EOPEN

opEn interOperable Platform for unified access and analysis of Earth observatioN data

H2020-776019

D5.3

Linked Open EO Data

Dissemination level:	Public
Contractual date of delivery:	Month 33, 31/07/2020
Actual date of delivery:	Month 34, 21/08/2020
Workpackage:	WP5 Semantic representation and the EOPEN ontology
Task:	T5.2 Linked open EO data T5.3 Reasoning for decision support
Type:	Demonstrator
Approval Status:	Approved
Version:	1.0
Number of pages:	60
Filename:	D5.3-Linked Open EO Data v1.0rc1.docx

Abstract

This deliverable describes the result of the work performed in the EOPEN project to provide Linked Open Earth Observation Data to application developers and users. The document starts with providing the relevant technical requirements extracted from D6.1 which have been used to drive the work. Then a short introduction to the Linked Data concepts, tools and technologies is provided, including the ontologies that are used afterwards to represent EO products metadata as Linked Data.

The report then describes the outcome of the activity including the EO data for which conversion rules have been defined, the processes and workflows that have been implemented, as well as the graphical components that may be used to search for and visualise the data in the EOPEN User Portal.

The document continues with explaining the potential of using Linked EO and non-EO data to bring advanced features to the EOPEN Pilot Use Cases before concluding and providing a way forward.

The mapping rules used to convert EO products metadata into Linked Data are provided in Appendix.

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 776019

History

Version	Date	Reason	Revised by	Approved By
0.1	07/07/2020	Initial Draft - TOC	Georgios Meditskos Maria Rousi Ilias Gialampoukidis (CERTH)	
0.10	13/08/2020	Internal Review (partial document)	Gabriella Scarpino (Serco)	
0.14	18/08/2020	Internal Review	Hakim Boulahya (SpaceApps)	
0.15	20/08/2020	Internal Review	Stelios Andreadis (CERTH)	Ilias Gialampoukidis (CERTH)
1.0	21/08/2020	Updated document after review Ready for submission		Leslie Gale (SpaceApps)

Author list

Organization	Name	Contact Information
SpaceApps	Bernard Valentin	bernard.valentin@spaceapplications.com

Executive Summary

In this deliverable we are describing the work performed in the EOPEN project for providing Linked Open Earth Observation Data (T5.2) to application developers and users as well as the tools for generating new Linked EO Data as necessary.

The Linked Data related concepts and techniques have already been introduced in D5.1 and their application to non-EO data in the context of the EOPEN Pilot Use Cases described in D5.2. The present deliverable focuses on the conversion, storage and use (including querying and visualisation) of EO products metadata that are openly and freely available.

The document shortly introduces Linked Data concepts and standards and refers to D5.1 and D5.2 when applicable. The software tools used in T5.2 are described. These in particular have permitted to generate, store, retrieve and visualise Linked EO Data.

The work accomplished in T5.2 and described in this document includes the following:

- The open Earth Observation data used in the EOPEN Pilot Use Cases has been identified. This consists in the Sentinel records harvested and served through a service API by the "Umbrella Application of Sentinel Hub" (KR10, see D3.1 "EO Data Acquisition from the Collaborative Ground Segment and Quality Control").
- Services required to convert and store the Linked Data have been deployed in the backend of the EOPEN Platform.
- Processes and workflows have been implemented and configured to generate, store, and provide a search interface to the Linked Data.
- Graphical components have been implemented to allow searching for and visualising Linked Data in the EOPEN User Portal.

As the result of this work has not been formally integrated in the EOPEN PUCs, the report provides ideas on how the Linked EO Data, combined with Linked non-EO Data (D5.2) can potentially be used in the use cases.

The mapping rules used to convert the EO products metadata obtained from the Umbrella Application to Linked Data are provided in Appendix A.

Abbreviations and Acronyms

AOI	Area Of Interest
API	Application Programming Interface
BBOX	Bounding Box
CLI	Command-Line Interface
CSV	Comma Separated Values
DBMS	DataBase Management System
EO	Earth Observation
GDB	Graph Database
GML	Geography Markup Language
GRD	Ground Range Detected
GUI	Graphical User Interface
IW	Interferometric Wide (SAR swath mode)
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
KB	Knowledge Base
KML	Keyhole Markup Language
KMZ	Keyhole Markup Language (Compressed)
LOD	Linked Open Data
NKUA	National and Kapodistrian University of Athens
O&M	Observation and Measurements
OGC	Open Geospatial Consortium
OPT	Optical
OWL	Web Ontology Language
R2RML	RDB to RDF Mapping Language
RDF	Resource Description Framework
REST	Representational state transfer
RML	RDF Mapping Language
SAR	Synthetic Aperture Radar
SPARQL	SPARQL Protocol and RDF Query Language
TOI	Time Of Interest
TTL	Turtle
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WKT	Well-Known Text
XML	eXtensible Markup Language
YAML	Yet Another Markup Language

Table of Contents

List of Figures and Tables

1 INTRODUCTION

As described in D5.2, the overall scope of WP5 is "*creating the appropriate semantic knowledge structures to map the information coming from different components of the EOPEN project.*" D5.2 lists the non-EO data sources that are used to constantly update the Knowledge Base integrated in the EOPEN Platform.

This document extends this list of data sources in adding EO data coming from the Umbrella Application of Sentinel Hub (KR10).

The architecture used to search for EO products metadata, convert these metadata into Linked Data, store this Linked Data into a Knowledge Base, where it becomes available for semantic querying, reasoning and visualisation is depicted in Figure 1.

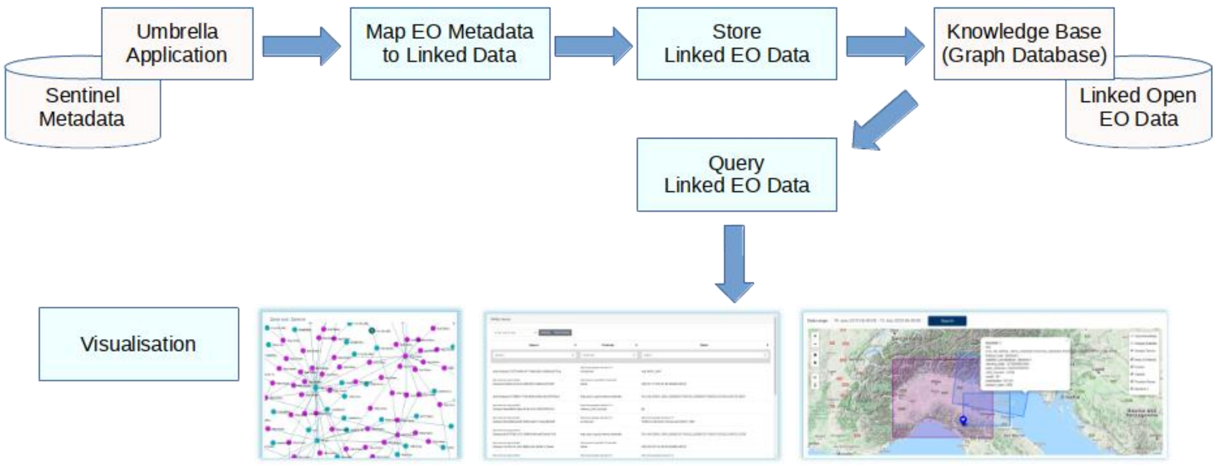


Figure 1 – Linked Open EO Data Components Architecture (T5.2)

The architecture components are as follows:

- The EO products metadata source is the Umbrella Application, as described in section 4.1.
- The transformation of the EO products metadata into Linked Data is performed using GeoTriples (section 3.2.5), configured with mapping rules expressed in RML (section 4.4.1.2). This is implemented in a generic, re-usable workflow process (section 4.3.1). The implemented mapping rules are provided in Appendix A.
- The generated Linked Data is stored in Strabon, a free and open-source graph database implemented by NKUA (section 3.2.6). Two dedicated processes have been implemented for storing and querying Linked Data in Strabon (sections 4.3.2 and 4.3.3).
- Other processes have been implemented which allows querying non-EO Linked Data from GraphDB and merge two or more Linked Data graphs (sections 4.3.4 and 4.3.6).
- Retrieved Linked Data graphs or search results may be visualised in the EOPEN User Portal using the appropriate graphical components (section 4.5).

In addition, the processes mentioned above have been integrated into workflows to provide two applications that may be invoked on-demand. The first application is used to extract EO data from the Umbrella Application service, transform it into Linked Data, and store the

Linked Data into Strabon (section 4.4.1). The second application searches for Linked EO Data in Strabon and Linked (non-EO) Data in GraphDB (see D5.2), merges the outputs and provides the merged results to graphical components for visualisation (section 4.4.2).

The work accomplished and reported in this deliverable covers the Research Activity 4.2 (RA4.2) "Linking of open EO data" described in the Grant Agreement.

2 RELEVANT REQUIREMENTS

All three releases of Deliverable 6.1 describe the technical specifications, including the System and the Application Requirements, as stemming from D2.1 and D2.2, and foreseen in the GA, in addition to the Architectural Design of the EOPEN Platform. Some of the technical specifications apply to the way Linked Data must be supported in the EOPEN Platform.

The needs and requirements that apply to Linked Data have been reproduced here. As can be seen, no requirements explicitly apply to Linked Earth Observation Data. The work performed in WP Task 5.2 has thus mainly been driven by more generic requirements.

D6.1 organises the requirements for the EOPEN Platform extensions into logical modules. One module specifically applies to Linked Data:

EOPEN must include a module capable of ingesting and managing linked data

This module named "Knowledge / Linked Data Management" implements Key Results KR08 "EOPEN ontology and reasoning support framework", covered by D 5.1 and D5.2, and KR09 "Linking Open EO data module", covered by this document.

The module has the following technical requirements:

Ref.	Statement
1.1	The module must be able to access the semantic knowledge structures and vocabularies needed to capture information and data pertinent to EOPEN
1.2	The module must be able to access to a triple store (KB)
1.3	The module must be able to populate the KB with analysis results
1.4	The module must be able to enable general purpose rule-based reasoning
1.5 (not EO)	The module must be able to determine the location of social media content based on text analysis results
1.6	The module must be able to process incoming queries and semantically retrieve data from the KB
1.7	The module must be able to provide the necessary Web interfaces (Web APIs) for other modules to invoke specific functions

Except requirement 5, all the statements can apply to Earth Observation data.

Next to the above, the Visualisation module (KR14) also includes an applicable requirement:

Ref.	Statement
2.1	The EOPEN Platform WebUI must be able to display linked data in a structured manner

In the course of the EOPEN project, processes and services have been implemented and deployed. Test workflows have been configured and executed to demonstrate the potential uses of the technology.

3 LINKED OPEN EO DATA

3.1 Introduction

The general motivation for using Linked Data is to facilitate the discovery and retrieval of heterogeneous data sets distributed in different locations, possibly maintained by different providers. Linked Data is generic enough to permit encoding information of different sources and nature, including the metadata of EO products. Linked Open EO Data focuses exclusively on the encoding of the metadata of EO products that are publicly and freely available.

The adoption of common structures and concepts, organised in ontologies, allows querying several data sources and merging the results in a seamless manner. The results from the different sources are automatically linked through the shared concepts.

Moreover, the merged results may be further analysed to discover new links between the elements retrieved from different sources. As an example, an "event" entity associated to a geographical location and an occurrence time may be linked to the satellite images that cover the same location at (almost) the same time.

Interlinked data can be navigated through by following the links for discovering the information incrementally in an intuitive manner. Graph structures may also be analysed as a whole, for example to discover clusters and isolated nodes or group of nodes. Complex queries can be executed to filter on specific link and node types and property values.

The Resource Description Framework (RDF), which is used to organise the Linked Data, encodes the nodes and the (directed) links as triples, where the source is referred to as the Subject, the link is the Predicate, and the destination the Object. Collections of triples are stored in so-called triplestores, or graph databases (GDB). And the SPARQL query language allows expressing search, extraction, modification, and deletion requests supported by these databases.

When serialized in a file, an RDF graph is encoded in one of the several formats recommended by the W3C, including RDF/XML, RDF/JSON, N3, Turtle, and N-Triples.

Section 3.2 presents shortly the Linked Data tools and technologies as most of these have already been introduced with more details in D5.1 or D5.2.

Section 3.3 presents the ontologies available for organising EO products metadata and indicates which options have been selected to add support for Linked open EO Data in the EOPEN Platform.

3.2 Tools and Technologies

3.2.1 Linked Data and RDF Graphs

Linked Data is a method for publishing structured data in a manner that element of information may be typed and interconnected. The structure and the element types should be created according to links and concepts specified in re-usable public ontologies. Different graphs created independently but complying with the same ontologies and controlled vocabularies may be connected and queried.

Resource Description Framework (RDF) is a framework for representing information on the Web. It is described in a series of W3C specifications and recommendations. RDF is the technology behind Linked Data.

The latest version, RDF 1.1, has been adopted by the W3C in 2014.

RDF was considered as a complex standard mostly due to its XML-based serialisation RDF/XML. More serialisation formats, easier to read and author have been defined, such as Turtle and N-triples.

The syntaxes that have been defined to serialize RDF data include the following:

- RDF/XML: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/
- RDF/JSON: http://www.w3.org/TR/2013/NOTE-rdf-json-20131107/
- Turtle: http://www.w3.org/TR/2014/REC-turtle-20140225/
- N-Triples: http://www.w3.org/TR/2014/REC-n-triples-20140225/

Read more about the RDF 1.1 recommendations in the official W3C "Primer" document: https://www.w3.org/TR/rdf11-primer/.

Available serialisation formats are used to import and export Linked Data into and from graph databases. As there is a clear separation between the RDF framework, the data model and its serialisation formats, the format may usually be selected by the developer depending on project specific constraints or personal preferences.

3.2.2 Graph Databases

Graph Databases (GDB) are databases which use graph structures to organise the data and allow applying semantic queries over the edges, nodes and their properties. Nodes can be seen as rows in a table in a Relational Database Management System (RDBMS) and edges as directed or not directed relations between these tables. Representing data in graphs allows, by applying a dedicated query language, to retrieve all the edges, nodes and their properties matching search criteria and obtaining sub-graphs or result lists.

Deliverable D5.2 provides a more detailed introduction to Graph Databases as well as a comparison with Relational Databases. It also reports that the main disadvantages of using graph databases are associated with the lack of security and maturity level, which is not as high with relational databases. Their main advantage is their flexibility.

3.2.3 SPARQL and GeoSPARQL

SPARQL [5] is a declarative language recommended by the W3C for extracting and updating information in RDF graphs. It is an expressive language that allows the description of quite complex relations among entities. GeoSPARQL [14] is an Implementation Specification (IS) of the OGC which extends the SPARQL language with the geospatial dimension.

SPARQL may be used to insert, search, update and delete data in a graph database.

When searching for data, it is important to distinguish between two options:

- SPARQL SELECT queries allow, similarly to traditional RDBMS SQL SELECT statements, to search for matching entries and obtain data organised in a flat table. The SELECT statement allows specifying the filter criteria as well as identifying the values to include in the responses. Each set of values can thus be compared to a record in a database table. It is only possible to re-build a graph using the result of a SPARQL SELECT query if the selected values match exactly the subject, predicate and object of triples existing in the database. But this is a particular case and it is not how sub-graphs should be extracted from a triplestore. SPARQL CONSTRUCT queries must be used for that. Outputs of SPARQL SELECT queries are suitable for visualisation in a tabular form (see section 4.5.3)

- SPARQL CONSTRUCT queries allow extracting sub-graphs from a graph database. As the result is a graph itself, it can be imported in another database, merged and interlinked with other RDF graphs, and further queried with other SPARQL statements. Outputs may be visualised as interactive networks of nodes and links, or graph browsers, for example (see section 4.5.1).

  Using CONSTRUCT queries, it is also possible to define SPARQL rules that can create new RDF data, combining existing RDF graphs into larger ones. Such rules are defined in terms of a CONSTRUCT and a WHERE clause: the former defines the graph patterns, i.e. the set of triple patterns that should be added to the underlying RDF graph upon the successful pattern matching of the graphs in the WHERE clause.

GeoSPARQL adds support for a spatial dimension. It provides the means to search for graph entries by applying functions and filters on their geometry (e.g. an image footprint, an event location). Dedicated functions allow applying transformations to geometries such as extending or shrinking an area with a buffer distance, calculating the distance between two geometries, calculating the union, intersection or difference between two geometries. Filters allow verifying, for example, if two geometries intersect, overlap, or are disjoint.

Table 1 shows an example GeoSPARQL query that searches for RDF graph entries of type "dcat:Dataset", whose geometry overlaps the polygon "POLYGON((22 40, 23 40, 23 41, 22 41, 22 40))" (expressed in WKT), and having start/end time properties on "2020-01-25":

Table 1 – Example GeoSPARQL SELECT Query

```
PREFIX dcat:  <http://www.w3.org/ns/dcat#>
PREFIX ical:  <http://www.w3.org/2002/12/cal/ical#>
PREFIX dct:   <http://purl.org/dc/terms/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX geo:   <http://www.opengis.net/ont/geosparql#>
PREFIX geof:  <http://www.opengis.net/def/function/geosparql/>
PREFIX eop:   <http://www.opengis.net/eop/2.1#>
PREFIX skos:  <http://www.w3.org/2004/02/skos/core#>
PREFIX epsg:  <http://www.opengis.net/def/crs/EPSG/0/>

SELECT DISTINCT ?Id ?Start ?End ?Geometry ?Direction
                ?Orbit_Number ?Polarization ?Mission
WHERE {
    ?s a dcat:Dataset ;
       ical:dtstart ?Start ;
       ical:dtend ?End ;
       dct:spatial ?Geometry ;
       dct:identifier ?Id ;
       eop:pass_direction ?Direction ;
       eop:orbit_number ?Orbit_Number ;
       eop:polarization ?Polarization ;
       eop:platform ?Platform .
    ?Platform skos:prefLabel ?Mission .

    FILTER (geof:sfOverlaps(?Geometry,
       "POLYGON((22 40, 23 40, 23 41, 22 41, 22 40));epsg:4326"^^geo:WKT))
    FILTER (?End > "2020-01-25T00:00:00+01:00"^^xsd:dateTime)
    FILTER (?Start < "2020-01-25T23:59:59+01:00"^^xsd:dateTime)
}
```

Graph entries that have a spatial dimension may be visualised on interactive maps, as shown in section 4.5.2.

3.2.4 R2RML and RML

The "RDB to RDF Mapping Language" (R2RML) is a recommendation from the W3C for a language that allows expressing mappings from relational databases to RDF graphs (https://www.w3.org/TR/r2rml/). The "RDF Mapping Language" (RML) is a draft specification from the W3C for expressing mapping from heterogeneous data structures (incl. XML, JSON and CSV) to the RDF data model (https://rml.io/specs/rml/).

GeoTriples, the tools used in EOPEN Task 5.2, supports both R2RML and RML. Because the EO products metadata to be transformed is encoded in the EOPEN Platform in JSON/GeoJSON, RML has been used to formalise the rules for generating the Linked open EO Data. The transformation using GeoTriples and RML rules is detailed in section 4.4.1.2.

