solves some of OnGIS goals in a similar fashion, but it does not support complex queries in terms of e.g. spatial joins, and it does not support participating multiple sources on one user's query.

The system Karma presented in [77] uses its own base linking ontology for integrating spatial data sources. The linking ontology seems rather limited, as opposed to standard Simple Feature and GML ontologies accompanying GeoSPARQL. It also performs linking of features in two data sources by their spatial similarity. However, it uses only limited RDF expressive power and does not support complex spatial queries.

Methods for geospatial data source discovery in [78] use ontologies for integrating heterogeneous and distributed geospatial data sources, but its searching capabilities are based on semantic similarity measures and unsupervised word sense disambiguation of natural-language user query and stored metadata of respective sources, as opposed to strict semantic relations used for searching in the OnGIS system.

There are efforts to implement spatial relations to OWL reasoners, see [79]. They model spatial relations based on Region Connection Calculus (specifically RCC-8) by OWL reasoning. Unfortunately, it does not fit OnGIS, since it does not use real spatial geometries, but the spatial relations between objects have to be stated explicitly. This is impossible on large data sets (because the binary Cartesian product of features may become unmanageable), and it does not support filtering based on a distance between features. The idea in this article is implemented in PelletSpatial[12], a Pellet extension.

The authors of [80] propose a system for mapping ontology axioms to SQL queries on a database with the focus on geospatial data. Though using ontologies, it does not rely on OWL or any other reasoning. Also, it does not focus on multiple data sources.

The system DO-ROAM [81] is quite similar to my OnGIS. It is a web service that focuses on finding places according to activities that a person could perform there. It uses its own OBDA system, which maps ontology concepts and properties to database queries in quite a simple way. The implemented OBDA system is probably not a general OWL reasoner. Again, it does not support multiple data sources and is not general enough to support reasoning over different domains.

There are other systems similar to OnGIS developed. For example, an ontology-driven geographical information system is developed in [82]. It uses a simple logic not based on OWL, but it is designed to use multiple spatial data sources. [83] proposes another ontology-integrated GIS, dealing with the issue of Semantic interoperability, and the usage of multiple data sources. For the GIS part, it uses ArcGIS[13]. It proposes usage of ontologies and a reasoner, but it does not state of which ontology language, nor what kind of reasoner would be used. [84] presents another framework for interoperability of GIS applications. It supports multiple data sources again, and it is based on custom logic for matching with custom semantics. It is based on ORHIDEA project [85], providing integration of information using mediation technology suitable for spatial data in databases.

A comprehensive overview of related work in the area of ontology-driven GIS integration is in [86].

As query containment is an important part of our query-brokering solution (see Section 4.4), query containment capabilities of several systems have been examined. FaCT [87] and its successor FaCT++ [88] are reasoners supposed

[12] `http://clarkparsia.com/pellet/spatial/`, cit. 2011-09-15, currently unavailable, and the project was not found elsewhere.

[13] `http://www.esri.com/software/arcgis/index.html`

to support query containment, but they do not support custom datatypes (which are used in GeoSPARQL). Unfortunately, description of how the query containment works could not be found, even digging in their LISP, respectively C++ source codes did not help.

Pellet[14] is another reasoner with query containment support. However, it has a problem with data properties since all variables in its query containment module are modeled as individuals: when there is a data property with a variable, there is a problem with illegal punning in OWL. Pellet also supports the query language SPARQL-DL [89], a SPARQL subset with OWL-based semantics.

The -ontop- system[15], specifically its SPARQL query engine Quest, has some support for query containment. However, it seems it is used only for removing redundant queries during query rewriting, not for searching [90, 91].

SPIN[16], which stands for SPARQL Inferencing Notation, is a SPARQL-based rule and constraint language able to represent arbitrary SPARQL query in RDF. However, the SPIN RDF representation of a query is not suitable for OWL query answering. Therefore it is difficult to use it for deciding query containment. For example, there is a problem with an OWL-illegal punning: SPIN uses the same property[17] for linking both to a resource (IRI), and a literal, which means the property would be both data property and object property, which is illegal in OWL.

In [92], the author shows a method to decide query containment on SPARQL with OWL 2 EL [93] ontologies using the translation to μ-calculus [94]. OWL 2 EL is of polynomial data complexity, which is higher than AC^0 complexity of OWL 2 QL (which allows data query answering be performed directly by relational databases). The method used is complex enough to support property paths and optional graph patterns, and to cover other OWL 2 profiles as well, e.g. the author states that it can be used for OWL 2 QL without inverse roles.

In [95], the authors deeply analyze the complexity of query containment over "well-designed" SPARQL queries supporting optional graph patterns and unions, without RDFS or any other entailment regimes. The results are however not reusable for my case, as I leverage an entailment regime – the OWL 2 QL reasoning.

In [96], there is an evaluation of three SPARQL query containment solvers, two supporting RDFS, one not. Two of them are actually μ-calculus containment solvers and need some translation.

In [97] and [98], the authors propose methods for SPARQL query containment in $\mathcal{SHI}$ description logics, respectively under RDFS entailment regime. Both methods use a translation into μ-calculus.

The query containment decision methods based on μ-calculus presented in [92, 97, 98] should be possible to extend for OWL 2 QL semantics, and

[14] `https://github.com/Complexible/pellet`, cit. 2016-07-07.

[15] `http://ontop.inf.unibz.it/`, cit. 2014-06-04.

[16] `http://spinrdf.org/`, cit. 2014-05-10.

[17] `http://spinrdf.org/sp#object`

probably could also be extended with geospatial reasoning. It would be interesting to see the result and compare its performance to my method for deciding query containment.

Good ways how to optimize querying multiple sources are in [99], where its authors nicely summarize existing and present new ways how to perform efficient SPARQL query federation. The techniques presented there could be used to optimize OnGIS in the future.

An alternative to GeoSPARQL is stSPARQL [100], another query language based on SPARQL, with spatial and temporal extension functions. Most features of the two query languages are similar, with some differences: GeoSPARQL has three families of topological relations modeled both as properties and functions (see Section 3.4.2), while stSPARQL has one family of topological relations as functions only. On the other hand, stSPARQL has temporal functions, geospatial aggregate functions, and functions for minimum bounding boxes (none of these are necessary for the current version of OnGIS). stSPARQL has been developed as part of Strabon [101], a semantic spatiotemporal RDF store. The main reason GeoSPARQL has been selected for representing semantic geospatial queries is its chance to become widely used, as it is defined by the well-known OGC.

Chapter 4

Geospatial Integration and Retrieval Proposal

The goals of OnGIS are two: to allow a user to enter a semantic geospatial query in a user-friendly way, and to answer the query by combining results from multiple independent sources in an efficient way.

To allow the two parts be independent, a common spatial query language had to be selected. Choosing GeoSPARQL for the purpose is a logical choice, as it is the most detailed and up-to-date standard for describing spatial queries over semantic data, for details see Section 3.4.2.

The two parts are implemented as separate prototypes presented in Chapter 5. These could be used as a base for a complete platform for answering complex geospatial queries: a query input web interface, a query analysis and distribution engine, a backend retrieving the actual data from many heterogeneous geospatial sources, a geoprocessing unit combining the retrieved results together, and a web map presenting the results. The complete picture is given in Section 4.1.

GeoSPARQL is based on RDFS reasoning. However, for deciding query containment of GeoSPARQL queries (used later in this chapter), I use technique requiring logical negation, which is not contained in RDFS. Therefore, I extended GeoSPARQL with the semantics of OWL 2 QL (see Section 3.3.4), an ontology language having the logical negation (a limited version, but sufficient for my purposes), and also suitable for its trade-off between expressive power, and computational properties (it is tractable, i.e. evaluable using a relational database). OWL 2 QL is, therefore, suitable also as the language for querying data from the respective sources, which support it. One such geospatial data source has been developed as a prototype in OnGIS in Section 5.1.1.

Using GeoSPARQL, an accepted standard, as the internal format also has the advantage that separate parts of OnGIS can be used individually, in connection with other systems using the standard.

Section 4.2 presents a proposal how to answer semantic queries from external non-semantic GIS sources. Section 4.3 gives an idea of two ways how to intuitively enter a complex spatial query. And finally Section 4.4 and Section 4.5 present how to represent what different sources contain and their querying capabilities, and how to query the representation in order to find the relevant sources to a query.

4.1 Overall Architecture

The overall architecture of the proposed system is in Fig. 4.1 with its main part in the grayish rectangle. The blue rectangles represent the designed modules, which are also implemented as separate prototypes. The gray rectangle (a geoprocessor) is a module, which has not been implemented, but is necessary to complete the system. The orange cylinders are external services, running existing, well-known technologies, serving geospatial data. The orange ovals are local data, loaded into OnGIS:

- A set of **background ontologies**, consisting of various domain and linking ontologies (e.g. DBpedia, LinkedGeoData), and OnGIS annotations, all loaded into memory. It is treated as one big ontology used for all reasoning in the system.
- Service capabilities, represented as **prototypical queries**, demonstrating, what data can each of the connected services provide (see Section 4.4).

On the input, there is a subsystem for entering a query presented in Section 4.3. Using the background ontology, it helps a user to compile a semantic query, which is formulated as GeoSPARQL on the module output.

The GeoSPARQL query is first processed in a query distribution module presented in Section 4.4 and Section 4.5. It decides which services are necessary to answer the query – in case no single source can answer it entirely, the query is split into parts already answerable be different services.

The set of split queries (again in GeoSPARQL form) continues to the backend module, presented in Section 4.2, which is able to transform each query part to an appropriate service request to the corresponding service, and to extract the response.

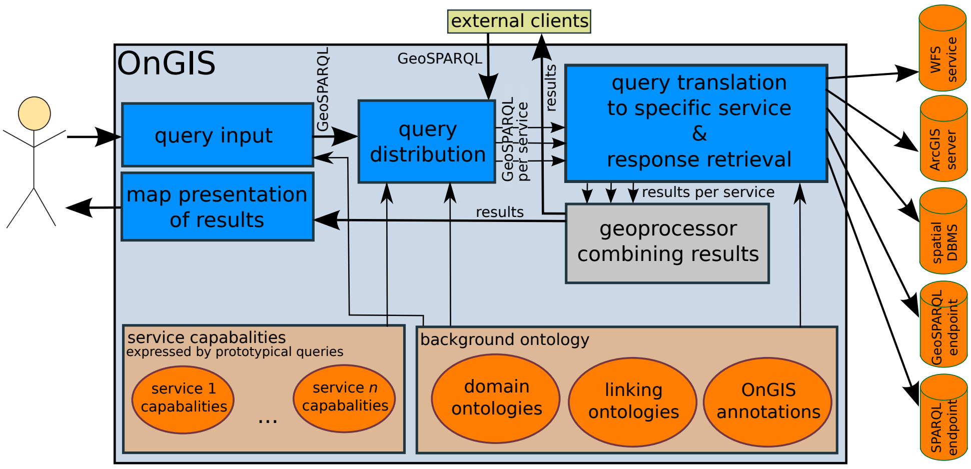


Figure 4.1: Overall architecture.

Finally the answers from all used services would be combined together (a part not yet implemented), and presented to the user on a map.

4.2 Geospatial Semantic Retrieval

The idea of OnGIS geospatial data retrieval is based on the fact, that there are many independent GIS services in the World, having different data, using different technologies, and having different capabilities of presenting and processing its data. It cannot be expected that every GIS service will publish its data as Linked Data, but still, we want to use them.

Therefore OnGIS is designed the way it can translate a semantic query in GeoSPARQL to a request suitable for the specific service technology and retrieve the results. To allow this, OnGIS has to be loaded with a background ontology, which has the role of a TBox used for OWL 2 QL reasoning. A GeoSPARQL query is reformulated using the TBox to an ABox-only query, which can be evaluated against the external services by the aforementioned translation. This way OnGIS treats all external services as ABox storages only, assuming all TBoxes are loaded into OnGIS background ontology in memory (they can be continuously updated from external sources), allowing reasoning to be always readily available (both for constructing a user's query and for translating the query to service requests). The assumption of having all TBoxes locally, and keeping the ABoxes (the data) at the original services seems to be practical, as the TBoxes are usually not that large (e.g. LinkedGeoData ontology has about 24 thousand triples, most of it labels), while GIS data are usually very large (e.g. OpenStreetMap has cca. 3.5 billion nodes).

To let the query translation module know, how to translate a query to a specific service request, each service domain ontology has to be annotated with information about that particular mapping. For that purpose, a set of custom OnGIS annotation has been developed, because existing approaches were not suitable or available (see Section 3.5).

The annotations annotate each class and property having its instances in the service with the means, how the instances can be retrieved from the service.

One of the transformations is to the SQL language for relational databases. It has been used for some GIS data converted to PostgreSQL with PostGIS extension, see Section 5.1.1. The annotations are:

- **`anno:DBDatabase`** – the annotation property used to annotate the service domain ontology itself, stating the connection details to the database (server location, username, etc.).
- **`anno:DBClassTable`** – the annotation property used to annotate a class, specifying in which table the instances are located, along with the names of columns containing the instance details. The value of the annotation is the table name, with attributes in round brackets after the table

name. The attributes are a comma separated list of equal-sign-separated key/value pairs, with the key having a meaning defined in the following sub-list:

  - **id** – the name of a column for numerical identification of instances,
  - **geom** – the name of a column for storing spatial data (e.g. PostGIS geometry column), in case the class represents a geometry; otherwise it may be missing, along with the two following fields,
  - **geomSRID** – the identification of spatial reference system used for spatial data (EPSG code), and
  - **geomType** – the WKT type of the spatial data (POINT, LINESTRING, POLYGON, or GEOMETRY for arbitrary).

  The annotation can contain more key/value pairs, which serve as filters, where the key is a column name, and the value is a value the instance has to have in the column in order to be considered an instance of the class.

- **`anno:DBObjectPropertyTable`** – the annotation property used to annotate an object property, defining in which table the pairs of instances are stored; the format is similar to the previous annotation, with the following meaning of attributes:

  - **id** – the name of a column for identification of subject instance,
  - **obj** – the name of a column for identification of object instance,
  - and some other auxiliary switches.

- **`anno:DBDataPropertyTable`** – the annotation property used to annotate a data property, defining in which table instances with their data values are stored; the format is similar to the two previous annotations, with the following meaning of attributes:

  - **id** – the name of a column for identification of subject instance,
  - **val** – the name of a column for the data value (always represented in character data type),
  - **lang** – the name of a column for storing the language of the data value (if it is a string),
  - **length** – the length of the character value column,
  - and some other auxiliary switches.

- **`anno:DBAnnotationToLiteralTable`**, **`anno:DBAnnotationToResourceTable`** – annotation properties similar to `anno:DBDataPropertyTable`, respectively `anno:DBObjectPropertyTable`, but containing annotation assertions instead of property assertions.

- `anno:DBGeometryTable` – the annotation property optionally annotating the service domain ontology itself, with the structure of attributes the same as of the DBClassTable; the purpose of this annotation is to mark a table in the data source as being a general storage of spatial data. This table is used when a geometry individual is searched without specifying which class is it an instance of, giving only e.g. spatial or property restrictions.

There are also two annotations for ArcGIS services – one for its WebService interface (`anno:ArcgisWebservice`), the other for its REST API (`anno:ArcgisRest`). They annotate a spatial class with the information about the corresponding endpoint and the layer identification.

Another annotation is designated for SPARQL endpoints with no special geospatial support – `anno:SparqlEndpoint`. The transformation simplifies the GeoSPARQL query using the W3C Basic Geo Vocabulary (see Section 3.2.2) to a simple SPARQL query.

An example of an annotation used on the IPR Praha domain is that the class of footpaths, being a subclass of `ipr:Layer` (which is a subclass of `geo:Feature`), is annotated with the ArcGIS REST API mapping annotation, defining how to retrieve the data from the footpaths layer at IPR Praha ArcGIS server.

Future work would be replacing the database mapping annotations with a current standard, e.g. R2RML, and also designing mappings for WFS and other GIS service interfaces.

4.3 Query Input Design

OnGIS tries to follow typical web search scenarios. A general user is not used to give a search engine a structured query, e.g. a SPARQL or GeoSPARQL, but enters a few keywords, and examines the results the query engine gives back instead. A common scenario, when a user does not exactly know how to describe what he/she is looking for, is iteratively refining entered keywords based on what the search engine returned in the previous search.

Therefore I tried to follow this process in two ways of entering a query; both share the basic principle, that a query is constructed gradually, with intermediate results visible on a map immediately.

4.3.1 List Query Design

The list mode of entering a query is based on a graphical list of boxes, representing entities (i.e. classes, individuals, data and object properties), which a user added.

The user fills the list one by one both by searching the background ontology using the labels, descriptions, definitions, etc. contained in the background ontology, and by searching the data sources themselves. Each time a new spatial class or individual is added, it is displayed on a map. When the user decides it is not one of the necessary query inputs, it can be removed from the list.

Once the user is satisfied with the components of the query, he/she can impose restrictions on them. For example individuals of one class in the query has to be spatially contained within individuals of another class in the query, or individuals of one class in the query must be within a certain distance to a specifically listed individual in the query. There can be object and value restrictions, too, e.g. individuals of a class in the query must satisfy some filter on a data property restricting them.

A prototype, how such query can look like, is in Section 6.1.1.

There are some auxiliary annotations that can help make the user experience better:

- **Searchable** specifies a data property, which should be searched by the connected services when a user enters a query. The background ontology is always searched, but data properties with this annotation are also searched in the connected services via the translation module, with the returned results presented to the user. This way, a user can search for a specific spatial individual in the services (e.g. a specific city in DBpedia), and restrict other results by the distance to it.
- **PartOf** specifies a relation between entities by an object property, defining the part-of relation – it is useful for linking an object to its integral components (these are quickly accessible in the user interface).
- **Priority** specifies a numeric priority of entities, which affects the order in which the entities should appear in the search result list.

An example of the annotations used on the IPR Praha domain is that the `PartOf` annotation relates `ipr:Layer` and `ipr:Service` by the `ipr:isLayerOfService` object property, `ipr:Attribute` and `ipr:Layer` by the `ipr:isAttributeOfLayer` object property, and `ipr:Legend` and `ipr:Layer` by the `ipr:isLegendOfLayer` object property. The `searchable` annotation marks the IPR Praha domain data properties `ipr:hasKeywords`, `ipr:hasDefinition`, `ipr:hasDescription`, etc.

