CCI+ Vegetation Parameters

System Specification Document (SSD)

Kris Vanhoof, Else Swinnen

September 2023

Distribution list

Author(s)	:	Kris Vanhoof, Else Swinnen
Reviewer(s)	:	Carolien Toté
Approver(s)	:	Clement Albergel
Issuing authority	:	VITO

Change record

Release	Date	Pages	Description of change	Editor(s)/Reviewer(s)
V1.0	25/09/2023	all	First version	See above
V1.1	17/10/2023	4	Complete acronyms list	Else Swinnen

Table of Contents

LIST OF ACRONYMS

ATBD	Algorithm Theoretical Basis Document
API	Application Programming Interface
C3S	Copernicus Climate Change Service
CCI	Climate Change Initiative
CDO	Climate Data Operators
CEDA	Centre for Environmental DataAnalysis
DIAS	Data and Information Access Services
ESA	European Space Agency
fAPAR	Fraftion of Absorbed Photosynthetically Active Radiation
FTP	File Transfer Protocol
IDE	Integrated Development Environment
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LAI	Leaf Area Index
LAN	Local Area Network
NAS	Network Attached Storage
NetCDF	Network Common Data Form
PUG	Product User Guide
PROBA	PRoject for On-Board Autonomy
R&D	Research and development
ROI	Region Of Interest
SAN	Storage Area Network
SIF	Solar Induced Fluorescence
SMAC	Simplified Method for Atmospheric Correction
SPOT	Système Pour l'Observation de la Terre
SUSE	Software- und SystemEntwicklung
TIP	Two-stream Inversion Package
TOA	Top of Atmosphere
TOC	Top of Canopy
TDS	Test Data Site
UPS	Uninterruptable Power Supply
URD	User Requirements Document
VGT	VEGETATION instrument on-board SPOT
VITO	Vlaamse Instelling voor Technologisch Onderzoek
VM	Virtual Machine
VP	Vegetation Parameters

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Scope of this document

This document describes the system developed and used for the ESA CCI Vegetation Parameters project and its elements used in cycle 1 for the generation of the CRDP-1.
The SSD exists in the context of other VP_cci documents: the User Requirements Document (URD) describes the requirements of the products to be delivered (VP-CCI_D1.1_URD) and the Algorithm Theoretical Basis Document (ATBD) describes the algorithm used for the generation of the products (VP-CCI_D2.1_ATBD).

The SSD describes the following sections:
- System overview (Chapter 2)
- System architecture (Chapter3)
- Workflow description (Chapter 4)

1.2 Related documents

Internal documents

Reference ID	Document
ID1	Climate Change Initiative Extension (CCI+) Phase 2 New ECVs: Vegetation Parameters – EXPRO+ - Statement of Work, prepared by ESA Climate Office, Reference ESA-EOP-SC-CA-2021-7, Issue 1.2, date of issue 26/05/2021
VP-CCI_D1.1_URD_V1.1	User Requirement Document: fAPAR and LAI, ESA CCI+ Vegetation Parameters https://climate.esa.int/media/documents/VP-CCI_D1.1_URD_V1.1.pdf
VP-CCI_D2.1_ATBD_V1.3	Algorithm Theoretical Basis Document: fAPAR and LAI, ESA CCI+ Vegetation Parameters http://climate.esa.int/media/documents/VP-CCI_D2.1_ATBD_V1.3.pdf
VP-CCI_D4.2_PUG_V1.2	Product User Guide: LAI and fAPAR, ESA CCI+ Vegetation Parameters http://climate.esa.int/media/documents/VP-CCI_D4.2_PUG_V1.2.pdf

External documents

Reference ID	Document
CCI Data Standards	ESA Climate Office, CCI Data Standards v2.3 (CCI-PRGM-EOPS-TN-13-0009
C3S_ATBD_SA	C3S ATBD of Surface Albedo, multi-sensor, D1.3.4-v2.0_ATBD_CDR_SA_MULTI_SENSOR_v2.0_PRODUCTS_v1.1

2 System overview

2.1 Main function and Processing chain

The main function of the VP_cci system is the repeated generation of the Vegetation Parameters products with an increasing number of input datasets (multi-sensor and multi-instrument) over the cycles and algorithms which undergo updating.
Overall, there are 4 separate steps in the processing: (1) the acquisition of the input data (level 1), (2) the pre-processing per sensor, (3) the retrieval algorithm OptiSAIL to generate LAI and fAPAR, and (4) the repackaging of the data.

In the first step, all level 1 input satellite imagery is downloaded. The data used is further described in section 4.1. The second step (see also section 4.2.1) includes the projection to a common grid and organization of the input data in 10°x10° tiles. Then these tiles are atmospherically corrected to Top-of-Canopy reflectance data (TOC). The third step starts when all TOC data is available and consists of the retrieval algorithm to calculate LAI and fAPAR (see section 4.2.2). At last, the data are prepared to be uploaded to CEDA (see section 4.3).

In cycle 1, we start directly at step 3 since the input TOC reflectances are already available.

2.2 System requirements

The main requirement is the system is to generate the Vegetation Parameters products according to the algorithm defined in the ATBD [VP-CCI_D2.1_ATBD] and the URD [VP-CCI_D1.1_URD]. These requirements are based on the system requirements formulated in the Statement of Work of the ITT (see Table 1).

Table 1: System requirements from the ITT

ID	Requirement description
R-19	The contractor shall ensure that the system is adequately dimensioned to accommodate the growing volumes of input and output data, and the increasing computational loads needed to process, re-process, quality control, validate, and disseminate multi-decadal, global, ECV data products, of the required climate quality, in a timely manner.
TR-8	The Contractor shall provide the required high performance processing resources necessary and perform the processing.
TR-28	Given the large amounts of data to be processed, the Contractor shall develop an automated high performance processing chain. This processor shall be implemented on a sufficiently powerful (possibly distributed) computing infrastructure that is capable of processing and reprocessing all the required products within the project schedule.

In addition, the system set-up should be able to generate the products in the format as defined in the contract and described in the Product User Guide (PUG) [VP-CCI_D4.2_PUG]. This means that the processing system should allow both tile-based and site-based processing.

3 System architecture

3.1 High-level system decomposition

The central component of the VP_cci system is the *LAI and fAPAR retrieval* process. It relies on efficient selection of input products based on observation time and coverage area. To enable this, a *Data Discovery* process builds a database with a catalogue of all available data for supported sensors, along with descriptions of sensors and product layouts. For some sensors, data may not be available in the processing environment, and in that case, it needs to be downloaded by a *Data Acquisition* process. The LAI and fAPAR retrieval process expects tiled TOC data, with uncertainties and various pixel status masks, so *Preprocessing* to obtain this data may be a prerequisite. Finally, the generated data is prepared by a *Repackaging and Transfer process* for public distribution. An illustration of these components can be found in Figure 1.