3.2.5 GeoTriples

GeoTriples (http://geotriples.di.uoa.gr/) is a free open-source software tool developed by the National and Kapodistrian University of Athens (NKUA). This tool allows transforming geospatial data from their original formats into RDF. Supported input formats include spatially-enabled relational databases (PostGIS and MonetDB), ESRI shapefiles and XML, GML, KML, JSON, GeoJSON and CSV documents. The transformations are controlled by mapping rules expressed in R2RML or RML (see section 3.2.4).

GeoTriples has two main components:

- A mapping generator that is capable of inspecting a geospatial data source to generate mapping rules that may be used to transfer data from the source to an RDF graph which uses the GeoSPARQL vocabulary.
- A mapping processor that applies mapping rules to geospatial data sources to transform the data into an RDF graph. The processor is capable of generating RDF graphs serialised in N3 (default), RDF/XML, and Turtle.

Figure 2 represents the high-level architecture of GeoTriples.

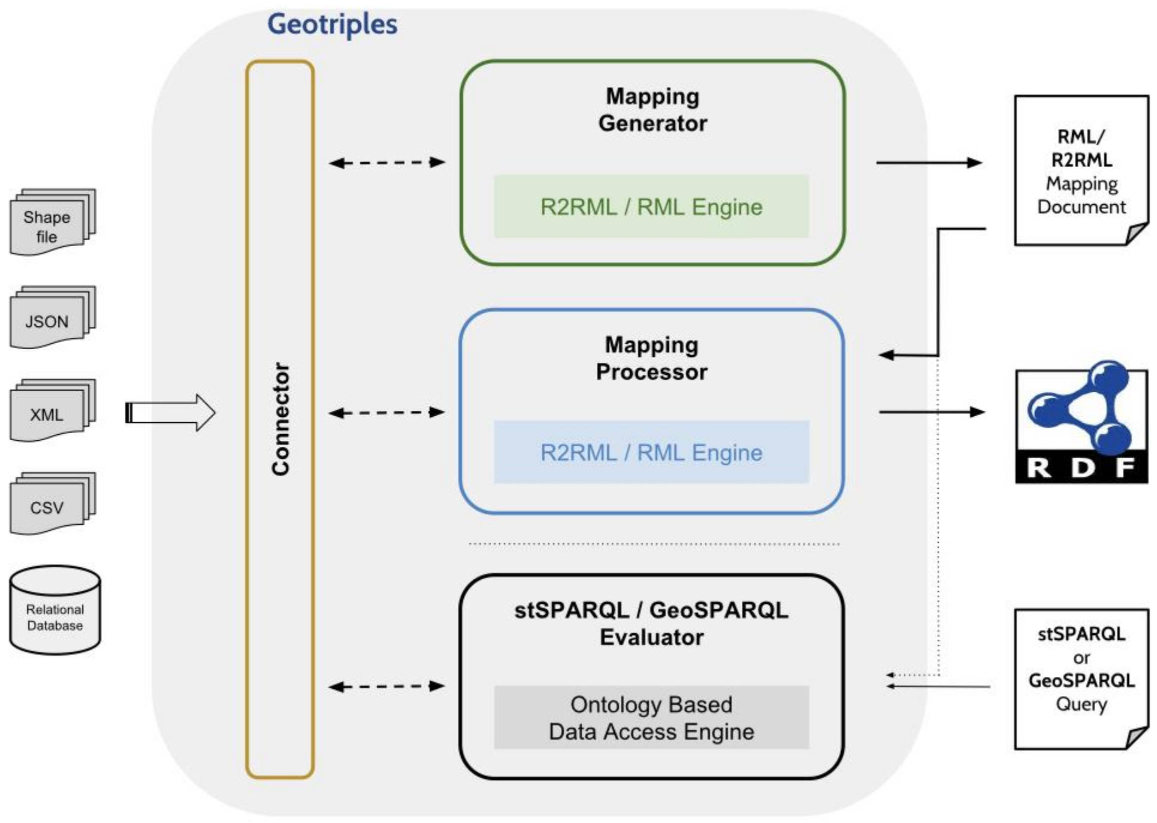


Figure 2 – The Architecture of GeoTriples

GeoTriples has been used in EOPEN Task 5.2 to transform EO products metadata formatted in JSON into RDF graphs encoded in N3. Table 2 provides the instructions for using GeoTriples on the command-line. In the EOPEN Platform, the tool has been integrated in a generic process that may be used to convert any supported geospatial data source into RDF Linked Data (see "Any to RDF Converter" in section 4.3.1), as demonstrated in the workflows described in section 4.4.1.

Table 2 – GeoTriples Command-Line Instructions (Fragments)

```
._____________________________________________________________.
|\.________________________________________________________./|
|\|                       GeoTriples                         |/|
|\|  a tool for transforming EO/geospatial data into RDF  |/|
 \.________________________________________________________./

Usage: geotriples-cmd [mode] [options] <source>|[mapping]
        Modes:
        generate_mapping -- Generate a mapping file for a:
                                1. Relational Database (MonetDB, Postgresql, Mysql etc.)
                                   Support for geometries only with MonetDB and Postgresql
                                2. Shapefile
                                3. XML (GML, KML etc), accompanied with an XSD schema definition

        dump_rdf         -- Convert the input to an RDF graph, using a mapping file. Input can be
                                1. Relational Database (MonetDB, Postgresql, Mysql etc.)
                                2. Shapefile
                                3. XML (GML, KML, etc)
                                4. JSON (GeoJSON)
                                5. CSV

        The last argument must be the path to the mapping file

Information
-----------
A. Usage for generate_mapping: geotriples-cmd generate_mapping [options] <source>

[...]

B. Usage for dump_rdf: geotriples-cmd dump_rdf [options] mapping
        mapping          mappingfile (R2RML or RML using -rml)

        Options:
```

```
        Database options:
                -u username             Database Username (Optional)
                -p password             Database Password (Optional)
                -jdbc JDBC URL          The JDBC URL of the input database
        Shapefile options:
                -sh source file         Input Source Shapefile
        XML/GML/KML options:
                -i inputfile            Input XML/GML/KML file if used with the -readyeop or -readykml arguments
                -readyeop               Use the embended EOP mapping of GeoTriples.
                -readykml               Use the embended KML mapping of GeoTriples.
        General options:
                -o outfile              Output file name (default: stdout)
                -b base IRI             e.g., http://data.linkedeodata.eu/talking-fields
                -f format               Can be N3, RDF/XML, TURTLE (default: N3)
                -rml                    Use RML processor (Default is R2RML processor)
        Examples:
        Shapefile (R2RML):      geotriples-cmd dump_rdf -o out.nt -b http://example.com -sh shapefile.shp mapping.ttl
        Shapefile   (RML):      geotriples-cmd dump_rdf -o out.ttl -f TURTLE -b http://example.com mapping.ttl
        XML         (RML):      geotriples-cmd dump_rdf -o out.nt -b http://example.com mapping.ttl
        JSON        (RML):      geotriples-cmd dump_rdf -o out.xml -f RDFXML -b http://example.com mapping.ttl
        Database  (R2RML):      geotriples-cmd dump_rdf -o out.nt -b http://example.com -u dbuser -p dbpassword -jdbc
jdbc:postgresql://localhost:5432/dbname mapping.ttl
```

3.2.6 Strabon

Strabon (http://www.strabon.di.uoa.gr/) is a free open-source RDF triplestore developed by the National and Kapodistrian University of Athens (NKUA). The implementation of Strabon started in EC project Teleios (http://www.earthobservatory.eu/Strabon) and has continued in follow EC projects such as LEO and MELODIES.

Due to its built-in support for spatial and temporal dimensions, it is well suited for storing geospatial data that change over time. It supports two extensions of SPARQL: stSPARQL, which accesses data in stRDF form and GeoSPARQL, which queries geospatial data. It is mainly used to represent temporal dimensions among the data by using events or facts that change over time as it also offers some temporal functions. It extends the Sesame RDF triplestore by supporting and handling thematic, spatial and temporal data.

Strabon has been used to store and manage the Linked open EO Data created in EOPEN Task 5.2.

3.2.7 GraphDB

GraphDB (https://www.ontotext.com/products/graphdb/) is an RDF graph database (triplestore) developed by Ontotext. Its last major version, GraphDB 9.0 is available in three flavours: the Enterprise, the Standard and the Free editions.

GraphDB supports the relevant standards including SPARQL 1.1 and GeoSPARQL.

GraphDB has been used in the context of EOPEN Tasks T5.1 and T5.3 to add support for non-EO Linked Data and semantic reasoning into the EOPEN Platform, as reported in D5.2. In this context, GraphDB is used to stored information about collected tweets, detected events, and identified flooded areas. The demonstration workflow described in section 4.4.2 uses the "Query GraphDB" process (section 4.3.4) to obtain non-EO data from GraphDB and merges them with EO data obtained from Strabon before generating a consolidated RDF graph.

3.3 Applicable Ontologies

The objective of EOPEN Task 5.2 is to provide the tools and the methodologies for generating and using the metadata of raw EO products as Linked Data (RDF graphs). It was not the intention to convert or re-encode the data itself (e.g. raster images), nor to support specific application domains such as floods, disaster risk management, etc. The latter is partially covered by the other WP5 tasks, as it is reported in D5.2.

The good practice when a new Linked Data model must be defined is to integrate in the model vocabulary from pre-existing ontologies whenever it is possible. There is no point to create new concepts for representing elements of information that already exist in other models. This applies for example, to phenomena start and end time, document identifier, title and description, geographical locations, media name, type and size, etc.

The data model that has been targeted in this work is the one described in the OGC Discussion Paper 16-074r0 "EO Metadata Discovery using Linked Data" [2]. This document specifies how EO collection and product properties may be encoded as Linked Data.

The discussion paper describes how each individual element of information available in the OGC Observation & Measurement (O&M) data model should be mapped to an RDF type, predicate or property. EO-specific properties are mapped using concepts from a new ontology (with URI "`http://www.opengis.net/eop/2.1#`") whereas non-EO specific properties are mapped using concepts available in other pre-existing popular ontologies.

For example:

- The Dublin Core Terms ontology (https://www.dublincore.org/specifications/dublin-core/dcmi-terms/) is used to map product identifier, title and abstract information.
- The Observation & Measurement (https://lov.linkeddata.es/dataset/lov/vocabs/oml) is used to encode information defined in ISO 19109 Geographic Information – Rules of Application Schema.
- The iCalendar ontology (https://www.w3.org/2002/12/cal/ical#) is used to encode temporal information (such as start and end date and time).
- The Data Catalog Vocabulary (DCAT, https://www.w3.org/TR/vocab-dcat-2/) is used to encode the information that is also present in data catalogues.
- The Media RSS of Yahoo (https://www.feedforall.com/mediarss.htm) is used to encode the information about the media content (e.g. image format and type).

EO product metadata include information about the data itself but should also indicate in which context it has been obtained (e.g. during a satellite mission such as Sentinel-1), which tools have been used to capture the data (e.g. Sentinel-1 platform 1A with a given sensor), and in which manner the data has been processed (e.g. Level-1a, Level-1b processors identification).

When platform or sensor specific information is included, it is needed to map the related properties to platform or sensor specific vocabulary. For this reason, the data model proposed in the OGC Discussion Paper integrates in the model vocabularies that are specific to optical missions and SAR missions, for example.

The rules implemented to transform the EO products metadata into Linked Data comply with the mapping proposed in the document. This is described in details in section 4.4.1.2.

Figure 3, below, shows a graphical representation of the RDF graph generated with the properties that are available (within the EOPEN Platform) about a given EO product.

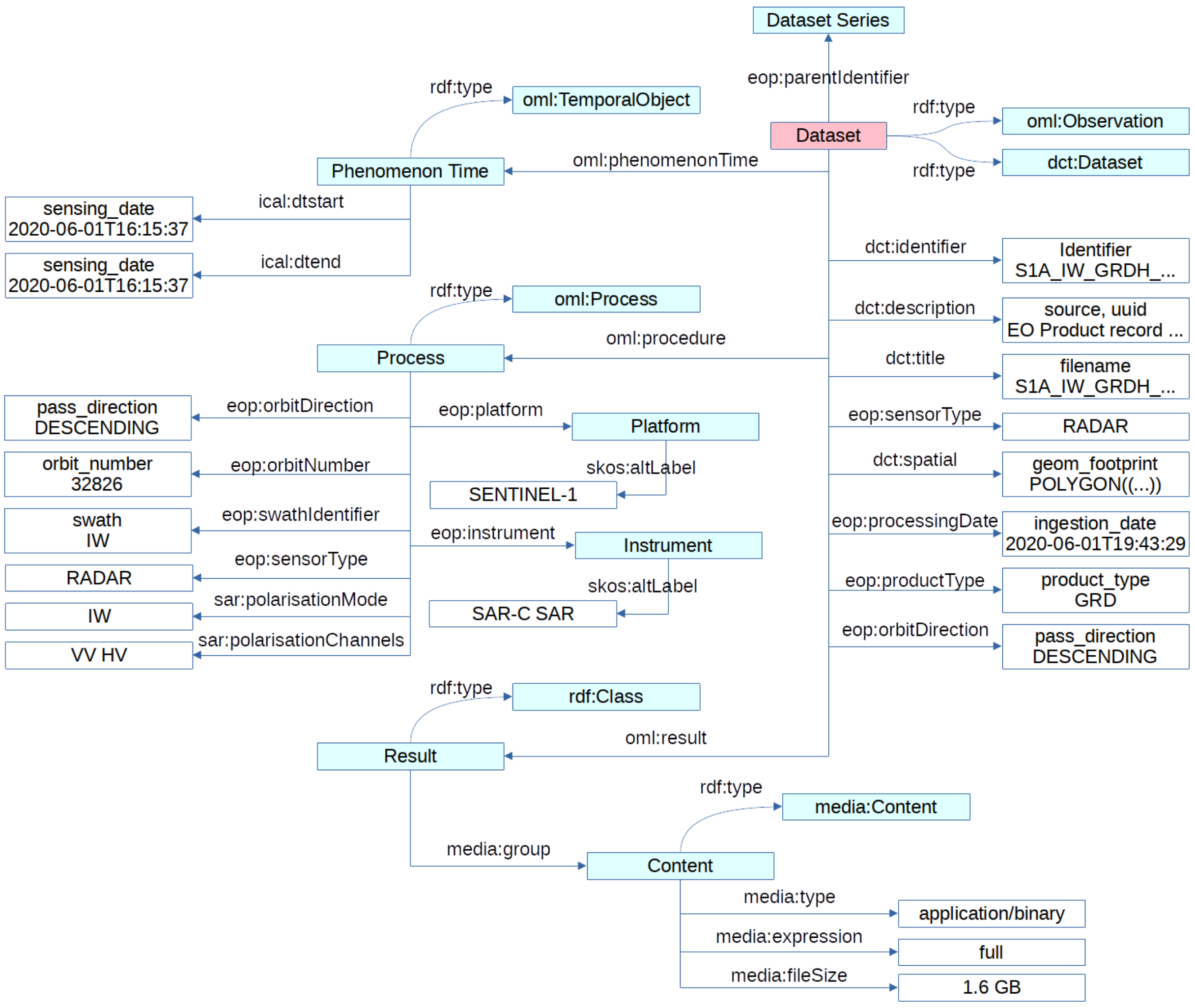


Figure 3 – EO Product (Dataset) Linked Data Model

It must be noted that a data model can always be extended with new links and properties, when appropriate. Also, should different models be of interest for representing the same information, nothing prevents generating the two (or more) alternatives and storing the resulting RDF graph in a graph database.

4 LINKED OPEN EO DATA IN THE EOPEN PLATFORM

This chapter starts with the description of the Earth Observation data (EO products) available in the EOPEN Platform and used by the three Pilot Use Cases and with the manner the EO products metadata are obtained.

The chapter continues with the list of services, processes and workflows that have been implemented and deployed in order to demonstrate the transformation of EO products metadata into Linked Data, the storage of this data in a triple store, and its exploitation using search and filter queries.

The last section introduces the different means that are available for visualising Linked open EO Data (applicable also to any Linked Data) in the EOPEN User Portal.

4.1 Available Earth Observation Data

Different types of Earth Observation data may be considered, usually organised in three categories: in-situ, air borne and space borne. In-situ data refers to the data collected on the ground or with a physical contact with the ground. These are typically collected by sensor installed at static or mobile stations (meteorological stations, controlled and autonomous rovers, etc.) In-situ data is often used as ground truth data as it directly records phenomena, unlike the air and space borne data that are altered by the air and the atmosphere that is present between the phenomena and the sensors.

Air borne data is recorded by sensors mounted in manned and unmanned flying vehicles such as airplanes, helicopters and drones (UAVs). Space borne data are collected by sensors located in space, such as on a satellite or a space station.

The data referred to in WP Task 5.2 concerns the space borne data and as the objective is to provide it to the Pilot Use Cases, the focus has been put on the EO open data harvested by the "Umbrella Application of Sentinel Hub", result of the work on KR10.

The Umbrella Application facilitates the search and discovery of the Sentinel products available in a series of Sentinel hubs by providing a unique API for obtaining the metadata of the available products. The EOPEN platform runs workflows at regular interval for keeping this centralised catalogue up-to-date. In particular, workflows are used to harvest the new products metadata, remove the products that are not directly available anymore (e.g. after they have been moved to a long term archive by the originating hubs), and measure the speed of data transfer for rating the hubs based on their download speed.

The Umbrella Application runs as a service within the EOPEN Platform. Details about the Umbrella Application may be found in EOPEN deliverable D3.1 "EO Data Acquisition from the Collaborative Ground Segment and Quality Control".

