Semantification of Geospatial Information for Enriched Knowledge Representation in Context of Crisis Informatics

Claus Stadler[1], Simon Bin[1], Lorenz Bühmann[1], Norman Radtke[1], Kurt Junghanns[1], Sabine Gründer-Fahrer[1] and Michael Martin[1]

[1]*Institute of Applied Informatics (InfAI), Leipzig*

Abstract

In the context of crisis informatics, the integration and exploitation of high volumes of heterogeneous data from multiple sources is one of the big chances as well as challenges up to now. Semantic Web technologies have proven a valuable means to integrate and represent knowledge on the basis of domain concepts which improves interoperability and interpretability of information resources and allows deriving more knowledge via semantic relations and reasoning. In this paper, we investigate the potential of representing and processing geospatial information within the semantic paradigm. We show, on the technical level, how existing open source means can be used and supplemented as to efficiently handle geographic information and to convey exemplary results highly relevant in context of crisis management applications. When given semantic resources get enriched with geospatial information, new information can be retrieved combining the concepts of multi-polygons and geo-coordinates and using the power of GeoSPARQL queries. Custom SPARQL extension functions and data types for JSON, XML and CSV as well as for dialects such as GeoJSON and GML allow for succinct integration of heterogeneous data. We implemented these features for the Apache Jena Semantic Web framework by leveraging its plugin systems. Furthermore, significant improvements w.r.t. GeoSPARQL query performance have been contributed to the framework.

Keywords

GeoSPARQL, Geospatial Analysis, RDF Knowledge Graphs, Crisis Informatics

1. Introduction

With the increase in frequency and severity of crises and hazards worldwide, the development of means and methods to cope with the corresponding interruptions of economic, social, cultural and political life has become one of nowadays most pressing societal issues. Crisis informatics seeks to aid crisis management and resilience efforts by use of modern information and communication technologies. Based on the rapid rise of the Web, the amount of relevant data made available by governmental, volunteered and scientific initiatives is continuously increasing. However, the processes of integration and sharing faces many challenges, among them the intrinsic heterogeneity of information in terms of data sources, data types, content and concepts. It could be observed that, the information required to answer a certain factual

International Workshop on Data-driven Resilience Research 2022, July 6, 2022, Leipzig, Germany

cstadler@informatik.uni-leipzig.de (C. Stadler); sbin@informatik.uni-leipzig.de (S. Bin)

question or gain a better understanding of a situation is available, but dispersed among and siloed in a multiplicity of information sources.

Semantic Web technologies have proven a valuable means to integrate and represent knowledge on the basis of domain concepts which improves interoperability and interpretability of information resources and allows deriving implicit knowledge via semantic relations and reasoning. The aim of this paper is to investigate the potential of representing and processing geospatial information within the semantic paradigm. Due to the spatio-temporal nature of crises, geographic information is one key to effective monitoring, planning and decision-making in crisis management application scenarios. Adding a spatial dimension to the Web of Data / Semantic Web not only requires means for geospatial representation and interlinking of entities and resources but also for inferring implicit spatial relations from the objects' geometry together with the possibility for flexible searching and querying geographic information all together.

In this paper, we show, on the technical level, how existing open source means can be used and supplemented as to efficiently handle geographic information and to convey exemplary results highly relevant in context of crisis management applications, motivated from use cases within our current project Coypu. Within a concrete implementation of the current common standard, GeoSPARQL 1.0, we analyse certain shortcomings and offer improvements. Moreover, we implement parts of the future standard, GeoSPARQL 1.1 and add further non-standard functions to it.

The paper is structured as follows: In Section 2, we give an overview of related work. Section 3 details the SPARQL functions used and provided by this work. Then in Section 4, an applied example combining all these features is presented. Finally, Section 5 concludes this paper.

2. State of the Art and Related Work

As for Semantic Web frameworks, Apache Jena[1] (Java) forms the basis for our work. It has been first presented in [1]. One if its most important features is its GeoSPARQL 1.0 implementation[2], which was presented in [2].

OpenStreetMap[3] (OSM) is a large open and free data source for world-wide geographic data. However, in order to make it usable with Semantic Web tooling, its published data needs to be RDFised. LinkedGeoData [3], first started in 2009, is an effort to add a spatial dimension to the Web of Data / Semantic Web by means of SPARQL-to-SQL rewriting against OSM data stored in a PostGIS database[4]. There are other approaches for producing RDF from OSM data by means of extract-transform-load (ETL). Sophox[4] by Yuri Astrakhan is one project of that category. Another approach specifically for mapping OpenStreetMap data to RDF and a related specialised triple-store implementation has been described in [5], which performs very fast. However, actually answering queries requires pre-computed data. We have used their `osm2rdf` tool as part of the pipeline for Section 4.

