Copernicus Core Services Interface Update

Deliverable D2.3

DATE
31 December 2022

ISSUE
2.0

GRANT AGREEMENT
no 870337

DISSEMINATION LEVEL
PU

PROJECT WEB-SITE
http://cure-copernicus.eu/

LEAD AUTHOR
Michal Opletal (GISAT)

CO-AUTHORS
Katerina Jupova,
Tomas Soukup (GISAT)

CONTENTS

1 INTRODUCTION

1.1 Purpose of the document

This document presents a technical description of the CURE Copernicus Core Services Interface (CCSI) and its implementation. The aim of this development activity is to provide a unified interface to streamline search and locate Copernicus Services products and other resources as requested by CURE cross-cutting applications. This will help automate the data input process into CURE applications and the CURE System, which will significantly enhance their replication potential for the future.

The CCSI is stored in this GitHub repository: https://github.com/gisat/ccsi, and it will constantly be updated after the delivery of this report in order to allow the efficient development of the CURE System in WP4.

This document leverages a CCSI concept introduced in D2.1, worked out into the first operational implementation. In particular, this document describes:

- the way how to query Copernicus Core Services Interface to collect the available products and metadata available
- the way Copernicus Core Services Interface implements OpenSearch standards
- parameters of Copernicus Core Services Interface response
- class and implementation of Copernicus Core Services Interface application

Described application components and their classes may be upgraded or changed during the application development and operation. For this purpose, this document is considered as living one and will be revised upon the application change requests.

1.2 Text convention

- Parameter *name* and parameter *value*, URI is represented by italic

2 COPERNICUS CORE SERVICES INTERFACE

This chapter shortly explains the concept of the CURE Copernicus Core Services Interface and how it is implemented, including specification of supported protocol, protocol standards, and access to the Interface.

2.1 Copernicus Core Services Interface concept

Products of Copernicus Services can be accessed from various access points like DIASes or CDSAPI etc. These access points differ from each other according to which Copernicus Services are hosted on the access point, how these Services can be searched and in the form of standard response.

The idea of the Copernicus Core Services Interface is intended to ease access for CURE applications and users to Copernicus Core services products such as the Sentinel satellites datasets and the Copernicus Core Services. These datasets are currently offered by various providers such as Creodias, Mundi, Onda, and others. Each provider offers a different set of products and ranges. CCSI should provide a unified interface that should allow to the user search datasets across these providers.

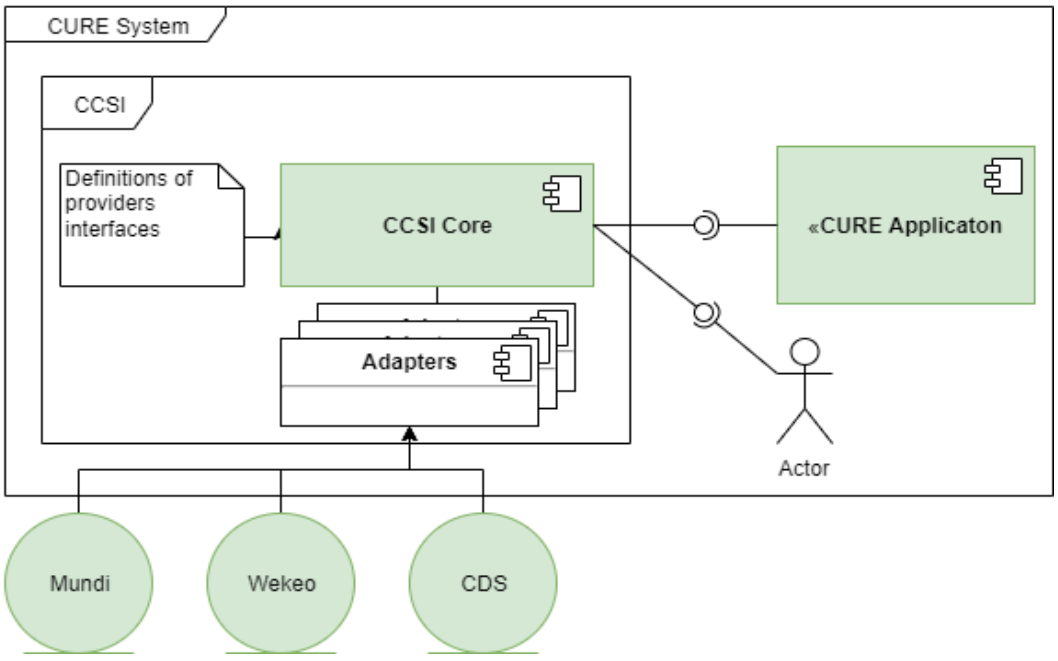

Figure 1. Position of Copernicus Core Services Interface in the CURE system.

