Querying OpenEO Datacubes through Virtual Knowledge Graphs

Albulen Pano

Free University of Bozen-Bolzano, Faculty of Engineering, 39100 Bolzano, Italy

Abstract

Earth Observation (EO) data are becoming a ubiquitous and readily exploitable source of data for decision making. The openEO platform is a renowned solution to query EO data using webAPIs. Running integrated queries encompassing both EO data and data from diverse domains is achieved using Knowledge Graphs (KGs). However, the volume of EO data makes its materialization in a knowledge graph impractical. In this contribution we look at the integration of data from openEO with other data sources using the Virtual Knowledge Graph paradigm, providing the advantages of KGs while avoiding the need of creating copies of the data. Our approach enables answering cross-domain user queries over openEO federated with data coming from other sources.

Keywords

Virtual knowledge graph, Earth observation, Ontology-based data access, Raster data, openEO

1. Introduction

The rapid rise in satellite deployments for Earth Observation (EO) has brought about a significant expansion of Earth imagery. Growth has impacted not only the amount of data, but also the sources that disseminate these data for public consumption. This heterogeneity has led to interoperability issues that have been addressed by the development of the *openEO platform* [1, 2], an Application Programming Interface (API) that allows connections to EO data through dedicated cloud backends. Access to a single Web API such as openEO ensures that the data is analysis-ready, allowing users to seamlessly manipulate and integrate it into their application. The openEO specification was submitted to the Open Geospatial Consortium to become a standard in December 2023 [3]. However, although openEO provides a single data source for EO data and can even federate individual openEO backends together [4], resolving the issue of heterogeneity *within* the EO domain, the issue of heterogeneity between openEO and other non-EO data sources needs to be further addressed.

Knowledge Graphs (KG) provide an approach to integrate heterogeneous data across a variety of data sources and data formats, most commonly using the Resource Description Framework (RDF)[1] standard to exchange data. The use of a graph data model offers a flexible way to conceptualize, model, and integrate knowledge by making relationships explicit. In the context of EO data, this would mean that every pixel would be connected via properties within the KG using an Internationalized Resource Identifier (IRI)[2], but also that EO data are represented as higher level conceptualizations. The enormous amount of satellite data collected and accessible via openEO renders the generation of such large KGs via this granular approach infeasible, as it would quickly overload data storage demands.

An alternative solution would be to deploy a *Virtual KG* (VKG) instead, which generates a KG only at query time for the data requested by the user query. The VKG paradigm offers an alternative which would avoid duplicating any openEO data and integrate it with other non-EO data sources. A prominent open-source solution that implements the VKG paradigm for relational database data is Ontop [5, 6], which however needs to be extended to support queries over Web APIs such as openEO. In this contribution we present an approach on how to integrate and query EO data via the openEO specification by adding new raster functions to Ontop. This approach provides both the opportunity to

SEBD 2025: 33rd Symposium On Advanced Database Systems, Ischia (Italy), 16 Jun - 19 Jun 2025

albulen.pano@unibz.it (A. Pano)

[1] https://www.w3.org/RDF/

[2] https://www.ietf.org/rfc/rfc3987.txt

perform cross-domain queries from EO data to other domains, and leverage the VKG approach to avoid storing any EO data locally.

The paper is organized as follows: Section 2 reviews related work, while Section 3 details the paradigm and outlines preliminary implementation details in Ontop. Finally, Section 4 offers concluding remarks and discusses future research directions.

2. Related Work

Software solutions are available to query raster data, including, e.g., relational database extensions such as PostGIS[3] in PostgreSQL, or multi-language solutions such as RasterFrames[4] (using Python, Scala, and SQL). However, the challenge of analyzing EO data and specifically their raster representation using knowledge graphs has not yet garnered consensus. The *Spatial Data on the Web Working Group* [7, 8] recognizes the difficulty of representing all raster spatial data as linked data. Indeed, a *spectrum of linkiness* is introduced whereby practitioners who wish to publish EO data do not have to either *(i)* publish all EO data points as RDF triples, which would be resource constraining, or conversely *(ii)* publish metadata on the data sources, which would be uninformative. We advocate for an approach that balances these two extremes by leveraging the Ontology-Based Data Access (OBDA) paradigm to construct VKGs. Through virtualization, raster data is published on-demand at query time, mitigating the challenge of converting large volumes of RDF triples. Additionally, OBDA provides a high-level conceptualization through an ontology, enabling greater abstraction beyond specific data points. This facilitates the integration of EO data with diverse sources while enhancing explainability.

