Reinforcing the AI4EU Platform by Advancing Earth Observation Intelligence, Innovation and Adoption

D4.2: Semantic search and discovery tools

Grant Agreement ID	101016798	**Acronym**	AI4COPERNICUS
Project Title	Reinforcing the AI4EU Platform by Advancing Earth Observation Intelligence, Innovation and Adoption		
Start Date	01/01/2021	**Duration**	36 Months
Project URL	https://ai4copernicus-project.eu/		
Contractual due date	30/6/2022	**Actual submission date**	30/6/2022
Nature	DEM	**Dissemination Level**	PU
Author(s)	Manolis Koubarakis, Eleni Tsalapati, Dharmen Punjani, Despina-Athanasia Pantazi (UoA)		
Contributor(s)	-		
Reviewer(s)	Iraklis Klampanos (NCSR-D)		

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 101016798.

Document Revision History *(including peer reviewing & quality control)*

Version	Date	Changes	Contributor(s)
v0.1	15/03/22	Introduction	Despina Pantazi
v1.0	10/06/22	First version (pending review by EC)	All

Executive Summary

This deliverable of WP4 (Implementation, customisation, integration and testing) stems from Task 4.3 (Implementation of the semantic catalogue and the semantic search and discovery functionality). In this deliverable, we present the implementation of the semantic catalogue to enable the semantic search of the EO knowledge included in the AI4EU resources, the EO data of CREODIAS, and the bootstrapping services and resources created in WP5. Moreover, we discuss the implementation of the Question Answering engine EarthQA, which is developed over the CREODIAS SPARQL endpoint. This engine can be used to retrieve satellite metadata served by the CREODIAS SPARQL endpoint using simple language, therefore making it easier for less technical users to discover data of interest.

Table of Contents

List of Figures

List of Tables

List of Terms & Abbreviations

Abbreviation	Definition
BSR	Bootstrapping Services and Resources
CO	Copernicus Ontology
EO	Earth Observation
ESA	European Space Agency
GRD	Ground Range Detection
KG	Knowledge Graph
MSI	Multispectral Instrument
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
QA	Question Answering
SAR	Synthetic Aperture Radar
SLC	Single Look Complex
QA	Question Answering
RDF	Resource Description Framework
W3C	World Wide Web Consortium

1 Introduction

This is the second deliverable of WP4 (Implementation, customisation, integration and testing) and, more specifically, Task 4.3 (Implementation of the semantic catalogue and the semantic search and discovery functionality).

1.1 Purpose and Scope

The purpose of this deliverable is to present the implementation of the semantic catalogue we designed in Task 3.2 (Design of the semantic catalogue and the semantic search and discovery functionality). The semantic catalogue is developed using Semantic Web technologies, and consists of the Copernicus ontology, the bootstrapping services and resources knowledge graph and the CREODIAS SPARQL endpoint ontology. The Question Answering engine *EarthQA* targets the CREODIAS SPARQL endpoint. Using *EarthQA*, a user can retrieve satellite metadata included in the CREODIAS SPARQL endpoint, by posing a natural language question to the engine. For example, if a user asks "*Find Sentinel-1 GRD images that show airports (and areas around) in Spain.*", *EarthQA* would generate the SPARQL Query, as shown below, and by executing this SPARQL query over CREODIAS SPARQL endpoint return the ID of all the images that are Sentinel-1 GRD images, and also contain airports in Spain to the user. The returned data is in the RDF format.

```
SELECT distinct ?title ?geom ?airport {
{
  SERVICE <http://dbpedia.org/sparql>
  {
    SELECT ?airport {
      ?airport a <http://schema.org/Airport> .
      ?airport <http://dbpedia.org/ontology/location>
           <http://dbpedia.org/resource/Spain> .
      }
  }
}
?feature a <http://ws.creodias.eu/metadata/feature> .
?feature <http://ws.creodias.eu/metadata/attribute#mission>
     <http://ws.creodias.eu/metadata/mission/Sentinel-1> .
?feature <http://ws.creodias.eu/metadata/attribute#productType>
     <http://ws.creodias.eu/metadata/productType/GRD> .
?feature <http://ws.creodias.eu/metadata/attribute#title> ?title .
?feature <http://ws.creodias.eu/metadata/attribute#geometry> ?geom
.
?hex <http://ws.creodias.eu/metadata/attribute#feature> ?feature .
?hex <http://ws.creodias.eu/metadata/object/airport> ?airport .
}  LIMIT 100
```

1.2 Approach for Work Package and Relation to other Work Packages and Deliverables

Work package WP4 (Implementation, customisation, integration and testing) started on M4 and ends on M24 of the project. It is led by partner CF with the collaboration of partners NCSR-D, UoA, TAS, ECMWF and UNITN. WP4 demonstrates the usability of the solution by the reference test and the use cases selected in the open calls (WP6).

WP4 has the following five tasks:

- Task 4.1 Integration of AI4EU platform with CREODIAS/WEKEO (M4-M12, lead: CF, contributor: TAS). The technical contribution of this task is the configuration of the environment to accommodate the requirements identified in the WP2.
- Task 4.2 Integration of tools for transformation, querying, interlinking and federating big linked geospatial data (M4-M12, lead: UoA). The technical contribution of this task is the integration of the linked data suite, developed by UoA, to the platform.
- Task 4.3 Implementation of the semantic catalogue and the semantic search and discovery functionality (M4-M12, lead: UoA, contributor: NCSR-D). The technical contribution of this task is the implementation of the semantic catalogue designed in Task 3.2.
- Task 4.4 Machine learning models for EO (M4-M12, lead: UNITN, contributors: NCSR-D, ECMWF). The technical contribution of this task is the identification and integration of different supervised machine learning techniques and models, taking into account the inputs from WP2.
- Task 4.5 Testing and operation of bootstrapping services (M7-M18, lead: CF, contributors: NCSR-D, UoA). The technical contribution of this task is the availability of dedicated environments for the use cases.

The present deliverable D4.2 is the second deliverable of WP4 and contains the contributions of the project in Task 4.3.

The implementation of the architecture and software components in WP4 are designed in WP3 (Technical position and architecture). WP3 started in M1 and will end on M30. It is led by UoA with the participation of partners NCSR-D, TAS, CF and UNITN.

The following tasks of WP3 are relevant to WP4:

- Task 3.1 Architecture specification, tools and components (M1-M18, lead: UoA, contributors: NCSR-D, TAS, CF, UNITN). The technical contribution of this task is the development of the software architecture of the project with a specific emphasis to interfacing with the AI4EU platform, CREODIAS and WEkEO.
- Task 3.2 Design of the semantic catalogue and the semantic search and discovery functionality (M1-M9, lead: UoA, contributor: NCSR-D). The technical contributions of this task are the development of a question answering engine for discovering Copernicus data, and the development of the Copernicus Knowledge Graph.

The following deliverable of WP3 is relevant to WP4:

- D3.1 Architecture, semantics and discoverability report (M18, R, PU, UoA). This deliverable describes the complete architecture of the cloud environment used. It also provides the design of the semantic catalogue and the semantic search and discovery functionality of AI4Copernicus.