[1] https://jena.apache.org
[2] https://jena.apache.org/documentation/geosparql/index.html
[3] https://www.openstreetmap.org
[4] https://sophox.org/, https://github.com/Sophox

A similar approach to integrating geospatial data as RDF and querying using GeoSPARQL on the example of hospitals outside of a flooded region has been presented in [6]. They appear to have used a custom GML converter. The inverse was attempted in [7], connecting a Web Feature Service to Linked Data, thus giving the Geographic information system access to the Web of Data.

Our RDF Processing Toolkit (RPT) features a command-line tool and Jena-based Java library for quickly setting up workflows that produce RDF from heterogeneous sources such as CSV, JSON, XML and RDF by means of SPARQL 1.1 syntax with extension functions. The effectiveness of RPT has been demonstrated for domains such as subject indexing and dataset such as in [8] and [9]. In this work, we further extend the toolkit to support the geospatial use cases described in this work.

In this regard, SPARQL Anything[10] covers a similar ground. The main difference is that SPARQL Anything extends the semantics of the SERVICE clause to perform opinionated RDF-isation of non-RDF sources. As a step towards future interoperability of these complementary approaches by means of allowing for plugin abstractions we contributed an improved custom SERVICE extension system to Jena[5].

3. Approach

We provide a comprehensive framework to integrate several kinds of geospatial data sources into RDF data. The main components are:

- RDF Processing Toolkit[6] can be easily used on command line to integrate a manifold of different data sources using SPARQL standards and extensions.
- Apache Jena[7] is a free and open source Java framework for building Semantic Web and Linked data applications. Our extension module JenaX[8] adds additional features like web API and JSON processing.
- GeoSPARQL[9] is maintained by the Open Geospatial Consortium. It defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data.

Our approach is to integrate and enhance GeoSPARQL as well as extend and enhance SPARQL so as to integrate and query various data sources, including geospatial data, using Semantic Web technologies. In what follows, our main contributions within each of the above component are shortly described.

3.1. Apache Jena

Streaming results functions The existing GeoSPARQL implementation in Apache Jena has been extended to return geo-objects in a streaming manner[10]. This greatly improves the

[5]https://github.com/apache/jena/pull/1388
[6]https://github.com/SmartDataAnalytics/RdfProcessingToolkit
[7]https://jena.apache.org/
[8]https://github.com/Scaseco/jenax
[9]https://www.ogc.org/standards/geosparql
[10]https://github.com/apache/jena/pull/1331

performance of SPARQL queries making use of LIMIT, as the result set computation is stopped once the limit is reached.

geof:asGeoJSON A function to convert the geometry data into the GeoJSON format [11]. This is especially helpful since many JavaScript frameworks support drawing GeoJSON data on a map[11].

3.2. JenaX

Our Jena extension framework JenaX features extensions for query rewriting, caching, additional optimisers, utilities and abstractions for common tasks and a set of SPARQL extensions for working with remote and local data in various formats. A selection of highly relevant features for processing spatial crisis data is as follows:

geof:lineMerge Some datasets such as OpenStreetMap represent large objects as a collection of geometries. Linemerge can be used to turn a collections of line strings, such as for a river or administrative boundary, into a single RDF literal.

geof:simplifyDp A function that uses the Douglas-Peucker algorithm to simplify polygons. A typical use case is to improve the performance of map visualisations on client devices (e.g. mobile phones) by reducing the number of vertices of large spatial objects.

geof:aggCollect An aggregate function to group geometries into a single RDF literal holding a WKT geometry collection for further processing (by using lineMerge or simplifyDp, for instance).

CSV Using `{ <source.csv> csv:parse (?jsonRow "excel -h -o") }`, a CSV source can be streamed into a set of solutions where each CSV row is represented as an RDF literal that holds a JSON array or object. A dialect such as `excel` can be specified followed by flags that provide additional control. For example, `-h` treats the first row as headers and, applying it in conjunction with `-o`, each row is returned as a JSON object – rather than an array – with the headers as keys.