The relations between objects can be entered in two ways:

simple The simple way is to add a spatial or data property restriction to a search result just by itself. Currently, only two spatial restrictions are possible: *inside* restriction and *distance* restriction. The inside restriction filters all other search results, so that they have to be contained inside the search result defining the restriction. The distance restriction is similar – all other search results have to be within the specified distance. Similarly, a data property restriction affects all other items. Apparently, this way is not very flexible and is useful only for simple searches with a few items.

by links Another way of specifying relations is by defining links. A link can be established between two items in the list, which filters one item by the other (in case the target is a spatial entity, it restricts by *inside* or *distance*; in case of a data property, it restricts by a data value; in case of an object property, it restricts by a relation to another item which is the target linked to).

The basic conjunction is the *or*. All spatial entities simply put to the list are displayed on the map by issuing one query for each list entity. When an unlinked restriction appears (spatial, a property), it restricts all other items. When linked, it restricts only the item it is linked to. Therefore the graph of a query can be represented by a forest (no cycles are possible), where there is the disjunction between the trees in the forest. As there is no disjunction (class union) in OWL 2 QL, the query forest has to be answered a tree at a time, and all the results displayed on a map at once (thus performing the disjunction).

4.3.2 Structured Query Design

The structured query mode enables an advanced user to enter a query via a mathematical-expression-like statement structured with parentheses. Similarly to the previous mode, a user cannot enter random strings, but with the aid of autocompletion mechanism, he/she can only pick the known terms of background ontology. However, the input expression displays the term in a human readable form (e.g. the `rdfs:label`), even though the query engine internally remembers the IRI.

It uses a single text input field with autocomplete. The field displays human readable string consisting of object labels, parenthesis, and spatial restriction keywords, but the engine holds its structured model in the background. When a user starts typing, autocomplete list suggests the user entities from the background ontology, from which he/she has to pick. If the user were to enter a free text, and the query engine would try to understand the text, it could happen that it could not assign semantic entities to the entered items uniquely, causing inexactnesses or errors.

The spatial restriction keywords can be (in the first version of our structured language) “WITHIN” and “NEAR”. The query can be structured using spaces, parentheses, and commas. Also, data value filters can be entered. The query resembles a mathematical expression, more specifically an expression in predicate logic, where the logical *and* ($\wedge$) is expressed by a space, and logical *or* ($\vee$) by a comma. As the used query language, conjunctive queries in OWL 2 QL serialized as GeoSPARQL, does not support disjunction, the comma is allowed on the first level of the query structure. If the comma is used, multiple GeoSPARQL queries are generated, one for each disjunct (an operand of the disjunction).

Let us define the syntax formally. A query is an expression E, which is defined as a list of conjunction clauses C, which are separated by commas. Clauses are lists of restrictions separated by spaces. A restriction can be one of:

- **class restriction** (R_C): represents restricting the results to a spatial class,
- **near restriction** (R_N): restricts the results by given maximum distance to spatial objects defined by another expression,
- **within restriction** (R_W): restricts the results to be spatially contained within spatial objects defined by another expression, and
- **value restriction** (R_V): represents a data property restriction, with a double-quoted value attached.

The rules how to define a structured query can be expressed as:

$$E ::= C_1, C_2, C_3, \dots \tag{4.1}$$

$$C ::= R_1{\sqcup}R_2{\sqcup}R_3{\sqcup}\dots \tag{4.2}$$

where R can be one of:

- $R_C ::= c$, where c is a spatial class or individual,
- $R_N ::= \text{NEAR}{\sqcup}(C){\sqcup}x$, where x is a distance value,
- $R_W ::= \text{WITHIN}{\sqcup}(C)$,
- $R_V ::= d{\sqcup}"x"$, where d is a data property, and x is a literal.

The parentheses surrounding C in R_N and R_W can be left out, when it is a single R_C, i.e. $C = R_C$. Therefore R_N and R_W can look like $\text{NEAR}{\sqcup}R_C{\sqcup}x$, respectively $\text{WITHIN}{\sqcup}R_C$.

An example prototype of structured query input form is in Section 6.1.2.

4.4 Representing Geospatial Source Contents and Capabilities

When performing complex queries, requiring integration of results from multiple sources, it is necessary to know, what each of the sources can offer. In the GIS domain, it is also good to know what geospatial operations a service can perform, as spatial transformations can be computationally intensive, especially when dealing with large data.

And why to design a new language for that purpose? The most general approach for describing querying capabilities has been chosen, to describe it with a set of queries the source can answer, in my case in the GeoSPARQL language. I call the capability representative queries the *prototypical queries*.

When a user enters a query, OnGIS has to compare it to the prototypical queries of sources, to find the most suitable one(s). The essential part of matching a query against a set of queries is the problem called query containment, which decides subsumption relation of two queries. One query is subsumed by another, $q_1 \sqsubseteq q_2$, whenever each result set of q_1 for data D is a subset of the result set of q_2 for the same data D.

I formulate the queries in a subset of SPARQL language, with the extension of some of the GeoSPARQL vocabulary and its semantics, and the OWL 2 QL semantics. From the SPARQL language, only *SELECT* queries are supported, with graph patterns and filters. Optional patterns, ordering, grouping, offsets, and limits are currently not supported.

From the GeoSPARQL query language, besides basic classes, properties, and serializations, topological relations, and some functions are supported. Specifically, the supported features are:

- classes `SpatialObject`, `Feature` and `Geometry`,
- object properties `hasGeometry` and `hasDefaultGeometry`,
- data properties `hasSerialization`, `asWKT`, and `asGML` (and parsing its literals in WKT and GML),
- all the three topological relation families (Simple Feature, Egenhofer, and RCC8),
- and the `distance` function (for details of all the above, see Section 3.4.2).

In Section 4.4.1, hierarchy on topological relations is defined, then Section 4.4.2 defines general query containment algorithm for OWL 2 QL, Section 4.4.3 gives reasoning extension for GeoSPARQL, and Section 4.4.4 presents a simple algorithm for matching variable queries.

4.4.1 Expanding GeoSPARQL ontology

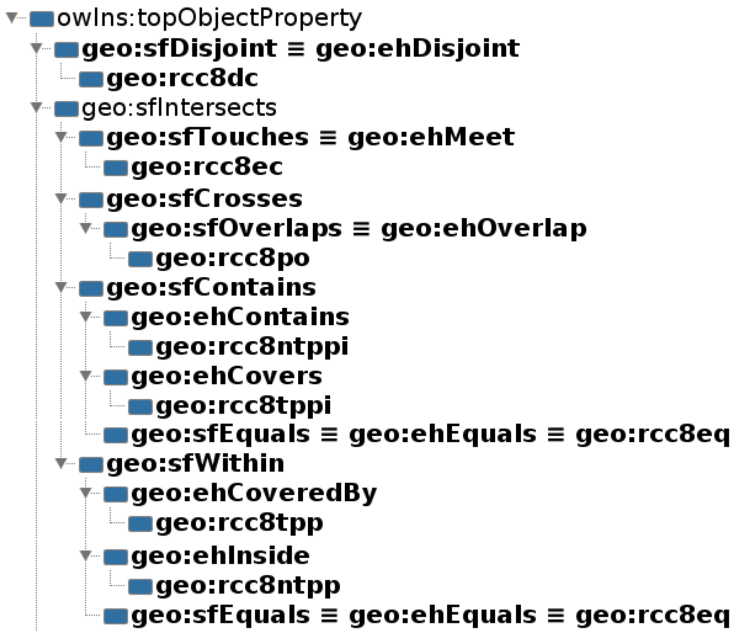
One part of supporting the semantics of GeoSPARQL is to understand the relations between the topological relations. The GeoSPARQL ontology contains only a list of all the topological relations, without any hierarchy. Therefore, I added a hierarchy comparing topological relations between different relation families, in order to support deciding query containment of queries using them. For example, both `rcc8tpp` and `rcc8ntpp` (tangential and non-tangential proper part from the RCC8 family) are sub-properties of `sfWithin`.

The complete hierarchy of relations was determined by their DE-9IM definitions, and its visualization in Protégé is in Fig. 4.2.

4.4.2 Query Containment Basics

First, let me define a query as a tuple of output variables, class restrictions, object and data property restrictions, and filters, formally

$$q = (V_o, R_c, R_{op}, R_{dp}, R_f), \tag{4.3}$$



```
owlns:topObjectProperty
  geo:sfDisjoint ≡ geo:ehDisjoint
    geo:rcc8dc
  geo:sfIntersects
    geo:sfTouches ≡ geo:ehMeet
      geo:rcc8ec
    geo:sfCrosses
      geo:sfOverlaps ≡ geo:ehOverlap
        geo:rcc8po
    geo:sfContains
      geo:ehContains
        geo:rcc8ntppi
      geo:ehCovers
        geo:rcc8tppi
      geo:sfEquals ≡ geo:ehEquals ≡ geo:rcc8eq
    geo:sfWithin
      geo:ehCoveredBy
        geo:rcc8tpp
      geo:ehInside
        geo:rcc8ntpp
      geo:sfEquals ≡ geo:ehEquals ≡ geo:rcc8eq
```

Figure 4.2: Extending GeoSPARQL with the hierarchy of topology object relations.

where

- V_o is a set of output variables of the query; a variable is in the rest of the text prefixed with the question mark,
- R_c is a set of class restriction on variables, $C(?x)$, where C is a class name,
- R_{op} is a set of object property restrictions in the form of either $OP(?x, ?y)$, $OP(?x, i)$, or $OP(i, ?y)$, where OP is an object property name, and i is an individual,
- R_{dp} is a set of data property restrictions in the form of either $DP(?x, ?y)$, $DP(?x, d)$, or $DP(i, ?y)$ where DP is a data property name, and d is a literal,
- R_f is a set of filters, restricting the result by functions. A filter is a predicate (a function returning a boolean value), which must be satisfied for the returned results. It has the form of $f(\underline{x})$, where f is a boolean function, and $\underline{x}$ is a tuple having the same arity as f and the proper types. Elements of $\underline{x}$ can be variables, individuals, literals, and other function calls.

The elements of R_c, R_{op} and R_{dp} are also called *triples* of the query q.

The main idea, how to decide query containment, is based on [4] and [5]: take the two compared queries, q_1 and q_2, and a background ontology $\mathcal{O}$ (TBox and ABox of valid axioms, on which background the query containment is to be decided) and transform the queries into two ABoxes, which, using a series of satisfiability checks, lead to deciding whether $\mathcal{O} \models q_1 \sqsubseteq q_2$. My modifications of the already proposed methods include adapting it for the OWL 2 QL logics (originally they support $\mathcal{DLR}$ logic, having relations of any arity, by a translation into satisfiability in $\mathcal{SHIQ}$ description logics), and adding support for spatial reasoning by extending completed ABox in Section 4.4.3.

First, I define a canonical ABox of a query (it is basically just substituting variables with individuals, one individual per variable):

$$\begin{aligned}
\mathrm{Can}(q) &= \mathrm{Can}(R_c) \cup \mathrm{Can}(R_{op}) \cup \mathrm{Can}(R_{dp}) \\
\mathrm{Can}(R_c) &= \{C(i_x) : C(?x) \in R_c\} \\
\mathrm{Can}(R_{op}) &= \{OP(i_x, b) : OP(?x, b) \in R_{op}\} \\
&\cup \{OP(a, i_y) : OP(a, ?y) \in R_{op}\} \\
&\cup \{OP(i_x, i_y) : OP(?x, ?y) \in R_{op}\} \\
\mathrm{Can}(R_{dp}) &= \{DP(i_x, d) : DP(?x, d) \in R_{dp}\} \\
&\cup \{DP(a, d_y) : DP(a, ?y) \in R_{dp}\} \\
&\cup \{DP(i_x, d_y) : DP(?x, ?y) \in R_{dp}\}
\end{aligned} \tag{4.4}$$

where a subscripted i is a fresh individual, C is a class, OP is an object property, DP is a data property, a subscripted d is a fresh (artificial) literal, and a letter prefixed with the question mark is a query variable.

Let me denote $A_1 = \mathrm{Can}(q_1)$, $A_2 = \mathrm{Can}(q_2)$, I_1 all individuals in A_1, I_{V1} all individuals representing variables in A_1, D_1 all literals in A_1, D_{V1} all literals representing variables in A_1, similarly I_2, I_{V2}, D_2, and D_{V2} for A_2, and $\mathcal{O}$ a background ontology.

A *completed ABox of a property* is an extended ABox built on top of canonical ABoxes:

$$\begin{aligned}
\mathrm{Com}(OP(i_x, i_y), q_1, q_2) &= \alpha(i_x, i_y, q_1, q_2) \cup \beta(i_x, i_y, q_1, q_2) \\
\alpha(i_x, i_y, q_1, q_2) &= \begin{cases} \{(i'_x, i_y) : i'_x \in I_1 \setminus I_{V1}\} & \text{if } i_x \in I_{V2} \\ \emptyset & \text{otherwise} \end{cases} \\
\beta(i_x, i_y, q_1, q_2) &= \begin{cases} \{(i_x, i'_y) : i'_y \in I_1 \setminus I_{V1}\} & \text{if } i_y \in I_{V2} \\ \emptyset & \text{otherwise} \end{cases} \\
\mathrm{Com}(DP(i_x, d_y), q_1, q_2) &= \gamma(i_x, d_y, q_1, q_2) \cup \delta(i_x, d_y, q_1, q_2) \\
\gamma(i_x, d_y, q_1, q_2) &= \begin{cases} \{(i'_x, d_y) : i'_x \in I_1 \setminus I_{V1}\} & \text{if } i_x \in I_{V2} \\ \emptyset & \text{otherwise} \end{cases} \\
\delta(i_x, d_y, q_1, q_2) &= \begin{cases} \{(i_x, d'_y) : d'_y \in D_1 \setminus D_{V1}\} & \text{if } d_y \in D_{V2} \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned} \tag{4.5}$$

Using those, I can define a *testing ontology* $\bar{\mathcal{O}}(a)$ as a function of axioms from A_2:

$$\begin{aligned}
\bar{\mathcal{O}}(C(i_x)) &= (\mathcal{O} \cup A_2) \setminus \{C(i_x)\} \cup A_1 \cup \{NC(i_x)\} \\
\bar{\mathcal{O}}(OP(i_x, i_y)) &= (\mathcal{O} \cup A_2) \setminus \{OP(i_x, i_y)\} \cup A_1 \\
&\cup \{NOP(i_x, i_y)\} \\
&\cup \{NOP(i'_x, i'_y) : (i'_x, i'_y) \in \mathrm{Com}(OP(i_x, i_y))\} \qquad (4.6) \\
\bar{\mathcal{O}}(DP(i_x, d_y)) &= (\mathcal{O} \cup A_2) \setminus \{DP(i_x, d_y)\} \cup A_1 \\
&\cup \{NDP(i_x, d_y)\} \\
&\cup \{NDP(i'_x, d'_y) : (i'_x, d'_y) \in \mathrm{Com}(DP(i_x, d_y))\},
\end{aligned}$$

where NC is a fresh class for each C with the restriction $NC \sqsubseteq \neg C$ added to $\bar{\mathcal{O}}(a)$, and similarly for NOP for each OP and NDP for each DP.

Then if there exists an assertion $a \in A_2$ such that $\bar{\mathcal{O}}(a)$ is consistent, or filters are not contained (see below), then $q_1 \not\sqsubseteq q_2$, otherwise $q_1 \sqsubseteq q_2$.

The proof of the correctness can be based on proofs in [4] and [5], as my steps are based on those articles with suitable modifications and simplifications for OWL 2 QL semantics.

Intuitive proof: in order to $q_1 \sqsubseteq q_2$ be valid, q_1 has to restrict results more than q_2. This is tested by taking one restriction in q_2 (as an axiom in A_2) at a time, negating it, and putting it together with the background ontology, A_1, and the rest of A_2; and in the case of a property, also some additional axioms. This altogether is tested for consistency; if it is consistent, it means that results are less restricted by q_1 than by q_2 (q_1 still has some results, even if it is restricted with the negation of a q_2 restriction, meaning skipping at least the results originally given by q_2), and therefore $q_1 \not\sqsubseteq q_2$. The additional axioms mentioned above are given by $\mathrm{Com}(OP)$ and $\mathrm{Com}(DP)$; they are necessary because a variable, in A_2 represented by an individual/literal, can be substituted by any non-variable individual/literal. For consistency checks in the query containment decisions, it is necessary to substitute only non-variable individuals/literals in A_1.

To analyze complexity of the decision, we follow complexity of the steps: the size of $\mathrm{Can}(q)$ is linear to the size of q, the size of Com(an assertion axiom, q_1, q_2) is also linear to the size of q_1, and the size of $\bar{\mathcal{O}}(a)$ is linear to the size of $\mathcal{O}$, q_1, and q_2 combined. Therefore, query containment decision requires approx. $|q_2| \times (|\mathcal{O}| + |q_1| + |q_2|)$ consistency checks in OWL 2 QL, i.e. a polynomial number in the query sizes and the background ontology size. And since consistency checks in OWL 2 QL are of NLogSpace-complete complexity [46], the overall complexity of the query containment is PolyTime. Note that this complexity is lower than the complexity ExpTime in [5], thanks to lower expressive power of OWL 2 QL.

To deal with filters, a different approach has to be used. They have multiple arity and they have different semantics (they do not have the open world assumption), thus using their reification, negation, and consistency checks do not solve their containment.

A simple scheme is used for deciding query containment with filters:

$$(\exists f_2 \in R_{f2} : \nexists f_1 \in R_{f1} : f_1 \sqsubseteq f_2) \Rightarrow q_1 \not\sqsubseteq q_2. \tag{4.7}$$

Intuitively the filter containment relation $\sqsubseteq$ expresses whether one filter is more restrictive than the other. It has to be defined according to the specific filter function definition.

4.4.3 Query Containment with GeoSPARQL

When deciding query containment with queries only on the symbolic level with individuals, topological relations are covered by extending them with a hierarchy, as in Section 4.4.1. But when there are geospatial literals involved, it gets more complicated.

First, I define the set of all topological relation restrictions of a geometry individual as

$$\begin{aligned} Rel(i, q_2) = \{&-TR(OP) : OP(i_x, i_y) \in R_{op2} \wedge (hG(i_x, i) \vee i_x = i)\} \\ \cup\{&TR(OP) : OP(i_x, i_y) \in R_{op2} \wedge (hG(i_y, i) \vee i_y = i)\}, \end{aligned} \tag{4.8}$$

where $hG \sqsubseteq$ `geo:hasGeometry`, R_{op2} is R_{op} in q_2, and $TR(OP)$ (topological relation restriction values for all topological relations) is defined in Table 4.1. Note that in a topological relation both a geometry individual and a feature individual can play roles (thus the logical *or* in the definition).

