EUROPEAN COMMISSION

Community research

Climate Information Portal

CLIPC DELIVERABLE (D -N°: 4.2)

Interoperability Demonstrator

File name: { CLIPC Interoperability Demonstrator.docx}

Dissemination level: PU (public)

Author(s):*, Garin Smith (DEIMOS Ltd), Sandro Fiore, Alessandro D'Anca, Giovanni Aloisio (CMCC)*

Reviewer(s): Victoria Bennett (STFC), Maarten Plieger (KNMI)

Reporting period: RP2

Release date for review: Sep 2016

Final date of issue: 16 Nov 2016

Revision table			
Version	**Date**	**Name**	**Comments**
1.0	12/10/2016		Ready for review
1.1	10/11/2016		Updated after feedback from reviewers
1.2	14/11/2016		Incorporated section on Ophidia framework
1.3	16/11/2016		Final after reviews

Abstract

This document covers two CLIPC Activities: the first relates to interoperability between Copernicus data infrastructures, the second describes work undertaken to extend an existing big data analytics platform for implementation in CLIPC.

In the first part, work is described which has as aim to enable access between UK's Sentinel Data Access Service (SEDAS), hosted by the UK Satellite Applications Catapult, to Sentinel satellite data held at STFC-CEDA. This work shows that, using standard interfaces, the data from STFC-CEDA can be searched and made accessible via an alternate infrastructure, with broader applications in future for sharing Sentinel data across institutions (most of whom have insufficient disk storage to hold full Sentinel data archives in house), but also for combining with other (CLIPC) activities and data services for a more general cross-domain and cross-data-provider data service infrastructure.

In the second part, the development is described of a new OCC-WPS compliant interface for the Ophidia big data analytics framework at CMCC, and the service is used to compute several climate indicators with large input datasets, which are made available through the CLIPC portal.

Table of contents

Executive Summary

The following report consists of two sections. The first one is related to the interoperability between Copernicus data infrastructures; specifically, we have demonstrated that Sentinel metadata hosted in the STFC CEDA environment can be extracted and used to create a searchable catalogue in the Catapult SEDAS environment.

This has been achieved by developing two new plugins for the SEDAS/archive4EO solution. The first plugin logs onto the CEDA environment and copies Sentinel manifest files into SEDAS/archive4EO for ingestion. The second plugin allows manifest files to be ingested by SEDAS/archive4EO. archive40 is developed by Deimos Space and has been deployed to a number of live environments where it has ingested and catalogued 10s of millions of records.

If required this proof of concept could be made live fairly easily. However some extra work may be required to ensure that the STFC service listing manifest files can be applied to a date or date range to allow new manifest files to be collected using a clear handshaking protocol.

The second section of the report concerns the activities carried out to extend the set of interfaces provided by the Ophidia Framework and how it has been used to calculate impact indicators in the context of the project. Specifically, to support users for their analytics and scientific operations on large volumes of data, the Ophidia Big Data Analytics Framework has been extended to support an OGC-WPS compliant interface.

With the aim of addressing interoperability aspects (e.g. within ESGF) in terms of data access and exposed services, this interface has been developed on top of the Ophidia server providing a web-based method for remotely submitting the execution of processes to the Ophidia platform and accessing the results. By means of the WPS interface, the Ophidia framework has been used to compute some impact analysis indicator starting from large input datasets (order of tens/hundreds of Gigabytes) like the SWE climatological mean, snow-off, snow-on and Length-of-snow-season, SST mean, anomaly and climatological mean.

The resulting indicators are available on the CLIPC Portal for downloading and further analysis and combine functions.

Section 1: Sentinel data access via interoperable infrastructures

1. Acronyms

Acronyms/shortnames	Definition
Airbus	Airbus aerospace company
Catapult	Satellite Applications Catapult, based in Harwell
Deimos	Deimos Space UK Ltd
ESA	European Space Agency
SAFE	Standard Archive Format for Europe (defined by ESA)
SEDAS	Sentinel Data Access Service
STFC	Science and Technology Facilities Council
HTTP	Hypertext Transfer Protocol
CEDA	Centre for Environmental Data Analysis
CEMS	Climate, Environment and Monitoring from Space
CSW	Catalog_Service_for_the_Web
OGC	Open Geospatial Consortium
SAFE	Standard Archive Format for Europe
SOAP	Simple Object Access Protocol
STFC	Science and Technology Facilities Council
XML	Extensible Markup Language
GML	Geography Markup Language
archive4EO	Archive for Earth Observation

2. Introduction

In CLIPC Milestone 22[1] « Copernicus Infrastructures Interoperability Report », the rationale was described for trialling interoperability in a number of Copernicus data related areas (standards, datasets and Sentinel satellite data).
In the specific context of access to Sentinel satellite data, an interoperability demonstrator has been developed by Deimos UK. The aim here is to enable access between infrastructures, namely via the UK's Sentinel Data Access Service (SEDAS), hosted by the UK Satellite Applications Catpult, to Sentinel satellite data held at STFC-CEDA. This work shows that, using standard interfaces, the data from STFC-CEDA can be searched and made accessible via an alternate infrastructure, with broader applications in future for sharing Sentinel data

[1] http://www.clipc.eu/media/clipc/org/documents/milestones/clipc_milestone22.pdf

across institutions (most of whom have insufficient disk storage to hold full Sentinel data archives in house), but also for combining with other (CLIPC) activities and data services for a more general cross-domain and cross-data-provider data service infrastructure.
To build this demonstrator, some user stories were agreed, and a technical architecture was developed, coded and tested. The system's architecture, and results from the demonstrator, are presented in this document.

3. User Stories

This section is concerned with the user stories that need to be implemented and demonstrated for this deliverable.

User Story 1 – Meta data file ingestion
As a service, I want SEDAS to ingest Sentinel 1 meta data files from STFC, so that I can later search the SEDAS catalogue to discover STFC products. STFC will host an HTTP service that provides a list of all the files to ingest. STFC will host the meta data files and they will also host the products described by the metadata.

Assumptions

- Although this user story concerns Sentinel 1 meta data. I want the technical solution to be capable of also ingesting Sentinel 2 and Sentinel 3 meta data
- I want the STFC HTTP service to be protected by an authentication mechanism

User Story 2 – Meta data searching
As a user, I want to send a CSW/OGC SOAP request to the SEDAS catalogue, so that I can discover useful meta data about Sentinel 1 files ingested from STFC.

4. Assumptions

This section states the assumptions relating to this deliverable.

Manifest files to be ingested can be taken from one or more of the following lists provided by STFC-CEDA:
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/s1a_ew_grd_m.list (approximately 70,000 records)
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/s1a_iw_grd_h.list (approximately 115,000 records)
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/s1a_iw_slc.list (approximately 132,000 records)

The above URLs are taken from
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/

All products will be hosted by STFC. The SEDAS catalogue will be used to help discover products hosted by STFC.

5. Architecture Overview

This section gives an overview of the technical architecture used to implement this demonstration.

The technical solution uses SEDAS (Sentinel Data Access Solution) at its core. The figure below shows a deployment diagram of the solution. SEDAS is hosted by CEMS (Climate, Environment and Monitoring from Space). CEMS is a facility at the Satellite Applications Catapult in Harwell, Oxford

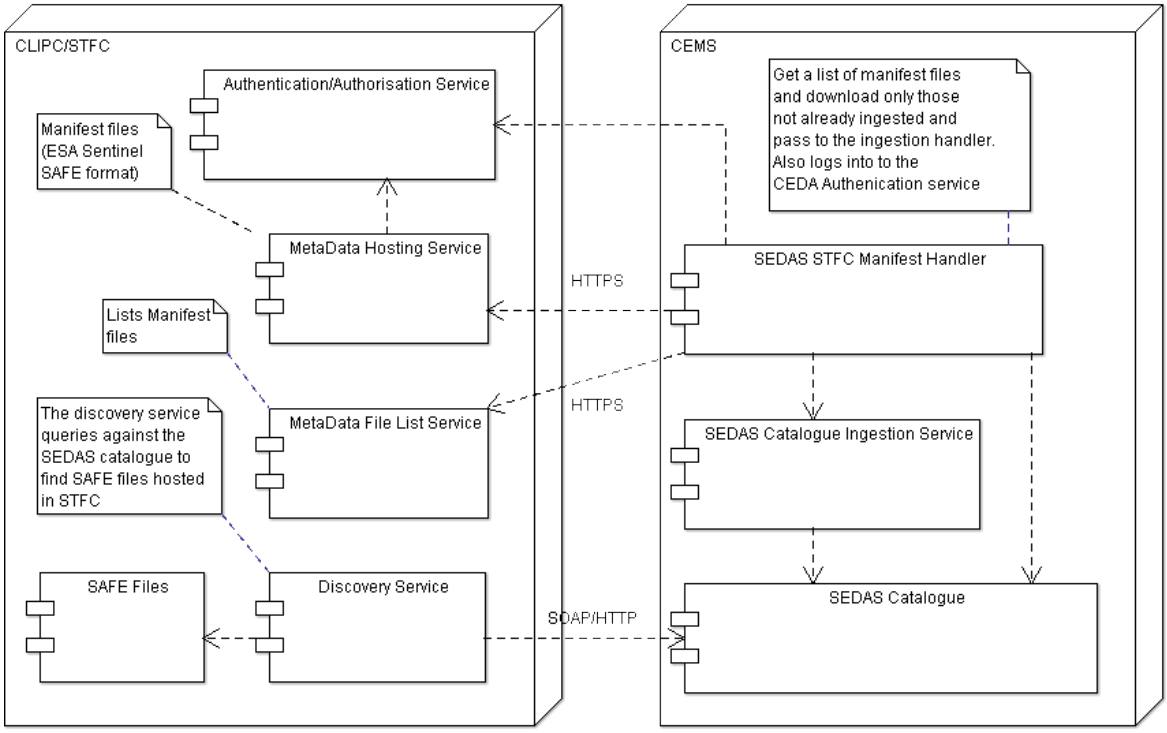


Figure 5.1 CLIPC and SEDAS Integration

In Summary

- STFC host Sentinel manifest files containing the only metadata to be ingested into the SEDAS catalogue.
- The manifest files come from ESA SAFE format files. See figure 4.2 for an example of a Sentinel 2 SAFE file format containing a manifest.safe file. The manifest.safe file is extracted from the SAFE file. This is the only file needed by SEDAS.
- STFC host an HTTP service to provide a list of manifest files ready for ingestion.
- It should be noted that only the manifest files are provided to SEDAS by STFC and only the manifest files are required by SEDAS.
- Deimos have developed an STFC Manifest Handler. This performs the following steps
    - Login to the CEDA service

- Get a list of manifest files to ingest
- Take a copy of all manifest.safe files that have not already been ingested
- Send these manifest files to the SEDAS ingestion service so they are added to the SEDAS catalogue.
- These steps are all fully automated

- CLIPC can then use the SEDAS catalogue to query STFC Metadata and use this to locate SAFE files hosted in STFC. SEDAS can be configured to give the location of the SAFE files hosted in STFC. The SAFE files are always hosted by STFC and not hosted by SEDAS for this user story.