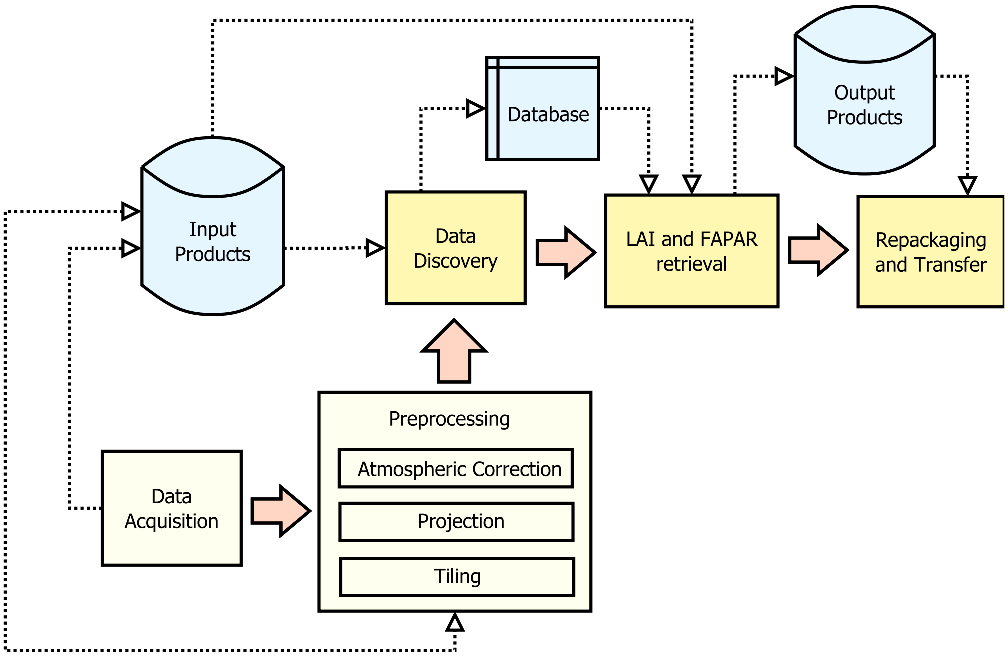


Figure 1: High-level system decomposition

Given the large amount of data that needs to be processed, there is a clear need for a distributed processing system, which is provided by the VITO's Hadoop cluster. Input data products for all sensors used in cycle 1 of this project are already available in the data archives on the NetApp storage system at the VITO processing centre. The Python programming environment is used for the development of all tooling and workflows around the LAI and FAPAR model implementations, with the PySpark library functioning as the interface to the Hadoop cluster through the Spark distributed processing engine.

3.2 Processing Environment

VITO has an energy efficient Tier 2 data centre. The computer rooms have typically an area of 8 x 8m and are filled with racks, setup in a cold aisle containment. All computer rooms have power supply redundancy using a UPS system.

3.2.1 Storage infrastructure

VITO has set-up a hybrid scale-out Storage System that unifies its SAN and NAS infrastructure, for the flexible storing and presenting of data towards the different processing chains and delivery services. The unique NetApp Storage Virtual Network technology guarantees a safe and performant presentation of the data towards all different network segments.

3.2.2 Network infrastructure

VITO has separated network segments for development, test/validation and production. All production segments are separated from each other as well. The VITO internet connectivity has an upload/download capacity of **10Gbit/sec**. VITO is connected via Belnet to GÉANT/Internet2. To serve the end user community that is not connected to the GÉANT network, VITO has an internet link to the scientific and the commercial Internet of 10 Gbit/s, both being provided by the ISP Belnet. Recently the entire internal core network switching has been upgraded to 40Gbit/sec towards the central redundant firewall and to 10Gbit/sec towards all crucial Top-Of-Rack switches in the datacentre upload/download capacity of 10Gbit/sec.

3.2.3 Hadoop system cluster for scalable processing and data analytics

Hadoop as a software framework for data-intensive distributed applications, is designed to process large amounts of data by separating the data into smaller chunks and performing large numbers of small parallel operations on the data. It is applied often for processing big data and performing big data analytics. The Hadoop cluster is based on the PROBA-V Mission Exploitation Platform (MEP), a private cloud environment, with a priority scheme being implemented to avoid delays in the availability of hardware resources for operational services. The platform is based on the Hortonworks distribution and Spark[1] which is used intensively to perform large parallel processing and to allow analytics on large time series of data. The operational Hadoop environment at VITO is provided with monitoring and maintenance tools. The cluster ensures high availability through an error rollover system and provides a very scalable and powerful cluster with direct access to the data archives.

Figure 2: Spark monitoring tool

The VITO Hadoop cluster currently consists of 7000+ cores and 25+ TB of memory , and still expanding based on project demand. The cluster is located in a private cloud and hence shared with several other tasks embedded in the Terrascope platform[2]. Every job submitted to this cluster is linked to a queue, and each queue has a priority level depending of the urgency of the processing. This can be configured based on urgency of (re-)processing actions. Currently the ESA CCI Vegetation Parameters project has a dedicated queue that is guaranteed to get a minimum of 5% of the total available cluster cores and memory, with a maximum of 100% if more resources are available during processing.

[1] https://spark.apache.org/
[2] https://terrascope.be/en/services

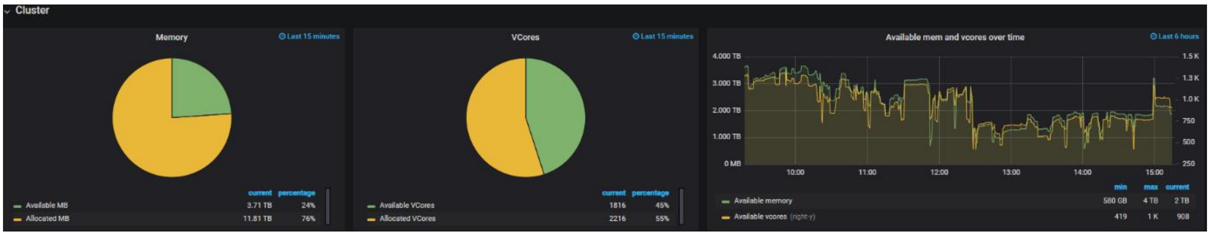


Figure 3: Cluster resource monitoring

3.2.4 Algorithm development

The VITO cloud system enables dynamic resource provisioning, and it is therefore providing a performing and scalable solution. OpenStack is used as cloud middleware for a private cloud solution at VITO. Pre-configured Virtual Machines (VM) are offered to the algorithm developers and can run on the OpenStack cluster at VITO, providing the environment needed for them to work with the data and develop/deploy applications on the platform, i.e. containing Integrated Development Environments (IDE), a rich set of tools and access to the complete data archive. Furthermore, users can customise this environment by downloading more data and tools. As such the algorithm developers have access to all input and intermediate datasets and can work in close cooperation with the workflow developers and hence enable a smooth transition between development and operations. These VM's enable the workflow developers to develop and test their new or revised workflows in a local Hadoop Spark environment with limited resources, before performing an acceptance test on the cluster.