Table 4.1: Values of restrictions $TR(OP)$ of topological relations.

Egenhofer OPs	$TR(OP)$	Simple Feature OPs	$TR(OP)$	RCC8 OPs	$TR(OP)$
`geo:ehEquals`	0	`geo:sfEquals`	0	`geo:rcc8eq`	0
`geo:ehOverlap`	0	`geo:sfIntersects`	0	`geo:rcc8po`	0
`geo:ehDisjoint`	-1	`geo:sfDisjoint`	-1	`geo:rcc8dc`	-1
`geo:ehContains`	-1	`geo:sfContains`	-1	`geo:rcc8ec`	0
`geo:ehCoveredBy`	0	`geo:sfCrosses`	0	`geo:rcc8ntpp`	1
`geo:ehCovers`	0	`geo:sfTouches`	0	`geo:rcc8ntppi`	-1
`geo:ehInside`	1	`geo:sfWithin`	1	`geo:rcc8tpp`	0
`geo:ehMeet`	0	`geo:sfOverlaps`	0	`geo:rcc8tppi`	0

The numerical values of $TR(OP)$ in Table 4.1 represent necessary conditions on the topological relation between two geometries in order to answer an OP containment. In order to $OP(x, a) \sqsubseteq OP(x, b)$, the relation between the features/geometries a and b has to be according to Table 4.2.

Table 4.2: Meanings of $TR(OP)$ values.

$TR(OP)$	condition
0	$a \equiv b$
1	$a \subseteq b$
-1	$b \subseteq a$

Then I can define an *effective topological relation restriction* of a geometry individual:

$$r_e(i, q_2) = \begin{cases} 0 & \text{if} \quad |Rel(i, q_2)| > 1, \\ \text{the only element in } Rel(i, q_2) & \text{otherwise.} \end{cases} \tag{4.9}$$

Using the topological relation restriction, I can define geometry containment relation:

$$x \sim_r y = \begin{cases} x \equiv y & \text{if} \quad r = 0, \\ x \subseteq y & \text{if} \quad r < 0, \\ y \subseteq x & \text{if} \quad r > 0, \end{cases} \tag{4.10}$$

where the relation $\subseteq$ between two geometries represents that one geometry is a subset (is within) another geometry and the relation $\equiv$ represents that the two geometries are the same.

Then I can extend completed ABox of a data property as:

$$\begin{aligned} \mathrm{Com}^2(hS(i_x, d_y)) = \mathrm{Com}(hS(i_x, d_y)) \cup \\ \cup \{(i_x, g) : g \in D_1 \setminus D_{V1} \wedge (d_y \sim_{r_e(i_x, q_2)} g)\} \end{aligned} \tag{4.11}$$

where $hS \sqsubseteq$ `geo:hasSerialization`, and the rest of the symbols is defined in Section 4.4.2. When $\mathrm{Com}^2(DP)$ is used for obtaining $\bar{\mathcal{O}}(a)$ instead of $\mathrm{Com}(DP)$, the query containment decision procedure is extended with GeoSPARQL topological relations reasoning.

This way, containment on even complex query patterns is answered correctly. To give an intuitive proof, I will continue the intuitive proof at the end of Section 4.4.2. Here the $\mathrm{Com}(DP)$, containing possible substitutions for variables, needs to be extended with substitutions also for geometry literals. But how to select, which geometry literals to substitute? It depends on how the geometry restricts the rest of the query, e.g. when an object has to be within a geometry, the geometry can be substituted with a geometry covering it. The impact spatial restrictions have on geometry substitutions is given by $TR(OP)$; for a specific object it is computed by $Rel(i, q_2)$, and the effect the spatial restrictions have on the substitution is given by $r_e(i, q_2)$. Then, by comparing geometries based on $r_e(i, q_2)$, it can be correctly decided which geometries to substitute for a geometry in a query to decide query containment with spatial semantics.

Note that since all steps to obtain Com^2 do not exceed PolyTime complexity, using this spatial extension does not affect the overall complexity of query containment.

Imagine the two queries in Listing 4.1, structure of which is displayed in Fig. 4.3. Note that the two polygons there are for example the areas of the Czech Republic (CZ, in both queries) and Slovakia (SK, in q_2 only); the two countries are neighbors.

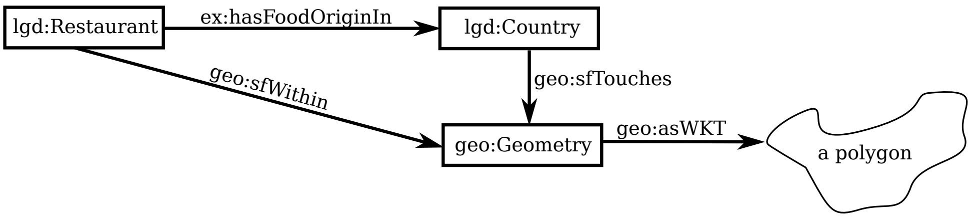


Figure 4.3: Example of a circle in a query.

Listing 4.1: q_1 and q_2 in circular spatial restriction example.

```
SELECT ?x WHERE {
    ?x a lgd:Restaurant.
    ?x geo:sfWithin ?g.
    ?x ex:hasFoodOriginIn ?c.
    ?c a lgd:Country.
    ?c geo:sfTouches ?g.
q1: ?g geo:asWKT "POLYGON((<CZ>))"^^geo:wktLiteral.
q2: ?g geo:asWKT "POLYGON((<CZ+SK>))"^^geo:wktLiteral.
}
```

Obviously, $q_1 \sqsubseteq q_2$ cannot be true, since the q_1 results contain restaurants in the Czech Republic with their food origin in Slovakia, while the results of q_2 do not. The containment would be true, if there would be only the `geo:sfWithin` spatial restriction.

To show how the proposed reasoning would reach the correct decision, let me show the steps:

$$Rel(i_g, q_2) = \{TR(\texttt{geo:sfTouches}), TR(\texttt{geo:sfWithin})\} = \{0, 1\} \quad (4.12)$$

$$r_e(i_g, q_2) = 0 \quad (4.13)$$

Therefore, the completed ABox for `geo:asWKT` (`<CZ+SK>` in q_2) will not be extended with `<CZ>` from q_1, thus testing consistency on $\bar{\mathcal{O}}$ of this data property will give consistent, meaning that $q_1 \not\sqsubseteq q_2$.

4.4.4 Resolving Variable Mapping

One problem in query containment is variable mapping. Using the simplest attitude, variables are converted to individuals and literals, as in Section 4.4.2, assuming they are uniquely identified by their names, and the query containment algorithm decides the query subsumption. It works if the variables are named consistently between the compared queries. However, this might not be the case in the real world – different systems may generate the template queries describing its source's capabilities, and they may name the variables differently.

I make only one assumption: the output variables are the same. This is a realistic assumption, since GIS systems, in simple cases, generate e.g. objects' geometries and labels, therefore the implemented semantic GIS system may fix constant output variable names and types.

Consider the following simple example of the two queries, written in GeoSPARQL syntax in Listing 4.2 and Listing 4.3.

Listing 4.2: q_1 in variable mapping example.

```
SELECT ?x ?wkt WHERE {
 ?x a lgd:Restaurant.
 ?x geo:hasGeometry ?g.
 ?g geo:asWKT ?wkt.
 ?x geo:sfTouches ?r.
 ?r a lgd:PrimaryRoad.
}
```

Listing 4.3: q_2 in variable mapping example.

```
SELECT ?x ?wkt WHERE {
 ?x a lgd:Amenity.
 ?x geo:hasGeometry ?g.
 ?g geo:asWKT ?wkt.
 ?x geo:sfTouches ?y.
 ?y a lgd:PrimaryRoad.
}
```

Intuitively, $q_1 \sqsubseteq q_2$, and $q_2 \not\sqsubseteq q_1$, no matter what is the name of the "hidden" variable. To solve this problem, a simple algorithm was designed. Its input are two queries, q_1 and q_2, with the assumption that the output variables are exactly the same. Then it tries to find a mapping of variables from q_2 to q_1 (super-query to sub-query) preserving the result of $q_1 \sqsubseteq q_2$.

The proposed algorithm depicted in Algorithm 1 and Algorithm 2 generates two sets (one for each query), each containing variable sets, as they are related together in a query (a set of two variables occurring on both sides of an object or data property assertion pattern, or a set of arbitrary amount of variables occurring in a filter). I denote the two sets as R_1 for q_1, and R_2 for q_2. Singleton sets in R_1 and R_2 are discarded. In the example in Listing 4.2 and Listing 4.3, they would be:

$$R_1 = \{\{x, g\}, \{g, wkt\}, \{x, r\}\}\,,\ R_2 = \{\{x, g\}, \{g, wkt\}, \{x, y\}\}\,. \tag{4.14}$$

Given R_1, R_2 and M, the algorithm recursively searches all possible variable mappings from q_2 to q_1. Initially, M is the identity of the query output variables (as stated before, the output variables must be the same). To continue the example, initially,

$$M = \{(x, x), (wkt, wkt)\}. \tag{4.15}$$

The output of the algorithm is a set of all possible variable mappings, that preserve relations between variables. The algorithm assumes there are no property chains, which is true in OWL 2 QL.

The result of the algorithm for the example above would be

$$O = \{\{(x,x),(wkt,wkt),(g,g),(y,r)\}\}\,. \tag{4.16}$$

With all the possible variable mappings, they all have to be used to decide query containment; if for any mapping $M \in O$, applied to q_2, it is true that $q_1 \sqsubseteq q_{2,M}$, the answer is $q_1 \sqsubseteq q_2$, otherwise $q_1 \not\sqsubseteq q_2$.

Algorithm 1 Variable mapping algorithm.

```
function VarMap($M, R_1, R_2$)
    if size($M$) = count(variables in $q_2$) then
        return $\{M\}$
    end if
    $O \leftarrow \emptyset$
    $V_M \leftarrow \emptyset$
    for all $(v_2, v_1) \in M$ do
        $W_2 \leftarrow \emptyset$
        $W_1 \leftarrow \emptyset$
        for all $R \in R_2$ do
            if $v_2 \in R$ then
                $W_2 \leftarrow W_2 \cup (R \setminus \{v_2\} \setminus \{x : (x,y) \in M\})$
            end if
        end for
        for all $R \in R_1$ do
            if $v_1 \in R$ then
                $W_1 \leftarrow W_1 \cup (R \setminus \{v_1\} \setminus \{y : (x,y) \in M\})$
            end if
        end for
        $V_M \leftarrow V_M \cup \{(W_2, W_1)\}$
    end for
    Sort $V_M$ from the lowest max size of the two sets in each element
    $A \leftarrow$ FindCandidates($V_M, \emptyset, \emptyset, \emptyset$)
    for all $a \in A$ do
        $O \leftarrow O \cup$ VarMap($M \cup \{a\}, R_1, R_2$)
    end for
    return $O$
end function
```

The direction of mapping from q_2 to q_1 is chosen because for $q_1 \sqsubseteq q_2$ to be true, q_1 has to be always more (or the same) restrictive than q_2. It means all variables in q_2 has to be restricted in q_1 the same way as in q_2, or more; therefore no variable in q_2 can be left unmapped to q_1. The opposite is not true, some variables in q_1 need not to be mapped to q_2; they just can make q_1 more restrictive.

The algorithm always finds the mapping M, for which $q_1 \sqsubseteq q_{2,M}$, when $q_1 \sqsubseteq q_2$. It is a way how to reduce the number of checking query containments. Testing query containment for all mappings of inner variables would take long since the number of mappings is exponential w.r.t. the number of variables.

Algorithm 2 Variable mapping algorithm, finding candidates.

```
function FindCandidates($V_M, M, U_1, U_2$)
    if $V_M = \emptyset$ then
        return $\{M\}$
    end if
    $O \leftarrow \emptyset$
    $(W_2, W_1) \leftarrow$ first element of $V_M$
    for all $(w_2, w_1) \in (W_2 \setminus U_2) \times (W_1 \setminus U_1)$ do
        $O \leftarrow O \cup$ FindCandidates($V_M \setminus \{(W_2, W_1)\}, M \cup \{(w_2, w_1)\}, U_1 \cup \{w_1\}, U_2 \cup \{w_2\}$)
    end for
    return $O$
end function
```

4.5 Using Lattice for Searching Relevant Sources

A lattice is a natural structure for representing a set of queries ordered by their containment, as the set is a partially ordered set: Every two queries are related in exactly one of three ways: $q_1 \sqsubseteq q_2$, $q_2 \sqsubseteq q_1$, or q_1 and q_2 are incomparable. Section 4.5.1 describes how to generate a lattice from a set of queries, Section 4.5.2 gives details on how to search one.

It takes some effort to build a lattice, but it is a structure suitable for efficient searching. It usually takes less query containment decisions for searching a relevant source when using a lattice, compared to the number of query containment decisions of a user's query against all available prototypical queries. This is confirmed by the experiments in Section 6.2.

A lattice is an algebraic structure, which has the least element and the greatest element (for details, see [102]). In the case of queries, the least element is the abstract query giving no results (called the *bottom*) and the greatest element is the abstract query giving all results (called the *top*).

4.5.1 Building Lattice

Algorithm 3 with the support functions in Algorithm 4 iteratively builds a lattice, where nodes represent sets of semantically equivalent queries. Each node has a set of data sources capable of answering the node's queries associated with it.

It starts with the lattice being only the root node, representing the query with the empty answer (as the top node of the lattice). It adds queries one by one, placing it into appropriate position of the lattice: if the added query *contains* a query (it is lower), but none of its children, add it as a child, as in Algorithm 3, line 28; if it is semantically equivalent to a query, unify it, as in Algorithm 3, line 10; otherwise work recursively, as in Algorithm 3, line 18. The function call initially starts with the inserted query and the root (the no-answer query) as arguments.

In the worst case, adding a query to the lattice requires the amount of query containment decisions equal to twice the size of the lattice (when caching of query containment results is in place, as the algorithm may encounter the same query pair multiple times), i.e. to build a lattice out of n prototypical queries requires the maximum of $n(n-1)$ query containment decisions. But in practical situations, it is usually less, see Section 6.2.

Algorithm 3 Lattice construction algorithm.

```
Require: $r \sqsubseteq q$
function InsertIntoLattice($q, r$)
    if $q \in$ children($r$) then
        return
    end if
    doAdd ← true
    inserted ← false
    for all $c \in$ children($r$) do
        if $q \sqsubseteq c$ then                              ▷ $r \sqsubseteq q \sqsubseteq c$
            if $c \sqsubseteq q$ then                          ▷ $r \sqsubseteq q \equiv c$
                unify($c, q$)
                return
            end if
            children($r$) ← children($r$) \ {$c$}
            children($q$) ← children($q$) ∪ {$c$}
            inserted ← true
            break
        else if $c \sqsubseteq q$ then                         ▷ $r \sqsubseteq c \sqsubseteq q$
            InsertIntoLattice($q, c$)
            doAdd ← false
        end if
    end for
    if doAdd then
        if not inserted then
            for all $c \in$ children($r$) do
                ConnectToChildren($q, c$)
            end for
        end if
        children($r$) ← children($r$) ∪ {$q$}
    end if
end function
```

Algorithm 4 Support functions for lattice construction algorithm.

```
Require: $q$ not comparable to $r$
function ConnectToChildren($q, r$)
    for all $c \in$ children($r$) do
        if $q \sqsubseteq c$ then                                  ▷ $q \sqsubseteq c$
            if not ChildrenContain($q, c$) then
                for all $x \in$ children($q$) do
                    if ChildrenContain($c, x$) then
                        children($q$) $\leftarrow$ children($q$) $\setminus \{x\}$
                    end if
                end for
                children($q$) $\leftarrow$ children($q$) $\cup \{c\}$
            end if
        else
            ConnectToChildren($q, c$)                   ▷ $q$ not comparable to $c$
        end if
    end for
end function
function ChildrenContain($r, x$)
    return $\bigvee_{c \in \text{children}(r)} ((x = c) \vee$ ChildrenContain($c, x$)$)$
end function
```

4.5.2 Searching Lattice

Algorithm 5 contains the algorithm for searching a user's query in the lattice constructed in Section 4.5. The function is called with the query searched for and the root of the lattice (the no-answer query) as the arguments, then it recursively searches the lattice. It returns all query nodes, which are equivalent to the user's query, or are the “directly” contained ones (which are contained in the user's query, but no other contained in the user's query contains them). In case the algorithm cannot recursively continue at the root level, which means that the user's query does not contain any of the children of root (and hence it would not contain any query in the lattice), it switches to the “splitting” mode (it continues with the Algorithm 6).

When in “splitting” mode, the user's query is split to subqueries, which are individually searched in the lattice.

Searching a lattice of n nodes should take, in the worst case, n query containment decisions in case the splitting is not used, and $n + n|q|$ query containment decisions when the splitting is used (where $|q|$ is the number of triples in q). This should happen only when the lattice is of a deformed shape (e.g. all nodes are children of the root, or it is a chain), in practical situations, it is less, see Section 6.2.

There exist many strategies how to split a query to subqueries in order to find sources capable of answering them. It is a compromise between how many sources must be involved in answering the user's query (which includes

Algorithm 5 Lattice searching algorithm.

```
Require: $r \sqsubseteq q$
function SearchLattice($q, r$)
    $S \leftarrow \{c \in \text{children}(r) : c \sqsubseteq q\}$        ▷ $r \sqsubseteq c \sqsubseteq q$
    if $S = \emptyset$ then
        if $r =$ root of the lattice then
            return UseSplitting($q, r$)
        else
            return $\{(q, r)\}$
        end if
    else
        return $\{\text{SearchLattice}(q, c) : c \in S\}$
    end if
end function
```

how much data must be transferred from the sources to the broker and how much processing the broker has to do to combine them) and the extensiveness of the answer (the found query is always contained in the user's query, but the query formed by combining queries from multiple sources may give more results to the user's query than a single query from an individual source).

One decision is when to do the splitting of the user's query, another one is when to try to join the subqueries back when some of them can be answered by a single source. The decision may be complex with different optimizations, and it is part of my future work.

