



ARKTALAS HOAVVA PROJECT

DELIVERABLE D-20: DESIGN OF THE ARKTALAS DATA ARCHIVE SYSTEM (ADAS)

Customer	ESA
Authors	Anton Korosov, Adrien Perrin, Arash Azamifard (NERSC)
Distribution	Consortium and ESA
ESA Contract Number	4000127401/19/NL/LF
Document Reference	
SoW Deliverable Reference	D-20
Version/Revision	1.2
Date of issue	3 July 2020

Approved by (NERSC)	Johnny A. Johannessen NERSC Project Manager	
Approved by (ESA)	Craig Donlon ESA Technical Officer	

Revision Change log

Issue	Date	Type	Change description
0.5	31 March 2020	Draft	Initial version
1.0	24 April 2020	Final draft	Revised
1.1	3 May 2020	1 st Final version	
1.2	3 July	2 nd Final version	Added Annex I

The Arktalas Data Archive System (ADAS): Deliverable D-20. This technical Note (TN-2) is associated with Task 2a: Arktalas Hoavva data collection and quality control. ADAS will be distributed and store priority satellite data and other complementary data sets including in-situ data and model fields covering areas north of 50 N. The ADAS will provide a means to search and retrieve data according to user queries. It will capitalize on existing data repositories residing among the partners, but also benefit from additional open and free access repositories on a case-by-case basis.

Table of Content

1. System Overview	3
1.1. Goals	3
1.2. Architecture.....	3
1.3. Workflows.....	4
1.3.1. Workflow 1: Harvest metadata	4
1.3.2. Workflow 2: Search using GUI.....	5
1.3.3. Workflow 3: Search using Python	6
2. Software components	7
2.1. Django-Geo-SPaaS	7
2.1.1. Purpose and architecture overview	7
2.1.2. UML diagram of database	8
2.2. Metadata harvesting	10
2.2.1. Purpose and architecture overview	10
2.2.2. Crawlers	10
2.2.3. Ingesters.....	10
2.2.4. Harvesters	11
3. AVS integration	12
ANNEX 1: Work progress.....	14

1. SYSTEM OVERVIEW

1.1. Goals

The main goal of the Arktalas Data Archive System (ADAS) is to facilitate search for geospatial data acquired over the high latitude seas and Arctic Ocean (all > 50 N) including satellite observations, numerical model outputs and in-situ measurements. ADAS will be integrated with the Arktalas Analysis and Visualization System (AVS) for facilitating visualization of selected datasets. Data for the Arctic Ocean over a period of at least 10+ years including the 2018/19 season will be shaping this database. The satellite sensor data to be acquired include (not exclusive): Synthetic Aperture Radar (SAR), Imaging microwave radiometry, Scatterometry, Visible optical imagery, Thermal infrared imagery, LIDAR, and Altimetry. Additional data set necessary to fulfil the aim and objectives of the Arktalas Hoavva study will, at least, include in-situ data and model output fields.

ADAS will be realized as a distributed data repository and a centralized data search interface. The data repository will comprise of the data services provided by the partners and services available from other providers including ESA Scientific Hub, ESA CCI Portal, OSI-SAF, Collaborative Ground Segments, NSIDC, etc. ADAS will provide search interfaces to access distributed data available via OpeNDAP, FTP or on local file servers at NERSC, IFREMER and ODL. The search interface will be realized as an online web form and as Python API e.g. in Jupyter Notebooks. The search engine will be available in Docker images or virtual machines that can be updated daily on the users' host machines. NERSC will keep the database updated on a central server (available as docker images). As such, the user will always have an updated search interface by provisioning his/her system on a daily basis.

The ADAS system design is presented in the following sections describing the overall architecture and workflows, the software components and the integration with AVS.

1.2. Architecture

As shown in Figure 1 the main component of ADAS is Django-Geo-SPaaS. It is an open source Python application written using Django-framework for storage and search of discovery metadata and providing web-based Graphical User Interface (GUI) and Python Application Programming Interface (API). The discovery metadata is stored in a relational database following GCMD-DIFF standard. Django Object-Relational Mapping (ORM) provides an efficient API for accessing the database. GUI uses Django ORM, displays a web-page for defining searching criteria and shows the results of search.

Django-Geo-SPaaS is complemented with Harvester, which is also a Python application for collecting discovery metadata from data repositories (e.g. ESA Scientific Hub, ESA CCI, OSI SAF, etc). Administrator launches harvesters to start collecting metadata from data repositories. Harvesters instantiate crawlers for automatic retrieval of all dataset URLs available in a data repository. Ingestors read metadata from these URLs and write it to the discovery metadata database.

A user access GUI for searching in a database and can then download the files from data repositories. Integration with AVS is realized as an application which uses Django-Geo-SPaaS for finding relevant datasets, downloading of these datasets from remote data repositories and launching AVS backend for visualization.