Table 2 gives a general overview of all IT infrastructure that is in place to support the project. As more projects make use of the infrastructure, the amount of resources will increase over time.

Table 2: IT infrastructure overview VITO Data center

COMPONENT	FUNCTIONALITY WITHIN PROJECT	TECHNICAL SPECIFICATIONS
Network Infrastructure		
LAN/SAN	Data Sharing, Data Exchange	LAN with 1/10 GBit/s (Gigabit Ethernet) based on TCP/IP with structured cabling infrastructure SAN FibreChannel 16/32 GBit/s
Internet/Firewall	Data Sharing, Data Exchange	VITO head office internet connection: 10 GBit/s with a 1 GBit/s redundant line 10Gbit/s redundant Next Generation Firewall with identity awareness, anti-Bot&Anti-Virus, IPS, Anti-spam, application control and URL filtering
Server Infrastructure		
Hadoop processing cluster	Large scale parallel satellite data processing	Cluster with capacity of 7000+ cores and 25+ TB of memory, which can be dynamically allocated to project needs
SUSE OpenStack Private Cloud	Service hosting, R&D, data exchange, satellite data processing	18 compute nodes, hosting 300+ internal and external VMs on CentOS image
VMWare cluster	Service hosting, R&D, data exchange	6 hypervisor servers hosting 200+ virtual instances
Storage Infrastructure		
Netapp storage environment	Online storage of datasets	Capacity is 9.0 PB of mostly Near-Line-SAS disks

Tape storage	Archive storage of datasets	Capability of handling LTO 4 to LTO8 tape technology. Current used capacity is 7.0 PB
Cloud Computing		
CreoDIAS – WeKEO	Data processing and exchange	Multiple tenants setup and direct full VPN setup to the CreoDIAS backend
AWS	Data processing and exchange	Multiple tenants setup and project based VPN tunnels
Remote Conferencing		
Teams & Zoom setups	Virtual and hybrid meetings	All of the VITO laptops and workstations have a MS Teams and Zoom environment setup for optimal virtual or hybrid meetings
Conference rooms with hybrid Polycom setup	Virtual and hybrid meetings	Most of the VITO conference rooms are equiped with a Polycom Trio conferencephone, Clickshare technology and an optional camera for full virtual or hybrid meetings

3.3 Database

3.3.1 Overview

The VP_cci processing system's LAI and fAPAR retrieval workflow relies heavily on selecting input products based on ROI and time range. This requires us to scan several large filesystem trees to discover products available at the processing centre and inspect their metadata (most notably start and end time). To avoid having to do this for every individual output product that has to be generated, we collect this data once before processing and store the results for later use.

In addition, the system needs to be extensible and flexible enough to accommodate multiple sensors, with new sensors added in future project cycles. To handle this, we keep specific sensor configurations and product layout descriptions independent of the processing workflow implementation.

Finally, we want to keep track of the processing status, and have the possibility to restart processing for selected products. Also, it should be straightforward to adjust processing parameters for selected products to test different configuration strategies.

To store all this information the VP_cci processing system uses a centralized database that contains:

- Processing configuration parameters per supported sensor
- Tile definitions per supported sensor
- Available input products per supported sensor
- Test data site definitions with selected years of interest
- Tasks for initiating processing and producing output products

To initialize the database, first a script `generate_database.py` registers sensor configurations, tiles and test data sites. Then a script `pl_cci_collect_products.py` traverses the selected input data directories, reads product metadata and registers available input products per sensor. This is implemented as a distributed Hadoop/Spark to reduce running time. Once these two scripts are finished, this database content is considered to be read-only, and is never modified during processing.

Two additional scripts `generate_tds_tasks.py` and `generate_transect_tasks.py` are used to start the actual processing. These scripts insert new processing request tasks that will be picked up by the processing workflow. Tile-based and site-based processing is handled by the same workflow, but with a slightly different workflow path optimized for different use cases.

The database structure diagram is shown in Figure 4. More information on the various tables can be found in sections 3.3.2 to 3.3.6.

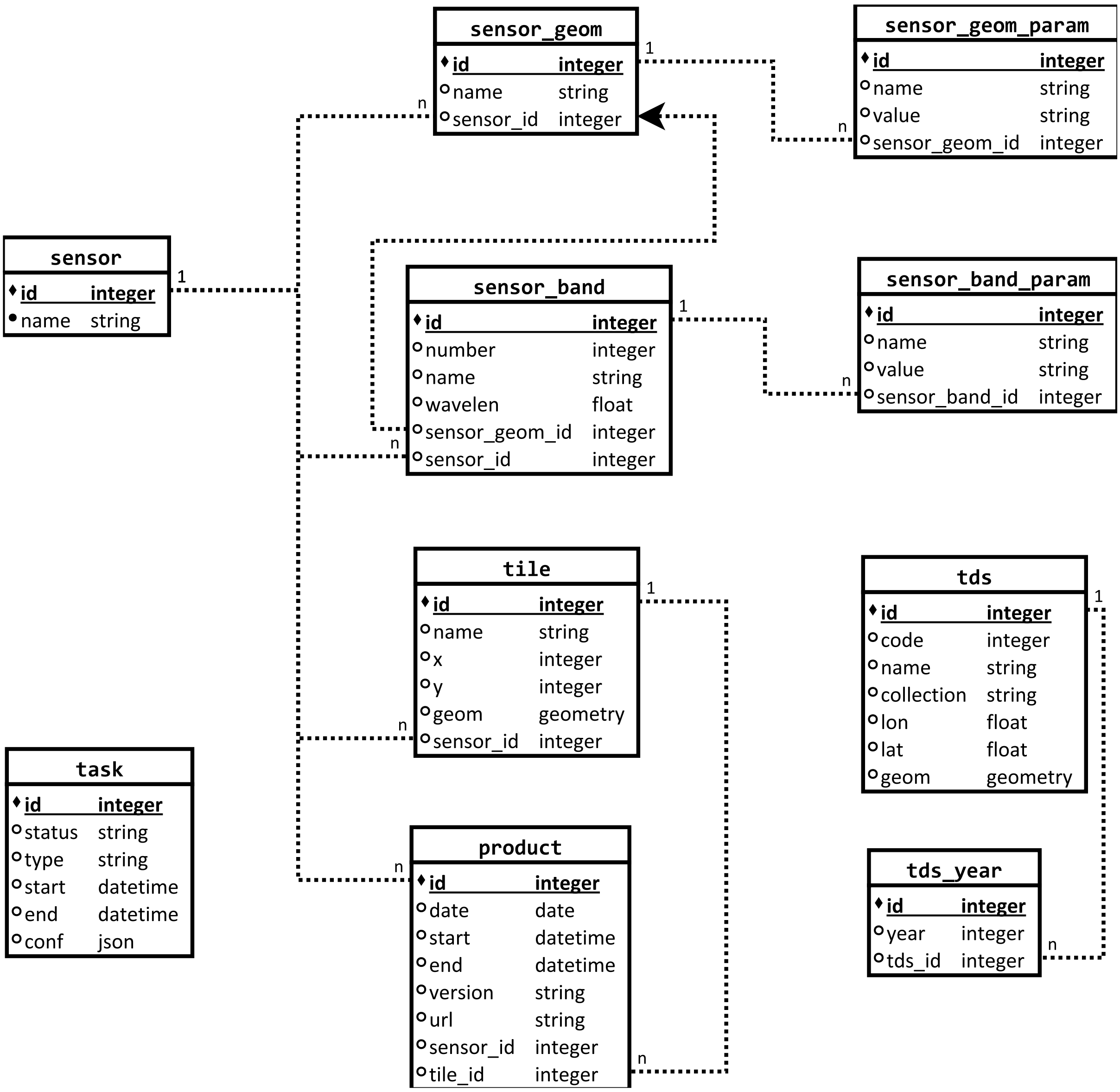


Figure 4: Database Structure

3.3.2 Sensor processing configurations

A list of supported sensors is maintained in the **sensor** table. An example of the data in this table is shown in Table 3.