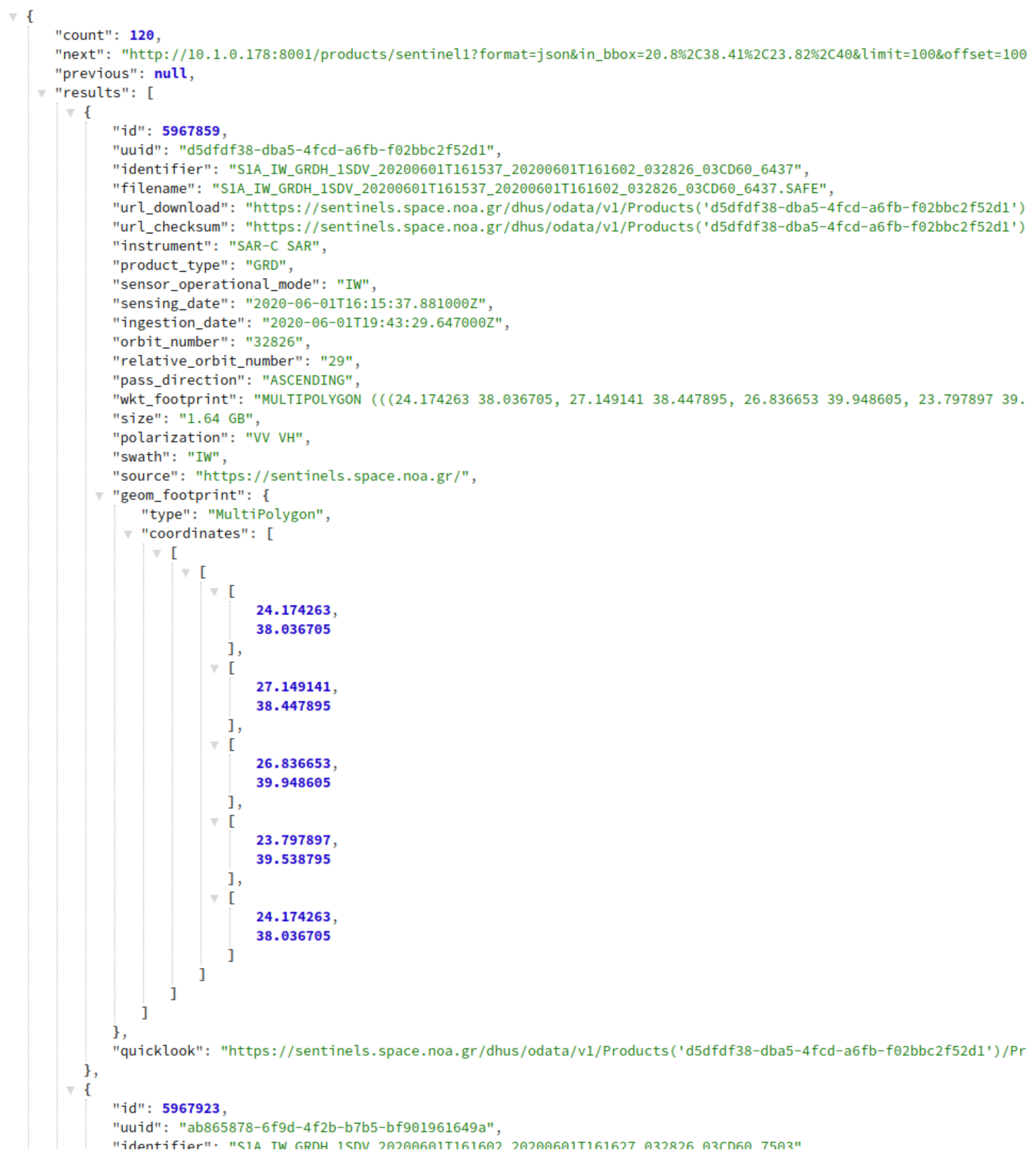
The Umbrella Application provides a single point of access to data from all available Sentinel missions: Sentinel-1, Sentinel-2, Sentinel-3 and Sentinel-5p.

Table 3 - Example Umbrella Application query and response

Query Parameters and URL
Parameters: • End-point: http://<ip-address>:<port>/products • Mission: sentinel1 • Product type: GRD • Bounding box: 20.8, 38.41, 23.82, 40 • Date range: 2020-01-01 / 2020-06-30 (both dates included) • Output format: JSON Resulting URL: http://<ip-address>:<port>/products/sentinel1?format=json &in_bbox=20.8,38.41,23.82,40&limit=100&offset=0&product_type=GRD &sensing_date__gte=2020-06-01&sensing_date__lte=2020-06-30

Response (JSON fragment)

The response document indicates that 120 images match the search criteria. As requested, the document only contains the first 100 entries. Links to navigate to the previous and next 100 records are provided as well (here, the "`previous`" link is `null` as we are in the first page):



```
{
    "count": 120,
    "next": "http://10.1.0.178:8001/products/sentinel1?format=json&in_bbox=20.8%2C38.41%2C23.82%2C40&limit=100&offset=100
    "previous": null,
    "results": [
        {
            "id": 5967859,
            "uuid": "d5dfdf38-dba5-4fcd-a6fb-f02bbc2f52d1",
            "identifier": "S1A_IW_GRDH_1SDV_20200601T161537_20200601T161602_032826_03CD60_6437",
            "filename": "S1A_IW_GRDH_1SDV_20200601T161537_20200601T161602_032826_03CD60_6437.SAFE",
            "url_download": "https://sentinels.space.noa.gr/dhus/odata/v1/Products('d5dfdf38-dba5-4fcd-a6fb-f02bbc2f52d1')
            "url_checksum": "https://sentinels.space.noa.gr/dhus/odata/v1/Products('d5dfdf38-dba5-4fcd-a6fb-f02bbc2f52d1')
            "instrument": "SAR-C SAR",
            "product_type": "GRD",
            "sensor_operational_mode": "IW",
            "sensing_date": "2020-06-01T16:15:37.881000Z",
            "ingestion_date": "2020-06-01T19:43:29.647000Z",
            "orbit_number": "32826",
            "relative_orbit_number": "29",
            "pass_direction": "ASCENDING",
            "wkt_footprint": "MULTIPOLYGON (((24.174263 38.036705, 27.149141 38.447895, 26.836653 39.948605, 23.797897 39.
            "size": "1.64 GB",
            "polarization": "VV VH",
            "swath": "IW",
            "source": "https://sentinels.space.noa.gr/",
            "geom_footprint": {
                "type": "MultiPolygon",
                "coordinates": [
                    [
                        [
                            [
                                24.174263,
                                38.036705
                            ],
                            [
                                27.149141,
                                38.447895
                            ],
                            [
                                26.836653,
                                39.948605
                            ],
                            [
                                23.797897,
                                39.538795
                            ],
                            [
                                24.174263,
                                38.036705
                            ]
                        ]
                    ]
                ]
            },
            "quicklook": "https://sentinels.space.noa.gr/dhus/odata/v1/Products('d5dfdf38-dba5-4fcd-a6fb-f02bbc2f52d1')/Pr
        },
        {
            "id": 5967923,
            "uuid": "ab865878-6f9d-4f2b-b7b5-bf901961649a",
            "identifier": "S1A_IW_GRDH_1SDV_20200601T161602_20200601T161627_032826_03CD60_7503"
```

This example shows that in the output generated by the Umbrella Application the geometries (product footprints) are encoded according to the GeoJSON standard but not the product properties. This prevents using the results in a generic GeoJSON viewer, for example, but has no impact in the work described in this document. The mapping rules for transforming the JSON entries into RDF must simply use the appropriate path to extract each product property.

D3.1, Appendix A describes the RESTful API for querying Sentinel data from the Umbrella Application. It details also the mission-specific filters that may be applied, for example, the polarization values for Sentinel-1 products and the cloud coverage of Sentinel-2 products.

4.2 Services

In the EOPEN Platform, services provide always running features to the processes deployed and executed as part of workflows and to visualisation components integrated in the EOPEN User Portal. The updated list of services running in the EOPEN Platform is provided in D6.5, the Final System User and Developer Guide.

Hereafter are described the services that are involved in the implementation of the Linked Data workflows described in section 4.4, below.

4.2.1 Umbrella Application of Sentinel Hubs

The Umbrella Application of Sentinel Hubs (KR10) has been shortly introduced in section 4.1, above. The design and implementation details may be found in deliverable D3.1.

This service is not part of the work on Linked Data. However, it is the catalogue used to discover the existing Sentinel products and to obtain their metadata. It is thus used by the first workflow described in section 4.4.1, below.

4.2.2 GraphDB

An instance of the GraphDB graph database (see sections 3.2.2 and 3.2.7) runs in the EOPEN Platform for providing the means to store and retrieve metadata from social media (tweets) and observations coming from the analysis of satellite images (e.g. flooded area masks). This information is collected and processed by PUC specific workflows. Before being inserted in GraphDB, the data is converted into Linked Data.

4.2.3 Strabon

An instance of the Strabon graph database (see sections 3.2.2 and 3.2.6) runs in the EOPEN Platform for providing the means to store and retrieve the metadata of EO products converted in Linked Data.

4.3 Processes

The processes are the tasks that are interconnected in a workflow and that are deployed in a worker node and remotely executed by the workflow engine integrated in the EOPEN Platform. Each process is implemented as a Dockerized service. After its only execution, a process is undeployed, freeing the execution resources for the deployment and execution of other processes.

Below are described the processes that have been implemented specifically for creating the test workflows described in section 4.4.

4.3.1 Any to RDF Converter

The "Any to RDF Converter" process allows converting structured data into RDF. As it uses NKUA's GeoTriples introduced in section 3.2.5, supported input formats include JSON, XML, CSV, and Shapefile, and it is able to serialise the resulting RDF into N3, XML and Turtle.

GeoTriples is a generic tool that is entirely controlled using mapping rules.

The process receives as inputs:

- the name of the input file,
- the name of the output file,
- the name of the file that contain the mapping rules, and
- the name of the file that defines the applicable RDF namespaces.

The mapping rules expressed in RML (see section 3.2.4) must contain the path and name of the input file. The input file itself is not to be provided as a parameter to GeoTriples. Hard-coding the input files in the RML file allows parsing and merging data stored in several input files and generating a consolidated RDF file. On the other hand, this is an inconvenience for creating a process that is potentially reusable to transform input files of various names into RDF.

To circumvent this limitation, instead of passing a fully defined RML file to the process, this may contain placeholders where the input file name must be inserted. The process takes care to substitute the placeholders with the name of the input file provided in parameter.

For example, this is an RML file fragment that contains a placeholder:

```
    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];
```

When this RML file is provided to the process together with the input file "/tmp/input.json", the RML file is updated to contain the path and name of the input file:

```
    rml:logicalSource [
        rml:source "/tmp/input.json";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];
```

As GeoTriples is implemented in Java, the Python-based process generates a command line and executes it in a sub-process.

Example generated command:

```
java -jar geotriples-master-cmd.jar dump_rdf -rml -ns namespaces.ns \
          -o output.rdf mapping.rml.ttl
```

The generated RDF file is stored in the process outputs folder if no absolute path is provided otherwise the absolute output path is used. The process outputs status string "OK" and the path and name of the output RDF file.

Figure 4 represents the "Any to RDF Converter" process as it appears in the Workflow Editor and Figure 5 shows the parameterisation form that is automatically generated to execute the process on-demand.

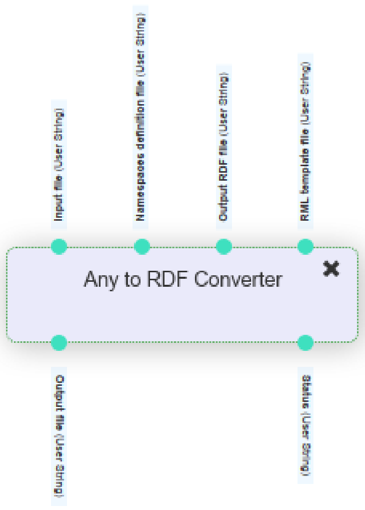
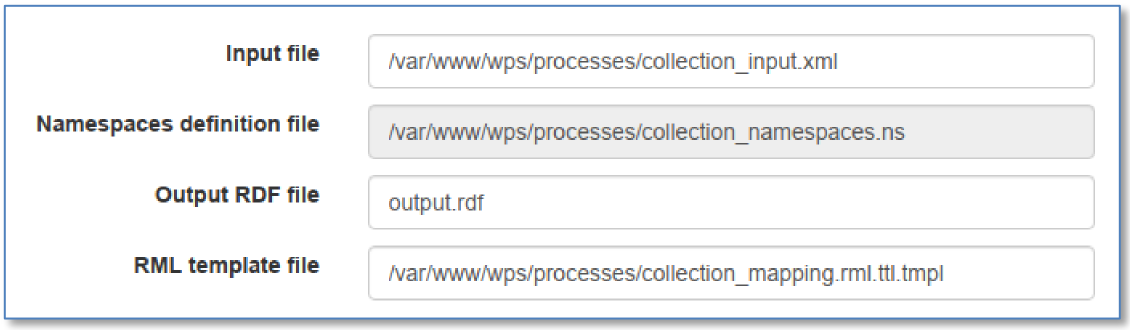

Input file (User String)
Namespaces definition file (User String)
Output RDF file (User String)
RML template file (User String)
Any to RDF Converter
Output file (User String)
Status (User String)

Figure 4 – "Any to RDF Converter" Process in the Workflow Editor



Input file	/var/www/wps/processes/collection_input.xml
Namespaces definition file	/var/www/wps/processes/collection_namespaces.ns
Output RDF file	output.rdf
RML template file	/var/www/wps/processes/collection_mapping.rml.ttl.tmpl

Figure 5 – "Any to RDF Converter" Process Parameterisation Form

4.3.2 Store in Strabon

This process is used to store data encoded in RDF into a Strabon graph database (such as the one running as a service in the EOPEN Platform).

The process requires the following input parameters:

- The URL (scheme, domain, port and path) of the Strabon endpoint.
- The path and name of the file that contains the encrypted Strabon user credentials.
- The path and name of the input RDF file
- The format of the RDF file: RDF/XML, N3, N-Triples or Turtle
- The URI of the graph in which the RDF data must be stored.

If the operation succeeds, the "Store in Strabon" process outputs the status string "OK".

Figure 6 represents a workflow that chains an "Any to RDF Converter" process with the "Store in Strabon" process, as it appears in the Workflow Editor. Figure 7 shows the parameterisation form that is automatically generated to execute this workflow.

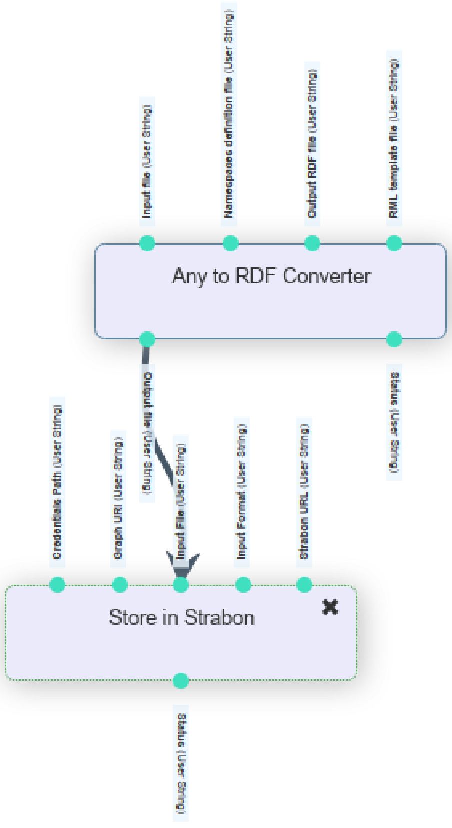
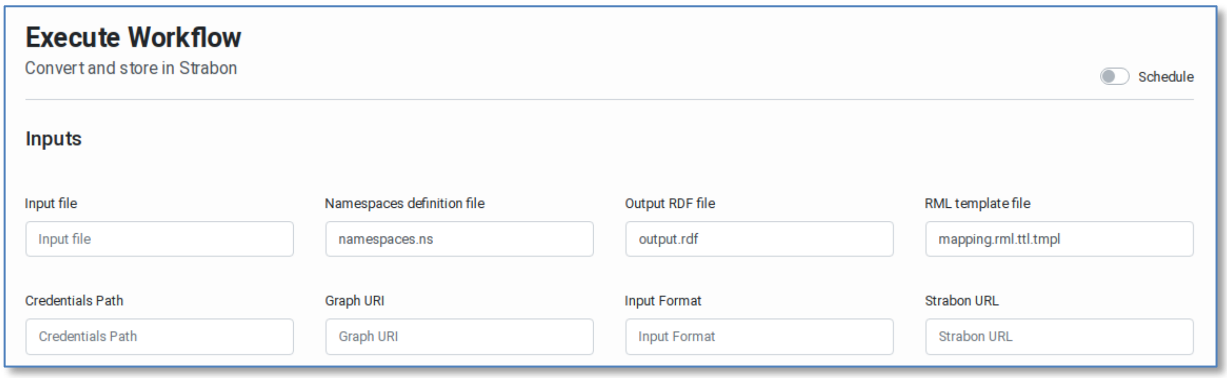

Input file (User String)
Namespaces definition file (User String)
Output RDF file (User String)
RML template file (User String)
Any to RDF Converter
Output file (User String)
Status (User String)
Credentials Path (User String)
Graph URI (User String)
Input File (User String)
Input Format (User String)
Strabon URL (User String)
Store in Strabon
Status (User String)

Figure 6 – "Convert to RDF and Store in Strabon" Workflow in Workflow Editor



Execute Workflow

Convert and store in Strabon

Schedule

Inputs

Input file	Namespaces definition file	Output RDF file	RML template file
Input file	namespaces.ns	output.rdf	mapping.rml.ttl.tmpl
Credentials Path	Graph URI	Input Format	Strabon URL
Credentials Path	Graph URI	Input Format	Strabon URL

Figure 7 – "Convert to RDF and Store in Strabon" Workflow Parameterisation Form

4.3.3 Query Strabon

This process is used to query a Strabon instance such as the Strabon service running in the EOPEN Platform using GeoSPARQL.

The process receives the following input parameters:

- The GeoSPARQL query (inline, not as a file name),
- The URL (scheme, domain, port and path) of the Strabon endpoint,
- The path and name of the output file,

- The format of the output file.

The process outputs a GeoSPARQL Results document (after a GeoSPARQL SELECT query) or an RDF graph (after a GeoSPARQL CONSTRUCT query). Depending on the nature of the query, different output formats are supported:

- SELECT outputs: SPARQL/XML, SPARQL/JSON, CSV, TSV, Binary, KML, KMZ, GeoJSON.
- CONSTRUCT outputs: RDF/XML, N-Triples, Turtle, N3, TriX, TriG, BinaryRDF.

The output file is stored in the process outputs folder if no absolute path is provided, otherwise the absolute output path is used. The process outputs status string "OK" and the path and name of the output file.

4.3.4 Query GraphDB

This process is used to query a GraphDB instance such as the GraphDB service running in the EOPEN Platform using GeoSPARQL.

The process receives the following input parameters:

- The GeoSPARQL query (inline, not as a file name),
- The URL (scheme, domain, port and path) of the GraphDB endpoint,
- The Identifier of the GraphDB repository to query,
- The path and name of the output file,
- The format of the output file.

The process outputs a GeoSPARQL Results document (after a GeoSPARQL SELECT query) or an RDF graph (after a GeoSPARQL CONSTRUCT query). Depending on the nature of the query, different output formats are supported:

- SELECT outputs: SPARQL/XML, SPARQL/JSON, BinaryRDF.
- CONSTRUCT outputs: RDF/XML, RDF/JSON, Turtle, N3.

The output file is stored in the process outputs folder if no absolute path is provided, otherwise the absolute output path is used. The process outputs status string "OK" and the path and name of the output file.

4.3.5 Query RDF File

This process is used to query an RDF graph serialised in a file using SPARQL.

The process receives the following input parameters:

- The SPARQL SELECT query (inline, not as a file name),
- The path and name of the file that contains the graph data,
- The format of the input file,
- The path and name of the output file,
- The format of the output file.

This process only supports SPARQL SELECT queries and can thus only produce SPARQL Results documents. Supported output format are XML and JSON.

The output file is stored in the process outputs folder if no absolute path is provided, otherwise the absolute output path is used. The process outputs status string "OK" and the path and name of the output file.