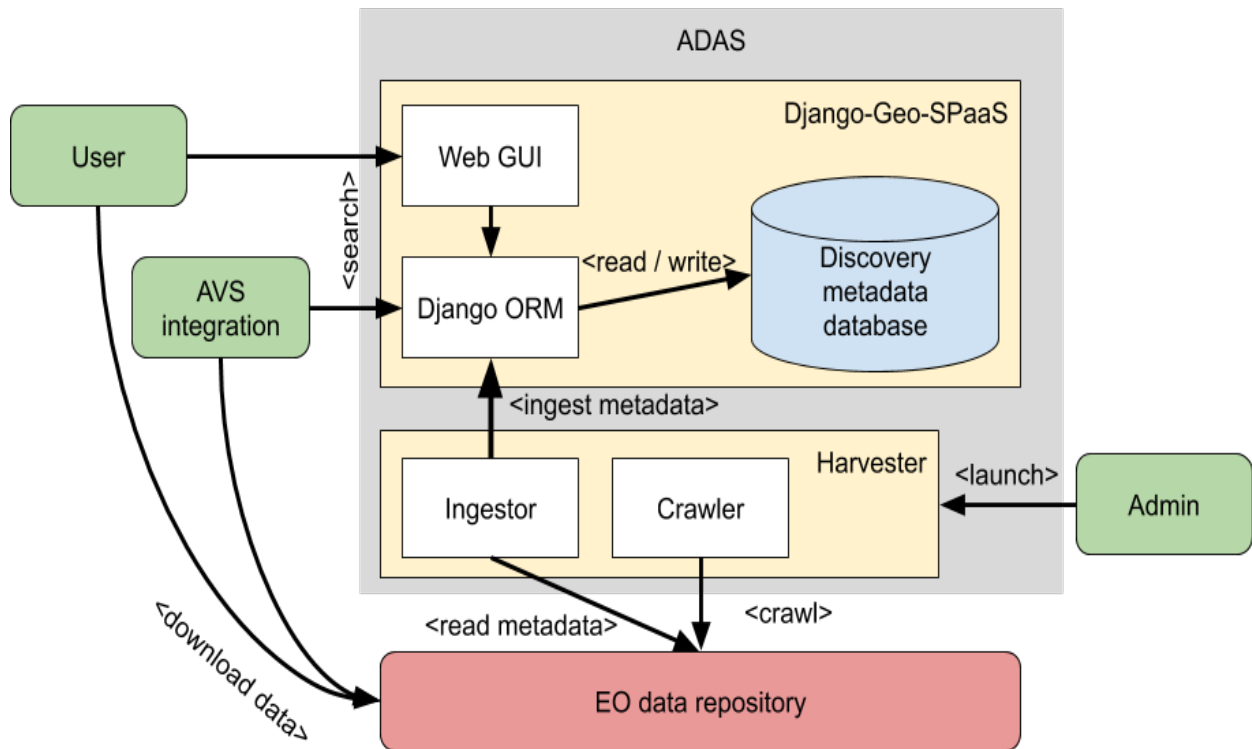


Figure 1. Overall architecture of ADAS (shown by gray block). Green blocks denote external clients, red block - distributed data repositories, yellow block - Django-Geo-SPaaS and Harvesters, blue block - centralized metadata database, white blocks - main software components. Arrow direction shows invocation sequence - arrows points from an active actor to a resource.

1.3. Workflows

The workflows presented sequentially in Figure 2, Figure 3 and Figure 4 illustrate the usage of the ADAS both from the administrator and user perspectives. These workflows justify the proposed software architecture and explain usage of each component, described in the next section. Workflows are given in chronological order, starting from filling up the database (WF1-Figure 1), followed by searching in the ADAS with a Graphical User Interface (WF2 – Figure 2) and finally searching in ADAS using Python (WF3 – Figure 3). The workflow of ADAS - AVS integration is outlined in section 3.

1.3.1. Workflow 1: Harvest metadata

Metadata harvesting is started by an administrator (not shown on the figure). According to its configuration, each harvester instantiates an ingester and crawlers adapted to the target data repository. The ingester iterates over a crawler, which lazily returns the URLs it gets while exploring the data repository. For each of these URLs, the ingester fetches the dataset metadata from the repository, normalizes it and writes it to the database. This workflow does not interact with users. This is an internal workflow which executes automatically with some time interval in order to maintain up-to-date metadata.