JSON Loading of JSON documents is non-streaming, which means that they are always fully copied into memory. The `xsd:json` datatype can be used with SPARQL's conventional machinery for parsing strings into RDF literals: `BIND('{"key": "value"}'^^xsd:json AS ?json)`. Attributes can be extracted using `BIND(json:path(?json, "$.name") AS ?name)`, and arrays can be unnested into individual bindings using `?jsonArray json:unnest (?jsonItem ?index)`. Note, that the maximum byte size of strings in Java is 2GB which is a limiting factor when passing documents as strings.

[11] https://plotly.com/python/mapbox-county-choropleth/

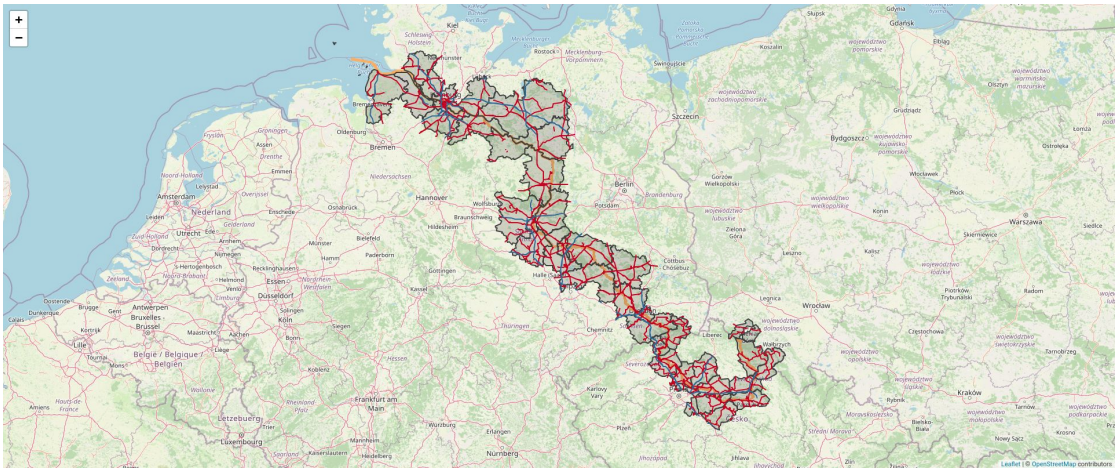

Figure 1: Elbe river with nearby transport and administrative regions

XML The functions for XML processing are similar to those of CSV and JSON: An XML document can be loaded from a local or remote source using `<source.xml> xml:parse ?xml` and XPath expressions of the form `?xml xml:unnest ("//item" ?item)` can be used to create individual SPARQL solution bindings from each match. Note, that our XML functions avoid needless string-serialization of the internal XML model which means that documents larger than 2GB can be ingested and processed. However, when eventually writing the query results out, no RDF literal's serialization must exceed that limit.

3.3. RDF Processing Toolkit

The RDF processing toolkit (RPT) is the command line front end that bundles various functionalities to realise SPARQL-based ETL workflows: Connecting to a remote SPARQL endpoint or starting a new local one, loading RDF data, executing queries that may exploit the aforementioned function extensions, and doing serialization of query results both in standard and custom formats. Environment variables can be accessed from within SPARQL, again using function extension `sys:getenv('envVarName')` as well as syntax convention: Any IRIs in SPARQL statements of the form `<env:varName>` will be substituted with a value derived from the corresponding environment variable's value. Further conventions exist to control the RDF term type, datatype and language tag.

4. Exemplary Results

In this section, we present selected examples that showcase what our tools can deliver.

Live API consumption and federation One show-case is the querying of Web APIs and the geographic data processing. Essentially, one can either perform live requests against those APIs or materialise them into the Knowledge Graph.

Listing 1: Retrieving the river Elbe with intersecting administrative areas

```
SELECT ?adm {
  BIND("rgb(43,131,186)" AS ?geomColor)
  SERVICE <cache:https://coypu.aksw.org/osm/sparql> {                  #  (1)
    SELECT (geof:lineMerge(geof:collect(?lit)) AS ?geom) {             #  (2)
      ?s a osm:relation ; osmkey:name:de "Elbe" ; osmrel:member/osm:id ?m .
      ?m geo:hasGeometry/geo:asWKT ?lit } }
  GRAPH <https://data.coypu.org/adm2> {
    ?adm spatial:intersectBoxGeom(?geom) .                             # (3a)
    ?adm geo:hasGeometry/geo:asGML ?adm_geom_lit_ .
    FILTER(geof:sfIntersects(?geom, ?adm_geom_lit_)) }                 # (3b)
}
```

Either approach has advantages and disadvantages. APIs on the one hand are up-to-date and easy to use but one might have to deal with request limits and coarse granularity. Materialised geographic data, on the other hand, can be linked with other existing concepts but could be out of sync after some time. For example, the Nominatim API hardly provides geometries of highways because it is mostly concerned with things that have addresses. If we materialise the OSM data ourselves, we can choose exactly the kind of data we want to load.