Currently, I propose a simple approach to split the user's query at the first level (the children of the lattice root) and then join those split subqueries which contain the same child of the root, see Algorithm 6. This reduces both the number of query containment decisions and the number of produced subqueries (i.e. sources necessary to use for the complete answer).

Note that the auxiliary function "join" of a set of subqueries simply returns a new query built from all the subqueries joined (the union of their triples) with the output variables being the union of the queries' output variables. The function "vars" returns a set of all variables appearing in a query or a triple. The symbol $q_1 \subseteq q_2$ represents the syntactic containment, i.e. whether the triples of q_1 are a subset of the triples of q_2 and similarly for their output variables.

Algorithm 6 Trying to split in lattice searching.

```
Require: $r \sqsubseteq q$
function UseSplitting($q, r$)
    $X \leftarrow \{(c, s_j) : c \in \text{children}(r) \wedge s_j = \text{join}(\{s \in \text{Split}(q) : c \sqsubseteq s\})\}$    ▷ $r \sqsubseteq c \sqsubseteq s$
    return $\{\text{SearchLattice}(s_j, c) : (c, s_j) \in X \wedge s_j \notin \text{Prune}(\{s_j : (c, s_j) \in X\})\}$
end function
```

The algorithm for the splitting of a query in Algorithm 7 splits the query into a set of subqueries, each subquery formed by a disjoint subset of triples of the original query. Every such triple subset is selected to be of minimal size, but keeping the condition that for every triple t in the subset, all triples sharing a non-output variable with t are in the same subset (a non-output variable is a variable not appearing in the original query SELECT statement).

This is illustrated in Fig. 4.4, where the query formed by all triples in the figure is the algorithm input, and the three subqueries represented by the triples in the encircled regions A, B, and C are the algorithm output.

In theory, a query could be split into single triples, and the broker could let those triples be answered individually by different sources; then the broker would have to do the conjunction itself, requiring that the individuals across different sources match – even the ones representing the user query's non-output variables. As I consider posing less requirements about the alignment between individuals in different sources as good practice, I proposed an algorithm to keep the originally inner variables not exposed by the split subqueries, just to require alignments on the output variables. It is also one way how to limit the number of queries to ask and to reduce the computational demands on the broker. But it is a matter of design choice.

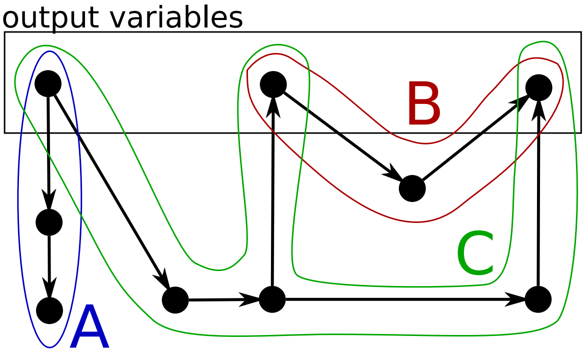


Figure 4.4: Splitting a query into subqueries. The dots represent distinct variables, the arrows represent predicates.

Algorithm 8 is another example of heuristics how to reduce the number of queries necessary to be answered by the sources. Note that by the symbol $|q|$ I denote the number of triples in a query q. First, it finds all query subsets which are redundant (note that $\mathcal{P}$ denotes a power set). Next, comes the heuristics. From these redundant subsets, I pick only the ones with the maximum number of elements (line 3), from these I pick only the ones with the minimum total number of triples with the intent of pruning the least restrictive queries (line 4). When there is more than one such subset available, I pick randomly.

Algorithm 7 Query splitting.

```
function Split($q$)
    $T \leftarrow$ all triples of $q$
    $v_o \leftarrow$ output variables of $q$
    $O \leftarrow \emptyset$
    while $T \neq \emptyset$ do
        $t \leftarrow$ one of $\{t \in T : \text{vars}(t) \cap v_o \neq \emptyset\}$
        $S \leftarrow \text{Spread}(t, T \setminus \{t\}, v_o)$
        $T \leftarrow T \setminus S$
        $O \leftarrow O \cup \{$a query with triples $S$ and output variables $\text{vars}(S) \cap v_o\}$
    end while
    return $O$
end function
function Spread($t, T, v_o$)
    $X \leftarrow \{x \in T : \text{vars}(t) \setminus v_o \cap \text{vars}(x) \neq \emptyset\}$
    return $\{t\} \cup \bigcup_{x \in X} \text{Spread}(x, T \setminus X, v_o)$
end function
```

Algorithm 8 Prunning queries.

```
function Prune($Q$)
    $P \leftarrow \{p \in \mathcal{P}(Q) : \text{join}(p) \subseteq \text{join}(Q \setminus p)\}$
    $P_1 \leftarrow \{p \in P : |p| = \max(\{|p| : p \in P\})$
    $P_2 \leftarrow \{p \in P_1 : \sum_{q \in p} |q| = \min(\{\sum_{q \in p} |q| : p \in P_1\})$
    return random element of $P_2$
end function
```

Chapter 5

Implemented Prototypes

Currently, OnGIS is implemented in two independent prototypes. The first, presented in Section 5.1, consists of a web user interface, allowing the two query input methods described in Section 4.3, and a backend translating queries to various GIS services presented in Section 4.2.

The main language used is Oracle Java 7[1]. The prototype in Section 4.3 is designed as a Java EE 6[2] server application, being run on an Oracle GlassFish 3[3] server. It uses many external libraries. Some parts of the frontend are developed using Javascript.

The second prototype in Section 5.2 demonstrates representing a set of geospatial services with prototypical queries and the algorithms for searching them according to a user's query.

5.1 Query Input and Data Retrieval

The OnGIS frontend consists of web pages for the two presented modes of entering a query. It is developed using the JavaServer Faces (JSF) 2[4] technology, with some code in Javascript, to enhance its interactivity. The web pages present the results of a query in the form of an interactive map. The map is based on the OpenLayers[5] library, which is a Javascript-based interactive map library supported in modern browsers, using both raster and vector graphics. It supports multiple map layers, which can be read from many raster and vector formats. When augmented with the Proj4js[6] Javascript coordinate transformations library, it can overlay vector data layers in different spatial reference systems.

The data retrieval backend has a modular design, there is a plugin for each supported GIS service technology. There is the OnGIS plugin API, containing methods for searching entities by a string, and for evaluating a

[1] `https://docs.oracle.com/javase/7/docs/`, cit. 2016-07-11.

[2] `http://docs.oracle.com/javaee/6/index.html`, cit. 2016-07-11.

[3] `https://glassfish.java.net/download-archive.html`, cit. 2016-07-11.

[4] `http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html`, cit. 2016-07-11.

[5] `http://openlayers.org/`, cit. 2016-07-10.

[6] `http://proj4js.org/`, cit. 2016-07-10.

GeoSPARQL query. The backend loads a background ontology into memory, to be available for query translations and for mapping to respective sources by the plugins (using the OnGIS annotations). For the background ontology manipulation, the OWL API[7] (version 3.4.9), a Java library for OWL ontologies manipulation, is used.

Currently, there are three basic plugins implemented and used:

- **OwlgresMM plugin** uses OwlgresMM, see Section 5.1.1. It translates GeoSPARQL queries to SQL queries for appropriate databases. It uses the annotations `anno:DBDatabase` and others (see Section 4.2).
- **ArcGIS plugin** is used for retrieving spatial data from ArcGIS servers. It is driven by the `anno:ArcgisWebservice` and `anno:ArcgisRest` annotations.
- **SPARQL plugin** does a simple GeoSPARQL to SPARQL rewriting. It is enabled by the `anno:SparqlEndpoint` annotation.

Currently, the data retrieval backend has a simple algorithm for splitting a query to parts based on the provided OnGIS annotations. When entities in a part of a query are annotated by a plugin annotations, that part of the query is presented to the plugin, translated, and answered by a source. This may produce unnecessary queries to data sources (e.g. when a data source would answer the query entirely, another one is asked as well, just because it is connected by the OnGIS annotations, too). This can be solved by incorporating the prototype presented in Section 5.2. Joining the two prototypes remains as a future work.

The ArcGIS plugin must greatly simplify the query, as ArcGIS server querying is rather limited (at least of the one available for testing, the geoportal of IPR Praha). It supports attribute filters, a result bounding box, and a spatial restriction: one of *intersects*, *envelope intersects*, *index intersects*, *touches*, *overlaps*, *crosses*, *within*, and *contains*[8] with a given geometry. Therefore some parts of a GeoSPARQL query can be translated: simple data property restrictions (using equality), and some spatial relations compatible with the mentioned supported ones. Note that some other practical restrictions can be translated to the supported ones, e.g. "within a distance d to a point p" can be translated by making a circle with the radius d and the center p, which can be used for the intersection relation.

The translation into SPARQL is even more simplifying, as basic geospatial support in RDF and SPARQL is for W3C Basic Geo Vocabulary only, having two floating point numbers of latitude and longitude in WGS 84 reference system only, with only the number comparison operations ($\leq$, $\geq$, $\ldots$) available. Therefore, only the bounding box restriction can be made this way,

[7] `http://owlapi.sourceforge.net/`, cit. 2016-07-10.

[8] Their definitions is given in the ArcGIS guide at `http://resources.arcgis.com/en/help/arcobjects-net/componenthelp/index.html#//002500000086000000`, cit. 2016-07-11.

which can be used as a prefilter for some simple queries, with proper filtering done as a local step in OnGIS after receiving the prefiltered data.

I will present the details of OwlgresMM plugin in the next section.

5.1.1 OwlgresMM

The OwlgresMM plugin uses OwlgresMM, an OWL 2 QL reasoner based on Owlgres. Owlgres[9] [57] is an open source Java implementation of a $DL\!-\!Lite^{\mathcal{H}}_{core}$ reasoner developed by Clark & Parsia[10], backed by a DBMS for persisting data, particularly it is tailored to work with PostgreSQL databases.

The original database schema used in Owlgres has two sets of database tables, one for TBox, with one table listing all classes, one for all object properties, etc., and one set for ABox. This original schema for ABox stores all class assertions in one table, all object property assertions in another table, and similarly for data properties and annotations.

Owlgres was first extended in [103] (my master thesis), where two other database schemas were designed, implemented and compared to the original one in Owlgres. The result of the master thesis is that for most cases, storing class assertions (respectively object and data property assertions) in separate tables per named class (respectively named object and data property), which is one of the new schemas, is the most efficient option.

OnGIS uses OwlgresMM (MM stands for *multiple tables scheme, with custom mappings*), developed as part of this thesis. It is my extension of Owglres based on the most efficient schema according to the master thesis, with three important new features added. The first is that it supports custom mappings between OWL entities and database tables. These mappings are currently defined by the OnGIS annotations `anno:DBClassTable` and others, as described in Section 4.2. This is useful especially when we want to connect OnGIS to an existing database with an already defined schema. Another change is that a TBox is not stored in a database at all (as was the case of original Owlgres), but it is loaded from an ontology document (e.g. an RDF/XML) into memory by OWL API instead, as it is typically feasible to fit it into memory, where it is handy both for query translation and for looking up OnGIS mapping annotations.

The final extension is the support of spatial queries. OwlgresMM supports some of GeoSPARQL vocabulary, e.g. spatial relations, distance filter, and bounding box (see Section 3.4.2). Currently, it supports only PostgreSQL with the PostGIS extension – the GeoSPARQL operations are translated to the PostGIS functions.

Listing 5.1 and Listing 5.2 show some excerpts of OpenStreetMap and GeoNames TBoxes in RDF/XML format with the database mapping annotations.

[9] `http://pellet.owldl.com/owlgres`, cit. 2011-10-07, currently unavailable, and unfortunately no other official distribution of the source codes and binaries was found.

[10] `http://clarkparsia.com/`, cit. 2011-10-07, rebranded as Complexible Inc. in 2015, `http://complexible.com/`, cit. 2016-07-06.

Listing 5.1: An excerpt of OpenStreetMap mapping ontology.

```
<rdf:Description
    rdf:about="http://linkedgeodata.org/ontology/HighwayResidential">
  <anno:DBClassTable>
    highw_residential(id=id,geom=way,geomSRID=4326,geomType=LINESTRING)
  </anno:DBClassTable>
</rdf:Description>
<rdf:Description rdf:about="http://linkedgeodata.org/ontology/NaturalWater">
  <anno:DBClassTable>
    nat_water(id=id,geom=way,geomSRID=4326,geomType=GEOMETRY)
  </anno:DBClassTable>
</rdf:Description>
<rdf:Description
    rdf:about="http://linkedgeodata.org/ontology/official_name">
  <anno:DBDataPropertyTable>
    off_names(id=id,val=val,lang=,length=255,nullable=false,indexed=true,
    multiple=true)
  </anno:DBDataPropertyTable>
</rdf:Description>
<rdf:Description rdf:about="http://linkedgeodata.org/ontology/Bus">
  <anno:DBClassTable>
    route_bus(id=id,geom=way,geomSRID=4326,geomType=GEOMETRY)
  </anno:DBClassTable>
</rdf:Description>
<rdf:Description rdf:about="http://linkedgeodata.org/ontology/subarea">
  <anno:DBObjectPropertyTable>
    subareas(id=id,obj=obj,nullable=false,indexed=true,multiple=true)
  </anno:DBObjectPropertyTable>
</rdf:Description>
```

Listing 5.2: An excerpt of GeoNames mapping ontology.

```
<rdf:Description rdf:about="http://www.geonames.org/ontology#Feature">
  <anno:DBClassTable>
    features(id=id,geom=pt,geomSRID=4326,geomType=POINT)
  </anno:DBClassTable>
</rdf:Description>
<rdf:Description rdf:about="http://www.geonames.org/ontology#name">
  <anno:DBDataPropertyTable>
    features(id=id,val=name,lang=,length=200,nullable=false,indexed=true,
    multiple=false)
  </anno:DBDataPropertyTable>
</rdf:Description>
<rdf:Description rdf:about="http://www.geonames.org/ontology#population">
  <anno:DBDataPropertyTable>
    features(id=id,val=population,lang=,length=12,nullable=true,
    indexed=true,multiple=false)
  </anno:DBDataPropertyTable>
</rdf:Description>
<rdf:Description rdf:about="http://www.geonames.org/ontology#shortName">
  <anno:DBDataPropertyTable>
    shortnames(id=id,val=val,lang=lang,length=200,nullable=false,
    indexed=true,multiple=true)
  </anno:DBDataPropertyTable>
</rdf:Description>
```

5.1.2 Importing Spatial Data to Relational Databases

For testing the OnGIS prototype, two databases have been set up and mapped: one with OpenStreetMap data, mapped to LinkgedGeoData TBox, and another one with GeoNames data, mapped to its TBox. The OpenStreetMap data are also available in RDF format for download from LinkedGeoData website[11], however they are released approximately once a year, so using the original OpenStreetMap source has the benefit of having up-to-date data. There are also alternative OpenStreetMap data download servers, which have the data split into continents and countries, so it is easy to obtain only the part of the World relevant to whatever use case.

To import the data, special tools had to be developed to read the respective data source formats and load it to spatially-enabled databases.

In case of the OpenStreetMap data, the basic idea is that all instances of each concept should be in a separate table (corresponding to the concept), and similarly for object and data properties (all pairs of objects of an object property in a single table, respectively all objects with their values of a data property in a single table). The reason for this decision is that there are many instances in relatively few classes and properties. This way a database schema was designed.

Because of the structure of OpenStreetMap data (nodes, ways and relations, labeled with tags; for details see [15]), a custom set of annotation properties for annotating LinkedGeoData ontology was developed. The annotations include:

- **OSMNode** – the annotation property indicating that the database table of this concept should be filled in with nodes satisfying constraints in the annotation,
- **OSMWay** – similar annotation property for ways,
- **OSMRelation** – similar annotation property for relations, and
- **OSMTag** – the annotation property indicating that the database table of this data property should be filled in with values of the specified tags.

These annotations are used only for import of OpenStreetMap data, not for querying. For querying, only the generic OnGIS annotations are used. These OpenStreetMap specific annotations are defined only because OpenStreetMap data cannot be directly accessed by SQL on a public server, therefore my own database has to be used with a custom import of the data.

To have an idea of what an OpenStreetMap XML data file looks like, Listing 5.3 shows a few simplified examples.

[11] `http://downloads.linkedgeodata.org/releases/`, cit. 2016-08-18.

Listing 5.3: OpenStreetMap data example.

```
<node id="172516" lat="50.0811476" lon="14.4045315"
    timestamp="2008-09-08T21:31:02Z">
  <tag k="highway" v="traffic_signals" />
</node>
<node id="248541" lat="50.0988063" lon="14.5984071"
    timestamp="2011-02-19T19:15:09Z">
  <tag k="highway" v="motorway_junction" />
  <tag k="name" v="Praha-Horni␣Pocernice" />
</node>
<way id="1614262" timestamp="2008-11-19T20:59:19Z">
  <nd ref="6926352"/>
  <nd ref="32606754"/>
  <nd ref="6926427"/>
  <nd ref="6926428"/>
  <nd ref="313218916"/>
  <nd ref="313218885"/>
  <nd ref="6926434"/>
  <tag k="highway" v="residential" />
  <tag k="name" v="Bakurinova" />
</way>
<relation id="1622387" timestamp="2011-06-13T00:31:24Z">
  <member type="way" ref="116788173" role="outer"/>
  <member type="way" ref="116788175" role="inner"/>
  <tag k="type" v="multipolygon" />
</relation>
```

These import annotation properties are used to perform a mapping between LinkedGeoData and OpenStreetMap XML data file, see Fig. 5.1. The developed utility reads the input XML file according to the import annotations, and stores the data into a relational database, as defined by the OnGIS annotations.

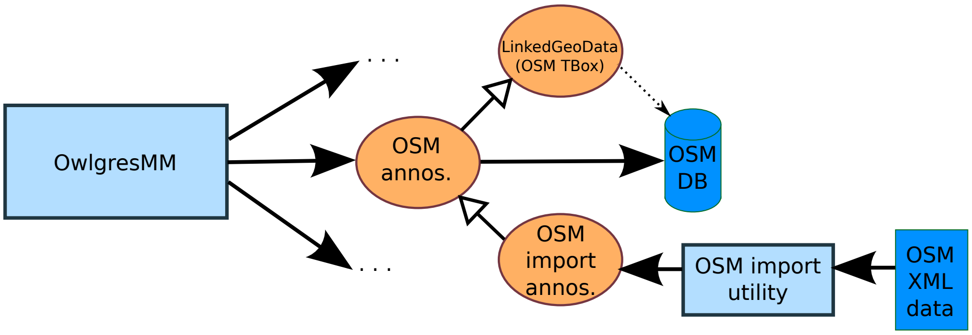


Figure 5.1: Architecture of OpenStreetMap import.