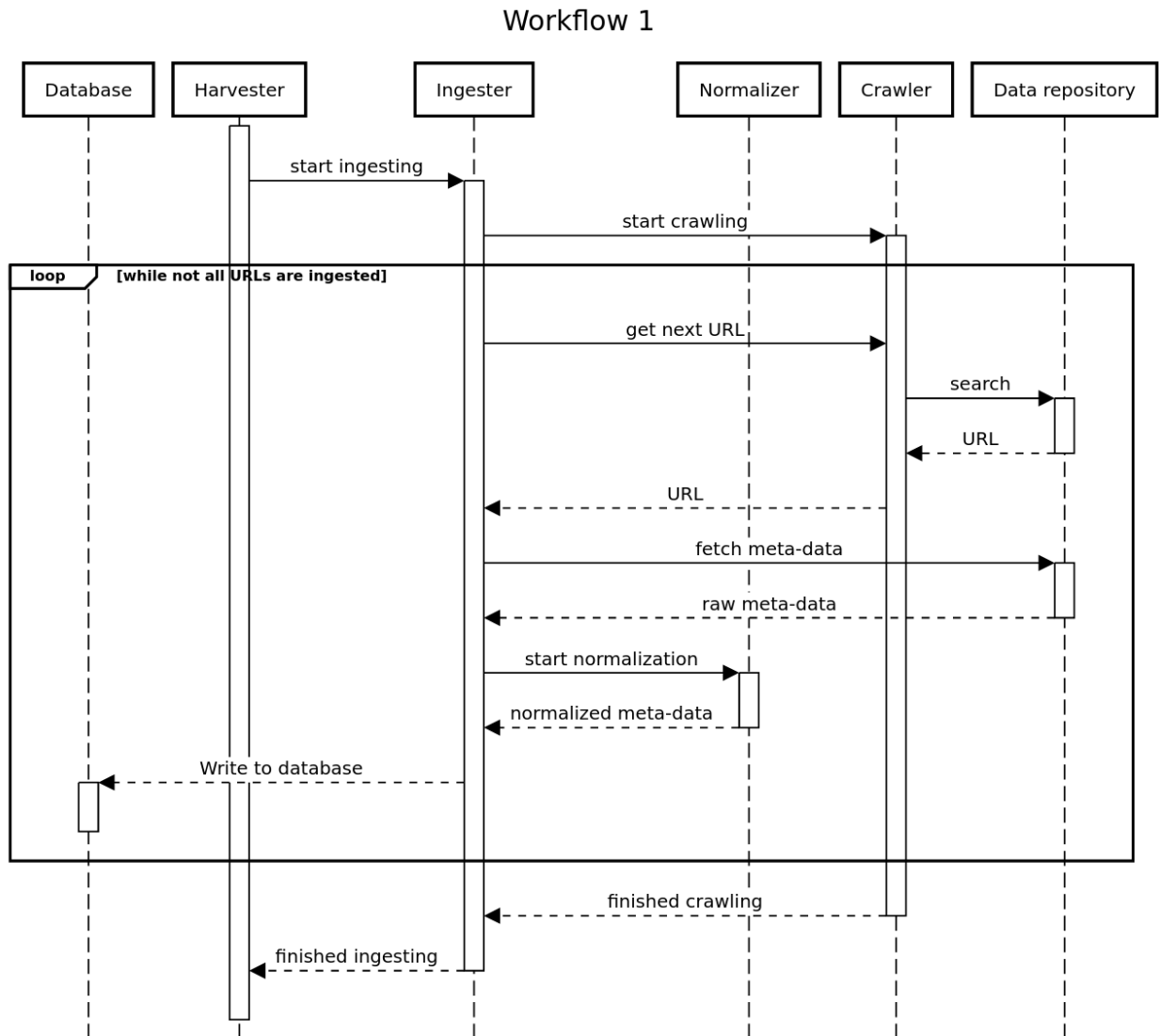


Figure 2. Sequence diagram of Workflow 1.

1.3.2. Workflow 2: Search using GUI

User interacts with GUI for finding the desired and specific dataset(s). The GUI provides the links and information of matched dataset from different data repositories. Searching the database is accomplished by Django-ORM. The database contains discovery metadata including URL for accessing the files. Django returns the list of URLs to the GUI and a user can list the URLs. Then, the user asks for downloading the desired dataset(s) among the results of the search. This is possible via the links of any dataset that is shown to the user as one part of the search result.