Figure 1 shows the schema basic interaction with CCSI within the CURE system as was described in the preliminary specification of CCSI in document D2.1

2.2 Copernicus Core Services Interface implementation

Copernicus Core Services Interface is developed as a RESTfull application, providing a search API build upon the OpenSearch specification. CCSI is built as a python Flask application utilizing Marshmallow and Pydantic library for serialization and deserialization of the responses between the interfaces. CCSI is deployable as a docker container.

CCSI in its current version stateless application primary acting as an adapter between the CCSI and dataset providers interfaces. All endpoints and provided links can be requested via GET request. Authorizations and proxy ordering data are handled on the CCSI side and the user is have not to provide any additional inputs except the query parameters. Query parameters are standardized across the application and each individual endpoint.

Descriptions of the parameters and their values are accessible via description documents for each application endpoint.

CCSI provides two modes of search: overall search and resource search. Overall search return to the user for certain set parameters a total number of results for each resource that matches the input set of the parameters and links for the search over particular resources. Resource search returns to the user for the same set of the input parameters, if matched, entries with datasets metadata from the provider. In the basic workflow, the user, who is not sure which resource to search for the data, can use the overall search to find the resource with the most records, and then the user can be redirected to the resource search.

CCSI provides a response in the format of standard OpenSearch Atom XML as well as custom JSON format. Response Items are standardized. The exact number and type of response items depend on the items of resource original response.

CCSI was developed on the prerequisite of implementation and usage of OpenSearch protocol across the DIAS providers. CCSI was designed to be able automatically to ingest new resources into the CCSI by parsing the resource's description document. Description document will be translated into the CCSI definition YAML file that configures dynamic creation of the search endpoints, validation, and translation input and output parameters. Due to the inconsistency and variations in the datasets providers in the CCSI design are also implemented partially concept of the adapters when for certain providers API is written specific adapter classes that share a common interface.

In order to facilitate interaction, querying and data download via CCSI, a python client published as python library publicly on PYPI was created.

2.3 Supported protocol

Copernicus Core Services Interface support OpenSearch API protocol:

- with query options as Parameters

- with also query options as Search Terms

2.4 Protocol standards

The OpenSearch protocol implemented in CCSI follows standards defined in the following documents:

Table 1. Implemented protocol standards

Document ID	Document Name	Issue	Link
	OpenSearch	1.1	http://www.opensearch.org/Home
OGC 10-157r3	Earth Observation Metadata Profile of Observations & Measurements	1.0	https://portal.opengeospatial.org/files/?artifact_id=47040
OGC 10-032r8	OGC OpenSearch Geo and Time Extensions	1.0	http://www.opengis.net/doc/IS/opensearchgeo/1.0
OGC 10-026r8	OGC OpenSearch Extension for Earth Observation	1.0	http://docs.opengeospatial.org/is/13-026r8/13-026r8.html

2.5 Access

Copernicus Core Services Interface access in the current version is open, and the querying access point can be reached through HTTP GET queries.

3 OPENSEARCH API

This chapter is dedicated to the description of the implementation of OpenSearch specification within the Copernicus Core Services Interface. This chapter describes the API endpoints structure, search parameters, building of queries and standard responses.

3.1 Building an OpenSearch endpoint URI

Copernicus Core Services Interface is queryable with OpenSearch queries through HTTP GET requests. CCSI has two basic variations of Open Search URIs addressing the resource/products exposed by the Open Search Service:

- URI addressing all catalogues/resources registered in CCSI:

<hostname>/<path>/<response from>/search?

- URI addressing specific catalogue or resource registered in CCSI:

<hostname>/<path>/<resourse>/<response from>/search?

where:
<hostname>/<path> is service root

<response from> is the requested format of responses and has two options *json* or *atom*.

URI for response in atom format:

<hostname>/<path>/atom/search?

URI for response in json format:

<hostname>/<path>/json/search?

