Project Acronym	***BlueBRIDGE***
Project Title	***Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth***
Project Number	***675680***
Deliverable Title	***BlueBRIDGE Resources Federation Facilities: Revised Version***
Deliverable No.	***D10.2***
Delivery Date	***January 2018***
Authors	***Leonardo Candela, Gianpaolo Coro, Paolo Fabriani, Luca Frosini, Nunzio Andrea Galante, Gabriele Giammatteo, Daniele Pavia, George Kakaletris, Lucio Lelii, Manuele Simi, Fabio Sinibaldi, Panagiota Koltsida***

BlueBRIDGE receives funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 675680

DOCUMENT INFORMATION

PROJECT	
Project Acronym	BlueBRIDGE
Project Title	Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth
Project Start	1st September 2015
Project Duration	30 months
Funding	H2020-EINFRA-2014-2015/H2020-EINFRA-2015-1
Grant Agreement No.	675680
DOCUMENT	
Deliverable No.	D10.2
Deliverable Title	BlueBRIDGE Resources Federation Facilities: Revised Version
Contractual Delivery Date	December 2017
Actual Delivery Date	March 2018
Author(s)	Leonardo Candela (CNR), Gianpaolo Coro (CNR), Paolo Fabriani (ENG), Luca Frosini (CNR), Nunzio Andrea Galante (ENG), Gabriele Giammatteo (ENG), Daniele Pavia (ENG) George Kakaletris (UOA), Lucio Lelii (CNR), Manuele Simi (CNR), Fabio Sinibaldi (CNR), Panagiota Koltsida (UOA)
Editor(s)	Panagiota Koltsida (UOA)
Reviewer(s)	Leonardo Candela (CNR)
Contributor(s)	n/a
Work Package No.	WP10
Work Package Title	Interfacing Infrastructures
Work Package Leader	ENG
Work Package Participants	CNR, ENG, UOA
Distribution	Public
Nature	Other
Version / Revision	1.0
Draft / Final	Final
Total No. Pages (including cover)	54
Keywords	gCube system, enabling services, computing infrastructures, cloud, data infrastructures, accounting, monitoring, external infrastructures.

DISCLAIMER

BlueBRIDGE (675680) is a Research and Innovation Action (RIA) co-funded by the European Commission under the Horizon 2020 research and innovation programme

The goal of BlueBRIDGE*, Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth*, is to support capacity building in interdisciplinary research communities actively involved in increasing the scientific knowledge of the marine environment, its living resources, and its economy with the aim of providing a better ground for informed advice to competent authorities and to enlarge the spectrum of growth opportunities as addressed by the Blue Growth societal challenge.

This document contains information on BlueBRIDGE core activities, findings and outcomes and it may also contain contributions from distinguished experts who contribute as BlueBRIDGE Board members. Any reference to content in this document should clearly indicate the authors, source, organisation and publication date.

The document has been produced with the funding of the European Commission. The content of this publication is the sole responsibility of the BlueBRIDGE Consortium and its experts, and it cannot be considered to reflect the views of the European Commission. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

The European Union (EU) was established in accordance with the Treaty on the European Union (Maastricht). There are currently 27 member states of the European Union. It is based on the European Communities and the member states' cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice, and the Court of Auditors (http://europa.eu.int/).

GLOSSARY

ABBREVIATION	DEFINITION
BlueBRIDGE	Building Research environments for fostering Innovation, Decision making, Governance and Education to support Blue growth
CPU	Central Processing Unit
EGI	European Grid Infrastructure
ETICS	e-Infrastructure for Testing, Integration and Configuration of Software
FHNManager	Federated Hosting Node Manager
GUI	Graphical User Interface
GWT	Google Web Toolkit
IAAS	Infrastructure As A Service
OCCI	Open Cloud Computing Interface
OGC	Open Geospatial Consortium
SDI	Spatial Data Infrastructure
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
VRE	Virtual Research Environment

TABLE OF CONTENT

TABLE OF FIGURES

DELIVERABLE SUMMARY

Deliverable D10.2 “BlueBRIDGE Resources Federation Facilities: Revised Version” is the revised version of the D10.1 deliverable, intended to report the various releases of the BlueBRIDGE facilities for infrastructures integration and federated resources management, including the latest developments and releases that took place between the 2 versions of the reports. It is of type “Other” and it is structured as follows: a description for each facility is available on this document in conjunction with a number of URLs, if needed, to the project’s wiki, providing more detailed descriptions and additional information for each component. This revised version of the document covers the whole period of the project, including the up to date information of the D10.1 deliverable and the new developed facilities aiming on providing a complete picture of the available federation facilities developed through the project’s lifetime.

In the period covered by the deliverable new developments took place and a set of new components are made available, contributing to the list of the available federation facilities. The list with the new components include: the SmartGears distribution via Docker container images, the gCube enabled services, the SDI service with the special workspace folder for SDI, the OAI-PMH service for external repositories integration and the CMEMS Dataset Importer. In addition, existing components have been enhanced and improved.

The deliverable is divided in three main sections following the structure of Work Package 10:

- T10.1 – facilities for integrating computing infrastructure facilities
- T10.2 – facilities for integrating data infrastructure resources
- T10.3 – facilities for federated resources management.

The deliverable provides, for each facility, a description, the documentation for developers and system administrators, how-to guides and usage instructions for different use cases and links to open source code and binaries.

EXECUTIVE SUMMARY

Deliverable D10.2 – "BlueBRIDGE Resources Federation Facilities: Revised Version" reports the release of the BlueBRIDGE facilities for integration of resources from external infrastructures and management of integrated resources. It is structured as follows: a description for each facility is available on this document in conjunction with a list of the main features, information about the design and architecture of it, a number of use cases and finally how it can be deployed. In case more information is needed or a more detailed documentation is available, URLs to the project's wiki are provided.

The set of the available facilities are split in three main categories: (a) facilities for integrating computing infrastructure resources, aiming on exploiting the computational power offered by external infrastructures in order to execute CPU intensive jobs, (b) facilities for integrating data infrastructure resources, where a number of components have been developed in order to integrate data from different data infrastructures to in the Virtual Research Environments (VREs), including biodiversity, geospatial and generic OAI-PMH repositories and (c) facilities for federated resources management, for effectively and efficiently managing the federated resources made available thanks to bridges developed in the previous categories.

The sections that follow provide the description and the details for each distinct facility, aiming on offering a complete guide to the intended readers.

The intended readers of this deliverable are (a) the community in the large willing to be informed on the solutions BlueBRIDGE offers for federated resources management, (b) the gCube developer community to know how to integrate, use and build on top of the facilities described by the document, and (c) the gCube administrators to know how to setup, configure and manage the integration of a gCube infrastructure with external resources.

1 INTRODUCTION

A gCube-based infrastructure is interconnected with several external resources for different reasons (e.g. datasets that needs to be imported in the infrastructure, algorithms that needs to be executed). The Work Package 10 aims to collect and analyse all the requirements related with the integration with external infrastructures and to address them with coherent, uniform and common solutions. This will allow to coordinate the effort and to have solutions better integrated in the gCube software.

The overall approach is to have a layer of specific software *bridges* (or *adapters*) that from one side are able to interact with the external resources and from the other side are compliant with gCube internal resource specifications. These components are responsible to wrap external resources into the gCube resource model and, therefore make them manageable from within gCube. On top of these bridges, a layer of common management and administrative tools will be responsible for the configuration, monitor, accounting and life-cycle management of the external resources.

Following this architecture, three categories have been identified to organize the activities in the Work Package:

- components to interface with computing infrastructures (typically used to deploy services or run algorithms externally) (cf. Sec.);
- components to interface with data infrastructures (typically to fetch or publish data) (cf. Sec.);
- components to control and monitor the external resources (cf. Sec.).

2 FACILITIES FOR INTEGRATING COMPUTING INFRASTRUCTURE RESOURCES

The integration of the gCube system with research/commercial cloud Infrastructure-As-A-Service (IaaS) infrastructures aims at adding versatility to the system by allowing programmatic and dynamic creation, configuration and decommissioning of cloud resources on external infrastructures. Common services and bridges between gCube and external IaaS providers are implemented in T10.1 (#698).

The aim of such facilities is to exploit the computational power offered by external infrastructures in order to execute CPU intensive jobs. During the reporting period, the activities in this task focused on the exploitation of the collaboration between BlueBRIDGE and the EGI-Engage project (H2020, grant n. 654142) to use the FedCloud and d4Science resources to offload DataMiner jobs.

2.1 SMARTGEARS DISTRIBUTION VIA DOCKER CONTAINER IMAGES

In order to improve interoperability among adopters of SmartGears and given the traction that container technology is gaining[1], this activity has led to the creation of a containerized SmartGears distribution. Docker was chosen as solution in order to adopt its Operating System (OS) virtualization technology based on Linux (the environment in which SmartGears has been developed and tested). Docker is possibly the most widespread technology in containerization, either when used directly or when leveraged by tools such as Google's Kubernetes. Virtualizing the SmartGears distribution with Docker is a way to make it more appealing and facilitate its adoption by parties interested in running SmartGears nodes. Since Docker seamlessly runs on heterogeneous platforms, it can be executed on any hosting OS. This makes it easier to deploy, for example, a development container locally on the developer's OSX/Linux/Windows workstation or a production container on on-premises production servers or on a cloud service such as Google Cloud Platform or Amazon Elastic Container Service.

2.1.1 KEY FEATURES

A Docker *image* is an immutable file that is essentially a snapshot of a deployment. Images are created with the build command, and they produce a *container* when started with the run command. Images are built starting from a *Dockerfile* and can be stored in a Docker registry such as Docker Hub for public distribution. The Dockerfile consists in a human readable text file where Docker directives are used to execute the commands necessary to build an image. Importantly, a Dockerfile can specify an image from which it starts to build atop by allowing the creation of hierarchies of images.

2.1.2 DESIGN

The *smartgears-base-image* is a Docker image built upon Ubuntu 14.04 LTS and incorporates a SmartGears distribution with all the required dependencies. Such dependencies mainly consist in either Oracle JDK 8 or OpenJDK 8 and Tomcat 7. The preferred Java Development Kit can be selected at image build time; in particular, OpenJDK is recommended when building an image that will be redistributed, in order to avoid potential licensing issues due to the Oracle JDK license restrictions[2].

[1] Containers: Real Adoption And Use Cases In 2017 White Paper

[2] http://www.oracle.com/technetwork/java/javase/readme-142177.html#redistribution

An SSH server (*sshd*) is configured and started at the creation of the container allowing remote connections and provisioning. While this approach is generally deemed unnecessary, since a user can attach to a running container from the host it is running on, an SSH connection allows remote orchestration/configuration management software such as Ansible to feed further web applications and/or configurations to the SmartGears instance. The user can optionally provision the public part of a private-public RSA/DSA encryption key at image build time to enable a passwordless connection to running container. This is obviously discouraged in case of a build that's going to be publicly distributed.

2.1.3 USE CASES

As previously explained in the introductory paragraph, a SmartGears container based on our image is the ready-to-run solution in every scenario where portability and rapid deployment are desirable or required, either for the instantiation of a single or multiple SmartGears instances or when leveraging container orchestration software and Container-as-a-Service platforms.