Workflow 2

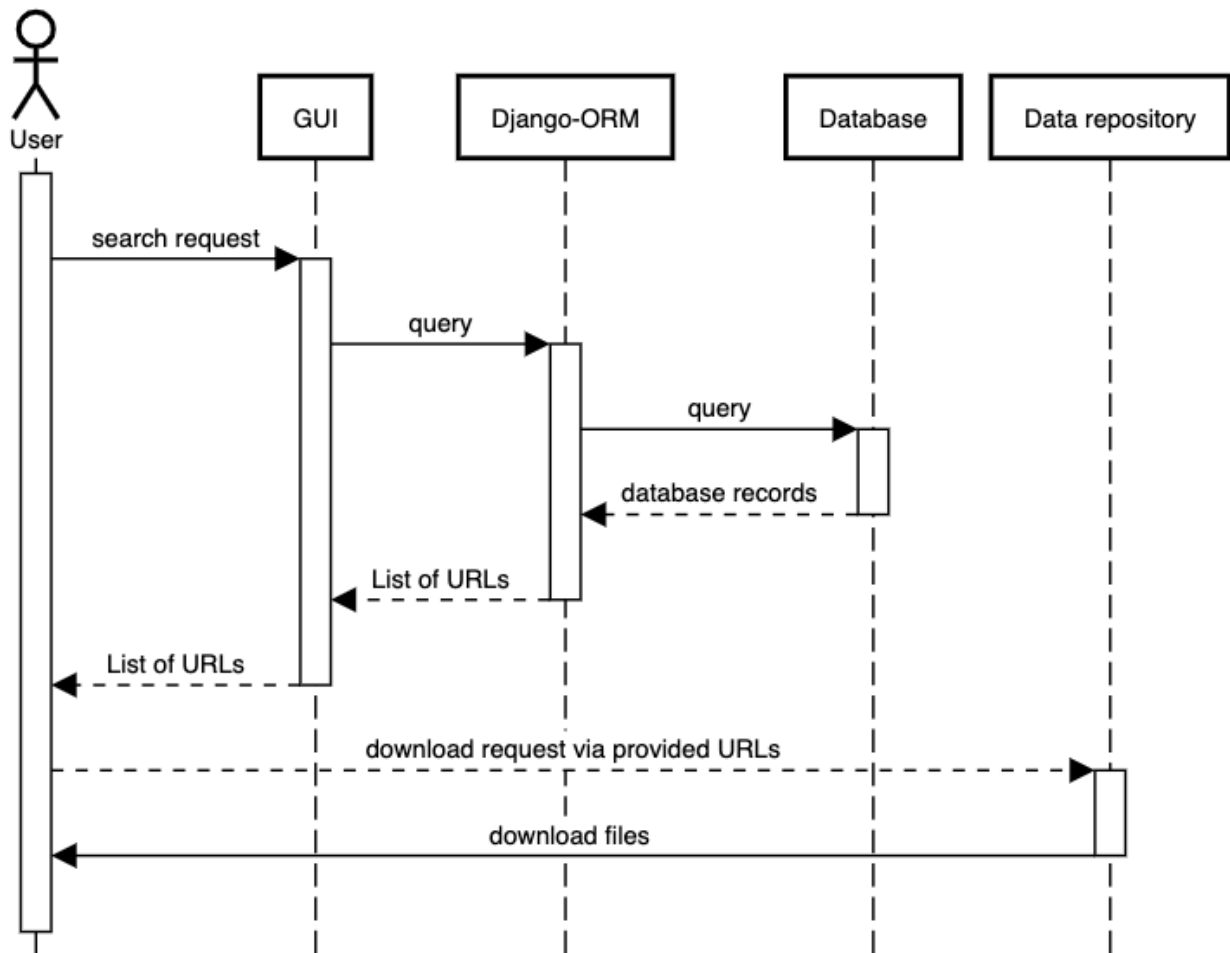


Figure 3. Sequence diagram of Workflow 2.

1.3.3. Workflow 3: Search using Python

Similar to the previous workflow, users can also write a script and execute it in order to accomplish the same task (searching and downloading some specific datasets). The main difference is that search is performed with Python language using Django-ORM directly. This allows for automation of download and processing in scripts.

Workflow 3

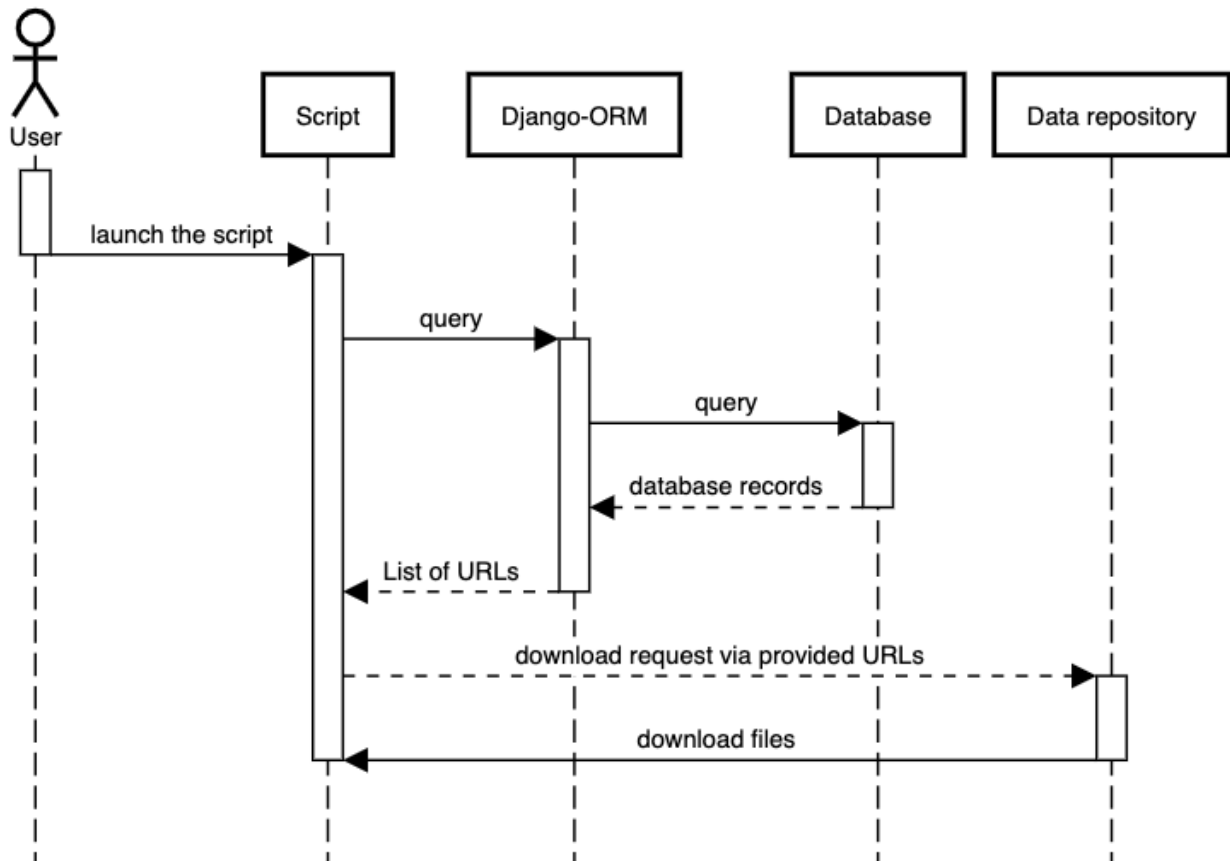


Figure 4. Sequence diagram of Workflow 3.

2. SOFTWARE COMPONENTS

2.1. Django-Geo-SPaaS

2.1.1. Purpose and architecture overview

Django Geo-SPaaS is defined by:

- A distributed catalog of discovery metadata on satellite, model and in situ data;
- Physical: a Python package API;
- Logical: a container for metadata and framework for algorithm development.

Geo-Scientific Platform as a Service (Geo-SPaaS) is a software designed as an open sourced geo scientific data management system (platform). The main purpose of Django Geo-SPaaS is to be a catalogue facilitating the search of geospatial data and enabling easy collocation of other datasets. It is designed to be a cross platform compatible software.

The Geo-SPaaS tools shall:

- Stimulate and simplify the use of EO data from satellites;
- Make geophysical data from satellites, in-situ measurements, and model simulations accessible to users in a common predefined structure;
- Facilitate the development of geophysical algorithms based on synergetic use of satellites, in-situ measurements, and models;
- Improve the routines for calibration and validation of data and models.