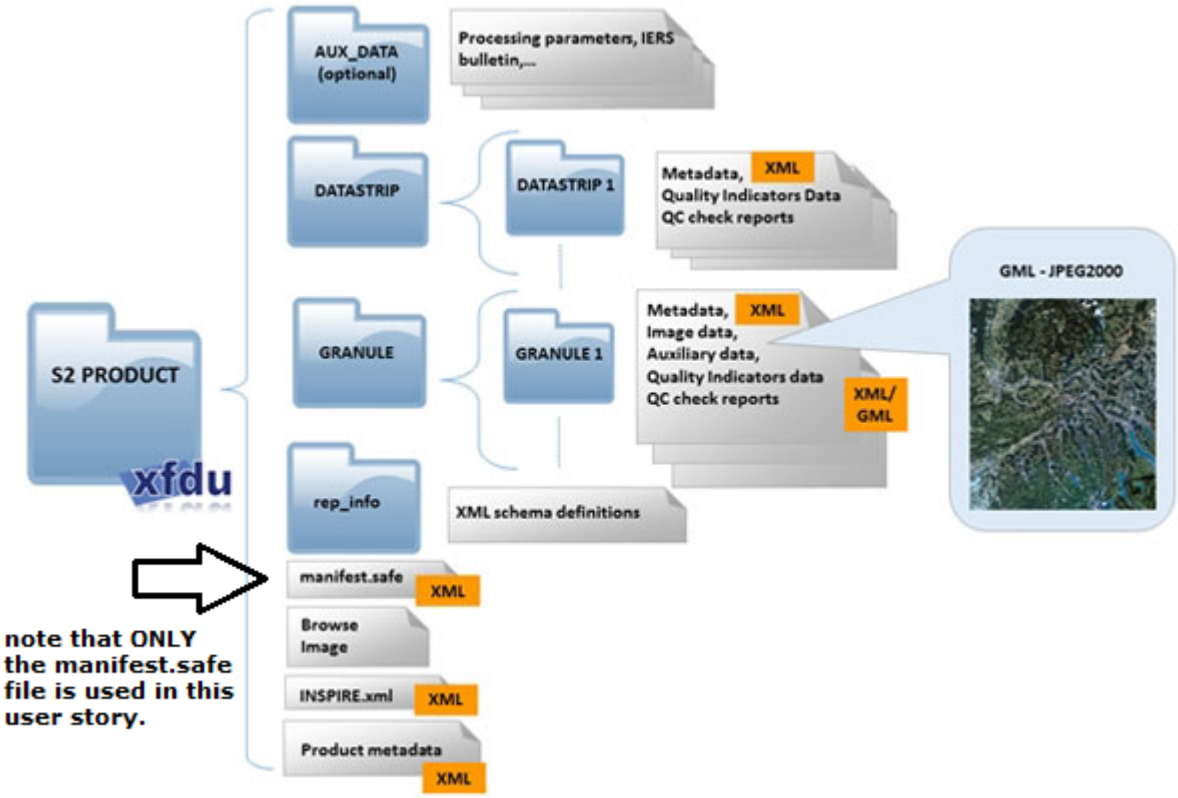


Figure 5.2 Sentinel SAFE File Format containing the manifest.safe

The following sequence diagrams show how the components in figure 4.1 are used to implement the key user stories defined in Chapter 3.

Sequence Diagram for User Story 1 – Meta data file ingestion

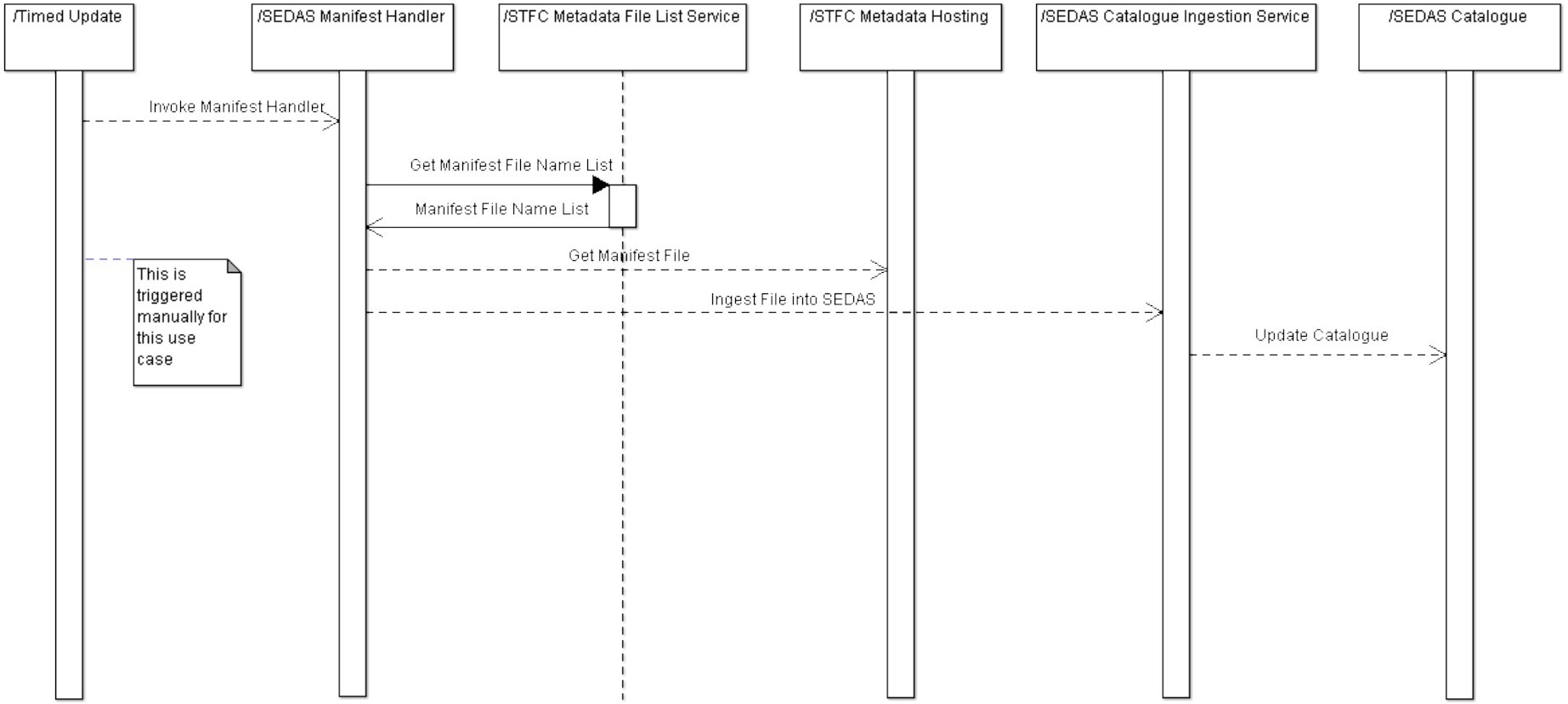


Figure 5.1 Meta data file ingestion into the SEDAS Catalogue

5.1 Sequence Diagram for User Story 2 – Meta data searching

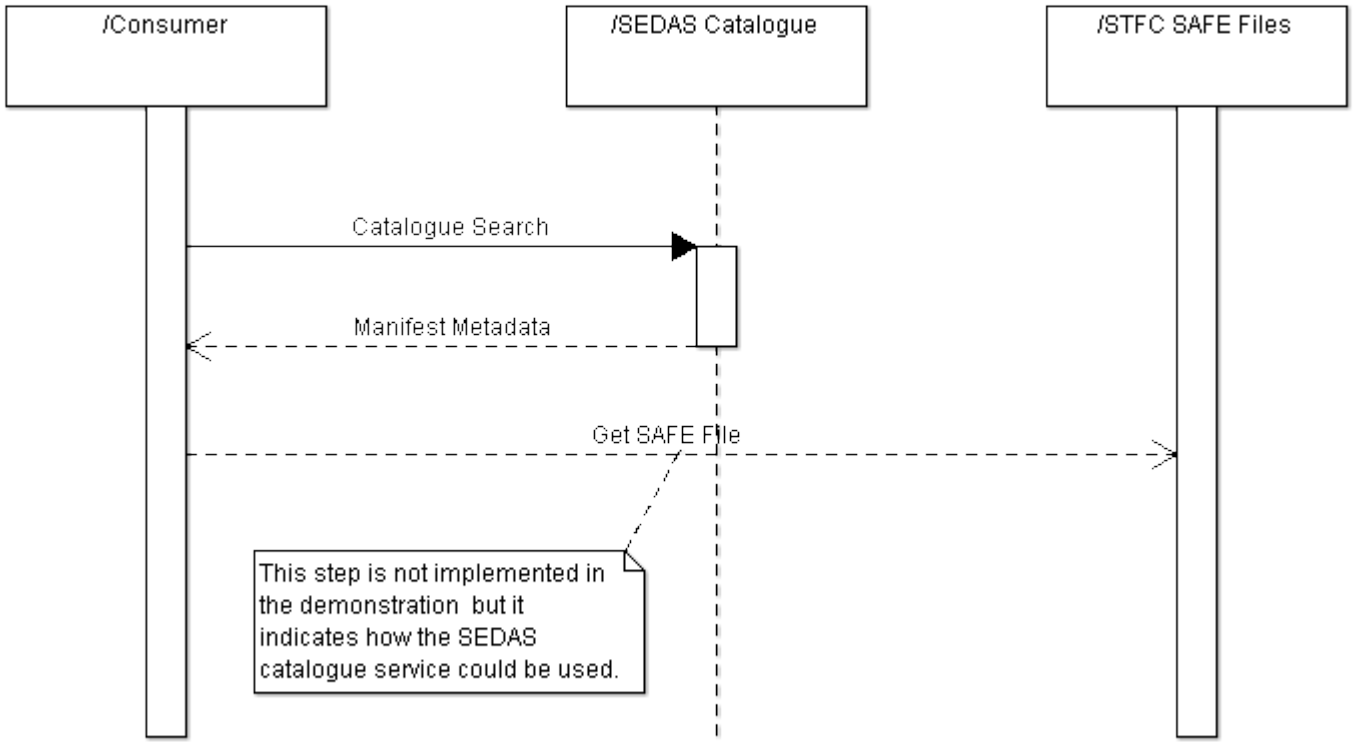


Figure 5.2 Meta data searching on the SEDAS Catalogue

6. Test Cases

The following test cases are used to demonstrate that the User Stories defined in Chapter 3.0 have been implemented correctly.

All the tests for ingestion are automated and use the following Linux command

sh loadSTFCMetadata.sh (when ingesting one or two files)

nohup sh loadSTFCMetadata.sh > ingest.log & (when ingesting many files and running in the background)

In this case the file loadSTFCMetadata.sh can be configured to determine how many manifest files are ingested.

All the tests that query the SEDAS Catalogue use a SOAP request send using the SOAPUI application. E.g.