Listing 5.4 shows a few examples of OpenStreetMap import annotations in RDF/XML format. Lines 1–4 specify that the table corresponding to `lgd:HighwayResidential` class (which is specified in the LinkedGeoData ontology) should be filled in with OpenStreetMap ways. The contents of the annotation restrict, which ways should by imported using the tags (it is similar for the other objects – all nodes, ways and relations can be all labeled with tags in OpenStreetMap). It is a comma separated list of "key=value" pairs. In this case, the ways have to have a *highway* tag with the value *residential*.

Lines 5–9 do the same thing for class `lgd:NaturalWater`, only this time it can be both nodes and ways. Lines 10–13 annotate that the table for data property `lgd:official_name` should be filled in with all values of the tags with the key *official_name* (wherever they appear). And finally the lines 14–19 fill in the table of class `lgd:Bus` with specific ways and relations (importing relations is a little more complicated – there are different types of relations, and each type requires different handling; for details see [15] and the prototype implementation.

Listing 5.4: Example of OpenStreetMap import annotations.

```
 1  <rdf:Description
 2     rdf:about="http://linkedgeodata.org/ontology/HighwayResidential">
 3    <OSMWay>highway=residential</OSMWay>
 4  </rdf:Description>
 5  <rdf:Description
 6      rdf:about="http://linkedgeodata.org/ontology/NaturalWater">
 7    <OSMWay>natural=water</OSMWay>
 8    <OSMNode>natural=water</OSMNode>
 9  </rdf:Description>
10  <rdf:Description
11      rdf:about="http://linkedgeodata.org/ontology/official_name">
12    <OSMTag>official_name</OSMTag>
13  </rdf:Description>
14  <rdf:Description
15      rdf:about="http://linkedgeodata.org/ontology/Bus">
16    <OSMWay>route=bus</OSMWay>
17    <OSMRelation>
18       rel_type=route,type=route,route=bus
19    </OSMRelation>
20  </rdf:Description>
```

The main part of GeoNames data is available in two tab-separated text files, the main file with point data and other attributes, and another one with alternate names. A special utility was developed to import the data into a database and to generate the mapping. GeoNames ontology expressivity is (in the notation of description logics) $\mathcal{ALEHOIN}(\mathcal{D})$, which is beyond the expressivity of OWL 2 QL, but it can be simplified by dropping a few axioms (minimum cardinality restrictions ≥ 1 can be replaced with existential quantifications, maximum cardinality restrictions have to be dropped – this, however, does not affect typical OnGIS queries).

The main modification of GeoNames ontology, that had to be made, was regarding the feature classes and codes. In the original ontology, they are modeled as individuals, which is impractical for querying in OnGIS and for linking with other ontologies. Therefore the classes and codes were translated from individuals into classes (with the addition of correct hierarchies – that for each class, its corresponding codes are the subclasses).

As the GeoNames data are of a tabular form (the two files represent two tables), the most natural way to store them in a database is using two relational tables. This means that all spatial features (points in this case) are in the same table, regardless of the class. The class is specified by column values, which are indexed (to speed up the class instance retrieval).

The importing architecture is very similar to the one used in case of OpenStreetMap. The GeoNames ontology is annotated with three specific annotation properties for GeoNames:

- **GNFeature** – the annotation property of a concept indicating, where to store features from the main file (to a table corresponding to the concept),
- **GNFeatureDataColumn** – the annotation property of a data property specifying, where to store data values from a specific column of the main feature file; the value of this annotation is the index of the selected column (in my case, the target data property aims to the same table as where the features are stored, just another table column), and
- **GNAltName** – the annotation property of a data property defining, where alternate names from the second file should be stored, optionally with a filter in the annotation value (alternate names file column index, equal sign, and a value the imported row's column has to satisfy).

An excerpt of GeoNames import annotations is in Listing 5.5.

Listing 5.5: GeoNames import annotations.

```
<rdf:Description
    rdf:about="http://www.geonames.org/ontology#Feature">
  <GNFeature></GNFeature>
</rdf:Description>
<rdf:Description
    rdf:about="http://www.geonames.org/ontology#name">
  <GNFeatureDataColumn>1</GNFeatureDataColumn>
</rdf:Description>
<rdf:Description
    rdf:about="http://www.geonames.org/ontology#population">
  <GNFeatureDataColumn>14</GNFeatureDataColumn>
</rdf:Description>
<rdf:Description
    rdf:about="http://www.geonames.org/ontology#shortName">
  <GNAltName>5=1</GNAltName>
</rdf:Description>
```

5.2 Lattice Construction and Searching Using Query Containment

The second prototype built to test OnGIS design in Section 4.4 and Section 4.5 is a standalone Java application. It loads the background ontology into memory using OWL API, then it can be fed with the prototypical queries, from which an in-memory lattice is constructed. The prototypical queries are read one by one and are iteratively inserted into the lattice.

The background ontology format is anything OWL API can parse, e.g. RDF/XML documents. The prototypical queries are expected to be stored in a file per data source, each file containing a set of prefixes first, then the queries in the GeoSPARQL format, each GeoSPARQL query on a single line.

When the prototype has all prototypical queries and a background ontology loaded, it can be used using a simple API method for searching. The method is given a user's query and returns a map of queries to the sources that can answer the queries. In case the splitting was not used, the map contains only the original user's query. When the splitting was used, the map contains the split sub-queries.

The algorithms for deciding query containment presented in Section 4.4 and Section 4.5 require consistency checks in OWL 2 QL, For that purpose, the Pellet reasoner version 2.3.1 is used. Pellet supports OWL-DL, which has higher expressive power than OWL 2 QL, so developing an optimized OWL 2 QL reasoner might make OnGIS faster.

For deciding geometry relations to support spatial reasoning, GeoTools[12] (version 14.0) together with JTS Topology Suite[13] (version 1.13) are used. For parsing and manipulating GeoSPARQL queries, Jena ARQ[14] (version 2.10.0) is used.

I tried to redesign all the algorithms for lattice construction and searching also for the opposite direction, where the *root* in the algorithms is the lattice *top* (the all-answer query). But especially the searching algorithms are more complicated. This is confirmed by my tests when in most cases the algorithms using the lattice *top* as the root are slower. The worst difference is for the test in Section 6.2.1, where splitting is used: the search time is more than ten times longer when using the lattice *top* as the root.

5.3 Other Tools

A tool important for working with ArcGIS servers, which is the case of IPR Praha geoportal, is ArcExplorer. It has been designed along the other OnGIS prototypes to construct an ontology of an ArcGIS server.

Given an ArcGIS server URL, ArcExplorer reads the server metadata using the ArcGIS web service client and produces an ontology describing the

[12] `http://www.geotools.org/`, cit. 2016-08-20.

[13] `http://tsusiatsoftware.net/jts/main.html`, cit. 2016-08-20.

[14] `https://jena.apache.org/documentation/query/`, cit. 2016-08-20.

server. It transforms the available map servers and their layers, attributes, and map legends into classes and properties. It labels them with the obtained metadata. The OnGIS annotations are used as a vocabulary for the available ArcGIS web services, ArcGIS REST services and WMS services (not all types of services need to be available for every ArcGIS map server).

Another tool is for obtaining relations between topological relation properties, which are presented in Fig. 4.2 in Section 4.4.1. Its input is a set of topological relations with their DE-9IM definitions, and the output is a directed graph in DOT format [104]. In OnGIS, it is used for the GeoSPARQL topological relations listed in Table 3.3 in Section 3.4.2. The tool compares two DE-9IM definitions character by character and decides, whether one relation is less or equally general than the other. This is true, when all their corresponding character pairs satisfy the same condition: a DE-9IM character r_1 is less or equally general than a DE-9IM character r_2 if and only if the following is satisfied:

$$(r_1 = r_2) \lor (r_2 = \text{'}*\text{'}) \lor ((r_2 = \text{'T'}) \land \text{isDigit}(r_1))\,. \tag{5.1}$$

The DE-9IM characters can have values 0, 1, 2, T, F, and $*$. For their description, see Section 2.2.

Chapter 6

Experiments

This chapter shows how the prototypes described in Chapter 5 perform on a few examples.

6.1 Query Input

As an example showing, how the query input works in the prototype, I chose a simple user's query: "Find me all restaurants in Prague 2 near to a park", where Prague 2 is a city district. I will use this example both in Section 6.1.1 and Section 6.1.2. Let us say that "near" means within 150 meters to us.

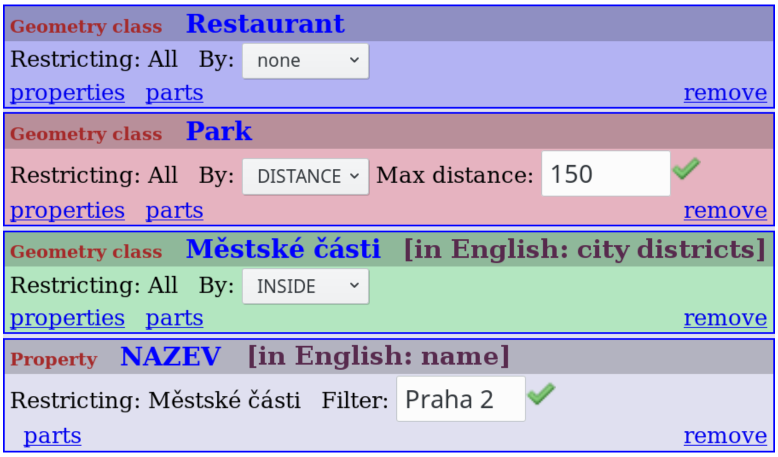
The data sources connected to the OnGIS prototype for the examples are:

- IPR Praha ontology (generated by ArcExplorer, see Section 5.3) linked to IPR Praha ArcGIS geoportal,
- GeoNames ontology mapped to local PostgreSQL database with GeoNames data,
- LinkedGeoData ontology mapped to local PostgreSQL database containing OpenStreetMap data for the Czech Republic, and
- DBpedia ontology linked (by the `anno:SparqlEndpoint` annotation) to the DBpedia SPARQL endpoint.

6.1.1 Query Input as List with Relations

To construct such query in the list form, a user must enter each part of the query individually into a text field and search for relevant terms from the loaded background ontologies. For example, the user enters the term *restaurant*, and OnGIS presents him with a list of results containing the term. One of the results is the `Restaurant` class from the LinkedGeoData ontology. The user can select it and add it to the query list, as in Fig. 6.1. Similarly, the user can deal with parks.

To restrict the results to the city district Prague 2, the user can try searching for city districts. In our case, we have also an ontology describing IPR Praha geoportal loaded into the background ontology, which contains a city districts



Geometry class **Restaurant**
Restricting: All By: none
properties parts remove

Geometry class **Park**
Restricting: All By: DISTANCE Max distance: 150 ✓
properties parts remove

Geometry class **Městské části [in English: city districts]**
Restricting: All By: INSIDE
properties parts remove

Property **NAZEV [in English: name]**
Restricting: Městské části Filter: Praha 2 ✓
parts remove

Figure 6.1: The list query with a few items.

layer ("městské části" in Czech). To create the IPR Praha geoportal ontology, the OnGIS tool ArcExplorer has been used; for details see Section 5.3. Since only Prague 2 is queried, the user has to restrict the city districts somehow. An easy way to do this is to get all properties of the districts class by clicking the *properties* link, which shows a popup with all properties having the class as the domain. Among others, there is also the *name* property ("název" in Czech). It can be added to the query list, where it can be assigned a filtering value ("Praha 2", meaning Prague 2).

As we have all items of our query in the list, we can add the spatial restrictions. The user can simply set the *distance* restriction of 150 meters to the parks class, which restricts all other search results to be within 150 m to a park. The Prague 2 restriction simply requires setting the *inside* restriction to the city districts class.

This list query produces GeoSPARQL queries in Listing 6.1 and Listing 6.2 posed to OwlgresMM for retrieving restaurants (the first one) and parks (the second one). The prototype also generated GeoSPARQL queries for obtaining the city districts, but since these cannot be answered from connected relational databases, OwlgresMM has discarded them.

The prefixes in the listings have been omitted for brevity. There is the standard GeoSPARQL `geo:` prefix used, along with `geof:` for GeoSPARQL functions. The prefix `geox:` is my simple extension of GeoSPARQL with the bounding box restriction function (`geox:bbox`).

Note that the truncated WKT polygon geometry literal with ellipsis is a geometry representing the city district Prague 2, which is obtained from the IPR Praha ArcGIS geoportal using an ArcGIS web service. The bounding box is used to restrict the results only to the currently visible map extent; the coordinates for the restriction are obtained from the current zoom and pan of the map viewing component in the user's browser. This helps to restrict the amount of data needed to query and transfer from the database to the front end.

To allow displaying also a label next to a geometry in the user's map, a simple method is used in the prototype. There is the `ongis:name` data property, which can be linked to domain specific data properties for various types of names by sub-property axioms. For example, `lgd:official_name`, `lgd:short_name`, and `lgd:historic_name` are sub-properties of `ongis:name`. Therefore, OwlgresMM plugin adds this data property to the optional part of every GeoSPARQL query, to obtain also some textual information to be displayed on the map.

Listing 6.1: Generated GeoSPARQL searching for restaurants.

```
SELECT DISTINCT ?x ?g ?l WHERE {
  ?x a <http://linkedgeodata.org/ontology/Restaurant>.
  ?x geo:asWKT ?g.
  ?fdo2 a <http://linkedgeodata.org/ontology/Park>.
  FILTER (geof:sfWithin(?x,
    "POLYGON((14.437196␣50.086591,␣14.437122␣50.086504,␣...))")).
  FILTER (geof:sfWithin(?fdo2,
    "POLYGON((14.437196␣50.086591,␣14.437122␣50.086504,␣...))")).
  FILTER (geof:distance(?x, ?fdo2, 150.0)).
  FILTER (geox:bbox(?x,
    "14.251363,50.032442,14.615151,50.133983")).
  FILTER (geox:bbox(?fdo2,
    "14.251363,50.032442,14.615151,50.133983")).
  OPTIONAL {
    ?x ongis:name ?l.
  }
}
```

Listing 6.2: Generated GeoSPARQL searching for parks.

```
SELECT DISTINCT ?x ?g ?l WHERE {
  ?x a <http://linkedgeodata.org/ontology/Park>.
  ?x geo:asWKT ?g.
  FILTER (geof:sfWithin(?x,
    "POLYGON((14.437196␣50.086591,␣14.437122␣50.086504,␣...))")).
  FILTER (geox:bbox(?x,
    "14.251363,50.032442,14.615151,50.133983")).
  OPTIONAL {
    ?x ongis:name ?l.
  }
}
```

OwlgresMM takes these two GeoSPARQL queries and translates them using the mapping annotations to SQL queries in Listing 6.3 and Listing 6.4, which are executed on the OpenStreetMap database.

As the LinkedGeoData class for restaurants, which is used in the query, is linked also to a restaurant class in the GeoNames ontology, querying only for restaurants would yield SQL queries both for OpenStreetMap and GeoNames data. But since the entered query is more complex, requiring the spatial restriction by parks, the simple rewriting mechanism in OwlgresMM used only the OpenStreetMap database, as it contains parks as well. GeoNames also contain a class for parks (`gn:L.PRK`), but it is not linked to `lgd:Park` for some reason.

The two SQL queries are truncated in the listings, the full versions are series of unions, where only the name data property table name differs – `d_official_name` on line 20 in Listing 6.3 and line 12 in Listing 6.4 is substituted with `d_short_name`, etc.

Listing 6.3: OwlgresMM generated SQL (truncated) for OpenStreetMap database for obtaining restaurants.

```
 1  SELECT DISTINCT subinner.x1 AS x1, subinner.x2 AS x2, dra_3.val AS x3 FROM (
 2    SELECT DISTINCT ST_AsText(ST_Transform(ca_1.way, 4326)) AS x1,
 3      ca_1.id AS x2
 4    FROM restaurant ca_1, park ca_2
 5    WHERE ST_Within(ST_Transform(ca_1.way, 4326),
 6      ST_SetSRID(ST_GeomFromText('POLYGON␣((14.437196␣50.086591,␣...))'),
 7        4326))
 8    AND ST_Within(ST_Transform(ca_2.way, 4326),
 9      ST_SetSRID(ST_GeomFromText('POLYGON␣((14.437196␣50.086591,␣...))'),
10        4326))
11    AND ST_DWithin(ST_Transform(ca_1.way, 102067),
12      ST_Transform(ca_2.way, 102067), 150.0)
13    AND ST_Transform(ca_1.way, 4326) &&
14      ST_SetSRID(ST_MakeBox2D(ST_Point(14.251363, 50.032442),
15        ST_Point(14.615151, 50.133983)),4326)
16    AND ST_Transform(ca_2.way, 4326) &&
17      ST_SetSRID(ST_MakeBox2D(ST_Point(14.251363, 50.032442),
18        ST_Point(14.615151, 50.133983)),4326)
19  ) subinner
20  LEFT JOIN d_official_name dra_3 ON subinner.x2=dra_3.id
21  UNION
22  ...
```

Listing 6.4: OwlgresMM generated SQL (truncated) for OpenStreetMap database for obtaining parks.

```
 1  SELECT DISTINCT subinner.x1 AS x1, subinner.x2 AS x2, dra_2.val AS x3 FROM (
 2    SELECT DISTINCT ST_AsText(ST_Transform(ca_1.way, 4326)) AS x1,
 3      ca_1.id AS x2
 4    FROM park ca_1
 5    WHERE ST_Transform(ca_1.way, 4326) &&
 6      ST_SetSRID(ST_MakeBox2D(ST_Point(14.251363, 50.032442),
 7        ST_Point(14.615151, 50.133983)),4326)
 8    AND ST_Within(ST_Transform(ca_1.way, 4326),
 9      ST_SetSRID(ST_GeomFromText('POLYGON␣((14.437196␣50.086591,␣...))'),
10        4326))
11  ) subinner
12  LEFT JOIN d_official_name dra_2 ON subinner.x2=dra_2.id
13  UNION
14  ...
```

Because the LinkedGeoData restaurants class is also linked to the class of restaurants in DBpedia ontology, the OnGIS prototype issues a request to DBpedia SPARQL endpoint, see Listing 6.5. Note that the query is rather

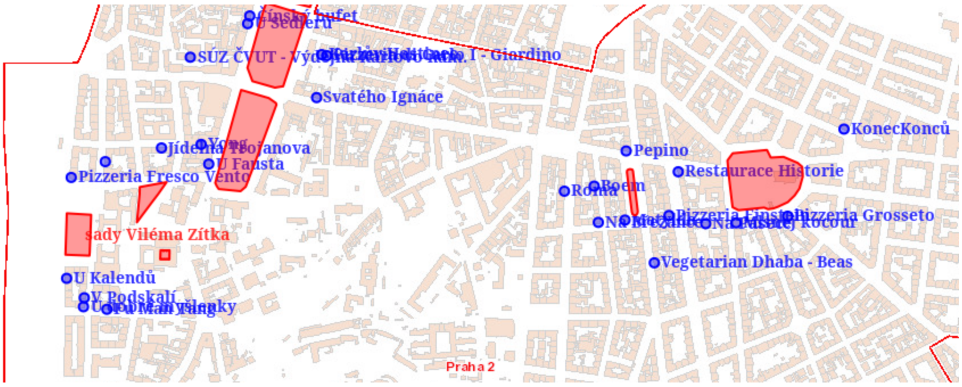


Figure 6.2: The resulting map of the query example.

simple, not performing the necessary spatial restrictions, because SPARQL itself is rather limited regarding spatial features. Therefore the necessary restrictions have to be applied after receiving the data, but this is currently not supported by the current OnGIS prototypes.