2.1.4 DEPLOYMENT

Here follows a concise summary of how to build and deploy a SmartGears image. For an in-depth guide please refer to the *SmartGears Docker Image* gcube wiki page[3].

2.1.4.1 BUILDING A SMARTGEARS DOCKER IMAGE

Given the extensive, public availability of roles to provision SmartGears applications available at CNR's Gitorious *ansible-playbooks*[4] repository, Ansible has been chosen as a mean to programmatically build the SmartGears Docker image. Ansible allows an automatic and reproducible build procedure that employs variables that get replaced at execution time. This enables, for example, the user to build an image with a specific SmartGears distribution version just by redefining or overriding an Ansible variable's default value.

In order to build a SmartGears Docker image with Ansible, an Ansible playbook is required. An Ansible playbook matches a target host running Docker with the related *smartgears-base-image* role (also hosted at CNR's Gitorious *ansible-playbooks* repository), while optionally providing one or more Ansible declaration files. An exemplifying Ansible playbook containing such logic is also available at the same Gitorious repository.

The *smartgears-base-image* role ships with a Dockerfile that contains all the steps required to build a valid image under the guise of an Ansible template. An Ansible template is a text file that optionally contains Ansible variables and conditionals in Jinja2 format. This approach gives a certain degree of freedom and permits the user to choose between executing Ansible or building the image directly using Docker - by leveraging the "docker build" command. Customizing the Dockerfile is a straightforward task given basic shell knowledge, making it easy for the tech savvy end user to amend the image that gets built by adding files, installing applications or making any other desirable customization.

[3] https://wiki.gcube-system.org/gcube/SmartGears_Docker_image

[4] https://gitorious.research-infrastructures.eu/infrastructure-management/ansible-playbooks

After an image has been built, creating another image that bundles SmartGears with one or more web applications can be achieved in two ways. The first one consists in writing a Dockerfile that builds a new image on top of the previously built *smartgears-base-image*. The new Dockerfile applies a number of user specified changes such as deploying a war file, creating a user, manipulating configuration files, adding an ssh key, etcetera. This is the most accessible method and it is the suggested way to create new SmartGears-derived images. An exemplifying Ansible role that includes the relative Dockerfile is available at CNR's Gitorious repository[5]. This Ansible role bundles SmartGears with a Home Library war file downloaded from BlueBRIDGE's Maven repository

The second method can be used only if an ssh key has been provisioned at build time. In this case, the user may connect via ssh to a running container. Using ssh to connect to a live container gives the user the freedom of acting upon the system as he would on a physical system or a virtual machine. In order to save the changes made on a live container the user needs to commit those changes on a new Docker image. Either way, the resulting image can then be used as source of one or more container instances or pushed to a Docker repository. When making changes via ssh to an image that will become public, removing any ssh key previously stored inside the container instance before committing is strongly recommended.

2.1.4.2 RUNNING A SMARTGEARS DOCKER IMAGE

There are two ways to spin an instance of the SmartGears Docker image: by executing the *ansible-container-bootstrap* Ansible role available at CNR's Gitorious repository[6] or by running it with the "*docker run*" command.

The *smartgears-container-bootstrap* Ansible role that automates the execution of a *smartgers-base-image* is particularly useful when combined with other roles that actually build and configure the container as an all-in-one, build-and-run approach.

If the user wants to manually run a SmartGears Docker image, a new container can be instantiated by using the *"docker run"* command. When instantiating a new SmartGears Docker image, several SmartGears parameters may be passed to configure the SmartGears instance before execution. Here's a quick reference of these parameters:

- CONTAINER_TOKENS: all the tokens required to join a specific infrastructure/VO/VRE, comma separated. Please refer to the relative LifeRay Portal procedures to know more on token generation. This parameter is mandatory.
- CONTAINER_MODE: valid values are either online or offline.
- CONTAINER_HOSTNAME: the hostname that the SmartGears instance will use. As an authorization system requirement, this needs to be the hostname used to generate the token(s).
- CONTAINER_PORT: the port that gets bound by the SmartGears instance while running.
- CONTAINER_INFRASTRUCTURE: the infrastructure that the SmartGears instance will be joining.

[5] https://gitorious.research-infrastructures.eu/infrastructure-management/ansible-playbooks/trees/master/library/roles/smartgears/smartgears-container-home-library

[6] https://gitorious.research-infrastructures.eu/infrastructure-management/ansible-playbooks/trees/master/library/roles/smartgears/smartgears-container-bootstrap

- PATCH_COMMON_SCOPES: Boolean value. Required for development purposes when running the node inside an infrastructure that is out of the distributed common-scope-maps.jar, a jar file distributed with SmartGears that contains a list of official infrastructure scopes. The user can create a new common-scope-maps.jar that contains custom endpoints and replace the one shipped with the *smartgears-base-image* Ansible role, placed inside the *files* directory. In order to apply such a patch, this parameter must be set to True. Default to "0".
- PATCH_COMMON_AUTHORIZATION: Boolean value. Like the PATCH_COMMON_SCOPES parameter, this parameter may be required for development purposes when running the node with an authorization endpoint that is not included with endpoints specified in the common-authorization.jar, a jar file distributed with SmartGears that contains an official list of authorization service endpoints. The user may create a new common-scope-maps.jar that contains custom endpoints and replace the jar file shipped with the smartgears-base-image Ansible role, placed inside the *files* directory. In order to apply such a patch, this parameter must be set to True. Default to "0".

As a final step, here is an example that illustrates how to execute a smartgears-home-library Docker container based off a smartgears-base-image:

docker run -t -e CONTAINER_TOKENS=token1,token2,token3 -e CONTAINER_MODE=online -e CONTAINER_HOSTNAME=myhostname -e CONTAINER_PORT=8080 -e CONTAINER_INFRASTRUCTURE=d4science smartgears-home-library:1.10.2-4.9.0-160502

For instructions on how to run a Docker image via orchestration tools or cloud services (Kubernetes, Docker swarm, Openshift, Openstack), please refer to the respective official documentation. For more detailed information on how to build, run and execute a SmartGears Docker image, please refer to the following official GCube wiki link[7]

2.2 MANAGEMENT OF FEDERATED GCUBE HOSTING NODES (FHNMANAGER)

This is a new facility entirely developed in the BlueBRIDGE project to address the project requirement of bridging with external computing infrastructures. It will be further developed during the project with the release of new functionalities.

A first step toward the integration of external IaaS providers is the FEDERATED HOSTING NODE MANAGER service. This service is capable to fulfil requests for creation, configuration, start, stop and removal of gCube Hosting nodes or SmartGears nodes equipped with pre-installed gCube services.

It has been implemented a specific connector to allow the FHNManager service to interact with the EGI FedCloud and d4Science infrastructures. The connector integrates with and exploits the standard OCCI interface and the FedCloud/d4Science authentication, o corresponding authorization system (based on the VOMS technology) and the OCCOPUS orchestrator (a service that enable the deploy and the elastic management of complex services).

[7] https://wiki.gcube-system.org/gcube/SmartGears_Docker_image

A mechanism to install and configure the Smart Executor service on the resources created via the FHNManager has been provided. This allows the execution of Dataminer jobs on the external resources.

2.2.1 KEY FEATURES

Registration and management of cloud providers

- Access credentials and cloud resources templates on a per-VRE basis:
- Credentials securely stored in the infrastructures

Cloud resources lifecycle management

- Creation, configuration and decommissioning of cloud resources;
- The integration facility allows to use templates for the configuration of cloud resources (e.g. CPU and memory characteristics);

Accountability and monitorability

- Accounting and monitoring data on usage and workload of resources created on the cloud are collected and made available in the infrastructure;
- Compliance with D4Science infrastructure policies and for the automatic scale-in/out;

User Interface

- Integrated in the VRE administration portal since it already contains all the other VRE administration tools;

Modular Architecture

- A plug-in based architecture for an easy integration of new cloud providers;

2.2.2 DESIGN

2.2.2.1 ARCHITECTURE

The architecture highlights a number of integration-specific components and the interaction among them as well as with existing components, either belonging to the D4Science infrastructure or to the EGI infrastructure. They are described in the following:

- **Cloud Libraries**: Third-party software providing language-specific APIs and data model to easily interact with clouds from within applications. Although they usually support a number of different clouds and cloud standards, there is no universal coverage for any of them; nor the API and data model they expose is uniform across libraries.
- **Connectors**: They are built on top of cloud libraries in order to abstract the specifics of their APIs and data models and expose a uniform interface to the upper layers. Connectors do not interact with any other service nor are expected to persist any information.

- **Federated Hosting Node Manager (FHNM)**: is the core part of the integration and is the place where all the business logic resides. It's the gateway for all the operations related to the management of external cloud infrastructures, via the most-appropriate connector/library; it manages connectors to the available clouds; it gathers accounting and status data and publishes them to the specific D4Science services.

- **Federated Hosting Node Manager Portlet**: Such portlet provides infrastructure and VRE administrators with a dashboard and a control panel to easily monitor and manage resources in external infrastructures. It enables administrators to register cloud infrastructures and credentials associated with them as well as virtual appliances and service profiles. D4Science Information System: Collects, holds and provides all the information related to the D4Science infrastructure. In particular, for the purpose of the integration of FedCloud/D4Science, it holds the list of available cloud sites along with credentials to access them, the list of running cloud resources and their status and the list of virtual appliances available for instantiation.

- **D4Science Monitoring system**: Data collected is analysed to produce alarms and/or take countermeasures in case of problems with one or more resources.

- The **Virtual Organization Membership Service (VOMS)**: VOMS manages the authorization attributes to be embedded in X.509 proxy certificates, needed to access FedCloud/D4Science sites.

- **EGI sites**: Host cloud resources instantiated through the components. The list of sites is available at https://appdb.egi.eu/browse/sites/cloud

- **AppDB**[8] repository maintains the overall set of available virtual appliances (such as gCube Smart Executor and DataMiner).

The solution has been developed by means of Java since it is the same technology of the gCube system and it made easier to integrate in the D4Science infrastructure and furthermore, a high availability for third-party libraries implementing cloud standards is available. A brief description of the service is visible in the following subsections.