<resourse> is a unique name of a registered catalogue or resource. Together with the rest of URI define the endpoint specific for the selected catalogue/resource. This endpoint accepts only catalogue/resource-specific parameters

3.2 OpenSearch Description Document

Implemented OpenSearch protocol is self-descriptive. Each endpoint exposes its own description document (OSDD). Description document provides a definition of available collections and parameters and their possible values.

URI for all description documents describing search parameter for querying all catalogues/resources is a form:

<hostname>/<path>/<response from>/search/description.xml

This description document provide register of all parameters that are accept by global endpoint

URI for all description document describing search parameter for querying specific catalogues/resources is in a form:

<hostname>/<path>/<response from>/<resourse>/search/description.xml

This description document provides a register of recourse specific parameters that are accepted by endpoint.

3.3 OpenSearch query parameters

Parameters are limited to a short list of metadata filters. Concrete set on the parameters depends on the endpoint.

3.3.1 Notations

All query parameters are defined as camel notations i.e. timeEnd, sensorType …

3.3.2 Filtering catalogues and resources

Copernicus Core Services Interface providing access to various catalogues and resources. Selection of particular resources is provided by parameters:

- *resourse*
  Resourse is a metadata filter used for selection of catalogues and resources. Is an option parameter that acquires only certain values. These values are identical with *resource* 's unique name defining each endpoint. Parameter is a multi-value parameter when multiple values are separated by “,”.

- collection
  Recourse is a metadata filter used for selection of collection. Certain collection can be provided by different recourses. Is an option parameter that acquires only certain values. One *resource* can have a multiple collections. Parameter is a multi-value parameter when multiple values are separated by “,”.

3.3.3 Temporal filtering

Date metadata can be filtered through temporal filtering. It provided by two parameters:

- *timestart*
- *timeend*

Expected format of parameter is in form “yyyy-mm-ddThh:nn:ss” that can be shortened of more time precise elements.

<hostname>/<path>/<resourse>/<response from>/search?timestart=2020-12-18T12:00:00

alternatively

<hostname>/<poth>/<resourse>/<response from>/search?timestart=2020-12-18

3.3.4 Geographical filtering

Geographical metadata can be filtered by parameters:

- *geometry*

Geometry parameter accepts geometry in WKT format coordinates in decimal degrees (EPSG:4326). Accepted geometries are: polygon, linestring, point

<hostname>/<path>/<recourse>/<response from>/search?geometry=POLYGON((-4.53 29.85,26.75 29.85,26.75 46.80,-4.53 46.80,-4.53 29.85))

- *bbox*

Bbox parameter filter the data based on the geographical bounding box. Coordinates are expected in decimal degrees (EPSG:4326) in order west, south, east, north. e.g. bbox=-61.3,14.3,-60.8,14.9

*<hostname>/<path>/<recourse>/<response from>/search?*bbox=-61.3,14.3,-60.8,14.9

- lat, lon, radius
  Lat, Lon, Radius parameters have to be provided together. Lat, Lon parameters are expected in decimal degrees (EPSG:4326). Radius as a float in meters

*<hostname>/<path>/<recourse>/<response from>/search?*lat=-61.3&lon=14.3&radius=1000

3.3.5 Filtering for specific product

Specific product can be selected by providing *productid*

*<hostname>/<path>/<recourse>/<response from>/search?*productid=z_cams_c_ecmf_20200616120000_prod_fc_sfc_062_gtco3

3.3.6 Entries filtering

Received entries for given query can be filtered by parameters:

- maxrecords
  Maxrecord parameter defines the number of entries per page. expected type is integer. Default value is 50

*<hostname>/<path>/<recourse>/<response from>/search?*maxrecords=50

- startindex

Startindex parameter defines from which index the entries will be returned. Minimum value is 1

*<hostname>/<path>/<recourse>/<response from>/search?*startindex=5

3.3.7 Special parameters

- custom

Copernicus Core Services Interface is intended to provide access to various catalogues/resources. The exposed parameters for filtering and specification of products provided by single recourse may differ from others. This resource specific parameters are registered with the prefix "*custom:*" e.g. parameter for the specification of orbitderection is labelled as *custom:orbitdirection*. List of custom parameters and their specification, pattern, optional or default values is accessible via description dokuments.