Research has been conducted on extending the GeoSPARQL syntax for vector functions to raster data, and respective prototype implementations have been built. GeoSPARQL+ [9] implements via Apache Jena an extension of GeoSPARQL that can handle raster operations and map individual observations within geospatial rasters or coverages to IRIs. GeoLD [10] provides its own syntax extensions and uses a SPARQL optimization engine built over the Apache Jena ARQ engine and a Web Coverage Processing Service (WCPS) node to delegate some of the computations to WCPS services using Petascope on the Rasdaman[5] array database. GeoLD outperforms GeoSPARQL+ in terms of query response time.

Additional approaches [11] used to analyze EO using VKGs include using PL/pgSQL, the PostgreSQL SQL Procedural Language, to first transform the data into a tabular format and then expose them as a VKG. This approach involves an expensive computational step of calculating all needed measurements for each raster grid cell, significantly increasing both the storage requirements and processing time. While physical data files can be hidden behind the openEO Web API, downloading EO data locally into, e.g., a netCDF or GeoTIFF file in order to then store them in a relational database, completely foregoes the storage advantage of querying data via openEO remotely.

The Plato data cube semantic [12] demonstrates an alternative use of Ontop over raster data by leveraging the Python xArray package to virtually retrieve data from raster files in the NetCDF format. The solution involves a caching mechanism over spatial indexes to improve query performance over vector and raster data. A custom *Raptor Join* [13], which efficiently filters raster data that overlap with vector data, is also demonstrated avoiding any conversion costs between the data. While the solution mentions Web APIs, it was not mentioned to have been tested over OpenEO. The approach provides an original solution to joining vector-raster data, implementing it in the context of VKGs. However, designing a new prototype solution for geospatial joins is unlikely to be as efficient as mature geospatial solutions, such as the raster extension in PostGIS or Rasdaman, and Plato has not been benchmarked with traditional geospatial systems.

OntoRaster [14] extends the GeoSPARQL syntax over VKGs to leverage the functionalities provided by the array database Rasdaman. The approach involves the use of PL/Python[6] queries over PostgreSQL

[3] https://postgis.net/docs/RT_reference.html
[4] https://rasterframes.io/
[5] http://www.rasdaman.org/
[6] https://www.postgresql.org/docs/current/plpython.html

acting as a data federator over Rasdaman. Rasdaman is a mature raster database which provides performance benefits when performing raster filtering based on vector data, however its query language, *rasql*, has limited features which pose limits to the OntoRaster solution. Furthermore, all raster files such as NetCDF or GeoTIFF need to be first ingested into Rasdaman before being queried by a VKG, which doubles storage requirements and introduces an additional data transformation step to the VKG construction pipeline.

Most of the research detailed earlier focuses on modelling EO classes and properties, as well as functions that extend the official GeoSPARQL standard, instead of considering comprehensive EO raster functions. Function calls that can execute raster or EO functions over remote endpoints have been discussed and implemented for Apache Jena and Virtuoso [15, 16], even including the definition of LDScript [17], a Linked Data script language on top of the SPARQL filter expression language. Descriptions of custom EO-related SPARQL functions, have been developed so that their signatures and expected output are independent of the development context, to facilitate interoperability [18].

A further contribution in this direction is the development of a module within the RDF Mapping Language (RML) called *RML-FNML* [19]. Rather than integrating a fixed set of functions, RML-FNML enhances RML with declarative function descriptions using the Function Ontology (FnO)[7], providing a more flexible and extensible approach. Practitioners are realizing that built-in functions in RML systems are insufficient to handle the heterogeneity in input data and to address data processing tasks where more flexibility is needed [20]. User-defined functions can be therefore introduced and prototypes for RML with custom functions have been demonstrated.

3. Implementation

We extend here the OntoRaster framework [14] to the openEO Web API. The architecture of the solution remains similar and is presented in Figure 1, with the difference that it uses openEO as backend, instead of Rasdaman. The following discussion of the implementation focuses on the challenges and additional features introduced by the support of a Web API such as openEO.