[8] https://appdb.egi.eu/

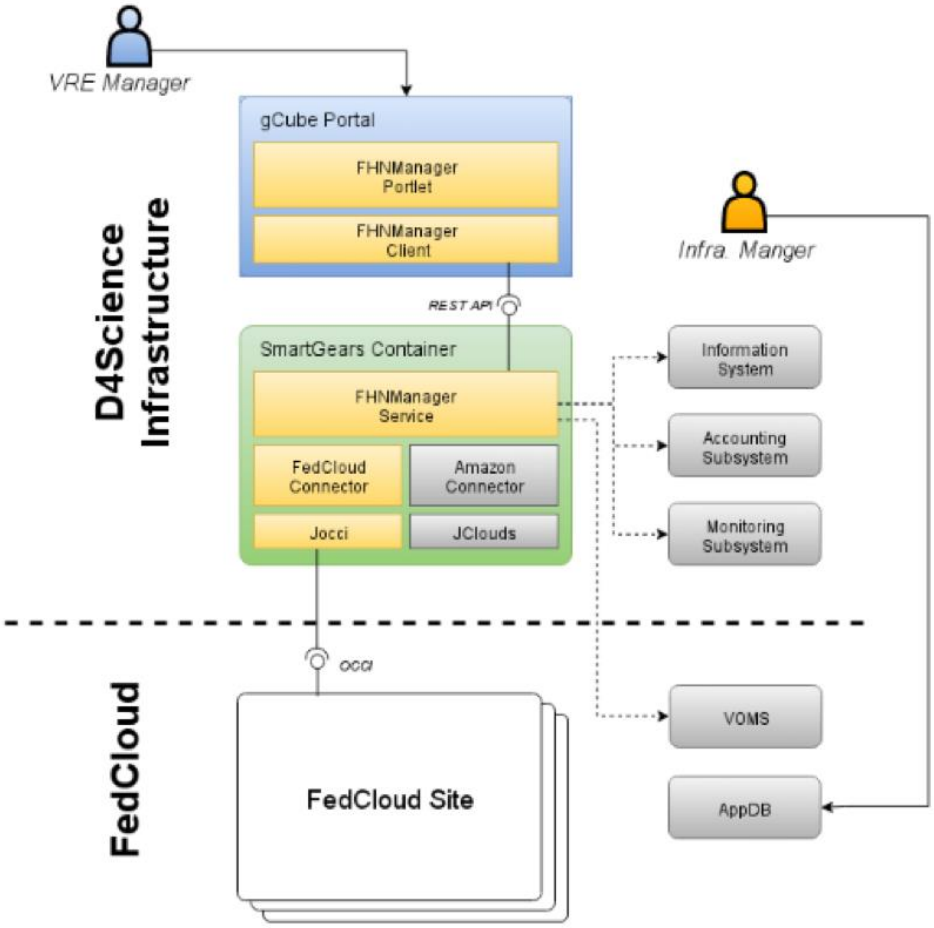


Figure 1. FHNManager Service: Overall View

2.2.2.2 DATA MODEL

The data model has been designed with the aim of representing the information related to cloud providers, nodes, templates, accounting and monitoring in a homogenous way independently from the cloud provider exposed API and technology. Connectors and third-party Cloud Libraries ensured a proper translation between the internal data model and the specific cloud provider model.

The main classes contained in the data model are related to:

- **VMProvider**: describing the site in terms of endpoint and credentials
- **NodeTemplate**: describing the kind of image to deploy (e.g., SmartExecutor, DataMiner)
- **ResourceTemplate**: describing the resource in terms of sizing and hardware characteristics (e.g., memory, core)
- **Node**: describing the virtual machine that have been created by means of the service

A detailed class diagram of the adopted data model is shown in the following picture.

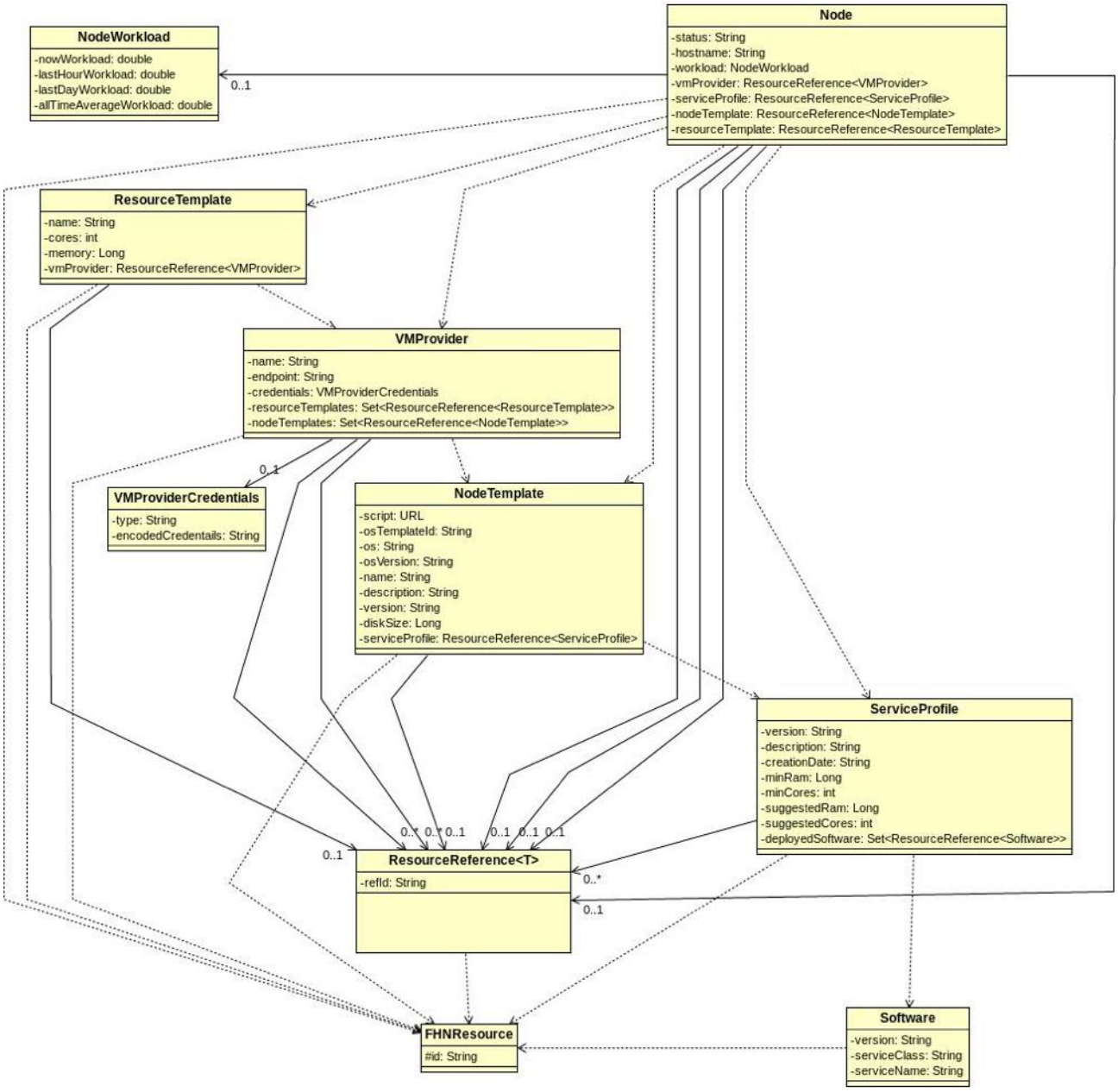


Figure 2. FHNManager Service: Data Model

2.2.2.3 FHNMANAGER SERVICE

The FHNManager is a web service implemented in Java and running in a gCube SmartGears container. It exposes a REST API that allows to access data on nodes and cloud providers as well as to execute operations to create, destroy, start and stop nodes.

In order to account the usage of cloud provider resources by the D4Science infrastructure, the service relies on the gCube accounting framework provided by the infrastructure: new usage records have been defined to track changes in the lifecycle of the resources on the cloud. Through the connectors, the service also supports the fetching of accounting data from the cloud provider (when available).

Concerning the monitoring data, the service implements the support to fetch it from the cloud provider (if available) as well as collect data directly from the resource via the gCube monitoring framework.

The current implementation of the service - and thus the portlet - does not support the registration of new cloud providers along with their associated credentials. However, since the registration entries for cloud providers are maintained in the D4Science Information System, VRE/Infrastructure managers can still use the editing functionalities provided by the Information system itself, although the editing capabilities provided are not tailored to the specific task.

Since the FHNManager is deployed in a SmartGears container, it is automatically registered to the D4Science infrastructure and it is automatically authorized to exploit infrastructure capabilities (Information, Accounting and Monitoring). In addition to properly generate and use X.509 proxy certificates, the host must also be equipped with VOMS clients (i.e. voms-proxy-init) and configured with trusted root certification authorities as distributed by EUGridPMA.

2.2.2.4 CONNECTORS

Connectors are built on top of cloud libraries (e.g., jOCCI) in order to abstract the specifics of their APIs and data models and expose a uniform interface to the upper layers. Connectors do not interact with any other service nor are expected to persist any information.

A library has been developed in order to allow the integration of gCube with EGI FedCloud/d4Science infrastructure. In particular, the usage of such library will facilitate the interaction with the sites exposing OCCI just by manipulating APIs.

Further information about the OCCI Connector Library can be found in the dedicated gCube Wiki page:

https://wiki.gcube-system.org/gcube/OCCI_library

2.2.2.5 THIRD-PARTY DEPENDENCIES

The FHNManager depends on the following third-party libraries and frameworks:

- **jOCCI**: java client library for clouds exposing the OCCI interface released under Apache License Version 2.0;
- **Jersey**: java framework to to realize REST APIs licensed as CDDL Version 1.1 and GPL Version 2;
- **Liferay 6.0 CE**: Enterprise portal framework released under LGPL 2.1;
- **GWT**: Java framework for portlet developing released under Apache License Version 2.0

2.2.2.6 EGI FEDCLOUD/D4SCIENCE MEMBERSHIP

D4Science-FedCloud integration is being tested in the “fedcloud.egi.eu” VO. Production-level operations have been moved to the “d4science.org” VO, registered on the “vomsmania.cnaf.infn.it” VOMS server and in production since December 2015.

2.2.3 USE CASES

It is possible to identify two main scenarios:

- the VRE Manager decides the number of SmartExecutor nodes needed in the VRE. Through the VRE administration portal, the VRE Manager chooses (from a list of templates) to create the required number of SmartExecutors. She/he chooses on which infrastructure the resources should be created and their characteristics in terms of computation and/or storage capacity. The gCube system, via the integration facility here described, will automatically create the new nodes, register them to the D4Science infrastructure and make them available in the VRE.

- the gCube system intelligently adapts the number of SmartExecutor nodes to the actual demand. In the VRE administration portal, the VRE Manager decides the minimum and maximum number of SmartExecutor expected for the VRE. Then, the gCube system via the integration facility here described, automatically scales-up/down the pool of SmartExecutors (by creating/destroying nodes) predicting the demand for SmartExecutors by analysing VRE accounting and monitoring data.

2.2.4 DEPLOYMENT

An administration guide to the service installation, including Deployment, Usage, Configuration steps and Authentication information, is available at:

https://wiki.gcube-system.org/gcube/FHNManager_Installation.

3 FACILITIES FOR INTEGRATING DATA INFRASTRUCTURE RESOURCES

A gCube infrastructure is capable of interfacing with several different data infrastructures to make their data available in the Virtual Research Environments (VREs). The software developed in task T10.2 aims to create common data model, approach and, where possible, access method to these infrastructures. Software for accessing Biodiversity and Geospatial data has been developed. They include a common abstract model for each type of data, services to retrieve data from the external infrastructures, libraries and tools to manage the data.

3.1 BIODIVERSITY DATA ACCESS

Biodiversity facilities were already part of gCube before BlueBRIDGE project started. However, in the context of BlueBRIDGE project some technology updates have been released.

Being part of the Data Access and Storage Facilities[9], Biodiversity Data Access components focus on uniform access to sources of biodiversity data of arbitrary location and size.