<hostname>/<path>/<recourse>/<response from>/search?custom:orbitdirection= descending

- *solr*
  Some registered catalogues/resources allow use of Apache Lucene free text search. Parameter *solr* is a boolean parameter that defines the searchterm following the free text search convention. If *solr=true,* search query is injected only into the resources that supports free text search. Default value is *false*

<hostname>/<path>/<recourse>/<response from>/search?searchterm=(platformname:Sentinel-1 AND producttype:SLC AND sensoroperationalmode:SM)&solr=true

3.3.8 Query string format

Query string accepted by CCSI OpenSearch API:

- is expected in form *parameter=value*

.../<response from>/search?searchterm=value

- query string is not case sensitive. Accepted response arguments are converted to lowercase
- multiple parameters a separated by “&” letter

.../<response from>/search?searchterm=value&productid=value

- is accepted multiple choice parameters i.e parameters with multiple values. Multiple values are separated by “,” letter if the original resource accept this parameters

.../<response from>/search?collection=cams,clms

3.3.9 OpenSearch search terms

Parameter *searchterm* can query all the non-standard queryable keywords. Together with parameter *solr* can also content free text search query.

3.4 OpenSearch response

Copernicu Core Services Interface provides results in atom or json form depending on the requested endpoint.

3.4.1 Atom response.

The global answer for the requested query is embedded in a <feed> XML element. Sub elements of <feed> consisted of ten elements defining the response head. Entries are embedded by <entry> XML element.

Response head

<title> - name of the service
<author> - author of service
<id> - uuid of request query
<totalResults> - total count of all founded entries
<startIndex> - number of the first returned results, default is 1
<itemsPerPage> - number of returned entries per page, defined by maxrecords parameter. Default value of return entries is 50.

In order to help the browsing process, the OpenSearch result provides useful links through the <link> XML element.

<link rel="self"> - refers to current query
<link rel="first"> - refers to first page of requested entries
<link rel="next"> - refers to next page of requested entries
<link rel="last"> - refers to last page of requested entries

Important parameters of response entry

The requested response consisted from XML elements describing product metadata. These metadata differs between the resources. If metadata from the original resource does not match or is missing in the list of standard Copernicus Core Services Interface. They are not printed out. Following list of XML elements represents the base set for every entry.

<link rel="enclousure"> - provides link to downloadable content
<link rel="path"> - provides link to mountable location, if exists
<link rel="search"> - provides link to entry itself
<id> - provides product id from original recourse
<ccsi:status> - refers if the product is available online if this information is available

<gml: *>,<georss:*> - geographic reference in Geography Markup Language format

Example of response


```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:dc="http://purl.org/dc/elements/1.1/”
xmlns:gml="http://www.opengis.net/gml"
xmlns:ccsi="http://192.168.99.102:5000/ccsi">
        <title>Copernicus Core Service Interface search results</title>
        <subtitle>Displaying 1 results</subtitle>
        <updated>2020-12-18T00:00:00</updated>
        <author>
                <name>Copernicus Core Service Interface</name>
        </author>
        <id>8b734ed6-4917-47ea-a597-0ea010658942</id>
        <totalResults>1</totalResults>
        <startIndex>0</startIndex>
        <itemsPerPage>1</itemsPerPage>
        <Query role="request"
        searchTerms="searchterm=water&catalogue=mundi&maxrecords=1"/>
        <link rel="search" type="application/opensearchdescription+xml"
        href="http://192.168.99.102:5000/atom/search/description.xml"/>
        <link rel="self" type="application/atom+xml"
        href="http://192.168.99.102:5000/atom/search?searchterm=water
        &catalogue=mundi&maxrecords=1&collection=clms&startindex=1&page=1"/>
        <link rel="first" type="application/atom+xml"
        href="http://192.168.99.102:5000/atom/search&searchterm=water&
        catalogue=mundi&maxrecords=1&collection=clms&startindex=1&page=1"/>
        <link rel="next" type="application/atom+xml"
        href="http://192.168.99.102:5000/atom/search&searchterm=water
        &catalogue=mundi&maxrecords=1&collection=clms&startindex=1&page=1"/>
        <link rel="last" type="application/atom+xml"
        href="http://192.168.99.102:5000/atom/search&searchterm=water
        &catalogue=mundi&maxrecords=1&collection=clms&startindex=1&page=1"/>
        <entry>
                <id>1dcfa016-e904-410d-ab57-1ba1607a4587</id>
                <title>Corine Land Cover 2000 - 2006 changes (raster 100m) - version
                18, Mar. 2016</title>
                <category term="Land cover"/>
                <category term="land use"/>
                <category term="land cover"/>
                <category term="landscape"/>
                <category term="landscape alteration"/>
                <category term="Copernicus"/>
```

```
<category term="CLCC2000-2006"/>
<category term="EEA39"/>
<category term="geospatial data"/>
<category term="environment"/>
<category term="imageryBaseMapsEarthCover"/>
<link rel="enclosure"
href="https://cs-clms.obs.otc.t-systems.com/CLMS_products/
pan-European/CLC/LCC2000-2006/raster/g100_ch00_06_V18_5.zip"/>
<link rel="search" type="application/atom+xml"
href=”http://192.168.99.102:5000/atom/search&?
uid=1dcfa016-e904-410d-ab57-1ba1607a4587"/>
<dc:identifier
<dc:identifier>1dcfa016-e904-410d-ab57-1ba1607a4587</identifier>
<dc:date >2018-01-12T17:36:17Z</date>
<dc:creator>sdi.eea</creator>
<gml:Polygon srsName="EPSG:4326">
        <gml:outerBoundaryIs>
                <gml:LinearRing>
                        <gml:coordinates>
                                        -29.086205435,12.994105341
                                        -29.086205435,12.993797401
                                        -29.086664115,12.993797401
                                        -29.086664115,12.994105341
                                        -29.086205435,12.994105341
                        </gml:coordinates>
                </gml:LinearRing>
        </gml:outerBoundaryIs>
</gml:Polygon>
<published>2018-09-28T14:26:45Z</published>
<ccsi:status>ONLINE</status>
    </entry>
</feed>
```