Note that the SPARQL query uses W3C Basic Geo Vocabulary for querying longitude and latitude of points and that the longitude and latitude are filtered to the bounding box of the map view extent, similarly to the GeoSPARQL queries earlier.

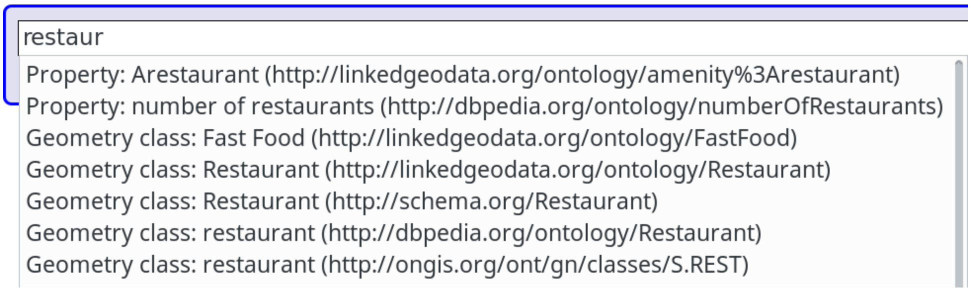
Listing 6.5: Generated SPARQL for DBpedia SPARQL endpoint querying for restaurants.

```
SELECT DISTINCT ?x ?lat ?lon ?l WHERE {
  ?x a <http://dbpedia.org/ontology/Restaurant>.
  ?x <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ?lat.
  ?x <http://www.w3.org/2003/01/geo/wgs84_pos#long> ?lon.
  FILTER (?lon >= 14.277582). FILTER (?lon <= 14.458047).
  FILTER (?lat >= 50.060109). FILTER (?lat <= 50.110843).
  OPTIONAL {
    ?x rdfs:label ?l.
    FILTER (LANG(?l)="en").
  }
}
```

Fig. 6.2 shows a map with the query results. It has been cut to make its labels legible. The red line is the Prague 2 boundary, the red areas are parks, and the blue points are restaurants. The two bigger parks on the left form Charles Square (Karlovo náměstí), while the park on the right is on Peace Square (náměstí Míru).

6.1.2 Query Input as Structured Expression

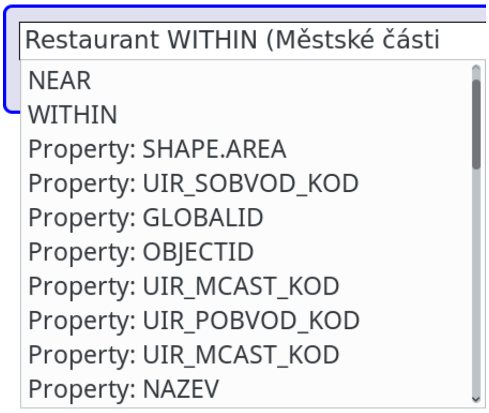
The structured way of entering a query uses one input text field, which does not allow submitting free text, but checks the input using autocomplete.



restaur

Property: Arestaurant (http://linkedgeodata.org/ontology/amenity%3Arestaurant)
Property: number of restaurants (http://dbpedia.org/ontology/numberOfRestaurants)
Geometry class: Fast Food (http://linkedgeodata.org/ontology/FastFood)
Geometry class: Restaurant (http://linkedgeodata.org/ontology/Restaurant)
Geometry class: Restaurant (http://schema.org/Restaurant)
Geometry class: restaurant (http://dbpedia.org/ontology/Restaurant)
Geometry class: restaurant (http://ongis.org/ont/gn/classes/S.REST)

Figure 6.3: An example of restaurant classes autocomplete in a structured query.



Restaurant WITHIN (Městské části

NEAR
WITHIN
Property: SHAPE.AREA
Property: UIR_SOBVOD_KOD
Property: GLOBALID
Property: OBJECTID
Property: UIR_MCAST_KOD
Property: UIR_POBVOD_KOD
Property: UIR_MCAST_KOD
Property: NAZEV

Figure 6.4: An example of a data properties autocomplete in a structured query.

When at least three characters are entered, an autocomplete list box starts suggesting entities from the background ontology and items found in the connected sources. Let me show this using a similar user's query like in the previous section.

First, the user types *restaurant* and picks the LinkedGeoData class from the autocomplete, as in Fig. 6.3. Then the user wants to enter the city district restriction – the restaurants need to be in Prague 2. For spatial restrictions, the input currently accepts two keywords, similarly to the list query in the previous section: *WITHIN* and *NEAR*. So the user enters *WITHIN* (with the aid of the autocomplete). As the city districts class has to be restricted to Prague 2, the expression is more complex, requiring parentheses. After an opening parenthesis and selecting city districts ("městské části") using the autocomplete, the autocomplete suggests data properties with the city districts domain right away (without typing anything), so it is easy to restrict it to the city district Prague 2 ("Praha 2"), as in Fig. 6.4. The data property value is double-quoted.

Similarly, the user enters the distance to parks restriction using the *NEAR* keyword. This keyword has a second parameter, the distance in meters (150 in our case). Entering "NEAR Park 150" would, therefore, restrict the restaurants to by within 150 meters from any park. But to be consistent

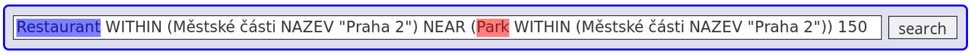

Restaurant WITHIN (Městské části NAZEV "Praha 2") NEAR (Park WITHIN (Městské části NAZEV "Praha 2")) 150

Figure 6.5: An example structured query input field with a complete query, where relevant parts have colored background corresponding to the map symbols.

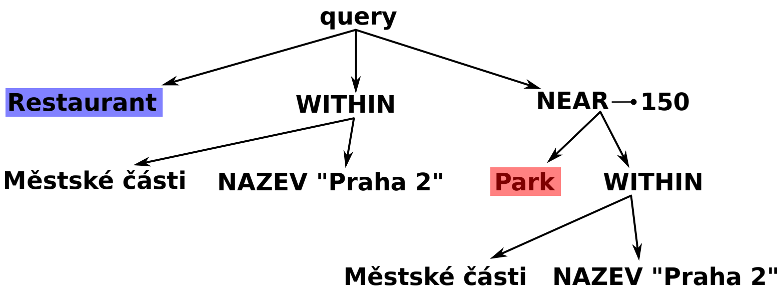


Figure 6.6: The tree of an example structured query.

with the previous example with the list query, let the user specify that the restaurants have to be within 150 meters from parks in Prague 2. The resulting complete structured query is in Fig. 6.5. The colored parts correspond to the colors in the map.

To understand the structure of the query, the tree representation of the query is in Fig. 6.6. A map of the results of this query looks the same as of the query in Section 6.1.1, see Fig. 6.2. Also, the generated GeoSPARQL queries for OwlgresMM are the same as in Listing 6.1 and Listing 6.2.

6.2 Searching Sources for a Query

This section gives three successfully tested examples using the prototype described in Section 5.2 and a comparison to another system. The first example does not feature any spatial restriction and uses splitting, the second example has no spatial restriction and does not use splitting, and the third example has spatial restrictions and does not use splitting. The size of the examples increases in the order they are listed.

Besides the listed examples, the prototype has been unit-tested by a set of small artificial test cases, each checking for a different aspect of the algorithms: if the splitting is used in the right situations, whether the topological relations reasoning works correctly, if the variable mapping works, whether the filter containment is decided correctly (currently only distance is supported), etc.

The background ontology of all examples in this section includes the GeoSPARQL ontology, including my extension presented in Section 4.4.1.

6.2.1 The First Example: Restaurants in Historical Buildings

Here is the first example, how the lattice construction and searching works. Consider the following background ontology $\mathcal{O}$ consisting of:

- LinkedGeoData ontology (prefix `lgd:`), which describes OpenStreetMap data. Among many others, it contains classes `lgd:Restaurant`, `lgd:Gym`, and `lgd:HistoricBuilding`, which are relevant to this example. It also contains the axiom `lgd:HistoricBuilding` $\sqsubseteq$ `lgd:Historic`.
- GeoNames ontology (prefix `gn:`), describing its own well-classified database of points. It is linked to LinkedGeoData. The following classes are relevant: `gn:S.REST` (restaurants) and `gn:S.HSTS` (historical sites). Also the following axioms are included: `lgd:Restaurant` $\equiv$ `gn:S.REST` and `lgd:Historic` $\equiv$ `gn:S.HSTS`
- My example ontology for gourmets (prefix `ex:`). It only contains one object property, `ex:hasFoodOrigin`.

Note that for classes C and D, $C \equiv D$ is only syntactic sugar for $C \sqsubseteq D$ and $D \sqsubseteq C$.

I will consider the following geospatial sources (services providing geospatial data, e.g. WFS servers, (Geo)SPARQL endpoints, etc.) for this example:

- s_{lgd}, having OpenStreetMap data, capable of answering queries with the following restrictions:
  - `lgd:Restaurant`$(?x)$
  - `lgd:Gym`$(?x)$
  - `lgd:Gym`$(?x)$, `lgd:HistoricBuilding`$(?x)$
  - `lgd:HistoricBuilding`$(?x)$
- s_{gn}, having GeoNames data, capable of answering queries with the following restrictions:
  - `gn:S.REST`$(?x)$
  - `gn:S.REST`$(?x)$, `gn:S.HSTS`$(?x)$
  - `gn:S.HSTS`$(?x)$
- s_{ex}, having example gourmet data, capable of answering queries with the following restrictions:
  - `ex:hasFoodOrigin`$(?x, ?c)$
  - `ex:hasFoodOrigin`$(?x, ?c)$, `lgd:Restaurant`$(?x)$

When OnGIS is loaded with the background ontology $\mathcal{O}$ and the sources s_{lgd}, s_{gn}, s_{ex}, the prototype creates the lattice in Fig. 6.7. Note that the root node *bottom* represents the no-answer query and all the prefixes are omitted for compactness. The abbreviation R stands for `Restaurant`, G for `Gym`, *HB* for `HistoricBuilding`, and *FO* for `hasFoodOrigin`. As it is a lattice, all nodes with no children depicted should be connected to the *top* node at the bottom, representing the all-answer query. But it has been skipped to make the figure simpler, and neither the OnGIS algorithms operates with the *top* node.

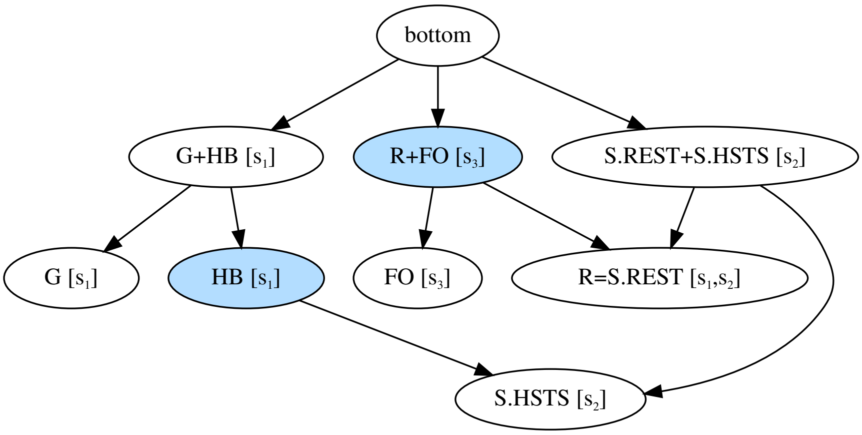


Figure 6.7: The lattice of the sources in the first example.

Now when a user asks the query in Listing 6.6, the prototype searches the lattice. Its algorithm cannot find a single source to answer the complete query, hence it splits the query, and comes up with two partial queries – one for s_{lgd} in Listing 6.7 and one for s_{ex} in Listing 6.8. The lattice nodes used for deciding which sources to use are darker in Fig. 6.7.

Listing 6.6: A user's query for the first example.

```
SELECT ?x ?c ?g WHERE {
  ?x a lgd:Restaurant.
  ?x a lgd:HistoricBuilding.
  ?x ex:hasFoodOrigin ?c.
  ?x geo:hasGeometry
       [geo:hasSerialization ?g].
}
```

Listing 6.7: First example result query for s_{lgd}.

```
SELECT ?x ?g WHERE {
  ?x a lgd:HistoricBuilding.
  ?x geo:hasGeometry
       [geo:hasSerialization ?g].
}
```

Listing 6.8: First example result query for s_{ex}.

```
SELECT ?x ?c ?g WHERE {
  ?x a lgd:Restaurant.
  ?x ex:hasFoodOrigin ?c.
  ?x geo:hasGeometry
       [geo:hasSerialization ?g].
}
```

Some statistics of the example using the OnGIS prototype are given in Table 6.1. The response times are computed from 20 lattice constructions and 200 lattice searches on a Linux laptop with Intel Core i7 @ 2.4 GHz with Oracle Java 8 (1.8.0_101) without any parallelization. As the reasoner used for deciding consistency is Pellet, which is designed for more expressive logic than OWL 2 QL, there is a space for optimization by developing a reasoner tailored for OWL 2 QL consistency checks.

Table 6.1: The first example statistics.

	Value/Average	Std. dev.
Background ontology	16,251 axioms	
Prototypical queries	9	
Lattice size	8	
Lattice construction		
Containment decisions	50	
Time	5.01 s	0.62 s
Lattice search		
Containment decisions	20	
Time	214 ms	30 ms

Notice that both numbers of containment decisions are lower than the theoretical maximums, $9(9-1) = 72$ for lattice construction, and $8+8\cdot 5 = 48$ for lattice searching (the splitting has been used).

6.2.2 The Second Example: Colleges in Passive Houses

In the second example, the background ontology consists of:

- LinkedGeoData ontology,
- GeoNames ontology,
- A simple example ontology, defining the class `ex:PassiveHouse`, being a subclass of building classes in both GeoNames and LinkedGeoData ontologies, and the class `ex:EngineeringCollege`, being a subclass of college classes in both GeoNames and LinkedGeoData ontologies.

There are three sets of prototypical queries, representing three sources, having 68 queries in total focusing mainly on amenities:

- s_{lgd}, a source capable of queries using LinkedGeoData vocabulary,
- s_{gn}, a source capable of queries using GeoNames vocabulary, and
- s_{ex}, a source having a single prototypical query consisting of `ex:PassiveHouse`$(?x)$, `ex:EngineeringCollege`$(?x)$.

The constructed lattice contains 61 nodes. Notice this is less than 68 input queries, since GeoNames ontology, which is linked to LinkedGeoData, is defining some of its classes equivalent to the LinkedGeoData ones. Therefore, OnGIS unifies the queries using those classes; some other classes have subclass relationship, so the lattice is non-trivial.

The testing user query in Listing 6.9 is asking for both a `lgd:College` and a `ex:PassiveHouse`. OnGIS correctly responds with the source s_{ex}, as `ex:PassiveHouse`$(?x)$, `ex:EngineeringCollege`$(?x)$ is the most general query contained within the user query.

Listing 6.9: User's query for the second example.

```
SELECT ?x ?g WHERE {
  ?x a lgd:College.
  ?x a ex:PassiveHouse.
  ?x geo:hasGeometry
       [geo:hasSerialization ?g].
}
```

The statistics of the second example are given in Table 6.2.

Table 6.2: The second example statistics.

	Value/Average	Std. dev.
Background ontology	16,288 axioms	
Prototypical queries	68	
Lattice size	61	
Lattice construction		
Containment decisions	3,753	
Time	36.7 s	2.4 s
Lattice search		
Containment decisions	57	
Time	746 ms	87 ms

Both numbers of containment decisions are lower than the theoretical maximums, $68(68-1) = 4,556$ for lattice construction, and 61 for lattice searching (the splitting has not been used).

6.2.3 The Third Example: Restaurants in the Czech Republic

The third example, also successfully tested in OnGIS, uses LinkedGeoData, GeoNames, and DBpedia ontologies as the background ontology. It has three sources, each one using one of the three ontologies, altogether containing more prototypical queries than in the previous example. Moreover, the prototypical queries contain spatial restrictions, representing that the sources contain data only in specified areas. For that purpose, the `geo:ehInside` topological relation is used, together with a polygon serialized by a WKT string. Thus, a prototypical query can look like the one in Listing 6.10.

Listing 6.10: Example of prototypical query in the third example.

```
SELECT ?x ?g WHERE {
  ?x a dbp:Restaurant.
  ?x geo:hasGeometry
      [geo:hasSerialization ?g].
  ?x geo:ehInside
      [a sf:Polygon; geo:asWKT
      "POLYGON((-31.3␣81,39.9␣81...))"
      ^^geo:wktLiteral].
}
```

The three sources are:

- s_{lgd}, having data described by LinkedGeoData. The source contains 43 prototypical queries, which are all spatially restricted by a rectangle bounding the area of Prague (the capital of the Czech Republic),
- s_{gn}, having data described by GeoNames ontology. The source contains 34 prototypical queries, which are all spatially restricted by a rectangle bounding the area of the Czech Republic,
- s_{dbp}, having data described by DBpedia ontology. The source contains 3 prototypical queries, which are all spatially restricted by a rectangle bounding the area of Europe.