The Species Product Discovery service is the broker service for discovering data from several external data sources (e.g. OBIS, GBIF, Worms, ITIS). The support for the data sources is implemented as PLUGINS of the broker service.

The goal of this subsystem is to offer a uniform access over different biodiversity repositories through a simple API.

The service has a dynamically extensible architecture, i.e. rely on independently developed plugins to adapt their APIs to a variety of back-ends within or outside the system.

When connected to remote data sources, the services may be widely replicated and their replicas know how to leverage the Enabling Services to scale horizontally to the capacity of the remote back-ends.

The query language and the data/object model are documented respectively at:

- **https://wiki.gcube-system.org/gcube/SPQL:_SPecies_Query_Language**
- **https://wiki.gcube-system.org/gcube/Species_Products_Discovery_Objects**

3.1.1 KEY FEATURES

- access API tailored to biodiversity data sources[10]
- unifying object model for Species Products[11]
- dynamically pluggable architecture of transformations (e.g. unified classification, synonyms presentation) from external sources of biodiversity data
- plugins for key biodiversity data sources, including OBIS, GBIF and Catalogue of Life

[9] https://wiki.gcube-system.org/gcube/Data_Access_and_Storage_Facilities

[10] https://wiki.gcube-system.org/gcube/Data_Access_and_Storage_APIs

[11] https://wiki.gcube-system.org/gcube/Species_Products_Discovery_Objects

- dynamic clustering of discovered items
- integration with workspace facilities
- export of discovered items in multiple formats, including DarwinCore and DarwinCore Archive
- flexible query language

3.1.2 DESIGN

Uniform access to sources of biodiversity data that expose different capabilities and different data models is a key requirement to support biodiversity studies.

This subsystem offers the possibility to retrieve, manage and elaborate biodiversity data under a single model, with a domain-specific API, and regardless of its source of origin.

3.1.2.1 ARCHITECTURE

The subsystem comprises the following components:

- **species-products-discovery-service**: a stateless Web Service that exposes read operations and implements it by delegation to dynamically deployable plugins for target repository sources within and outside the system;
- **species-products-discovery-library**: a client library that implements a high-level facade to the remote APIs of the Species manager service;
- **obis-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **OBIS**[12] data source;
- **gbif-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **GBIF**[13] data source;
- **catalogueoflife-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **Catalogue of Life**[14] data source;
- **flora-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **Brazilian Flora**[15] data source.
- **worms-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **Worms**[16] data source.

[12] http://www.iobis.org/

[13] https://www.gbif.org/

[14] http://www.catalogueoflife.org/

[15] http://floradobrasil.jbrj.gov.br/

[16] http://www.marinespecies.org/

- **wordss-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **World Register of Deep Sea Species**[17] data source.
- **species-link-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **speciesLink**[18] data source, by using the international standard TAPIR protocol.
- **itis-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **ITIS**[19] data source.
- **ncbi-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **NCBI**[20] data source.
- **irmng-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **IRMNG**[21] data source, through Darwin Core Archive format.
- **asfis-spd-plugin**: a plugin of the Species Products Discovery service that interacts with **ASFIS**[22] data source.

A diagram of the relationships between these components is reported in the following figure:

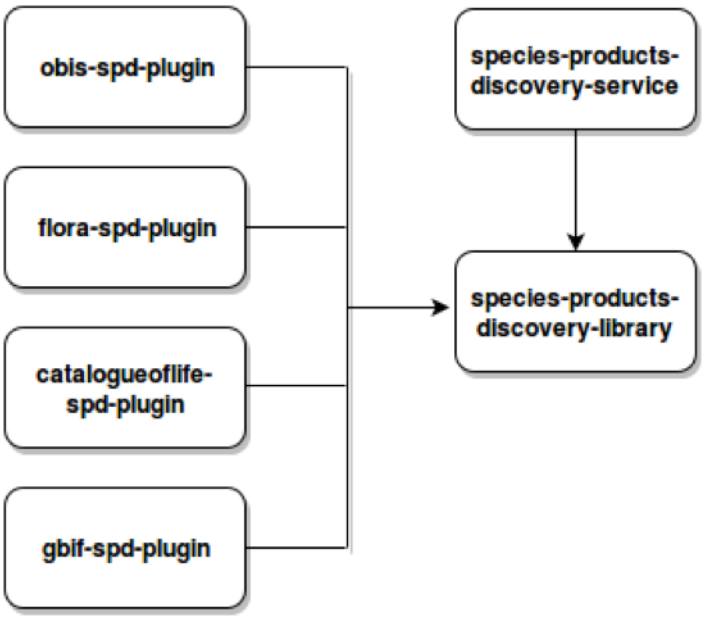


Figure 3. Biodiversity Data Access Subsystem Architecture

[17] http://www.marinespecies.org/deepsea/

[18] http://splink.cria.org.br/

[19] https://www.itis.gov/

[20] https://www.ncbi.nlm.nih.gov/

[21] http://www.cmar.csiro.au/datacentre/irmng/

[22] http://www.fao.org/fishery/collection/asfis/en

3.1.3 USE CASES

The subsystem is particularly suited to support abstraction over biodiversity data. Every biodiversity repository can be easily integrated in this subsystem developing a plugin.

The development of any plugin of the Species Manager services immediately extends the ability of the systems to discovery new biodiversity data.

3.1.4 DEPLOYMENT

All the components of the subsystem must be deployed together in a single node. This subsystem can be replicated in multiple hosts; this does not guarantee a performance improvement because the scalability of this system depends on the capacity of external repositories contacted by the plugins. There are no temporal constraints on the co-deployment of services and plugins. Every plugin must be deployed on every instance of the service.

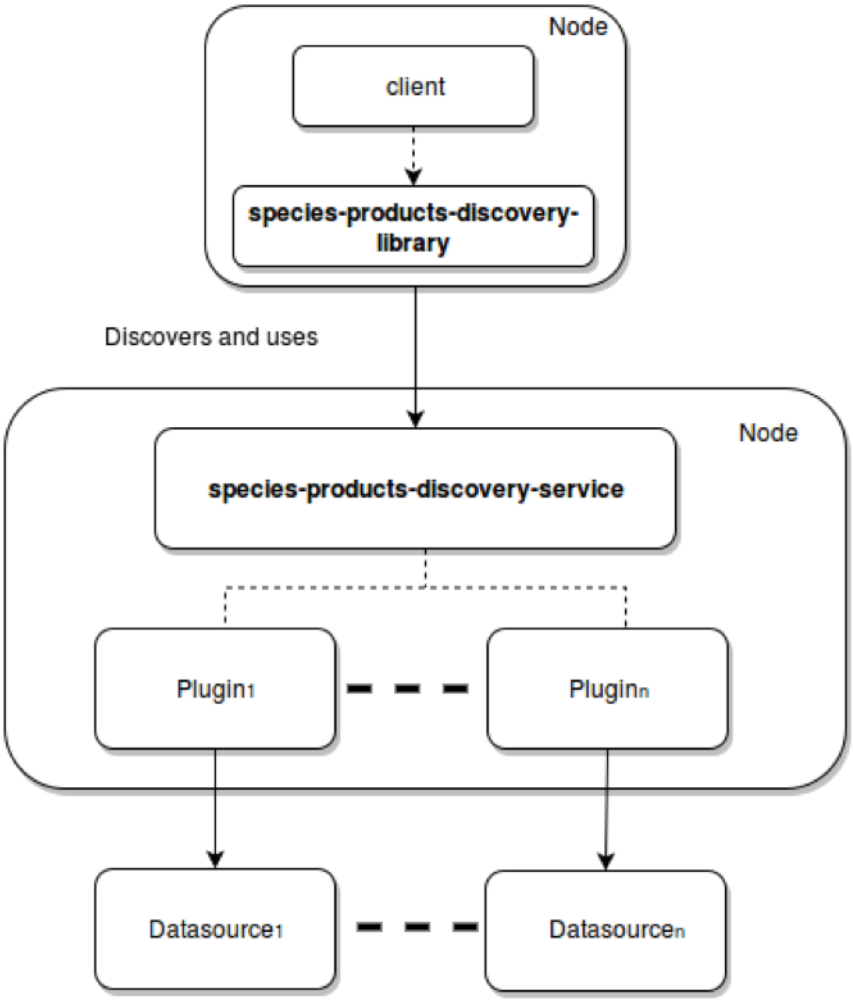


Figure 4. Biodiversity Data Access Deployment

3.2 SPATIAL DATA INFRASTRUCTURE FACILITIES

Geospatial facilities were already part of gCube before BlueBRIDGE project started. However, in the context of BlueBRIDGE project major technology updates have been released.

SDI Facilities are implemented relying on state-of-art technologies and OGC standards, offering:

- Spatial Data discovery: A catalogue service relying on CSW-based service (namely GeoNetwork);

- Spatial Data Storage and Access: A federation of repositories based on GeoServer and THREDDS technologies;
- Processing: A cluster of OGC-WPS services relying on gCube platform for data analytics;
- Visualization: A set of web applications for browsing and visualization of spatial data;
- SDI Features management: A dedicated REST gCube service for management and monitoring of SDI capabilities.

Third party services such as GeoNetwork, Geoserver and THREDDS are provisioned as gCube-enabled services, integrating gCube SmartGears capabilities.

Geospatial Data Access components focus on providing a uniform access to geo-spatial information, fundamental for performing data analysis on species behaviour and preferences. It includes functionalities for retrieving environmental information associated to some areas, in terms of physical and chemical properties.

Main components are:

- GCUBE-Enabled services: third-party geospatial technologies interacting with gCube technologies;
- GIS-Interface: java library for geospatial dataset and metadata management;
- SDI-Service: main control point for the entire SDI.

3.2.1 SPATIAL DATA DISCOVERY AND ACCESS

Discovering geo-spatial information about some points of the ocean can be fundamental for performing data analysis on species behaviour and preferences. Furthermore, it can be useful to scientists belonging to different communities who want to share their data or to have a complete vision of the environmental setup of some zones. Geospatial Data Discovery includes functionalities for retrieving environmental information associated to some areas, in terms of physical and chemical properties.
In summary, the Geospatial Data Discovery system provides *distributed, searchable or to be generated, environmental features associated to geographical points or areas*.