3.1. openEO Data

Several openEO cloud-based infrastructure providers exist which provide computing resources, storage capacity and the networking capabilities required to handle large scale data processing requirements. Two such platforms are Copernicus[8], hosted by Copernicus Data and Information Access Services (DIAS), and the one hosted at the Institute for Earth Observation at EURAC Research[9]. Platforms such as the one managed by EURAC provide, in addition to Sentinel satellite data[10], a better integration with regional datasets.

EO data is typically represented as raster data, structured as an n-dimensional data cube that encapsulates all collected dimensions. The data is retrieved primarily from satellites, e.g., Sentinel constellation. Datacubes are routinely filtered by domain experts based on spatial extent, temporal extent, and the light band of interest. In a datacube with dimensions x, y, t, where x is the longitude (east-west), y the latitude (north-south) and t the time, each spatio-temporal coordinate is associated with one or more values corresponding to the spectral bands dimension. For example, the satellite Sentinel-2 has 13 individual bands (including, e.g., *B02*, which captures visible blue light, useful for water body analysis and atmospheric studies).

We will conduct our analysis and demonstration using openEO data accessed through the Copernicus Data Space Ecosystem. Our study focuses specifically on cities within South Tyrol, utilizing municipal boundary *shapefiles* obtained from the regional geocatalogue[11] as our vector data.

[7] https://fno.io/rml/
[8] https://dataspace.copernicus.eu/analyse/openeo
[9] https://www.eurac.edu/en/institutes-centers/institute-for-earth-observation/projects/openeo
[10] https://sentinels.copernicus.eu/
[11] https://geonetwork1.civis.bz.it/geonetwork/geonetwork/ita/catalog.search#/home

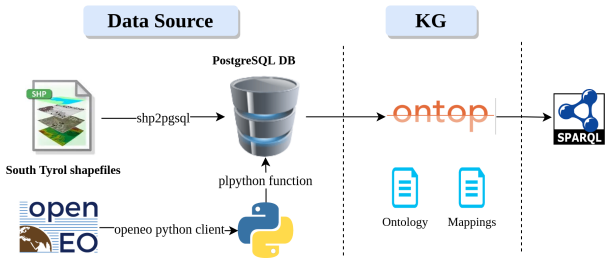

Figure 1: OntoRaster architecture for openEO

3.2. openEO queries

Queries over the openEO Web API are expressed as a JSON *process graph*[12], a directed acyclic graph where nodes represent individual processes, and edges define the flow of data between them, and with exactly one sink node (returning the final result). A process in openEO is an operation that performs a specific task on a set of parameters and returns a result [21]. Processes are applied sequentially in the process graph, e.g., a first process can load a collection of data and a second process can take the result as input to filter that data. Individual processes can also be applied as subprocesses, e.g., statistical functions such as mean, maximum, or minimum applied over the temporal dimension.

3.3. Ontop

As OBDA system, we use *Ontop v5.2.0*, providing query answering capabilities over VKGs. Ontop supports the RDF data model, the SPARQL[13] query language, the OWL 2 QL[14] profile of the OWL ontology language, and the R2RML[15] mapping language. We have extended Ontop by implementing a set of SPARQL functions mapped to corresponding openEO functions. In our setting, a SPARQL query is able to emulate, through composition of SPARQL functions, a process graph which a user would specify either in the openEO python package or via web interfaces. Our implementation takes the SPARQL input and reformulates it into a corresponding process graph in JSON, which is then sent to the openEO endpoint for execution.

We illustrate our implementation with a query that uses the band S8 (*thermal infrared*) to approximate temperature (ignoring atmospheric effects). For this query, we defined the SPARQL function `openeo:load_collection` inside Ontop, and mapped it to the `load_collection` function of the openEO API. The function accepts the satellite name, start time, end time, spatial extent, and bands as arguments. These are then passed to a predefined PL/Python function, which in turn receives the process name (e.g., `load_collection`) along with the arguments.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX openeo: <http://www.openeo-ontop.org#>

SELECT ?avg_temp_kelvin {
  ?g geo:asWKT ?geomWKT .
  ?g rdfs:label ?name .
  FILTER(LANG(?name) = "it" && (STR(?name) = "Bolzano")) .
  BIND ("2023-09-01T00:00:00Z"^^xsd:dateTime AS ?start_time) .
  BIND ("2023-09-07T00:00:00Z"^^xsd:dateTime AS ?end_time) .
  BIND ("SENTINEL3_SLSTR" AS ?satellite) .
  BIND ("S8" AS ?band) .
  BIND (openeo:load_collection(?satellite, ?geomWKT, ?start_time, ?end_time, ?band)
      AS ?coll1) .
```

[12] https://api.openeo.org/v/0.3.0/processgraphs/
[13] https://www.w3.org/TR/sparql11-query/
[14] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL
[15] https://www.w3.org/TR/r2rml/