Each of these sources contains a class for restaurants, and these classes are linked together as being equal. So when a user asks the query in Listing 6.11 (restaurants and their geometries within a rectangular area slightly larger than the Czech Republic), OnGIS correctly answers with the source s_{gn}, which is the most fitting one. Even though the other two sources contain equal classes for restaurants, the corresponding prototypical query in s_{lgd} is narrower (is strictly contained within) than the selected prototypical query in s_{gn}, and the corresponding prototypical query in s_{dbp} is wider than the user's query (does not satisfy to be contained within the user's query).

Also note that the example user's query uses the topological relation `geo:sfWithin`, while the sources for spatially restricting their data use `geo:ehInside`. Since `geo:ehInside` is a sub-property of `geo:sfWithin` (see Fig. 4.2), the spatial restriction given by the user is weaker than the spatial restrictions given by the sources, therefore the query can be answered by s_{gn}. When the user's query would use a stricter spatial restriction, e.g. `geo:rcc8ntpp`, none of the sources could satisfy the query.

The resulting statistics of the third example are in Table 6.3. Again, both numbers of containment decisions are lower than the theoretical maximums, $80(80-1) = 6,320$ for lattice construction, and 80 for lattice searching (the splitting has not been used).

Listing 6.11: User's query for the third example.

```
SELECT ?x ?g WHERE {
  ?x a lgd:Restaurant.
  ?x geo:hasGeometry
      [geo:hasSerialization ?g].
  ?x geo:sfWithin
      [a sf:Polygon; geo:asWKT
      "POLYGON((11␣52,20␣52,20...))"
      ^^geo:wktLiteral].
}
```

Table 6.3: The third example statistics.

	Value/Average	Std. dev.
Background ontology	31,331 axioms	
Prototypical queries	80	
Lattice size	80	
Lattice construction		
Containment decisions	5,568	
Time	239.9 s	10.2 s
Lattice search		
Containment decisions	69	
Time	3.66 s	0.23 s

6.2.4 Comparison with Other Systems

I tried to compare my proposed method of query containment with other implementations found available. As stated in Section 3.5, using the reasoners FaCT and FaCT++ [88, 87] was not technically possible.

I succeeded using the query containment method used in Pellet (Pellet is also used as the reasoner for consistency checks in my prototype), however with some limitations. A look into the Java class responsible for query containment in Pellet, `QuerySubsumption`[1], reveals it is partially similar to a part of the solution presented in Section 4.4.2, based on [4, 5]. For deciding, whether $\mathcal{O} \models q_1 \sqsubseteq q_2$, it computes $T = \mathrm{Can}(q_1) \cup \mathcal{O}$, and tries to answer q_2 over T. Then $q_1 \sqsubseteq q_2$ iff the result is not empty. The translation seems simpler compared to my approach, but it is not clear how easy it would be to extend it with spatial reasoning, and it uses conjunctive query answering (which is NP-complete in the case of OWL 2 QL), instead of consistency checks (NLogSpace-complete in OWL 2 QL). As conjunctive query answering has higher complexity, it is expected that the query containment based on Pellet would be slower even without spatial reasoning. And the time results in Table 6.4 confirm that.

[1] Available at `https://github.com/Complexible/pellet/blob/master/query/src/main/java/com/clarkparsia/pellet/sparqldl/engine/QuerySubsumption.java`, cit. 2016-08-07.

As it does not support data properties, I tested it only with a simplified version of my first example, where I skipped the `geo:hasGeometry` part for retrieving geometries both in prototypical queries and in the user's query (which would be needed for an actual source querying, but it may be skipped in the example).

I replaced my query containment algorithm with the one Pellet provides and kept the rest of the OnGIS prototype the same (lattice construction and searching). Then the first example gave the same results; also the lattice has not changed. The result times comparing the simplified first example using my versus Pellet query containment are in Table 6.4.

Table 6.4: The simplified first example response time statistics with my and Pellet query containment.

	Average	Std. dev.
Lattice construction		
My query containment	713 ms	442 ms
Pellet query containment	1017 ms	659 ms
Lattice search		
My query containment	136 ms	23 ms
Pellet query containment	146 ms	24 ms

Unfortunately, other query containment decision systems mentioned in Section 3.5 have not been found or have not been accessible.

Chapter 7

Conclusion

This thesis proposes OnGIS, a system for easy access to integrated GIS data from different heterogeneous sources, that even a non-expert user (without any GIS or domain specific knowledge) can use. A prototypical implementation has been presented, and the whole concept was proved feasible on a few test cases.

The proposed design consists of several parts:

- Two methods of user-friendly query inputs are presented in Section 4.3:
    - a list-based method, where query parts, represented as boxes, are iteratively added to a list, where can be linked together or spatially restricted, and
    - a structured expression, resembling mathematical expression, which forms a query tree by using parentheses.
- Mapping annotations, which annotate entities of a domain ontology with the means where and how to obtain the data (i.e. instances and literals), are described in Section 4.2. Mapping annotations to relational databases, SPARQL endpoints, and ArcGIS web services and REST services are presented.
- A method for finding relevant sources for a user's query is described in Section 4.4 and Section 4.5. Its first step is to describe each data source with a set of prototypical queries. Then a lattice can be constructed out of the prototypical queries using the proposed method of geospatial query containment. When a user asks a query, the lattice is searched using the query containment method, and the source capable of answering the query is returned. When no such source is found, the user's query is split into pieces, and the sources for the pieces are given.

All the designed components are implemented and successfully tested in Chapter 6. Also, a comparison to an existing query containment method using a simplified test case (as the compared system does not support everything needed to support spatial queries) is there, suggesting my solution is faster, besides being easily extensible for the geospatial purposes.

The query containment method uses queries expressed in GeoSPARQL language with OWL 2 QL semantics. It is based on [4] and [5], where my contribution is adapting the authors' methods to OWL 2 QL and extending it with geospatial reasoning.

My another contribution is extending an existing *DL−Lite* reasoner, Owlgres [57], with completely new functions: support for having TBox in memory, while translating ABox queries into SQL (originally, all axioms were in a database), support for spatial GeoSPARQL queries, translated into PostGIS functions, and support for custom mappings from classes and relations to database table (based on one of database schemas presented in [103]).

The plans for the future are completing the prototypes to a production-ready application, adding the geospatial processing for completing an answer when results from multiple sources are obtained for a split query, and finding more interesting open sources and connecting them to OnGIS. Also, the implementation of the lattice construction and searching algorithms can be optimized, e.g. by parallelization.

Bibliography

[1] Wikipedia. *Region connection calculus*. 2016, accessed on 2016-08-24. Available from: `http://en.wikipedia.org/wiki/Region_connection_calculus`

[2] Artale, A.; Calvanese, D.; Kontchakov, R.; et al. The DL-Lite Family and Relations. *J. of Artificial Intelligence Research*, volume 36, 2009: pp. 1–69.

[3] European Commission. *INSPIRE, Infrastructure for Spatial Information in the European Community*. 2016, accessed on 2016-08-26. Available from: `http://inspire.ec.europa.eu/`

[4] Horrocks, I.; Tessaris, S.; Sattler, U.; et al. Query containment using a DLR ABox. In *Ltcs-report LTCS-99-15, LuFG Theoretical Computer Science, RWTH*, 1999.

[5] Horrocks, I.; Sattler, U.; Tessaris, S.; et al. How to Decide Query Containment under Constraints Using a Description Logic. In *Logic for Programming and Automated Reasoning*, *Lecture Notes in Artificial Intelligence*, volume 1955, edited by M. Parigot; A. Voronkov, Springer Berlin Heidelberg, 2000, ISBN 978-3-540-41285-4, pp. 326–343, doi: 10.1007/3-540-44404-1_21. Available from: `http://dx.doi.org/10.1007/3-540-44404-1_21`

[6] Clementini, E.; Felice, P.; Oosterom, P. A small set of formal topological relationships suitable for end-user interaction. In *Advances in Spatial Databases*, *Lecture Notes in Computer Science*, volume 692, edited by D. Abel; B. Chin Ooi, Springer Berlin Heidelberg, 1993, ISBN 978-3-540-56869-8, pp. 277–295, doi:10.1007/3-540-56869-7_16. Available from: `http://dx.doi.org/10.1007/3-540-56869-7_16`

[7] The PostGIS Development Group. *PostGIS 2.2.1dev Manual*. 2016, accessed on 2016-06-26. Available from: `http://postgis.net/documentation/`

[8] Butler, H.; Daly, M.; Doyle, A.; et al. *The GeoJSON Format Specification*. 2008, accessed on 2016-06-19. Available from: `http://geojson.org/geojson-spec.html`

[9] Open Geospatial Consortium. *OpenGIS Geography Markup Language (GML) Encoding Standard*. 2007. Available from: `http://www.opengeospatial.org/standards/gml`

[10] Open Geospatial Consortium. *OGC WCS 2.0 Interface Standard – Core*. 2012, accessed on 2016-06-21. Available from: `http://www.opengeospatial.org/standards/wcs`

[11] Open Geospatial Consortium. *OpenGIS Web Feature Service 2.0 Interface Standard*. 2010, accessed on 2016-06-21. Available from: `http://www.opengeospatial.org/standards/wfs`

[12] The PostgreSQL Global Development Group. *PostgreSQL 9.5.2 Documentation*. 2016, accessed on 2016-06-26. Available from: `https://www.postgresql.org/docs/manuals/`

[13] Kubíček, P. *GIS ve veřejné správě II.* Institute of Geography, Masaryk University, Czech Republic, 2013, accessed on 2016-06-28. Available from: `https://is.muni.cz/el/1431/podzim2013/Z7262/um/GIS_VS_02_FIN.pdf` [in Czech]

[14] Esri. *ArcGIS for Server – Publish Services*. 2016, accessed on 2016-08-04. Available from: `http://server.arcgis.com/en/server/10.4/publish-services/windows/`

[15] OpenStreetMap Foundation. *OpenStreetMap Wiki*. 2011, accessed on 2011-08-09. Available from: `http://wiki.openstreetmap.org/wiki/Main_Page`

[16] Wikipedia. Freedom of Information Act (United States) — Wikipedia, The Free Encyclopedia. 2016, [Online; accessed 2016-06-26]. Available from: `https://en.wikipedia.org/w/index.php?title=Freedom_of_Information_Act_(United_States)&oldid=724597042`

[17] Tauberer, J. *Open Government Data: The Book*. Second edition, 2014. Available from: `https://opengovdata.io/`

[18] Poláček, J. *Otevřená data a služby v resortu ČÚZK*. Český úřad zeměměřický a katastrální, 2014, accessed on 2016-06-26. Available from: `http://www.apkg.upol.cz/wp-content/uploads/2014/09/6_SemPra_B2.pdf` [in Czech]

[19] Open Geospatial Consortium. *OGC Catalogue Services 3.0 – General Model*. 2016, accessed on 2016-08-04. Available from: `http://www.opengeospatial.org/standards/cat`

[20] McLeod, D.; Heimbigner, D. A Federated Architecture for Database Systems. In *Proceedings of the May 19-22, 1980, National Computer Conference*, AFIPS '80, New York, NY, USA: ACM, 1980, pp. 283–289, doi:10.1145/1500518.1500561. Available from: `http://doi.acm.org/10.1145/1500518.1500561`

[21] Lenzerini, M. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, New York, NY, USA: ACM, 2002, ISBN 1-58113-507-6, pp. 233–246, doi:10.1145/543613.543644. Available from: `http://doi.acm.org/10.1145/543613.543644`

[22] Ziegler, P.; Dittrich, K. R. Three Decades of Data Integration - All Problems Solved? In *In 18th IFIP World Computer Congress (WCC 2004), Volume 12, Building the Information Society*, 2004, pp. 3–12.

[23] W3C. *Linked Data*. 2015, accessed on 2016-08-04. Available from: `https://www.w3.org/standards/semanticweb/data`

[24] Berners-Lee, T. *Linked Data*. 2006, accessed on 2016-07-02. Available from: `https://www.w3.org/DesignIssues/LinkedData.html`

[25] Heath, T.; Bizer, C. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, first edition, 2011, accessed on 2016-07-02. Available from: `http://linkeddatabook.com/`

[26] Vláda České republiky. *Akční plán České republiky Partnerství pro otevřené vládnutí na období let 2014 až 2016*. 2014, accessed on 2016-07-02. Available from: `https://apps.odok.cz/attachment/-/down/VPRA9R9BDNJD` [in Czech]

[27] Mráček, J.; et al. *Jak otevírat data?* Fond Otakara Motejla, 2014, accessed on 2016-07-02. Available from: `http://www.otevrenadata.cz/res/data/001/003498.pdf` [in Czech]

[28] W3C. *Resource Description Framework (RDF)*. 2014, accessed on 2016-06-21. Available from: `http://www.w3.org/RDF/`

[29] W3C. *RDF Schema 1.1*. 2014, accessed on 2016-06-23. Available from: `http://www.w3.org/TR/rdf-schema/`

[30] Beckett, D. *RDF 1.1 N-Triples*. 2014, accessed on 2016-07-04. Available from: `https://www.w3.org/TR/n-triples/`

[31] Beckett, D.; Berners-Lee, T.; Prud'hommeaux, E.; et al. *RDF 1.1 Turtle*. 2014, accessed on 2016-07-04. Available from: `https://www.w3.org/TR/turtle/`

[32] Berners-Lee, T.; Connolly, D. *Notation3 (N3): A readable RDF syntax*. 2011, accessed on 2016-07-04. Available from: `https://www.w3.org/TeamSubmission/n3/`

[33] Sporny, M.; Longley, D.; Kellogg, G.; et al. *JSON-LD 1.0*. 2014, accessed on 2016-07-04. Available from: `https://www.w3.org/TR/json-ld/`

[34] W3C. *RDF 1.1 XML Syntax*. 2014, accessed on 2016-07-04. Available from: `https://www.w3.org/TR/rdf-syntax-grammar/`

[35] Stadler, C.; Lehmann, J.; Höffner, K.; et al. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal*, volume 3, no. 4, 2012: pp. 333–354. Available from: `http://jens-lehmann.org/files/2012/linkedgeodata2.pdf`

[36] Stadler, C.; Lehmann, J.; Auer, S. *LinkedGeoData Ontology*. University of Leipzig, accessed on 2011-04-06. Available from: `http://linkedgeodata.org/ontology/`

[37] Vatant, B. *The GeoNames Ontology*. GeoNames, 2012, accessed on 2016-02-27. Available from: `http://www.geonames.org/ontology/documentation.html`

[38] W3C Semantic Web Interest Group; Brickley, D. *Basic Geo (WGS84 lat/long) Vocabulary*. 2004, accessed on 2016-07-04. Available from: `https://www.w3.org/2003/01/geo/`

[39] Auer, S.; Bizer, C.; Kobilarov, G.; et al. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN 978-3-540-76298-0, pp. 722–735, doi:10.1007/978-3-540-76298-0_52. Available from: `http://dx.doi.org/10.1007/978-3-540-76298-0_52`

[40] Baader, F.; Calvanese, D.; McGuinness, D. L.; et al. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003, ISBN 0-521-78176-0.

[41] Zolin, E. *Complexity of reasoning in Description Logics*. 2013, accessed on 2016-07-04. Available from: `http://www.cs.man.ac.uk/~ezolin/dl/`

[42] World Wide Web Consortium. *OWL 2 Web Ontology Language, Document Overview*. Second edition, 2012. Available from: `https://www.w3.org/TR/owl2-overview/`

[43] Calvanese, D.; De Giacomo, G.; Lembo, D.; et al. DL-Lite: Tractable Description Logics for Ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005, pp. 602–607.

[44] Calvanese, D.; De Giacomo, G.; Lembo, D.; et al. Linking Data to Ontologies: The Description Logic DL-LiteA. In *Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006)*, *CEUR Electronic Workshop Proceedings*, volume 216, 2006. Available from: `http://ceur-ws.org/`

[45] Calvanese, D.; De Giacomo, G.; Lembo, D.; et al. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. of Automated Reasoning*, volume 39, no. 3, 2007: pp. 385–429.

[46] World Wide Web Consortium. *OWL 2 Web Ontology Language, Profiles, OWL 2 QL*. Second edition, 2012. Available from: `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

[47] W3C. *SPARQL 1.1 Overview*. 2013, accessed on 2016-07-06. Available from: `https://www.w3.org/TR/sparql11-overview/`

[48] W3C. *SPARQL Query Language for RDF*. 2008, accessed on 2016-06-21. Available from: `https://www.w3.org/TR/rdf-sparql-query/`

[49] Open Geospatial Consortium. *OGC GeoSPARQL — A Geographic Query Language for RDF Data*. 2012. Available from: `http://www.opengeospatial.org/standards/geosparql`

[50] Lopez, X. *GeoSPARQL — A geographic query language for RDF data, A proposal for an OGC Draft Candidate Standard*. Oracle, 2010. Available from: `http://www.ogcnetwork.net/system/files/Spatial_SPARQL_Lopez.pdf`

[51] W3C. *RIF Overview (Second Edition)*. 2013, accessed on 2016-06-21. Available from: `http://www.w3.org/TR/rif-overview/`

[52] Telang, A.; Chakravarthy, S.; Huang, Y. Information Integration Across Heterogeneous Sources: Where Do We Stand and How to Proceed? In *COMAD*, Computer Society of India / Allied Publishers, 2008, ISBN 978-81-8424-370-3, pp. 186–197.

[53] Levy, A. Y.; Rousset, M.-C. Combining Horn Rules and Description Logics in CARIN. *Artif. Intell.*, volume 104, no. 1-2, 1998: pp. 165–209.

[54] Donini, F. M.; Lenzerini, M.; Nardi, D.; et al. A Hybrid System with Datalog and Concept Languages. In *In Trends in AI, volume LNAI 549*, Springer Verlag, 1991, pp. 88–97.

[55] Levy, A. Y.; Rajaraman, A.; Ordille, J. J. Query-Answering Algorithms for Information Agents. In *AAAI-96*, 1996.

[56] Bouquet, P.; Giunchiglia, F.; van Harmelen, F.; et al. C-OWL: Contextualizing Ontologies. In *ISWC*, Lecture Notes in Computer Science, Springer Verlag, October 2003, pp. 164–179.