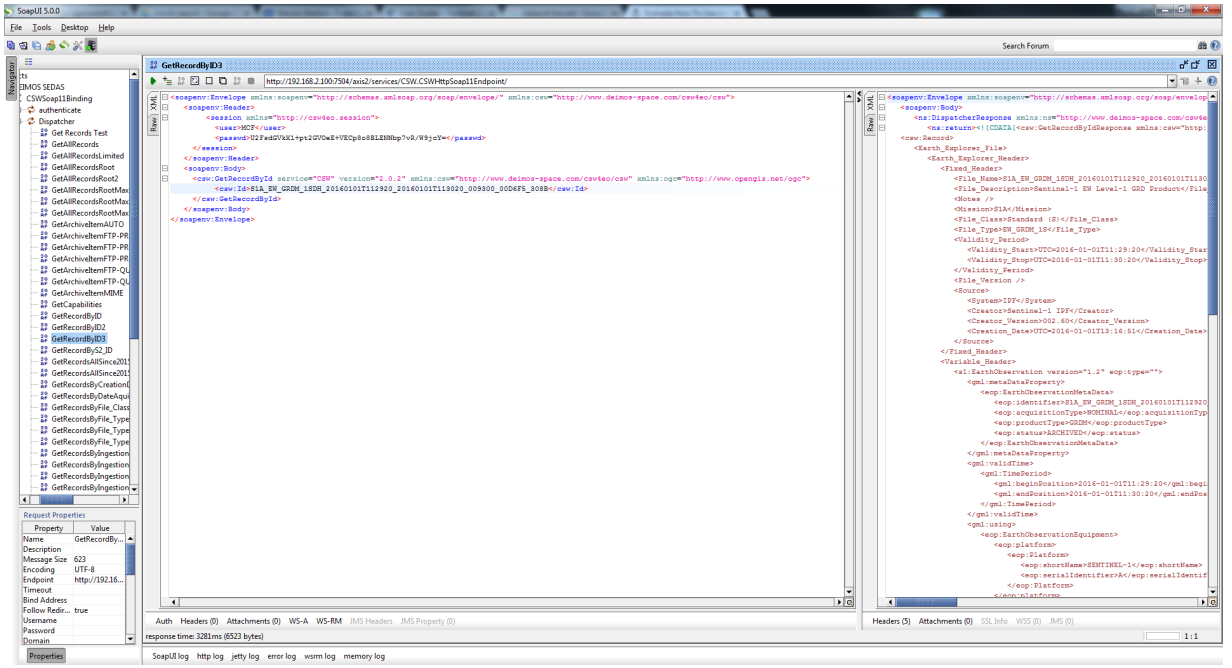
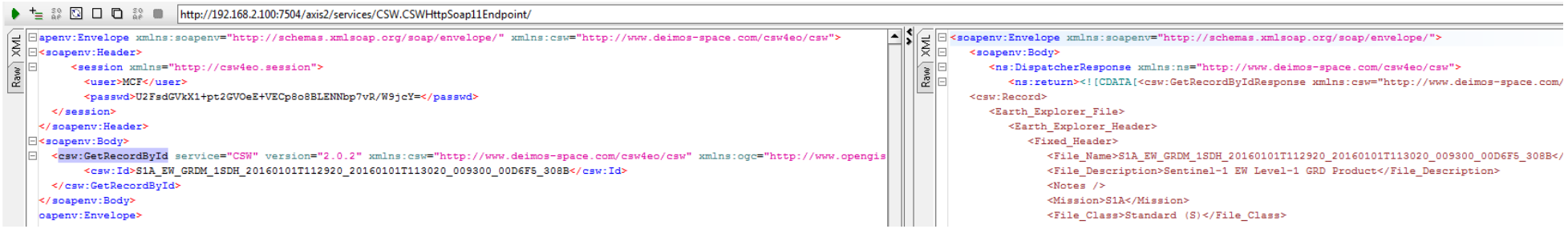


Figure 6.1 Example SOAP request

Test Case ID		UserStory1-1		
User Story Being Tested		User Story 1 – Meta data file ingestion		
Test Title		Ingest 1 new file		
Description		Show a new manifest file can be ingested and discovered. This test will 1) Login to the STFC service and get a list of manifest files available for ingestion 2) Get a copy of the first manifest file in the list 3) Ingest that file into SEDAS 4) Search SEDAS for the metadata for that file to prove that ingestion was successful.		
Modules Under Test		All Modules		
Pre-Conditions		Manifest files hosted in STFC		
Dependencies		1) SEDAS STFC manifest handler is installed and configure 2) SEDAS configured to ingest manifest files		
Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status (pass/fail)**
1	Configure manifest handler to ingest one file and run the handler script loadSTFCMetadata.sh. Check the output from the manifest handler.	The following file should be moved to ingestion by the manifest handler S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_009300_00D6F5_308B.manifest	Success "Moving S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_009300_00D6F5_308B.manifest for ingestion."	pass
2	Search for the manifest file with a CSW SOAP request to prove it was ingested. Use the csw:GetRecordById request to do this	The following file should be searchable S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_009300_00D6F5_308B.manifest	Success using csw:GetRecordById see screenshot below	Pass

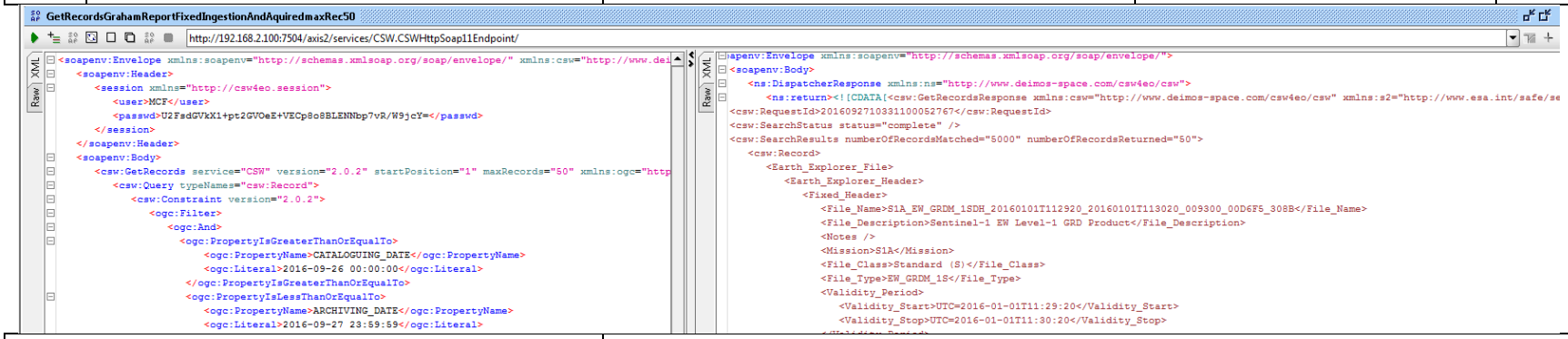


Test Case ID		UserStory1-2		
User Story Being Tested		User Story 1 – Meta data file ingestion		
Test Title		Ingest 1 new file and 1 file previously ingested		
Description		Show a new manifest file can be ingested and discovered. The files will be obtained from STFC after logging in to CEDA. Show the manifest previously ingested will not be sent for ingestion again.		
Modules Under Test		All Modules		
Pre-Conditions		Manifest files hosted in STFC		
Dependencies		1) SEDAS STFC manifest handler is installed and configure 2) SEDAS configured to ingest manifest files		
Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status (pass/fail)**
1	Configure manifest handler to ingest one file and run the handler script. Check the output from the manifest handler.	This file should be ingested by the manifest handler S1A_EW_GRDM_1SDH_20160101T112620_20160101T112720_009300_00D6F5_208C.manifest The following file should be ignored because already ingested above S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_009300_00D6F5_308B.manifest	Success "S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_009300_00D6F5_308B.manifest already ingested. Moving S1A_EW_GRDM_1SDH_20160101T112620_20160101T112720_009300_00D6F5_208C.manifest for ingestion."	pass
2	Search for the manifest file with a CSW	The following file should be searchable S1A_EW_GRDM_1SDH_20160101T112620_20160101T11272	Success using	Pass

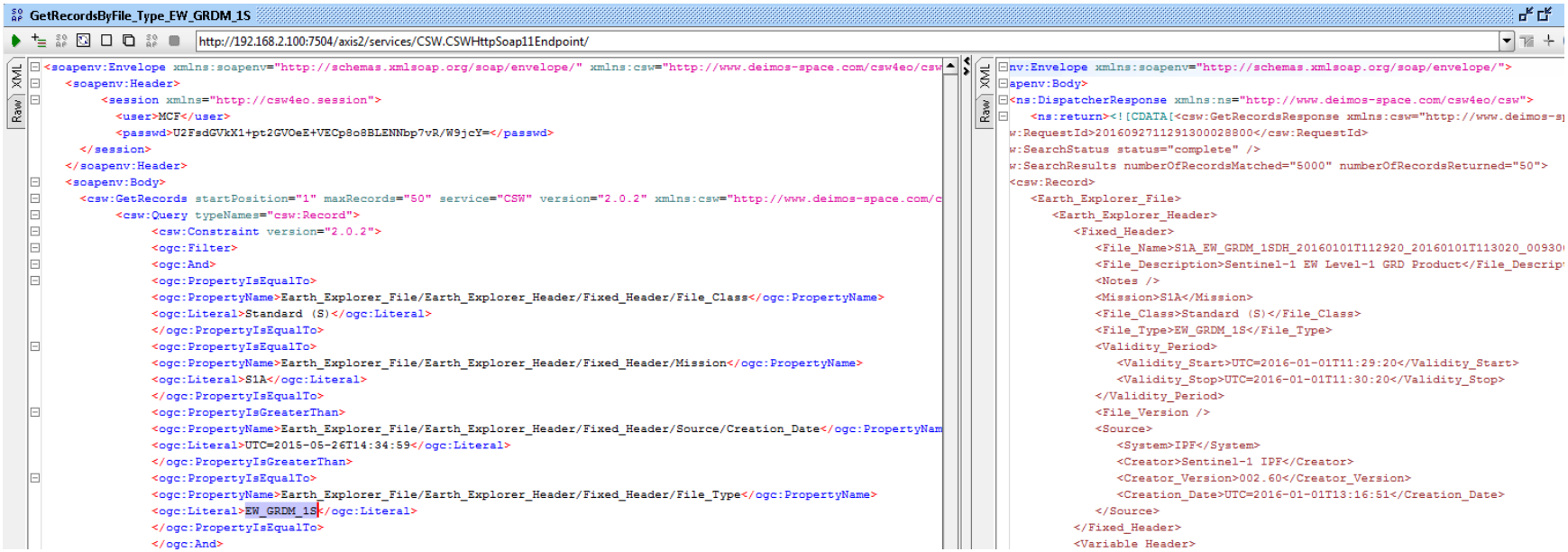
	SOAP request to prove it was ingested	0_009300_00D6F5_208C.manifest	csw:GetRecordById see screenshot below	