The Geo-SPaaS will enable researchers to cover larger areas, increase temporal and spatial resolution, and become more cost efficient.

2.1.2. UML diagram of database

The metadata is stored based on the metadata catalog of Django-Geo-SPaaS. The metadata catalog is implemented as a relational database. The metadata catalogue will act as a mechanism for storing and accessing descriptive metadata and will allow users to search and query the data items based on a chosen attribute. A Metadata catalogue contains granular metadata describing the structure, location and content of available satellite, modelling and in situ datasets. Information is stored in a relational database. The core table in the database aggregates the most common information relevant for each dataset, including:

- spatial and temporal coverage (start/stop date, geographical reference, resolution, etc);
- physical file name and access protocols (e.g. local file system or FTP or OpenDAP, etc);
- source of data (satellite/sensor or model, etc).

Figure 5 below shows the relations between the different parts of the metadata catalog. More specific metadata, which is relevant to only a few datasets, is stored in separate tables linked to the core table. The data access protocols provide seamless access to data from local and remote repositories, viewed as a single virtual distributed file system integrated with the tools, so that any Geo-SPaaS member's local data repository is virtually merged into a single shared distributed data archive.

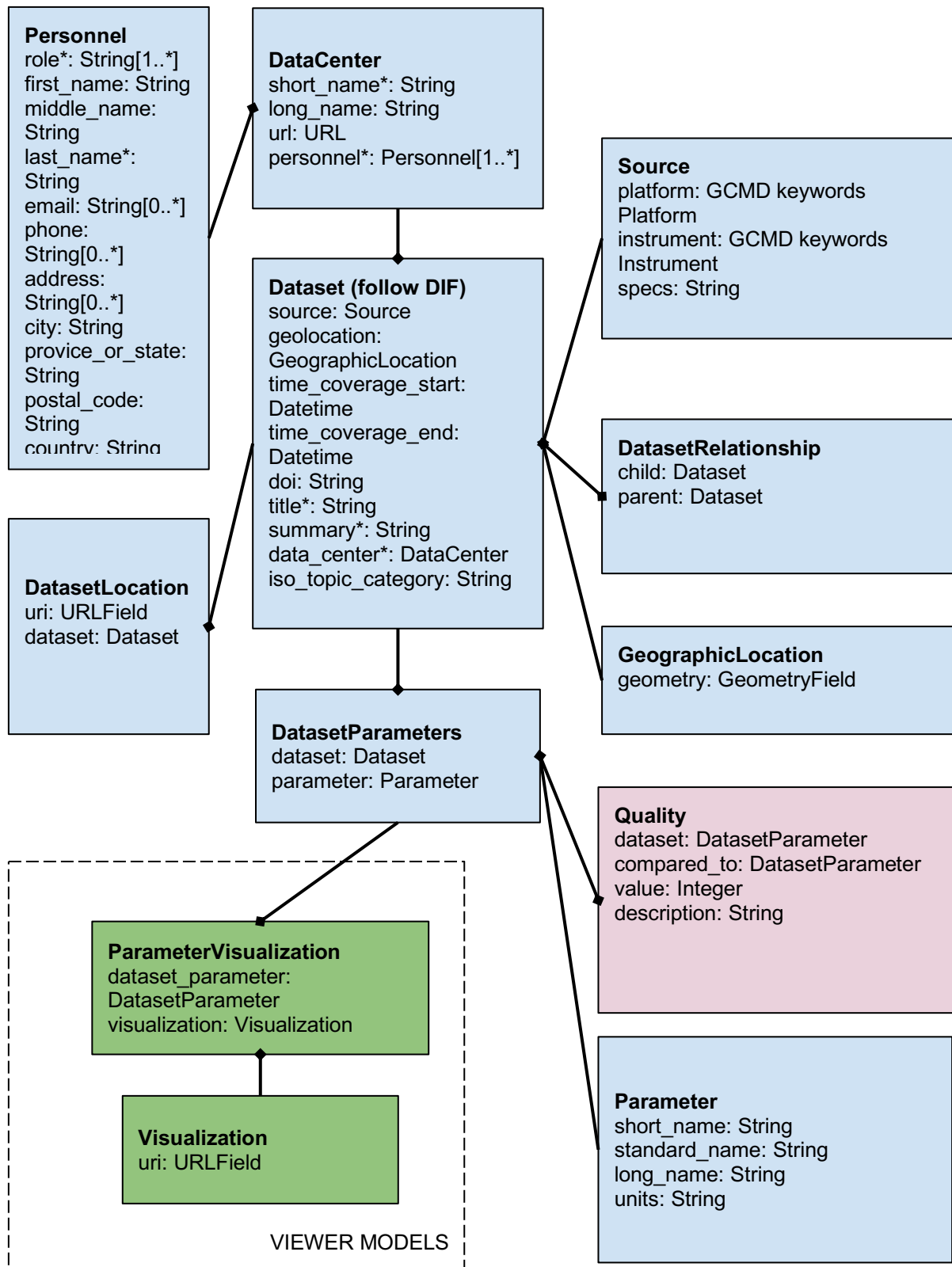


Figure 5. The UML diagram displaying a relational database of the metadata catalogue. The asterisks behind some attribute names indicate a requirement for compliance with CEOS IDN DIF. The brackets indicate multiplicity, i.e., at least 0 or 1 instance. Note that only the most important fields are shown in the database tables.

