

Final Report

BARRACUDA

November 30, 2017

Authors:

Sandro Fiore (CMCC), Fabrizio Antonio (CMCC)
sandro.fiore@cmcc.it, fabrizio.antonio@cmcc.it

External advisor:

Tobias Weigel (DKRZ, Senior RDA expert)
weigel@dkrz.de

Executive Summary

In complex data domains, unique and persistent identifiers (PIDs) are at the core of proper data management and access. Yet, it is useful to know the metadata elements in advance to set the services for data processing. Such metadata is called PID Information Type (PIT).

The working group on Persistent Identifier Information Types of the Research Data Alliance focused on the essential types of information associated with persistent identifiers by developing a conceptual model for structuring typed information.

We present here the design of the RDA-PIT support into Ophidia, a big data analytics research effort aimed at providing support for the access, analysis and mining of scientific (n-dimensional array based) data. The document highlights the most relevant implementation aspects related to the integration of the PID-resolving interface into Ophidia. Moreover, the report provides an application of the multi-model experiment for the precipitation trend analysis implemented in the EU INDIGO-DataCloud project.

Objectives

BARRACUDA (pid-BAseD woRkflow foR climAte Change Using ophiDiA) is a collaboration project about a case study on Climate models intercomparison data analysis led by the CMCC Foundation in the context of the EU H2020 INDIGO-Data Cloud Project¹ and mainly related to the climate change community organized within the European Network for Earth System²modelling.

The case study is directly connected to the Coupled Model Intercomparison Project³, one of the most internationally relevant and largest climate

¹<https://www.indigo-datacloud.eu/>

²ENES - <https://verc.enes.org/community/about-enes>

³CMIP - <http://cmip-pcmdi.llnl.gov/cmip5/>

experiments as well as to the Earth System Grid Federation⁴ infrastructure in terms of existing eco-system/service.

The proposed case study deals with relevant multi-model experiments like the trend analysis and itaimsat, providing a reference implementation for the coming CMIP6-based multi-model intercomparison data analysis experiments.

The project will include the integration of a PID handle service for the PIDs management, in order to bring this multi-model climate analytics experiment case study one step forward, by adopting the RDA recommendation on the PID Information Types⁵ (PIT) framework into the data lifecycle of the climate experiment proposed in this application.

The Ophidia big data analytics framework⁶, a high-performance array-database engine exploited in the INDIGO-Data Cloud project for climate change data analysis, will be extended to include the RDA-PIT support in order to:

- uniquely identify a climate data resource managed in the analysis/processing system through PIDs;
- make the whole system provenance-aware by adding additional metadata elements to the existing PIDs and assign new ones to the output of a data analysis experiment run by Ophidia.

The proposed PID-based extensions will be tested using a PID Handle Service instance kindly provided by DKRZ⁷.

Initial State

The ESGF infrastructure currently offers basic PID management, but it does not cover full provenance tracing and data processing. In particular, the process of re-publishing analysis output data with PIDs is a step not yet implemented in production-level environments like ESGF, but of great relevance for the involved community especially now that server-side processing components are going to be added to the ESGF software stack.

In general, the project outcomes will reinforce the multi-model climate analytics experiment case study by (i) making new data products interoperable, (ii) enabling data provenance and experiments reproducibility at large scale and (iii) following a more complete and interoperable workflow lifecycle for data publication in close cooperation with the ESGF eco-system/services.

RDA PIT Specification

The working group on Persistent Identifier Information Types of the Research Data Alliance focused on the essential types of information associated with persistent identifiers by developing a conceptual model for PID record properties, aggregated types and profiles, useful to enable typed information structure design, as well as an application programming interface to access typed

⁴ESGF - <http://esgf.llnl.gov>

⁵RDA PID Information Type WG -

<https://b2share.eudat.eu/records/a6b1b95fe7ac4ed98e882d6bc7c7e775>

⁶www.ophidia.cmcc.it

⁷DKRZ - <https://www.dkrz.de/>

information and a demonstrator to implement the interface. The WG has managed to elucidate the basic terminology concerning PID Information Types and the necessary interaction among technical components that allow typing.

The unique and persistent identifiers (PIDs) prove essential to correctly manage and access data / for proper data management and access in complex data contexts. Whereas more basic identifier services rely on the association between the identifier and a data object providing an identity, more complex ID services allow to capture this relationship, thus exposing it directly through the service itself as identifier metadata. In this way, information retrieval is standardized and the complexity reduced. The repository and catalog load is consequently smaller, because it's possible to access the identifier metadata without resorting to the resource itself.

Identifier metadata enable discovery, access, integrity and authenticity verification services and a variety of other use cases. To do so, the provided service should be able to establish whether the types of information required are available in the PID metadata.

The RDA PID Information Types Working Group was set up to identify a framework for PID information types, as well as winning approval for some basic information types and defining the integration process concerning other types of information.

The goal of the PIT WG activities was to harmonize the basic information types associated with persistent identifiers. In order to achieve this, the necessary tools and concepts had to be developed first. The main outcome in conceptual terms was a framework for information type definition whereas the key outcome in practical terms was the design of an API and a prototypical implementation revealing core types usage. The term "PID Information Type" encompasses more precise notions of identifier metadata elements and their aggregates. PID records actually contain a small subset of digital object metadata, also known as the *persistent state information*, that is useful to retrieve at least some state information from the repositories, which is exposed in a simple format through the generic PID Information Types API.

The WG came to the conclusion that the acceptance of a common set of core types relies on practice. It actually takes time and the work of a small group doesn't suffice. Practice does not assure stability though, therefore some governance would definitely be necessary. What turns out to be a key element is the type registry.

The absence of a fixed set of accepted core types didn't prevent the WG to provide insight concerning information types definition and their potential use. The structure of registered properties and their aggregates is a key element as well as the functionality of the type registry. The main conceptual outcome concerns the framework for structuring types and their availability by means of the type registry.

Conceptual model

The service providing the PIT API has been conceived with a modular design to provide different infrastructures with different adapters, since existing PID infrastructures have different capabilities and designs. The users of the PIT API might be end-user services at times, yet it's usually the case of other infrastructure components. The adoption of the PIT API by PID infrastructures

and its integration into their services brings about a convergence of the two lowest architecture layers.

Metadata represent an important and widely used cross-disciplinary solution. PID records are actually a kind of metadata, yet smaller and unable to replace established metadata systems.

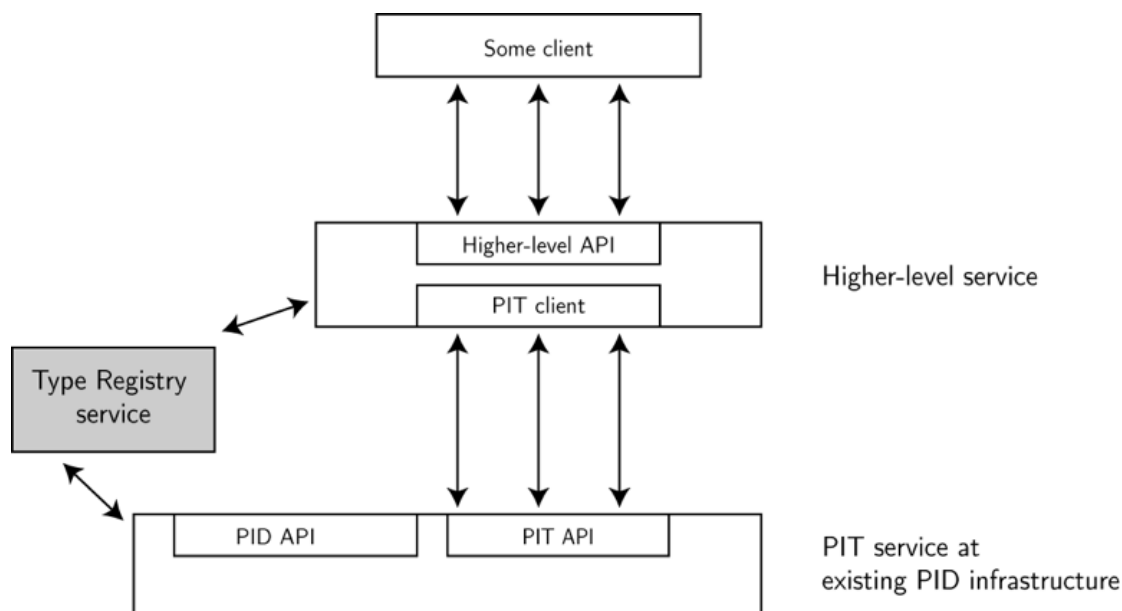


Fig. 1: The high-level architecture for the PID Information Types API prototype is composed of several layers, and the PIT API lies between existing PID infrastructures and their native PID APIs and some consumer clients. The Type Registry forms a core dependency situated aside from the layer structure.

Data model

The core entities have been split into two main models that illustrate their usage and naming specifications.

The property-type-profile model

This model was used in the prototype and every PID record is made up of a number of properties. Every property has its PID and the key elements are the name, range and values. The PID record only stores the PID and the values, whereas the name and range are gathered in the type registry. The latter is where every property is registered, and besides the property range and the name, the type registry provides additional information, such as a description text and source information (author, creation date, contact details).

A type is made up of a number of both mandatory and optional properties. Every type is registered in the type registry, along with the PID, description text and provenance information. A PID record matches a type if all mandatory properties of that type are provided, whereas filtering a PID record by type will show all mandatory and optional properties.

A profile consists of several types. A PID record matches a profile if all mandatory properties of all profile types are provided. There are two profile models: in the first one, a profile is not necessarily identifiable and carries no global PID, whereas the second model provides for the registration of the profile in the type registry, thus increasing interoperability.

The reason behind profile inclusion is the attempt to prevent types from proliferating, specifically when a community intends to use a predefined type just by adding or excluding an additional property.

The type-profile model

This is a simpler model that allows to type the elements of a PID record and register those types in the type registry. Every type bears a PID and consists of a name, range and value, like the property of the property-type-profile model. Similarly, types are aggregated into profiles, which can refer to optional and mandatory types. It is possible to check if a PID record matches a profile, as in the case of the property-type-profile model. There is no equivalent entity in the type-profile model for the profiles of the property-type-profile model.

Model selection

The property-type-profile model may generally be closer to an understanding of PID records from a domain metadata perspective. Therefore, the type-profile model is simpler and should be preferred over the property-type-profile.

In the full property-type-profile model, types aggregate properties and profiles aggregate types. Services will typically work with types, by requiring a specific PID record to provide information conforming to a particular type. Profiles may or may not be used. Therefore, the type-profile can be expressed as a particular subset of the full model.

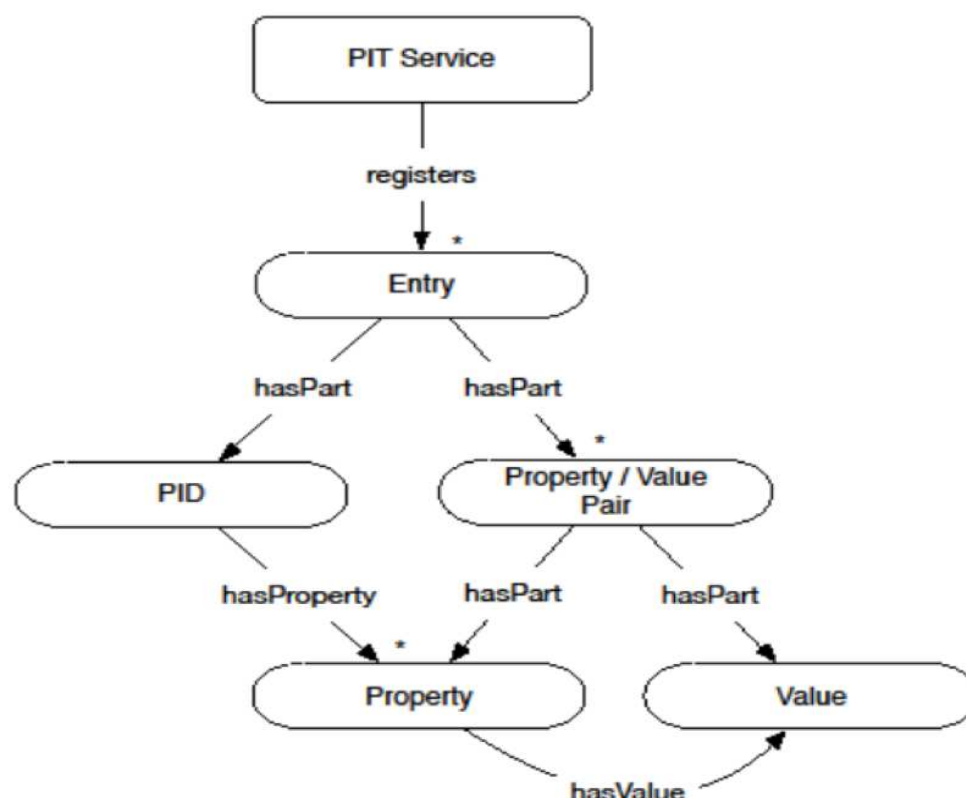


Fig. 2: PIT data model.

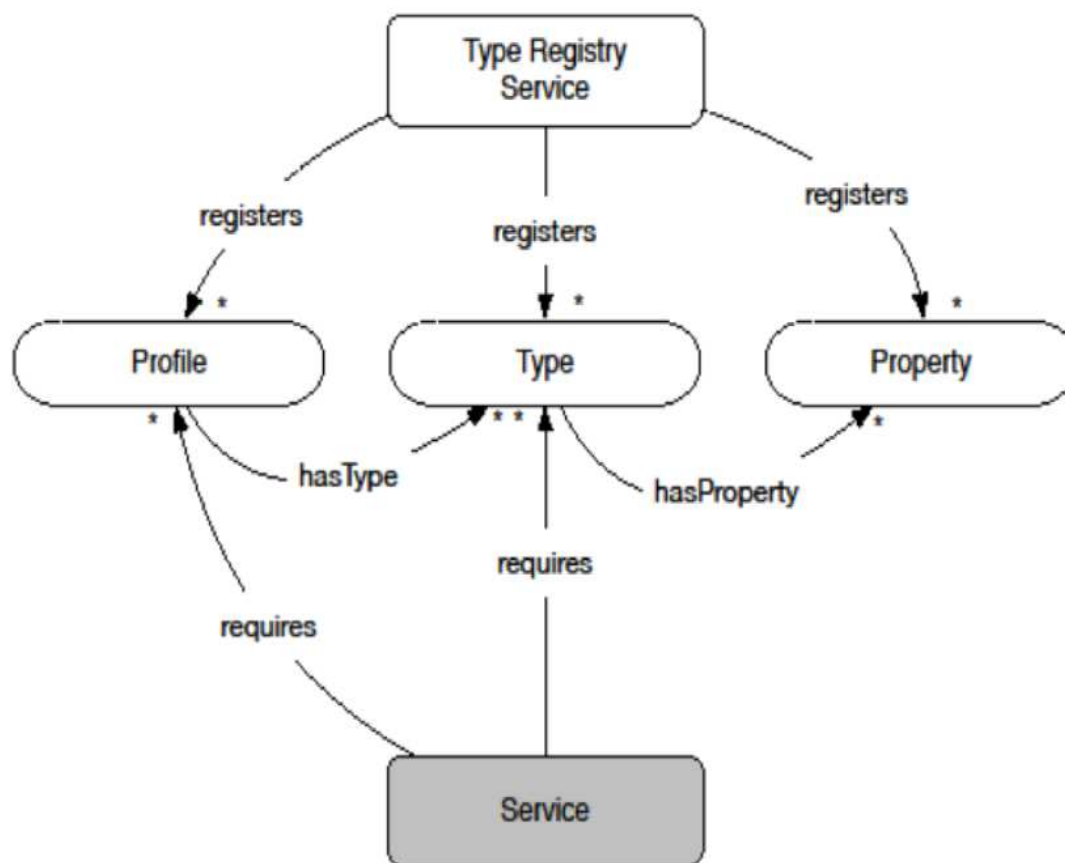


Fig. 3: Type Registry data model. A service may require types and/or profiles; none of these are mandatory.