3.4.2 Json response

Json feed representation is based on the mimicking the XML structure describe above.

The response is in the form of the array containing the Feed object. Feed object is consisting of three attributes: *entries*, *head* and *total results*

response : [{Feed}] = [{entries: [Entry], head: [Tag(Link], totaltResult: int]

Attribute *totalResult* is an integer that indicates the total number found for a certain set of the parameters

Attribute head is an array containing the objects Tag of type *link* and relations “self”, “first”, “next” and “last”. Object TAG is a JSON representation of XML tag. The meaning of the relations is the same as in the case of Atom format. These Tags are designed to help iterate through paged responses.

Attribute *entries* is an array of object Entry that is an array of Tags. Set of the Entry’s Tags defined the record. The basic structure of the Tag consists of attributes: *attrib, tag, and text* their values except the *tag* are optional. All attributes are the reflection of XML tag structure.

attrib - tag attributes

tag - the name of the tag, similar to the XML

text - text body of the tag

Attribute *attrib* holds the Attrib object htat consists of another optional attribute as: *href, rel and type.* Attribute *href* defines the URI, *rel* the object relation same as in the XML case and *type* defines URI content.

Example of the tag providing the link to the dataset:

```
{
        "attrib": {
          "href": "URI",
          "rel": "enclosure",
          "type": "application/unknown"
        },
        "tag": "link",
        "text": ""
      },
```

3.5 Response types

Following response types are supported

Table 2. Implemented response types

Response type	Description
200 Success	Successful response
201 Accepted	Request Accepted

3.6 Error handling

Table 3. Error Handling

Error type	description
400 Bad Request	Request has an invalid syntax
413 Request Entity Too Large	The request originates too many returnable hits
429 Too many requests	Too many requests on the resource endpoint
500 Internal Server Error	
501 Not Implemented	Unsupported operator
503 Service Unavailable	Service is temporarily not available
504 Gateway Timeout	Failing to produce an answer within a giving

4 COPERNICUS CORE SERVICES INTERFACE APPLICATION

In this chapter are described the processes and operations used in the Copernicus Core Service application. These processes include the implemented resources search logic, request parsing and validation, transformation operation, response parsing, and registration of resources.

4.1 Validation of request and finding relevant resources

Copernicus Core Services Interface is intended to provide access to various catalogs resources. In order to accomplish this task and not waste processing time searching over the resources that have not requested products is applied simple logic to find relevant resources.

- The incoming request is parsed into a single dictionary containing request parameters and their respective values and then check it, if the request contains only valid parameters against the central register of parameters. If any of input parameters is invalid, the process is aboard with custom error page 501 with an error message
- If the parameters are valid, in the central register of resources are taken reference on the resources that use the same set parameters as is in the input. These resources are added into the resources pool.

4.1.1 Request transformation

Validated query parameters are transformed from CCSI API query parameters into the from expected by particular resource API.