In the first experiment, we want to visualise the river Elbe on a map. Following the first of the two mentioned options, this can be implemented using a live consumption approach. In that case, we apply a SPARQL query that uses basic federation[12] to retrieve content from a Wikidata. In the last step, the geospatial information is extracted from the raw JSON result (using ***url:text***, `xsd:json` and *`json:path`*). This is shown in the Appendix, File 2.

The second approach requires transformation before the geospatial data can be made available in a Knowledge Graph. In this example, data from OpenStreetMap is transformed to RDF using a "simple" mapping. While simple mapping does not follow best practices in RDF/OWL modelling, it has the advantage that it can be executed quickly and still enables access to all of OpenStreetMap's data with SPARQL. This way we can retrieve all information about a river using simple triple patterns. Large objects in OSM are typically modelled as collections of smaller ones, so in order to to obtain a single line-string (or line-collection; whatever is appropriate) for a river it has to be assembled from fragments. We can use our custom GeoSPARQL functions `lineMerge` and `collect` to do this as demonstrated as part of Listing 1. The full query is shown in the Appendix, File 3.

Geo-linking We continue to work with the example of the Elbe river and further expand it with other Knowledge Graphs that we have collected in our triple store. For example, a graph with railroad network, another graph with road network, a graph with countries and administrative regions, like states, or a graph with company data and their locations.

This way, we can use GeoSPARQL to answer the questions like: "Through which adminis-

[12]SERVICE <https://query.wikidata.org/sparql> {...}

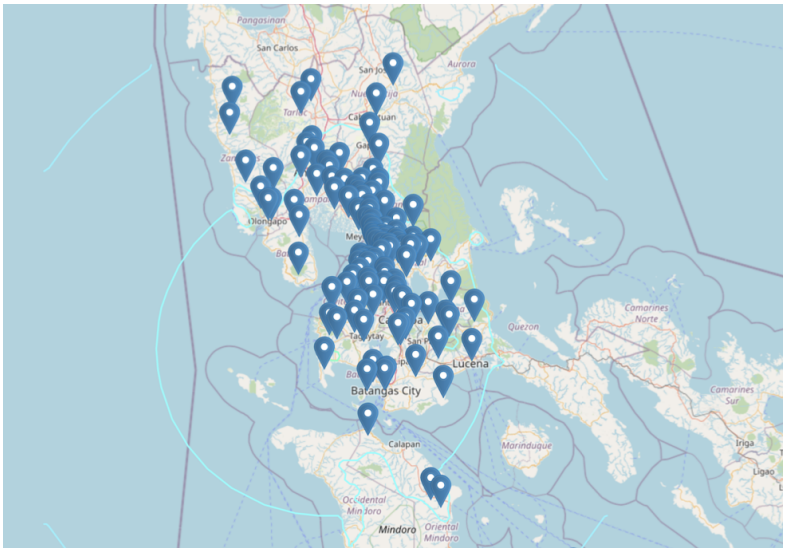

Figure 2: Earthquake region with affected companies

trative regions does the river flow?", "What major transportation infrastructure is nearby the river?", "What companies are nearby?" These queries are more easily answered when we have Geographic information available in SPARQL, than if we only have non-geographic SPARQL available.

In Figure 1, the river Elbe is shown, now linked to information of the railroad and road network and to administrative regions. The query in Listing 1 demonstrates (1) how to merge all line-strings belonging to the Elbe (2) cache the result and (3) use it find intersecting administrative areas. The complete query is linked in the Appendix, File 4.

In the Coypu project, we envision applying the same principles to similar use cases. For example, with sufficient data about flooded region we could find out which roads might be blocked, which companies might be affected, or which state should provide disaster relief.

Earthquake regions Another use-case within the Coypu project is to find companies potentially affected by earthquakes, as shown in Figure 2.