API overview

The RDA PIT API provides facilities to query and manipulate typed information available from persistent identifiers and shared through type registries.

The API has been designed in order for it to be agnostic towards the underlying identifier system, provided that it is possible to store and retrieve additional information associated with a single identifier.

Ophidia and Terminal

Ophidia [1] is a research effort aimed at addressing big data challenges for eScience. It represents a framework providing a complete environment for efficient scientific data analysis and management of multi-dimensional heterogeneous data sets.

The system provides an array-based storage model and a hierarchical storage organization to partition and distribute the multidimensional scientific data sets across multiple compute nodes. It implements the data cube abstraction from OLAP systems and leverages parallel computing, data distribution, jointly with a native in-memory engine to perform parallel I/O operations. A complete description of the Ophidia Framework architecture, its main components, along with the design choices is provided in background works [2][3].

Currently, the Ophidia framework provides more than 50 parallel (MPI-based) and sequential operators to perform (i) data cube-oriented operations like data import/export, subsetting and reduction, and (ii) metadata management, data

cube provenance and file system management. Additionally, the framework implements a wide set of primitives to perform operations on multi-dimensional binary arrays, which include, among the others, arithmetical/mathematical functions, time-series aggregation, statistical computations, linear regression and interpolation. Complex workflows, including more operators and primitives, can be executed by submitting a JSON description of the list of tasks and related dependencies.

Ophidia supports a number of different communications and security protocols, so that it has several front-end interfaces:

- REST: it provides a REST architecture based on HTTPS protocol;
- WS-I: it defines a web service based on SOAP over HTTPS;
- OGC: it is a Web Processing Service (WPS);
- GSI/VOMS: it represents a GSI-enabled interface with support for Virtual Organisations.

Regardless of the interface adopted to access Ophidia, the user can access, process, view datasets (without necessarily downloading them), execute operators and primitives, share data with other users and transfer them from/to other repositories. In this way, the productivity of scientists and researchers is greatly improved.

Ophidia includes a rich shell-like command-line tool, named Ophidia Terminal, and a more programmatic interface, named PyOphidia, that can be adopted by users to submit commands to Ophidia service.

Ophidia Terminal, designed to run over a classic desktop computer as well as a node of a cluster, is similar to the “bash” program present in almost all Unix-like environments and includes several useful features such as command history management, auto-completion of operators, arguments and admitted values, the management of specific environment variables and user-defined aliases, a contextual manual with the description of commands and variables, the management of key combinations for the smart-editing of the command line, browsing of local and remote file system, rendering of responses and more. Moreover, the tool deals with the authentication and authorization protocol (e.g. it automatically appends user credentials or access tokens to commands to be submitted), facilitating the use of Ophidia interfaces.

In general, an interactive session with the Ophidia Terminal could be described by the use case shown in Figure 4.

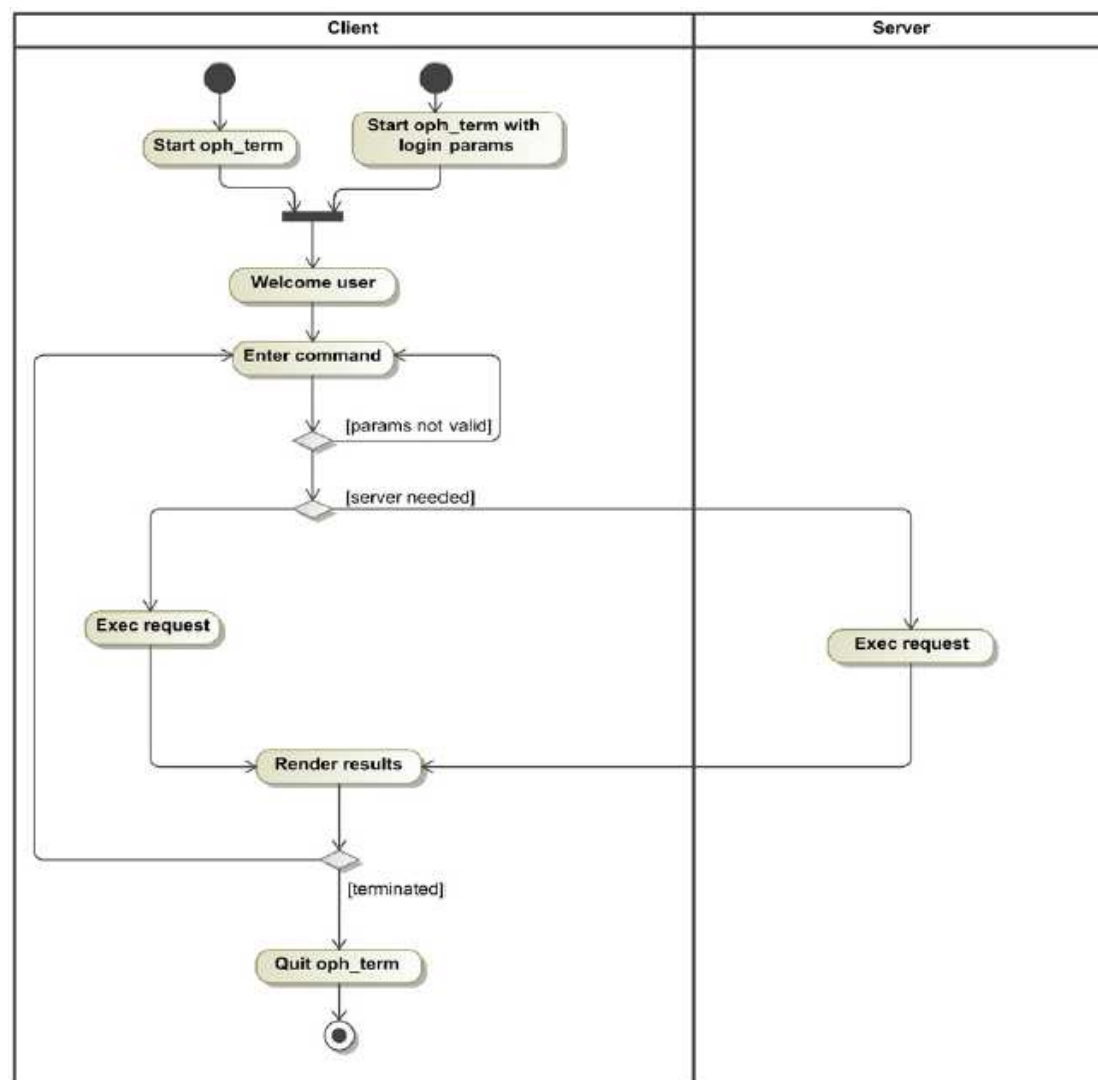


Fig. 4: Flow diagram of Ophidia Terminal.

It is possible to start the terminal simply by running the executable “oph_term” with no special options, quickly accessing to the working environment, or setting some parameters, at command line (e.g. user credentials) as well as by means of some environmental variables. Then, the user is able to submit commands.

Unless a command is referred to a local action (e.g. an update of stored user credentials), for each command the Ophidia Terminal formats the request to be sent to Ophidia service according to the selected specific interface, then sends the message and waits for a reply. Upon receipt, the application formats the output to be shown to the user (by parsing the response) in case of success or it displays a notice in case of errors.

To execute a generic operator, the user needs to submit a command having the following syntax.

```
oph_operator key1=value1;key2=value2;...;keyN=valueN;
```

The command name is equal to the operator name, whereas the key-value pairs, containing the equal sign ‘=’ and separated by semicolons, correspond to

operator arguments and related values. Keys must be single words and values cannot clearly contain semicolons.

After sending the request, the behavior of the terminal depends on the execution mode:

- in case of asynchronous mode, the tool displays the “job identifier” (JobID) and shows the prompt for the next to command without waiting for job completion;
- in case of synchronous mode, the tool waits for the job to be completed before displaying the identifier and the output.

In both cases, the Ophidia Terminal extracts the “working session identifier” from the JobID and automatically sets the environmental variable `OPH_SESSION_ID` to that value. The role of this identifier, required as argument by most Ophidia operators, is to group the commands related to a scientific experiment and the corresponding data cubes. In case these data are shared among more users, the owner simply has to grant others access to the session and send them the session identifier. In this way, another user can set the variable `OPH_SESSION_ID` to the shared session identifier and access data using the terminal.

The Ophidia Terminal allows the user to submit single commands and workflows interactively as well as in batch mode, likewise a bash command.

In Ophidia, each response is a JSON file that may include objects of different types: simple text; grids and tables; trees; graphs. The terminal is able to recognize this type of objects and represent them in the most appropriate manner on the screen: textually as well as graphically. The user can choose several visualization options: displaying the objects in a compact and tabular format (basic mode) or printing out additional metadata (extended mode), showing the results within a GUI or saving them as image on disk, by using the colour setting of the hosting shell or user-defined colours, etc.. In addition, a response can simply be displayed as is: this option is very useful when the output has to be processed by an external tool.

Concerning the use of a GUI to display responses (only for desktop environment), it is interesting to note how the terminal handles the output of an Ophidia operator named `OPH_CUBEIO`. For a given cube, this operator returns a report with all the transformations performed on an input dataset to produce that cube (provenance) and some details about the cubes created at each step and the related operators. In most cases, the report is a linear sequence of operations starting, for example, from the insertion of a new dataset into the system (import), passing through a series of operations of data subsetting and reduction, ending with the publication of the files containing the analyzed data (export). In other cases, however, the operations can create strongly interlinked graphs. In these situations, the ability to have a graphical view of the entire workflow is undoubtedly a better solution. The Ophidia Terminal gives its users this possibility. Figure 5 shows a sample image generated by processing the response of `OPH_CUBEIO`. In this case, two datasets are imported into Ophidia and compared: some preliminary operations (subsetting, reduction, evaluation of a primitive) are applied on input datasets before the intercomparison (`OPH_INTERCUBE`) and, finally, the result is processed by a

primitive(OPH_APPLY). The nodes of this graph are related to the cubes generated at each step and identified by an URL, whereas the edges refer to the operations. The graph also reports the references to source files.

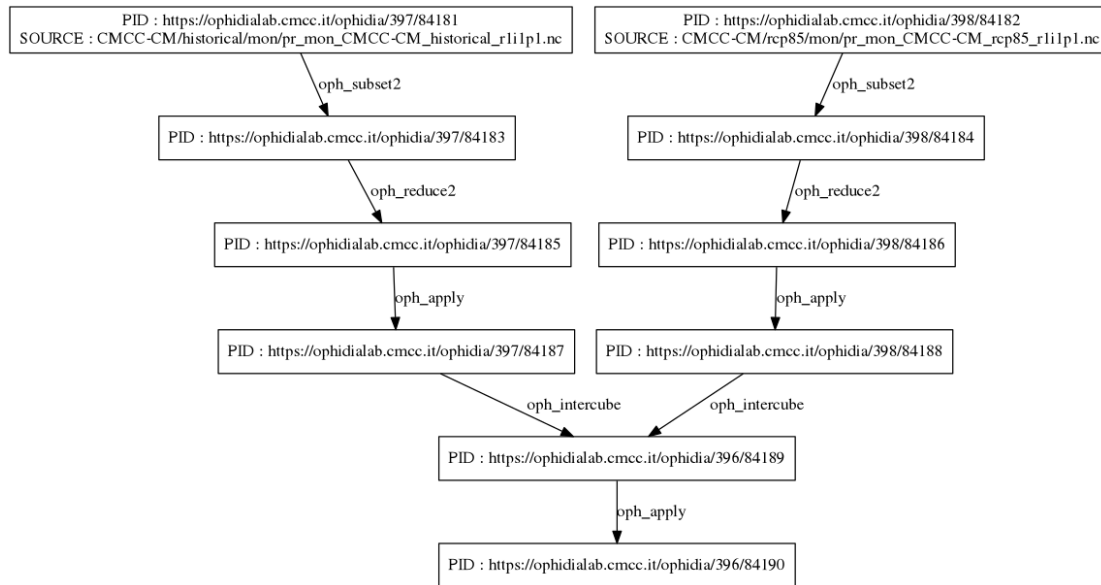


Fig. 5 : Example of response associated with OPH_CUBEIO.

The Ophidia Terminal also supports other interesting features. For instance, it can be used to validate the JSON description on a workflow and to show its task graph before starting the execution. The user simply has to submit the command “check” setting the name of the JSON file (e.g. workflow.json) as an argument:

```
check ./workflow.json arg1 arg2 ... argN
```

During workflow execution , the user is also able to graphically see its progress. The command “view” has been designed to this aim:

```
view WORKFLOW_ID
```

In this example, WORKFLOW_ID is an identifier set by Ophidia service when the request for a workflow execution is accepted. The identifier is sent back to the submitter so that the user can refer to that workflow in next requests.

INDIGO and ENES multi-model experiment on Precipitation Trend Analysis

INDIGO project overview

INDIGO-DataCloud (INtegrating Distributed data Infrastructures for Global ExpLOitation) [4-5] is a project funded under the Horizon2020 framework program of the European Union, led by the National Institute for Nuclear Physics (INFN). The project main objective is the development of a data and computing platform targeting scientific communities and deployable on multiple hardware, to be provisioned over private or public e-infrastructures.

In the context of Cloud computing, resources as IaaS (Infrastructure as a Service) are already offered by the public and private sectors, but there is still a lack at PaaS (Platform as a Service) and SaaS (Software as a Service) levels.

The INDIGO-DataCloud project aims to fill these gaps by overcoming current challenges in the Cloud computing, storage and network areas, such as, for example: i) orchestrating and federating Cloud, Grid and HPC (public or private) resources ii) overcoming performance issues that limit the massive adoption of virtualized Cloud resources in large data centers; iii) managing dynamic and complex workflows for scientific data analysis.

The project extended existing PaaS solutions, allowing public and private e-infrastructures to integrate their existing services and make them available through AAI services compliant with GEANT's interfederation policies, thus guaranteeing transparency and trust in the provisioning of such services. It allows the execution of applications on Cloud and Grid based infrastructures, as well as on HPC clusters.