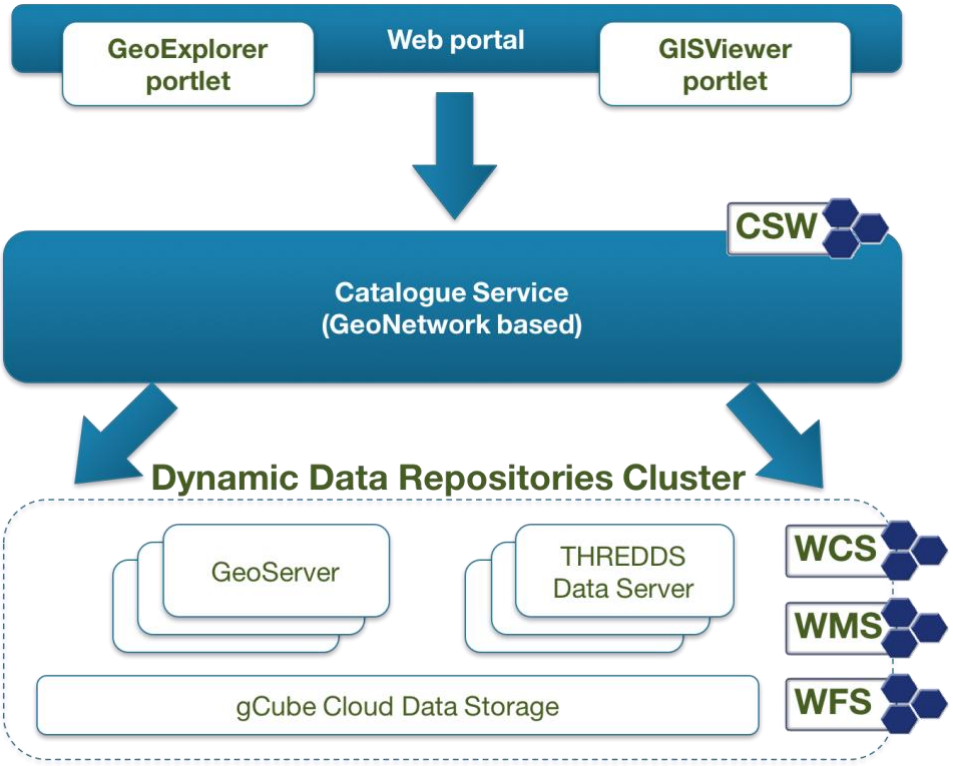


Figure 5. Spatial Data Discovery Architecture

3.2.2 KEY FEATURES

Uniform model and access API over environmental data

- heterogeneous external datasources investigation;
- uniform access to OGC compliant services.

Pluggable external sources

- a plug-in based architecture allows for an easy attachment of new datasources.

Data consumption with OGC standard protocols

- it is possible to produce data under standard protocols like WFS, WCS, WMS etc.;
- it is possible to perform processing requests by means of standard protocols, like WPS.

3.2.3 SUBSYSTEMS

Geospatial Data Discovery and Access comprises the following component:

- GCUBE-Enabled Services (https://gcube.wiki.gcube-system.org/gcube/GCube-Enabled_geo-services): Third-party geospatial technologies able to interact with gCube features;
- SDI-Service (https://gcube.wiki.gcube-system.org/gcube/SDI-Service): REST Web service acting as a main access point for the SDI;
- GIS-Interface (https://gcube.wiki.gcube-system.org/gcube/GIS_Interface): A java library allowing the interaction with SDI's Catalogue and Storage;

- DataTransfer 2 framework (https://gcube.wiki.gcube-system.org/gcube/Data_Transfer_2): REST API towards GCUBE-Enabled services file system with extensible ad-hoc logic (Please refer to 4.6 **Data Transfer 2**);
- Workspace special folder for SDI (https://wiki.gcube-system.org/gcube/Workspace): A web-based application that allows users to access and manage their dedicated storage which can be synchronized to SDI Storage;
- GeoExplorer (https://wiki.gcube-system.org/gcube/Geo_Explorer): A web-based application that allows users to navigate, organize, search and discovery internal or external layers;
- GISViewer (https://gcube.wiki.gcube-system.org/gcube/GIS_Viewer): a web-based application to interactive explore, manipulate and analyze spatial datasets.

3.2.3.1 GCUBE-ENABLED SERVICES

GCUBE-Enabled Services are third-party services capable of exploiting the gCube authorization framework. While dealing with these services, authentication and authorization of http(s) requests can be obtained by specifying a gcube-token just like any other gCube Service, relieving the user from dealing with:

- Non-standard authentication APIs
- Specific instance credentials use

DESIGN

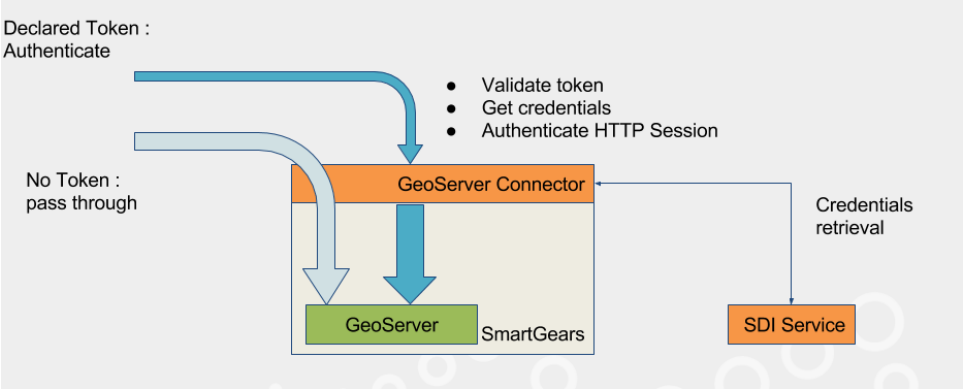


Figure 6. gCube-Enabled SDI Services

This solution is obtained by providing third-party services installations (such as Thredds, GeoServer) with an ad-hoc request interceptor which deals with authorization. Such components are called connector and rely on the SDI-Service in order to translate requests' gcube-token to actual service instance credentials.

gCube-Enabled Services are provided with a fully equipped SmartGears' distribution, which offers:

- Authorization framework (security policies support)
- Accounting framework (usage monitoring)
- DataTransfer2 framework (read / write controlled interface to file system)

The presence of Data Transfer 2 framework along with GCUBE-Enabled services, allows the implementation of advanced ad-hoc logic useful to interact with the service itself. Please refer to 4.6 **Data Transfer 2.**

GCUBE-Enabled services are designed in order to best address the following use cases:

- Direct interaction with third-party services REST API and exposed OGC Interfaces
- Embedding third-party services GUI in a gCube Portal

3.2.3.2 SDI-SERVICE

The SDI-Service is a REST gCube service that acts as main control point for the entire SDI. It handles configuration of geo-services, while providing simple APIs to interact with it. This document outlines the design rationale, key features, and high-level architecture, the options for their deployment and as well some use cases.

DESIGN

SDI facilities are prominent features in data infrastructures, and they must serve heterogeneous needs. In order to keep control of the SDI over the infrastructure, the SDI-Service acts as a central point of interaction for applications. The choice of REST API has been made in order promote its interoperability across applications implementation.

The picture below describes the overall architecture of gCube SDI and the interaction between its components. The SDI-service manages all instances across contexts, allowing for a better control of policies, and acting as gateway for SDI configuration retrieval.

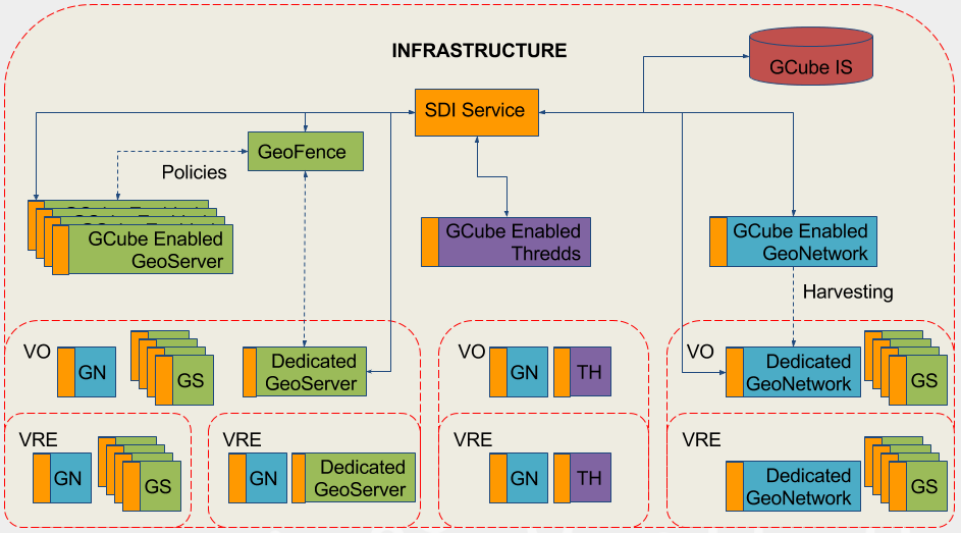


Figure 7. SDI Service Overall Architecture

KEY FEATURES

Key Features offered by SDI-Service are:

- SDI Configuration management
  - APIs to easily query the IS and the SDI's geo-services configuration details
  - APIs to configure geo-services through the infrastructure contexts
  - APIs to monitor SDI status
- GeoSpatial datasets management
  - APIs to publish both data and metadata into the SDI
  - APIs to manage SDI collections and access policies

USE CASES

SDI Service is designed in order to best address the following use cases:

- Retrieve endpoint and credentials of SDI geo-services
- Configure geo-services on a per-context base
- Publish/update geospatial metadata

3.2.3.3 GIS-INTERFACE

GIS-Interface is a java library which exposes methods to access / modify and publish spatial data and related metadata on a Spatial Data Infrastructure (SDI). The library relies on the java libraries geoserver-manager and geonetwork-manager provided by GeoSolutions (vendor of GeoNetwork and GeoServer) in order to interact with REST APIs offered by GIS Services.

GIS-Interface offers advanced APIs to manage both data and metadata in atomic operations, allowing also to directly exploit APIs offered by GeoSolutions libraries.

USE CASES

The library is designed in order to best address the following use cases:

- Publish/modify/delete geospatial layers
- Search layers in the SDI catalogue

3.2.3.4 WORKSPACE SPECIAL FOLDER FOR SDI

A workspace represents a collaborative area where users can exchange and organize information objects (*items*) according to their specific needs. Every user of any Virtual Research Environment is provided with this area for the exchange of workspace objects with other users. Such an area is further organized in *folders* as to resemble a classic folder-based file system.

The added value of this collaborative area is represented by the item types it can manage in a seamless way. They range from binary files to compound information objects representing tabular data, species distribution maps, time series. Every item in the workspace is equipped with a rich metadata including bibliographic information like title and creator as well as lineage data.

For what concerns Spatial Data Access and Storage, the workspace offers the capability to synchronize a folder to the Storage Area of THREDDS Catalogs, in order to manage it. This allows users to directly store, delete, update geospatial datasets and metadata into THREDDS Catalogues and GeoNetwork.

KEY FEATURES

Workspace Synchronization allows users to

Manage Thredds Catalogues content

- Inspect datasets and related metadata
- Update content by uploading/removing files from the Workspace

Manage Thredds Catalogues configuration

- Inspect catalogue configuration file
- Modify catalogue configuration file

Generate Thredds Dataset ISO Metadata

- Automatically extract and publish metadata from synchronized datasets

DESIGN

Workspace synchronization exploits gCube-Enabled Technology (in particular the DataTransfer 2 framework) in order to Create, Read, Update, Delete datasets from THREDDS storage area.

It also exploits DataTransfer 2 THREDDS Plugins (see 4.6 DataTransfer 2) in order to perform more complex tasks such as THREDDS catalogs configuration and ISO metadata extraction from synchronized datasets.