Table 3: Example sensor table contents

id	name
1	probav
2	vgt1
3	vgt2
...	

Per sensor, the **sensor_band** table stores a list of radiometric bands, each with an associated band-geometry, as illustrated in the example in Table 4.

Table 4: Example sensor_band table contents

id	number	name	wavelen	sensor_geom_id	sensor_id
1	1	band1	463.5	1	1
2	2	band2	655.0	1	1
3	3	band3	839.0	1	1
4	4	band4	1602.5	2	1
...					

Per sensor band, the **sensor_band_param** table stores the product specific data layers to use for radiometric band, uncertainty and quality data, in addition to various related processing parameters. Table 5 shows an example of the data that is stored.

Table 5: Example sensor_band_param table contents

id	name	value	sensor_band_id
1	SDR	/LEVEL2B/RADIOMETRY/band1/TOC	1
2	SDR_uncertainty	/LEVEL2B/RADIOMETRY/band1/TOC_ERR	1
3	uncertainty_factor	= 2.236068	1
4	min_rel_uncertainty	= 0.06	1
5	quality	/LEVEL2B/QUALITY/SM_probav_v2	1
6	quality_bitmask	0x80	1
7	quality_bitvalue	0x80	1
...			

Per sensor, there can be one or more band-geometries, depending on whether the radiometric bands are coregistered. These can be found in the **sensor_geom** table. For example, for Proba-V there are different band geometries for VNIR and SWIR bands, as shown in Table 6.

Table 6: Example sensor_geom table contents

id	name	sensor_id
1	vnir	1
2	swir	1
...		

Per sensor band-geometry, the **sensor_geom_param** table stores the product specific data layers to use for solar and viewing angles. Snow- and cloud-mask information is also stored here as these are shared by all bands. See Table 7 for an example of the data in the database.

Table 7: Example sensor_geom_param table contents

id	name	value	sensor_geom_id
1	SZA	/LEVEL2B/GEOMETRY/SZA	1
2	SAA	/LEVEL2B/GEOMETRY/SAA	1
3	VZA	/LEVEL2B/GEOMETRY/VNIR/VZA	1
4	VAA	/LEVEL2B/GEOMETRY/VNIR/VAA	1
5	snow	/LEVEL2B/QUALITY/SM_probav_v2	1
6	snow_bitmask	0x4	1
7	snow_bitvalue	0x4	1
8	cloud	/LEVEL2B/QUALITY/SM_probav_v2	1
9	cloud_bitmask	0x3	1
10	cloud_bitvalue	0x0	1
...			

3.3.3 Sensor tile definitions

Per sensor, the **tile** table contains a list of tiles, depending on the tiling grid used for the sensor's products. Note that in the processing workflow, we look for products based on their geometry, so inconsistencies between the tile numbering of different sensors are not an issue. See Table 8 for some sample database tile records.

Table 8: Example tile table contents

id	name	x	y	geom	sensor_id
1	X00Y00	0	0	POLYGON((-180 75, -180...))	1
2	X00Y01	0	1	POLYGON((-180 65, -180...))	1
3	X00Y02	0	2	POLYGON((-180 55, -180...))	1
4	X00Y03	0	3	POLYGON((-180 45, -180...))	1
...					

3.3.4 Sensor input products

For each available input product, its location, start time, end time, and matching tile is stored in the **products** table, as illustrated in Table 9.

Table 9: Example product table contents

id	date	start	end	version	url	sensor_id	tile_id
1	2000-01-13	2000-01-13 09:14:04	2000-01-13 09:19:53	1.0.1	file://...	2	728
2	2000-01-13	2000-01-13 22:30:37	2000-01-13 22:31:16	1.0.1	file://...	2	990
...							

3.3.5 Test data site definitions

A list of test data sites is stored in the database in the **tds** table. A test data site has a small geometry associated with it, typically covering an area of around 3 by 3 km around the test data site coordinate. An example of the records in this database table is shown in Table 10.

Table 10: Example tds table contents

id	code	Name	collection	lon	lat	geom
1	0	ABRACOS_HILL	LANDVAL V1.1	-62.3583	-10.76	POLYGON((...))
2	1	ADAMOWKA	LANDVAL V1.1	59.75	51.75	POLYGON((...))
3	2	AGUASCALIENTES	LANDVAL V1.1	-102.32	21.7	POLYGON((...))
...						

For each test data site there is a list of years that is of special interest, and that is stored in the **tds_year** table, as can be found in the sample table contents in Table 11.

Table 11: Example tds_year table contents

id	year	tds_id
1183	2006	932
1184	2007	932
1185	2008	932
...		

3.3.6 Processing tasks

Tasks are stored in the **task** table and are the main mechanism for driving the processing workflow. They have a status that can be used to track processing progress and for reporting problems. The valid status values for a task are shown in Table 12.

Table 12: Task status values

Status	Meaning
`PENDING`	Task is waiting to be processed
`RUNNING`	Task is currently being processed
`SUCCESS`	Task finished successfully, with all timeseries dates completed
`FAILURE`	Task finished with an error, no timeseries dates generated
`INCOMPLETE`	Task finished with an error, some timeseries dates not completed
`CANCELLED`	Tasks is cancelled and no longer needs to be processed

Some example task database records can be found in Table 13.

Table 13: Example task table contents

id	Status	type	start	End	conf
1	SUCCESS	optisail	2023-09-06 10:39:32	2023-09-06 10:59:23	{…}
2	SUCCESS	optisail	2023-09-06 10:39:36	2023-09-06 11:45:17	{…}
...					