INDIGO also provided a flexible and modular presentation layer connected to the PaaS and SaaS frameworks developed within the project, allowing innovative user experiences and dynamic workflows, also from mobile appliances.

Precipitation Trend Analysis workflow

Precipitation trend analysis has received notable attention during the past century due to its connection with global climate change, as stated by the scientific community. For this reason, a number of models for this atmospheric variable have been defined.

To better understand the model accuracy of phenomena, the results obtained by each model have to be compared against historical data first, to identify possible anomalies. Then, the anomalies have to be compared among the models (ensemble analysis) to score them and, hence, evaluate the related phenomena.

Figure 6 shows the workflow (next referred to as the “experiment”) to analyze the precipitation trend over a given spatial domain by comparing the anomalies related to a number of models in the context of CMIP5 Federated Archive.

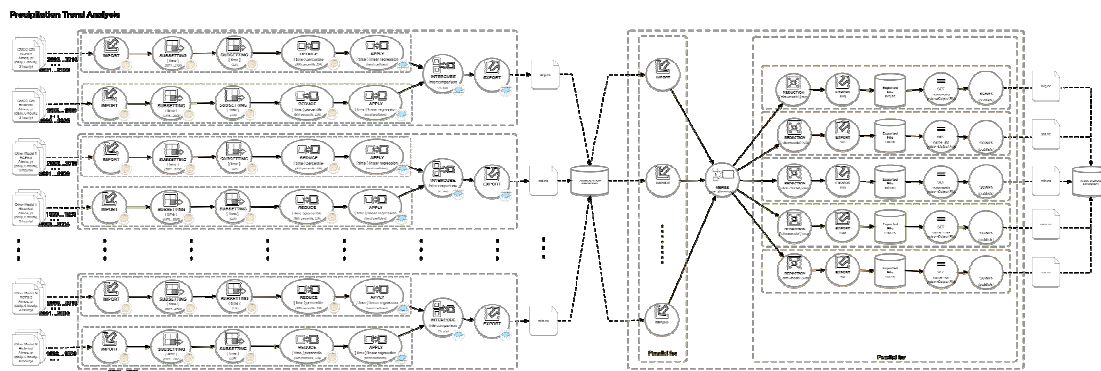


Fig. 6: Experiment for precipitation trend analysis

The experiment consists of a number of sub-workflows, which can be executed in parallel, followed by a final workflow performing an ensemble analysis.

Each sub-workflow is associated with a specific climate model involved in the CMIP5 experiment. A scenario must be also defined as input.

The sub-workflow is aimed at performing the following tasks:

- discovery of the two datasets (historical and future scenario data);
- evaluation of the precipitation trend separately for both datasets;
- comparison of the trends over the considered spatial domain;
- 2D map generation.

The ensemble analysis includes the following three steps:

- data gathering;
- data re-gridding;
- accuracy evaluation.

In Figure 6, the sub-workflows are shown on the left within cyan rectangles. The tasks related to the historical data process are in green rectangles, whereas the tasks that process data resulting from the model are in red rectangles. The tasks of the ensemble analysis are on the right within yellow rectangles. It is composed by a first parallel import of the previous outputs, a central circle for the merge operation and the final parallel data re-gridding and accuracy evaluation.

Note that the time domain related to historical data is fixed. For instance, the range 1976-2005 is adopted for the experiment. The time domain related to models shall have the same duration (e.g. 30 years) though it clearly refers to a future time range like 2071-2100.

Table 1 provides a list of the models currently supported by the precipitation trend analysis experiment, with the related time frequencies, scenarios, grid resolution and institute. More info can be found in [6]

Model name	Time frequencies	Scenarios	Lat x Lon (°)	Institute (institute ID)
CCSM4	daily - monthly	RCP 4.5 - RCP 8.5	0.95 x 1.25	National Center for Atmospheric Research (NCAR)
CMCC-CM	daily - monthly	RCP 4.5 - RCP 8.5	0.8 x 0.8	Centro Euro-Mediterraneo sui Cambiamenti Climatici (CMCC)
CMCC-CMS	daily - monthly	RCP 4.5 - RCP 8.5	1.9 x 1.9	Centro Euro-Mediterraneo sui Cambiamenti Climatici (CMCC)
CNRM-CM5	daily - monthly	RCP 4.5 - RCP 8.5	1.4 x 1.4	Centre National de Recherches Météorologiques / Centre Européen de Recherche et Formation Avancées en Calcul Scientifique (CNRM-CERFACS)

CanESM2	daily - monthly	RCP 4.5 - RCP 8.5	2.8 x 2.8	Canadian Centre for Climate Modelling and Analysis (CCCMA)
INM-CM4	daily - monthly	RCP 4.5 - RCP 8.5	1.5 x 2.0	Institute for Numerical Mathematics (INM)
IPSL-CM5A-MR	daily - monthly	RCP 4.5 - RCP 8.5	1.25 x 2.5	IPSL-CM5A-LR Institut Pierre-Simon Laplace (IPSL)
MIROC5	daily - monthly	RCP 4.5 - RCP 8.5	1.4 x 1.4	Atmosphere and Ocean Research Institute (The University of Tokyo), National Institute for Environmental Studies, and Japan Agency for Marine-Earth Science and Technology (MIROC)
MPI-ESM-MR	daily - monthly	RCP 4.5 - RCP 8.5	1.9 x 1.9	Max Planck Institute for Meteorology (MPIM)
MRI-CGCM3	daily - monthly	RCP 4.5 - RCP 8.5	1.1 x 1.1	Meteorological Research Institute (MRI)
NorESM1-M	daily - monthly	RCP 4.5 - RCP 8.5	1.9 x 2.5	Norwegian Climate Centre (NCC)

Table 1: List of the models included into the PTA

Project Outcomes

Module 1: Extended scenario

The Ophidia terminal is a robust, comprehensive, effective and extremely usable command line client that allows users to submit Ophidia commands to the Ophidia server.

For each user request, the terminal sends a new message with the appropriate connection parameters and the string corresponding to the request for execution of a particular operator, which is then interpreted by the functions of the Ophidia framework.

Once the operator execution terminates, the Oph-Terminal receives the corresponding response message with the operator output data, which are then interpreted and appropriately displayed to the user.

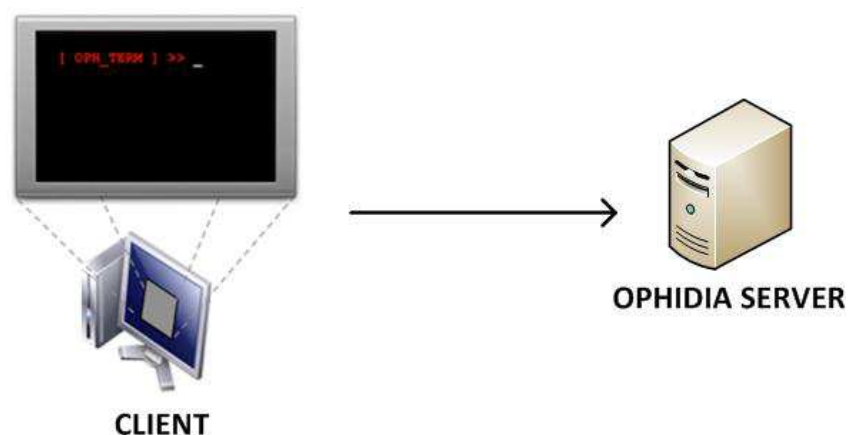


Fig. 7 - General Client-Server interaction

This simple scenario has been extended to integrate the RDA-PIT support into the Ophidia big data framework. Fig. 8 shows the main components and interactions relating to the new PID-based workflow for climate change using Ophidia.

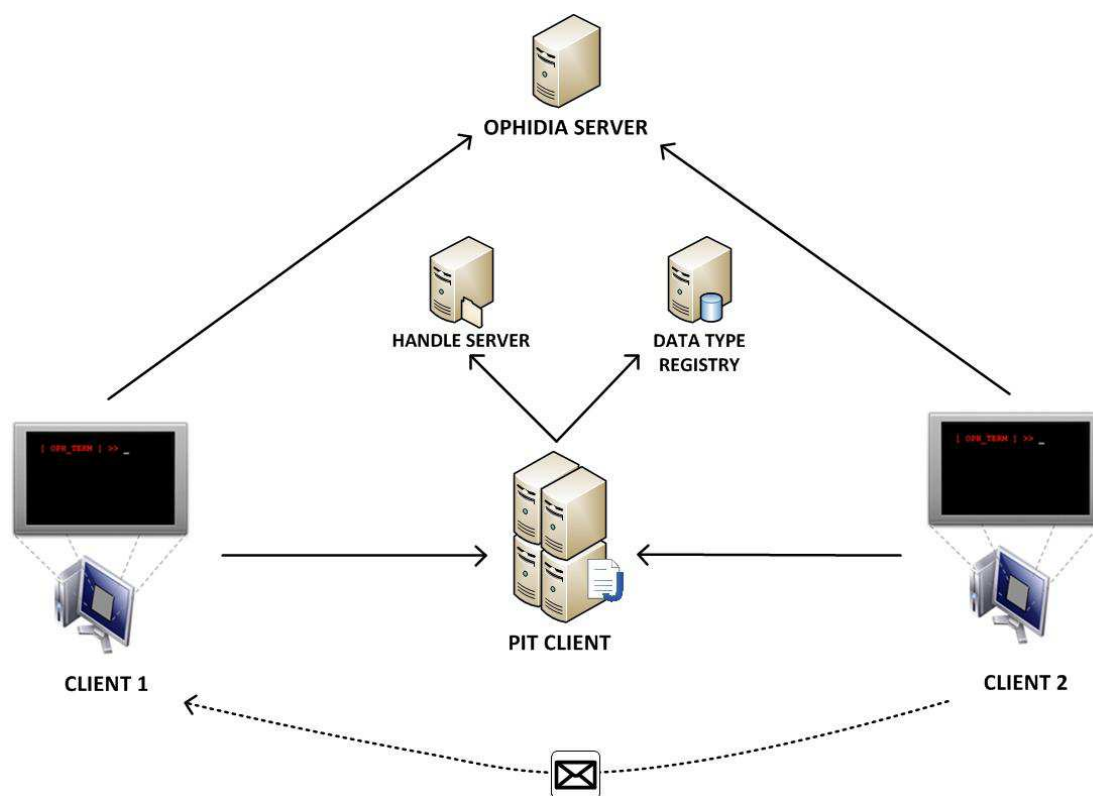


Fig. 8 - Extended Client-Server scenario

The Handle Server stores information resources in the form of persistent identifiers; the Data Type Registry provides a special class of additional metadata, named PID Information Types, used to collect information about the data and that it is usually necessary to know before the data is processed. The interaction with information types and typed PID records happens via a Web Service API (PIT Service). The RDA PID Information Types WG has implemented

a prototypical servlet implementation, which needs to be deployed in a Java application server (e.g. Apache Tomcat) at client side. The prototype relies on the type registry prototype provided by the Data Type Registries WG and a preliminary Handle Server v8 installation provided by the CNRI. The REST Service must be properly configured, for it to know which identifier service and type registry to contact.

Clients might need to communicate with each other: the owner of a session can finely grant access privileges to other users, thus sharing the experiment.

UML Diagrams

The main diagrams that support the design of the Ophidia extension for RDA-PIT are illustrated as follows.

Use Cases diagram

In this section, a Use Cases-based requirements analysis is presented.

It is possible to identify two types of user actor:

- a producer user, who, after running an experiment (for example, by executing a workflow), stores the related data and metadata information in the Handle System;
- a consumer user, who i) searches for handles by some criteria, ii) looks-up a handle to retrieve all PID Information Types values, and iii) performs several operations and analysis using the extracted information.

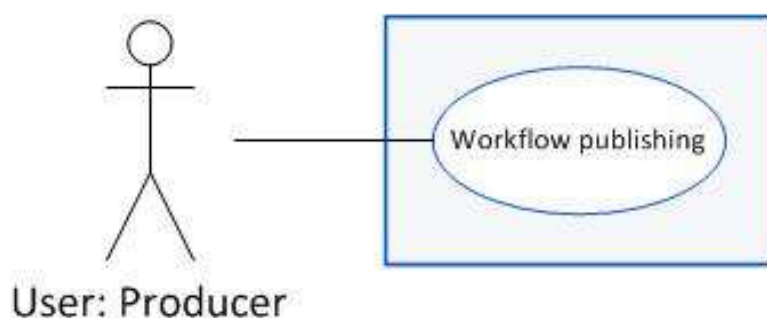


Fig. 9 - Use Cases diagram for the PRODUCER actor

Workflow publishing

As a PRODUCER user, I want to publish a workflow on the PID Handle Service so that I can share the experiment with other users.

Primary actors:

- User
- System (Ophidia Terminal)
- PIT Service
- Handle Service

Triggers:

- The user runs the insert command

Preconditions

- The user is logged in to the System and has executed a workflow

Main Flow

1. The User runs the insert command providing all the required arguments in the form of key-value pairs.
2. The System validates the input arguments.
3. The System sends a POST request to the PIT Service.
4. The PIT Service sends a PUT request to the Handle Service.
5. The Handle Service creates the new handle record.
6. The System notifies that the creation has been successfully performed, showing the PID of the newly created handle record.

Postconditions

- A new handle record is created at Handle Server side, featuring a randomly and uniquely generated identifier,.

Extensions

- 3.a. Arguments verification fails (missing or too many arguments).

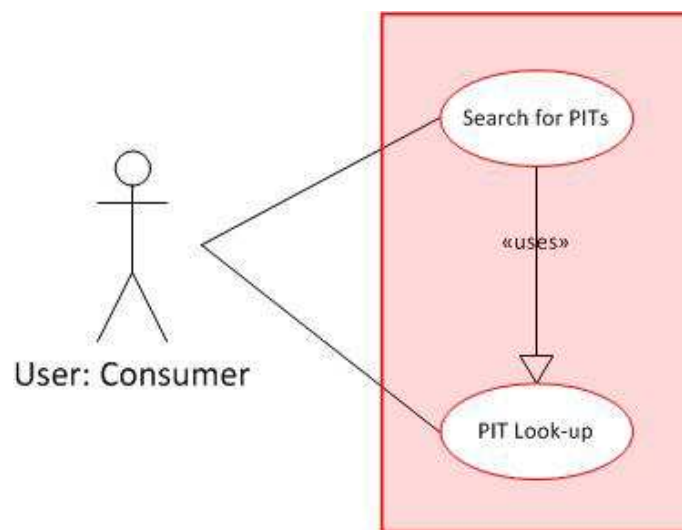


Fig. 10 - Use Cases diagram for the CONSUMER actor

PIT Look-up

As a CONSUMER user, I want to look-up a handle by PID in order to retrieve all the handle records and analyze them.