In this example, we combine again two approaches available in our tools. First, we consume a live feed of significant earthquakes published by the U.S. Geological Survey, Department of Interior – using ***`url:text`*** and *`json:path`*. Next, we match it to company locations that we have stored in a Knowledge Graph using GeoSPARQL and the function ***`spatial:withinBoxGeom`***. Finally, we convert the GeoJSON to WKT for display purposes, using the `geo:geoJSONLiteral` data type and the *`spatialF:transformDatatype`* function. The query for this is presented in the Appendix, File 5.

Mapping Geography Markup Language Another common format used with geospatial data is the Geography Markup Language. Many Geographic data sets are available as Geography Markup Language (GML). Often, the GeoServer program is used to host data sets which can be

Listing 2: Extracting GML from an XML document referenced by URL from the environment

```
SELECT * {
  BIND(xml:parse(<env://INPUT>) AS ?xmlstream) .
  ?xmlstream xml:unnest
    ("/*[local-name()='FeatureCollection']/*[local-name()='member']/*" ?item) .
  BIND(xml:path(?item, "/*/geonode:portname/text()") AS ?portname) . [  ...  ]
  BIND(xml:path(?item, "/*/*/*[namespace-uri()='http://www.opengis.net/gml/3.2']")
     AS ?geo_) .
  BIND(strdt(str(?geo_), geo:gmlLiteral) AS ?geoLitGML) .
}
```

exported as GML.

The GML format is XML based. We can transform it into RDF, for example by using our RDF Processing Toolkit. The key functions are shown in Listing 2.

Using ***xml:parse***, the GML file is opened from the location specified in the `INPUT` environment variable. Then, for each fragment of the GML document matching the XPath expression specified in the ***xml:unnest*** method, a new `?item` is created. Later on, the various properties in the items are mapped to RDF using **`BIND`** and *`xml:path`*. GeoSPARQL then allows us to consume the GML Geometries contained in the GML document, using the standard *`strdt`* function. If we have the GML mapped to RDF, we can again use this new Knowledge Graph. The full sample query to map a GML data source is shown in the Appendix, File 6.

One application that is enabled by the fusion of data sources, such as the Water Ports database maintained by the World Food Programme (WFP), is the comparison to data mapped in the OpenStreetMap Crowd Sourced Mapping data of the world. Here, we can investigate how many port structures can be located in OSM, and which ones are in the WFP dataset. We do this using the function `?osm_geometry spatial:nearbyGeom(?wfpdb_geometry 3 units:kilometre)` to find port structures in OpenStreetMap that are within three kilometres of a port inside the WFP database. The full query for this is shown in the Appendix, File 7.

Pitfalls We conclude this section by summarising some quite common and relevant ones within the context under consideration.

One problem that we found was, that consuming GML in Jena GeoSPARQL did not work. We traced this down to a wrong XML namespace used in the GeoSPARQL standard, and opened an issue about it which led to a fix in the specification.

We encountered another issue when downloading GML from a GeoServer. It is possible that the requested Coordinate Reference System is not supported. In that case, the GeoServer might silently respond with an invalid Coordinate Reference System URI, effectively making the GeoSPARQL engine unable to process the data properly.

A further issue arises when using SPARQL to RDFize of large XML documents, such as ones downloaded from a GeoServer. The amount of data that fits into an RDF literal is typically capped by the maximum string length (in Java 2GB). In order to raise the cap to the available amount of RAM, we revised `xml:parse` to perform a streaming parse of the XML input into

Listing 3: Coordinate reference system transformations in order to calculate buffer in kilometres

```
SELECT * {
  BIND(spatialF:transformSRS(?river_geom,
          <http://www.opengis.net/def/crs/EPSG/0/5243>) AS ?river_geom_metric) .
  BIND(geof:buffer(?river_geom_metric,
          "1"^^xsd:double, <http://www.opengis.net/def/uom/OGC/1.0/kilometer>)
    AS ?river_with_buffer) .
  BIND(spatialF:transformSRS(?river_with_buffer,
          <http://www.opengis.net/def/crs/OGC/1.3/CRS84>) AS ?final_geom) .
}
```

an in-memory model that can be subsequently chunked into serialisable RDF literals with `xml:unnest`.

When looking at the SPARQL query needed for Figure 1, there is another thing to be aware of: When calculating a buffer around the river to find entities nearby the river, it is not directly possible to do in metres. This is because of the World Geodetic System being in degree. We have to choose a metric reference system and convert the geometries back and forth. The required SPARQL functions are shown in Listing 3.