Tasks have a **`type`** parameter that is used in the processing workflow to select a sub-workflow, depending on the LAI and FAPAR retrieval algorithm to be used. In addition, each task has a set of configuration parameters stored in JSON format, dependent on the task type. Typically the **`geom`**, **`year`**, **`interval`** and **`window`** parameters are provided to specify the ROI and timeseries dates to be generated. Optionally a specific list of dates can be requested. For an example configuration see Figure 5.

```
conf = {
  "name": "X19Y00",
  "geom": "POLYGON((10 75, 10 65.008929, ...))",
  "crop": false,
  "year": 2019,
  "interval": "5D",
  "window": "10D",
  "outdir": "optisail/2019/X19Y00"
}
```

Figure 5: Task configuration in JSON format

4 Workflow

The LAI and FAPAR retrieval workflow is the main component of the VP_cci processing system. It generates an LAI and FAPAR image timeseries for a specified geographical area and date range.

The timeseries date range is sampled at a specified interval, every 5 days by default. At each sample date in the timeseries, a selection is made of the available input products that overlap the area of interest and a configurable observation window around the sample date (10 days by default). The selected input data products, that may originate from a mix of different sensors, are then combined by either the *OptiAlbedo+TIP* or the *OptiSAIL* model to generate an LAI and FAPAR output product, with a number of additional output layers in the case of *OptiSAIL*.

The LAI and FAPAR retrieval workflow expects input products with cloud, snow and quality masks, in addition to TOC reflectance and uncertainty layers. If the required product layers are not available at the processing centre, data may need to be downloaded and/or preprocessed by applying atmospheric correction, reprojection and tiling.

A high-level view of the dataflow through the VP_cci processing system is shown in Figure 6.

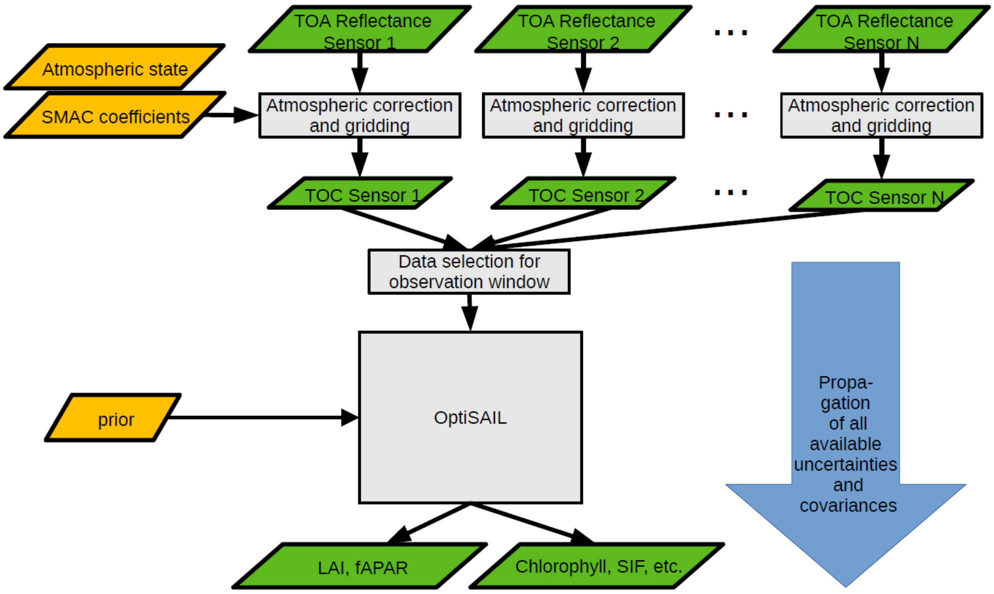


Figure 6: High-level view of dataflow

The implementation of the different workflow components is done in the Python programming language, using the PySpark API for distributed processing on VITO's Spark/Hadoop cluster environment. The LAI and FAPAR retrieval models are implemented as independent command line programs and are invoked from the workflow code through a Python wrapper layer.

4.1 Input data acquisition

The input data used in cycle 1 are intermediate data from the C3S_312b_Lot5 contract [C3S_ATBD_SA]. This dataset consists of surface reflectance data from SPOT4/5-VGT1/2 and Proba-V at 1 km spatial resolution. This dataset was already available at the processing centre. For more details see the ATBD [VP-CCI_D2.1_ATBD]. No additional data was acquired for the processing.

4.2 Data processing

4.2.1 Pre-processing per sensor

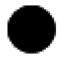
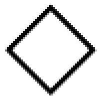
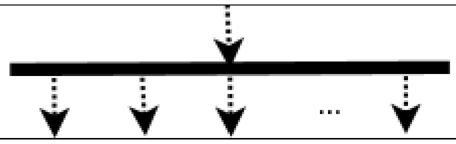
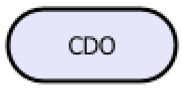
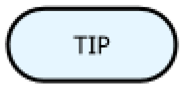
As mentioned in section 4.1, the data used in cycle 1 were already pre-processed and available for further processing.

4.2.2 LAI and FAPAR retrieval workflow

4.2.2.1 Diagram Symbols

A list of symbols used in the workflows diagrams of the next sections can be found in Table 14.

Table 14: Workflow diagram symbols

	Workflow start
	Workflow end
	Summing junction
	Condition
	Operation
	Parallel execution
	Database operation
	OptiAlbedo operation
	CDO operation
	TIP operation
	OptiSAIL operation

4.2.2.2 Workflow driver

The workflow driver is responsible for querying the database for tasks to be processed. Each task has a set of configuration parameters, notably:

- The type of LAI and FAPAR retrieval algorithm to be used
- The region of interest, this can be a larger tile or a smaller test data site geometry
- The timeseries dates to be generated

Depending on the algorithm requested, the workflow driver then delegates the actual processing to a sub-workflow. The workflow driver actions and decision tree are illustrated in Figure 7.