1.3 Organization of the Deliverable

The rest of the deliverable is organized as follows. Section 2 describes the components of the semantic catalogue we developed, while in Section 3 we discuss the implementation of the Question Answering (QA) engine *EarthQA* that is developed over the CREODIAS SPARQL endpoint. In Section 4, we provide a summary of this deliverable.

2 The Semantic Catalogue

In this section we discuss the components of the semantic catalogue, as defined in deliverable 3.1. The purpose of the semantic catalogue is to enable the semantic search of the Earth observation knowledge. Hence, as described in Section 3, EarthQA targets the semantic catalogue, which contains all the information we need for the AI4EU and the AI4Copernicus services and resources, and the EO data of CREODIAS. In particular, it contains the Copernicus Ontology (CO), the Ontology of the CREODIAS SPARQL endpoint, and the metadata of the bootstrapping services and resources of AI4Copernicus. In the following sections, we describe each one of these components.

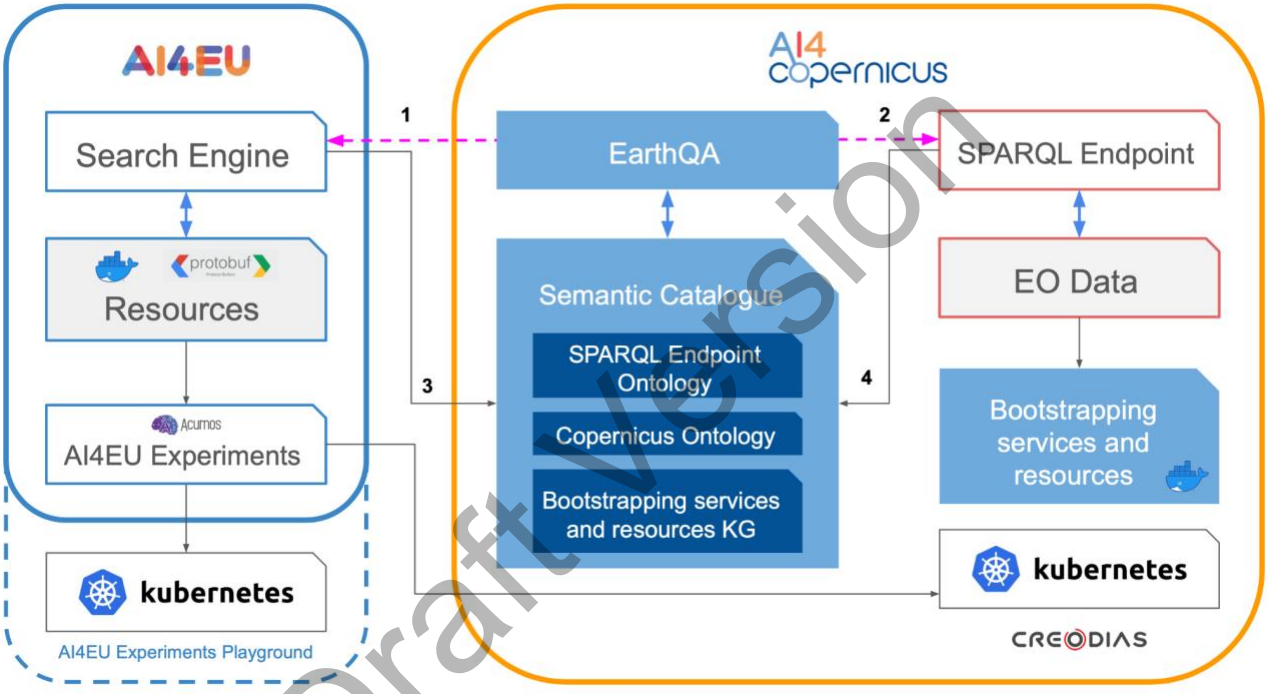

Figure 2.1: The AI4Copernicus architecture

2.1 CREODIAS Semantic Data

CREODIAS data can be queried using either the EO Search API or SPARQL. The EO Search API is backed by the EO Data Finder which conforms to OpenSearch[1] standard. Data sets are organised in so-called collections, corresponding to various satellites. A query may search for data in all collections, or in one particular collection only. Queries can be executed as HTTP GET calls, and provide outputs both in JSON and XML formats. The SPARQL interface is a W3C standard, Linked Data endpoint allowing RDF data to be retrieved and manipulated, based on the specialised, well-developed, semantic graph database Allegrograph; this interface is connected to the CREODIAS EO Browser, but can also be easily used by any third party SPARQL clients. The metadata about the collected data from different satellites are converted to RDF data using the ontology. The overview of ontology is shown in figure

[1] opensearch.org

2.2. The SPARQL endpoint can be found at https://sparql.creodias.eu:20035/#/repositories/creodias/.

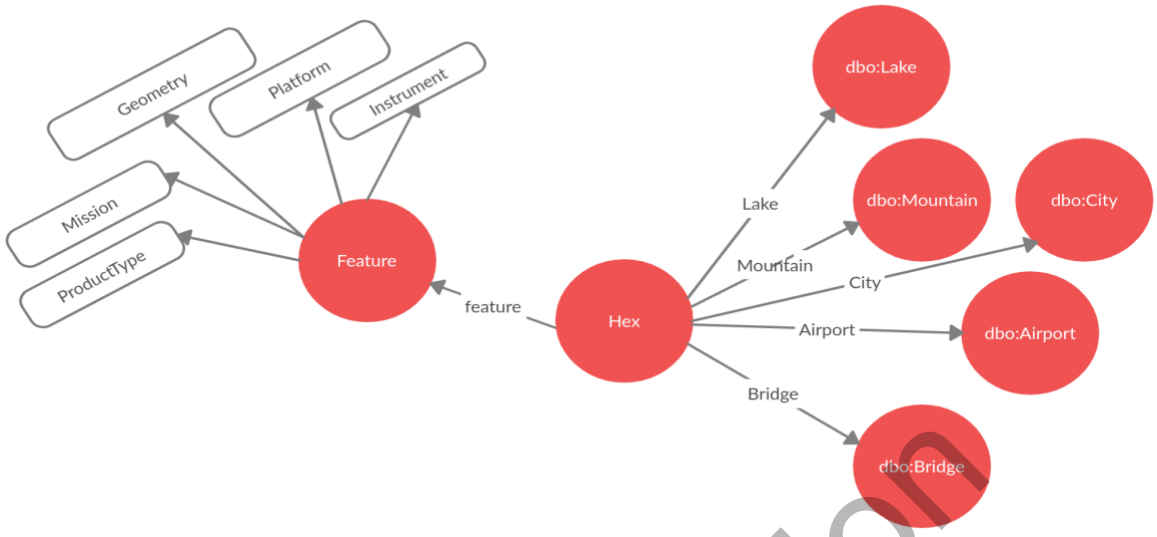

Figure 2.2: Ontology Overview of the CREODIAS SPARQL Data

2.2 The Copernicus Ontology

The Copernicus ontology is described in detail in D3.1. The scope of the Copernicus ontology is to capture general knowledge about Satellite Remote Sensing and its applications, to capture knowledge about EO datasets as well as about finer-detail geospatial and temporal aspects of the data. CO is expressed in the W3C Web Ontology Language (OWL)[2] and it contains 465 classes and nearly 1600 axioms (some of them imported from external ontologies). It is openly available[3].