Primary actors:

- User
- System (Ophidia Terminal)
- PIT Service
- Handle Service
- DTR Service

Triggers:

- The user runs the look-up command

Preconditions

- The user is logged in to the System

Main Flow

1. The User runs the *look-up* command providing the desired PID.
2. The System validates the input arguments.

3. The System sends a GET request to the PIT Service.
4. The PIT Service sends a GET request to the Handle Service.
5. The Handle Service resolves the handle record and replies with an array of handle values.
6. For each handle value, the PIT Service sends a GET request to the DTR Service to resolve the type identifier.
7. The DTR Service replies with the type definition.
8. The PIT Service replaces the type identifier with the corresponding type name.
9. The PIT Service gives back to the System an enriched handle record.
10. The System shows all the handle values (type name-type value pairs) forming the desired handle record.

Extensions

- 2.a. Arguments verification fails (missing argument).
- 5.a. No handle found. Go to step 10.
- 10.a. The System returns a message indicating no handle has been found.

Search for PITs

As a CONSUMER user, I want to search for handle records by key-value pairs in order to retrieve a list of useful handles.

Primary actors:

- User
- System (Ophidia Terminal)
- PIT Service
- Handle Service
- DTR Service

Triggers:

- The user runs the search command

Preconditions

- The user is logged in to the System

Main Flow

1. The User runs the search command by providing your own search keys.
2. The System validates the input arguments.
3. The System sends a GET request to the PIT Service to list handles under the working prefix.
4. The PIT Service sends a GET request to the Handle Service.
5. The Handle Service replies with a list of all stored handles.
6. For each handle, the System performs a look-up operation to retrieve all the handle records, then it applies the user filter.
7. The System gives back only handles satisfying the user search keys.

Extensions

- 2.a. Arguments verification fails (missing or too many arguments).
- 7.a. No handle satisfies the search criteria.

Sequence diagram

The following diagrams show how the main objects involved in the use cases described above operate and the interaction among them.

Workflow publishing

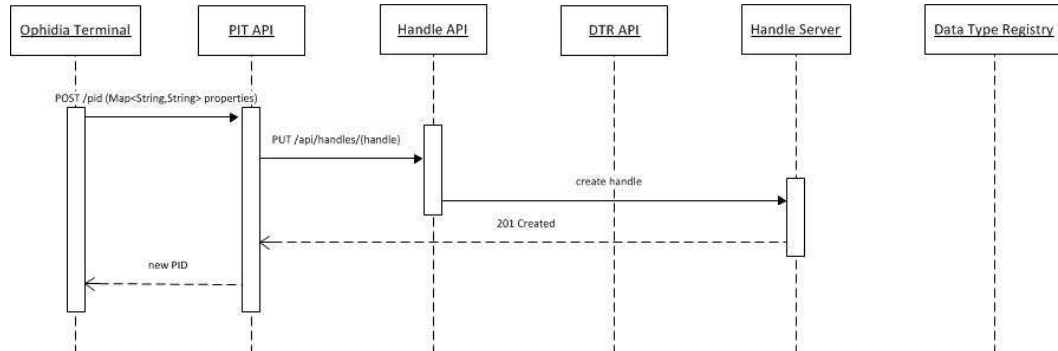


Fig. 11 - Sequence diagram for the "Workflow publishing" use case

The client, via the dedicated insert command, sends a POST request to the PIT Service. The key-value pairs, named *properties* and provided as input arguments by the user, are sent in the request body in JSON format. The PIT Service forwards the request to the Handle Server HTTP REST interface by means of a PUT request to enable storing the enclosed entity under the supplied Request-URI. If the operation is successful, a new handle record will be stored in the Handle Server and the PIT Service will notify the PID of the newly created handle to the client.

PIT Look-up

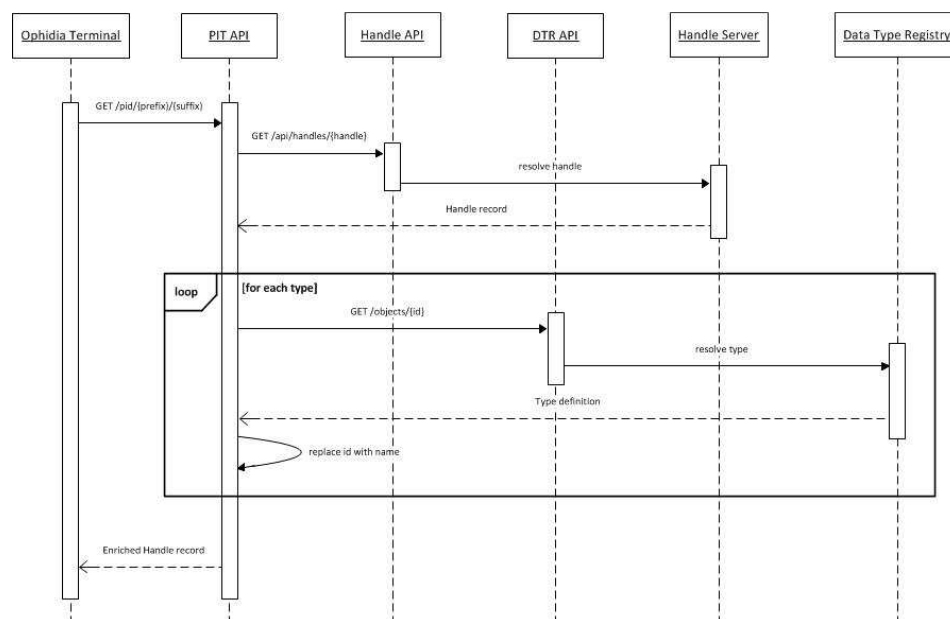


Fig. 12 - Sequence diagram for the "PIT Look-up" use case

The client, via the dedicated look-up command, sends a GET request to the PIT Service to resolve the specified identifier in the form of {prefix}/{suffix}. It also sets the *include_property_names* query parameter to *true* to retrieve the types name in addition to the types identifier and value. The PIT Service i) forwards the request to the Handle Server to retrieve the handle record, then, for each type forming the record, ii) it contacts the Data Type Registry to get the type definition and resolve the type identifier. The PIT Service replies to the client with an enriched handle record, containing the types identifiers, names and values.

Search for PITs

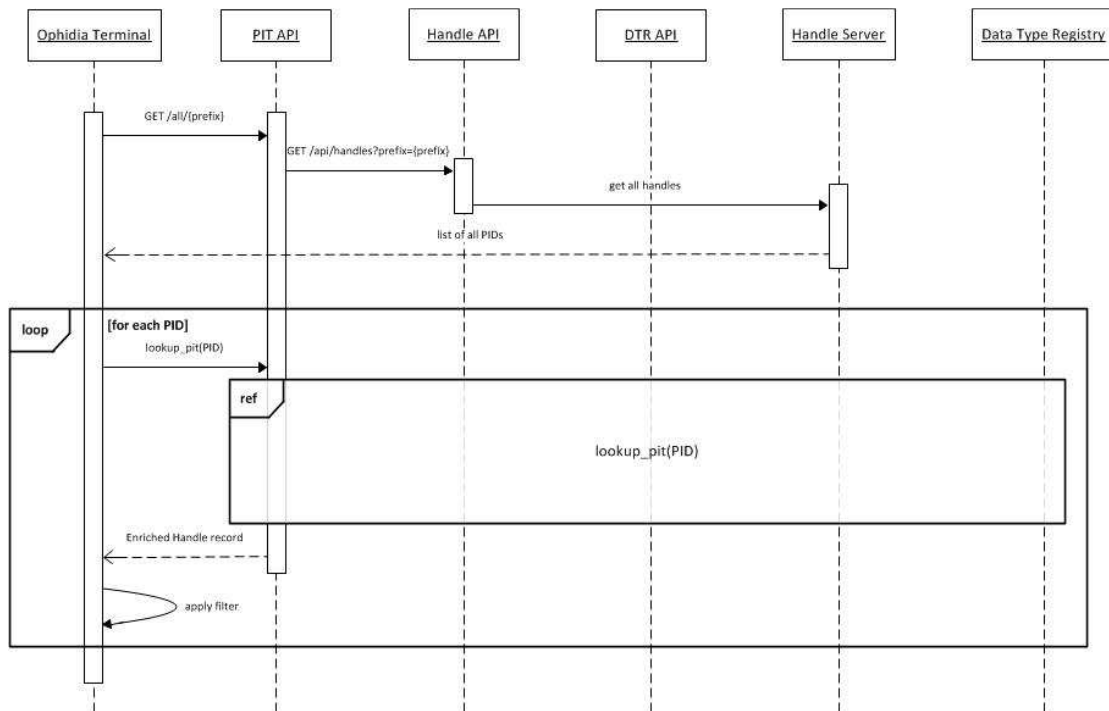


Fig. 13 - Sequence diagram for the "Search for PITs" use case

The client, via the dedicated search command, sends a GET request to the PIT Service to get all the handles stored under the specified prefix. The PIT Service forwards the request to the Handle Server and replies with a list of handles. For each of them, the client performs a look-up operation and checks whether the handle record values comply with the search criteria provided by the user.

Deployment diagram

Figure 14 shows the Deployment and Component diagram, which depicts the software components, how they are distributed over the hardware resources and the interactions among the hardware and software components forming the system.

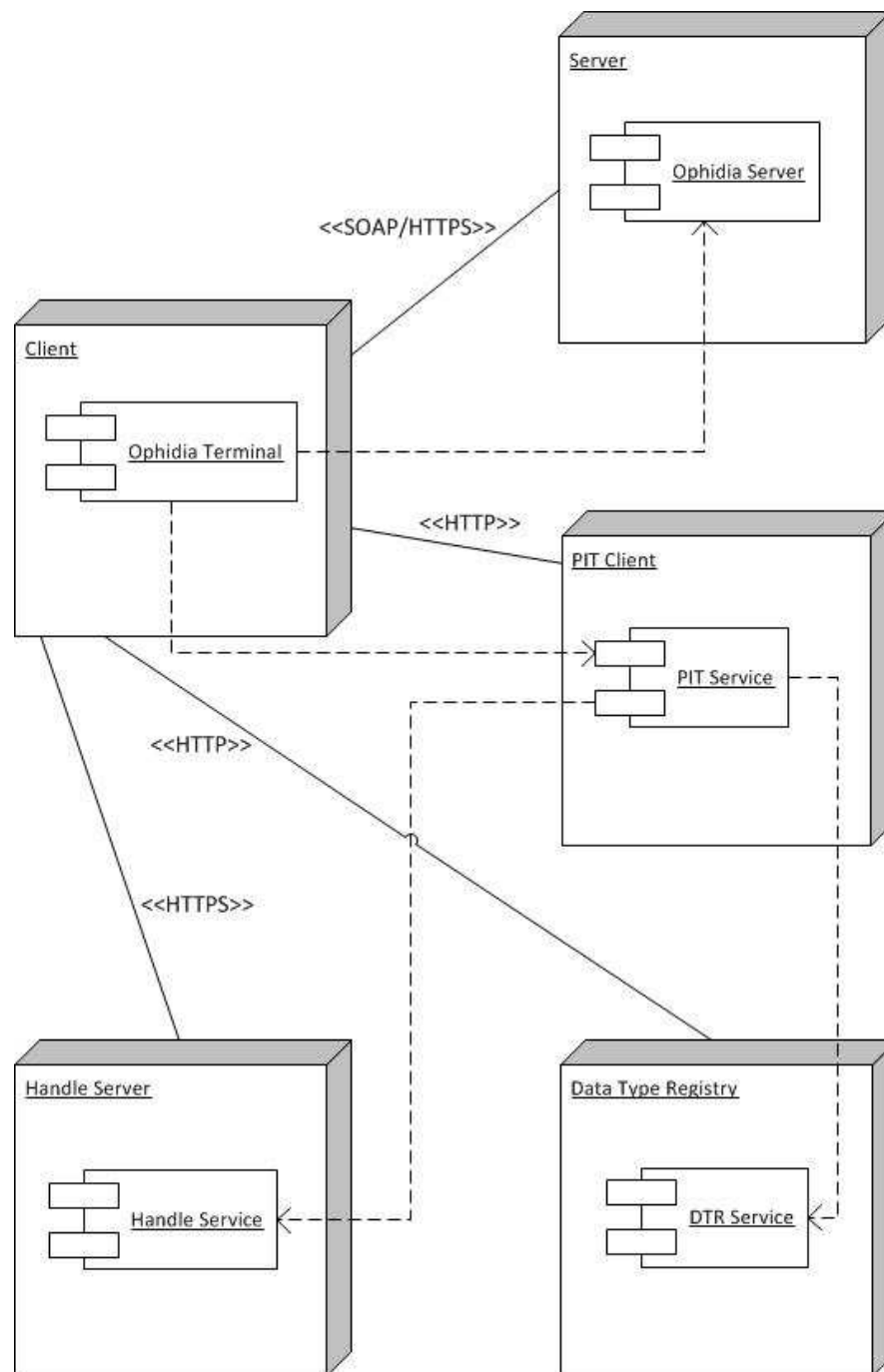


Fig. 14 - Deployment and components diagram

The Ophidia Terminal contains a client function that takes care of the interaction with the server through the SOAP protocol. SOAP may operate on different network protocols, among which the most commonly used is HTTP. In Ophidia, the HTTPS is used to secure communication with the SSL or GSI/VOMS infrastructure.

The PIT REST Web Service is designed to be agnostic towards the underlying identifier system. The PIT API prototype is written using Jersey 2 based on JAX-RS 2.0. It provides facilities to query and manipulate typed information available from persistent identifiers and all methods return JSON-encoded responses [7].

The client host communicates with both Handle Server and Data Type Registry using the HTTP/HTTPS protocol, as well as with the PIT Service.

Concerning the Handle Server HTTP interface, in addition to serve requests in the native binary Handle protocol of RFC 3652, version 8 is also able to serve requests made using a REST API [8], with requests and responses expressed via JSON encoding. All authenticated transactions, such as the creation of a new handle, must use HTTPS, otherwise they will be ignored resulting in a *403 Forbidden* response status.

In the same way, the PIT Service interacts with the Data Type Registry to retrieve information about the PID Information Types by exploiting its REST API.

Several basic tests have been carried out on the **PID Handle Service managed at DKRZ** (<https://handle.dkrz.de>), HTTP interface on port 8004) and using the **ePIC PID Information Types Registry**⁸ for types in preparation (<http://dtr.pidconsortium.eu:8081>).

Module 2: Design of the typed digital object for the PTA experiment

The structure of the typed digital object used to meet the precipitation trend analysis experiment requirements relies on well-defined information types, as defined in the reference Data Type Registry.

As shown in Figure 15, the PTA object is built out of a finite combination of non Basic PITs made available on the ePIC PID Information Type Registry for the preparation of types, a special data type registry environment where types are prepared and made stable by users before they become candidates and finally get approved.

PTA object

PIT Name	Range	PIT Identifier
creatorName	STRING	21.T11148/388da36a3d045e1b029a
email-address	STRING	21.T11148/e117a4a29bfd07438c1e
title	STRING	21.T11148/ec5125d411135ed263de
description	STRING	21.T11148/d6532ef6dc2b2a4ea01e
URL	STRING	21.T11148/e0efc41346cda4ba84ca
location	STRING	21.T11148/1bba2359c61cfee6948c
identifier-general	STRING	21.T11148/38330bcc6a40ca85e5b4
date-time	STRING	21.T11148/a045f55e2a7fc9d60a5b

Fig. 15 - Structure of the PTA typed digital object

⁸<http://dtr.pidconsortium.eu/>

Each non Basic PIT relies on a Basic PIT, which doesn't refer to other PITs but only uses the elementary JSON-Types ("string", "boolean", "integer", "number", no objects and arrays) with possible restrictions given for example by regular expressions or *minLength* and *maxLength* validation keywords[9].