[57] Stocker, M.; Smith, M. Owlgres: A Scalable OWL Reasoner. In *OWLED, CEUR Workshop Proceedings*, volume 432, CEUR-WS.org, 2008.

[58] Acciarri, A.; Calvanese, D.; Giacomo, G. D.; et al. QuOnto: Querying ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.

[59] Motik, B.; Patel-Schneider, P. F.; Horrocks, I. *OWL 1.1 Web Ontology Language: Structural Specification and Functional-Style Syntax*. 2008. Available from: `http://www.w3.org/TR/2008/WD-owl11-syntax-20080108/`

[60] Corona, C.; Ruzzi, M.; Savo, D. F. Filling the gap between OWL 2 QL and QuOnto: ROWLKit. In *Description Logics '09*, 2009.

[61] Calvanese, D.; Giacomo, G. D.; Lembo, D.; et al. Mastro-i: Efficient Integration of Relational Data through DL Ontologies. In *Proc. of the 20th Int. Workshop on Description Logics (DL 2007)*, 2007, pp. 227–234.

[62] Calvanese, D.; Giacomo, G. D.; Lembo, D.; et al. The Mastro System for Ontology-based Data Access. *Semantic Web Journal*, volume 2, no. 1, 2011: pp. 43–53.

[63] Henss, J.; Kleb, J.; Grimm, S.; et al. A Database Backend for OWL. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529, edited by R. Hoeksta; P. F. Patel-Schneider, 2009.

[64] Battle, R.; Kolas, D. Enabling the geospatial Semantic Web with Parliament and GeoSPARQL. *Semantic Web Journal*, volume 3, no. 4, Oct. 2012: pp. 355–370, ISSN 1570-0844, doi:10.3233/SW-2012-0065.

[65] Bizer, C.; Seaborne, A. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC2004 (posters)*, November 2004. Available from: `http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/Bizer-D2RQ-ISWC2004-Poster.pdf`

[66] Cyganiak, R.; Bizer, C.; Garbers, J.; et al. *The D2RQ Mapping Language*. 2012, accessed on 2016-07-09. Available from: `http://d2rq.org/d2rq-language`

[67] W3C. *R2RML: RDB to RDF Mapping Language*. 2012, accessed on 2016-07-09. Available from: `https://www.w3.org/TR/r2rml/`

[68] Michel, F.; Djimenou, L.; Faron-Zucker, C.; et al. xR2RML: Relational and Non-Relational Databases to RDF Mapping Language. Research Report ISRN I3S/RR 2014-04-FR, CNRS, Jan. 2015. Available from: `https://hal.archives-ouvertes.fr/hal-01066663`

[69] Wessel, M.; Möller, R. Flexible Software Architectures for Ontology-Based Information Systems. *Journal of Applied Logic – Special Issue on Empirically Successful Computerized Reasoning*, volume 7, no. 1, 2009: pp. 75 – 99, ISSN 1570-8683, doi:http://dx.doi.org/10.1016/j.jal.2007.07.006. Available from: `http://www.sciencedirect.com/science/article/pii/S157086830700064X`

[70] Haarslev, V.; Moeller, R.; Wessel, M. *Racer*. 2016, accessed on 2016-06-21. Available from: `https://www.ifis.uni-luebeck.de/index.php?id=385`

[71] Zhao, T.; Zhang, C.; Wei, M.; et al. Ontology-Based Geospatial Data Query and Integration. In *GIScience*, *Lecture Notes in Computer Science*, volume 5266, Springer, 2008, ISBN 978-3-540-87472-0, pp. 370–392.

[72] Zhang, C.; Zhao, T.; Li, W. The framework of a geospatial semantic web-based spatial decision support system for Digital Earth. *Int. J. Digital Earth*, volume 3, no. 2, 2010: pp. 111–134.

[73] Lutz, M.; Kolas, D. Rule-Based Discovery in Spatial Data Infrastructure. *Transactions in GIS*, volume 11, no. 3, 2007: pp. 317–336, ISSN 1467-9671, doi:10.1111/j.1467-9671.2007.01048.x. Available from: `http://dx.doi.org/10.1111/j.1467-9671.2007.01048.x`

[74] Visser, U.; Stuckenschmidt, H.; Schlieder, C. Interoperability in GIS-enabling technologies. In *Proceedings of the 5th AGILE Conference on Geographic Information Science*, Citeseer, 2002, p. 291.

[75] Visser, U. *Intelligent information integration for the Semantic Web*, volume 3159. Springer, 2005, ISBN 3540229930.

[76] Fensel, D.; van Harmelen, F.; Horrocks, I.; et al. OIL: an ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems*, volume 16, no. 2, Mar 2001: pp. 38–45, ISSN 1541-1672, doi:10.1109/5254.920598.

[77] Zhang, Y.; Chiang, Y.-Y.; Szekely, P.; et al. A semantic approach to retrieving, linking, and integrating heterogeneous geospatial data. In *Joint Proceedings of the Workshop on AI Problems and Approaches for Intelligent Environments and Workshop on Semantic Cities*, ACM, 2013, pp. 31–37.

[78] Bogdanović, M.; Stanimirović, A.; Stoimenov, L. Methodology for geospatial data source discovery in ontology-driven geo-information integration architectures. *Web Semantics: Science, Services and Agents on the World Wide Web*, volume 32, 2015: pp. 1–15.

[79] Katz, Y.; Grau, B. C. Representing Qualitative Spatial Information in OWL-DL. In *Proceedings of OWL: Experiences and Directions*, 2005.

[80] Baglioni, M.; Masserotti, M. V.; Renso, C.; et al. Improving Geodatabase Semantic Querying Exploiting Ontologies. In *GeoSpatial Semantics*, Springer, 2011.

[81] Codescu, M.; Horsinka, G.; Kutz, O.; et al. DO-ROAM: Activity-Oriented Search and Navigation with OpenStreetMap. In *GeoSpatial Semantics*, Springer, 2011.

[82] Fonseca, F. T.; Egenhofer, M. J.; Agouris, P.; et al. Using Ontologies for Integrated Geographic Information Systems. *Transactions in GIS*, volume 6, no. 3, 2002: pp. 231–257, ISSN 1467-9671, doi:10.1111/1467-9671.00109.

[83] Heydari, N.; Mansourian, A.; Taleai, M.; et al. Ontology-based GIS web service for increasing semantic interoperability among organizations involving drilling in city of Tehran. In *GSDI 11 World Conference*, 2009.

[84] Stoimenov, L.; Stanimirovic, A.; Dordevic-Kajan, S. Semantic Interoperability Using Multiple Ontologies. In *Proceedings of 8th AGILE Conference on GIScience*, 2005.

[85] Stoimenov, L.; Djordjevic-Kajan, S.; Stojanovic, D. Integration of GIS data sources over the Internet using mediator and wrapper technology. In *Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean*, volume 1, 2000, pp. 334 –336 vol.1, doi:10.1109/MELCON.2000.880434.

[86] Buccella, A.; Cechich, A.; Fillottrani, P. Ontology-driven geographic information integration: A survey of current approaches. *Computers & Geosciences*, volume 35, no. 4, 2009: pp. 710 – 723, ISSN 0098-3004, doi:http://dx.doi.org/10.1016/j.cageo.2008.02.033.

[87] Horrocks, I. *The FaCT System*. 2003, accessed on 2016-07-06. Available from: `http://www.cs.man.ac.uk/~horrocks/FaCT/`

[88] Tsarkov, D.; Horrocks, I. *FaCT++*. 2007, accessed on 2016-06-21. Available from: `http://owl.man.ac.uk/factplusplus/`

[89] Sirin, E.; Parsia, B. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED*, *CEUR Workshop Proceedings*, volume 258, edited by C. Golbreich; A. Kalyanpur; B. Parsia, CEUR-WS.org, 2007.

[90] Rodrıguez-Muro, M.; Calvanese, D. Quest, a system for ontology based data access. In *OWL: Experiences and Directions Workshop (OWLED), Heraklion*, 2012.

[91] Rodríguez-Muro, M. *Quest Performance for 1.7*. 2012, accessed on 2016-05-15. Available from: `http://ontop-obda.blogspot.cz/2012_01_01_archive.html`

[92] Chekol, M. W. On the Containment of SPARQL Queries under Entailment Regimes. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 936–942. Available from: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12027`

[93] World Wide Web Consortium. *OWL 2 Web Ontology Language, Profiles, OWL 2 EL*. Second edition, 2012. Available from: `https://www.w3.org/TR/owl2-profiles/#OWL_2_EL`

[94] Bradfield, J.; Stirling, C. *Handbook of modal logic*, volume 3, chapter Modal Mu-Calculi. Elsevier, Nov 2006, ISBN 9780444516909, pp. 721–756.

[95] Pichler, R.; Skritek, S. Containment and Equivalence of Well-designed SPARQL. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-2375-8, pp. 39–50, doi: 10.1145/2594538.2594542. Available from: http://doi.acm.org/10.1145/2594538.2594542

[96] Chekol, M. W.; Euzenat, J.; Genevès, P.; et al. Evaluating and benchmarking SPARQL query containment solvers. In *Proc. 12th International semantic web conference (ISWC)*, volume 8219, Sydney, Australia: Springer Verlag, Oct. 2013, pp. 408–423, doi:10.1007/978-3-642-41338-4_26. Available from: https://hal.inria.fr/hal-00917911

[97] Chekol, M. W.; Euzenat, J.; Genevès, P.; et al. SPARQL Query Containment Under SHI Axioms. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, AAAI Press, 2012, pp. 10–16. Available from: http://dl.acm.org/citation.cfm?id=2900728.2900730

[98] Chekol, M. W.; Euzenat, J.; Genevès, P.; et al. SPARQL Query Containment under RDFS Entailment Regime. In *Proceedings of Automated Reasoning: 6th International Joint Conference*, edited by B. Gramlich; D. Miller; U. Sattler, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-31365-3, pp. 134–148, doi: 10.1007/978-3-642-31365-3_13. Available from: http://dx.doi.org/10.1007/978-3-642-31365-3_13

[99] Groppe, S.; Heinrich, D.; Werner, S. Distributed Join Approaches for W3C-Conform SPARQL Endpoints. *Open Journal of Semantic Web (OJSW)*, volume 2, no. 1, 2015: pp. 30–52, ISSN 2199-336X. Available from: http://www.ronpub.com/publications/OJSW_2015v2i1n04_Groppe.pdf

[100] Koubarakis, M.; Kyzirakos, K.; Karpathiotakis, M.; et al. *Introduction in stRDF and stSPARQL*. 2012, accessed on 2016-07-27. Available from: http://www.strabon.di.uoa.gr/files/stSPARQL_tutorial.pdf

[101] Kyzirakos, K.; Karpathiotakis, M.; Koubarakis, M. *Strabon: A Semantic Geospatial DBMS*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-35176-1, pp. 295–311, doi:10.1007/978-3-642-35176-1_19. Available from: http://dx.doi.org/10.1007/978-3-642-35176-1_19

[102] Wikipedia. *Lattice (order)*. 2016, accessed on 2016-08-23. Available from: https://en.wikipedia.org/wiki/Lattice_(order)

[103] Šmíd, M. *Using Databases for Description Logics*. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2009.

[104] Graphviz – Graph Visualization Software. *The DOT Language*. 2016, accessed on 2016-08-25. Available from: `http://www.graphviz.org/content/dot-language`

Abbreviations

GIS

Symbol	Meaning
CRS	Coordinate reference system, defines a map projection – how coordinates are mapped from one system to another.
DE-9IM	Dimensionally Extended Nine-Intersection Model, a topological model describing spatial relations by 3×3 matrices.
GeoJSON	JSON-based geometry serialization format.
GeoTIFF	TIFF (a raster image format) extended with geographical metadata, e.g. CRS, bounding box.
GIS	Geographical Information System.
GML	Geography Markup Language, an XML based serialization format for vector geometries.
OGC	Open Geospatial Consortium, `http://www.opengeospatial.org/`, which sets many GIS standards.
RCC8	Region Connection Calculus 8, a family of topological relations.
WCS	Web Coverage Service, an OGC protocol for providing coverage data.
WFS	Web Feature Service, an OGC protocol for providing and manipulating source vector data of a feature.
WKT, WKB	Well-Known Text, Well-Known Binary a text, respectively binary serialization format for vector geometries.
WMS	Web Map Service, an OGC protocol for serving raster maps.

Semantic Web

Symbol	Meaning
DL−Lite	A tractable description logic.
GeoSPARQL	A SPARQL extension with spatial support.
OWL	Web Ontology Language, defined by W3C.
OWL 2 QL	OWL profile based on *DL−Lite*.
RDF	Resource Description Framework, a data model used for Semantic Web.
SPARQL	SPARQL Protocol and RDF Query Language, a query language for RDF.
URI	Uniform Resource Identifier, a superset of URL and URN.
W3C	World Wide Web Consortium, `http://www.w3.org/`, which sets many Internet standards.

Other

Symbol	Meaning
IPR Praha	Prague Institute of Planning and Development.
RDBMS	Relation Database Management System.

Publications of the Author Relevant to the Thesis

Journal Publications

[1] **Šmíd, M. (90% contribution)**; Křemen, P. OnGIS: Semantic Query Broker for Heterogeneous Geospatial Data Sources. *Open Journal of Semantic Web (OJSW)*, volume 3, no. 1, 2016. pp. 32–50. ISSN 2199-336X. Available from `http://www.ronpub.com/publications/OJSW_2016v3i1n03_Smid.pdf`.

[2] **In review: Šmíd, M.** OnGIS: Geospatial Data Integration Using Semantic Technologies and Query Containment. *International Journal on Semantic Web and Information Systems (IJSWIS)*. ISSN: 1552-6283. In review process since April 2015. A revised version has been sent after the first review cycle.

Publications in Conference Proceedings

[1] **Šmíd, M. (90% contribution)**; Kouba, Z. OnGIS: Ontology Driven Geospatial Search and Integration. In *Terra Cognita Workshop on Foundations, Technologies and Applications of the Geospatial Web*, CEUR Workshop Proceedings, Tilburg, 2012. pp. 27–38. ISSN 1613-0073. Available from `http://ceur-ws.org/Vol-901/paper3.pdf`.

The paper has been cited in:

- Boonprapasri, T.; Sriharee, G. An applied ontology: A semantic query builder for health GIS system. In *2015 International Computer Science and Engineering Conference (ICSEC)*, IEEE, 2015. pp. 1–6.

[2] **Šmíd, M. (90% contribution)**; Kouba, Z. OnGIS: Metody vyhledávání v geografických datech řízené ontologiemi. In *Digitální technologie v geoinformatice, kartografii a dálkovém průzkumu Země*, ČVUT, Fakulta stavební, Katedra mapování a kartografie, Praha,

2012. pp. 107–116. ISBN 978-80-01-05131-3. Available from http://gkinfo.fsv.cvut.cz/index.php/sbornik. [in Czech]

Other Publications

[1] **Šmíd, M.** *Proposal of an Ontology-Based GIS.* Ph.D. Thesis Proposal, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2011.

Remaining Publications of the Author

[1] **Šmíd, M. (50% contribution)**; Truman, S.; Mulholland, P.; Zdráhal, Z.; Crouch, S. G-Learn: An Exploratory Learning Environment for School Level Geography. In *Znalosti 2009 - sborník příspěvků*, Vydavatel'stvo STU, Bratislava, 2009. pp. 213–224. ISBN 978-80-227-3015-0.

[2] Bierhoff, I.; Goossen, B.; Wintjens, K.; Huijnen, C.; Křemen, P.; Kouba, Z.; Válek, F.; **Šmíd, M. (10% contribution)**; Blaško, M.; Panis, P. Participatory Design of Netcarity Services Using Different Perspectives. In *Proceedings of the first International AEGIS Conference*, AEGIS project, Seville, 2010. pp. 202–209. Available from `http://www.epr.eu/aegis/wp-content/uploads/2011/02/Conference_Proceedings.pdf`.

[3] Křemen, P.; **Šmíd, M. (40% contribution)**; Kouba, Z. OWLDiff: A Practical Tool for Comparison and Merge of OWL Ontologies. In *Twenty-Second International Workshop on Database and Expert System Applications*, IEEE Computer Society, Los Alamitos, 2011. pp. 229–233. ISBN 978-0-7695-4486-1. ISSN 1529-4188. Available from `http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=6059822`.

The paper has been cited in:

- Konev, B; Ludwig, M; Walther, D et al. The Logical Difference for the Lightweight Description Logic EL. *Journal of Artificial Intelligence Research*, volume 44, 2012. pp. 633–708. ISSN 1076-9757.
- Gonçalves, RS.; Parsia, B.; Sattler, U. Concept-based semantic difference in expressive description logics. In *International Semantic Web Conference*, Springer, 2012. pp. 99–115.
- Dinh, D.; Dos Reis, JC.; Pruski, C.; Da Silveira, M.; Reynaud-Delaître, C. Identifying change patterns of concept attributes in ontology evolution. In *European Semantic Web Conference*, Springer, 2014. pp. 768–783.

- Ochs, C.; Perl, Y.; Geller, J.; Haendel, M.; Brush, M.; Arabandi, S.; Tu, S. Summarizing and visualizing structural changes during the evolution of biomedical ontologies using a Diff Abstraction Network. *Journal of biomedical informatics*, volume 56, Elsevier, 2015. pp. 127–144.
- Erdur, RC.; Dikenelli, O.; Alatli, O.; Ekinci, EE.; Akar, Z. Integrating linked data space with agents using the environment abstraction. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, IEEE, 2012. pp. 625–630.
- and others (Google Scholar reports 22 citations in total).

[4] Křemen, P.; Mička, P.; Blaško, M.; **Šmíd, M. (25% contribution)** Ontology-Driven Mindmapping. In *Proceedings of the 8th International Conference on Semantic Systems*, ACM, New York, 2012. pp. 125–132. ISBN 978-1-4503-1112-0. Available from `http://dl.acm.org/citation.cfm?id=2362517`.

[5] Křemen, P.; Blaško, M.; **Šmíd, M. (15% contribution)**; Kouba, Z.; Ledvinka, M.; Kostov, B. MONDIS: Using Ontologies for Monument Damage Descriptions. In *Znalosti 2014*. VŠE, Praha, 2014. pp. 66–69. ISBN 978-80-245-2054-4. Available from `http://znalosti.eu/images/znalosti2014_proceedings.pdf`.