2.2. Metadata harvesting

2.2.1. Purpose and architecture overview

The central element of the ADAS is its database, which contains basic metadata for all the datasets referenced in the system. These datasets are hosted on a number of local and remote repositories, eliciting the need for a tool capable of automatically going through these repositories, collect the relevant metadata and insert it into the database. This is the job of the metadata harvesting algorithm as shown in Figure 6, containing the following key steps:

- Given a set a data repositories:
 - Crawl through each repository to extract the URLs to relevant datasets;
 - For each of these URLs, fetch the raw metadata of the dataset from the repository;
 - Normalize this metadata, i.e. extract the relevant metadata from the raw metadata in the format expected by GeoSPaaS;
 - Write the normalized metadata to the database.

The harvesting algorithm is realized as a Python application with open source code (<https://github.com/nansencenter/django-geo-spaas-harvesting>) and is composed of the following three main components:

- crawlers: explore data repositories and find URLs of useful datasets;
- ingesters: given the URL of a given dataset, fetches its metadata and writes it to the database;
- harvesters: orchestrates crawlers and harvesters to get metadata from various repositories.

2.2.2. Crawlers

The role of crawlers is to explore a data repository and find the URLs of the relevant datasets. They are iterables which, given a data repository URL, return the URLs found when exploring this repository. The currently available crawlers are:

- OpenDAP (tested on PO.DAAC's OpenDAP repository);
- OData API (tested on Copernicus Sentinel API Hub).

2.2.3. Ingesters

The role of ingesters is to write to the database the relevant metadata about the datasets found by crawlers. Given the URL of a dataset, an ingester fetches the metadata from this URL, normalizes it into the format needed for Geo-SPaaS, and writes it into the database. The tasks of an ingester are primarily I/O bound, so they are multi-threaded. Two thread pools are respectively used:

- to fetch and normalize the metadata;
- to write the metadata to the database.

The threads of the first thread pool put the normalized metadata in a queue. The threads which write to the database read from this queue. To extract the relevant metadata from the raw metadata, most ingesters use the metanorm library. The currently available ingesters are:

- DDX ingester: uses the DDX metadata provided by OpenDAP repositories;
- Copernicus OData API ingester: specific to the OData API from Copernicus API hub;
- Nansat ingester: uses Nansat to open a local or remote file and get its metadata.

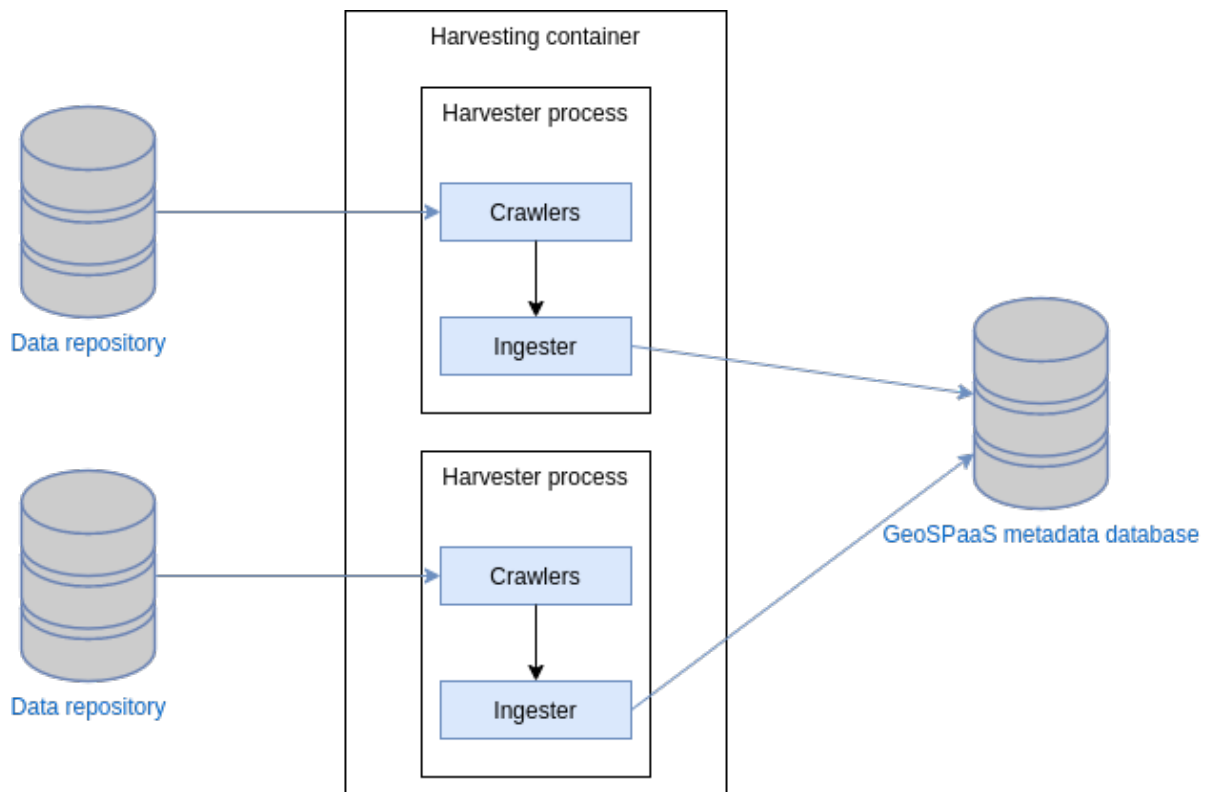


Figure 6: Metadata harvesting mechanism. Gray blocks on the left show remote data repositories. Gray block on the right - discovery metadata database. The central block shows a harvesting Docker container. It launches parallel harvesting processes, which launch crawlers and ingesters. Arrows show direction of metadata flow - from remote data repositories, to crawlers and ingesters, to a local metadata database.