- Based on the definitions set up in the resource definition YAML fila are dynamically buildup transformations function that are in correct order applied on the parameter value and traform value into the for expecting by particular resource API.
- For certain resources was not able to apply funcional approche, because resource API defer to much from OpenSEarch specification or has one specific API. In this case base ond the dataset is applied on the input parameters adapter approach. Adapters are specialized classes written especially for particular resource API/dataset
- Translated input parameters are then pass into the Connection class that is responsible for building the request, handling the exceptions. Due to the variety of API used across the resource providers, in the CCSI is several Conection Subclasses that handle specific API types.

4.1.2 Response transformation

Incoming resource responses are parsed into the standard types representing single XML structure of Feed, Entry and Tags.

In CCSI are implemented several Parsesr able to parse XML or JSON format resource response. LXML and GDAL library was implemented to parse xml type responses and correctly handle various geographic reference formats used in resources responses. Pydantic class and response schemas build uopn this library are used to parse and transform JSON response.

Due to the large variation in the type and implemented specification of resource responses, CCSI for parsing the incoming response used the adapter when specific adapters are used for sepcyfic resources

4.2 List of registered resources

The following list are listed resources integrated into CCSI to date [2020/12/18]

- creodias - Sentinel 1 endpoint OpenSearch description
- creodias - Sentinel 2 endpoint OpenSearch description
- creodias - Sentinel 3 endpoint OpenSearch description
- mundi - Sentinel 1 endpoint OpenSearch description
- mundi - Sentinel 2 endpoint OpenSearch description
- mundi - Sentinel 3 endpoint OpenSearch description
- mundi - CLMS endpoint OpenSearch description
- ondata - Sentinel 3 endpoint OpenSearch description
- cams - ERA4 dataset Dataset description
- cds - ERA5 dataset Dataset description
- wekeo - C3S Dataset description
- wekeo - CAMS Dataset description
- wekeo - CLMS Dataset description
- wekeo - Sentinel 1 Dataset description
- wekeo - Sentinel 2 Dataset description
- wekeo - Sentinel 3 Dataset description

5 REMARKS AND ISSUES

In this chapter, there are some challenges and issues described that emerge during the development of Copernicus Core Service Interface and its practical operational use as part of WP4 development.

1.1 OpenSearch

An important prerequisite of the CCSI idea is implementation of the correct OpenSearch protocol on the side of the various DIASes. This prerequisite was found not to be fully fulfilled for various reasons, which either compromised CCSI use or generated extra effort to overcome current situation. Following chapter outlines some of the findings related to OpenSearch availability.

OpenSearch Implementation

OpenSearch is not offered by all DIASes or is not the preferable API protocol for data access. Wekeo DIAS relies purely on its own HDA API with no OpenSearch implementations. Creodias implements EOData Catalogue API that differs from OpenSearch. EOData catalogue can be accessed similarly to OpenSearch but with the price of losing some additional filtering possibilities. ONDA offers OpenSearch protocol, but it is as well not the preferable one. More powerful and better integrated access to the data can be done by using the Odata Catalog API.

OpenSearch descriptions

An important part of the OpenSearch is a description document that is describing the endpoint structure and all query parameters and their value format and optional values. This description document is autogenerated and in an ideal situation, this document can be parsed and facilitated or rather allow to automate the integration of the endpoint into the other applications. In many cases there was found that the description document is not available in the right form and therefore useful. As example, the ONDA and Sobloo DIASes (during the initial development phase) offer a description document that does not specify the format and mainly optional values of query parameters. Without knowing the optional parameters, it is impossible to integrate these OpenSearch endpoints into the CCSI.

Note: From 2023, Soleboo transforms into a purely API data access service and mentioned problem with description document shall be removed.

OpenSearch parameters inconsistency

OpenSearch protocol definitions with EO extensions are listed in section 2.4. Despite these definitions that specified which parameters can be used in OpenSearch to describe EO products metadata, during the implementations of CCSI we have faced multiple inconsistencies in use of certain parameters and their actual meaning between different DIASes OpenSearch implementations. As example, in the case of Creodias and Mundi Sentinel-2 endpoints, the parameters like *sensorMode, productType* and *platform* has a different meaning and thus return different response. A typical problem across the DIASes is the interchange of *orbitNumber* and *relativeOrbitNumber* parameters. As result, these

inconsistencies compromise any attempts for purely automated ingestion of OpenSearch protocols into the CCSI framework because setting of the new endpoint has to be done manually and always manually adapted to ensure the consistency between the CCSI description document and endpoint document.