The main domains included in CO are listed below.

D1. General knowledge about Satellite Remote Sensing and its applications
D2. Knowledge about EO programmes and specific satellites, e.g. Copernicus and the Sentinels
D3. Knowledge about EO datasets
D4. Geospatial and temporal knowledge
D5. Knowledge about publications on the domain

The top-level part of CO is presented in figure 2.3.

Knowledge for D1-D3 is collected from domain experts, technical documentations (e.g., [VBJ+20]), ESA and CREODIAS websites, and the documents OGC 17-003r2 [C20], OGC 17-084r1 [C21]. For the geospatial knowledge we use DBpedia (which is also used from CREODIAS) and for the representation of the temporal knowledge we use the time ontology[4]. To represent knowledge about publications on the domain, the ontology of the Open Research Knowledge Graph [ASV+21] is reused.

[2] https://www.w3.org/OWL/
[3] http://pyravlos-vm5.di.uoa.gr/CopernicusOntology_BootstrappingKG.zip
[4] https://www.w3.org/TR/owl-time/

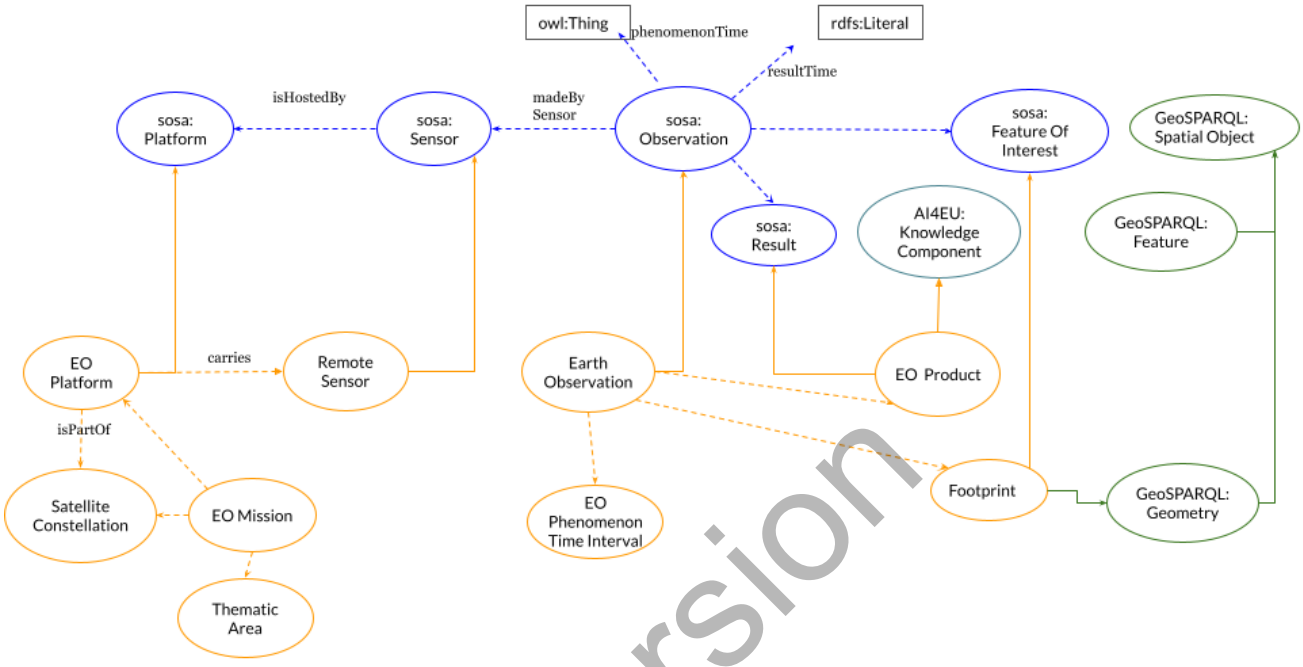

Figure 2.3: Top-level part of the Copernicus Ontology

2.3 The Bootstrapping Services and Resources KG

According to D5.1, the bootstrapping services and resources provided by AI4Copernicus are the following:

The datasets:

- TimeSen2Crop
- VectorDataOfHumanFeatures
- EnergyDataset

The services:

- Sentinel-1 GRD pre-processing
- Sentinel-1 SLC pre-processing
- Sentinel-2 pre-processing
- Sentinel-1 Change detection – Amplitude Change Detection and Multi-temporal Coherence
- Sentinel-2 Change detection
- Deep Network for pixel-level classification of S2 patches,
- Harmonization of pre-processed Time Series of Sentinel-2 data,
- Long Short-Term Memory Neural Network for Sentinel-2,
- Pre-Trained Long Short-Term Memory for GeoTIFF samples for Agriculture
- Probabilistic downscaling of CAMS air quality model data

We have created the KG Bootstrapping Services and Resources[5] in which the aforementioned services/datasets and their metadata are described in detail. The KG is mapped to both AI4Copernicus and AI4EU.

The classification of the datasets and the services in Bootstrapping Services and Resources KG (BSR KG) is presented in figure 2.4:

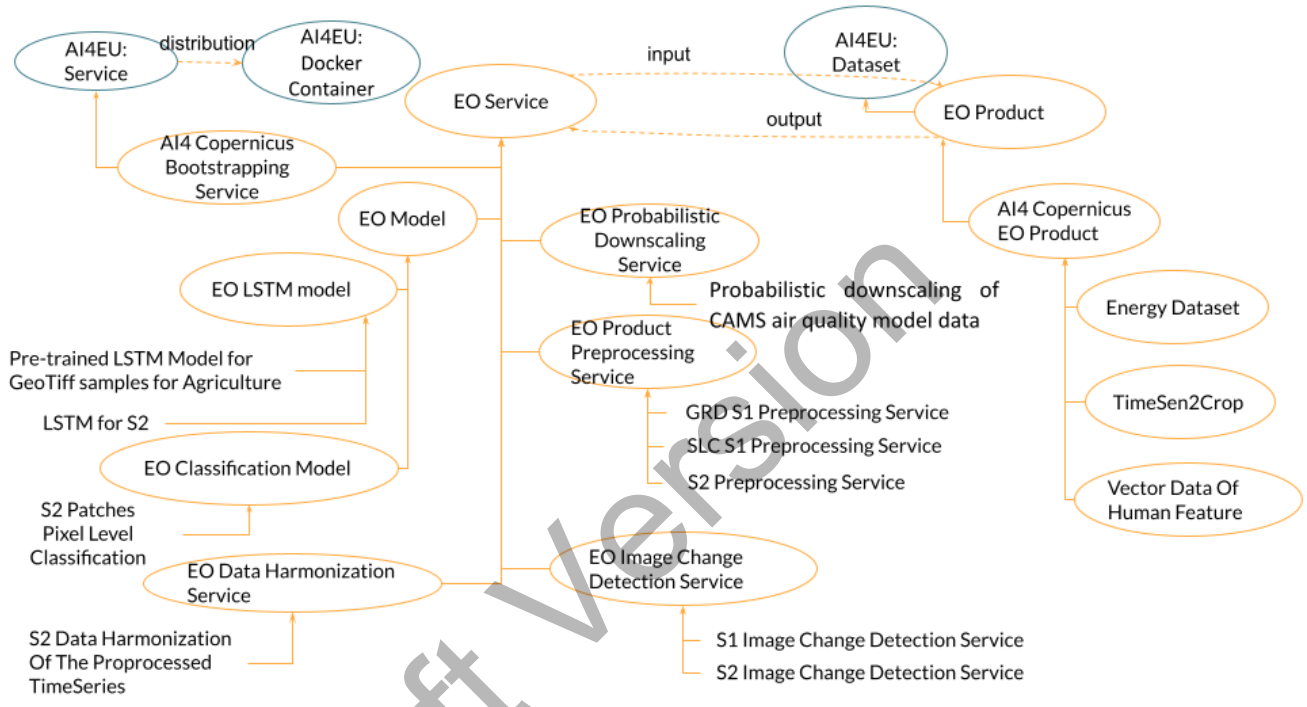

Figure 2.4: Top-level part of the Bootstrapping Services and Resources KG

Notice that, although it is not depicted in the graph for visual clarity, the bootstrapping services, such as *bsr:LSTM for S2,* are instances of both generic classes (*bsr:EO LSTM model,* in this case) expressing the type of service provided, and of the class *bsr:AI4CopernicusBootstrappingService*. Additionally, the description, the input and the output of each service is represented formally. Finally, as it is depicted in figure 2.4, all upper level classes are mapped to the respective classes of the AI4EU ontology.

3 The Implementation of the Question Answering Engine EarthQA

In this section we will discuss the implementation of *EarthQA,* the Question Answering (QA) engine, that is developed over the CREODIAS SPARQL endpoint. The *EarthQA* engine is developed using the Qanary methodology [B+16, BKDL17] and the Frankenstein platform [S+18].

[5] http://pyravlos-vm5.di.uoa.gr/CopernicusOntology_BootstrappingKG.zip

3.1 Related Work

There is currently no published work on a question answering engine for satellite data like *EarthQA*. H2020 project SnapEarth (https://snapearth.eu/) is developing a search engine for satellite data called EarthSearch. Currently, there are no publications on EarthSearch on the web site of the project. EarthSearch is developed by the French company QWANT (https://www.qwant.com/) in collaboration with other partners of SnapEarth.

In 2019, the European Space Agency published a call for the development of "*A New Generation of Linked Earth Observation Data Search Engine*". To the best of our knowledge, no project has been funded under that call although the call has been visionary and shares many goals with our work on EarthQA. This year (2022), the Φ-Lab of the European Space Agency published a call for "*Demonstrator precursor Digital Assistant interface for Digital Twin Earth (DTE)*" where a question answering engine is also envisioned.

3.2 The QA Pipeline

Qanary is a lightweight component-based QA methodology for the rapid engineering of QA pipelines [B+16, BKDL17]. Frankenstein [S+18] is the most recent implementation of the ideas of Qanary; this makes it an excellent framework for developing reusable QA components and integrating them in QA pipelines. Frankenstein is built using the Qanary methodology developed by Both et al. [B+16] and uses standard RDF technology to wrap and integrate existing standalone implementations of state-of-the-art components that can be useful in a QA system. The Qanary methodology is driven by the knowledge available for describing the input question and related concepts during the QA process. Frankenstein uses an extensible and flexible vocabulary [S+16] for data exchange between the different QA components. This vocabulary establishes an abstraction layer for the communication of QA components. While integrating components using Frankenstein, all the knowledge associated with a question and the QA process is stored in a process-independent knowledge base using the vocabulary. Each component is implemented as an independent micro-service implementing the same RESTful interface. During the start-up phase of a QA pipeline, a service registry is automatically called by all components. As all components are following the same service interface and are registered to a central mediator, they can be easily activated and combined by developers to create different QA systems.

Thus we take advantage of the Frankenstein framework to create QA components which collectively implement the QA pipeline reusing the components from GeoQA [P+18] and adding some more components that build complete QA pipeline over CREODIAS SPARQL endpoint.

The *EarthQA* pipeline for CREODIAS consist of following components :

- Concept Identifier (reused from GeoQA)
- Instance Identifier (TagMeDisambiguate ,reused from GeoQA)
- Spatial relation Identifier (reused from GeoQA)
- Property Identifier (reused from GeoQA)

- Date Identifier (HeidelTime tool)
- ProductType Identifier
- Other Metadata Identifier
- Query Generator (reused from GeoQA)