A brief explanation of the meaning of each type in the PTA context is illustrated as follows:

creatorName	Name of the workflow submitter
email-address	Email address of the workflow submitter (useful for session sharing requests)
title	Name of the workflow
description	A short description of the typed object
URL	Reference to the workflow on GitHub
location	Reference to the output (datacube or file) of the workflow
identifier-general	Full workflow identifier (session id + workflow id)
date-time	Handle insertion timestamp

Module 3: Terminal extensions

The Ophidia Terminal has been extended by adding three commands which allow to respectively meet each of the use cases described above:

- `pit_insert`
- `pit_lookup`
- `pit_search`

In a multi-user scenario, these new features would enable different users to share their own experiments: a "producer" user could run a workflow and store the related data and metadata information, allowing a "consumer" user to search, extract and analyze them.

pit_insert command

The `pit_insert` command allows a user to publish an experiment on the PID Handle Service after running it.

In the PTA experiment scenario, a PTA typed digital object will be instantiated through the user inputs and stored at the Handle Server side. The newly created handle record will have a randomly and uniquely generated persistent identifier. The user has to provide semicolon separated key-value pairs in the form of

```
pit_insert PIT_name_1=PIT_value_1;...;PIT_name_N=PIT_value_N
```

where *PIT_name_i* stands for the name specified in the *type_i* property definition at Data Type Registry side, and *PIT_value_i* is the corresponding value provided by

the user. Values containing spaces must be typed in quotes, which could also be used (but not required) for single-word values.

All the PITs forming the PTA object are mandatory except the *date-time* type, which will be automatically set to the handle insertion date/time using the "YYYY-MM-ddHH:mm:ss" format, in accordance with the regular expression constraint defined for this type.

The command relies on the RDA-PID Information Types API and, in particular, on the **registerPID** method provided to create new identifiers:

```
@POST
@Path(value="/pid")
@Consumes(value="application/json")
@Produces(value="application/json")
public javax.ws.rs.core.Response registerPID(Map<String,String> properties)

Generic POST method to create new identifiers. The method determines an identifier name automatically, based on a purely random (version 4) UUID.

Parameters:
    properties - a map from string to string, mapping property identifiers to values.

Returns:
    a simple string with the newly created PID name.
```

Fig. 16 - RDA-PIT API *POST* /pid method

The PIT REST Service exploits, in turn, the Handle HTTP JSON REST API, which provides a PUT method to create a handle or replace its records:

PUT /api/handles/{handle}

PUT /api/handles/{handle}?index={index}

Create the handle {handle} or replace its handle record. If query parameters specify specific indices, add or replace those specific handle values in the handle record.

Request entity: an array of handle values, or an object with property values an array of handle values (other properties are ignored).

URI query parameters:

- **index={index}**
Specifies that only the handle value with index {index} should be added or replaced. The query parameter can be repeated to indicate a collection of handle values. The indices must match the indices of the handle values in the request entity.
- **index=various**
A shortcut to indicate that the handle values given in the request entity should be added or replaced.
- **overwrite={true|false}**
If true, replace handle records of handles which already exist, or replace handle values which already exist. If false, return 409 Conflict for attempts to PUT an existing handle or existing handle values. Default: true.
- **mintNewSuffix={true|false}**
If true, the handle to be created is formed by appending a random server-generated string to the {handle} parameter. Note that the slash should be included in the {handle} parameter. Default: false.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "handle": The handle specified in the request.
- "message": For error responses, an error message.

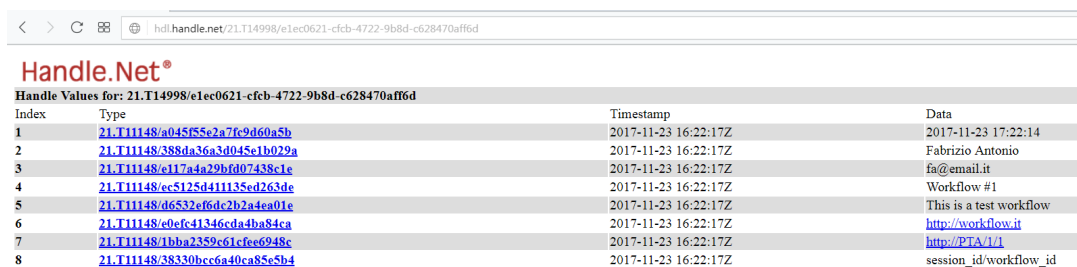
Fig. 17 - Handle API *PUT* /api/handles/{handle} method

The following command will create a new PTA handle record on the DKRZ Handle Server:

```
[46..1016] >> pit_insert creatorName="Fabrizio Antonio";email-address="fa@email.it";title="Workflow #1";description="This is a test workflow";URL="http://work
flow.it";location="http://PTA/1/1";identifier-general="session_id/workflow_id"
{"21.T11148/a045f5e2a7fc9d60a5b":"2017-11-23 17:22:14","21.T11148/388da36a3d045e1b029a":"Fabrizio Antonio","21.T11148/e117a4a29bfd07438c1e":"fa@email.it","21
.T11148/ec5125d411135ed263de":"Workflow #1","21.T11148/d6532ef6dc2b2adea01e":"This is a test workflow","21.T11148/e0efc41346cda4ba84ca":"http://workflow.it","21
.T11148/1bba2359c61cfec6948c":"http://PTA/1/1","21.T11148/38330bcc6a40ca85e5b4":"session_id/workflow_id"}
New PIT successfully created: 21.T14998/e1ec0621-cfcb-4722-9b8d-c628470aff6d
```

Fig. 18 - pit_insert example

The newly created PID can be resolved via the Handle global resolver (<http://hdl.handle.net>) or, alternatively, via the Handle Service running at DKRZ (<http://handle.dkrz.de:8004>):



Index	Type	Timestamp	Data
1	21.T11148/a045f5e2a7fc9d60a5b	2017-11-23 16:22:17Z	2017-11-23 17:22:14
2	21.T11148/388da36a3d045e1b029a	2017-11-23 16:22:17Z	Fabrizio Antonio
3	21.T11148/e117a4a29bfd07438c1e	2017-11-23 16:22:17Z	fa@email.it
4	21.T11148/ec5125d411135ed263de	2017-11-23 16:22:17Z	Workflow #1
5	21.T11148/d6532ef6dc2b2adea01e	2017-11-23 16:22:17Z	This is a test workflow
6	21.T11148/e0efc41346cda4ba84ca	2017-11-23 16:22:17Z	http://workflow.it
7	21.T11148/1bba2359c61cfec6948c	2017-11-23 16:22:17Z	http://PTA/1/1
8	21.T11148/38330bcc6a40ca85e5b4	2017-11-23 16:22:17Z	session_id/workflow_id

Fig. 19 - Resolution of a PID created by the pit_insert command

pit_lookup command

The pit_lookup command allows the user to retrieve all the handle values for the specified PID.

The user only has to provide the identifier of the handle to be resolved:

pit_lookup PID=identifier

The command relies on the **resolvePID** method provided by the PIT API to return all or some properties of an identifier:

```
@GET
@Path(value="/pid/{prefix}/{suffix}")
@Produces(value="application/json")
public javax.ws.rs.core.Response resolvePID(@PathParam(value="prefix")
String identifierPrefix,
@PathParam(value="suffix")
String identifierSuffix,
@QueryParam(value="filter_by_property")@DefaultValue(value="")
String propertyIdentifier,
@QueryParam(value="filter_by_type")
List<String> typeIdentifiers,
@QueryParam(value="include_property_names")@DefaultValue(value="false")
boolean includePropertyNames)
throws IOException,
InconsistentRecordsException
```

Fig. 20 - RDA-PIT API GET /pid/{prefix}/{suffix} method

The *include_property_names* query parameter is set to true in order to resolve the identifier of each PIT forming the handle record: in this way, the object extracted from the Handle Server is enriched by replacing the PIT identifiers with their own corresponding names.

The PIT Service interacts with both Handle Server and Data Type Registry.

In the former case, it exploits the **GET /api/handles/{handle}** method provided by the Handle REST API to resolve the handle record:

GET /api/handles/{handle}

Resolves the handle record for handle {handle}.

URI query parameters:

- **index={index}**
Specifies that only the handle value with index {index} should be resolved. The query parameter can be repeated to indicate a collection of handle values.
- **type={type}**
Specifies that only the handle values with type {type} should be resolved. If {type} ends with a period all period-delimited subtypes are included. The query parameter can be repeated. Multiple index and type parameters indicate that all handle values either of a matching index or a matching type should be resolved.
- **auth=[true|false]**
If true, perform an authoritative resolution, bypassing cache and sending the request to a primary server. Default: false. This flag is ignored in requests sent directly to an end server (instead of a proxy).
- **publicOnly=[true|false]**
If true, only resolve publicly readable handle values. If false, resolve all values, potentially resulting in a 401 Unauthorized response. Default: true for unauthenticated requests, false for authenticated requests.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "handle": The handle specified in the request.
- "message": For error responses, an error message.
- "values": An array of handle values.

Fig. 21 - Handle API GET /api/handles/{handle} method

Then, it uses the **GET /objects/<id>** method provided by the DTR API to resolve a PIT identifier. After extracting the property description for a given PIT, the PIT Service replaces its identifier with the corresponding name, as stated above.

Resource	Description
GET /objects/<id>	Retrieves an object by id.

Fig. 22 - DTR API GET /objects/<id> method

The following command retrieves all the information related to the handle record created by the previous pit_insert operation:

```
[46..1016] >> pit_lookup PID=21.T14998/elec0621-cfcb-4722-9b8d-c628470aff6d
creatorName      Fabrizio Antonio
title            Workflow #1
description       This is a test workflow
date-time        2017-11-23 17:22:14
URL              http://workflow.it
identifier-general session_id/workflow_id
email-address     fa@email.it
location          http://PTA/1/1
```

Fig. 23 - pit_lookup example

pit_search command

The `pit_search` command allows the user to search for different handles by some criteria.

The user has to provide semicolon separated key-value pairs in the form of

```
pit_search PIT_name_1=PIT_value_1;...;PIT_name_k=PIT_value_k
```

where `PIT_name_i` stands for the PID Information Type name and `PIT_value_i` is the type specific value the user is looking for. As for the insert command, values containing spaces must be typed in quotes.

Both the RDA-PIT API and the Handle API only allow to retrieve a single handle by its persistent identifier, but they provide no methods to perform reverse look-up operations.

Therefore, the `pit_search` command has been implemented as follows.

The PIT API has been extended by adding a new method, **GET /all/{prefix}**, which exploits the **GET /api/handles?prefix={prefix}** method provided by the Handle API to list all handles under the specified prefix.

GET /api/handles?prefix={prefix}

List handles under prefix {prefix}.

URI query parameters:

- **prefix={prefix}**
Required. Specifies the prefix of the handles to be listed.
- **page={page}**
pageSize={pageSize}
Specify paginated listing. The page number {page} is zero-based. If the page size {pageSize} is zero a count of handles is returned but no handles. If either parameter is missing or negative all handles are returned.

Response entity: an object with the following properties:

- "responseCode": Handle protocol response code for the message.
- "prefix": The prefix specified in the request.
- "message": For error responses, an error message.
- "totalCount": The total number of handles under the given prefix.
- "handles": An array of strings.

Fig. 24 - Handle API GET /api/handles/?prefix={prefix} method

For each PID in the extracted list, a look-up operation is performed to retrieve the handle values and apply the search filter provided by the user.

Once all handles have been processed, only the ones that meet the search criteria are retained and shown on the terminal.

The following example describes a search operation with a filter on the *creatorName* and *title* PITs.

Suppose we've got four PTA handle objects related to two different experiments and created by two different users: two handles created by "Fabrizio Antonio" for "Workflow #1", one handle created by "Fabrizio Antonio" for "Workflow #2" and one handle created by "Mario Rossi" for "Workflow #1".

Handle Values for: 21.T14998/e1ec0621-cfcb-4722-9b8d-c628470aff6d			Handle Values for: 21.T14998/9aaab6d9-9a16-4ef8-9676-b4346dad27e3		
Index	Type	Data	Index	Type	Data
1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-23 17:22:14	1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-24 12:48:13
2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio	2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio
3	21.T11148/e117a4a29bfd07438c1e	fa@email.it	3	21.T11148/e117a4a29bfd07438c1e	fa@email.it
4	21.T11148/ec5125d411135ed263de	Workflow #1	4	21.T11148/ec5125d411135ed263de	Workflow #1
5	21.T11148/d6532ef6dc2b2a4ea01e	This is a test workflow	5	21.T11148/d6532ef6dc2b2a4ea01e	This is another test workflow
6	21.T11148/e0efc41346cda4ba84ca	http://workflow.it	6	21.T11148/e0efc41346cda4ba84ca	http://workflow.it
7	21.T11148/1bba2359c61cfee6948c	http://PTA/1/1	7	21.T11148/1bba2359c61cfee6948c	http://PTA/1/2
8	21.T11148/38330bcc6a40ca85e5b4	session_id/workflow_id	8	21.T11148/38330bcc6a40ca85e5b4	session_id/workflow_id

Handle Values for: 21.T14998/c5113e77-8189-4c3a-8e8c-5f91172fa851			Handle Values for: 21.T14998/ac9b17a3-a1b1-4e7c-9c0c-a4feb2671c78		
Index	Type	Data	Index	Type	Data
1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-24 13:04:19	1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-24 12:49:01
2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio	2	21.T11148/388da36a3d045e1b029a	Mario Rossi
3	21.T11148/e117a4a29bfd07438c1e	fa@email.it	3	21.T11148/e117a4a29bfd07438c1e	mr@email.it
4	21.T11148/ec5125d411135ed263de	Workflow #2	4	21.T11148/ec5125d411135ed263de	Workflow #1
5	21.T11148/d6532ef6dc2b2a4ea01e	This is a test workflow	5	21.T11148/d6532ef6dc2b2a4ea01e	This is a test workflow
6	21.T11148/e0efc41346cda4ba84ca	http://workflow.it	6	21.T11148/e0efc41346cda4ba84ca	http://workflow.it
7	21.T11148/1bba2359c61cfee6948c	http://PTA/1/2	7	21.T11148/1bba2359c61cfee6948c	http://PTA/1/2
8	21.T11148/38330bcc6a40ca85e5b4	session_id/workflow_id	8	21.T11148/38330bcc6a40ca85e5b4	session_id/workflow_id

Fig. 25 - Example of PTA digital objects stored on the DKRZ Handle Server

Suppose we're interested in the experiments related to "Workflow #1" run by "Fabrizio Antonio". Figure 26 shows the corresponding command and the resulting output.

```
[46..1016] >> pit_search creatorName="Fabrizio Antonio";title="Workflow #1"
Searching for handles...
21.T14998/9AAAB6D9-9A16-4EF8-9676-B4346DAD27E3
21.T14998/E1EC0621-CFCB-4722-9B8D-C628470AFF6D
```

Fig. 26 - pit_search example with multiple filters

If we wanted to extract information about the experiment "Workflow #1" run by any user, we should remove the filter on the "creatorName" PIT, as shown in Figure 27.

```
[46..1016] >> pit_search title="Workflow #1"
Searching for handles...
21.T14998/9AAAB6D9-9A16-4EF8-9676-B4346DAD27E3
21.T14998/AC9B17A3-A1B1-4E7C-9C0C-A4FEB2671C78
21.T14998/E1EC0621-CFCB-4722-9B8D-C628470AFF6D
```

Fig. 27 - pit_search example with single filter

Module 4: PTA Use case

This section describes a typical use scenario related to the PTA experiment.