4.3.6 RDF Merger

This process merges two or more RDF graphs serialised in individual files into a single graph. Internally, the process uses the Python library "`rdflib`"[1] to load the individual files into a conjunctive graph and to serialise the resulting graph in an output file.

The process receives the following input parameters:

- The path and name of up to five files that contain the RDF graphs,
  The format of each input file is automatically detected.
- The path and name of the output file,
- The format of the output file.

Supported output format are: N3, NT, Turtle, RDF/XML, TriX, TriG, N-Quads and pretty XML.

The output RDF file is stored in the process outputs folder if no absolute path is provided, otherwise the absolute output path is used. The process outputs status string "OK" and the path and name of the output file.

4.4 Workflows

Two workflows have been prepared to test and demonstrate the usage of the conversion, storage, extraction and merging of Linked Data:

- The first workflow obtains Sentinel products metadata from the Umbrella Application, converts them to Linked Data and stores the result in the Strabon service.
- The second workflow uses GeoSPARQL to search for Linked Data in GraphDB and Strabon, merges the matching sub-graphs and outputs the results in a SPARQ Results document, ready for visualisation.

4.4.1 EO Products Metadata Conversion to Linked Data and Storage in Strabon

4.4.1.1 Workflow Description

This workflow has a number of unconnected input parameters. By default, unconnected inputs are used to automatically generate the parameterisation form presented to the user when the workflow is executed on-demand. To prevent displaying too many parameters to the user, input parameters may be given a default value and flagged to be hidden.

[1] https://rdflib.readthedocs.io/en/stable/index.html

The remaining input parameters that the user is invited to edit are the following:

1. Mission (e.g. "sentinel1")
2. Bounding box (AOI)
3. Date range (TOI)
4. Amount of results in the response (default: 100)
5. Offset of the results in the response (default: 0)

---

6. Namespace definition file
7. RML template file containing the mapping rules
8. Output RDF file

---

9. File format of the RDF input file
10. Strabon endpoint URL
11. File containing the encrypted Strabon user credentials
12. Target graph URI

The first five parameters are used to build the HTTP request sent to the Umbrella Application to search for matching EO products. The next three parameters configure the "any to RDF converter" and the last four parameters configure the process that stores the generated RDF file into Strabon.

Figure 9 shows the parameterisation form displayed to the users who want to manually execute the workflow. The form is pre-filled-in with the default values configured in the workflow definition via the Workflow Editor.

Figure 8 shows the workflow as edited in the Workflow Editor. The first series of tasks are necessary to build the URL for fetching the EO products metadata from the Umbrella Application.

The main steps are as follow:

1. Use input functions and template renderers to build the URL to query the Umbrella Application. The use of these built-in functions is described in the Developer Guide D6.5.
2. Query the Umbrella Application and save the response (JSON file) on disk. This is performed by a generic "file fetcher" process which issues an HTTP GET request and stores the response on disk (see D6.5 for details).
3. Convert the JSON file containing the EO products metadata into an RDF graph
4. Store the RDF graph in Strabon

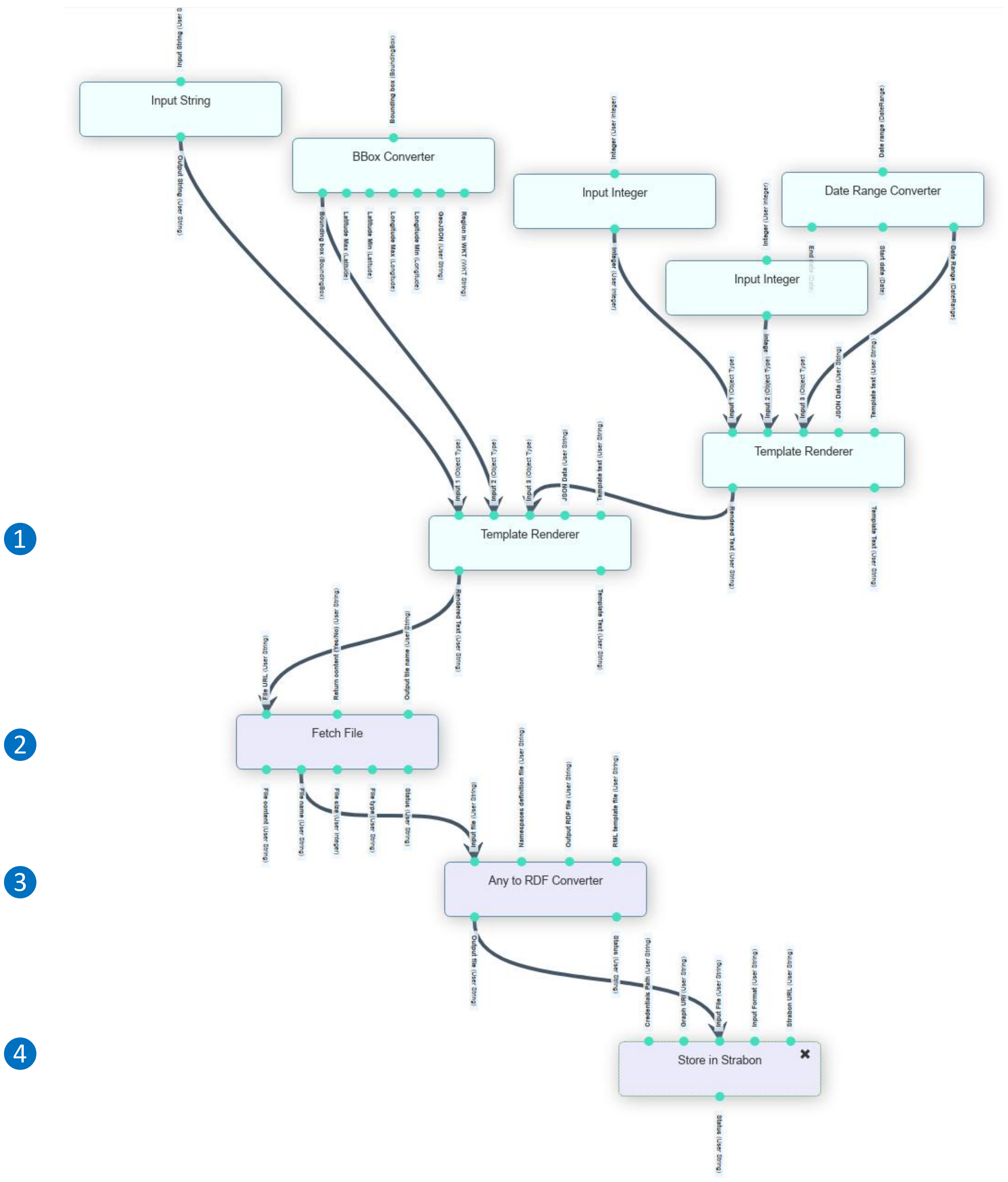
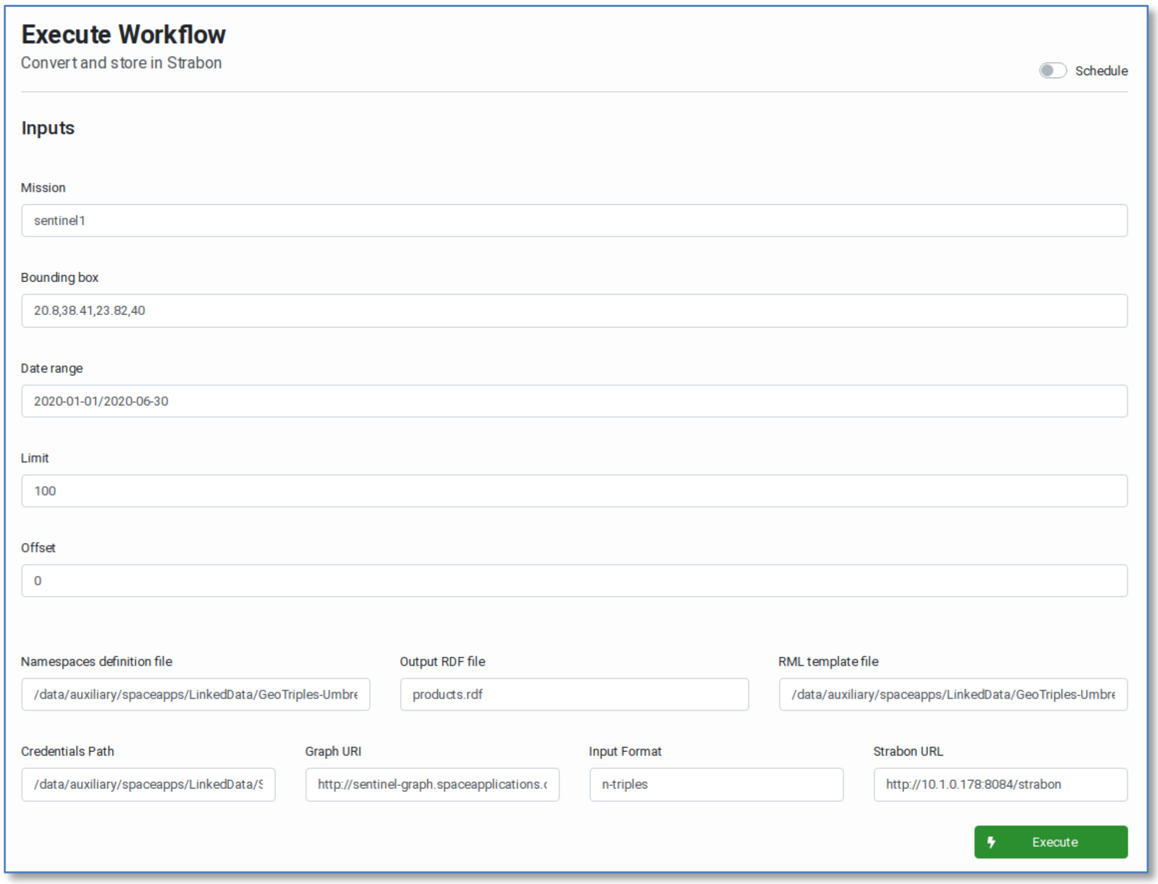


Figure 8 – Linked Data Conversion and Storage Workflow



Execute Workflow

Convert and store in Strabon

Schedule

Inputs

Mission	sentinel1
Bounding box	20.8,38.41,23.82,40
Date range	2020-01-01/2020-06-30
Limit	100
Offset	0
Namespaces definition file	/data/auxiliary/spaceapps/LinkedData/GeoTriples-Umbre
Output RDF file	products.rdf
RML template file	/data/auxiliary/spaceapps/LinkedData/GeoTriples-Umbre
Credentials Path	/data/auxiliary/spaceapps/LinkedData/S
Graph URI	http://sentinel-graph.spaceapplications.c
Input Format	n-triples
Strabon URL	http://10.1.0.178:8084/strabon

Figure 9 – Linked Data Conversion and Storage Workflow Parameterisation Form

4.4.1.2 Conversion of the products metadata

The key element in this workflow is the set of mapping rules that are used to convert the EO products metadata encoded in JSON (see section 4.1) into an RDF graph encoded in N-Triples.

The implemented mapping rules target the graph structure described in the OGC Discussion Paper 16-074r0 "EO Metadata Discovery using Linked Data" [2]. This document specifies how EO collection and product properties may be encoded as Linked Data. However, as shown in the example in Table 3, page 21, the Umbrella Application only provides a subset of these properties. The mapping is thus limited to the available properties. The following tables describe how each property is mapped to an entity in the Linked Data. In some cases, a property becomes a literal value (e.g. `identifier`, `pass_direction`, `product_type`) but it may also be used to create an object having itself properties (e.g. `url_download`). In the latter cases, special rules are used to construct the links with the appropriate subjects, predicate type and objects.

Complete examples are provided in Appendix A, page 54.

Table 4 – Namespaces and Prefixes

Namespaces and Prefixes
`@prefix rr: <http://www.w3.org/ns/r2rml#> .`
`@prefix rml: <http://semweb.mmlab.be/ns/rml#> .`
`@prefix rrx: <http://www.w3.org/ns/r2rml-ext#> .`
`@prefix rrxf: <http://www.w3.org/ns/r2rml-ext/functions/def/> .`
`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .`
`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .`
`@prefix geo: <http://www.opengis.net/ont/geosparql#> .`
`@prefix geof: <http://www.opengis.net/def/function/geosparql/> .`
`@prefix opt: <http://www.opengis.net/opt/2.1#> .`
`@prefix eop: <http://www.opengis.net/eop/2.1#> .`
`@prefix om: <http://www.opengis.net/om/2.0#> .`
`@prefix ical: <http://www.w3.org/2002/12/cal/ical#> .`
`@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .`
`@prefix oml: <http://def.seegrid.csiro.au/ontology/om/om-lite#> .`
`@prefix mrss: <http://search.yahoo.com/mrss/> .`
`@prefix ows: <http://www.opengis.net/ows/2.0#> .`
`@prefix xlink: <http://www.w3.org/1999/xlink#> .`
`@prefix dcat: <http://www.w3.org/ns/dcat#> .`
`@prefix dct: <http://purl.org/dc/terms/> .`
`@prefix skos: <http://www.w3.org/2004/02/skos/core#> .`
`@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .`
`@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .`
`@prefix dc: <http://purl.org/dc/elements/1.1/> .`
`@prefix gmd: <http://www.isotc211.org/2005/gmd> .`
`@prefix gml: <http://www.opengis.net/gml/3.2#> .`
`@prefix gcmd: <http://gcmdservices.gsfc.nasa.gov/kms/concept/> .`
`@prefix media: <http://search.yahoo.com/mrss/> .`

Table 5 – Dataset Object Metadata Mapping Rules

Properties	Mapping Rules
	`<#Observation>`
uuid	`# Specify the subject URI`
	`rr:subjectMap [`
	`    rr:template "http://www.w3.org/ns/dcat#Dataset/{uuid}"`
	`];`
Implicit subject types	`# Hard-coded triple: <subject> rdf:type dcat:Dataset`
	`rr:predicateObjectMap [`
	`    rr:predicate rdf:type;`
	`    rr:object dcat:Dataset`
	`];`
	`# Hard-coded triple: <subject> rdf:type oml:Observation`
	`rr:predicateObjectMap [`
	`    rr:predicate rdf:type;`
	`    rr:object oml:Observation`
	`];`
Implicit sensor type (for Sentinel-1)	`# Hard-coded: <subject> eop:sensorType "RADAR"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate eop:sensorType;`
	`    rr:object "RADAR"`
	`];`

source, uuid

```
    # Triple: <subbject> dct:description "{uuid}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate dct:description;
        rr:objectMap [
            rr:template "EO Product record {uuid} from the Umbrella
Application (source: {source})";
            rr:datatype xsd:string
       ]
    ];
```

identifier

```
    # Triple: <subject> dct:identifier "{identifier}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate dct:identifier;
        rr:objectMap [
            rml:reference "identifier"
       ]
    ];
```

geom_footprint

```
    # Triple:
    # <subject> dct:spatial "{geom_footprint}"^^geo:wktLiteral
    rr:predicateObjectMap [
        rr:predicate dct:spatial;
        rr:objectMap [
            rrx:function rrxf:asWKT;
            rrx:argumentMap([rml:reference "geom_footprint"]);
            rr:datatype geo:wktLiteral;
        ];
    ];
```

ingestion_date

```
    # Triple:
    # <subject> eop:processingDate "{ingestion_date}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:processingDate;
        #rr:predicate dct:issued;
        rr:objectMap [
            rml:reference "ingestion_date";
            rr:datatype xsd:dateTime
       ]
    ];
```

product_type

```
    # Triple: <subject> eop:productType "{poduct_type}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:productType;
        rr:objectMap [
            rml:reference "product_type";
            rr:datatype xsd:string
       ]
    ];
```

pass_direction

```
    # Triple: <subject> eop:orbitDirection "DESCENDING"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:orbitDirection;
        rr:objectMap [
            rml:reference "pass_direction";
            rr:datatype xsd:string
       ]
    ];
```

Link to the Phenomenon Time object (Table 6)

```
    # Triple: <subject> oml:phenomenonTime <#PhenomenonTime>
    rr:predicateObjectMap [
        rr:predicate oml:phenomenonTime;
        rr:objectMap [
            rr:parentTriplesMap <#PhenomenonTime>
        ];
    ];
```

Link to the Process object (Table 7)	`    # Triple: <subject> oml:procedure <#Process>`
	`    rr:predicateObjectMap [`
	`        rr:predicate oml:procedure;`
	`        rr:objectMap [`
	`           rr:parentTriplesMap <#Process>`
	`        ];`
	`    ];`
Link to the Result object (Table 10)	`    # Triple: <subject> oml:result <#Result>`
	`    rr:predicateObjectMap [`
	`        rr:predicate oml:result;`
	`        rr:objectMap [`
	`           rr:parentTriplesMap <#Result>`
	`        ];`
	`    ];`
	`.`

Table 6 – Dataset Phenomenon Time Mapping Rules

Properties	**Mapping Rules**
	`<#PhenomenonTime>`
source, uuid	`    # Specify the subject URI and RDF type`
	`    rr:subjectMap [`
	`        rr:template "{source}/{uuid}/phenomenonTime"`
	`    ];`
	`    rr:predicateObjectMap [`
	`        rr:predicate rdf:type;`
	`        rr:object oml:TemporalObject`
	`    ];`
sensing_date	`    # Triple: <subject> ical:dtstart "{sensing_date}"^^xsd:dateTime`
	`    rr:predicateObjectMap [`
	`        rr:predicate ical:dtstart;`
	`        rr:objectMap [`
	`            rml:reference "sensing_date";`
	`            rr:datatype xsd:dateTime`
	`        ]`
	`    ];`
sensing_date (again)	`    # Triple: <subject> ical:dtend "{sensing_date}"^^xsd:dateTime`
	`    # The Umbrella App. does not provide the sensing end timestamp.`
	`    rr:predicateObjectMap [`
	`        rr:predicate ical:dtend;`
	`        rr:objectMap [`
	`            rml:reference "sensing_date";`
	`            rr:datatype xsd:dateTime`
	`        ]`
	`    ];`
	`.`

Table 7 – Dataset Process Object (O&M EarthObservationEquipment) Mapping Rules

Properties	Mapping Rules
	`<#Process>`

source, uuid	`# Specify the subject URI and RDF type`
	`rr:subjectMap [`
	`    rr:template "{source}/{uuid}/process"`
	`];`
	
	`rr:predicateObjectMap [`
	`    rr:predicate rdf:type;`
	`    rr:object oml:Process`
	`];`

pass_direction	`# Triple:`
	`# <subject> eop:orbitDirection "{pass_direction}"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate eop:orbitDirection;`
	`    rr:objectMap [`
	`        rml:reference "pass_direction";`
	`        rr:datatype xsd:string`
	`    ]`
	`];`

swath	`# Triple: <subject> eop:swathIdentifier "{swath}"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate eop:swathIdentifier;`
	`    rr:objectMap [`
	`        rml:reference "swath";`
	`        rr:datatype xsd:string`
	`    ]`
	`];`

orbit_number	`# Triple:`
	`# <subject> eop:orbitNumber "{orbit_number}"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate eop:orbitNumber;`
	`    rr:objectMap [`
	`        rml:reference "orbit_number";`
	`        rr:datatype xsd:integer`
	`    ]`
	`];`