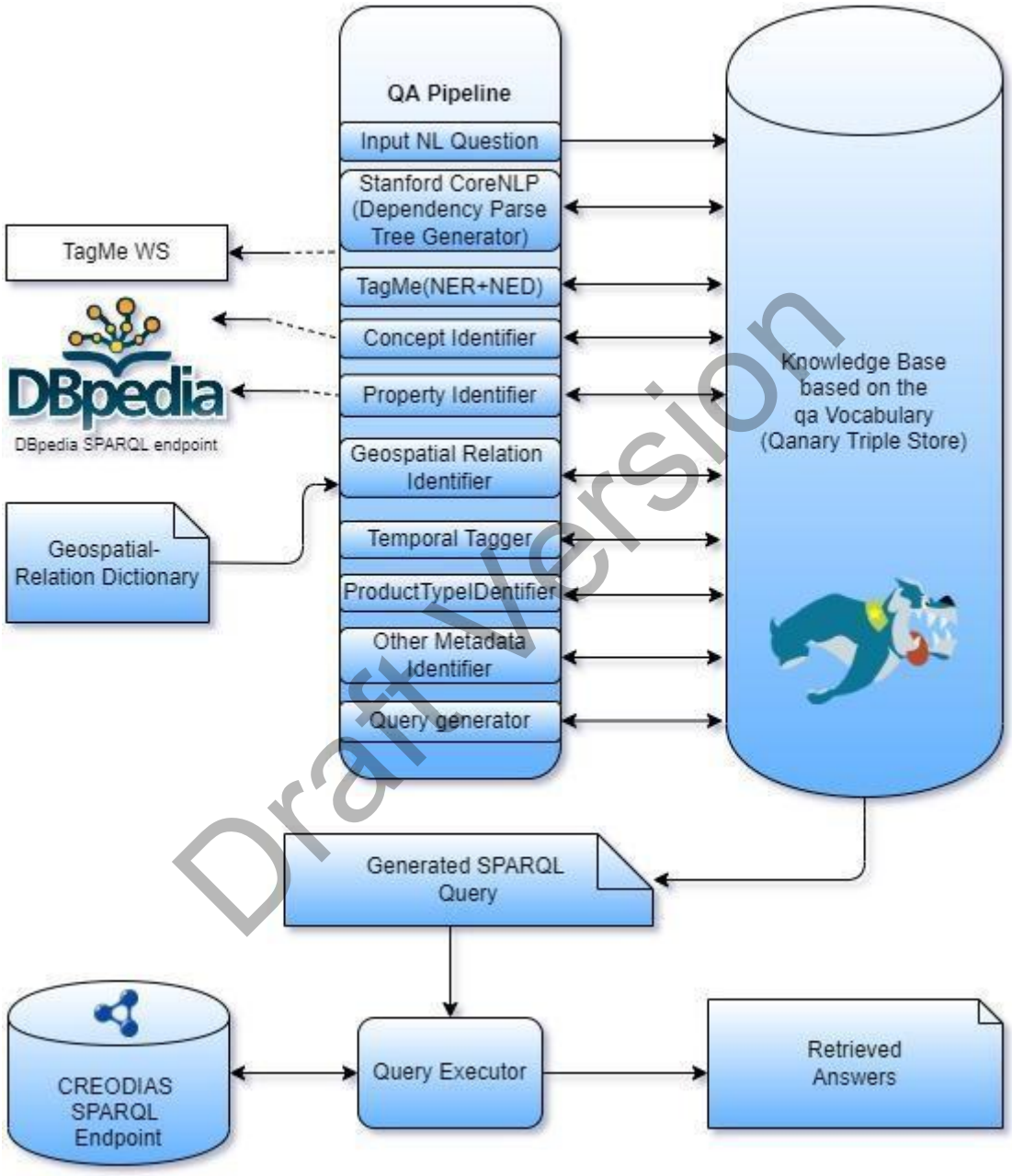

Figure 3.1: Architecture of Implementation of EarthQA

It is to be noted that components reused from GeoQA have been modified or used as it is per requirement of task. Now we will give a detailed description of all the components of the *EarthQA* pipeline.

3.2.1 Dependency Parse Tree generator

This component carries out part-of-speech tagging and generates a dependency parse tree for the input question using the Stanford CoreNLP software. The dependency parse tree is produced in CoNLL-U format [N+16].

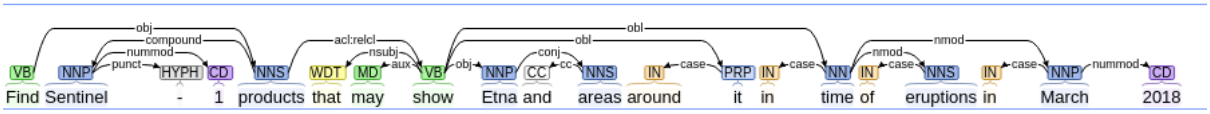

Figure 3.2: The dependency parse tree

3.2.2 Concept Identifier

This component is reused from the GeoQA [P+18] and modified as per requirement of the task and target data. The concept identifier module identifies the types of features specified by the user in the input question and maps them to the corresponding classes in the DBpedia ontology. We use the equivalent ontology-oriented term concept for a feature type. For example, if the input question is "*Find Sentinel-1 GRD images that show mountains and areas around in Spain*", then the term "*mountains*" is identified as a feature type and mapped to the class `dbo:Mountain` in the DBpedia ontology. The matching classes are found using string matching on the labels of the classes (the Java library function java.util.regex.Pattern.matcher() is used) together with lemmatization from Stanford CoreNLP and synonyms from Wordnet. The CREODIAS contains limited classes from DBpedia. Thus, we only disambiguate the identified classes to the following list of classes available on CREODIAS SPARQL endpoint : *Settlement, BodyOfWater, Building, River, Mountain, Sea, NaturalEvent, Volcano, Region, Country, ProtectedArea, Bridge, Airport, EthnicGroup, Event, Earthquake, Lake, Island, NaturalPlace, Dam, MountainRange, AdministrativeArea, Glacier, City*. In its final stage, the concept identifier annotates the appropriate node of the dependency parse tree with its results.

3.2.3 Instance Identifier

This component is reused from the GeoQA [P+18]. The next useful information to be identified in an input question is the features mentioned. These can be e.g., the country Ireland or the city Dublin or the river Shannon etc. We use the equivalent ontology-oriented term instance(s) for features here. Once instances are identified, they are mapped to DBpedia resources using the entity recognition and disambiguation tool TagMeDisambiguate [FS10]. Take the example question "*Find time series (December 2017/2016) of Sentinel-1 images that show the Svartisen glacier in Norway*". The term "*Svartisen glacier*" and "*Norway*" is the identified instance (feature) and it is disambiguated to the wikipedia link and we get DBpedia resource `dbr:Svartisen_Glacier` and `dbr:Norway` by owl:sameAs link from DBpedia Virtuoso endpoint2. In its final stage, the instance identifier annotates the appropriate node of the dependency parse tree with its results.

3.2.4 Spatial relation Identifier

This component is reused from the GeoQA [P+18]. Geospatial questions such as the ones targeted by GeoQA almost always include a qualitative geospatial relation such as "*borders*" or "*within*". The current implementation supports the 14 geospatial relations shown on table 3.1.

These include some topological, some distance and some cardinal direction relations [EF91, F92, SK21]. Table 3.2 gives a dictionary of the various synonyms for these relations that can appear instead of them in a question. The semantics of topological relations are as in the dimensionally extended 9-intersection model [CF96].

Like the previous modules, this module first identifies geospatial relations in the input question, and then maps them to a spatial function of the GeoSPARQL vocabulary, or a data property with a spatial semantics in the DBpedia ontology. As we have already discussed in the introduction, DBpedia contains limited explicit or implicit geospatial knowledge using latitude/longitude pairs, and properties such as dbp:northeast for cardinal direction relations or class-specific properties such as dbo:city (e.g., for class dbr:River). We make use of qualitative geospatial knowledge from DBpedia expressed using the data properties just mentioned (although this knowledge is rather scarce as discussed in [RJG16]). As an example, for the question "*Find sentinel images containing rivers that cross London.*", the geospatial relation "*crosses*" is identified from the verbs in the dependency tree, and it is mapped to the spatial function `geof:sfCrosses` of the GeoSPARQL vocabulary.

Table 3.1: Geospatial relation categories and relations

Category	Geospatial Relation
Topological relations	"within","crosses","borders"
Distance relations	"near","at most x units", "at least x units"
Cardinal Direction relation	"north of", " south of", "east of", "west of", "northwest of", "northeast of", "southwest of", and "southeast of"

Table 3.2: Geospatial relations and their synonyms

Geospatial relation	Synonyms in dictionary
within	In, inside, is located in, is included in
crosses	Cross, intersect

near	Nearby, close to, around
borders	is/are at the border of, is/are at the outskirts of, at the boundary of
North of	Above of
South of	below
East of	To the right
West of	To the left

In its final stage, the geospatial relation identifier annotates the appropriate node of the dependency parse tree with its results.

3.2.5 Property Identifier

This component is reused from GeoQA [P+21]. The property identifier module identifies attributes of features specified by the user in input questions and maps them to corresponding properties in DBpedia. To answer questions like "*Find all Sentinel-1 GRD images that show large lakes of an area greater than 100 sq km*", we need information about the area of lakes. We can retrieve this information from DBpedia. We use table 3.3 for this task. The identified concept from the concept identifier module is used to search table 3.3 to get `dbp:area` in the case of example questions mentioned before. We stress that table 3.3 contains only examples of classes, properties and values that are of interest to the example questions. In reality the table contains 33,632 entries and covers all the listed classes of DBpedia in the section concept identifier. This table has been generated by querying DBpedia and stored in different files with their class names. We use string similarity measures while searching table 3.3. In its final stage, the property identifier annotates the appropriate node of the dependency parse tree with its results.