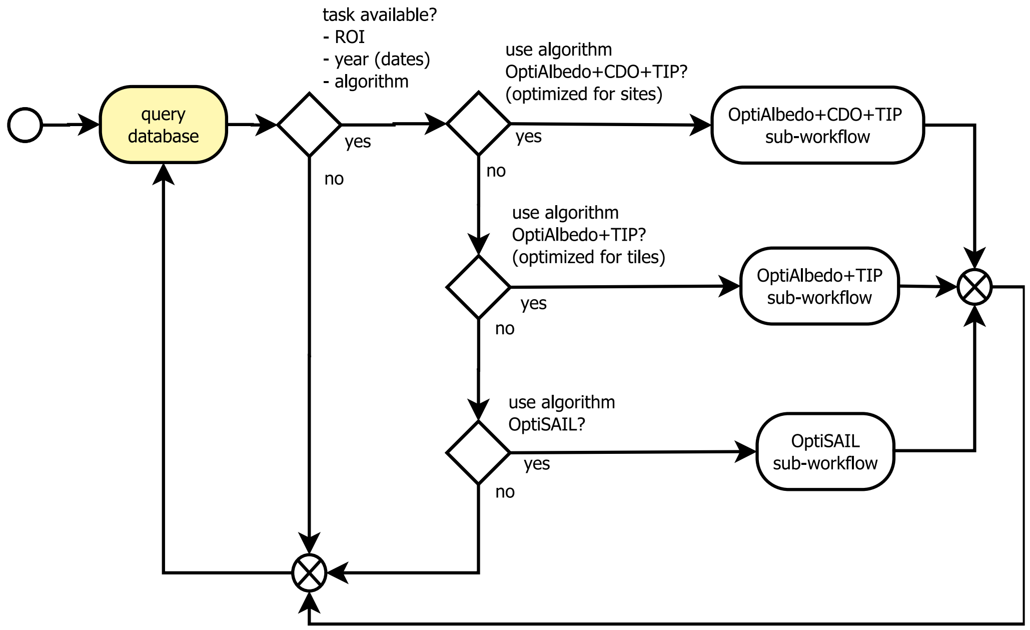


Figure 7: Workflow driver

4.2.2.3 OptiAlbedo+CDO+TIP Sub-Workflow

This sub-workflow uses *OptiAlbedo* together with *TIP* for LAI and FAPAR retrieval but is specifically intended to handle the small ROIs associated with the test data sites. *TIP* is optimized to work with larger blocks of data, so to accommodate this the per-date *OptiAlbedo* outputs are first combined into one complete timeseries file by using the *CDO*[3] (Climate Data Operators).

The input for this sub-workflow is a database task table entry, which specifies the region of interest and the timeseries dates to be generated, amongst other configuration parameters. For a typical task, a series of dates is generated for one year, with a 5-day interval.

Next, for each timeseries date, the database is queried for a list of input data products that have an observation time within a window around that date (normally 10 days) and that overlap the area of interest. For each type of input product, the sensor configuration parameters are fetched from the database. This information is used to generate the configuration files for *OptiAlbedo* (see section 4.2.3.6).

Then, as illustrated in Figure 8, the *OptiAlbedo* program is run in parallel for all timeseries dates. All separate output files are then combined into a single file using the *CDO mergetime* operator, which is then passed to *TIP* to generate the final LAI and FAPAR timeseries output.

Finally, the task status is updated in the database. If a complete timeseries could be generated without errors, the task is considered to have completed successfully. If the workflow was unable to generate a timeseries the task is considered to have failed. However, if a timeseries could be generated, despite some dates failing, the product is marked as incomplete.

---

[3] https://code.mpimet.mpg.de

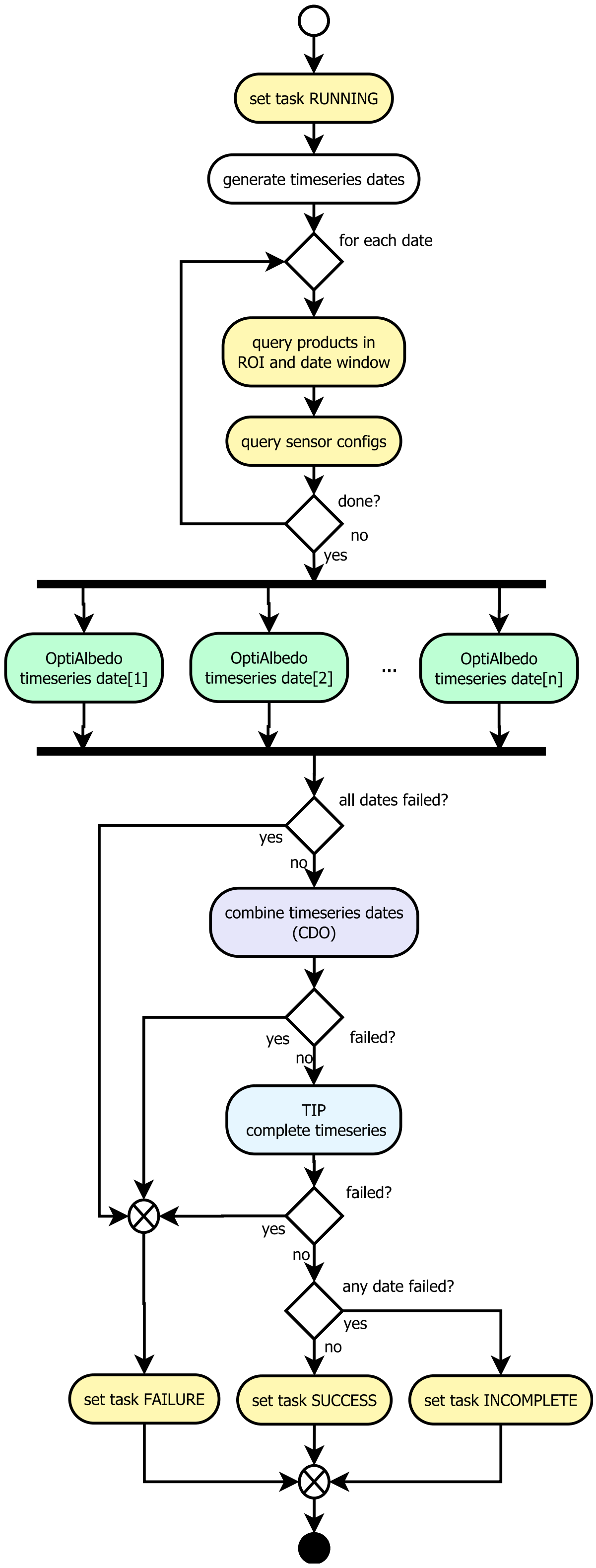


Figure 8: OptiAlbedo+CDO+TIP Sub-Workflow

4.2.2.4 OptiAlbedo+TIP Sub-Workflow

This sub-workflow uses *OptiAlbedo* together with *TIP* for LAI and FAPAR retrieval and is specifically intended to handle larger ROIs like those for complete tiles. TIP is optimized to work with larger blocks of data, which suits tile processing.

As with the other sub-workflows, the input here is a database task table entry, which specifies the region of interest and the timeseries dates to be generated, amongst other configuration parameters. For a typical task, a series of dates is generated for one year, with a 5-day interval.

Next, for each timeseries date, the database is queried for a list of input data products that have an observation time within a window around that date (normally 10 days) and that overlap the area of interest. For each type of input product, the sensor configuration parameters are fetched from the database. This information is used to generate the configuration files for *OptiAlbedo* (see section 4.2.3.6).

Then, as illustrated in Figure 9, the *OptiAlbedo* program, followed by the *TIP* program, is run in parallel for all timeseries dates. This then immediately produces the output files for all dates in the final LAI and FAPAR timeseries.

Finally, as with the other sub workflows, the task status is updated in the database. If a complete timeseries could be generated without errors, the task is considered to have completed successfully. If the workflow was unable to generate a timeseries the task is considered to have failed. However, if a timeseries could be generated, despite some dates failing, the product is marked as incomplete.