Implicit (Radar Products)	`# Hard-coded: <subject> eop:sensorType "RADAR"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate eop:sensorType;`
	`    rr:object "RADAR"`
	`];`

sensor_oper-ational_mode (Radar Products)	`# Triple: <subject> sar:polarisationMode "IW"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate sar:polarisationMode;`
	`    rr:objectMap [`
	`        rml:reference "sensor_operational_mode";`
	`        rr:datatype xsd:string`
	`    ]`
	`];`

polarization
(Radar Products)
	`# Triple:`
	`# <subject> sar:polarisationChannels "VV VH"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate sar:polarisationChannels;`
	`    rr:objectMap [`
	`        rml:reference "polarization";`
	`        rr:datatype xsd:string`
	`    ]`
	`];`

Link to Platform object
(Table 8)
	`# Triple: <subject> eop:platform <#Platform>`
	`rr:predicateObjectMap [`
	`   rr:predicate eop:platform;`
	`   rr:objectMap [`
	`        rr:parentTriplesMap <#Platform>;`
	`   ];`
	`];`

Link to Instrument object
(Table 9)
	`# Triple: <subject> eop:instrument <#Instrument>`
	`rr:predicateObjectMap [`
	`   rr:predicate eop:instrument;`
	`   rr:objectMap [`
	`        rr:parentTriplesMap <#Instrument>;`
	`   ];`
	`];`
	`.`

Table 8 – Dataset Platform Object Mapping Rules

Properties	**Mapping Rules**
	`<#Platform>`

Static platform URI	`# Specify the Platform URI (from NASA GCMD concepts[2])`
	`rr:subjectMap [`
	`    rr:constant gcmd:007c3084-89db-458e-8387-14e192b6cb8e;`
	`];`

Static platform label	`# Hard-coded: <subject> skos:altLabel "SENTINEL-1"^^xsd:string`
	`# Use other RML templates to convert Sentinel-2, etc. metadata`
	`rr:predicateObjectMap [`
	`    rr:predicate skos:altLabel;`
	`    rr:object "SENTINEL-1"`
	`];`
	`.`

[2] Sentinel platforms:
Sentinel-1: https://gcmd.earthdata.nasa.gov/kms/concept/007c3084-89db-458e-8387-14e192b6cb8e
Sentinel-2: https://gcmd.earthdata.nasa.gov/kms/concept/2ce20983-98b2-40b9-bb0e-a08074fb93b3
Sentinel-3: https://gcmd.earthdata.nasa.gov/kms/concept/8a19f309-46ee-424b-be9f-e7e57e5b8ca0
Sentinel-5p: https://gcmd.earthdata.nasa.gov/kms/concept/77e9a75e-2c3b-428b-8b21-b6a902dd8fe
List of platforms in JSON format:
https://gcmdservices.gsfc.nasa.gov/kms/concepts/concept_scheme/platforms/?format=json

Table 9 – Dataset Instrument Object Mapping Rules

Properties	Mapping Rules

Static instrument URI
 	
instrument
	<#Instrument>
	
	    # Specify the Instrument URI (from NASA GCMD concepts[3])
	    rr:subjectMap [
	        rr:constant gcmd:1c53d85e-3792-4081-9748-192fd3140aa6;
	    ];
	
	    # Triple: <subject> skos:altLabel "SAR-C SAR"^^xsd:string
	    # Use other RML templates to convert Sentinel-2, etc. metadata
	    rr:predicateObjectMap [
	        rr:predicate skos:altLabel;
	        rr:objectMap [
	            rml:reference "instrument";
	            rr:datatype xsd:string
	        ]
	    ];
	.

Table 10 – Dataset Result Object Mapping Rules

Properties	Mapping Rules

Result subject URI	<#Result>
	
	    # Specify the subject URI and RDF type
	    rr:subjectMap [
	        rr:template "{source}/{uuid}/result"
	    ];
	
	    rr:predicateObjectMap [
	        rr:predicate rdf:type;
	        rr:object rdf:Class
	    ];

Link to Media Content object (Table 11)
 	
Link to Quicklook object (Table 12)
	# Triple: <subject> media:group <#Content>
	    rr:predicateObjectMap [
	       rr:predicate media:group;
	       rr:objectMap [
	            rr:parentTriplesMap <#Content>;
	       ];
	    ];
	
	    # Triple: <subject> media:group <#ContentQuicklook>
	    rr:predicateObjectMap [
	       rr:predicate media:group;
	       rr:objectMap [
	            rr:parentTriplesMap <#ContentQuicklook>;
	       ];
	    ];
	.

[3] Sentinel-1 C-SAR instrument: https://gcmd.earthdata.nasa.gov/kms/concept/1c53d85e-3792-4081-9748-192fd3140aa6

Table 11 – Dataset Media Content Object Mapping Rules

Properties	**Mapping Rules**
	`<#Content>`
Media Content URI and type	`    # Specify the subject URI and RDF type`
	`    rr:subjectMap [`
	`        rr:template "{url_download}"`
	`    ];`
	`    rr:predicateObjectMap [`
	`        rr:predicate rdf:type;`
	`        rr:object media:Content`
	`    ];`
Implicit media type	`    # Hard-coded media type`
	`    # <subject> media:type "application/binary"^^xsd:string`
	`    rr:predicateObjectMap [`
	`        rr:predicate media:type;`
	`        rr:object "application/binary"`
	`    ];`
Implicit media expression	`    # Hard-coded media expression`
	`    # <subject> media:expression "full"^^xsd:string`
	`    rr:predicateObjectMap [`
	`        rr:predicate media:expression;`
	`        rr:object "full"`
	`    ];`
size	`    # Triple: <subject> media:fileSize "{size}"^^xsd:string`
	`    rr:predicateObjectMap [`
	`        rr:predicate media:fileSize;`
	`        rr:objectMap [`
	`            rml:reference "size";`
	`            rr:datatype xsd:string`
	`       ]`
	`    ];`
	`.`
Note:	No mapping is foreseen in the recommendation for a checksum URL.
Note:	The size value should be an amount of bytes. The Umbrella Application returns a non-integer value (e.g. "1.65 GB")

Table 12 – Dataset Quicklook Content Object Mapping Rules

Properties	**Mapping Rules**
	`<#ContentQuicklook>`
Quicklook URI and type	`    # Specify the subject URI and RDF type`
	`    rr:subjectMap [`
	`        rr:template "{quicklook}"`
	`    ];`
	`    rr:predicateObjectMap [`
	`        rr:predicate rdf:type;`
	`        rr:object media:Content`
	`    ];`

Implicit media type	`# Hard-coded media type`
	`# <subject> media:type "image/jpeg"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate media:type;`
	`    rr:object "image/jpeg"`
	`];`

Implicit medium	`# Hard-coded media medium`
	`# <subject> media:medium "image"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate media:medium;`
	`    rr:object "image"`
	`];`

Implicit media expression	`# Hard-coded media expression`
	`# <subject> media:expression "sample"^^xsd:string`
	`rr:predicateObjectMap [`
	`    rr:predicate media:expression;`
	`    rr:object "sample"`
	`];`

Link to Media Category object (Table 13)	`# Triple: <subject> media:category <#QuicklookCategory>`
	`rr:predicateObjectMap [`
	`   rr:predicate media:category;`
	`   rr:objectMap [`
	`        rr:parentTriplesMap <#QuicklookCategory>;`
	`   ];`
	`];`
	`.`

Table 13 – Dataset Quicklook Category Object Mapping Rules

Properties	**Mapping Rules**

Quicklook Media Category URI and type	`<#QuicklookCategory>`
	
	`    # Specify the subject URI and RDF type`
	`    rr:subjectMap [`
	`        rr:template "{source}/{uuid}/result/quicklook/category"`
	`    ];`
	
	`    rr:predicateObjectMap [`
	`        rr:predicate rdf:type;`
	`        rr:object media:Category`
	`    ];`

Implicit quicklook about property	`    # Hard-coded QUICKLOOK category`
	`    # <subject> rdf:about "http://[...]/1.0#QUICKLOOK"`
	`    rr:predicateObjectMap [`
	`        rr:predicate rdf:about;`
	`        rr:object "http://www.opengis.net/spec/EOMPOM/1.0#QUICKLOOK"`
	`    ];`
	`.`

Table 14 contains a fragment of the Linked Data (RDF) document generated by GeoTriples embedded in the "Any to RDF Converter" process.

Table 14 – Generated Linked EO Data (Fragment)

Generated Linked EO Data (Fragment)

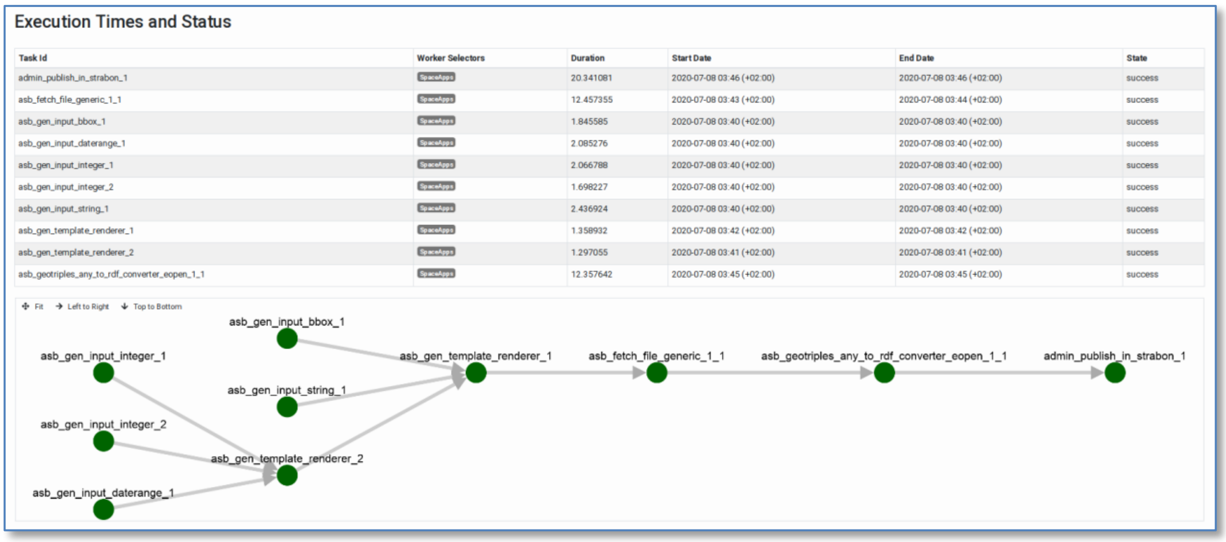
`<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>`
`<http://www.opengis.net/eop/2.1#orbitDirection> "DESCENDING"^^<http://www.w3.org/2001/XMLSchema#string> .`
`<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10> <http://purl.org/dc/terms/spatial>`
`"<http://www.opengis.net/def/crs/EPSG/0/4326> MULTIPOLYGON (((22.735907 39.305035, 23.118069 40.806633,`
`20.056725 41.211155, 19.741592 39.710724, 22.735907`
`39.305035)))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .`
`<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>`

```
<http://www.opengis.net/eop/2.1#sensor_operational_mode> "IW"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://def.seegrid.csiro.au/ontology/om/om-lite#Observation> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://www.opengis.net/eop/2.1#polarization> "VV VH"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://www.opengis.net/eop/2.1#sensorType> "RADAR"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://def.seegrid.csiro.au/ontology/om/om-lite#result> <https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-
87ee-0a5a5aba7b10/result> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/result>
<http://search.yahoo.com/mrss/group> <https://data.sentinel.zamg.ac.at/odata/v1/Products('dfccba6e-746a-4201-
87ee-0a5a5aba7b10')/$value> .
<https://data.sentinel.zamg.ac.at/odata/v1/Products('dfccba6e-746a-4201-87ee-0a5a5aba7b10')/$value>
<http://search.yahoo.com/mrss/expression> "full"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/odata/v1/Products('dfccba6e-746a-4201-87ee-0a5a5aba7b10')/$value>
<http://search.yahoo.com/mrss/type> "application/binary"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/odata/v1/Products('dfccba6e-746a-4201-87ee-0a5a5aba7b10')/$value>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://search.yahoo.com/mrss/Content> .
<https://data.sentinel.zamg.ac.at/odata/v1/Products('dfccba6e-746a-4201-87ee-0a5a5aba7b10')/$value>
<http://search.yahoo.com/mrss/fileSize> "1.63 GB"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/result> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Class> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://def.seegrid.csiro.au/ontology/om/om-lite#phenomenonTime> <https://data.sentinel.zamg.ac.at/dfccba6e-
746a-4201-87ee-0a5a5aba7b10/phenomenonTime> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/phenomenonTime>
<http://www.w3.org/2002/12/cal/ical#dtstart> "2020-01-
01T04:39:23.890000Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/phenomenonTime>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://def.seegrid.csiro.au/ontology/om/om-
lite#TemporalObject> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/phenomenonTime>
<http://www.w3.org/2002/12/cal/ical#dtend> "2020-01-
01T04:39:23.890000Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://www.opengis.net/eop/2.1#productType> "GRD"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10> <http://purl.org/dc/terms/description>
"EO Product record dfccba6e-746a-4201-87ee-0a5a5aba7b10, with filename
S1A_IW_GRDH_1SDV_20200101T043923_20200101T043948_030602_038181_45E0.SAFE from the Umbrella Application (source:
https://data.sentinel.zamg.ac.at)"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://www.opengis.net/eop/2.1#processingDate> "2020-01-
01T08:46:25.439000Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://www.w3.org/ns/dcat#Dataset> .
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10> <http://purl.org/dc/terms/identifier>
"S1A_IW_GRDH_1SDV_20200101T043923_20200101T043948_030602_038181_45E0"^^<http://www.w3.org/2001/XMLSchema#string>
.
<http://www.w3.org/ns/dcat#Dataset/dfccba6e-746a-4201-87ee-0a5a5aba7b10>
<http://def.seegrid.csiro.au/ontology/om/om-lite#procedure> <https://data.sentinel.zamg.ac.at/dfccba6e-746a-
4201-87ee-0a5a5aba7b10/process> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#orbitDirection> "DESCENDING"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#swathIdentifier> "IW"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#instrument> <http://gcmdservices.gsfc.nasa.gov/kms/concept/1c53d85e-3792-4081-
9748-192fd3140aa6> .
<http://gcmdservices.gsfc.nasa.gov/kms/concept/1c53d85e-3792-4081-9748-192fd3140aa6>
<http://www.w3.org/2004/02/skos/core#altLabel> "SAR-C SAR"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#orbitNumber> "30602"^^<http://www.w3.org/2001/XMLSchema#integer> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#sensorType> "RADAR"^^<http://www.w3.org/2001/XMLSchema#string> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process> <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> <http://def.seegrid.csiro.au/ontology/om/om-lite#Process> .
<https://data.sentinel.zamg.ac.at/dfccba6e-746a-4201-87ee-0a5a5aba7b10/process>
<http://www.opengis.net/eop/2.1#platform> <http://gcmdservices.gsfc.nasa.gov/kms/concept/007c3084-89db-458e-
8387-14e192b6cb8e> .
<http://gcmdservices.gsfc.nasa.gov/kms/concept/007c3084-89db-458e-8387-14e192b6cb8e>
<http://www.w3.org/2004/02/skos/core#altLabel> "SENTINEL-1"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/3a1164e8-da04-4402-9953-50e877802e19>
<http://www.opengis.net/eop/2.1#orbitDirection> "DESCENDING"^^<http://www.w3.org/2001/XMLSchema#string> .
<http://www.w3.org/ns/dcat#Dataset/3a1164e8-da04-4402-9953-50e877802e19> <http://purl.org/dc/terms/spatial>
"<http://www.opengis.net/def/crs/EPSG/0/4326> MULTIPOLYGON (((22.735899 39.305023, 23.118059 40.806622,
20.056717 41.21114, 19.741587 39.710712, 22.735899
39.305023)))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
[...]
```

4.4.1.3 Workflow Execution Report

The Execution Report generated in the EOPEN Platform contains all the process input and output values as well as a link for accessing the generated files in the EOPEN Datastore.

It also contains the start and duration of each process as well as a graphical representation of the complete workflow (see Figure 10).


Execution Times and Status

Task Id	Worker Selectors	Duration	Start Date	End Date	State
admin_publish_in_strabon_1	SpaceApps	20.341081	2020-07-08 03:46 (+02:00)	2020-07-08 03:46 (+02:00)	success
asb_fetch_file_generic_1_1	SpaceApps	12.457355	2020-07-08 03:43 (+02:00)	2020-07-08 03:44 (+02:00)	success
asb_gen_input_bbox_1	SpaceApps	1.845585	2020-07-08 03:40 (+02:00)	2020-07-08 03:40 (+02:00)	success
asb_gen_input_daterange_1	SpaceApps	2.085276	2020-07-08 03:40 (+02:00)	2020-07-08 03:40 (+02:00)	success
asb_gen_input_integer_1	SpaceApps	2.066788	2020-07-08 03:40 (+02:00)	2020-07-08 03:40 (+02:00)	success
asb_gen_input_integer_2	SpaceApps	1.698227	2020-07-08 03:40 (+02:00)	2020-07-08 03:40 (+02:00)	success
asb_gen_input_string_1	SpaceApps	2.436924	2020-07-08 03:40 (+02:00)	2020-07-08 03:40 (+02:00)	success
asb_gen_template_renderer_1	SpaceApps	1.358932	2020-07-08 03:42 (+02:00)	2020-07-08 03:42 (+02:00)	success
asb_gen_template_renderer_2	SpaceApps	1.297055	2020-07-08 03:41 (+02:00)	2020-07-08 03:41 (+02:00)	success
asb_geotriples_any_to_rdf_converter_eopen_1_1	SpaceApps	12.357642	2020-07-08 03:45 (+02:00)	2020-07-08 03:45 (+02:00)	success

Fit
Left to Right
Top to Bottom
asb_gen_input_bbox_1
asb_gen_input_integer_1
asb_gen_template_renderer_1
asb_fetch_file_generic_1_1
asb_geotriples_any_to_rdf_converter_eopen_1_1
admin_publish_in_strabon_1
asb_gen_input_string_1
asb_gen_input_integer_2
asb_gen_template_renderer_2
asb_gen_input_daterange_1

Figure 10 – Linked Data Generation and Storage Workflow Execution Report (Fragment)

4.4.2 Interoperable semantic query to retrieve EO and non-EO data

4.4.2.1 Workflow Description

This workflow is used to demonstrate the possibility to query two or more graph databases containing EO and non-EO data, merge the results, and obtain either a new graph or a search result list. Both can be further processed or visualised.

Figure 11 shows the workflow as edited in the Workflow Editor. The steps are as follow:

1. A "template renderer" is used to build the SPARQL query that will be used to search for Linked Data in several graph databases. This general purpose built-in function is described in the Developer Guide D6.5. A CONSTRUCT SPARQL query is used to obtain sub-graphs from the databases.
2. In parallel: Query a GraphDB instance and a Strabon instance. The output RDF graphs are stored on disk.
3. Merge the two RDF graphs into a single one and store the result on disk.
4. Apply a SPARQL query on the resulting graph to obtain a SPARQL Results document.