As stated in the "UML Diagrams" paragraph, two types of user can be identified:

- a "producer" (user P) runs the experiment and publishes it on the Handle Server, storing one or more PTA objects to share with the other users;
- a "consumer" (user C) can search for objects by some criteria, retrieve detailed information for a particular object, perform an in-depth analysis on the experiment and its corresponding output data.

Producer user

The user P submits the PTA experiment by means of the Ophidia Terminal, providing all the required input arguments:

```
[46..1016] >> ./Precipitation_Trend_Analysis.json 2 CMCC-CM|CMCC-CMS rcp85 day 0.9 1976_2006 2071_2101 30:45|0:40 r360x180  
[JobID]:  
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1240
```

Fig. 28 - Submission of the Precipitation Trend Analysis workflow

During the workflow execution, P can check its current status by calling the *view* command and specifying the *Workflow ID*. Figure 29 shows an intermediate snapshot of the status of the workflow: the green, yellow, grey and orange disks represent completed, skipped, unscheduled or unselected, and running tasks, respectively.

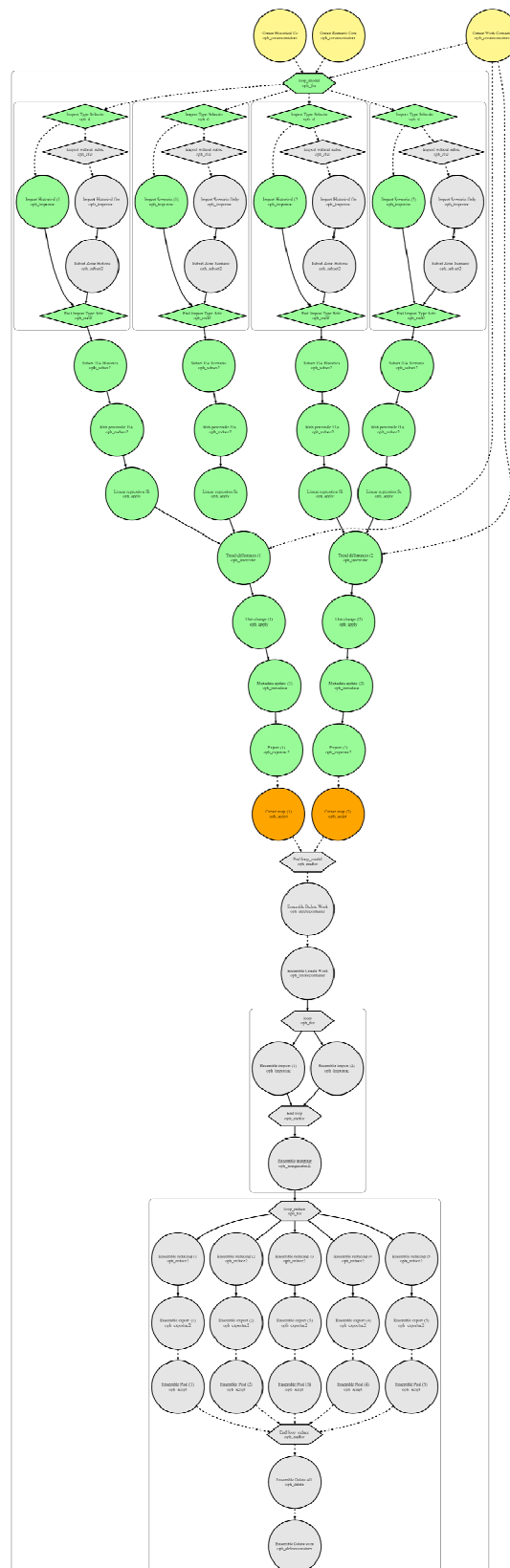


Fig. 29 - Intermediate snapshot of the status of the workflow

Once the execution ends, the user can publish the experiment.

First, P can obtain additional details, such as the workflow task list, by exploiting the *view* command again:

```
[46..1016] >> view 103#1240
[103#1240] /optimized/precip_trend_analysis.json 2 CMCC-CM[CMCC-CMS rcp85 day 0.9 1976_2006 2071_2101 30:45]0:40 r360x180 [https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1240]

[Response]:
Workflow Image File : precip_trend_analysis.svg
Workflow Status
-----
OPH_STATUS_COMPLETED

Workflow Progress
-----
=====
| NUMBER OF COMPLETED TASKS | NUMBER OF SKIPPED TASKS | TOTAL NUMBER OF TASKS |
=====
| 59 | 16 | 75 |
=====

Workflow Task List
-----
=====
| OPH_JOB_ID | SESSION CODE | WORKFLOW ID | MARKER ID | PARENT MARKER ID | TASK NAME | TYPE | EXIT STATUS |
=====
| https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1241 | 46558764513069252461504248441721016 | 103 | 1241 | 1240 | Create Work Container | SIM_PLE | OPH_STATUS_COMPLETED |
| https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1242 | 46558764513069252461504248441721016 | 103 | 1242 | 1240 | Create Historical Container | SIM_PLE | OPH_STATUS_SKIPPED |
=====
```

Fig. 30 - Output of the view command

Then, P can obtain the output of each task by referring to the task by its *Marker ID*.

The PTA experiment consists of a number (two in the example above) of sub-workflows executed in parallel, followed by a final workflow performing an ensemble analysis. So, in addition to the output datacubes resulting from the ensemble analysis, P could also publish the intermediate results coming from the two single analysis, in order to allow a comprehensive analysis of the experiment.

Figures 31 and 32 show the EXPORT tasks, extracted from the workflow task list provided by the *view* command, useful to retrieve the PID of the corresponding datacubes.

https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1276	46558764513069252461504248441721016	103	1276	1240	Export (2)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1277	46558764513069252461504248441721016	103	1277	1240	Create map (2)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1278	46558764513069252461504248441721016	103	1278	1240	Trend differences (1)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1279	46558764513069252461504248441721016	103	1279	1240	Unit change (1)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1280	46558764513069252461504248441721016	103	1280	1240	Metadata update (1)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1281	46558764513069252461504248441721016	103	1281	1240	Export (1)	SIM_PLE	OPH_STATUS_COMPLETED

Fig. 31 - Portion of the view command output related to the single analysis EXPORT tasks

https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1297	46558764513069252461504248441721016	103	1297	1240	Ensemble export (1)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1298	46558764513069252461504248441721016	103	1298	1240	Ensemble export (5)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1299	46558764513069252461504248441721016	103	1299	1240	Ensemble export (3)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1300	46558764513069252461504248441721016	103	1300	1240	Ensemble export (4)	SIM_PLE	OPH_STATUS_COMPLETED
https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1301	46558764513069252461504248441721016	103	1301	1240	Ensemble export (2)	SIM_PLE	OPH_STATUS_COMPLETED

Fig. 32 - Portion of the view command output related to the ensemble analysis EXPORT tasks

RDA Europe 3 - Collaboration Project Final Report

For each task, P can use the *Workflow ID - Marker ID* pair to obtain the desired information:

```
[46..1016] >> view 103#1276
[103#1276] oph exportnc2 force=yes;output_name=precip_trend_analysis_CMCC-CMS;output_path=/INDIGO/precip_trend_input/46558764513069252461504248441721016/103;c
wd=/cdd:/home/fantonio/prec_trend_analysis_parallel_workflow;cube=https://ophidialab.cmcc.it/ophidia/374/85288; [https://ophidialab.cmcc.it/ophidia/sessions/
46558764513069252461504248441721016/experiment?103#1276]

[46..1016] >> view 103#1281
[103#1281] oph exportnc2 force=yes;output_name=precip_trend_analysis_CMCC-CM;output_path=/INDIGO/precip_trend_input/46558764513069252461504248441721016/103;c
wd=/cdd:/home/fantonio/prec_trend_analysis_parallel_workflow;cube=https://ophidialab.cmcc.it/ophidia/374/85292; [https://ophidialab.cmcc.it/ophidia/sessions/4
6558764513069252461504248441721016/experiment?103#1281]

[46..1016] >> view 103#1297
[103#1297] oph exportnc2 force=yes;misc=yes;output_name=avg;cwd=/;cdd=/home/fantonio/prec_trend_analysis_parallel_workflow;cube=https://ophidialab.cmcc.it/oph
idia/416/85296; [https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/experiment?103#1297]
```

Fig. 33 - Commands to obtain the output of a task by referring to it by its Marker ID

Finally, for each datacube to be published, P can create and store a PTA object on the Handle Server by using the *pit_insert* command:

```
[46..1016] >> pit_insert creatorName="Fabrizio Antonio";email-address="fa@email.it";title="Precipitation Trend Analysis";description="Single analysis for CMCC
-CM";URL="https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json";location="https://ophidialab.cmcc.it/ophidia/374/852
88";identifier-general="https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"
{"21.T11148/a045f55e2a7fc9d60a5b":"2017-11-26 01:03:16","21.T11148/388da36a3d045e1b029a":"Fabrizio Antonio","21.T11148/e117a4a29bfd07438c1e":"fa@email.it","21
.T11148/ec5125d41135ed263de":"Precipitation Trend Analysis","21.T11148/d6532ef6dc2b2a4ea01e":"Single analysis for CMCC-CM","21.T11148/e0efc41346cda4ba84ca":"
https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json","21.T11148/1bba2359c61cfee6948c":"https://ophidialab.cmcc.it/oph
idia/374/85292","21.T11148/38330bcc6a40ca85e5b4":"https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"}
New PIT successfully created: 21.T14998/947b7c55-bbb6-4bd5-a119-29f217a76995

[46..1016] >> pit_insert creatorName="Fabrizio Antonio";email-address="fa@email.it";title="Precipitation Trend Analysis";description="Single analysis for CMCC
-CM";URL="https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json";location="https://ophidialab.cmcc.it/ophidia/374/85
288";identifier-general="https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"
{"21.T11148/a045f55e2a7fc9d60a5b":"2017-11-26 01:04:11","21.T11148/388da36a3d045e1b029a":"Fabrizio Antonio","21.T11148/e117a4a29bfd07438c1e":"fa@email.it","21
.T11148/ec5125d41135ed263de":"Precipitation Trend Analysis","21.T11148/d6532ef6dc2b2a4ea01e":"Single analysis for CMCC-CM","21.T11148/e0efc41346cda4ba84ca":"
https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json","21.T11148/1bba2359c61cfee6948c":"https://ophidialab.cmcc.it/oph
idia/374/85288","21.T11148/38330bcc6a40ca85e5b4":"https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"}
New PIT successfully created: 21.T14998/bece26cd-20c6-4188-be60-bc11844171e2

[46..1016] >> pit_insert creatorName="Fabrizio Antonio";email-address="fa@email.it";title="Precipitation Trend Analysis";description="Ensemble analysis: avg";
URL="https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json";location="https://ophidialab.cmcc.it/ophidia/416/85296";i
dentifier-general="https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"
{"21.T11148/a045f55e2a7fc9d60a5b":"2017-11-26 01:05:14","21.T11148/388da36a3d045e1b029a":"Fabrizio Antonio","21.T11148/e117a4a29bfd07438c1e":"fa@email.it","21
.T11148/ec5125d41135ed263de":"Precipitation Trend Analysis","21.T11148/d6532ef6dc2b2a4ea01e":"Ensemble analysis: avg","21.T11148/e0efc41346cda4ba84ca":"https
://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json","21.T11148/1bba2359c61cfee6948c":"https://ophidialab.cmcc.it/ophidi
a/416/85296","21.T11148/38330bcc6a40ca85e5b4":"https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json"}
New PIT successfully created: 21.T14998/91733b3c-d641-4b07-901f-cc6ebc5ee14
```

Fig. 34 - pit_insert commands for publishing the PTA experiment

The three resulting handles are shown in Figure 35.

Handle Values for: 21.T14998/947b7c55-bbb6-4bd5-a119-29f217a76995		
Index	Type	Data
1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-26 01:03:16
2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio
3	21.T11148/e117a4a29bfd07438c1e	fa@email.it
4	21.T11148/ec5125d41135ed263de	Precipitation Trend Analysis
5	21.T11148/d6532ef6dc2b2a4ea01e	Single analysis for CMCC-CM
6	21.T11148/e0efc41346cda4ba84ca	https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
7	21.T11148/1bba2359c61cfee6948c	https://ophidialab.cmcc.it/ophidia/374/85292
8	21.T11148/38330bcc6a40ca85e5b4	https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json

Handle Values for: 21.T14998/bece26cd-20c6-4188-be60-bc11844171e2		
Index	Type	Data
1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-26 01:04:11
2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio
3	21.T11148/e117a4a29bfd07438c1e	fa@email.it
4	21.T11148/ec5125d41135ed263de	Precipitation Trend Analysis
5	21.T11148/d6532ef6dc2b2a4ea01e	Single analysis for CMCC-CMS
6	21.T11148/e0efc41346cda4ba84ca	https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
7	21.T11148/1bba2359c61cfee6948c	https://ophidialab.cmcc.it/ophidia/374/85288
8	21.T11148/38330bcc6a40ca85e5b4	https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json

Handle Values for: 21.T14998/91733b5c-de41-40b7-901f-cce6e0c5ee14		
Index	Type	Data
1	21.T11148/a045f55e2a7fc9d60a5b	2017-11-26 01:05:14
2	21.T11148/388da36a3d045e1b029a	Fabrizio Antonio
3	21.T11148/e117a4a29bfd07438c1e	fa@email.it
4	21.T11148/ec5125d411135ed263de	Precipitation Trend Analysis
5	21.T11148/d6532ef6dc2b2a4ea01e	Ensemble analysis: avg
6	21.T11148/e0efc41346cda4ba84ca	https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
7	21.T11148/1bba2359c61cfec6948c	https://ophidialab.cmcc.it/ophidia/416/85296
8	21.T11148/38330bcc6a40ca85e5b4	https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json

Fig. 35 - PTA objects

Consumer user

The user C, who is interested in studying and analyzing the PTA experiment, will perform the following actions:

- search of PTA objects related to a PTA experiment run by a specific user.
The user C will execute the *pit_search* command providing a filter on the *title* (the name of the workflow) and the *creatorName*(the owner of the workflow) PITs:

```
[46..1016] >> pit_search creatorName="Fabrizio Antonio";title="Precipitation Trend Analysis"
Searching for handles...
21.T14998/91733B5C-DE41-40B7-901F-CCE6E0C5EE14
21.T14998/947B7C55-BBBC-4B05-A119-29F217A76995
21.T14998/BECE26CD-20C6-4188-BE60-BC11844171E2
```

Fig. 36 - Search of PTA objects

With regard to the “Producer user” scenario described above, the command will output the PIDs of the three handles previously created by the user P.