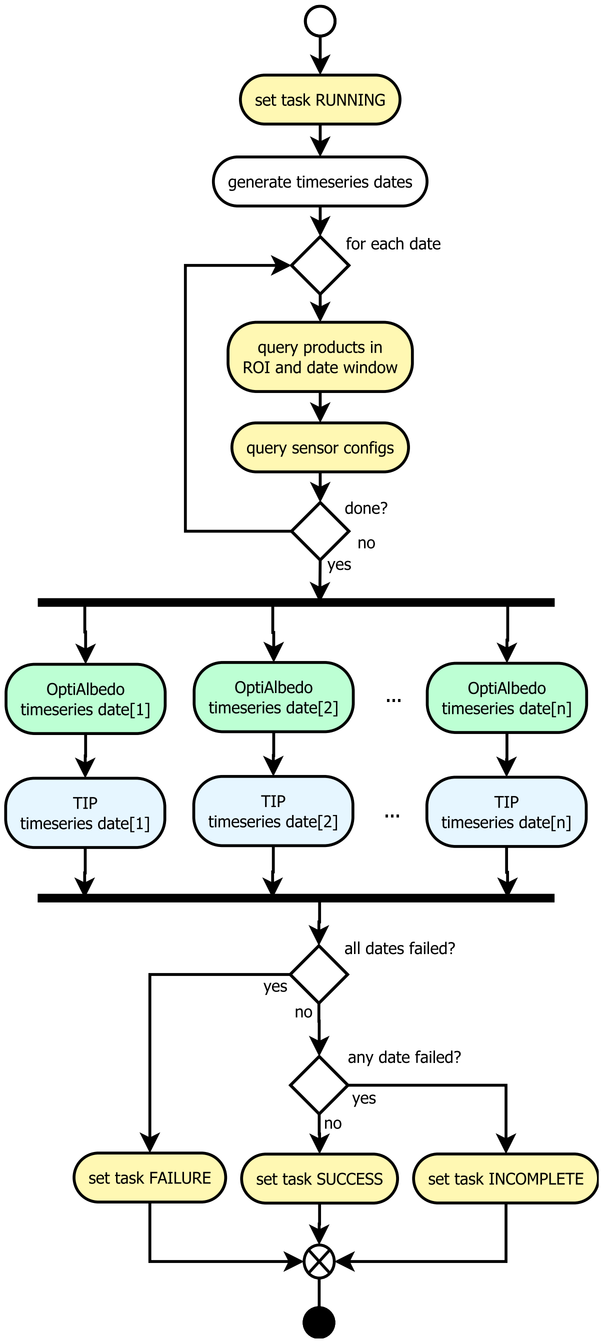


Figure 9: OptiAlbedo+TIP Sub-Workflow

4.2.2.5 OptiSAIL Sub-Workflow

This sub-workflow uses *OptiSAIL* for LAI and FAPAR retrieval. No special cases are needed for test data sites or tiles, as this sub-workflow handles smaller and larger ROI's equally well.

As with the other sub-workflows, the input here is a database task table entry, which specifies the region of interest and the timeseries dates to be generated, amongst other configuration parameters. For a typical task, a series of dates is generated for one year, with a 5-day interval.

Next, for each timeseries date, the database is queried for a list of input data products that have an observation time within a window around that date (normally 10 days) and that overlap the area of interest. For each type of input product, the sensor configuration parameters are fetched from the database. This information is used to generate the configuration files for *OptiSAIL* (see section 4.2.3.6).

Then, as illustrated in Figure 10, the *OptiSAIL* program is run in parallel for all timeseries dates. This then immediately produces the output files for all dates in the final LAI and FAPAR timeseries.

Finally, as with the other sub workflows, the task status is updated in the database. If a complete timeseries could be generated without errors, the task is considered to have completed successfully. If the workflow was unable to generate a timeseries the task is considered to have failed. However, if a timeseries could be generated, despite some dates failing, the product is marked as incomplete.

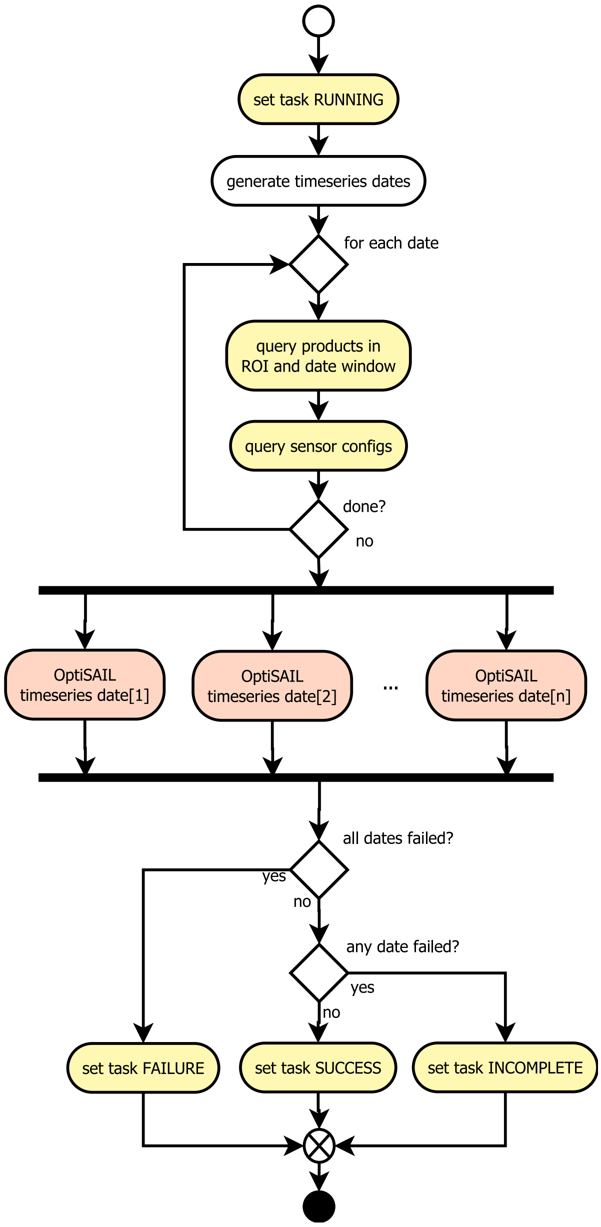


Figure 10: OptiSAIL Sub-Workflow

4.2.3 Executables

4.2.3.1 Wrapper Interfaces

The *OptiAlbedo*, *TIP*, *OptiSAIL* and *CDO* workflow operations are implemented as external command line executables. To make them callable from the workflow's Python code they are wrapped as importable Python modules. In addition, each wrapper module is implemented in such a way that it can be executed directly as a command line program, for testing purposes.

A wrapper module prepares the executable environment by configuring the dynamically linked library path, setting environment variables that affect executable initialization and optionally generating configuration files. It then runs the executable as a subprocess, while capturing output messages and runtime metrics and redirecting these to a log file.

4.2.3.2 OptiSAIL

This program implements one of the supported LAI and fAPAR retrieval models.

Input:

- ROI
- Date and observation window
- Input products for one or more sensors, as NetCDF files
- Input product layout specifications, including
  - Solar and viewing angles
  - TOC reflectance bands
  - Uncertainty bands
  - Cloud, snow and quality masks
- Land/Sea mask

Output:

- NetCDF file

The inputs are provided through a configuration file, see section 4.2.3.6 for more information.