Because we want to semantically merge the results from two or more SPARQL queries, in step 2 we must use CONSTRUCT queries and not SELECT queries. A SELECT query produces SPARQL results that may be compared to a traditional search result list. These results may be concatenated but the output may not be considered as a semantic graph. A CONSTRUCT query, on the other hand, produces a sub-graph that can be merged with other sub-graphs to obtain the linked data that matches the original SPARQL queries. This linked data may be further queried, new connections can be discovered and added in the graph, and it may be serialized in any of the available formats, including RDF/JSON, RDF/XML, N3, N-Triples and Turtle.

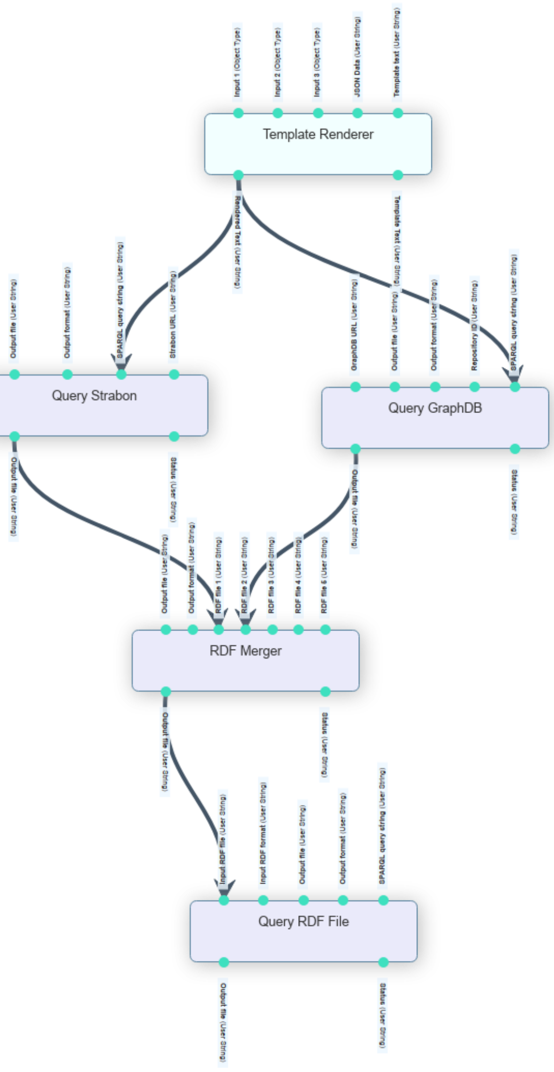

Input 1 (Object Type)
Input 2 (Object Type)
Input 3 (Object Type)
JSON Data (User String)
Template text (User String)
Template Renderer
Rendered Text (User String)
Template Text (User String)
Output file (User String)
Output format (User String)
SPARQL query string (User String)
Strabon URL (User String)
Query Strabon
GraphDB URL (User String)
Output file (User String)
Output format (User String)
Repository ID (User String)
SPARQL query string (User String)
Query GraphDB
Output file (User String)
Status (User String)
Output file (User String)
Status (User String)
Output file (User String)
Output format (User String)
RDF file 1 (User String)
RDF file 2 (User String)
RDF file 3 (User String)
RDF file 4 (User String)
RDF file 5 (User String)
RDF Merger
Output file (User String)
Status (User String)
Input RDF file (User String)
Input RDF format (User String)
Output file (User String)
Output format (User String)
SPARQL query string (User String)
Query RDF File
Output file (User String)
Status (User String)

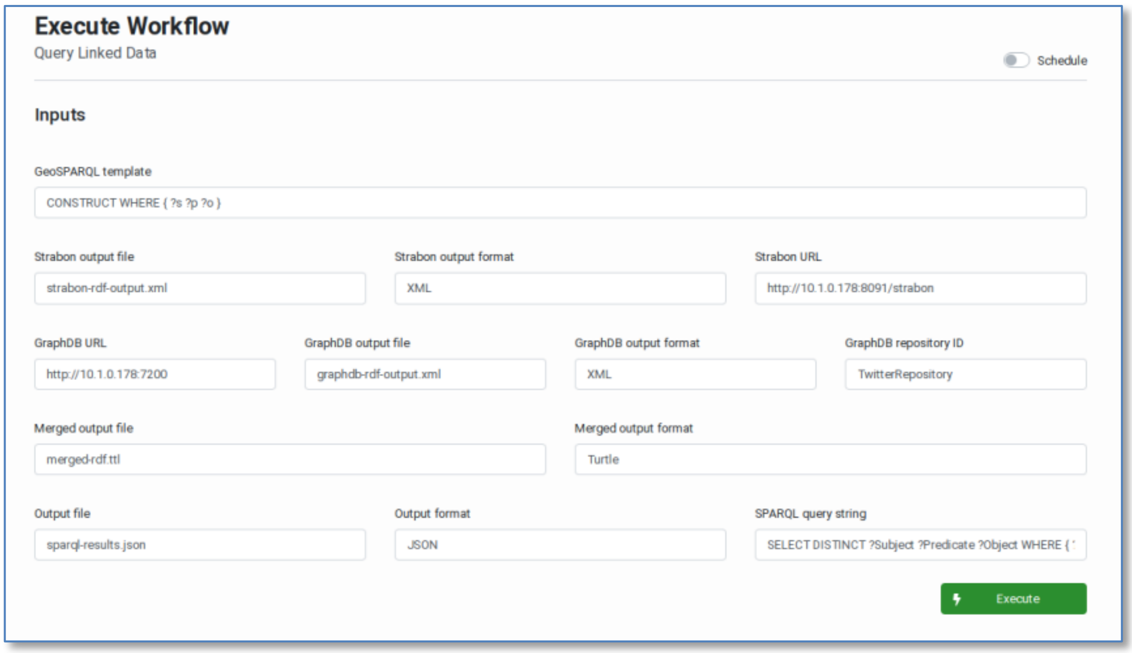
Figure 11 – EO and non-EO Linked Data Query Workflow

Figure 12 shows the parameterisation form displayed to the users who want to manually execute the workflow.

The input parameters are:

1. The GeoSPARQL query to be sent to the graph databases
2. The Strabon end-point URL, output file name and format
3. The GraphDB end-point URL, repository ID, output file name and format
4. The name and format of the merged RDF graph
5. The SPARQL query to be used to generate the SPARQL Results document, the name of the output file and format

The form is pre-filled-in with the default values configured in the workflow definition via the Workflow Editor.



Execute Workflow
Query Linked Data

Schedule

Inputs

GeoSPARQL template	CONSTRUCT WHERE { ?s ?p ?o }
Strabon output file	strabon-rdf-output.xml
Strabon output format	XML
Strabon URL	http://10.1.0.178:8091/strabon
GraphDB URL	http://10.1.0.178:7200
GraphDB output file	graphdb-rdf-output.xml
GraphDB output format	XML
GraphDB repository ID	TwitterRepository
Merged output file	merged-rdf.ttl
Merged output format	Turtle
Output file	sparql-results.json
Output format	JSON
SPARQL query string	SELECT DISTINCT ?Subject ?Predicate ?Object WHERE { '

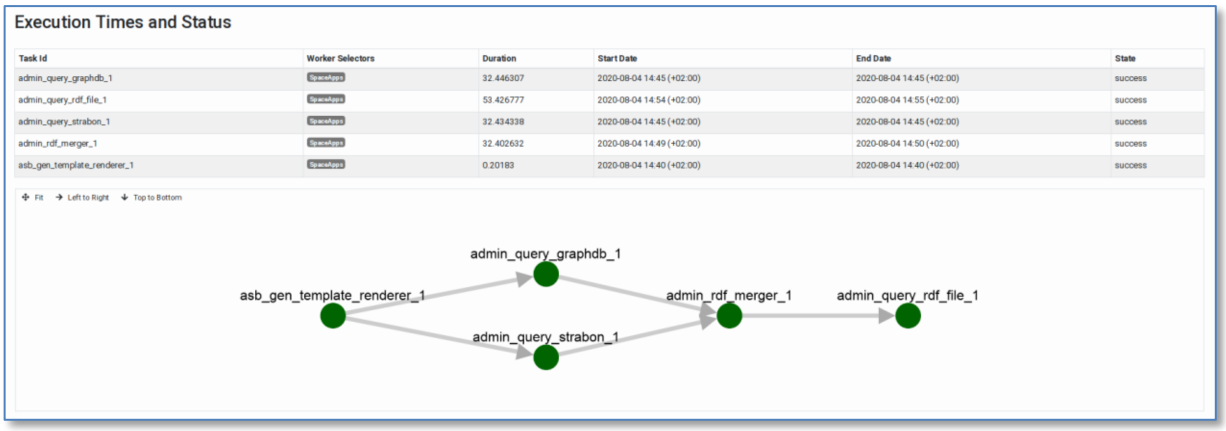
Figure 12 – EO and non-EO Linked Data Query Parameters

When the workflow is fully executed, we have on disk two versions of the selected Linked Data: the first version contains an RDF graph (which can be further queries, stored in a graph database, or displayed as an interactive graph), and the second one contains a SPARQL Results document that is suitable for visualisation in a more tabular manner (see section 4.5.3).

4.4.2.2 Workflow Execution Report

The Execution Report generated in the EOPEN Platform contains all the process input and output values as well as a link for accessing the generated files in the EOPEN Datastore.

It also contains the start and duration of each process as well as a graphical representation of the complete workflow (see Figure 13).

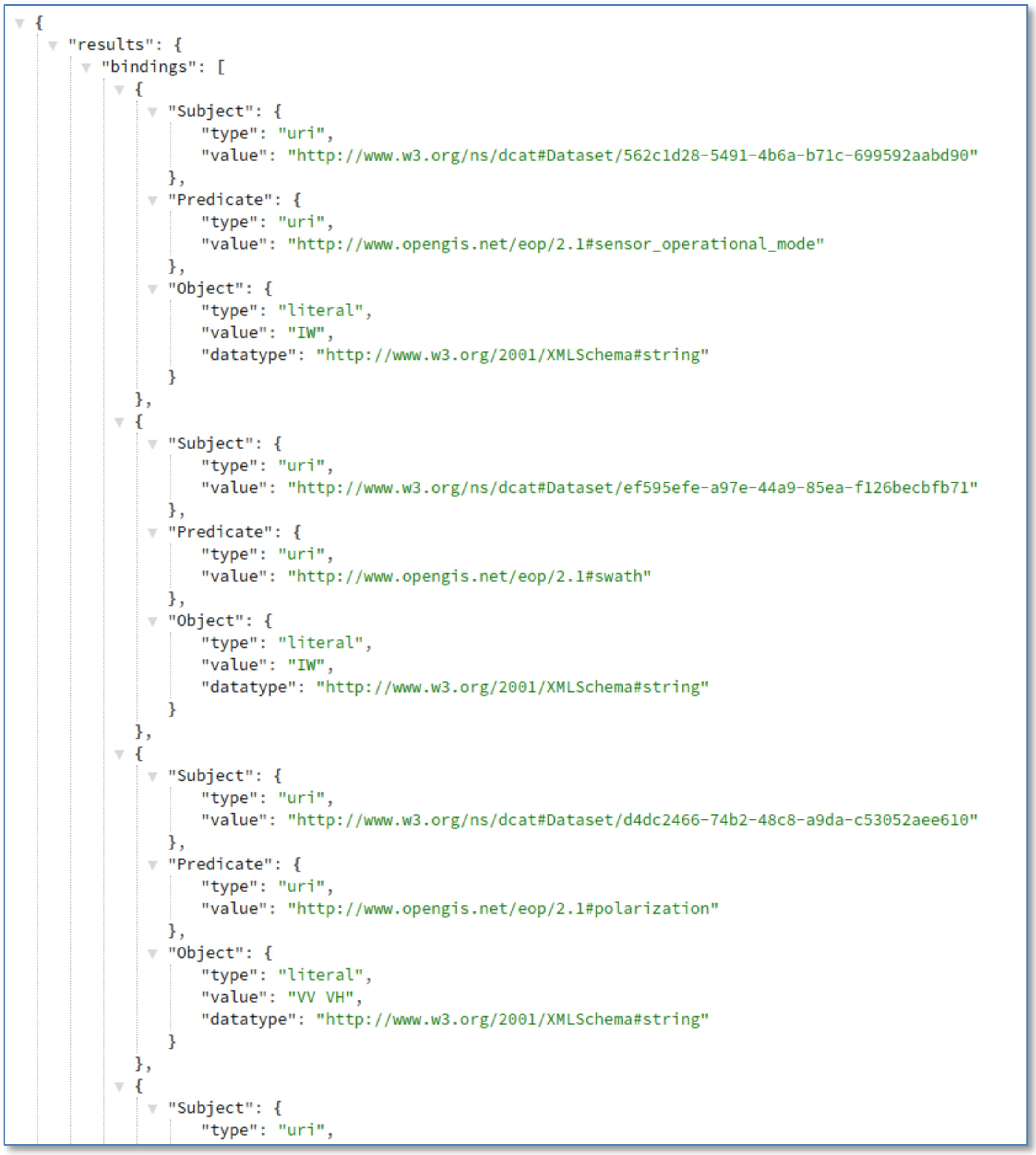


Execution Times and Status

Task Id	Worker Selectors	Duration	Start Date	End Date	State
admin_query_graphdb_1	SpaceApps	32.446307	2020-08-04 14:45 (+02:00)	2020-08-04 14:45 (+02:00)	success
admin_query_rdf_file_1	SpaceApps	53.426777	2020-08-04 14:54 (+02:00)	2020-08-04 14:55 (+02:00)	success
admin_query_strabon_1	SpaceApps	32.434338	2020-08-04 14:45 (+02:00)	2020-08-04 14:45 (+02:00)	success
admin_rdf_merger_1	SpaceApps	32.402632	2020-08-04 14:49 (+02:00)	2020-08-04 14:50 (+02:00)	success
asb_gen_template_renderer_1	SpaceApps	0.20183	2020-08-04 14:40 (+02:00)	2020-08-04 14:40 (+02:00)	success

Figure 13 – Linked Data Generation and Storage Workflow Execution Report (Fragment)

Figure 14 shows a fragment of SPARQL Results document formatted in SPARQL/JSON. This format is particularly suited for visualisation in a Web interface.



```
{
    "results": {
        "bindings": [
            {
                "Subject": {
                    "type": "uri",
                    "value": "http://www.w3.org/ns/dcat#Dataset/562c1d28-5491-4b6a-b71c-699592aabd90"
                },
                "Predicate": {
                    "type": "uri",
                    "value": "http://www.opengis.net/eop/2.1#sensor_operational_mode"
                },
                "Object": {
                    "type": "literal",
                    "value": "IW",
                    "datatype": "http://www.w3.org/2001/XMLSchema#string"
                }
            },
            {
                "Subject": {
                    "type": "uri",
                    "value": "http://www.w3.org/ns/dcat#Dataset/ef595efe-a97e-44a9-85ea-f126becbfb71"
                },
                "Predicate": {
                    "type": "uri",
                    "value": "http://www.opengis.net/eop/2.1#swath"
                },
                "Object": {
                    "type": "literal",
                    "value": "IW",
                    "datatype": "http://www.w3.org/2001/XMLSchema#string"
                }
            },
            {
                "Subject": {
                    "type": "uri",
                    "value": "http://www.w3.org/ns/dcat#Dataset/d4dc2466-74b2-48c8-a9da-c53052aee610"
                },
                "Predicate": {
                    "type": "uri",
                    "value": "http://www.opengis.net/eop/2.1#polarization"
                },
                "Object": {
                    "type": "literal",
                    "value": "VV VH",
                    "datatype": "http://www.w3.org/2001/XMLSchema#string"
                }
            },
            {
                "Subject": {
                    "type": "uri",
```

Figure 14 – SPARQL Results Document of Merged Linked Data (Fragment)

4.5 Visualisation of RDF Graphs and SPARQL Results

As explained above, searching for Linked Data in graph databases may lead to results that are still organised in graphs (result of CONSTRUCT SPARQL queries) or to flat lists of subject-predicate-object triples (result of SELECT SPARQL queries). Different means may be used to visualise and interact with the two types of results.

4.5.1 Visualisation of RDF Graphs

Graph data is typically displayed as interactive networks of nodes and links. An example is the output of the EOPEN User Portal component that displays the communities detected in social media (Figure 15).

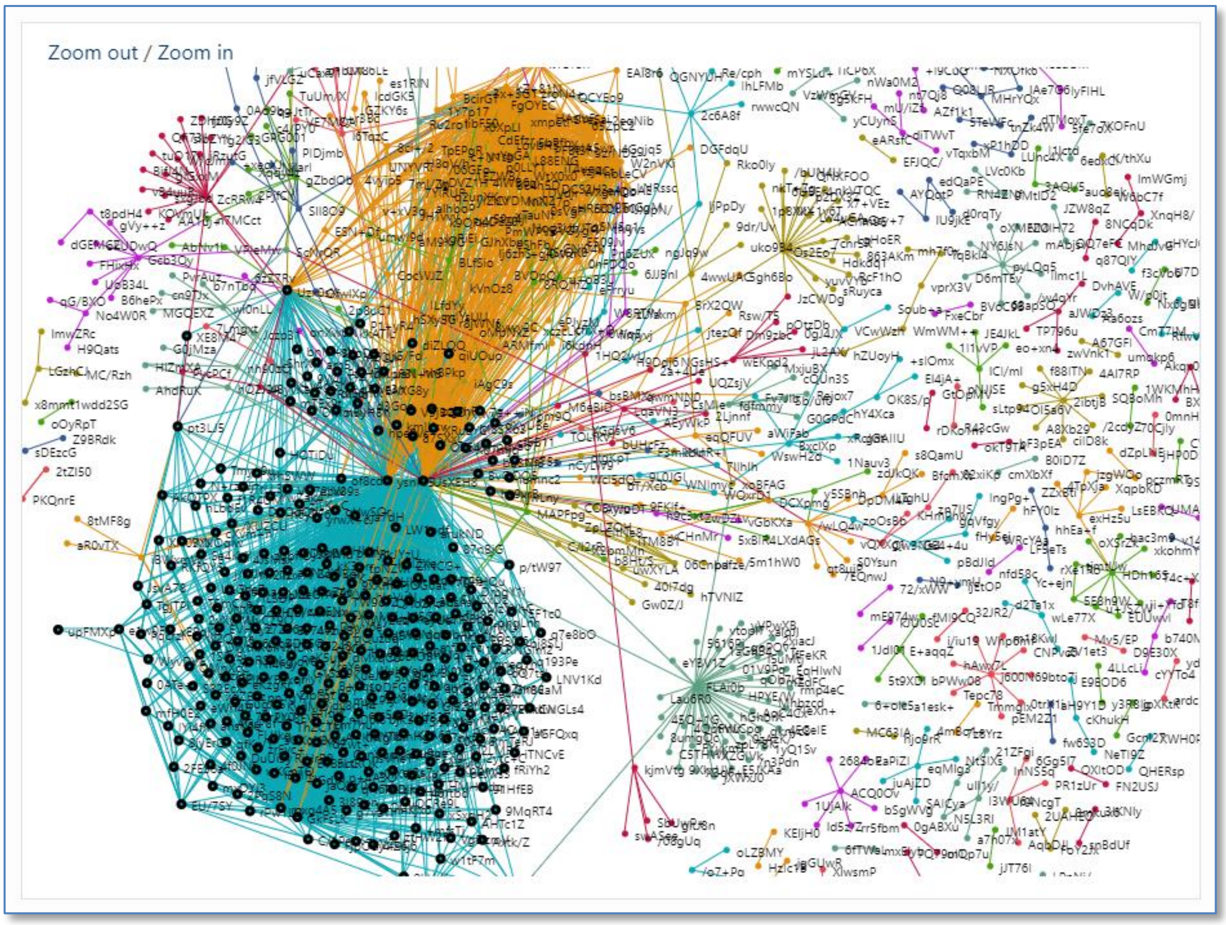


Figure 15 – Social Media Communities Graph (source: D6.5)

The same Dashboard component has been used to visualise RDF graphs containing both non-EO data retrieved from GraphDB and EO products metadata retrieved from Strabon. An example result is shown in Figure 16.