2.2.4. Harvesters

The role of harvesters is to aggregate crawlers and ingesters into an element which can be used to harvest data from a given provider. Each harvester has at least two attributes:

- a list of crawlers;
- an ingester.

The harvester iterates over each of the crawlers and feeds the URLs to the ingester. The currently available harvesters are:

- PO.DAAC harvester: harvests VIIRS and MODIS data from NASA's PO.DAAC repository;

- Copernicus Sentinel harvester: harvests Sentinel 1, 2 and 3 data from the Copernicus API Hub.

The entry point of the metadata harvesting application is the harvest.py script. It takes care of several tasks:

- based on the configuration file, instantiate and run each harvester in a separate process so that each data repository can be harvested in parallel;
- when receiving a SIGTERM or SIGINT signal, shut down the harvesters gracefully and dump their state so that the harvesting can be resumed where it stopped.

3. AVS INTEGRATION

Integration between AVS and ADAS is realized on file level through an automated Downloader (Figure 7). ADAS is responsible for searching relevant datasets and AVS - for visualization. Key components of ADAS in this perspective are Harvesters that collect metadata from remote data repositories and Django-Geo-SPaaS that stores metadata in a database and provides searching interface. Moreover, as shown in Figure 8 the Downloader is launched by an administrator for a given range of dates, regions of interest and dataset sources. It finds the relevant datasets using ADAS, downloads them and provides the files to AVS. The AVS backend automatically scans the temporary directory and generates PNGs files for the web frontend. A user can access the frontend to interactively browse the visualise satellite products.

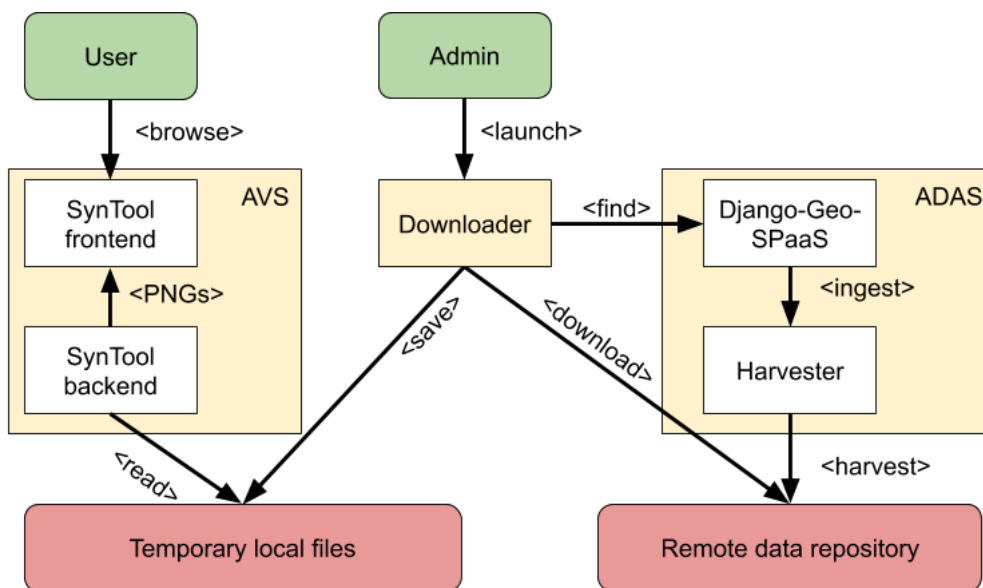


Figure 7. Overview of ADAS and AVS integration. Green blocks show external clients, yellow blocks - ADAS, AVS and Downloader, white blocks - main software components, red blocks - local and remote data repositories. Arrow direction shows invocation sequence - arrows points from an active actor to a resource.