4.2.3.3 OptiAlbedo

This program implements the surface albedo estimation model, which is used as an input for the TIP model.

Input:

- ROI
- Date and observation window
- Input products for one or more sensors, as NetCDF files
- Input product layout specifications, including
  - Solar and viewing angles
  - TOC reflectance bands
  - Uncertainty bands
  - Cloud, snow and quality masks
- Land/Sea mask

Output:

- A NetCDF file

The inputs are provided through a configuration file, see section 4.2.3.6 for more information.

4.2.3.4 TIP

This program implements one of the supported LAI and FAPAR retrieval models.

Input:
- Output of OptiAlbedo, as a NetCDF file

Output:
- A NetCDF file

4.2.3.5 CDO

CDO[4] (Climate Data Operators) is a collection of command line tools to manipulate and analyse Climate and NWP model Data. The LAI and fAPAR retrieval workflow uses the *mergetime* operator to combine multiple OptiAlbedo NetCDF output files for separate dates into one NetCDF file containing the complete timeseries.

Input:
- A list of NetCDF files

Output:
- A NetCDF file

4.2.3.6 Example Configuration File

Figure 11 shows an example configuration file used by OptiSAIL and OptiAlbedo, with some repetitive sections truncated for brevity.

```
processing_mode = tile
obs_interval_h = 240
valid_date = 20140426T12:00
n_geometries = 106
n_geometry_settings = 4
n_bands = 8
n_band_settings = 8
n_obs = 212
n_sensors = 2
n_files = 53
roi_latsouth = 45.008929
roi_latnorth = 55
roi_lonwest = 0
roi_loneast = 9.991071
slab_size = 224
land_mask_file_id = 1
land_mask = /LEVEL2B/QUALITY/SM
land_bitmask = 0x8
land_bitvalue = 0x8
output_retrieval = .../X18Y02_optisail_2014-04-26.nc
output_covariance = .../X18Y02_optisail_2014-04-26.nc
```

[4] https://code.mpimet.mpg.de

```
<centre_wavelengths>
1 463.5
...
8 1635.0
</centre_wavelengths>
<sensor_names>
1 Proba-V_CENTER
2 VGT2
</sensor_names>
<files>
1 fqfilename .../c3s_L2B_20140421_X18Y02_114528_1KM_2_PROBAV_SM_V1.0.1.nc
...
53 fqfilename .../c3s_L2B_20140501_X18Y02_094348_1KM_3_PROBAV_SM_V1.0.1.nc
</files>
<band_settings>
1 sensor_id 1
1 wavelength_ix 1
1 SDR /LEVEL2B/RADIOMETRY/band1/TOC
1 SDR_uncertainty /LEVEL2B/RADIOMETRY/band1/TOC_ERR
1 uncertainty_factor = 2.236068
1 min_rel_uncertainty = 0.06
1 quality /LEVEL2B/QUALITY/SM_probav_v2
1 quality_bitmask 0x80
1 quality_bitvalue 0x80
...
8 sensor_id 2
8 wavelength_ix 8
8 SDR /LEVEL2B/RADIOMETRY/band4/TOC
8 SDR_uncertainty /LEVEL2B/RADIOMETRY/band4/TOC_ERR
8 uncertainty_factor = 2.236068
8 min_rel_uncertainty = 0.06
8 quality /LEVEL2B/QUALITY/SM
8 quality_bitmask 0x00
8 quality_bitvalue 0x00
</band_settings>
<geometry_settings>
1 SZA /LEVEL2B/GEOMETRY/SZA
1 SAA /LEVEL2B/GEOMETRY/SAA
1 VZA /LEVEL2B/GEOMETRY/VNIR/VZA
1 VAA /LEVEL2B/GEOMETRY/VNIR/VAA
1 snow /LEVEL2B/QUALITY/SM_probav_v2
1 snow_bitmask 0x4
1 snow_bitvalue 0x4
1 cloud /LEVEL2B/QUALITY/SM_probav_v2
1 cloud_bitmask 0x3
1 cloud_bitvalue 0x0
...
</geometry_settings>
```

```
<geometries>
1 file_id 1
1 status_file_id 1
1 geometry_settings_id 1
...
106 status_file_id 53
106 geometry_settings_id 2
</geometries>
<observations>
1 file_id 1
1 geometry_id 1
1 band_settings_id 1
...
212 file_id 53
212 geometry_id 106
212 band_settings_id 4
</observations>
```

Figure 11: OptiAlbedo/OptiSAIL configuration file

4.3 Data repackaging and transfer

Before public distribution we repackage the NetCDF output products generated by the LAI and FAPAR retrieval workflow. We implemented this in Python as a Spark/Hadoop process to reduce running time.

The repackaging process ensures that our products are compliant with *ESA/CCI Data Standards*[5], which implies compliance to *CF-1.8 conventions*[6]. At this point we also include the Digital Object Identifier (DOI) assigned to us by CEDA as a NetCDF attribute.

For sites the final filename format is:

```
ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-site_<site-id>_<site-name>-<YYYY>-fv1.0.nc
```

And for tiles the final filename format is:

```
ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-tile_<tile>-<YYYYMMDD>-fv1.0.nc
```

Repackaging also reduces the total data volume, because we remove project-internal data layers and files. We additionally merge the tiny per-date NetCDF files for test data sites into one-year timeseries NetCDF files, which greatly reduces storage overhead and number of files.

After repackaging, we obtain a filesystem hierarchy as shown in Figure 12.

```
|- sites/
|   |- 2000/
|   |- 2001/
|   |- ...
|   `- 2020/
|      |- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-site_00001_ABRACOS_HILL-2020-fv1.0.nc
|      |- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-site_00002_ADAMOWKA-2020-fv1.0.nc
```

[5] https://climate.esa.int/en/explore/esa-cci-data-standards
[6] http://cfconventions.org/

```
|      |- ...
|      `- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-site_00932_BILANTAO_NE_SW-2020-fv1.0.nc
|
|- transect/
|   |- 2000/
|   |- 2001/
|   |- ...
|   `- 2020/
|      |- X19Y00
|      |- X19Y01
|      |- ...
|      `- X20Y10
|         |- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-tile_X20Y10-20200101-fv1.0.nc
|         |- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-tile_X20Y10-20200106-fv1.0.nc
|         |- ...
|         `- ESACCI-VEGETATION-L3S-VP_PRODUCTS-MERGED-tile_X20Y10-20201231-fv1.0.nc
|
`- metadata.yaml
```

Figure 12: Filesystem Hierarchy after repackaging

After repackaging we transfer all data from the processing centre at VITO to the CEDA Archive[7] using the open-source *NcFTP*[8] FTP client, with the following command:

```
ncftpput -R -v -u <user> -p <password> arrivals.ceda.ac.uk /CCI-Vegetation-Parameters <src-dir>
```

[7] https://archive.ceda.ac.uk/
[8] https://www.ncftp.com/ncftp