```
BIND (openeo:reduce_dimension(?coll1, "t", "mean") AS ?coll2) .
BIND (openeo:aggregate_spatial(?coll2, ?geomWkt, "mean") AS ?avg_temp_kelvin) . }
```

3.4. PostGIS

The vector data from the shapefiles is loaded directly into a geometry table in PostgreSQL 17 and its geometry extension PostGIS 3.5.0. Ontop already supports queries over relational databases with geospatial support such as PostGIS, but is limited to the *Well-Known Text* (WKT) RDF geometry serialization for GeoSPARQL functions.

PL/Python is a procedural language that exploits functions written in Python to return PostgreSQL query results with PostgreSQL datatypes. There is a well-maintained openEO Python client[16] that can accept openEO process graphs as input. We use Python 3.11 with openEO 0.33.0 to connect to openEO cloud backends of choice, and shapely 2.0.2 to convert geometries expressed as WKT into a format that is compliant with openEO process graphs.

As part of the Python implementation, we create several templates for openEO processes, which can be completed with arguments coming from the SPARQL functions described above. The input retrieved from Ontop will replace each of the arguments named `argi` (see, e.g., the example below for `load_collection`).

```
unique_node_id: {
      "process_id": load_collection,
      "arguments": {
        "id": arg1,
        "spatial_extent": { arg2 },
        "temporal_extent": [ arg3, arg4 ],
        "bands": [ arg5 ]
      }
    }
```

4. Conclusions and Perspectives

We have shown how to query openEO Web API via SPARQL, using the Ontop VKG system and PostgreSQL as a federator with its PL/Python functions.

An area that we plan to further explore are User Defined Processes inside openEO, which would allow us to avoid the nesting of SPARQL functions and to use instead single functions calls over custom process graphs. We are currently working on a new mapping language in Ontop to further improve the mapping of geospatial concepts, rather than relying on SPARQL functions, as in our current proposal. E.g., incorporating notions from *RML-FNML* will allow the modelling of concepts at a higher level of abstraction, e.g., heatwaves and wildfires, again avoiding to use nesting of SPARQL function calls. This would also improve interoperability with other ontologies. A further question that arises is how to handle the important aspect of privacy in such a distributed, heterogeneous setting, which natural leads to the presence of data with different degrees of confidentiality. To address this, we are considering to incorporate into the new mapping language the facet of privacy-preservation, contingent upon the identity of the user running a query.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

[16] https://openeo.org/documentation/1.0/python/

Acknowledgments

This work has been carried out while Albulen Pano was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome in collaboration with Free University of Bozen-Bolzano. This work was supported by Agenzia per la cybersicurezza nazionale under the programme for promotion of XL cycle PhD research in cybersecurity – B83C24005640005. The views expressed are those of the authors and do not represent the funding institution.

References

[1] OpenEO, OpenEO: An Open Earth Observation Processing Platform, https://openeo.org/, 2024. Accessed: 2024-12-10.

[2] A. Jacob, M. Mohr, P. J. Zellner, J. Dries, M. Claus, C. Briese, P. Griffitjs, E. Pebesma, OPENEO PLATFORM BRINGS ANALYSIS-READY DATA ON DEMAND, in: Proceedings of the 2021 Conference on Big Data from Space, EURAC, Virtual Event, 2021, pp. 45–48. URL: https://creativecommons.org/licenses/by/4.0/, except otherwise noted, the reuse of this document is authorised under the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

[3] Open Geospatial Consortium, OGC API – Processes: Part 1: Core, https://portal.ogc.org/files/106981, 2024. Draft Standard, Accessed: 2024-12-10.