Table 3.3: DBpedia property example

DBpedia Class	DBpedia Property	Label of property
dbo:Lake	`http://dbpedia.org/property/area`	area
dbo:Lake	`http://dbpedia.org/property/volume`	volume

dbo:Mountain	http://dbpedia.org/property/height	height
dbo:Mountain	http://dbpedia.org/property/elevation	elevation

3.2.6 Temporal Tagger

This module identifies temporal keywords and annotates it with appropriate date. For this we have used the already available temporal tagger/date parser tool. We compared the tools HeidelTime [SJG15] and Hawking date parser (*https://github.com/zoho/hawking*) and selected the one which gives most/all of the correct date from the input question. Both tools are open source and their code is available on GitHub. In the table below we show the output of the tools for a given input question.

Table 3.4: Comparison of HeidelTime and Hawking Date Parser tool

Question	HeidelTime(Rule based)	Hawking Date parser(ML based)
Find Sentinel-1 products that may show Etna and areas around it in time of eruptions in March 2018	<TIMEX3 tid="t4" type="DATE" value="2018-03">March 2018</TIMEX3>	Start : 2018-03-01T00:00:00.000+02:00 End : 2018-04-01T00:59:59.000+03:00
Find time series (December 2017/2016) of Sentinel-1 images that show Svartisen glacier in Norway	<TIMEX3 tid="t4" type="DATE" value="2017-12">December 2017</TIMEX3>/ <TIMEX3 tid="t2" type="DATE" value="2016">2016</TIMEX3>	can not find dates
Find Sentinel-3A Water Full Resolution (WFR) products with the data collected in January 2018	<TIMEX3 tid="t3" type="DATE" value="2018-01">January 2018</TIMEX3>	Start : 2018-01-01T00:00:00.000+02:00 End : 2018-01-31T23:59:59.000+02:00
Find Sentinel images taken during the summer months of 2020 which cover Athens, Greece and can be used to	<TIMEX3 tid="t2" type="DATE" value="XXXX-SU">the summer</TIMEX3> <TIMEX3 tid="t1"	Start : 2020-05-01T01:00:00.000+03:00 End : 2020-07-01T00:59:59.000+03:00

study oceans.	type="DATE" value="2020">2020</TIMEX3>	
Find Sentinel images that can be used to detect burned down villages in the Rakhine State of Myanmar during August and September 2017	<TIMEX3 tid="t5" type="DATE" value="2017-08">August</TIMEX3> <TIMEX3 tid="t4" type="DATE" value="2017-09">September 2017</TIMEX3>	Start : 2016-08-01T01:00:00.000+03:00 End : 2016-09-01T00:59:59.000+03:00

In its final stage, the temporal tagger annotates the appropriate node of the dependency parse tree with its results.

3.2.7 Product Type Identifier

This component identifies metadata about Mission, Platform and Product type from the input question. We can tell using product type, that specific product type can be available from a specific Mission. E.g., *Water Full Resolution products are observed through Sentinel-3*. The list of product types, its platform and mission is listed in the table 3.5 below.

Table 3.5: List of Product type as per Mission

Mission	Platform	Product Type	Description of Product Type
Sentinel-1	S1A	GRD	Ground Range Detected
Sentinel-1	S1A	SLC	Single Look Complex
Sentinel-1	S1A	RAW	Raw
Sentinel-1	S1A	OCN	Ocean
Sentinel-1	S1B	GRD	Ground Range Detected
Sentinel-1	S1B	SLC	Single Look Complex

Sentinel-1	S1B	RAW	Raw
Sentinel-1	S1B	OCN	Ocean
Sentinel-2	S2A	L1C	Level-1C
Sentinel-2	S2B	L1C	Level-1C
Sentinel-3	S3A	EFR	output during EO processing mode for Full Resolution
Sentinel-3	S3A	ERR	output during EO processing mode for Reduced Resolution
Sentinel-3	S3A	WFR	Water Full Resolution
Sentinel-3	S3A	WRR	Water Reduced Resolution
Sentinel-3	S3A	LAN	Land Products
Sentinel-3	S3A	LFR	Land Full Resolution
Sentinel-3	S3A	LRR	Land Reduced Resolution
Sentinel-3	S3A	LST	Land Surface Temperature
Sentinel-3	S3A	RBT	Radiance and Brightness Temperature
Sentinel-3	S3A	SRA	--
Sentinel-3	S3A	WAT	Water Products
Sentinel-3	S3A	WST	Water Single Temperature

We first try to find if there exists a product type in the input question. For this we produce n-grams of the question and find the string similarity using different string similarity methods between these n-grams and product type listed in table 3.5. We start with 3-grams then bigrams and monograms. For 3-grams and bigrams we use token based string similarity algorithm Jaccard string similarity score and for monogram we use edit distance based string similarity algorithm JaroWinkler string similarity. The threshold used for similarity is 90% for all the cases. We search product types with 3-grams, bigrams and monograms in order. So if we find some product type with any of the n-gram with at

least 90% threshold we stop there and select that product type and appropriate Mission. Take the example question "*Find Sentinel-3A Water Full Resolution products with the data collected in January 2018*" to understand the process. So here we first generate following 3-grams :

[*Find Sentinel-3A Water :: Sentinel-3A Water Full :: Water Full Resolution :: Full Resolution products :: Resolution products with :: products with the:: with the data :: the data collected :: data collected in :: collected in January :: in January 2018*]

We find string similarity scores and get the "*WFR*" and Mission Sentinel-3 and platform S3A from the question that fulfils the condition of threshold above 90% and we stop the process. After getting the product type and Mission we take monograms and add a platform if the question contains it. If we do not get any product type mentioned in the input question then we look for the mission and platform mentioned in the input question. We use Jaro-Winkler string similarity with a 90% threshold. Thus based on identified product type/ mission/ platform we annotate appropriate nodes of dependency parse tree with it.

3.2.8 Other Metadata Identifier

This module will identify other metadata about the feature/product like cloud coverage, orbit direction, processing level, swath etc. We follow a similar method as in the product type identifier module. In this component we search for the metadata of the image that are as following: *Cloud cover, Orbit direction, Swath, nssdcIdentifier, polarisation, processingFacilitySite, resourceSize, orbitNumber, processingLevel, processingFacilitySoftwareName, productIdentifier, processingFacilityName, processingFacilityCountry, resolution, sensorMode, missionTakeid, organisationName*. Take the example question "*Find Sentinel-2 MSI products with cloud cover below 10%*". Here we identify cloud cover from the question as metadata of a feature. The appropriate node of the dependency parse tree is annotated with the identified metadata in the end.

3.2.9 Query Generator

This component is reused from GeoQA [P+18, P+21]. This module takes output generated by all the previous modules and based on a set of rules generates the SPARQL query that can be executed over the CREODIAS SPARQL endpoint.

GeoQA identifies a list of question patterns from which we have shown only those that are of interest to us in table 3.6. In addition to the question pattern that has been identified in GeoQA [P+18, P+21] we also have considered the pattern "*CP*" also shown in the table below. In this table C stands for "*concept*", I for "*instance*" , R for "*geospatial relation*" , P for "*property*" following the terminology we have mentioned above. For each pattern, the table gives an example question and the corresponding SPARQL query template. The query templates contain slots (strings starting with an underscore) that can only be identified when an example question is encountered and will be completed by the query generator, as shown below.