Persistence

CCSI relies on the OpenSearch endpoints that are ingested into the application and the description of these endpoints. During the development there was found that the definitions end roots of endpoints often change over time which happened e.g. in the case of Mundi DIAS. In such a case all configurations of endpoints had to be revised and updated.

1.2 Other Issues

Missing catalogs

One of the main goals of CCSI was to provide access to full offer of Copernicus Core Services products across individual Copernicus Services. This goal wasn't possible to fulfill completely. Currently, DIASes provides the access to all Copernicus Cores Service products, but only a certain portion of them can be accessed directly via API. As example, in the case of Creodias, there is no catalog for these products and these can be accessed only via the file system. In other cases, products are available offline and have to be ordered with a long latency period.

Ordering system

Some of providers do not provide direct access to all datasets and some of them have to be ordered on request basis. Ordering system is provided logically via HTTP POST request, returning the order ID. Data access is then provided with order ID when the data is prepared by service. CCSI on other hand is stateless and from definitions all data have to be accessed via HTTP GET. To overcome this problem special proxy endpoint was implemented and order id is rewritten into the url.

Connection restriction

In some DIASes there are connection restriction applied. As example, in the case of ONA there is an issue with the restriction of max request per minute as restriction of the maximum request per minute from single IP is a basic defense mechanism against the DDOS attacks. Nevertheless, in the case of ONDA, together with the absence of an ordering system, to send a query about the order status in the testing phase often conflicted with this limit and any other further request form CCSI IP was prohibited for a certain time

Changing of API parameters

In order to effectively support WP3 applications, the CCSI implementation, beside the OpenSearch endpoints, integrated also other APIs. This has led to the complete refactoring of the CCSI in the midterm. As example, a typical representative of the non-OpenSearch endpoints is Wekeo HDA. Wekeo works as a proxy and provides the access to the data of other data providers like other DIASes or core services. Wekeo is built around the concept of adapters that provides communication between the

Wekeo and the provider. When the provider changes the API or Wekeo decides to change the adapter, the service is temporarily inaccessible and sometimes the new adapter accepts different parameters. Nevertheless, these changes happen mostly unnoticed and cause temporal inaccessibility of CCSI itself and requiring a change in the CCSI endpoint definition. There is currently no clear mechanism implemented which would allow interfaces like CCSI to cope with these changes in transparent and automated way.

Rolling and Updated Archives

Beside API implementation, there is one major blockage identified represented by rolling archives. Especially for applications requiring long temporal series of data this is a major problem for Copernicus Services operational utility as rolling archives provide input data only for certain time (often too limited) backwards (e.g. Wekeo Sentinel3). Similarly, some datasets are not periodically updated, only bulh updates are available time to time (e.g. ERA5 CAMS) causing the operational service based on such datasets is not available anytime for any time or period requested by user.

6 CONCLUSION

This document presents description of the current version of the CURE Copernicus Core Services Interface (CCSI) and its components and functionalities. The interface described here represents the updated and enhanced version of the CURE Copernicus Core Services Interface.

Current version of the Interface provides a link between the CURE system (including cross-cutting applications) and various repositories storing Sentinel satellite imageries, the Copernicus Services´ products or other (third-party) datasets which serve as inputs for CURE applications for operational demonstration. Current development status of Interface allows using it for particular searching and retrieving data resources requested by individual CURE cross-cutting applications.

The whole idea of CCSI is still valid and might be a valuable service beyond CURE application development. Nevertheless, as discussed above the development of CCSI faced the numerous challenges and obstacles resulting from the lack of the standardization of the search interfaces and responses implementation across the resource providers. Due to these challenges the original scope of CCSI functionalities was forced to be downscaled and certain planned functionalities e.g. like an automatic resource ingestion one cancelled. Instead, in order to not compromise expected support to the CURE applications, a considerable effort has been refocused on bridging inconsistencies and finding solutions to problems and new elements needed to be introduced (e.g. general and custom adapters).

As the final conclusion, a CURE lesson learned from the CCSI implementation. In order to serve real operational user needs, the Copernicus Services support needs to be further consolidated and streamlined into unified data streams offer, to overcome still often fragmented and ‘work in-progress’ situation of many current services. There is ongoing process driven by user uptake activities in current Copernicus implementation period, which shall guarantee further evolution in this direction. In addition, recent DIAS consolidation activities (e.g. CDAS) shall also contribute to the same target.