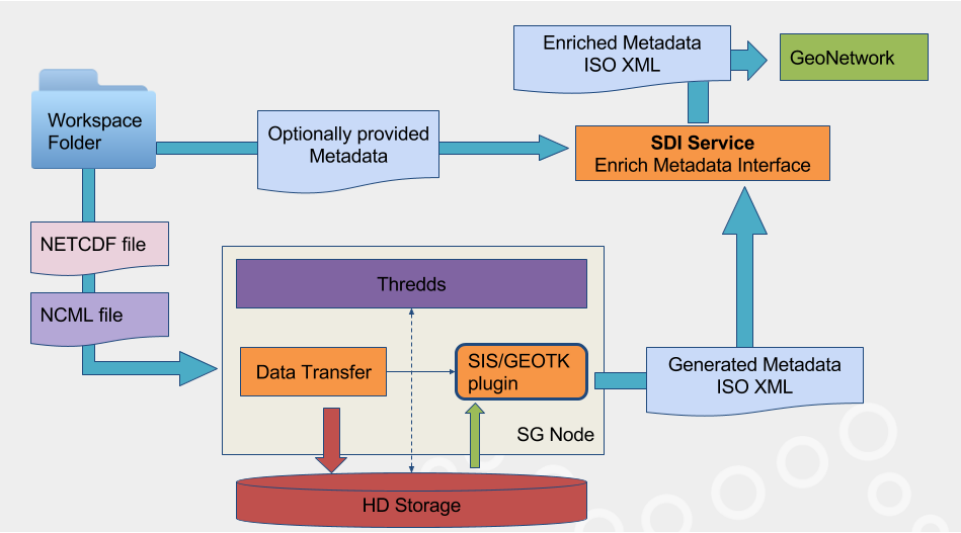


Figure 8. Workspace Special Folder for SDI

3.2.4 SPATIAL DATA VISUALIZATION

Geospatial data, that is tabular data with associated geographical indications, need data visualization as crucial function. This allows scientists to investigate on produced distribution probability maps, to analyse the results of some geospatial processing or to retrieve graphically the distribution of environmental features. The number of elements making up a distributed storage system for geographical layers can be difficult to setup, along with a visualization utility for that. The Geospatial Data Visualization is a set of utilities for retrieving and visualizing information stored on a distributed geospatial layers network, relying on a GeoNetwork implementation.

Geospatial Data Visualization provides the following facilities:

- the ability to retrieve layers from a distributed network based on the GeoNetwork software;
- the ability to visualize layers coming from different kinds of data sources;
- the ability to add, modify and investigate those layers from a graphical interface.
-

The entire system is helped by the following technology:

- Geo Network: a software for realizing a distributed network of WMS or WFS layers;
- Geo Server: a software allowing users to share and edit geospatial data.

In summary, the Geospatial Data Visualization system provides a unique access for visualizing, retrieving and analysing distributed geospatial data.

3.2.4.1 KEY FEATURES

Uniform access over geospatial GIS layers

- investigation over layers indexed by GeoNetwork;
- visualization of distributed layers;
- addition of remote layers published in standard OGC formats (WMS or WFS).

Filtering and analysis capabilities

- possibility to perform CQL filters on layers;
- possibility to trace transect charts;
- possibility to select areas for investigating on environmental features.

Search and indexing capabilities

- possibility to sort over titles on a huge quantity of layers;
- possibility to search over titles and names on a huge quantity of layers;
- possibility to index layers by invoking GeoNetwork functionalities.

3.2.4.2 GEOEXPLORER

GeoExplorer is a web application that allows users to navigate, organize, search and discover internal or external geospatial data layers. This document outlines the design rationale, key features, and high-level architecture, as well as the deployment context.

The main objective of this application is to offer centralized environment for access to the entire spatial data in a certain scope. By external visual components, as GisViewer[23] it's also possible to explore and analyze a personal set of selected resources.

The service is able to interface to other infrastructural services in order to expand the number of functionalities and applications to the data under analysis.

DESIGN

GeoExplorer is based on Visual Information-Seeking Mantra[24], which is an approach to summarize the best user interactions in order to get some information from a visual application: Overview first, Zoom and filter, Details on demand.

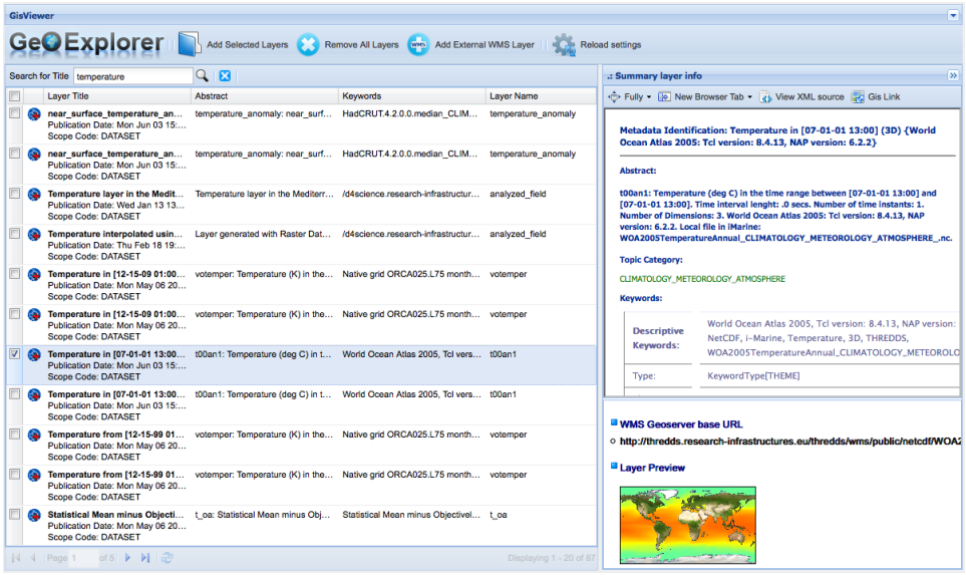
The application uses a cloud network of geospatial resources called GeoNetwork[25], these resources are typically GeoServer[26] Layers. It's still possible to get external layers by their WMS[27] representation.

[23] https://wiki.gcube-system.org/gcube/GIS_Viewer

[24] http://www.infovis-wiki.net/index.php/Visual_Information-Seeking_Mantra

[25] https://geonetwork-opensource.org/

[26] http://geoserver.org/



ARCHITECTURE

The subsystem comprises the following components:

- **GeoExplorer Central Controller**: manage all the internal components and their interactions;
- **Layers Grid Manager**: a set of objects and widgets that allows layers discovering by a grid;
- **Layer Info Panel**: widget that show all layer info (details on demand);
- **External WMS Layer Importer**: a set of objects for import an external WMS layer;
- **Layer Preview Generator**: a utility for obtain a preview image of a layer;
- **Data Retrieving Logic**: a set of objects that manage data filter, paging and ordering, exploiting the Geonetwork OGC Catalog Service for the Web (CSW)[28];
- **gCube GeoExplorer**: an external wrapper which interact with some gCube components (such ASL, Workspace Light Tree, and so on).

A diagram of the relationships between these components is reported in Figure 9.

[27] https://en.wikipedia.org/wiki/Web_Map_Service

[28] https://en.wikipedia.org/wiki/Catalog_Service_for_the_Web

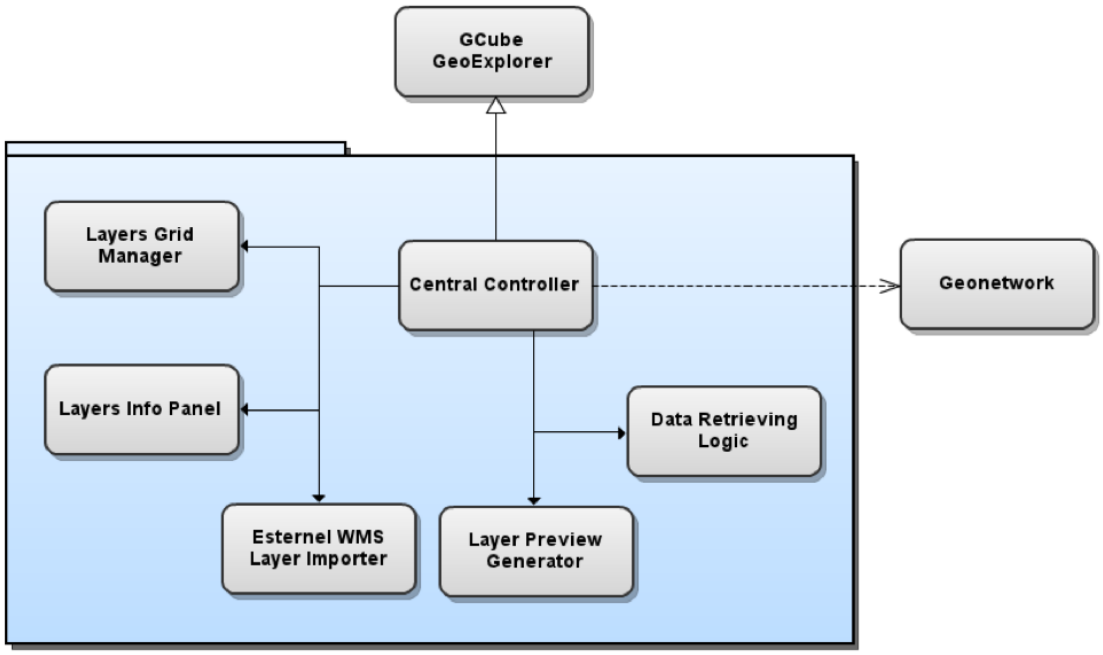


Figure 9. GeoExplorer Architecture

DEPLOYMENT

All the components of the service must be deployed together in a single node. This subsystem can be replicated on multiple hosts and scopes, this does not guarantee a performance improvement because it is a management system for a single input dataset.

USE CASES

The subsystem is particularly suited when experiment have to be performed on occurrence points referring to a certain species or family. The set of operations which can be applied, even lying on state-of-the-art and general purpose algorithms, have been studied and developed for managing such kind of information.

3.2.4.3 GISVIEWER

GisViewer is a web application system that allows you to interactive explore, manipulate and analyze geographic data.

The goal is to provide a standalone interactive analysis tools for information that has one or more spatial characteristics, such as environment maps, geospatial areas, points of occurrence, and so on, all in a web app that must have a simple and intuitive interface that is also the most general possible.

DESIGN

GISViewer is based on Exploratory Data Analysis (EDA) approach, using synergistically computer capabilities to do complex calculations and the man's ability to identify cognitive patterns.

The web-app uses the layer concept: a layer is a minimal visual form of spatial data, it can be a region map, a vector (such as a points set), a variables distribution, and so on. The GISViewer can show one or more layers in an interactive map, allowing users to dissect both information of a layer, and the relationships

between more layers. It's possible to examine more overlapped layers, by setting their transparencies, formats and filters. Tabular geospatial data can be extract by interactive selection of geographical area. Finally, geospatial data can be exported using several output formats.