[4] M. Mohr, E. Pebesma, J. Dries, S. Lippens, B. Janssen, D. Thiex, G. Milcinski, B. Schumacher, C. Briese, M. Claus, A. Jacob, P. Sacramento, P. Griffiths, Federated and reusable processing of earth observation data, Scientific Data 12 (2025) 194. doi:10.1038/s41597-025-04513-y.

[5] G. Xiao, D. Lanti, R. Kontchakov, S. Komla-Ebri, E. Güzel-Kalayci, L. Ding, J. Corman, B. Cogrel, D. Calvanese, E. Botoeva, The Virtual Knowledge Graph system Ontop, in: International Semantic Web Conference, Springer, 2020, pp. 259–277. doi:10.1007/978-3-030-62466-8_17.

[6] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web Journal (2016).

[7] W3C/OGC Spatial Data on the Web Working Group, Spatial Data on the Web Working Group, https://www.w3.org/2021/sdw/, 2021. Accessed: 2024-12-10.

[8] W3C Education and Outreach Working Group, How to Make Your Data Easier to Use with Quality Metadata, https://www.w3.org/TR/eo-qb/, 2023. W3C Working Group Note, Accessed: 2024-12-10.

[9] T. Homburg, S. Staab, D. Janke, Geosparql+: Syntax, semantics and system for integrated querying of graph, raster and vector data, in: J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, L. Kagal (Eds.), The Semantic Web – ISWC 2020, Springer International Publishing, Cham, 2020, pp. 258–275.

[10] S. B. Almobydeen, J. R. Viqueira, M. Lama, Geosparql query support for scientific raster array data, Computers & Geosciences 159 (2022) 105023. doi:https://doi.org/10.1016/j.cageo.2021.105023.

[11] Y. Hamdani, G. Xiao, L. Ding, D. Calvanese, An ontology-based framework for geospatial integration and querying of raster data cube using virtual knowledge graphs, ISPRS International Journal of Geo-Information 12 (2023). doi:10.3390/ijgi12090375.

[12] D. Bilidas, A. Mantas, F. Yfantis, G. Stamoulis, M. Koubarakis, J. M. T. Habas, E. S. Marco, F. Castel, C. Laine, The semantic data cube system plato and its applications, in: IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium, 2024, pp. 2514–2518. doi:10.1109/IGARSS53475.2024.10640737.

[13] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, E. Scudiero, The raptor join operator for processing big raster + vector 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. 324–335. doi:10.1145/3474717.3483971.

[14] A. Ghosh, A. Pano, G. Xiao, D. Calvanese, Ontoraster: Extending vkgs with raster data, in:

S. Kirrane, M. Šimkus, A. Soylu, D. Roman (Eds.), Rules and Reasoning, Springer Nature Switzerland, Cham, 2024, pp. 108–123.
[15] M. Atzori, Toward the web of functions: Interoperable higher-order functions in sparql, in: P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, C. Goble (Eds.), The Semantic Web – ISWC 2014, Springer International Publishing, Cham, 2014, pp. 406–421.
[16] D. S. Ferru, M. Atzori, Write-once run-anywhere custom sparql functions, in: 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), 2016, pp. 176–178. doi:10.1109/ICSC.2016.64.
[17] O. Corby, C. Faron-Zucker, F. Gandon, Ldscript: A linked data script language, in: C. d'Amato, M. Fernandez, V. Tamma, F. Lecue, P. Cudré-Mauroux, J. Sequeda, C. Lange, J. Heflin (Eds.), The Semantic Web – ISWC 2017, Springer International Publishing, Cham, 2017, pp. 208–224.
[18] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent function reuse, Future Generation Computer Systems 110 (2020) 946–959. doi:https://doi.org/10.1016/j.future.2019.10.006.
[19] J. Arenas-Guerrero, P. Espinoza-Arias, J. A. Bernabé-Diaz, P. Deshmukh, J. L. Sánchez-Fernández, O. Corcho, An rml-fnml module for python user-defined functions in morph-kgc, SoftwareX 26 (2024) 101709. doi:https://doi.org/10.1016/j.softx.2024.101709.
[20] J. Arenas-Guerrero, Handling data transformations in virtual knowledge graphs with rml view unfolding, in: K. Stefanidis, K. Systä, M. Matera, S. Heil, H. Kondylakis, E. Quintarelli (Eds.), Web Engineering, Springer Nature Switzerland, Cham, 2024, pp. 424–427.
[21] Copernicus Data Space Ecosystem, Copernicus data space ecosystem: openeo processes, https://documentation.dataspace.copernicus.eu/APIs/openEO/Processes.html, 2025. Accessed: 2025-03-19.