- look-up of each retrieved PID to extract all the data and metadata values stored in the corresponding typed digital object:

```
[46..1016] >> pit_lookup PID=21.T14998/91733B5C-DE41-40B7-901F-CCE6E0C5EE14
creatorName      Fabrizio Antonio
title            Precipitation Trend Analysis
description       Ensemble analysis: avg
date-time        2017-11-26 01:05:14
URL              https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
identifier-general https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json
email-address     fa@email.it
location         https://ophidialab.cmcc.it/ophidia/416/85296

[46..1016] >> pit_lookup PID=21.T14998/947B7C55-BBBC-4B05-A119-29F217A76995
creatorName      Fabrizio Antonio
title            Precipitation Trend Analysis
description       Single analysis for CMCC-CM
date-time        2017-11-26 01:03:16
URL              https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
identifier-general https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json
email-address     fa@email.it
location         https://ophidialab.cmcc.it/ophidia/374/85292

[46..1016] >> pit_lookup PID=21.T14998/BECE26CD-20C6-4188-BE60-BC11844171E2
creatorName      Fabrizio Antonio
title            Precipitation Trend Analysis
description       Single analysis for CMCC-CMS
date-time        2017-11-26 01:04:11
URL              https://github.com/OphidiaBigData/ophidia-workflow-catalogue/Precipitation_Trend_Analysis.json
identifier-general https://ophidialab.cmcc.it/ophidia/sessions/46558764513069252461504248441721016/json/request/103.json
email-address     fa@email.it
location         https://ophidialab.cmcc.it/ophidia/374/85288
```

Fig. 37 - Look-up of PTA objects

- send an email to the workflow owner for the latter to share the session, thus gaining access to all workflow information (workspace, data, results, exported objects). The email address of the owner is given by the corresponding PIT in the extracted objects, while the session identifier is

part of the value of the *identifier-general* PIT. The owner of the session can finely grant access privileges by using the OPH_MANAGE_SESSION Ophidia operator.

At this point, the user C will be able to perform any kind of analysis. Specifically, the user can:

- get the provenance of an output datacube, that is the hierarchy of all datacubes used to generate it and of those derived from it.

As stated above, in the specific case of the PTA experiment, the workflow consists of two parallel single analyses followed by an ensemble analysis, which includes a data re-gridding step. This requires a second INPUT task between the two phases. The user can retrace the full provenance of a workflow output datacube (resulting from the ensemble analysis) as sum of the partial provenances obtained by executing the OPH_CUBEIO Ophidia operator on multiple datacubes, among those published for the experiment. The identifiers of the intermediate and final datacubes are stored as values for the *location* PIT, as shown in Figure 37.

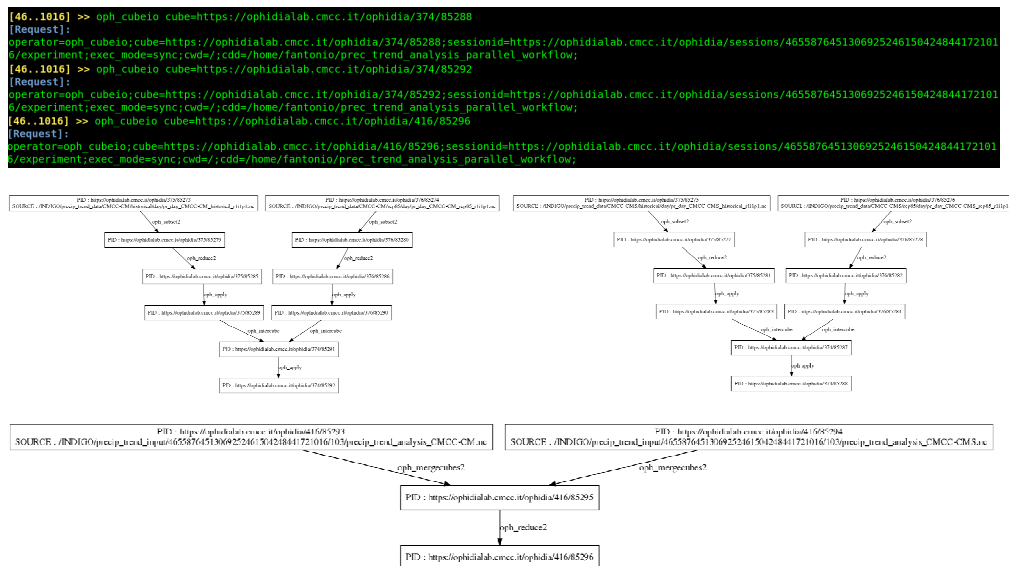


Fig. 38 - Provenance of a PTA experiment output datacube

- download the original (parameterized) workflow version, using the URL stored in the corresponding *URL* PIT;
- analyze the workflow instance (submitted version with input arguments substitution) run by the owner of the workflow, using the *identifier-general* PIT;
- view the workflow status, progress and task list using the *view* command and the workflow ID;
- rerun the experiment by submitting the workflow with different inputs;
- export data of a datacube into multiple NetCDF files using the OPH_EXPORTNC Ophidia operator;
- access and analyze the intermediate and final results using the datacubes identifiers specified in the *location* PIT of each PTA objects.

For example, the user can view metadata information about a datacube and its dimensions using the OPH_CUBESCHEMA operator:

```
[46..1010] >> oph_cubeschema cube=https://ophidialab.cmcc.it/ophidia/416/85296
[Request]:
operator=oph_cubeschema;cube=https://ophidialab.cmcc.it/ophidia/416/85296;sessionId=https://ophidialab.cmcc.it/ophidia/sessions/4655876451306925246150
21016/experiment;exec_mode=sync;cwd=/;cdd=/home/fantonio/prec_trend_analysis_parallel_workflow;
[Response]:
Datacube Information
=====
| PTID | CREATION DATE | MEASURE | MEASURE TYPE | LEVEL | NUMBER OF FRAGMENTS | SOURCE FILE |
=====
| https://ophidialab.cmcc.it/ophidia/416/85296 | 2017-11-25 18:55:06 | precip_trend | float | 2 | 1 | |
=====
Datacube Additional Information
=====
| DESCRIPTOR | HOST x CU | DBMS x HO | DATABASES x DB | FRAGMENTS x DATAB | ROWS x FRAGME | ELEMENTS x R | COMPRESS | CUBE SIZE | UNIT | NUMBER OF E |
| ON | BE | ST | MS | ASE | NT | OW | ED | E | T | TS |
=====
| 1 | 1 | 1 | 1 | 1 | 181 | 361 | no | | | 65341 |
=====
Dimension Information
=====
| NAME | TYPE | SIZE | HIERARCHY | CONCEPT LEVEL | ARRAY | LEVEL | LATTICE NAME |
=====
| lat | double | 181 | oph_base | cell | no | 1 | |
| new_dim | long | ALL | oph_base | ALL | yes | 0 | |
| lon | double | 361 | oph_base | cell | yes | 1 | |
=====
```

Fig. 39 - OPH_CUBESCHEMA example

Dissemination Activities / Publications

This work has also been mentioned during the INDIGO Review which took place on Nov 16 and 17 in Brussels.

#	Event	Date	Publication/Dissemination activity
1	ESGF2017 Conference	4-8/12/2017	Presentation
2	AGU2017 Conference	11/12/2017	Presentation
3	CF2017	15/5/2017	Presentation
4	HPC Summit 2017	16/5/2017	Presentation
5	RDA School	25/5/2017	Presentation
6	Invited lecture at BSC	25/5/2017	Presentation
7	EGI-INDIGO Summit	11/5/2017	Presentation
8	EGU2017	25/04/2017	Presentation

Summary & Conclusions

The report provides an introduction to the design and implementation of the Ophidia support for RDA-PIT. As discussed in this work, the big data analytics framework has been extended to include the PID Handle Service client API. In a multi-user scenario, these new features would enable different users to share their own experiments.

In that regard, a multi-model climate analytics experiment case study implemented in the context of the EU INDIGO-DataCloud project has been provided. A user can run the experiment and share it with other users by storing several typed digital objects on the identifier system; on the other hand, another user could search and access these objects in order to analyze the related data and metadata.

We aim to improve the current set of PIT-based commands in future work, by adding new functionalities, such as replacing or deleting an existing object, and we also plan to evaluate and, if needed, optimize system performance.

Annex I - PTA workflow

```
{
  "name": "precip_trend_analysis",
  "author": "CMCC Foundation",
  "abstract": "Workflow for the analysis of precipitation trends
related to different scenarios. ${1} is ncores; ${2} is the list of
models (e.g. CMCC-CM|CMCC-CMS); ${3} is the scenario (e.g. rcp45 or
rcp85); ${4} is the frequency (e.g. day or mon); ${5} is the
percentile (e.g. 0.9); ${6} is the past time subset (e.g. 1976_2006);
${7} is the future time subset (e.g. 2071_2101); ${8} is the
geographic subset (e.g. 30:45|0:40); ${9} is the grid of output map
using the format r<lon>x<lat> (e.g. r360x180), i.e. a global regular
lon/lat grid; ${10} import type (optional), set to '1' in case only
subsetting data have to be imported (default); ${11} I/O server type
(optional).",
  "exec_mode": "async",
  "cwd": "/",
  "ncores": "${1}",
  "on_exit": "oph_delete",
  "host_partition": "test",
  "tasks": [
    {
      "name": "Create Work Container",
      "operator": "oph_createcontainer",
      "arguments": [
        "container=work",
        "dim=lat|lon|time",
        "dim_type=double|double|double",
        "hierarchy=oph_base|oph_base|oph_time",
        "compressed=no",
        "ncores=1",
        "base_time=1976-01-01",
        "calendar=standard",
        "units=d"
      ],
      "on_error": "skip"
    },
    {
      "name": "Create Historical Container",
      "operator": "oph_createcontainer",
      "arguments": [
        "container=historical",
        "dim=lat|lon|time",
        "dim_type=double|double|double",
        "hierarchy=oph_base|oph_base|oph_time",
        "compressed=no",
        "ncores=1",
        "base_time=1976-01-01",
        "calendar=standard",
        "units=d"
      ],
      "on_error": "skip"
    },
    {
      "name": "Create Scenario Container",
      "operator": "oph_createcontainer",
      "arguments": [
        "container=scenario",
        "dim=lat|lon|time",
        "dim_type=double|double|double",

```

```

    "hierarchy=oph_base|oph_base|oph_time",
    "compressed=no",
    "ncores=1",
    "base_time=2070-01-01",
    "calendar=standard",
    "units=d"
  ],
  "on_error": "skip"
},
{
  "name": "loop_model",
  "operator": "oph_for",
  "arguments": [
    "key=model",
    "values=${2}",
    "parallel=yes"
  ],
  "dependencies": [
    { "task": "Create Work Container"},
    { "task": "Create Historical Container"},
    { "task": "Create Scenario Container"}
  ]
},
{
  "name": "Import Type Selection Historical",
  "operator": "oph_if",
  "arguments": [ "condition=${10}" ],
  "dependencies": [
    { "task": "loop_model" }
  ]
},
{
  "name": "Import Historical",
  "operator": "oph_importnc",
  "arguments": [
    "container=historical",
    "exp_dim=lat|lon",
    "imp_dim=time",
    "measure=pr",
    "src_path=/INDIGO/precip_trend_data/@{model}/historical/${4}/pr_${4}_@{model}_historical_rlilpl.nc",
    "compressed=no",
    "exp_concept_level=c|c",
    "filesystem=local",
    "imp_concept_level=${4}",
    "ndb=1",
    "ndbms=1",
    "nhost=1",
    "import_metadata=yes",
    "check_compliance=no",
    "units=d",
    "subset_dims=time|lat|lon",
    "subset_filter=${6}|${8}",
    "subset_type=coord",
    "offset=0|2|2",
    "ioserver=${11}"
  ],
  "dependencies": [
    { "task": "Import Type Selection Historical" }
  ]
},

```

```

{
  "name": "Import without subsetting Historical",
  "operator": "oph_else",
  "arguments": [ ],
  "dependencies": [
    { "task": "Import Type Selection Historical" }
  ],
},
{
  "name": "Import Historical Only",
  "operator": "oph_importnc",
  "arguments": [
    "container=historical",
    "exp_dim=lat|lon",
    "imp_dim=time",
    "measure=pr",
    "src_path=/INDIGO/precip_trend_data/@{model}/historical/${4}/pr
_${4}_@{model}_historical_rlilpl.nc",
    "compressed=no",
    "exp_concept_level=c|c",
    "filesystem=local",
    "imp_concept_level=${4}",
    "ndb=1",
    "ndbms=1",
    "nhost=1",
    "import_metadata=yes",
    "check_compliance=no",
    "units=d",
    "ioserver=${11}"
  ],
  "dependencies": [
    { "task": "Import without subsetting Historical" }
  ],
},
{
  "name": "Subset Zone Historical",
  "operator": "oph_subset2",
  "arguments": [
    "subset_dims=time|lat|lon",
    "subset_filter=${6}|${8}",
    "offset=0|2|2"
  ],
  "dependencies": [
    { "task": "Import Historical Only", "type": "single" }
  ],
},
{
  "name": "End Import Type Selection Historical",
  "operator": "oph_endif",
  "arguments": [ ],
  "dependencies": [
    { "task": "Import Historical", "type": "single" },
    { "task": "Subset Zone Historical", "type": "single" }
  ],
},
{
  "name": "Subset JJA Historical",
  "operator": "oph_subset2",
  "arguments": [
    "subset_dims=time",