Test Case ID		UserStory1-3		
User Story Being Tested		User Story 1 – Meta data file ingestion		
Test Title		Ingest 5000 manifest files		
Description		Show thousands of manifest files can be ingested and discovered. The files will be obtained from STFC after logging in to CEDA.		
Modules Under Test		All Modules		
Pre-Conditions		Manifest files hosted in STFC		
Dependencies		1) SEDAS STFC manifest handler is installed and configure 2) SEDAS configured to ingest manifest files		
Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status (pass/fail)**
1	Configure manifest handler to ingest 5,000 files	The manifest handler should stop after process 5,000 files. Check output from the handler to confirm this is correct.	Success “Moving S1A_EW_GRDM_1SDH_20160128T164228_20160128T164328_009697_00E269_EA81.manifest for ingestion. Processing record number: 5000 row number: 5000 Exit - max records exceeded: 5000”	pass

2	Search for all the manifest files with a CSW SOAP request.	5,000 records should be returned. Proving they were all ingested correctly.	Success using csw: csw:GetRecords see screenshot below	Pass



Test Case ID		UserStory2-1		
User Story Being Tested		User Story 2 – Meta data searching		
Test Title		Show an OGC CSW (Catalog Service for the Web) request for products with a file type of EW_GRDM_1S		
Description		Return only EW_GRDM_1S file type products		
Modules Under Test		All Modules		
Pre-Conditions		Manifest files hosted in STFC		
Dependencies		1) SEDAS STFC manifest handler is installed and configure 2) SEDAS configured to ingest manifest files		
Step	**Test Steps**	**Expected Result**	**Actual Result**	**Status (pass/fail)**
1	Search for all Standard, S1A products with a file type of EW_GRDM_1S	Should return 50 files from 5,000.	Success See screen shot below.	pass


GetRecordsByFile_Type_EW_GRDM_1S
http://192.168.2.100:7504/axis2/services/CSW.CSWHttpSoap11Endpoint/
XML
Raw
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:csw="http://www.deimos-space.com/csw4eo/csw
<soapenv:Header>
<session xmlns="http://csw4eo.session">
<user>MCF</user>
<passwd>U2FsdGVkX1+pt2GVOeE+VECp8o8BLENNbp7vR/W9jcY=</passwd>
</session>
</soapenv:Header>
<soapenv:Body>
<csw:GetRecords startPosition="1" maxRecords="50" service="CSW" version="2.0.2" xmlns:csw="http://www.deimos-space.com/c
<csw:Query typeNames="csw:Record">
<csw:Constraint version="2.0.2">
<ogc:Filter>
<ogc:And>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>Earth_Explorer_File/Earth_Explorer_Header/Fixed_Header/File_Class</ogc:PropertyName>
<ogc:Literal>Standard (S)</ogc:Literal>
</ogc:PropertyIsEqualTo>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>Earth_Explorer_File/Earth_Explorer_Header/Fixed_Header/Mission</ogc:PropertyName>
<ogc:Literal>S1A</ogc:Literal>
</ogc:PropertyIsEqualTo>
<ogc:PropertyIsGreaterThan>
<ogc:PropertyName>Earth_Explorer_File/Earth_Explorer_Header/Fixed_Header/Source/Creation_Date</ogc:PropertyNam
<ogc:Literal>UTC=2015-05-26T14:34:59</ogc:Literal>
</ogc:PropertyIsGreaterThan>
<ogc:PropertyIsEqualTo>
<ogc:PropertyName>Earth_Explorer_File/Earth_Explorer_Header/Fixed_Header/File_Type</ogc:PropertyName>
<ogc:Literal>EW_GRDM_1S</ogc:Literal>
</ogc:PropertyIsEqualTo>
</ogc:And>
XML
Raw
nv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
apenv:Body>
<ns:DispatcherResponse xmlns:ns="http://www.deimos-space.com/csw4eo/csw">
<ns:return><![CDATA[<csw:GetRecordsResponse xmlns:csw="http://www.deimos-s
w:RequestId>20160927112913000288800</csw:RequestId>
w:SearchStatus status="complete" />
w:SearchResults numberOfRecordsMatched="5000" numberOfRecordsReturned="50">
<csw:Record>
<Earth_Explorer_File>
<Earth_Explorer_Header>
<Fixed_Header>
<File_Name>S1A_EW_GRDM_1SDH_20160101T112920_20160101T113020_00930
<File_Description>Sentinel-1 EW Level-1 GRD Product</File_Descrip
<Notes />
<Mission>S1A</Mission>
<File_Class>Standard (S)</File_Class>
<File_Type>EW_GRDM_1S</File_Type>
<Validity_Period>
<Validity_Start>UTC=2016-01-01T11:29:20</Validity_Start>
<Validity_Stop>UTC=2016-01-01T11:30:20</Validity_Stop>
</Validity_Period>
<File_Version />
<Source>
<System>IPF</System>
<Creator>Sentinel-1 IPF</Creator>
<Creator_Version>002.60</Creator_Version>
<Creation_Date>UTC=2016-01-01T13:16:51</Creation_Date>
</Source>
</Fixed_Header>
<Variable Header>

7. Conclusions

In conclusion, we have demonstrated that we can successfully implement the user stories using the tools we have selected and the new plugins we have developed to enable us to communicate with the STFC technical interface.

The next section describes how we might want to implement this demonstrator in an operational environment.

8. Forward look: Implementing an Operational Environment

This section describes how this demonstrator could be made operationally live if required.

8.1 SEDAS and archive4EO

The SEDAS ingestion engine and catalogue is based upon archive4EO. Archive4EO has been developed by Deimos Space.

archive4EO has been deployed to a number of live environments containing 10s of millions of records. The ingestion time in live is around 2 seconds for a typical SAFE file, however the time to only ingest a manifest file will be quicker.

archive4EO is operationally live and running in the CEMS environment operated by the Satellite Applications Catapult, based in Harwell. Currently archive4EO ingests manifest files provided by Airbus.

8.2 STFC Manifest Handler

This section describes the SEDAS STFC Manifest Handler used in this demonstration.

The manifest handler can be configured to operate from the command line as follows
nohup sh loadSTFCMetadata.sh s1a_ew_grd_m.list > nohup.txt &

The manifest handler is a shell script that is responsible for collecting manifest files from STFC using the STFC Metadata File List Service.

Note that

- The file name containing the manifest file list can be passed as a parameter.
- The manifest handler can be configured to read only the first x number of records if required.
- The manifest handler could be triggered as a regular scheduled job if required. E.g. nohup sh loadSTFCMetadata.sh <date_value> > nohup.txt &

8.3 Operational Options

This sections describes how SEDAS and archive4EO could be configured to support the indexing of STFC manifest files if required.

a) Standalone Mode

A separate instance of SEDAS (ingestion and catalogue) could be deployed just for cataloguing STFC files.

b) Combined Mode

The existing SEDAS service could ingest product from both Airbus and STFC. In this case it might make sense for the catalogue to know the source of the product. So for instance we could record the source of the metadata being either STFC or Catapult.

8.4 STFC Manifest List

It might be useful to query all manifest files by date or time so that they could be ingested on a daily basis. E.g. rather than
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/s1a_ew_grd_m.list
have
http://browse.ceda.ac.uk/browse/neodc/sentinel1a/metadata/master_list/s1a_ew_grd_m?date=28091026.

This could be implemented as a REST API. We could then run a daily ingestion job.

Section 2: Extensions to the Ophidia Big Data analytics framework, and implementation in CLIPC

9. Introduction

The Ophidia Big Data Analytics Framework[1][2][3] has been designed and developed for providing a platform addressing scientific use cases related to large volumes of multidimensional data. In this context, the Ophidia project aims at providing a big data analytics solution running data intensive tasks on High Performance Computing (HPC) environment to address scientific needs in (near) real-time. Ophidia provides the user the data cube abstraction to deal with large amount of data: it provides both a set of data cube primitives (to manipulate the multidimensional datasets) and a workflow support to define more complex operators.

In the climate change domain the Ophidia platform has been applied in different national and international projects and efforts, as in the context of the Coupled Model Intercomparison Project Phase 5 (CMIP5) by performing scientific analysis of the global climate simulations produced at the Euro-Mediterranean Centre on Climate Change. In the CLIPC project a new OGC-WPS[4] compliant interface for processing activities regarding climate data has been developed on top of the Ophidia Server. The processing service has been used to compute several indicators with large input datasets, like Snow Water Equivalent (SWE) climatological mean, snow-off, snow-on and Length-of-snow-season, Sea Surface Temperature (SST) mean, anomaly and climatological mean. The outputs of these indicators have been published and are available on the CLIP-C data platform.

10. The Ophidia software stack

The Ophidia architecture consists of different layers.

The lower layer is the storage system which represents the hardware resources managing the data store of the Ophidia architecture. It consists of a set of storage devices storing information in the form of datacubes.

The storage system is accessed via the I/O nodes exploiting a set of I/O servers which implement the parallel I/O feature for accessing the underlying storage system. The current implementation of the I/O server relies on a MySQL DBMS[5]; it exploits the Ophidia primitives to support array-based data type and functionalities. The datacubes managed by the I/O servers are organized in a hierarchical structure and are partitioned in several tables (or fragments) distributed across multiple databases, multiple instance of MySQL servers running on different hosts. The compute nodes are used by the Ophidia framework to run the data analytics processes, which are based on a set of (parallel and sequential) operators providing several functionalities for accessing and manipulating data and metadata. These operators run in parallel or in a sequential way. Each operator activated on the compute nodes contacts the

I/O node layer to access, analyze and transform datacubes as a whole, exploiting the underlying Ophidia primitives. So, while the compute node layer performs the analysis operations at datacube level, it relies on the I/O nodes for the actual manipulation of the n-dimensional data. To interact with the back-end layers, the Ophidia server has been developed; it represents the front-end of the Ophidia platform exposing a standard web service interface. The Ophidia server manages the user authentication and authorization, allows the remote Ophidia operator submissions (dispatching jobs and managing queues and priorities) and provides also a complete user session management support.