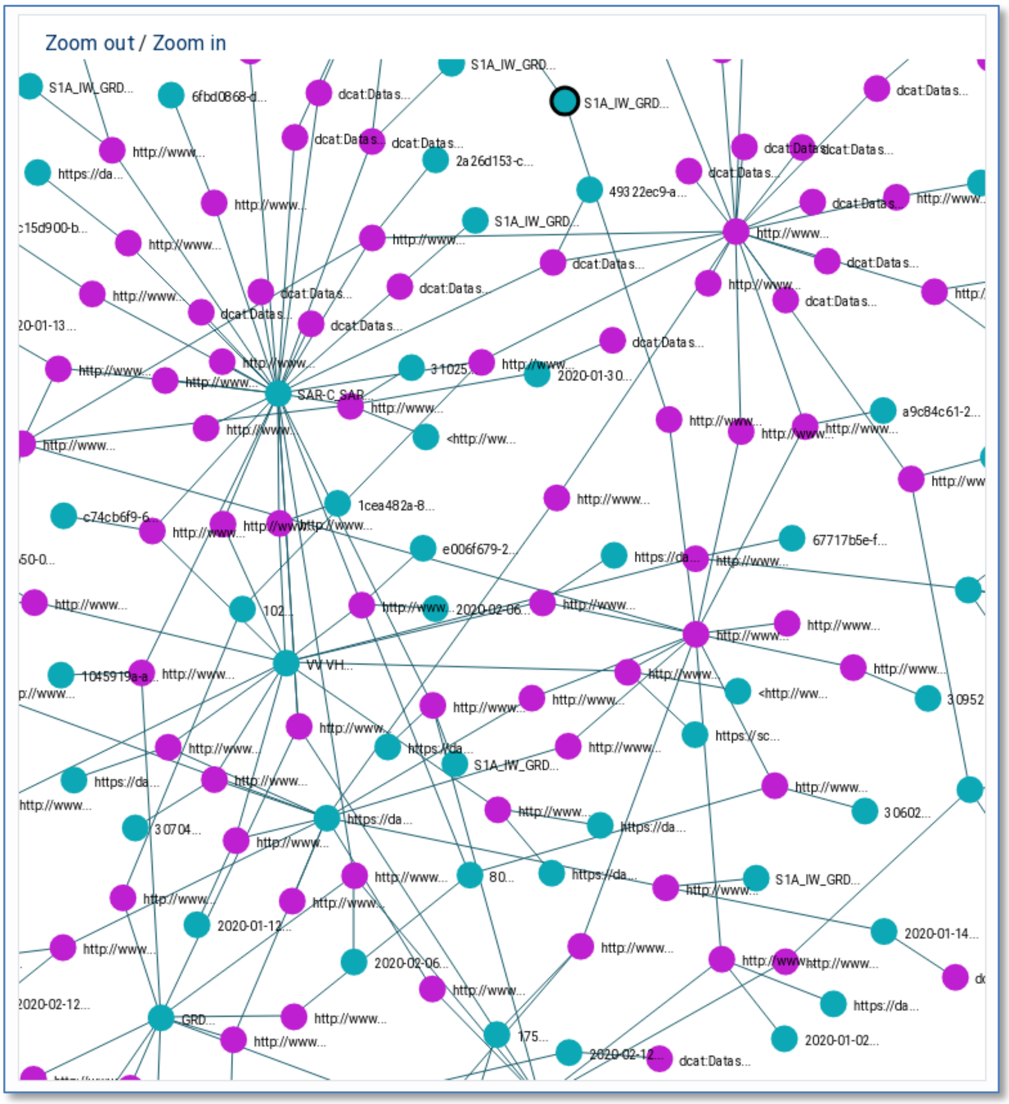


Figure 16 – EO and non-EO Entities Graph

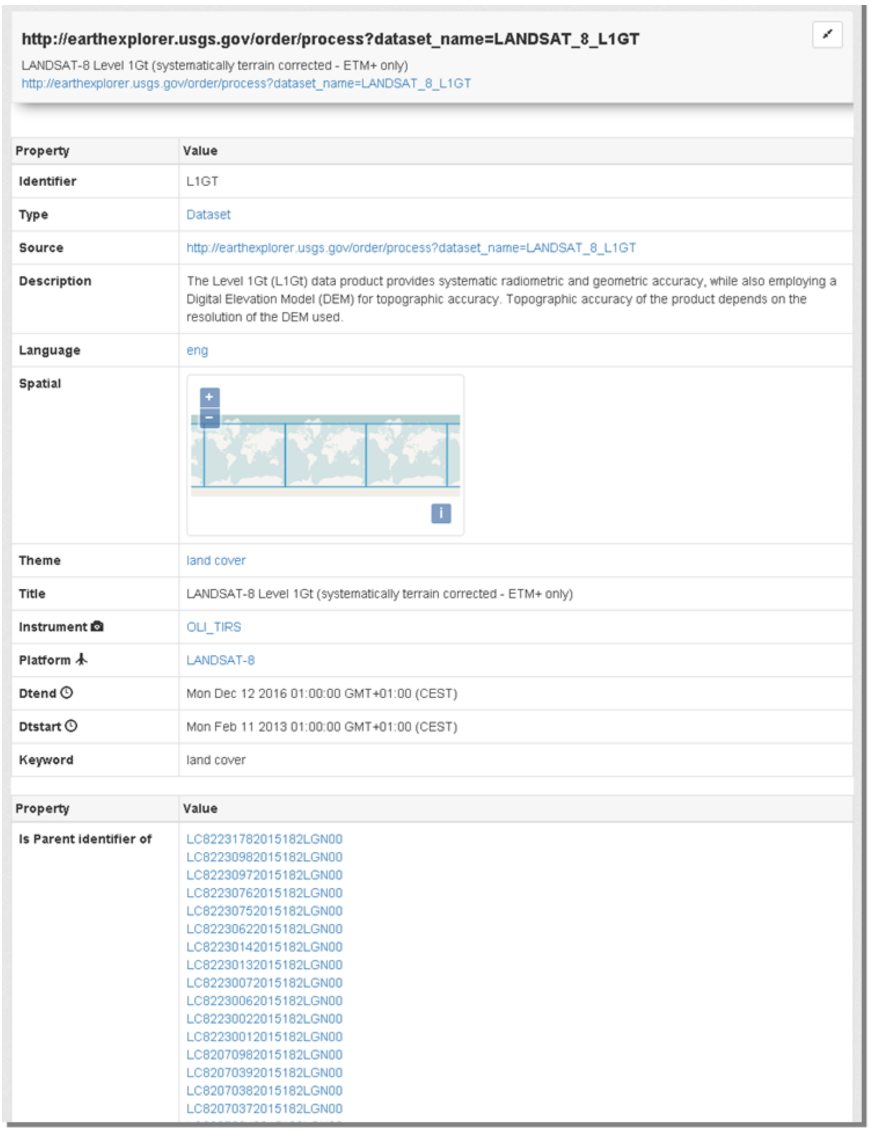
The size of an RDF graph may rapidly become too big to be usable in a user interface. To circumvent this problem the visualisation component may be extended to dynamically filter the types of nodes (RDF Subjects and Objects) and links (RDF Predicates) to be displayed on the screen. Another enhancement is the possibility to dynamically configure the colour and size of the nodes and links depending on their type. This allows keeping the information visible on the screen and highlighting only the information of interest.

Selecting an element (link or node) in the graph may reveal all the information available about it in the graph in a side component, and in particular the list of RDF triples in which it is present.

A second way to visualise an RDF graph is using a table-based browser. The browser shows in table information known about a given subject that is its links to objects (which may be subjects on their turn) and literal values (properties).

In such a graph browser the objects are active links which when clicked reveals the information available about that object (as a subject). This allows navigating in the graph interactively.

Figure 17 shows an example of Linked Data browser implemented by Space Applications Services in a past project. It shows a Landsat-8 dataset series (EO products collection) entry, the predicates (in the first table column) and the objects and properties (in the second column). The objects are clickable links which allow navigating to their dedicated pages.



http://earthexplorer.usgs.gov/order/process?dataset_name=LANDSAT_8_L1GT

LANDSAT-8 Level 1Gt (systematically terrain corrected - ETM+ only)

http://earthexplorer.usgs.gov/order/process?dataset_name=LANDSAT_8_L1GT

Property	Value
Identifier	L1GT
Type	Dataset
Source	http://earthexplorer.usgs.gov/order/process?dataset_name=LANDSAT_8_L1GT
Description	The Level 1Gt (L1Gt) data product provides systematic radiometric and geometric accuracy, while also employing a Digital Elevation Model (DEM) for topographic accuracy. Topographic accuracy of the product depends on the resolution of the DEM used.
Language	eng
Spatial	
Theme	land cover
Title	LANDSAT-8 Level 1Gt (systematically terrain corrected - ETM+ only)
Instrument	OLI_TIRS
Platform	LANDSAT-8
Dtend	Mon Dec 12 2016 01:00:00 GMT+01:00 (CEST)
Dtstart	Mon Feb 11 2013 01:00:00 GMT+01:00 (CEST)
Keyword	land cover

Property	Value
Is Parent identifier of	LC82231782015182LGN00 LC82230982015182LGN00 LC82230972015182LGN00 LC82230762015182LGN00 LC82230752015182LGN00 LC82230622015182LGN00 LC82230142015182LGN00 LC82230132015182LGN00 LC82230072015182LGN00 LC82230062015182LGN00 LC82230022015182LGN00 LC82230012015182LGN00 LC82070982015182LGN00 LC82070392015182LGN00 LC82070382015182LGN00 LC82070372015182LGN00

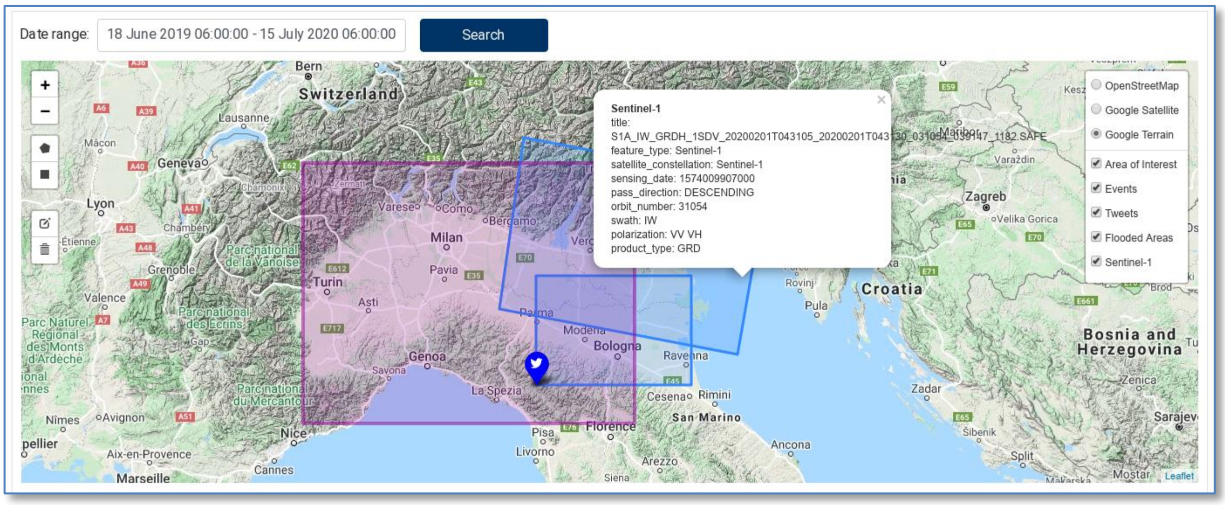
Figure 17 – Linked EO Data Browser showing a Landsat-8 Dataset Series

4.5.2 RDF Graphs with Spatio-Temporal Dimensions

RDF graphs that link subjects to temporal and locations properties may be filtered by time of interest (TOI) and area of interest (AOI) but may also be displayed on interactive maps.

The merged graph, output of the workflow described in section 4.4.2, contains multiple subjects that fulfil these criteria, namely tweets, detected events, flooded locations and EO products (satellite images).

Figure 18 shows the Geolocalised Linked Data Dashboard component described in D6.5 re-purposed to display both EO and non-EO Linked Data. The figure shows data filtered using the box in purple (AOI) drawn by the user and the date range (TOI) selector. Displayed data include tweets, events and an EO product from the Sentinel-1 mission. The popup displays some of the EO product properties.


Date range: 18 June 2019 06:00:00 - 15 July 2020 06:00:00

Sentinel-1
title:
S1A_IW_GRDH_1SDV_20200201T043105_20200201T043130_031054_039147_1182.SAFE
feature_type: Sentinel-1
satellite_constellation: Sentinel-1
sensing_date: 1574009907000
pass_direction: DESCENDING
orbit_number: 31054
swath: IW
polarization: VV VH
product_type: GRD

Figure 18 – Geo-localised Linked EO and non-EO Data in a Dashboard Component

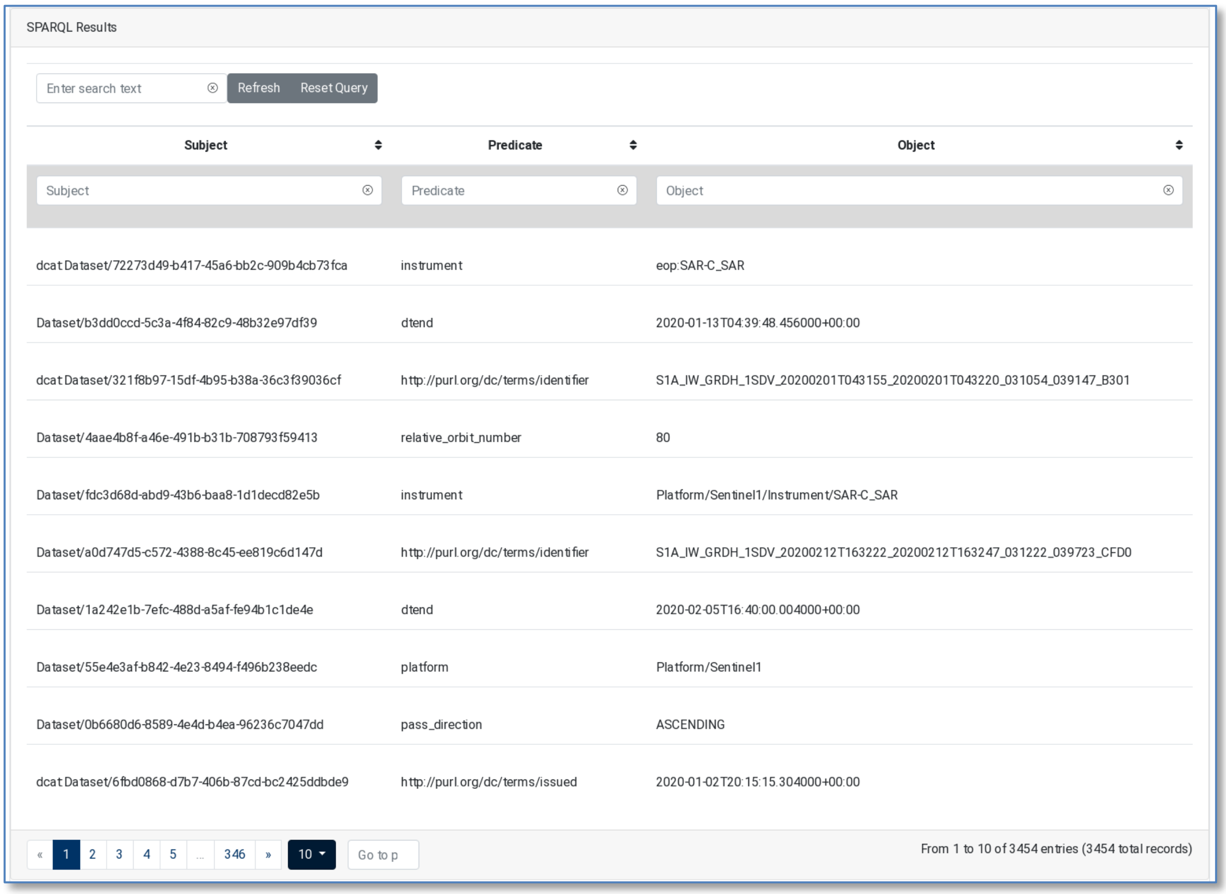
4.5.3 Generic SPARQL Results Table

As explained above, SPARQL Results which are obtained by issuing SELECT queries are not organised in graphs. Each entry in the result list is a record as defined in the SELECT statement. This may be a triple, if the query explicitly selects subjects, predicates and objects ("`SELECT ?s ?p ?o WHERE …`") but it may contain other combinations of values as well.

A Dashboard component (Figure 19) has been implemented to allow listing SPARQL Results listed in a table. The table adapts automatically to the values present in the result records and each column includes a field that allows filtering the table rows using string fragments.

The table is also paginated. By default, 10 entries are displayed.

The component loads the SPARQL Results document in memory which means that using the filters and the pagination do not require the component to further communicate with the backend.