```

RDA Europe 3 - Collaboration Project Final Report

```
"subset_filter=1976-06_1976-09,1977-06_1977-09,1978-06_1978-
09,1979-06_1979-09,1980-06_1980-09,1981-06_1981-09,1982-06_1982-
09,1983-06_1983-09,1984-06_1984-09,1985-06_1985-09,1986-06_1986-
09,1987-06_1987-09,1988-06_1988-09,1989-06_1989-09,1990-06_1990-
09,1991-06_1991-09,1992-06_1992-09,1993-06_1993-09,1994-06_1994-
09,1995-06_1995-09,1996-06_1996-09,1997-06_1997-09,1998-06_1998-
09,1999-06_1999-09,2000-06_2000-09,2001-06_2001-09,2002-06_2002-
09,2003-06_2003-09,2004-06_2004-09,2005-06_2005-09"
],
"dependencies": [
{ "task": "End Import Type Selection Historical", "type":
"single" }
],
{
"name": "90th percentile JJA Historical",
"operator": "oph_reduce2",
"arguments": [
"operation=quantile",
"dim=time",
"concept_level=y",
"order=${5}"
],
"dependencies": [
{ "task": "Subset JJA Historical", "type": "single" }
],
{
"name": "Linear regression Historical",
"operator": "oph_apply",
"arguments": [
"query=oph_gsl_fit_linear_coeff(measure)",
"measure_type=auto"
],
"dependencies": [
{ "task": "90th percentile JJA Historical", "type": "single" }
],
{
"name": "Import Type Selection Scenario",
"operator": "oph_if",
"arguments": [ "condition=${10}" ],
"dependencies": [
{ "task": "loop_model" }
],
{
"name": "Import Scenario",
"operator": "oph_importnc",
"arguments": [
"container=scenario",
"exp_dim=lat|lon",
"imp_dim=time",
"measure=pr",
"src_path=/INDIGO/precip_trend_data/@{model}/${3}/${4}/pr_${4}_
@{model}_${3}_r1l1p1.nc",
"compressed=no",
"exp_concept_level=c|c",
"filesystem=local",
"imp_concept_level=${4}",
"ndb=1",
```

```

"ndbms=1",
"nhost=1",
"import_metadata=yes",
"check_compliance=no",
"units=d",
"subset_dims=time|lat|lon",
"subset_filter=${7}|${8}",
"subset_type=coord",
"offset=0|2|2",
"ioserver=${11}"
],
"dependencies": [
{ "task": "Import Type Selection Scenario" }
],
{
"name": "Import without subsetting Scenario",
"operator": "oph_else",
"arguments": [ ],
"dependencies": [
{ "task": "Import Type Selection Scenario" }
],
{
"name": "Import Scenario Only",
"operator": "oph_importnc",
"arguments": [
"container=scenario",
"exp_dim=lat|lon",
"imp_dim=time",
"measure=pr",
"src_path=/INDIGO/precip_trend_data/@{model}/${3}/${4}/pr_${4}_
@{model}_${3}_r1ilp1.nc",
"compressed=no",
"exp_concept_level=c|c",
"filesystem=local",
"imp_concept_level=${4}",
"ndb=1",
"ndbms=1",
"nhost=1",
"import_metadata=yes",
"check_compliance=no",
"units=d",
"ioserver=${11}"
],
"dependencies": [
{ "task": "Import without subsetting Scenario" }
],
{
"name": "Subset Zone Scenario",
"operator": "oph_subset2",
"arguments": [
"subset_dims=time|lat|lon",
"subset_filter=${7}|${8}",
"offset=0|2|2"
],
"dependencies": [
{ "task": "Import Scenario Only", "type": "single" }
]
}

```



```

    },
    {
      "name": "End Import Type Selection Scenario",
      "operator": "oph_endif",
      "arguments": [ ],
      "dependencies": [
        { "task": "Import Scenario", "type": "single" },
        { "task": "Subset Zone Scenario", "type": "single" }
      ]
    },
    {
      "name": "Subset JJA Scenario",
      "operator": "oph_subset2",
      "arguments": [
        "subset_dims=time",
        "subset_filter=2071-06_2071-09,2072-06_2072-09,2073-06_2073-
09,2074-06_2074-09,2075-06_2075-09,2076-06_2076-09,2077-06_2077-
09,2078-06_2078-09,2079-06_2079-09,2080-06_2080-09,2081-06_2081-
09,2082-06_2082-09,2083-06_2083-09,2084-06_2084-09,2085-06_2085-
09,2086-06_2086-09,2087-06_2087-09,2088-06_2088-09,2089-06_2089-
09,2090-06_2090-09,2091-06_2091-09,2092-06_2092-09,2093-06_2093-
09,2094-06_2094-09,2095-06_2095-09,2096-06_2096-09,2097-06_2097-
09,2098-06_2098-09,2099-06_2099-09,2100-06_2100-09"
      ],
      "dependencies": [
        { "task": "End Import Type Selection Scenario", "type":
"single" }
      ]
    },
    {
      "name": "90th percentile JJA Scenario",
      "operator": "oph_reduce2",
      "arguments": [
        "operation=quantile",
        "dim=time",
        "concept_level=y",
        "order=${5}"
      ],
      "dependencies": [
        { "task": "Subset JJA Scenario", "type": "single" }
      ]
    },
    {
      "name": "Linear regression Scenario",
      "operator": "oph_apply",
      "arguments": [
        "query=oph_gsl_fit_linear_coeff(measure)",
        "measure_type=auto"
      ],
      "dependencies": [
        { "task": "90th percentile JJA Scenario", "type": "single" }
      ]
    },
    {
      "name": "Trend differences",
      "operator": "oph_intercube",
      "arguments": [
        "operation=sub",
        "measure=precip_trend",
        "container=work"
      ],

```

RDA Europe 3 - Collaboration Project Final Report

```

        "dependencies": [
          { "task": "Create Work Container" },
          { "task": "Linear regression Scenario", "argument": "cube",
"type": "single" },
          { "task": "Linear regression Historical", "argument": "cube2",
"type": "single" }
        ],
      },
      {
        "name": "Unit change",
        "operator": "oph_apply",
        "arguments": [
          "query=oph_mul_scalar(measure,86400)",
          "measure_type=auto"
        ],
        "dependencies": [
          { "task": "Trend differences", "type": "single" }
        ],
      },
      {
        "name": "Metadata update",
        "operator": "oph_metadata",
        "arguments": [
          "mode=insert",
          "metadata_key=standard_name|long_name|units",
          "metadata_value=precipitation_trend|PrecipitationTrend|mm    d-1
y-1",
          "variable=precip_trend"
        ],
        "dependencies": [
          { "task": "Unit change", "type": "single" }
        ],
      },
      {
        "name": "Export",
        "operator": "oph_exportnc2",
        "arguments": [
          "force=yes",
          "output_name=precip_trend_analysis_{model}",
          "output_path=/INDIGO/precip_trend_input/{OPH_SESSION_CODE}/{O
PH_WORKFLOW_ID}"
        ],
        "dependencies": [
          { "task": "Metadata update", "type": "single" }
        ],
      },
      {
        "name": "Create map",
        "operator": "oph_script",
        "arguments": [
          "script=precip_trend_analysis",
          "args=precip_trend_analysis_{model}.nc ${8} ${9}"
        ],
        "dependencies": [
          { "task": "Export" }
        ],
      },
      {
        "name": "End loop_model",
        "operator": "oph_endfor",
        "arguments": [],

```

```

"dependencies": [
{ "task": "Create map" }
],
{
"name": "Ensemble Delete Work Container",
"operator": "oph_deletecontainer",
"arguments": [
"container=ensemble",
"hidden=no",
"delete_type=physical"
],
"dependencies": [
{ "task": "End loop_model" }
],
"on_error": "skip"
},
{
"name": "Ensemble Create Work Container",
"operator": "oph_createcontainer",
"arguments": [
"container=ensemble",
"dim=lat|lon",
"dim_type=double|double",
"hierarchy=oph_base|oph_base",
"compressed=no",
"ncores=1"
],
"dependencies": [
{ "task": "Ensemble Delete Work Container" }
],
"on_error": "skip"
},
{
"name": "loop",
"operator": "oph_for",
"arguments": [
"key=model",
"values=${2}",
"parallel=yes"
],
"dependencies": [
{
"task": "Ensemble Create Work Container",
"type": "single"
}
],
},
{
"name": "Ensemble import",
"operator": "oph_importnc",
"arguments": [
"cwd=",
"measure=precip_trend",
"src_path=/INDIGO/precip_trend_input/@{OPH_SESSION_CODE}/@{OPH_
WORKFLOW_ID}/precip_trend_analysis_{model}.nc",
"import_metadata=yes",
"nfrag=1",
"container=ensemble",
"grid=map",
"ioserver=${11}"

```

```

],
"dependencies": [
{
"task": "loop",
"type": "single"
}
],
},
{
"name": "End loop",
"operator": "oph_endfor",
"arguments": [],
"dependencies": [
{
"task": "Ensemble import",
"type": "all"
}
]
},
{
"name": "Ensemble merging",
"operator": "oph_mergetcubes2",
"arguments": ["dim=new_dim"],
"dependencies": [
{
"task": "End loop",
"type": "all",
"argument": "cubes"
}
]
},
{
"name": "loop_reduce",
"operator": "oph_for",
"arguments": [ "key=index", "values=avg|min|max|var|std",
"parallel=yes" ],
"dependencies": [ { "task":"Ensemble merging", "type": "single"
} ]
},
{
"name": "Ensemble reducing",
"operator": "oph_reduce2",
"arguments": [
"operation=@{index}",
"dim=new_dim"
],
"dependencies": [
{
"task": "loop_reduce",
"type": "single"
}
]
},
{
"name": "Ensemble export",
"operator": "oph_exportnc2",
"arguments": [
"force=yes",
"misc=yes",
"output_name=@{index}"
],

```

```

    "dependencies": [
    {
    "task": "Ensemble reducing",
    "type": "single"
    }
    ],
    {
    "name": "Ensemble Post",
    "operator": "oph_script",
    "arguments": [
    "script=precip_trend_analysis_ensemble",
    "args=@{index}"
    ],
    "dependencies": [
    {
    "task": "Ensemble export"
    }
    ],
    },
    {
    "name": "End loop_reduce",
    "operator": "oph_endfor",
    "arguments": [ ],
    "dependencies": [ { "task":"Ensemble Post" } ]
    },
    {
    "name": "Ensemble Delete all cubes",
    "operator": "oph_delete",
    "arguments": [
    "cube=[container=ensemble]"
    ],
    "dependencies": [
    {
    "task": "End loop_reduce"
    }
    ],
    "on_error": "skip"
    },
    {
    "name": "Ensemble Delete container",
    "operator": "oph_deletecontainer",
    "arguments": [
    "container=ensemble",
    "hidden=no",
    "delete_type=physical"
    ],
    "dependencies": [
    {
    "task": "Ensemble Delete all cubes"
    }
    ],
    "on_error": "skip"
    }
    ]
}

```

Annex II - PIT usage in the Ophidia Terminal (user manual)

This section provides a short user manual on how to use the three Ophidia Terminal extensions that have been implemented to include the RDA-PIT support into the Ophidia big data analytics framework.

Oph-Terminal provides different environment variables and commands that can be used to store recurring values and to perform “local” and “remote” operations.

The set of environment variables has been extended to meet the RDA-PIT requirements. Specifically, three new variables allow the terminal to contact the proper PIT Service and know the prefix of the Handle Server instance where the typed digital objects will be stored.

Here is a description of the main environment variables:

- OPH_USER is the username required to access Ophidia;
- OPH_PASSWD is the password required to access Ophidia;
- OPH_SERVER_HOST is the address/DNS name of the host where the Ophidia Server is running;
- OPH_SERVER_PORT is the port number of the Ophidia Server;
- **PIT_SERVER_HOST** is the address of the PIT REST Service;
- **PIT_SERVER_PORT** is the port of the PIT REST Service;
- **PIT_HANDLE_SERVER_PREFIX** is the prefix of the Handle Server instance.

The new environment variables are highlighted in bold.

The program environment has also been extended with three new commands - **pit_insert**, **pit_search**, **pit_lookup** - which enable a user to publish an experiment and to analyze experiments shared by other users.

pit_insert

The **pit_insert** command allows users to create and store a PTA typed digital object at the Handle Server side.

The command can be executed by means of a submission string with the following syntax:

pit_insert PIT_name_1=PIT_value_1;...;PIT_name_N=PIT_value_N

The above string consists of key-value pairs containing the equal sign and separated by semicolons. The values assigned to the parameters *PIT_name_1*, ..., *PIT_name_N* can include spaces and, in this case, they must be typed in quotes. The names of these parameters refer to the names of the PITs forming the PTA object defined in the module 2.

All the PITs are mandatory except the *date-time* type, which will be automatically set to the handle insertion date/time.

The newly created handle record will have a randomly and uniquely generated persistent identifier, which is the command output shown at the terminal.

pit_search

The **pit_search** command allows users to search for different PTA objects by some criteria.

The command can be executed by means of a submission string with the same syntax as the `pit_insert` string:

pit_search PIT_name_1=PIT_value_1;...;PIT_name_K=PIT_value_K

The above string consists of key-value pairs representing the user search filter. The command will show at the terminal the list of persistent identifiers related to the typed objects matching the user criteria.

pit_lookup

The `pit_lookup` command allows users to retrieve all the PITs values for the specified PTA object.

The user only has to provide the identifier of the handle to be resolved:

pit_lookupPID=identifier

The command will show at the terminal all the PITs forming the extracted object and their own corresponding values.

References

- [1] - Ophidia. Online: <http://ophidia.cmcc.it>. Last accessed: 21/11/2017.
- [2] - S. Fiore, A. D'Anca, C. Palazzo, I. T. Foster, D. N. Williams, and G. Aloisio. Ophidia: Toward big data analytics for science. In Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013, pages 2376–2385.
- [3] - D. Elia, S. Fiore, A. D'Anca, C. Palazzo, I. Foster, D. N. Williams, and G. Aloisio. An in-memory based framework for scientific data analytics. In Proceedings of the ACM International Conference on Computing Frontiers (CF '16), May 16-19, 2016, Como, Italy, pages 424-429.
- [4] INDIGO-DataCloud. Online: <https://www.indigo-datacloud.eu>. Last accessed: 21/11/2017
- [5] Davide Salomoni, Isabel Campos Plasencia, Luciano Gaido, Giacinto Donvito, P. Fuhrman, Jordi Marco, A. Lopez-Garcia, Pablo Orviz, Ignacio Blanquer, Germán Moltó, Marcin Plóciennik, Michal Owsiak, Michal Urbaniak, Marcus Hardt, Andrea Ceccanti, B. Wegh, J. Gomes, Mário David, Cristina Aiftimiei, L. Dutka, Sandro Fiore, Giovanni Aloisio, Roberto Barbera, Riccardo Bruno, Marco Fargetta, Emidio Giorgio, S. Reynaud, L. Schwarz: INDIGO-Datacloud: foundations and architectural description of a Platform as a Service oriented to scientific computing. CoRR abs/1603.09536 (2016)
- [6] S. Fiore, M. Plóciennik, C. Doutriaux, C. Palazzo, J. Boutte, T. Žok, D. Elia, M. Owsiak, A. D'Anca, Z. Shaheen, R. Bruno, M. Fargetta, M. Caballer, G. Moltó, I. Blanquer, R. Barbera, M. David, G. Donvito, D. N. Williams, V. Anantharaj, D. Salomoni, and G. Aloisio, "Distributed and cloud-based multi-model analytics experiments on large volumes of climate change data in the Earth System Grid Federation eco-system", Workshop on "Big Data Challenges, Research, and Technologies in the Earth and Planetary Sciences", IEEE Big Data Conference 2016, pp. 2911-2918, DOI: 10.1109/BigData.2016.7840941.
- [7] - RDA-PID Information Types 0.1 API. Online: <https://smw-rda.esc.rzg.mpg.de/apidocs/>
- [8] - Handle HTTP JSON REST API. In Technical Manual Version 8.1 Preliminary edition, CNRI, November 2015, pages 65-75.
- [9] - U. Schwardmann. Automated Schema Extraction for PID Information Types. Göttingen, 2016.