Another interesting performance issue arose when we tried to geo-match all companies located in the United States. Because the pre-filter is using an envelope of the USA geometry, and there are a few tiny uninhabited islands which are claimed by the U.S., the bounding box almost encompasses the whole world. Thus, it is more efficient to break the U.S. down into its parts before locating the companies.

If we want to find power lines in e.g. OSM, we are quickly overwhelmed with the huge number of matches. But not all power lines are be relevant for our project. Fortunately, we can reduce the result set via voltage and/or length features, depending on country conventions.

5. Conclusion

In this work, we contributed to and showed the usefulness of three major components with several features: First of all, to Apache Jena, we contributed batch service requests, GeoSPARQL performance improvements, GeoJSON export, and various fixes, such as for the issue that could cause compressed datasets to only become loaded partially. Secondly, in our JenaX extensions module, we provide enhancements for querying remote web APIs from inside SPARQL, we improved GeoSPARQL support thereby implementing some features from the upcoming GeoSPARQL 1.1 standard as well as the presently non-standard functions `collect`, `union`, `lineMerge`, `simplify`, `lat`, `lon`, `centroid`, and a GeoJSON reader, among others. Furthermore, we provide extensions for JSON, CSV, XML and generic array processing inside SPARQL. This way, SPARQL can be exploited to serve as the host language for the integration of data in virtually any format. And finally, we presented and produced the RDF Processing Toolkit, a versatile RDF processing program which can make use of all our extensions. We will continue to work on and further improve these tools.

Acknowledgements The authors acknowledge the financial support by the Federal Ministry for Economic Affairs and Energy of Germany in the project Coypu (project number 01MK21007[A-L]).

References

[1] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the semantic web recommendations, in: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, WWW Alt. '04, Association for Computing Machinery, New York, NY, USA, 2004, p. 74–83.

[2] G. Albiston, T. Osman, H. Chen, Geosparql-jena: Implementation and benchmarking of a geosparql graphstore, Semantic Web Journal (2019).

[3] C. Stadler, J. Lehmann, K. Höffner, S. Auer, Linkedgeodata: A core for a web of spatial open data, Semantic Web Journal 3 (2012) 333–354.

[4] J. Lehmann, S. Athanasiou, A. Both, A. Garcia-Rojas, G. Giannopoulos, D. Hladky, K. Hoeffner, J. J. L. Grange, A.-C. N. Ngomo, M. A. Sherif, C. Stadler, M. Wauer, P. Westphal, V. Zaslawski, Managing Geospatial Linked Data in the GeoKnow Project, Studies on the Semantic Web, 2015, pp. 51–78.

[5] H. Bast, P. Brosi, J. Kalmbach, A. Lehmann, An efficient rdf converter and sparql endpoint for the complete openstreetmap data, in: Proceedings of the 29th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 536–539.

[6] T. Homburg, C. Prudhomme, F. Würriehausen, A. Karmacharya, F. Boochs, A. Roxin, C. Cruz, Interpreting Heterogeneous Geospatial Data Using Semantic Web Technologies, in: International Conference on Computational Science and Its Applications (ICCSA 2016), Beijing, China, 2016, pp. 240–255.

[7] J. Jones, W. Kuhn, C. Keßler, S. Scheider, Making the Web of Data Available Via Web Feature Services, Springer International Publishing, Cham, 2014, pp. 341–361.

[8] L. Wenige, C. Stadler, S. Bin, L. Bühmann, K. Junghanns, M. Martin, Automatic subject indexing with knowledge graphs, in: LASCAR Workshop at the Extended Semantic Web Conference (ESWC), 2020.

[9] C. Stadler, L. Wenige, S. Tramp, K. Junghanns, M. Martin, Rdf-based deployment pipelining for efficient dataset release management, in: SEMANTICS Posters&Demos, 2019.

[10] E. Daga, L. Asprino, P. Mulholland, A. Gangemi, Facade-x: An opinionated approach to sparql anything, in: M. Alam, P. Groth, V. de Boer, T. Pellegrini, H. J. Pandit (Eds.), Volume 53: Further with Knowledge Graphs, volume 53, IOS Press, 2021, pp. 58–73. URL: https://sparql-anything.cc/.

[11] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, S. Hagen, The GeoJSON Format, Technical Report 7946, 2016.

Appendix

Complete SPARQL queries available at: https://github.com/AKSW/D2R2_2022_GeoQueries