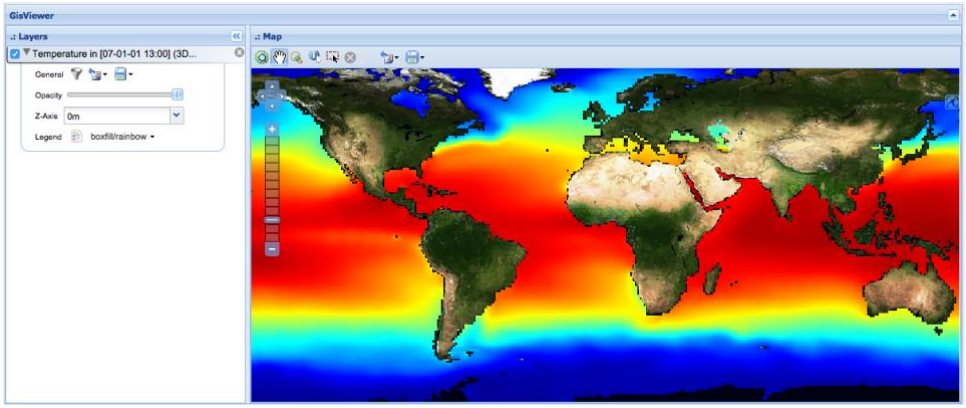


Figure 10. GISViewer Screenshot

ARCHITECTURE

The subsystem comprises the following components:

- **GISViewer Central Controller**: manage all the internal components and their interactions;
- **Layers Managers**: a set of internal objects and widgets that represent a layer info;
- **Openlayers Map**: a set of internal objects and widgets that manage the map, taking advantage of the OpenLayers[29] framework;
- **Data Features**: a set of internal object and widgets that show and represent tabular geospatial data;
- **Utils**: a set of utilities, it includes a CQL[30] filter manager and a WMS[31]/WFS[32] request URL maker;
- **Geonetwork**: an external network of spatial objects, used for highly scalable storing of geospatial resources;
- **GCubeGisViewer**: an external wrapper which interact with some gCube components (such ASL, Workspace Light Tree, and so on).

A diagram of the relationships between these components is reported in the following figure.

[29] http://openlayers.org/

[30] https://en.wikipedia.org/wiki/Contextual_Query_Language

[31] https://en.wikipedia.org/wiki/Web_Map_Service

[32] https://en.wikipedia.org/wiki/Web_Feature_Service

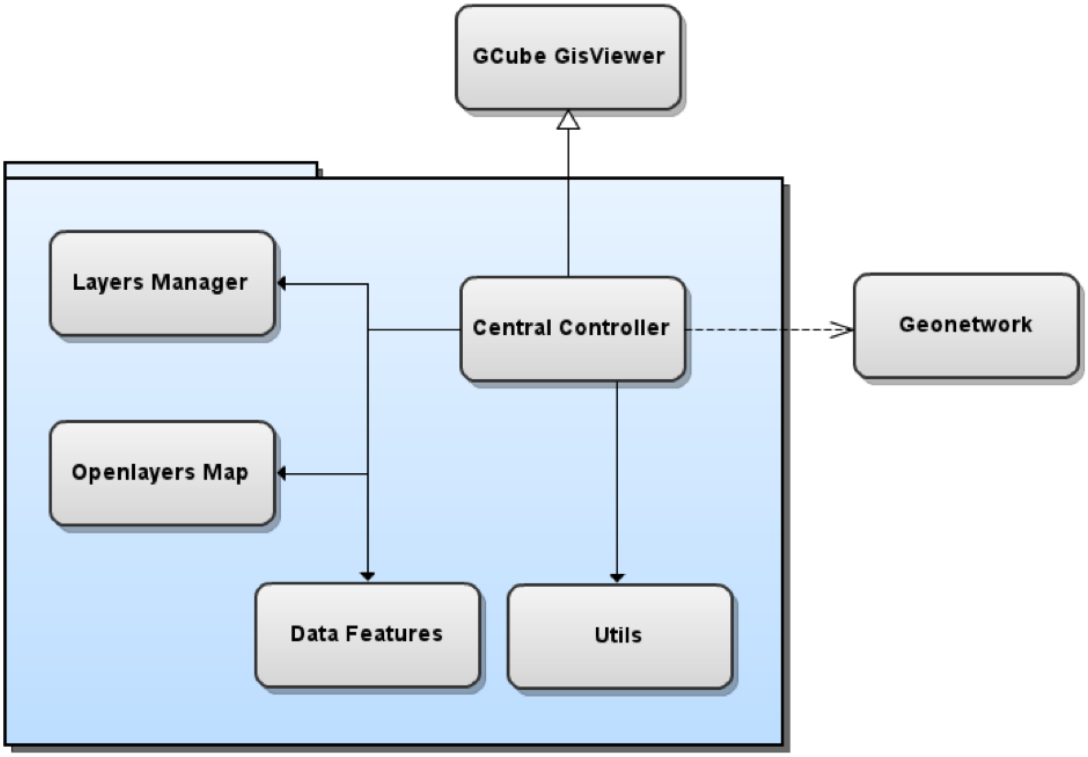


Figure 11. GISViewer Architecture

DEPLOYMENT

All the components of the widget must be deployed together in a single node. This subsystem can be replicated on multiple hosts and scopes, this does not guarantee a performance improvement because it is a management system for a single input dataset.

3.3 EXTERNAL OAI-PMH REPOSITORIES INTEGRATION

External repositories that expose data and metadata through interoperable standards, like the OAI-PMH [33] protocol are of interest for integration into the gCube system, allowing clients to seamlessly discover them through a set of common tools that are available through the gCube and the corresponding gateways and portlets.

To achieve integration and harvesting of such sources a REST service is available and accompanied by a portlet allowing simple and easy integration of such sources. The service has been designed in a pluggable way, in order to offer the ability to use different storage mechanisms for the harvested data. In the context of the BlueBRIDGE project the index framework is used as the main storage system, which automatically makes the data searchable through it.

[33] OAI-PMH Protocol: https://www.openarchives.org/pmh/

3.3.1 KEY FEATURES

Some of the main key features of this service are:

- Ability to integrate an oai-pmh compatible source;
- Default and per-source re-harvesting periods to ensure up to date data / metadata in the infrastructure;
- Ability to work on either the default oai-dc [34]schema or all the available ones from each repository;
- Plug-ability in terms of the storage mechanism used. Support exists for a noSQL storage mechanism based on Mongo DB, file storage and the gCube index – search framework.

3.3.2 DESIGN

The OAI-PMH service has been designed following a pluggable approach to ensure simple extension and additional support of storage mechanisms. The main service follows the REST paradigm and exposes a REST API to be used by its clients. In addition, the service is SmartGears enabled, thus allowing to be deployed in gCube hosting nodes and take advantage of the gCube authorization framework.

On top of the service, a portlet has been implemented and is available through the gateways allowing the end users to simply add repositories for harvesting and monitor the status of each one. The portlet is developed using standard web technologies, the JavaScript[35] language and tools like Bootstrap[36] and jQuery[37]. A responsive design has been taken into account.

3.3.3 USE CASES

The main use case of this service is the integration of external compliant repositories together with internal ones, aiming on enhancing the interoperability and allowing to be searched through a common API. To this end, in the context of the BlueBRIDGE project has been used in collaboration with the index – search framework in order to make external sources available to the VREs.

3.3.4 DEPLOYMENT

The service follows the REST paradigm and can be easily deployed in a SmartGears node. Configuration regarding the storage mechanism to use needs to be configured and in the context of the BlueBRIDGE project the gCube index – search framework should be used and thus both should be available under the desired VRE.

---

[34] OAI-DC: https://www.openarchives.org/OAI/openarchivesprotocol.html#dublincore

[35] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[36] https://getbootstrap.com/

[37] https://jqueryui.com/

3.4 CMEMS DATASET IMPORTER

D4Science provides biotic and abiotic geospatial data from the Copernicus Marine Environment Monitoring Service (CMEMS)[38] that, only for the Mediterranean Region, includes about 150 datasets. CMEMS provides facilities to programmatically browse and download hosted data. Besides WMS and FTP access, a REST APIs exposed by a number of "Motu"[39] web services is also available. Motu is a high-efficiency service that manages the heterogeneity between the data of different CMEMS providers. In particular, Motu handles, extracts and transforms oceanographic huge volumes of data without performance collapse. Interaction with Motu is supported mainly through an open-source Python client[40], which enables extracting and downloading data through command line. This client, however, shows a couple of limitations that do not immediately fit VRE needs: it does not provide browsing capabilities, meaning that one should identify the dataset, as well as other request parameters (e.g. bounding box, time frame) either by browsing the website or interacting directly with the Motu server REST API. Furthermore, the size of the downloaded dataset is typically limited by Motu servers (currently about 1GB); although it's possible to submit multiple requests to a server, the client does not provide support for merging data afterwards.

D4Science facilitates the access to CMEMS data with a Dataset Importer facility[41] to allow users to browse and select datasets, configure the area of interest, specify a basic post-processing action (e.g. average) and have data published and regularly updated in the D4Science THREDDS Data Server and indexed in GeoNetwork. Products can then be made available in all interested VREs. The adopted approach simplifies the access to CMEMS metadata and datasets, overcoming some of the limitations of the Motu server and client and provides users with an easy-to-use command-line and web interface to manage synchronized datasets.

3.4.1 KEY FEATURES

The key features of the CMEMS Dataset Importer are:

- Support one-time and scheduled CMEMS Dataset import tasks;
- Web and Command-line interfaces to search, browse and management of import tasks;
- Full Motu API implementation (either XML and HTML). HTML API are used where no equivalent XML operations are available;
- Support for CAS authentication;
- Support for large dataset downloads (i.e. > 1GB). The request is split in a number of smaller requests and finally merged in a single output;
- Parallel request submission and dataset download, within the limits enforced by the server;
- Robust parameter management. Default values are set in the client to avoid server errors;

[38] http://marine.copernicus.eu

[39] https://github.com/clstoulouse/motu

[40] https://github.com/clstoulouse/motu-client-python

[41] https://wiki.gcube-system.org/gcube/CMEMS_Dataset_Importer

3.4.2 DESIGN

The core of the CMEMS Importer is realized as a plugin for the Smart Executor[42], offering facilities to execute tasks in a scheduled and reliable way, and monitor their progress and status. The CMEMS Importer consists of the following components:

- **cmems-client:** is responsible for the interaction with CMEMS services. It provides 1) facilities to browse and search the CMEMS catalogue, as available online[43]; 2) a complete Java implementation of a Motu client, exposing most-useful REST and HTML APIs[44]; the Motu client overcomes the limit of 1GB maximum download size enforced by most Motu servers by automatically splitting the request in small chunks and merging them back to a single dataset.
- **cmems-importer-se-plugin:** is the component of the Importer in charge of downloading the relevant (part of the) dataset from the various CMEMS servers, uploading it to the gCube THREDDS Server and keeping it updated. It's implemented as a plugin for the SmartExecutor, the import/update action can be either executed once or on a periodic basis, as requested by the user.
- **cmems-importer-client:** is a client library to easily interact with the SmartExecutor to schedule, start, stop and monitor the execution of an import task, as well as to list the tasks currently running/scheduled. It also includes a command-line interface (CLI) to perform all the relevant actions (search and browse datasets, import task management).
- **cmemes-importer-gui:** is a graphical user interface (GUI) with the same capabilities offered by the CLI above, integrated in the gCube Portal.
- **dataset-importer-common:** a set of utility classes and methods and clients for other gCube services common to the other CMEMS Importer components.