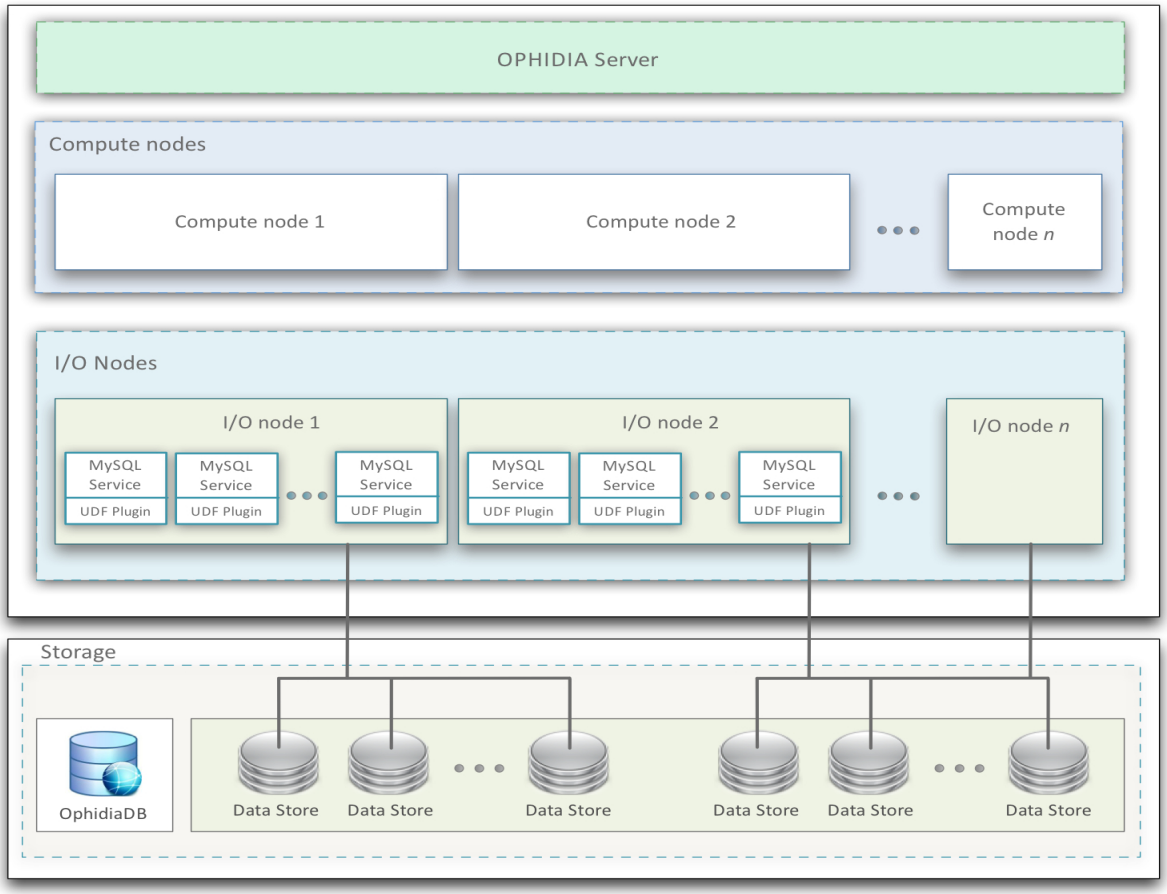


Figure 1. The Ophidia Analytics Framework software stack

11. WPS Implementation

The OpenGIS Web Processing Service (WPS) Interface Standard offers a web-based method for starting the execution of remote processes and access to its results providing the rules for the standardization of the inputs and outputs (in terms of requests and responses). Processes include any kind of computation that operates on spatially georeferenced data (raster or vector). In these terms, a service which exposes a WPS interface allows simple or complex computations, remotely triggered by a web request. Defining an interface for publishing of

geospatial processes on the web, the WPS specification regulates the way a client should request a specific execution and how the outputs will be managed.

A WPS service is based on three methods:

- GetCapabilities
- DescribeProcesses
- Execute

The GetCapabilities method is used to obtain the list of processes that can be launched; its response is an XML document containing the features of the provided services (title, abstract, keywords, etc.), the description of the service provider (name, site, contacts, etc.) and list of the supported processes (identifier, title, abstract, etc.).

The DescribeProcesses method is used to obtain the description of one or more processes (included in the GetCapabilities response) in terms of input parameters, output parameters, data types required, default values, etc. The DescribeProcess response is an XML document containing a brief description of the process (identification, title, abstract, etc.), the list of the input parameters (data type, default value, if it is mandatory, etc.), the list of the output parameters (with the related data type).

The Execute method is used to start the execution of a process (included in the GetCapabilities response). The request can be sent via an XML document, or via a REST command. The Execute request contains the values of the input parameters and how the server will manage the outputs (sent back to the client or stored on the server).

The WPS implementation of Ophidia relies on the PyWPS module[6], Python Web Processing Service, which is an Apache embedded Python module enabling a WPS interface. The Ophidia WPS interface implements a process for PyWPS, which translates the WPS Request into JSON Request for the Ophidia Web Service interface and replies to the WPS client encapsulating the related JSON Response within the WPS Response.

The WPS module for Ophidia is formed by two Python scripts which implement a WPS Python process. The first script (*ophidia.py*) contains the WPS interface definition with references to the process input and output parameters and their data types. Specifically, these parameters are:

- Input
  - Userid: represent the Ophidia identifier of the user who requests the submission;
  - Passwd: represent the Ophidia password of the user who requests the submission;
  - Request: contains the JSON file corresponding to the Ophidia user request: it could be related to the execution of a single operator or a complex workflow of tasks.
- Output
  - Jobid: represent the job identifier as generated by the Ophidia Framework;
  - Response: contains the JSON file corresponding to the response as generated by the Ophidia Framework at the operator (or workflow) completion.

The *execute* process has been implemented in order to be as much as possible independent from the specific operator (or workflow) the user submits. In this perspective, the definition of

the operations to perform is described in the JSON file (representing the Ophidia submission) embedded within the WPS request. This allows to:

1. define a single JSON file containing the sequence of the Ophidia operators, valid for both a WPS or a Web Service submission;
2. exploit the same *execute* method to submit each operator (or combination of operators in a workflow) provided by the Ophidia platform.

The implemented *execute* method relies on a generic *submit* function defined within the second python script *ophsubmit.py*. This function represents the core of the workflow submission to the Ophidia framework exploiting a WPS interface. As stated before, it translates the incoming WPS request in a JSON request able to be accepted by the underlying Ophidia Server. At the job completion, it extracts from the Ophidia response the generated job identifier providing the client the Response and the Jobid outputs.

The following image shows an example of interaction between a generic WPS client and the Ophidia Server exploiting the WPS interface.

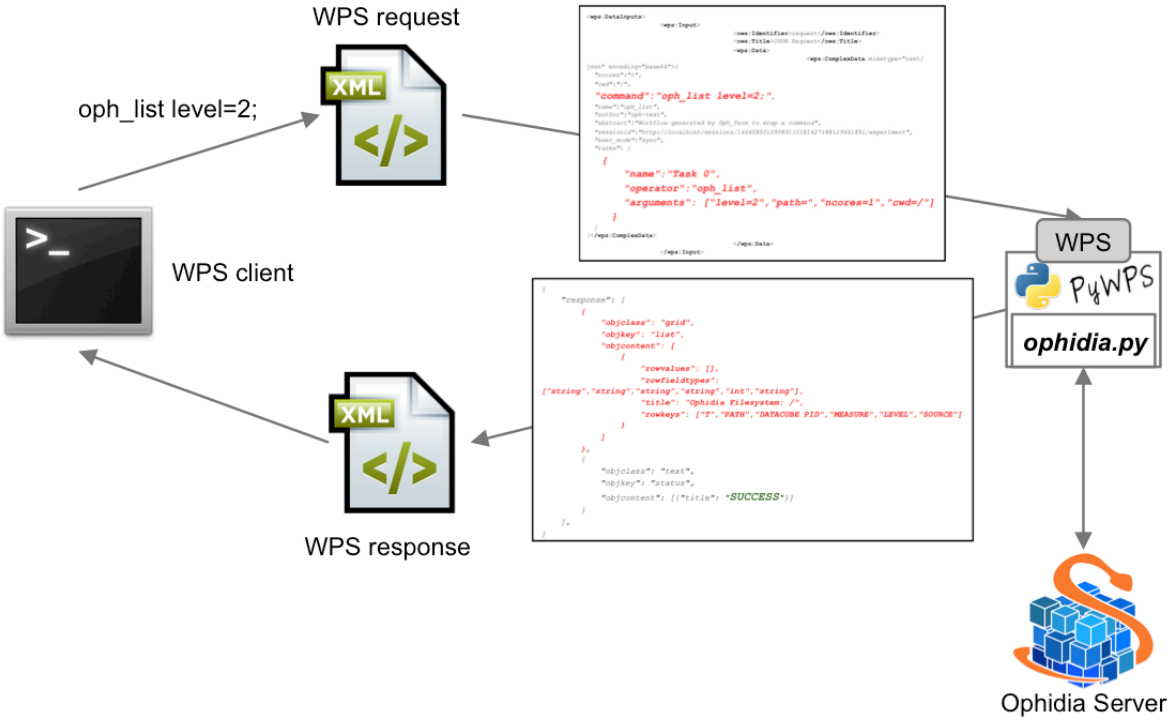


Figure 2. Interactions between a WPS client and the Ophidia WPS interface

As shown in the previous image, the WPS request sent by the client contains the Ophidia submission request in the form of a JSON formatted text; in the same way, the *ophidia.py* module embeds the JSON formatted Ophidia response within the WPS response.

12. CLIPC climate indicators computation

The Ophidia WPS interface has been used to compute several CLIPC indicators starting from large input datasets. The computed indicators required to define a workflow of operations where some tasks are performed in parallel exploiting the support for parallel executions offered by Ophidia. As previously mentioned, these workflows have been defined by means of a file in JSON format in which the different operations to be performed are listed, the dependencies among them and the level of parallelism to use during the run. It is worth noting that the submission of an Ophidia workflow relies on two different levels of parallelism: the first is related to individual jobs (Ophidia operators) that can be submitted using multiple cores. The second level allows to run multiple jobs in parallel if there are no flow dependencies among them. The definition of an Ophidia workflow allows to define both these levels of parallelism by setting some parameters within the related JSON file.

Specifically the following indicators have been computed:

- SWE (Snow Water Equivalent) climatological mean starting from 1.7GB of input data;
- Snow-off, Snow-on and Length-of-snow-season starting from 50GB of input data;
- SST (Sea Surface Temperature) mean, anomaly and climatological mean starting from 350GB of input data.

In the following the workflows and the procedures used to compute these indicators will be illustrated.

12.1 SWE monthly climatological mean

The objective of this task is to compute the Snow Water Equivalent monthly climatological mean starting from a set of input data between October 1979 and June 2008.

Input data are classified as “ESA GlobSnow snow water equivalent L3B monthly aggregated” and come from FMI - Finnish Meteorological Institute. The considered input variable is the Snow Water Equivalent, specifically “Snow water equivalent values on 25 km by 25 km Equal Area Scalable Earth (EASE)-grid, produced by assimilating passive radiometer data with snow depth information from synoptic weather station network. SWE information is provided for terrestrial non-mountainous regions of Northern Hemisphere, excluding glaciers and Greenland” (as extracted from the NetCDF input files).

The dataset involved is formed by 282 input files, one for each month between the considered dates (information about some months are missing). In total the entire dataset occupies 1.7 GB. In order to illustrate through a schematic representation the workflow of the operations needed to compute the indicator the DAWML language has been used. It allows to represent in detail the data analytics workflows and the tasks involved in a simple and intuitive way.