Subject	Predicate	Object
dcat:Dataset/72273d49-b417-45a6-bb2c-909b4cb73fca	instrument	eop:SAR-C_SAR
Dataset/b3dd0ccd-5c3a-4f84-82c9-48b32e97df39	dtend	2020-01-13T04:39:48.456000+00:00
dcat:Dataset/321f8b97-15df-4b95-b38a-36c3f39036cf	http://purl.org/dc/terms/identifier	S1A_IW_GRDH_1SDV_20200201T043155_20200201T043220_031054_039147_B301
Dataset/4aae4b8f-a46e-491b-b31b-708793f59413	relative_orbit_number	80
Dataset/fdc3d68d-abd9-43b6-baa8-1d1decd82e5b	instrument	Platform/Sentinel1/Instrument/SAR-C_SAR
Dataset/a0d747d5-c572-4388-8c45-ee819c6d147d	http://purl.org/dc/terms/identifier	S1A_IW_GRDH_1SDV_20200212T163222_20200212T163247_031222_039723_CFD0
Dataset/1a242e1b-7efc-488d-a5af-fe94b1c1de4e	dtend	2020-02-05T16:40:00.004000+00:00
Dataset/55e4e3af-b842-4e23-8494-f496b238eedc	platform	Platform/Sentinel1
Dataset/0b6680d6-8589-4e4d-b4ea-96236c7047dd	pass_direction	ASCENDING
dcat:Dataset/6fbd0868-d7b7-406b-87cd-bc2425ddbde9	http://purl.org/dc/terms/issued	2020-01-02T20:15:15.304000+00:00

From 1 to 10 of 3454 entries (3454 total records)

Figure 19 – Visualisation of SPARQL Results in the EOPEN User Portal

5 POSSIBLE USES

Most EO product catalogues allow users to search and discover the available products either via a Web-based graphical interface (integrating a form and a map viewer) or via a programming interface (standard or not) exposed by a service. These catalogues allow searching for products using the traditional product type, area of interest (AOI) and date range (TOI) parameters and possibly other mission and product specific parameters such as the orbit number and direction, the sensing mode, the polarisation (SAR products) and the cloud coverage (optical products). This fits the needs of many use cases but may be too limited for others.

For example, it is not possible, using these interfaces to search for products "close enough" to a location (that is using a buffer around their footprint), having gaps of maximum "x" days between them, and having an intersection area bigger than "y" square kilometres.

Taking non-EO data into account, we can even extend this kind of request, for examples by filtering out from the result list the EO products that include (geographically) less than "z" tweets related to floods posted in the same week.

These two advanced examples can become reality when the information (EO products metadata, tweet and event properties, etc.) is encoded as Linked Data and the appropriate means to query these data is made available.

For short-term phenomena such as floods, Linked Data allows searching for and merging in a seamless manner all (and only) the relevant information (incl. EO products, tweets and flood maps) in just a few steps where traditional catalogues and interfaces would typically require querying different sources multiple times (for example for searching for EO products from different missions), fusing the results that are not readily compatible, and finally removing from the list the entries that do not meet the criteria.

Linked Data, and in particular the SPARQL query language also allows searching for graph fragments that show a given "pattern". This for example allows searching for similar occurrences of an event in a location by searching for all flood maps with a flooded area above a certain value, linked to at least "x" tweets about flood, produced in the same period of the year, and with similar weather conditions (provided the weather conditions are also available as Linked Data).

For longer term phenomena such as food security and climate change, Linked Data allows identifying all the spatio-temporal raster data (incl. images and maps) produced between two dates and intersecting a given area to obtain an heterogeneous list of products that may be fused for example to experiment with novel kinds of data analytics on timeseries.

Linked Data, and in particular the standards and the technologies behind it (RDF, RML, SPARQL, graph databases, etc.) allow obtaining, manipulating and analysing the data in a flexible and intuitive manner. The adoption of independent and well-supported ontologies brings a semantic homogeneity to the data. Standard formats allow using with generic tools.

Potential users must be informed about the possibilities of Linked Data in order to identify applicable requirements and implement the tools that fit their needs.

6 CONCLUSIONS

In this document we have described the work performed in the EOPEN Task 5.2 "Linked open Earth Observation Data", including the tools and technologies that have been involved, the EO data that has been transformed and made available as Linked Data, and the software that has been developed and integrated in the EOPEN Platform.

We have shown that by decomposing the tasks in elementary steps, it is possible to create processing workflows that meet the needs of the users. The demonstration workflows described in the document are used to transform structured EO products metadata into Linked Data, store this in a graph database, but also to query these databases, merge the responses and visualise the results in graphical, Web-based components available in the EOPEN User Portal. These workflows integrate steps (processes) that are generic enough to be re-configured without requiring further software development effort. For example, the manner EO products metadata is transformed into Linked Data is entirely controlled by mapping rules expressed in RML, storing and querying Linked Data can be done in graph databases deployed locally in the EOPEN Platform but also accessible remotely on the Internet, Linked Data is obtained using standard SPARQL queries that can be rendered on-the-fly, and visualisation components can be generic enough to display the data that is formatted as tables or graphs.

A reason the result of this work has not been formally integrated in the EOPEN Pilot Use Cases is that it was not clear at the beginning, when the user requirements have been elicited, what could be the benefit of using Linked Data, and Linked EO Data in particular, in the use cases. Chapter 5 provides ideas for integrating Linked Data in use cases, possibly in future activities.

7 REFERENCES

[1] Bereta, K., Smeros, P. and Koubarakis, M., 2013. *Representation and Querying of Valid Time of Triples in Linked Geospatial Data*, 10th Extended Semantic Web Conference (ESWC 2013). Montpellier, France. May 26-30, 2013. http://www.strabon.di.uoa.gr/files/eswc2013.pdf

[2] Coene, Y., Gilles, M., Tran Minh, T., 2016, *EO Metadata Discovery using Linked Data*, OGC Discussion Paper 16-074r0, version 0.2.0, 25/03/2016.

[3] Das, S., Sundara, S., Cyganiak, R., 2012, *R2RM: RDB to RDF Mapping Language*, W3C Recommendation, 27/09/2012. https://www.w3.org/TR/r2rml/.

[4] Dimou, A. et al. 2014. RML: *A generic language for integrated RDF mappings of heterogeneous data*. CEUR Workshop Proceedings, 2014. https://rml.io/specs/rml/

[5] Harris, S. and Seaborne, A. 2013. *SPARQL 1.1 Query Language*, W3C, 21/03/2013, https://www.w3.org/TR/sparql11-query/.

[6] ISO/TC 211, 2012. *Geographic Information -- Metadata -- XML schema implementation -- Part 2: Extensions for imagery and gridded data*. Reference: ISO 19139-2:2012 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57104

[7] Koubarakis, M. and Kyzirakos, K. 2010. *Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010.

[8] Kyzirakos, K. et al. 2012. *Strabon: A semantic geospatial DBMS*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2012.

[9] Kyzirakos, K., Karpathiotakis M. and Koubarakis, M., 2012. *Strabon: A Semantic Geospatial DBMS*, 11th International Semantic Web Conference (ISWC 2012), Boston, USA, 11-15 November 2012. http://www.strabon.di.uoa.gr/files/iswc-strabon.pdf.

[10] Kyzirakos, K., Karpathiotakis, M., Bereta, K., Garbis, G., Nikolaou, C., Smeros, P., Giannakopoulou, S., Dogani, K., and Koubarakis, M., 2013. *The Spatiotemporal RDF Store Strabon*, 13th International Symposium on Spatial and Temporal Databases (SSTD 2013), Munich, Germany, August 21-23, 2013. http://www.strabon.di.uoa.gr/files/sstd2013-demo.pdf

[11] Kyzirakos, K., Manegold, S. (CWI), Vlachopoulos, I., Savva, D., and Koubarakis, M. (NKUA), 2014. *GeoTriples: a Tool for Publishing GeospatialData as RDF Graphs Using R2RML Mappings*, TerraCognita 2014. http://event.cwi.nl/terracognita2014/terra2014_3.pdf

[12] Kyzirakos, K. et al. 2018. *GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings*. Journal of Web Semantics. (2018). DOI:https://doi.org/10.1016/j.websem.2018.08.003.

[13] Open Geospatial Consortium, 2014. *Earth Observation Metadata profile of Observations & Measurements*, Reference: OGC 10-157r4 Version 1.1, 10/06/2014 http://www.opengeospatial.org/standards/om

[14] Open Geospatial Consortium, 2011, *OGC GeoSPARQL - A Geographic Query Language for RDF Data*, Reference OGC 11-052r4, version 1.0, 10/09/2012 https://www.ogc.org/standards/geosparql.

[15] Open Geospatial Consortium, 2015. *OGC Observations and Measurements – JSON implementation*, reference: OGC 15-100r1, published 09/12/205, https://portal.opengeospatial.org/files/64910

Appendix A. Mapping Definitions

Not a single mapping definition may fit all the missions and product types. Indeed, this happens when necessary values are not provided in the input data. In that case, the values and the related triples must be hard-coded in the mapping rules. This is the case, for example, for creating the "Platform" object using products metadata received from the Umbrella Application. Because the results do not contain this information, a mapping definition file must be created for each platform (or mission) and the space missions and sensors information (e.g. "SENTINEL-1") must be hard-coded in it.

A second reason is that different sensors produce different types of products (e.g. radar and optical images) and for each type there is a series of specific properties, such as the polarisation of the radar signal and the cloud coverage in optical images.

A.1. Sentinel-1 (SAR) Product Metadata Mapping Definition (RML)

Table 15 contains the RML-encoded mapping rules for converting products metadata obtained from the Umbrella Application into Linked Data. The mapping rules intentionally omit the quicklook related objects (provided separately in Appendix A.2, page 59) as these contain `null` values in the search result document.

Table 15 – Sentinel-1 Product Mapping Rules

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix rrx: <http://www.w3.org/ns/r2rml-ext#> .
@prefix rrxf: <http://www.w3.org/ns/r2rml-ext/functions/def/> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix mail: <http://example.com/mail#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix geof: <http://www.opengis.net/def/function/geosparql/> .
@prefix eop: <http://www.opengis.net/eop/2.1#> .
@prefix opt: <http://www.opengis.net/opt/2.1#> .
@prefix sar: <http://www.opengis.net/sar/2.1#>
@prefix om: <http://www.opengis.net/om/2.0#> .
@prefix ical: <http://www.w3.org/2002/12/cal/ical#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix oml: <http://def.seegrid.csiro.au/ontology/om/om-lite#> .
@prefix mrss: <http://search.yahoo.com/mrss/> .
@prefix ows: <http://www.opengis.net/ows/2.0#> .
@prefix xlink: <http://www.w3.org/1999/xlink#> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix gmd: <http://www.isotc211.org/2005/gmd> .
@prefix gml: <http://www.opengis.net/gml/3.2#> .
@prefix gcmd: <http://gcmdservices.gsfc.nasa.gov/kms/concept/> .
@prefix media: <http://search.yahoo.com/mrss/> .

<#Observation>
    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI
    rr:subjectMap [
        rr:template "http://www.w3.org/ns/dcat#Dataset/{uuid}"
    ];
    # Hard-coded triple: <subject> rdf:type dcat:Dataset
    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object dcat:Dataset
    ];

    # Hard-coded triple: <subject> rdf:type oml:Observation
    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object oml:Observation
    ];

    # Hard-coded: <subject> eop:sensorType "RADAR"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:sensorType;
        rr:object "RADAR"
    ];

    # Triple: <subbject> dct:description "{uuid}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate dct:description;
        rr:objectMap [
            rr:template "EO Product record {uuid}, with filenname {filename} from the Umbrella
Application (source: {source})";
            rr:datatype xsd:string
       ]
    ];

    # Triple: <subject> dct:identifier "{identifier}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate dct:identifier;
        rr:objectMap [
            rml:reference "identifier"
       ]
    ];

    # Triple: <subject> dct:spatial "{geom_footprint}"^^geo:wktLiteral
    rr:predicateObjectMap [
        rr:predicate dct:spatial;
        rr:objectMap [
            rrx:function rrxf:asWKT;
            rrx:argumentMap([rml:reference "geom_footprint"]);
            rr:datatype geo:wktLiteral;
        ];
    ];

    # Triple:
    # <subject> eop:processingDate "{ingestion_date}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:processingDate;
        #rr:predicate dct:issued;
        rr:objectMap [
            rml:reference "ingestion_date";
            rr:datatype xsd:dateTime
       ]
    ];

    # Triple: <subject> eop:productType "{poduct_type}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:productType;
        rr:objectMap [
            rml:reference "product_type";
            rr:datatype xsd:string
       ]
    ];

    # Triple: <subject> eop:orbitDirection "DESCENDING"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:orbitDirection;
        rr:objectMap [
            rml:reference "pass_direction";
            rr:datatype xsd:string
       ]
    ];
    # Link to Phenomenon Time object (see Table 4, below)
    # Triple: <subject> oml:phenomenonTime <#PhenomenonTime>
    rr:predicateObjectMap [
        rr:predicate oml:phenomenonTime;
        rr:objectMap [
            rr:parentTriplesMap <#PhenomenonTime>
        ];
    ];

    # Link to the Process object (see Table 5, below)
    # Triple: <subject> oml:procedure <#Process>
    rr:predicateObjectMap [
        rr:predicate oml:procedure;
        rr:objectMap [
           rr:parentTriplesMap <#Process>
        ];
    ];

    # Link to the Result object (see Table 8, below)
    # Triple: <subject> oml:result <#Result>
    rr:predicateObjectMap [
        rr:predicate oml:result;
        rr:objectMap [
           rr:parentTriplesMap <#Result>
        ];
    ];
.

<#PhenomenonTime>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{source}/{uuid}/phenomenonTime"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object oml:TemporalObject
    ];

    # Triple: <subject> ical:dtstart "{sensing_date}"^^xsd:dateTime
    rr:predicateObjectMap [
        rr:predicate ical:dtstart;
        rr:objectMap [
            rml:reference "sensing_date";
            rr:datatype xsd:dateTime
        ]
    ];

    # Triple: <subject> ical:dtend "{sensing_date}"^^xsd:dateTime
    # The Umbrella App. does not provide the sensing end timestamp.
    rr:predicateObjectMap [
        rr:predicate ical:dtend;
        rr:objectMap [
            rml:reference "sensing_date";
            rr:datatype xsd:dateTime
        ]
    ];
.

<#Process>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];
    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{source}/{uuid}/process"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object oml:Process
    ];

    # Triple: <subject> eop:orbitDirection "{pass_direction}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:orbitDirection;
        rr:objectMap [
            rml:reference "pass_direction";
            rr:datatype xsd:string
       ]
    ];

    # Triple: <subject> eop:swathIdentifier "{swath}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:swathIdentifier;
        rr:objectMap [
            rml:reference "swath";
            rr:datatype xsd:string
       ]
    ];

    # Triple: <subject> eop:orbitNumber "{orbit_number}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:orbitNumber;
        rr:objectMap [
            rml:reference "orbit_number";
            rr:datatype xsd:integer
       ]
    ];

    # Hard-coded: <subject> eop:sensorType "RADAR"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate eop:sensorType;
        rr:object "RADAR"
    ];

    # Triple: <subject> sar:polarisationMode "IW"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate sar:polarisationMode;
        rr:objectMap [
            rml:reference "sensor_operational_mode";
            rr:datatype xsd:string
       ]
    ];

    # Triple: <subject> sar:polarisationChannels "VV VH"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate sar:polarisationChannels;
        rr:objectMap [
            rml:reference "polarization";
            rr:datatype xsd:string
       ]
    ];

    # Link to Platform object
    # Triple: <subject> eop:platform <#Platform>
    rr:predicateObjectMap [
       rr:predicate eop:platform;
       rr:objectMap [
            rr:parentTriplesMap <#Platform>;
       ];
    ];

    # Link to the Instrument object
    # Triple: <subject> eop:instrument <#Instrument>
    rr:predicateObjectMap [
       rr:predicate eop:instrument;
       rr:objectMap [
            rr:parentTriplesMap <#Instrument>;
        ];
    ];
.

<#Platform>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the Platform URI (from NASA GCMD concepts )
    rr:subjectMap [
        rr:constant gcmd:007c3084-89db-458e-8387-14e192b6cb8e;
    ];

    # Hard-coded: <subject> skos:altLabel "SENTINEL-1"^^xsd:string
    # Use other RML templates to convert Sentinel-2, etc. metadata
    rr:predicateObjectMap [
        rr:predicate skos:altLabel;
        rr:object "SENTINEL-1"
    ];
.

<#Instrument>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the Instrument URI (from NASA GCMD concepts )
    rr:subjectMap [
        rr:constant gcmd:1c53d85e-3792-4081-9748-192fd3140aa6;
    ];

    # Triple: <subject> skos:altLabel "SAR-C SAR"^^xsd:string
    # Use other RML templates to convert Sentinel-2, etc. metadata
    rr:predicateObjectMap [
        rr:predicate skos:altLabel;
        rr:objectMap [
            rml:reference "instrument";
            rr:datatype xsd:string
        ]
    ];
.

<#Result>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{source}/{uuid}/result"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object rdf:Class
    ];

    # Link to Media Content object (see Table 9, below)
    # Triple: <subject> media:group <#Content>
    rr:predicateObjectMap [
       rr:predicate media:group;
       rr:objectMap [
            rr:parentTriplesMap <#Content>;
       ];
    ];
.

<#Content>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{url_download}"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object media:Content
    ];

    # Hard-coded media type
    # <subject> media:type "application/binary"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:type;
        rr:object "application/binary"
    ];

    # Hard-coded media expression
    # <subject> media:expression "full"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:expression;
        rr:object "full"
    ];

    # Triple: <subject> media:fileSize "{size}"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:fileSize;
        rr:objectMap [
            rml:reference "size";
            rr:datatype xsd:string
        ]
    ];
.
```

A.2. Quicklook Object Properties Mapping Definition (RML)

Table 16 contains the rules to be added to the main rules, above, to add quicklook information to the generated Linked Data.

Table 16 – Quicklook Objects Mapping Rules

To be inserted in the `<#Result>` mapping graph:

```
    # Link to Media Quicklook object (see Table 10, below)
    # Triple: <subject> media:group <#QuicklookContent>
    rr:predicateObjectMap [
       rr:predicate media:group;
       rr:objectMap [
            rr:parentTriplesMap <#QuicklookContent>;
       ];
    ];
```

Quicklook Content and Category:

```
<#QuicklookContent>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{quicklook}"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object media:Content
    ];

    # Hard-coded: <subject> media:type "image/jpeg"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:type;
        rr:object "image/jpeg"
    ];

    # Hard-coded: <subject> media:medium "image"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:medium;
        rr:object "image"
    ];

    # Hard-coded: <subject> media:expression "sample"^^xsd:string
    rr:predicateObjectMap [
        rr:predicate media:expression;
        rr:object "sample"
    ];

    # Link to Media Category object
    # Triple: <subject> media:category <#QuicklookCategory>
    rr:predicateObjectMap [
       rr:predicate media:category;
       rr:objectMap [
            rr:parentTriplesMap <#QuicklookCategory>;
       ];
    ];
.

<#QuicklookCategory>

    rml:logicalSource [
        rml:source "${input_file}";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$.results"
    ];

    # Specify the subject URI and RDF type
    rr:subjectMap [
        rr:template "{source}/{uuid}/result/quicklook/category"
    ];

    rr:predicateObjectMap [
        rr:predicate rdf:type;
        rr:object media:Category
    ];

    # Hard-coded: <subject> rdf:about "http://[...]/1.0#QUICKLOOK"
    rr:predicateObjectMap [
        rr:predicate rdf:about;
        rr:object "http://www.opengis.net/spec/EOMPOM/1.0#QUICKLOOK"
    ];
.
```