Table 3.6: Query Pattern Templates

Pattern	Example Question	Template
CRI	Mountains of Spain	`select ?x where { ?x rdf:type _Concept. ?x _Relation _Instance. }`
CRIRI	Villages in Rakhine State of Myanmar	`select ?x where { ?x rdf:type _Concept1. ?x _Relation1 _Instance1. _Instance1 _Relation2 _Instance2. }`
IRI	Svartisen glacier in Norway	`ask { _Instance1 _Relation2 _Instance2. }`
CP	Lakes with area greater than 100 sq km	`select ?x where { ?x rdf:type _Concept. ?x _Property ?property }`

For each input question, the slots in the template are replaced by the query generator with the output of the previous modules, to generate a SPARQL query. For example, for the question "*Find Sentinel-1 GRD images from Spain that show mountains and areas around them*", the identified pattern is *CRI*. The question pattern is identified by searching the dependency parse tree in which the nodes have been annotated with the results of the concept, instance, property and geospatial relation identifier modules presented above. We walk through the parse tree with inorder traversal and identify the question pattern. If the question does not follow any of the patterns no query will be generated.

The appropriate templates are selected from table 3.6, their slots are filled with the resources identified earlier and the corresponding SPARQL query is generated. See example below.

Question : Mountains of Spain
SPARQL :

```
select ?x where {
         ?x rdf:type dbo:Mountain.
         ?x dbo:Country dbr:Spain.
}
```

As DBpedia does not contain spatial properties with GeoSPARQL vocabulary the solution is to have the query generator take into account class and property information from the ontologies of DBpedia. This is illustrated by the SPARQL query above where we make use of the fact that the property dbo:country is used in DBpedia to refer to the country containing a mountain. To implement this strategy we keep a table with three columns which contains triples of the form domain-property-range for each property in DBpedia. Some example rows can be seen in the table 3.7 below.

Table 3.7: domain-property-range Table

Domain	Property	Range
Mountain	within	http://dbpedia.org/ontology/country
Mountain	within	http://dbpedia.org/ontology/city
River	crosses	http://dbpedia.org/ontology/Bridge
River	crosses	http://dbpedia.org/ontology/city
Lake	within	http://dbpedia.org/ontology/country
Lake	within	http://dbpedia.org/ontology/city

Just to add the note that GeoQA generates SPARQL as well as GeoSPARQL queries over OSM, GADM and DBpedia. While for our use we only need the part of the code that generates DBpedia SPARQL queries and only that has been reused in the component.
Now after checking if the input question contains any of the patterns from table 3.6, we need to add the triples that have been identified by other modules. In addition to the question pattern we have template triples for Hex, Mission, date and Other Metadata as well.

The Hex triples. As per the ontology of CREODIAS rdf data all the entities from DBpedia are connected to spatially indexed Hexagon instances. Thus we need to see if the question is asking for sentinel images that might contain specific DBpedia entities. If yes we need to add the triples that would get us instances of images that are connected to the spatially indexed Hex instances that are associated with DBpedia entities. Thus Hex triples are added if and only if following conditions are satisfied.

1) If any of the question patterns in table 3.6 is identified.

2) If there exists no question pattern from table 3.6 but it contains I(Instance) identified in the input question.

If condition 1 is satisfied we add generated SPARQL query from pattern and following Hex triples.

```
?hex <http://ws.eodias.eu/metadata/attribute#feature> ?x .
?hex ?pred ?instance.
```

Now if condition 2 is satisfied then we add the following triples.

```
?hex <http://ws.eodias.eu/metadata/attribute#feature> ?x .
?hex ?pred _Instance_ .
```

For example, "*Find Sentinel-1 products that may show Etna and areas around it in time of eruptions in March 2018*." the following exact triple would be added to the SPARQL query.

```
?hex <http://ws.eodias.eu/metadata/attribute#feature> ?x .
?hex ?pred <http://dbpedia.org/resource/Mount_Etna> .
```

The Mission triples. Mission template triples contain the following template.

```
?x <http://ws.eodias.eu/metadata/attribute#mission> _mission_ .
?x <http://ws.eodias.eu/metadata/attribute#platform> _platform_ .
?x <http://ws.eodias.eu/metadata/attribute#productType>
_productType_ .
```

Based on the identified Mission, platform and product type from the input question in the product type identifier module we replace the slots in the above template. We just use the triple for which the component has been identified only. E.g., "*Find all Sentinel-1 GRD images that show large lakes of an area greater than 100 sq km*". In this question identified components are mission and Product type and not the platform. Thus following triples would be generated from the above template and added in the generated SPARQL query.

```
?x <http://ws.eodias.eu/metadata/attribute#mission>
<http://ws.eodias.eu/metadata/mission/Sentinel-1> .
?x <http://ws.eodias.eu/metadata/attribute#productType>
<http://ws.eodias.eu/metadata/productType/GRD>.
```

The Other Metadata. Here we add triples for other metadata if it has been identified from the input question. For this we straight forward add the triples with the property of identified metadata. E.g. "*Find Sentinel-2 MSI products with cloud cover below 10%*" , added triple would be following.

```
?x <http://ws.eodias.eu/metadata/attribute#cloudCover> ?cc .
filter(?cc<10 && ?cc>=0)
```

The Date triples. Date triple contains the following triple by default.

```
?x <http://ws.eodias.eu/metadata/attribute#startDate> ?date .
```

As can be seen in the Temporal tagger module that HeidleTime does not give us start date and end date as output it just annotates text with the date. Thus we identify the end date and start date from these dates in our code and add the appropriate triples in the above template. So for example the question "*Find time series (December 2017/2016) of Sentinel-1 images that show Svartisen glacier in Norway*" we would generate the following date triples.

```
 ?x <http://ws.eodias.eu/metadata/attribute#startDate> ?date .
bind(year(?date) as ?year) .  bind(month(?date) as ?month) .
 filter(?year>=2016 && ?year<=2017 && ?month=12 ) .
```

After generating all the triples we generate a SPARQL query adding all the generated triples discussed above. So for the example question "*Find Sentinel-1 products that may show Etna and areas around it in time of eruptions in March 2018*" our query generator generates the following SPARQL query.

Generated SPARQL Query :

```
select distinct ?title ?geom where {

?hex <http://ws.eodias.eu/metadata/attribute#feature> ?x .
?hex ?pred <http://dbpedia.org/resource/Mount_Etna> .
?x a <http://ws.eodias.eu/metadata/feature> .
?x <http://ws.eodias.eu/metadata/attribute#title> ?title .
?x <http://ws.eodias.eu/metadata/attribute#geometry> ?geom .
?x <http://ws.eodias.eu/metadata/attribute#mission>
<http://ws.eodias.eu/metadata/mission/Sentinel-1> .
?x <http://ws.eodias.eu/metadata/attribute#startDate> ?date .
bind(year(?date) as ?year) .  bind(month(?date) as ?month) .
filter(?year=2018 && ?month=03 ) .

} LIMIT 1000
```

3.2.10 Query Executor

This module takes the query generated from the query generator and executes it over CREODIAS SPARQL endpoint. To retrieve the results from the CREODIAS SPARQL endpoint we do HTTP GET requests at their SPARQL endpoint. The following parameters are used in HTTP GET requests.

```
query = *SPARQL Query *
queryLn= SPARQL
limit =  e.g. 100
Infer = false
returnQueryMetadata = true
```

It returns SPARQL XML as response and we parse it using Apache Jena Java API. An example HTTP GET request is the following:
"https://sparql.creodias.eu:30035/repositories/creodias?query=select+distinct+*+where+%7B+%3Fx+%3Fp+%3Fo.%7D+limit+20&queryLn=SPARQL&limit=10&infer=false&returnQueryMetadata=true".