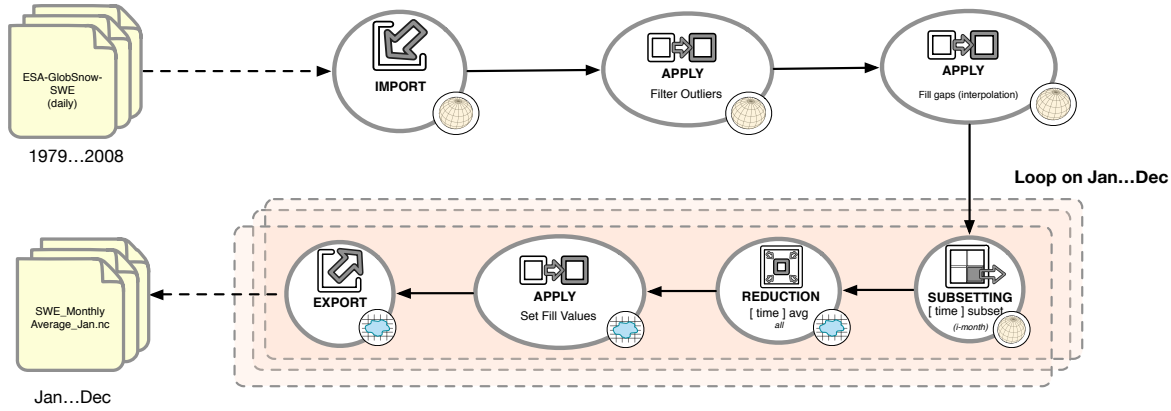


Figure 3. DAWML schema for the SWE climatological mean workflow

The DAWML schema shows the sequence of the tasks performed by the Ophidia framework for the comuptation of the SWE climatological mean.

The first phase corresponds to the import of the input files in the Ophidia platform data storage. Then some preprocessing operations are applied to the just imported input data in order to filter the outlier values and fill the gaps due to the missing values. In the second stage, a loop performed on each month (Jan, ..., Dec) allows to extract the climatological mean by a subset on each considered month and a reduction to compute the related mean.

A final step exports in a set of NetCDF files (one for each month) the output dataset.

The related JSON description of the involved tasks contains a more in detail definition of the operations to submit to the Ophidia platform: it represents the input given to the system to submit the workflow through the Ophidia framework. As stated before, it describes the tasks to be performed defining the dependencies among them: the Ophidia server will adapt the scheduling of the jobs accordingly to the flow of the depedencies defined within the JSON file.

Another kind of representation of the same workflow could be extracted directly from the Ophidia terminal using the command "*check <wf_json_file>*". In this case, the input JSON file will be automatically translated in a visual representation corresponding to the list of the operations to execute. Different forms are used to show tasks and loops.

The following images correspond to a section of the JSON workflow used to start the submission and the visual representation of the workflow using the command check.

```
{
    "name": "Loop on months",
    "operator": "oph_for",
    "arguments": [
        "name=index",
        "counter=1:12",
        "values=Oct|Nov|Dec|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep",
        "parallel=yes"
    ],
    "dependencies": [
        { "task": "Filter2", "type": "single" }
    ]
},
{
    "name": "Extract months",
    "operator": "oph_subset",
    "arguments": [
        "subset_dims=time",
        "subset_filter=&index:12:end",
        "description=Data related to mounth @index"
    ],
    "dependencies": [
        { "task": "Loop on months", "type": "single" }
    ]
},
{
    "name": "Evaluate average",
    "operator": "oph_reduce2",
    "arguments": [
        "operation=avg",
        "dim=time",
        "description=Average of data related to @index"
    ],
    "dependencies": [
        { "task": "Extract months", "type": "single"}
    ]
},
```

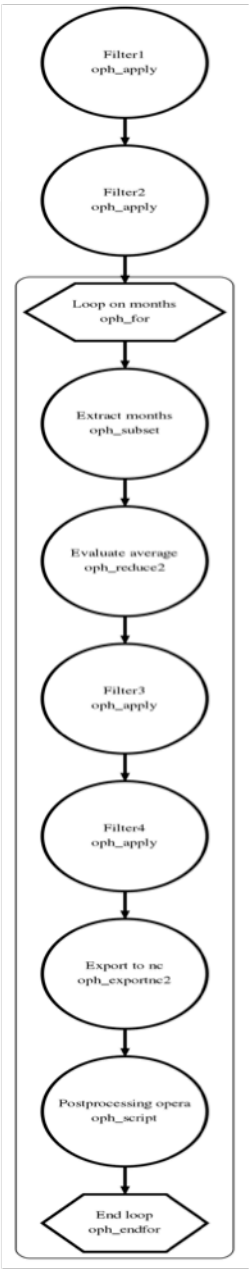


Figure 4. SWE JSON schema and visual representation of the Ophidia workflow

The execution of the workflow can be monitored using the command "*view*" in the Ophidia terminal. In this case, the terminal shows an animation corresponding to the flow of the execution during the processing. Specifically, the green circles represents the completed tasks, the orange circles the running tasks while the grey circles the pending tasks (for dependencies or resources availability reasons).

In total 76 tasks are performed and the output of the workflow consists of 12 NetCDF files, each one 6MB in size; the output dataset occupies 72 MB in total. The following images represent a snapshot extracted during the workflow execution and a map related to the SWE climatological mean in the month of February (created by Adaguc[7]).

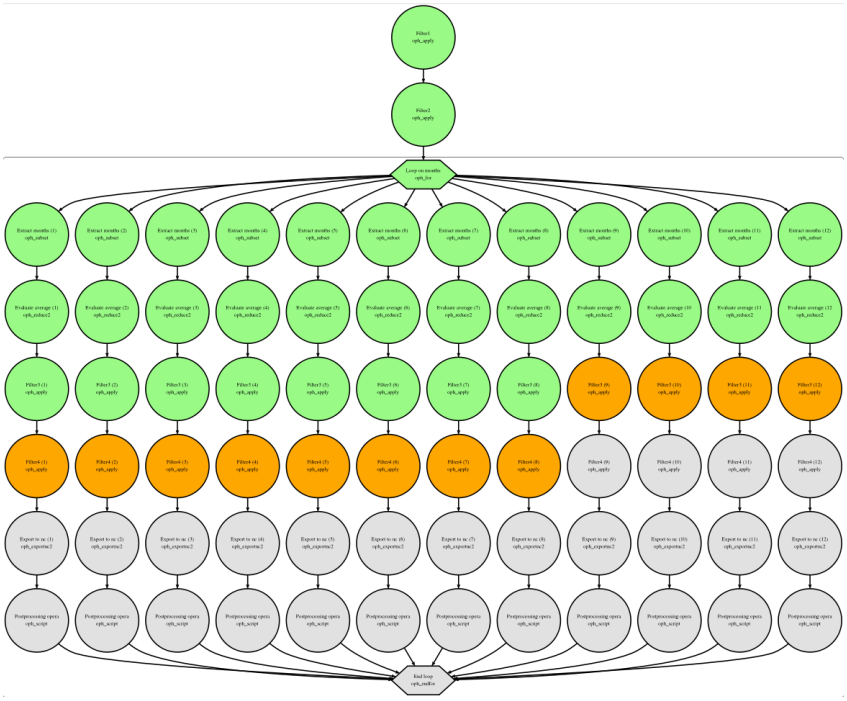


Figure 5. SWE workflow execution

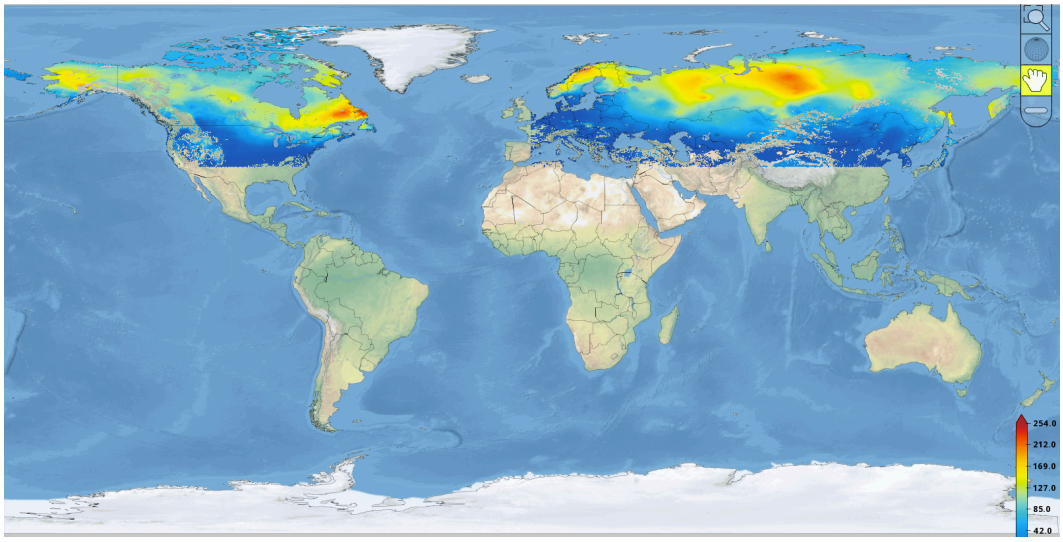


Figure 6. SWE February climatological mean

12.2 Snow-off, Snow-on and Length of Snow season

The second group of indicators presented in this report and obtained exploiting the WPS interface deployed on top of the Ophidia platform are the Snow-off, Snow-on and Length of Snow season. These indicators have been grouped in a single one because each of them relies

on the same set of input data and, in addition, a single Ophidia workflow can be used to perform the computation.

Input are ESA Globsnow Snow Water Equivalent data (L3A), with a spatial resolution of 25x25 sq. Km evaluated on the Northern emisphere. The dataset comes subdivided in 6341 NetCDF files, one of each day in the period between 1979-09-11 and 2012-12-31; even in this case, many input files are missing. The total size of the input dataset is 50GB.

From a scientific point of view, we define Snow Season the period between the 1st of July and the 30th of June of the next year. The objective of this workflow is to compute, within a single execution the following indicators:

- Snow-on: First day of snow season when the snow water equivalent is greater than zero;
- Snow-off: Day with SWE=0 after last day with SWE>0;
- Length of Snow Season: (Snow-off) - (Snow-on).

As for the previous case, a DAWML schema has been produced to represent the workflow to submit.

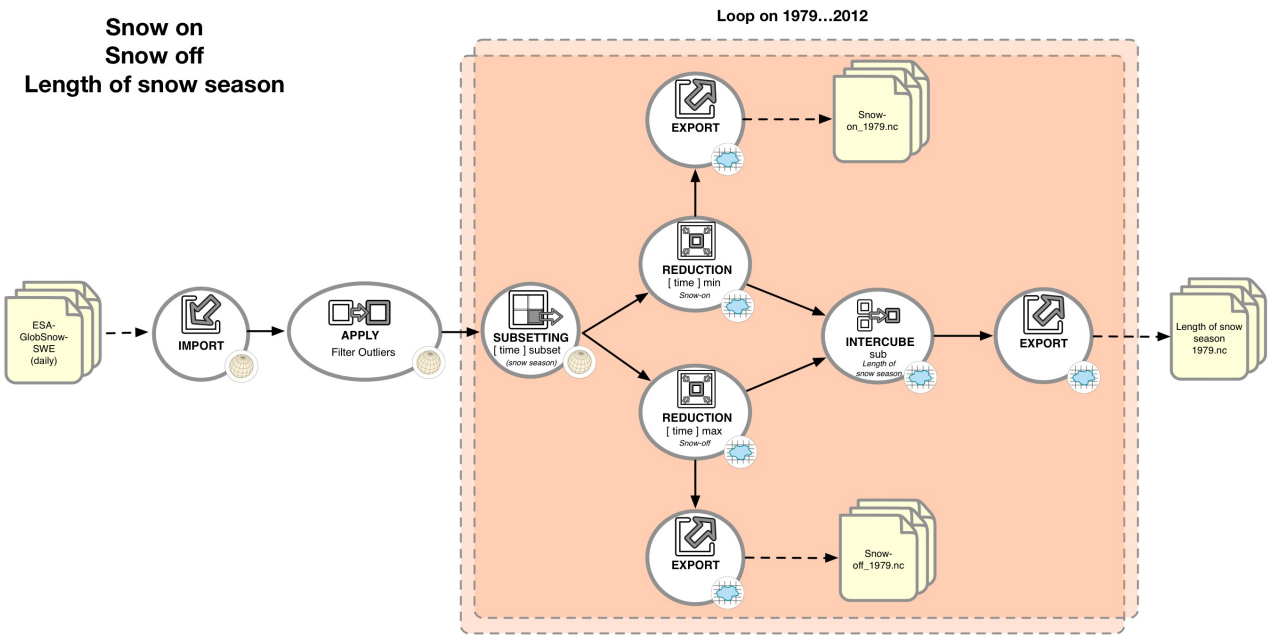


Figure 7. DAWML schema for the Snow-off, Snow-on, Length of snow season workflow

Contrary to what has been described in the previous case, as shown in the figure, three different branches start from a single initial chain; each of them is related to one of the previously mentioned indicators.