AVS-ADAS Integration Workflow

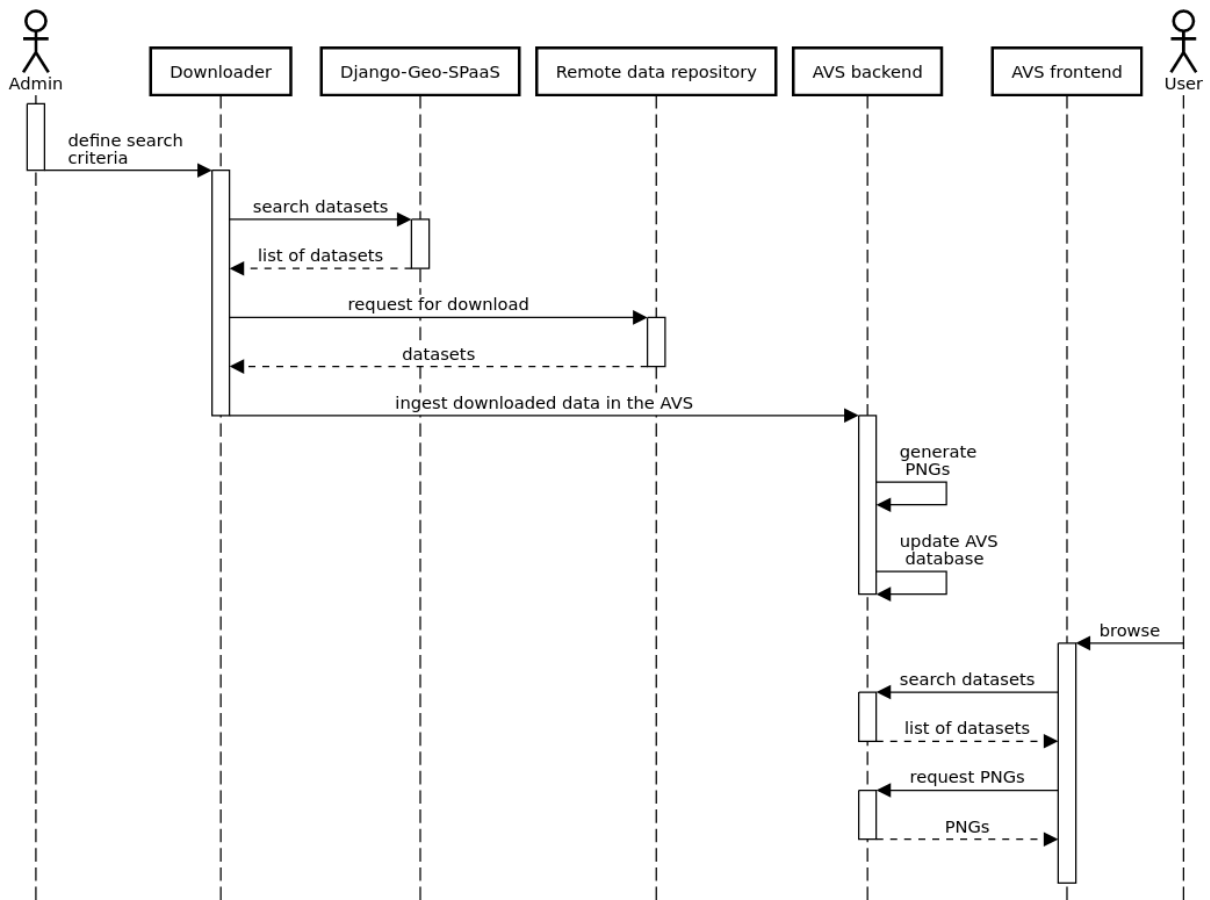
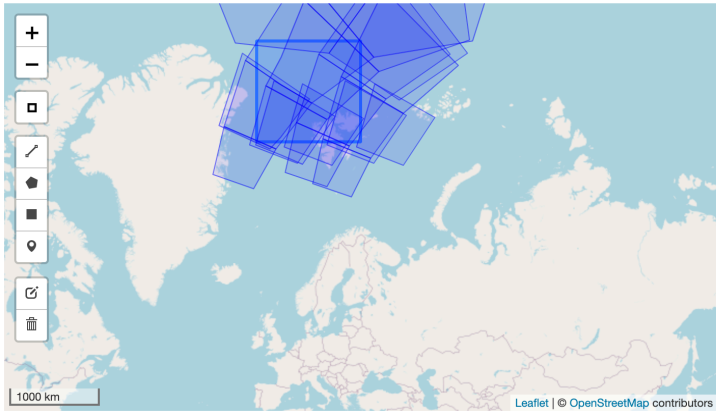


Figure 8. Sequence diagram for the AVS - ADAS integration workflow.

ANNEX 1: WORK IN PROGRESS

2020-07-01: PM-2

- The following software packages were released and published on GitHub:
 - o Pythesint-1.4.10
 - o Nansat-1.2.6
 - o Django-Geo-SPaas-1.0.2
 - o Django-Geo-Spaas-Adas-Viewer-0.1.4
- Software for deploying the ADAS web-service was developed
- A test web-page with ADAS service became available at <https://web.nersc.no/adas>



The screenshot shows a web interface for the ADAS service. On the left, there is a map of the Arctic region with several blue rectangular swaths overlaid, representing satellite coverage. Below the map are search filters: 'Time coverage start' (2000-01-01 00:00:00), 'Time coverage end' (2020-07-01 07:59:06), 'Source' (SENTINEL-1A/SAR, SENTINEL-1B/SAR), and 'name (or part of the name) of parameter'. A 'Search all meta-data' button is also present.

Filename
https://scihub.copernicus.eu/dhus/odata/v1/Products('13a3de17-8db5-4b35-a59f-34e5bca12b7')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('40fd9d20-a668-45e0-abf9-3fd2f6a12336')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('c2aad3d4-3522-4cf6-a83f-98a0b0f3406c')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('13057946-2dac-4da1-93c6-f97f898411c8')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('7057c8ae-93d3-4064-b178-df77531b3a4b')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('f0f6f808-a54c-42ce-8c3c-df12644809eb')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('3ba933dd-05e1-4258-8340-1a96c8d03f96')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('009c1f6b-9305-47cc-8b91-96cbb697594')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('fcb8e54c-9182-4a92-a67c-33c84cb23563')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('9c15b6c2-9cfa-4f29-a291-9d4941450a55')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('757bfa17-e092-4b00-87bc-e9892f1ae5ef')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('dfb1c44a-dd32-4eed-87b3-52f942f2bd55')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('fbe99c40-3936-4e85-99fe-8abf418b046c')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('91f1d110-95a9-40b4-95c9-839eff74ee98')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('212e13f1-7408-4019-af26-26eabef3edc')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('03351cdc-0cad-4131-99a2-cb61d5491479')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('796ca1b8-9773-4183-b936-0317b1d26d70')/\$value
https://scihub.copernicus.eu/dhus/odata/v1/Products('eff189b4-5889-4ee5-97e5-5d70791957fb')/\$value

Figure 9. Preliminary design of the ADAS web-interface