4 Conclusions

In this deliverable, we presented the implementation and the components of the semantic catalogue we designed. Our goal is to use this semantic catalogue to enable the semantic search of the EO knowledge included in the AI4EU resources, the EO data of CREODIAS, and the bootstrapping services and resources created in WP5. Moreover, we discussed the implementation of the Question Answering engine *EarthQA*, which is developed over the CREODIAS SPARQL endpoint. User-friendly semantic searching over EO data is expected to provide users with the facility to find the resources by typing their requirements in natural language.

5 References

[ASV+21] S. Auer, M. Stocker, L. Vogt, G. Fraumann, A. Garatzogianni, ORKG: Facilitating the Transfer of Research Results with the Open Research Knowledge Graph. Research Ideas and Outcomes 7: e68513, 2021

[B+16] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, C. Lange, Qanary - A methodology for vocabulary driven open question answering systems, in: H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, C. Lange (Eds.), The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings, volume 9678 of Lecture Notes in Computer Science, Springer, 2016, pp. 625–641. URL: https://doi.org/10.1007/978-3-319-34129-3_38. doi:10. 1007/978-3-319-34129-3_38.

[BKDL17] A. Both, K. Singh, D. Diefenbach, I. Lytra, Rapid engineering of QA systems using the light-weight qanary architecture, in: J. Cabot, R. D. Virgilio, R. Torlone (Eds.), Web Engineering - 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings, volume 10360 of Lecture Notes in Computer Science, Springer, 2017, pp. 544–548. URL: https://doi.org/10.1007/978-3-319-60131-1_40. doi:10.1007/978-3-319-60131-1_40.

[C20] Yves Coene. OGC 17-003r2 - Earth Observation Dataset Metadata Vocabulary. Technical report, 2020.

[C21] Y. Coene, U. Voges, O. Barois. OGC EO Collection GeoJSON(-LD) Encoding Best Practice, 2021.

[CF96] E. Clementini, P. D. Felice, A model for representing topological relationships between complex geometric features in spatial databases, Inf. Sci. 90 (1996) 121–136. URL: https://doi.org/10.1016/0020-0255(95)00289-8.doi:10.1016/0020-0255(95)00289-8.

[EF91] M. J. Egenhofer, R. D. Franzosa, Point set topological relations, International Journal of Geographical Information Systems 5 (1991) 161–174. URL: https://doi.org/10.1080/02693799108927841. doi:10.1080/02693799108927841.

[F92] A. U. Frank, Qualitative spatial reasoning about distances and directions in geographic space, J. Vis. Lang. Comput. 3 (1992) 343–371. URL: https://doi.org/10.1016/1045-926X(92)90007-9. doi:10.1016/1045-926X(92)90007-9.

[FS10] P. Ferragina, U. Scaiella, TAGME: on-the-fly annotation of short text fragments (by wikipedia entities), in: J. Huang, N. Koudas, G. J. F. Jones, X. Wu, K. Collins-Thompson, A. An (Eds.), Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26- 30, 2010, ACM, 2010, pp. 1625–1628. URL: https://doi.org/10.1145/1871437.1871689. doi:10.1145/ 1871437.1871689.

[N+16] J. Nivre, M. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. T. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, D. Zeman, Universal dependencies v1: A multilingual treebank collection, in: N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis (Eds.), Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016, European Language Resources Association (ELRA), 2016.

[P+18] D. Punjani, K. Singh, A. Both, M. Koubarakis, I. Angelidis, K. Bereta, T. Beris, D. Bilidas, T. Ioannidis, N. Karalis, et al., Template-based question answering over linked geospatial data, in: Proceedings of the 12th Workshop on Geographic Information Retrieval, 2018, pp. 1–10.

[P+21] Dharmen Punjani, Markos Iliakis, Theodoros Stefou, Kuldeep Singh, Andreas Both, Manolis Koubarakis, Iosif Angelidis, Konstantina Bereta, Themis Beris, Dimitris Bilidas, Theofilos Ioannidis, Nikolaos Karalis, Christoph Lange, Despina-Athanasia Pantazi, Christos Papaloukas, Georgios Stamoulis. "Template-Based Question Answering over Linked Geospatial Data." CoRR abs/2007.07060, 2021.

[RJG16] B. Regalia, K. Janowicz, S. Gao, VOLT: A provenance-producing, transparent SPARQL proxy for the on-demand computation of linked data and its application to spatiotemporally dependent data, in: H. Sack, E. Blomqvist,M. d'Aquin, C. Ghidini, S. P. Ponzetto, C. Lange (Eds.), The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings, volume 9678 of Lecture Notes in Computer Science, Springer, 2016, pp. 523–538. URL: https://doi.org/10.1007/978-3-319-34129-3_32. doi:10.1007/978-3-319-34129-3_32.

[S+16] K. Singh, A. Both, D. Diefenbach, S. Shekarpour, D. Cherix, C. Lange, Qanary - the fast track to creating a question answering system with linked data technology, in: H. Sack, G. Rizzo, N. Steinmetz, D. Mladenic, S. Auer, C. Lange (Eds.), The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers, volume 9989 of Lecture Notes in Computer Science, 2016, pp. 183–188. URL: https://doi.org/10.1007/978-3-319-47602-5_36. doi:10.1007/978-3-319-47602-5_36.

[S+18] K. Singh, A. S. Radhakrishna, A. Both, S. Shekarpour, I. Lytra, R. Usbeck, A. Vyas, A. Khikmatullaev, D. Punjani, C. Lange, M. Vidal, J. Lehmann, S. Auer, Why reinvent the wheel: Let's build question answering systems together, in: P. Champin, F. L. Gandon, M. Lalmas, P. G. Ipeirotis (Eds.), Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018, ACM, 2018, pp. 1247–1256. URL: https://doi.org/10.1145/3178876.3186023. doi:10.1145/3178876.3186023.

[SJG15] Strötgen, Jannik, and Michael Gertz. "A baseline temporal tagger for all languages." Proceedings of the 2015 conference on empirical methods in natural language processing. 2015.

[SK21] S. Skiadopoulos, M. Koubarakis, Composing cardinal direction relations, in: C. S. Jensen, M. Schneider, B. Seeger, V. J. Tsotras (Eds.), Advances in Spatial and Temporal Databases, 7th International Symposium, SSTD 2001, Redondo Beach, CA, USA, July 12-15, 2001, Proceedings, volume 2121 of Lecture Notes in Computer Science, Springer, 2001, pp. 299–320.

[VBJ+20] P. Vincent, M. Bourbigot, H. Johnsen, R. Piantanida, Sentinel-1 Product Specification, Ref: S1-RS-MDA-52-7441, S-1 MPC Nomenclature: DI-MPC-PB S-1, MPC Reference: MPC-0240, ESA Unclassified, 2020.