The first step corresponds to the import of the input files in the Ophidia platform data storage. Then some preprocessing operations are applied in order to filter the outlier values. In the second stage, a loop performed on each year (1979-2012) is performed. For each year the

related snow season is selected then the workflow splits itself in two separate branches. The first branch performs a minimum operation to compute the snow-on while the second one exectues a maximum operation to extract the snow-off. Both these branches complete the execution exporting 33 NetCDF files for each one related to the snow-on and snow-off dates for the considered years. A third branch performs a date subtraction between snow-on and snow-off exporting the resulting 33 files related to the Length of Snow Season indicator.

The following images correspond to a section of the JSON schema used and the visual representation of the workflow using the command *check*.

```
{
   "name": "Extract year",
    "operator": "oph_subset2",
    "arguments": [
        "subset_dims=time",
        "subset_filter=@{year}",
        "description=Selection of year @{year}"
    ],
    "dependencies": [
        { "task": "Loop on years", "type": "single" }
    ]
},
{
   "name": "Snow on",
    "operator": "oph_reduce",
    "arguments": [
        "operation=min",
        "description=Selection of Snow on for @{year}"
    ],
    "dependencies": [
        { "task": "Extract year", "type": "single" }
    ]
},
{
    "name": "Snow off",
    "operator": "oph_reduce",
    "arguments": [
        "operation=max",
        "description=Selection of Snow off for @{year}"
    ],
    "dependencies": [
        { "task": "Extract year", "type": "single" }
    ]
},
{
    "name": "Evaluate Max-Min",
    "operator": "oph_intercube",
    "arguments": [
        "output_measure=length",
        "operation=sub",
        "description=Length snow for @{year}"
    ],
    "dependencies": [
        { "task": "Snow off", "type": "single" },
        { "task": "Snow on", "type": "single", "argument":"cube2" }
    ]
}
```

Figure 8. Snow-off, Snow-on, Length of snow season JSON schema

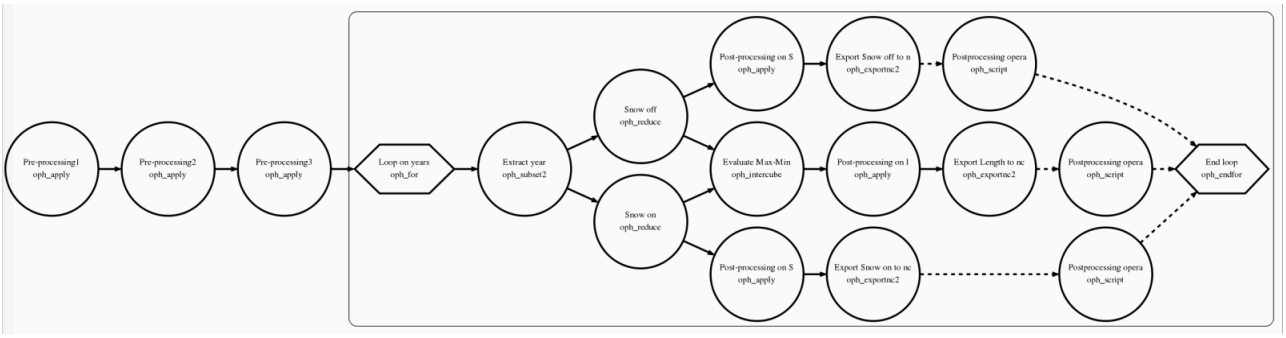


Figure 9. Snow-off, Snow-on, Length of snow season visual representation of the Ophidia workflow

In total this workflow performs 599 tasks producing 99 files (6MB each, 594 MB totally).

As in the previous case, a snapshot extracted during the execution (monitored with the Ophidia terminal command *view*) is shown. In addition, it is provided a map of the snow-off date related to the snow season 2004-2005 created using Adaguc.

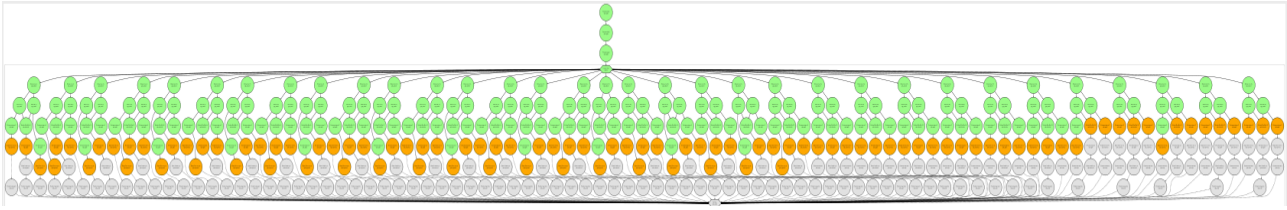


Figure 10. Snow-off, Snow-on, Length of snow season workflow execution

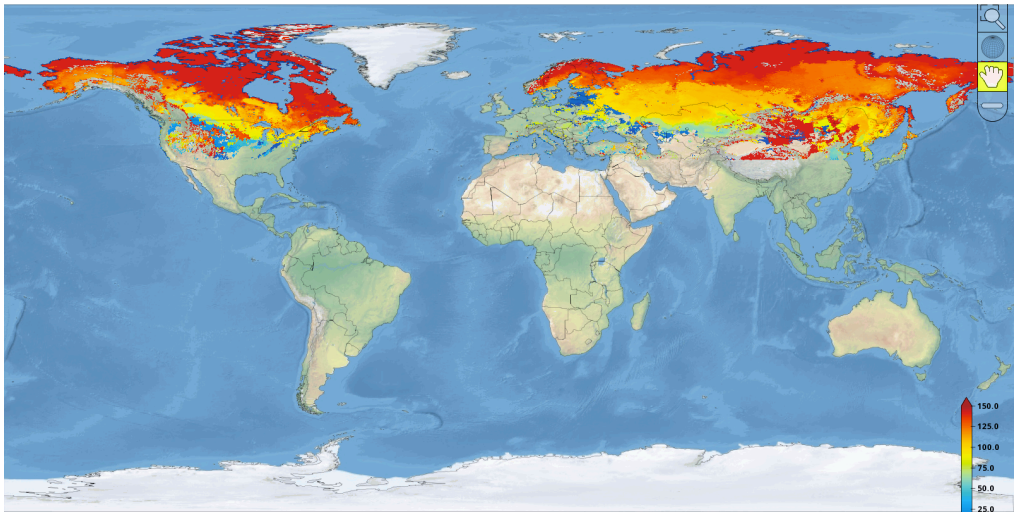


Figure 11. Snow-off related to 2004-2005

12.3 SST climatological mean, anomaly, mean

The third group of indicators is related to the SST (Sea Surface Temperature) variable; specifically they represent the SST mean, monthly climatological mean and anomaly. As for the previous one, these indicators have been grouped in a single execution (and workflow) as they rely on the same set of input data.

Input are ESA SST CCI OSTIA L4 products (OSTIA L4 product from the ESA SST CCI project, produced using OSTIA reanalysis system v2.0) with a spatial resolution of 0.05 degrees (global domain). The input dataset is formed by 7062 NetCDF files, one of each day in the period between 1991-09-01 and 2010-12-31. The total size of the input dataset is 343 GB of uncompressed data.

The worklow used for the computation produces the three mentioned indicators exploiting three parallel branches.

The next image shows the DAWML schema related to the workflow.

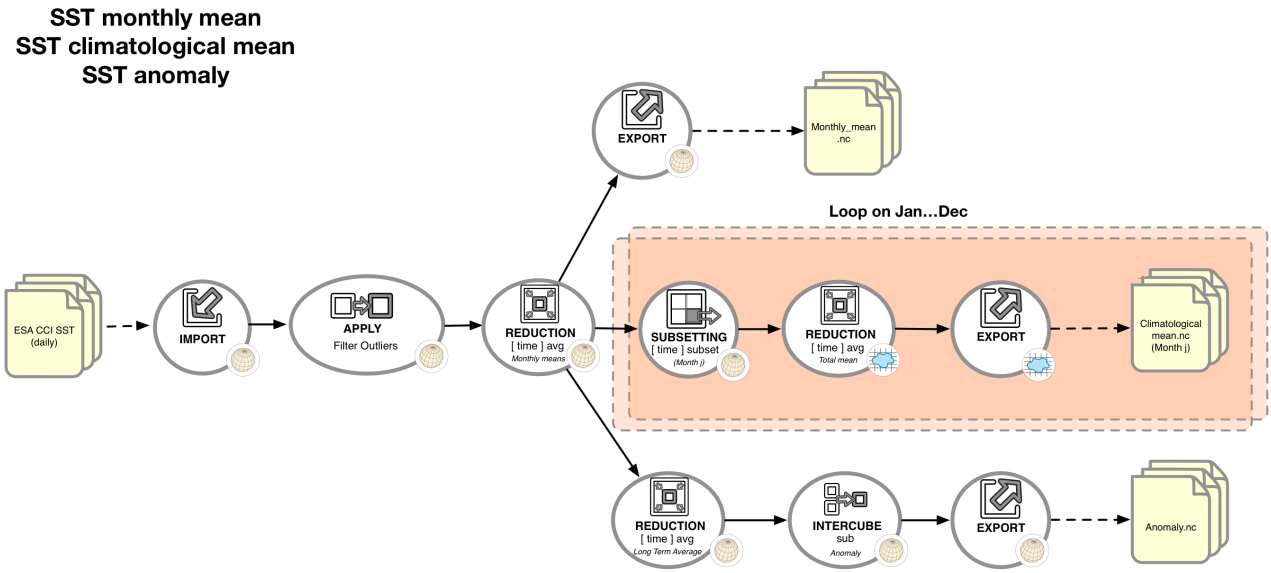


Figure 12. DAWML schema for the SST workflow

As the previous image shows, only a single branch of the workflow is involved in a loop in order to compute the monthly climatological mean for each month in the considered period.

As usual, the first step is the import of the input files in the Ophidia platform data storage and some preprocessing operations in order to filter the outlier values. Then a global monthly mean is computed. The result of this operation corresponds to the first indicator so data are exported in a NetCDF file. The middle branch of the workflow represents the monthly climatological mean computation. Using a loop, for each month (Jan,..., Dec) an Ophidia reduction operator computes the mean on the considered period (1991-2010). The resulting outputs are exported as NetCDF files. The last chain corresponds to the anomaly computation

as the difference between the previously extracted global monthly mean and the long term average (exploiting the intercube operator). The result is exported in a NetCDF file.

The following images correspond to a section of the JSON workflow and the visual representation of the workflow using the command check within the Ophidia terminal.

```
        {
            "name": "Loop for subset years",
            "operator": "oph_for",
            "arguments": [ "name=index","counter=1:20", "values=2010-01-01_2011-01-01",
"parallel=yes" ]
        },
        {
            "name": "Subset on 20 years",
            "operator": "oph_subset2",
            "arguments": [
                "subset_dims=time",
                "subset_filter=@index",
                "cube=$1"
            ]
             "dependencies": [
                { "task": "Loop for subset years",
                  "type": "single"}
            ]
        },
        {
            "name": "Compute annual mean SST over time",
            "operator": "oph_reduce2",
            "arguments": [
                "dim=time",
                "operation=avg"
            ],
            "dependencies": [
                {"task": "Subset on i-year",
                "type": "single"
            }]
        },
        {
                "name" : "Compute anomalies",
                "operator" : "oph_intercube",
                "arguments" : [
                        "operation=sub",
                        "output_measure=anomalies" ],
                "dependencies" : [
                {
                        "argument" : "cube", "task" : "Compute annual mean SST over time",
                        "type" : "single"},
                {
                  "argument" : "cube2", "task" : "Compute long-term mean SST over time",
                  "type" : "single" }]
        }
```

Figure 13. SST JSON schema

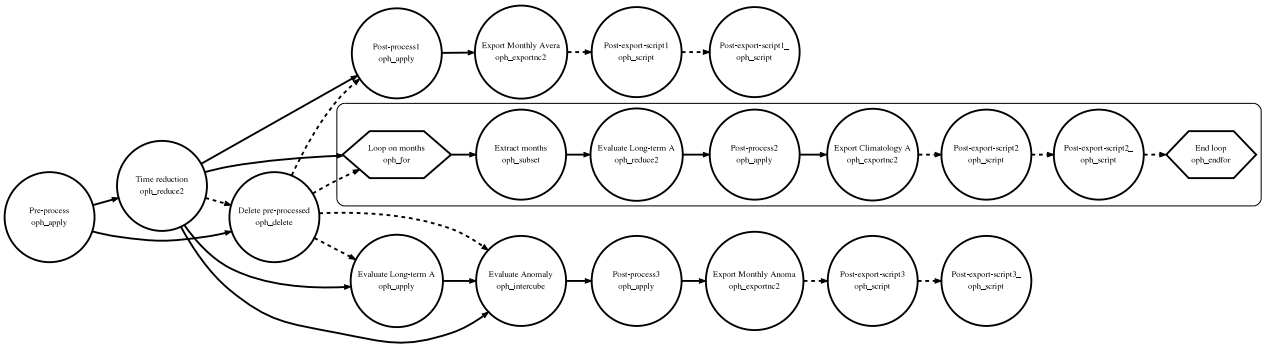


Figure 14. SST visual representation of the Ophidia workflow

In total this workflow performs 87 tasks producing 14 NetCDF output files (12*50MB + 2*12GB, 24.6 GB totally).

As in the previous case, the next images show a snapshot extracted during the execution and a map of the SST mean related to the 2010 (using Adaguc).

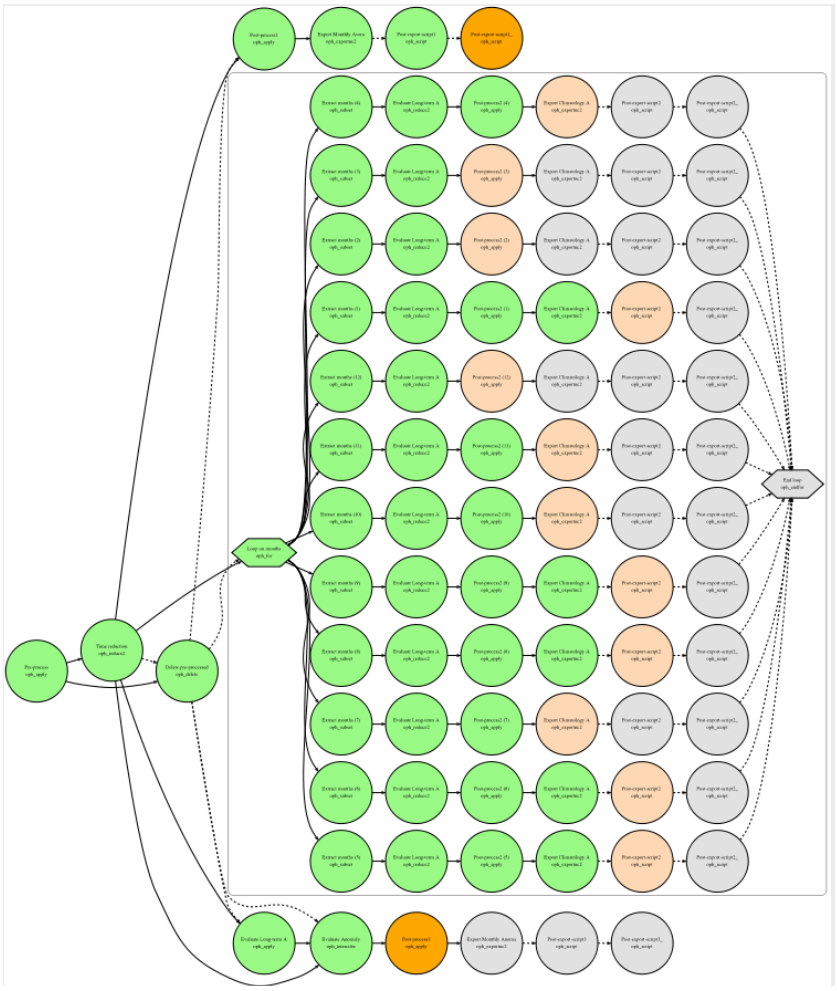


Figure 15. SST workflow execution

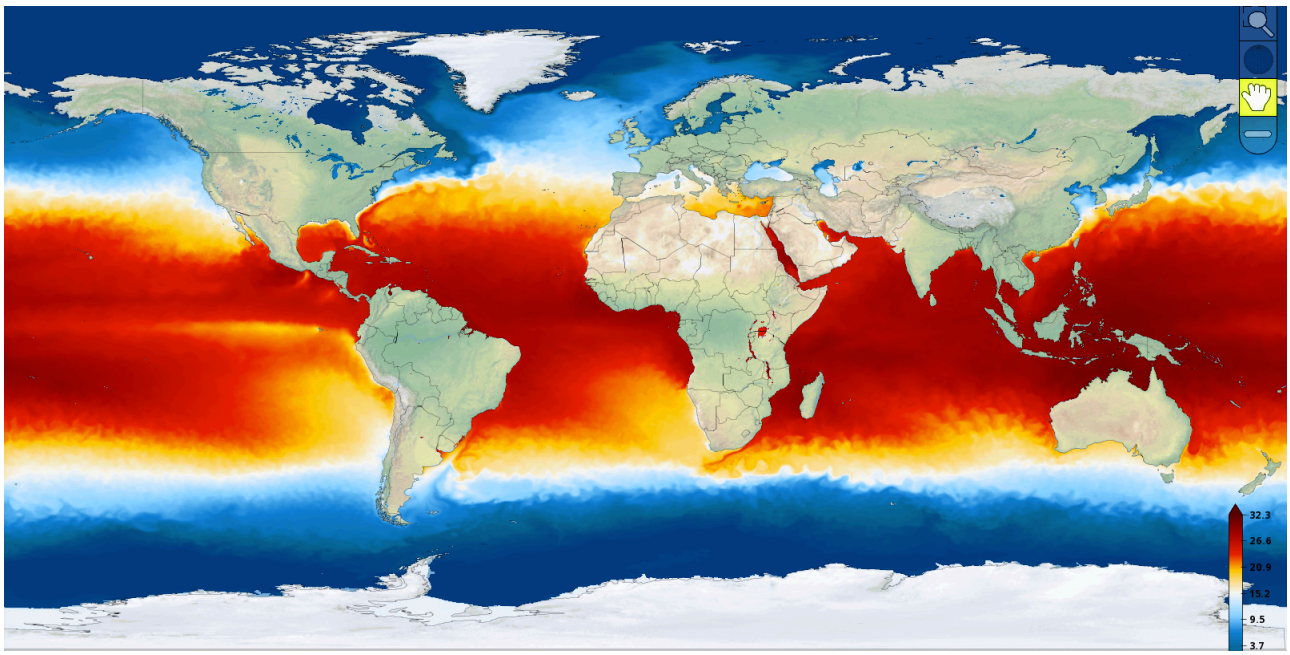


Figure 16. SST mean related to 2010

13. Conclusions

The WPS extension of the Ophidia Analytics Framework represents a useful interface developed on top of the Ophidia Server module able to accept remote job submissions following the OGC Standards for climate data processing. It is worth noting that no modification is required to the workflow JSON file related to the Ophidia submission: the same JSON schema could be used for both execution, WPS or WS-I based (for instance through the Ophidia terminal). Moreover, exposing a generic WPS interface to Ophidia processing services enables the application and re-use of Ophidia in future infrastructures.

Almost all the forementioned indicators (SWE related, SST based and the Snow-off), calculated exploiting the Ophidia processing and analysis features by using a parallel approach, have been uploaded on the CLIPC infrastructure and are available on the CLIPC portal for the users download and analysis.

14. References

[1] S. Fiore, A. D'Anca, D. Elia, C. Palazzo, I. Foster, D. Williams, G. Aloisio, "Ophidia: A Full Software Stack for Scientific Data Analytics", proc. of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014), July 21 – 25, 2014, Bologna, Italy, pp. 343-350, ISBN: 978-1-4799-5311-0

[2] S. Fiore, C. Palazzo, A. D'Anca, I. T. Foster, D. N. Williams, G. Aloisio, "A big data analytics framework for scientific data management", IEEE BigData Conference 2013: 1-8

[3] C. Palazzo, A. Mariello, S. Fiore, A. D'Anca, D. Elia, D. N. Williams, G. Aloisio, "A Workflow-Enabled Big Data Analytics Software Stack for eScience", The Second International Symposium on Big Data Principles, Architectures & Applications (BDAA 2015), HPCS 2015, Amsterdam, The Netherlands, July 20-24, 2015, pp. 545-552

[4] http://www.opengeospatial.org/standards/wps

[5] http://www.mysql.com/

[6] http://pywps.org/

[7] http://adaguc.knmi.nl/