The CMEMS Importer interact with the following gCube/external services:

- **CMEMS online catalogue**: holding metadata about all datasets along with references to the servers delivering them with various protocols (Subsetter, WMS, DGF, FTP);
- **Motu servers:** a number of servers, operated by different research institutions, providing authenticated access to CMEMS datasets;
- **CAS authentication server:** a single Identity Provider performing user authentication on behalf of all services in the CMEMS environment.
- **gCube THREDDS Data Server:** is the gCube web server holding metadata and data access to all the CMEMS imported datasets.
- **SmartExecutor:** manages the scheduled execution of all import tasks.

[42] https://gcube.wiki.gcube-system.org/gcube/SmartExecutor

[43] http://marine.copernicus.eu/services-portfolio/access-to-products

[44] https://github.com/clstoulouse/motu#ClientsAPI

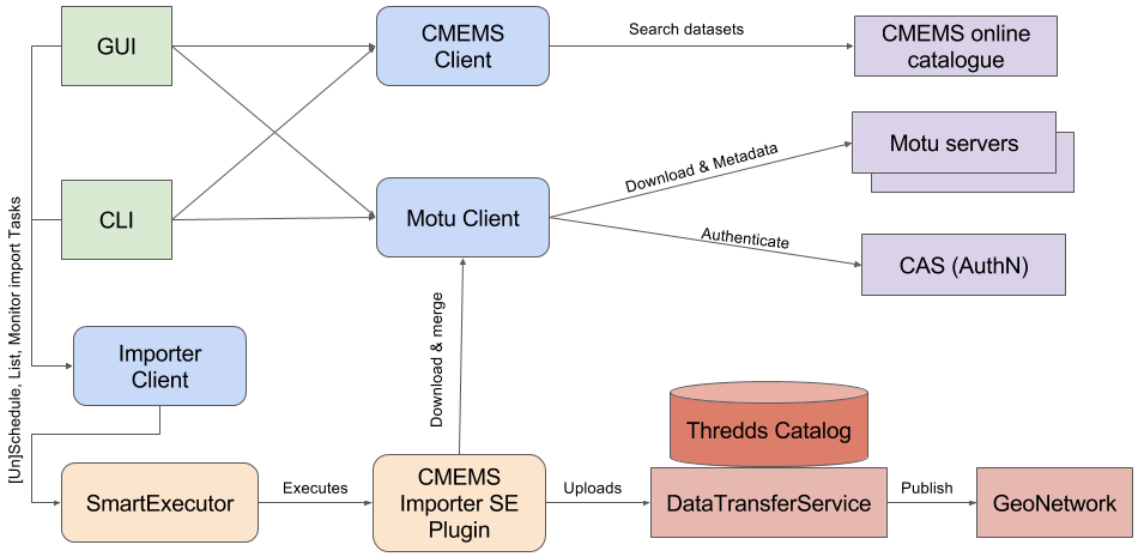


Figure 12. CMEMS Dataset Importer Architecture

The CMEMS Importer depends on a few external components including common utility packages (Apache Commons[45]), testing and logging facilities (JUnit[46] and SLF4J[47]).

3.4.3 DEPLOYMENT

The CMEMS Dataset Importer is delivered as a set of three deployable artefacts:

- The **SmartExecutor plugin** (embedding the *cmems-importer-se-plugin, cmems-client* and *dataset-importer-common*) which is deployed on a SmartExecutor node in the gCube infrastructure.
- The **CMEMS Importer GUI** (embedding the *cmemes-importer-gui, cmems-importer-client, cmems-client* and *dataset-importer-common*) is deployed as a portlet in the gCube Infrastructure Gateway.
- The **CMEMS Importer CLI** (embedding the *cmems-importer-client* and *dataset-importer-common*) can be deployed on any host, also outside the gCube infrastructure, equipped with an adequate authorization token (see section 4.3 gCube Authorization Framework)

[45] https://commons.apache.org

[46] http://junit.org

[47] https://www.slf4j.org

4 FACILITIES FOR FEDERATED RESOURCES MANAGEMENT

This section includes the work done in T10.3 of the project that brought to the release a set of gCube libraries, tools, services and front-end applications to effectively and efficiently manage the federated resources made available thanks to bridges developed in T10.1 and T10.2.

4.1 FHNMANAGER PORTLET

The FHNManager Portlet is a front-end application for the administration of gCube hosting nodes running on external integrated cloud infrastructures. The application is completely integrated in the gCube portal the same look-and-feel of the other gCube administration applications. It relies on the FHNManager service as backend.

This facility has been entirely developed in the BlueBRIDGE project. Updates are expected in the future to release new functionalities and enhance it.

4.1.1 KEY FEATURES

Key features offered by the portlet are:

- CRUD operations on FHNManager model entities;
- Start/Stop nodes on federated providers
- Report generation and sharing

4.1.2 DEPLOYMENT

In order to install the portlet, it is necessary to obtain last stable war from gCube distribution and deploy it in a gCube Portal; this way, the GUI will be visible to VRE/VO managers.

The portlet relies on portal's scope and user management facilities; thus, no additional configuration is needed.

4.2 GCUBE ACCOUNTING FRAMEWORK

The gCube accounting framework allows to collect information on infrastructure resources usage for "internal" as well as "external" resources. Once collected "usage records" are made available for consumption, e.g., authorized users will be provided with a portlet to analyse them. Data is automatically harmonized, aggregated and stored by the framework. Moreover, the framework includes also an analytics library.

The gCube accounting framework has been completely rewritten in BlueBRIDGE to meet scalability and extensibility requirements specific of this project that the old accounting framework could not address.

4.2.1 KEY FEATURES

Open and extensible accounting model

- the underlying accounting model is flexible to adapt to diverse provider needs;

Highly modular and extensible architecture

- the entire subsystem comprises a large number of components clearly separating the functional constituents

Diverse options for storage

- the subsystem can rely on an array of diverse solutions for actually storing records

4.2.2 DESIGN

4.2.2.1 ARCHITECTURE

The following picture shows the gCube Accounting architecture, that is logically divided in three different layers:

- Accounting Consumer Layer
- Accounting Enabling layer
- Accounting Backend layer

Moreover, a fourth layer (which is not gCube-based) exist to take care of the storage of the collected information:

- Accounting Storage layer

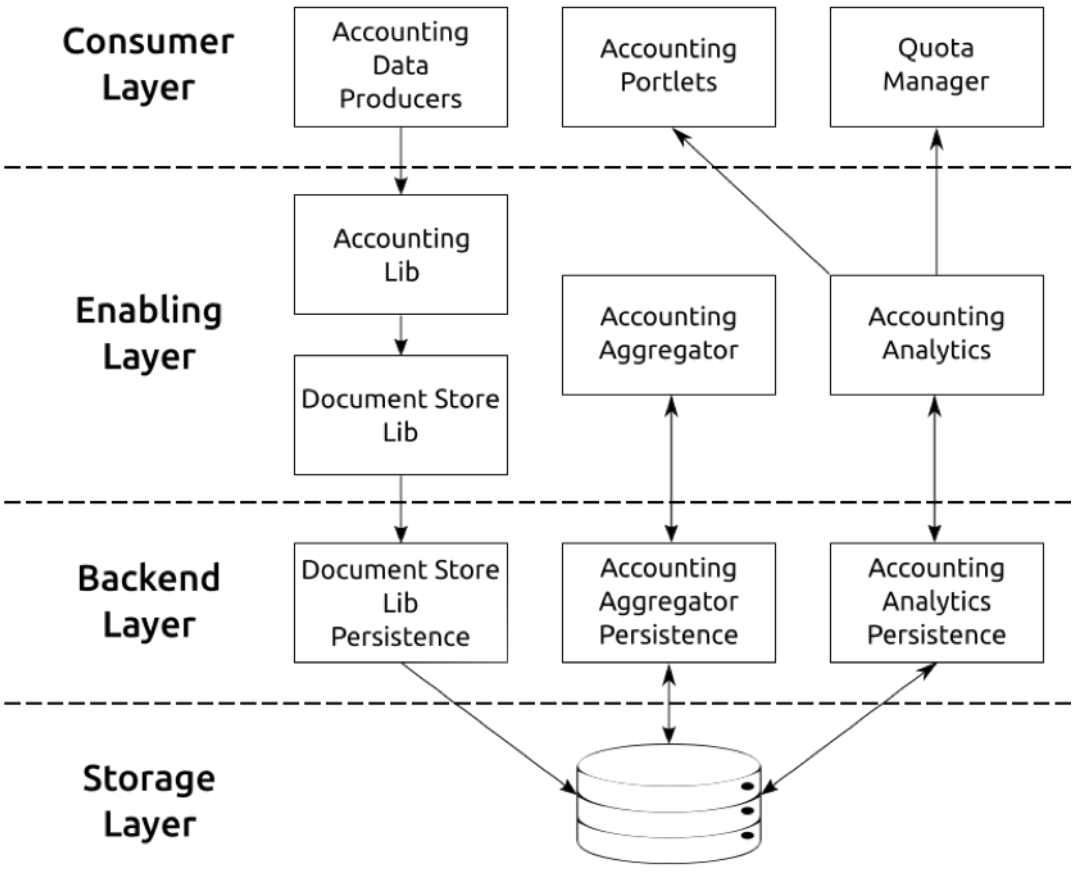


Figure 13. gCube Accounting Architecture

All the developed component must respect this architecture. In particular, the design and the implementation has to respect the following rules:

- Each enabling layer has its own correspondent back-end implementation.
- Each back-end implementation must be dynamically discovered at run-time. This allow to decouple the deployment of a different back-end from the development of the enabling layer. In other word each component on the enabling layer MUST NOT have any dependency over a certain back-end implementation.

The only entitled software to depend to the back-end component are:

- bundles (e.g. smartgear-bundle, portal-bundle);
- provisioning script.

4.2.2.2 ACCOUNTING ENABLING LAYER

- **Accounting Lib**[48] is responsible of collecting, harmonizing and storing accounting data. Accounting Lib is mainly based and developed on top of facilities provided by **Document Store Lib**[49].
- **Accounting Analytics**[50] is responsible of exposing a common access point interface to query the collected accounting data.
- **Accounting Aggregator**[51] is responsible of aggregating the collected accounting data.

4.2.2.3 ACCOUNTING STORAGE LAYER

This layer is not developed by gCube. Rather it relies on a technology guaranteeing High Availability (HA). In the current settings, it is implemented by relying on CouchBase[52]. CouchDB[53] and MongoDB[54] are also supported. OrientDB[55] has been also investigated.

4.2.2.4 ACCOUNTING BACKEND LAYER

This comprises:

[48] https://wiki.gcube-system.org/gcube/Accounting_Lib

[49] https://wiki.gcube-system.org/gcube/Document_Store_Lib

[50] https://wiki.gcube-system.org/gcube/Accounting_Analytics

[51] https://wiki.gcube-system.org/gcube/Accounting_Aggregator

[52] https://www.couchbase.com/

[53] https://couchdb.apache.org/

[54] https://www.mongodb.com/

[55] http://orientdb.com/

- document-store-lib-BACKEND (implementations available: document-store-lib-couchdb, document-store-lib-couchbase, document-store-lib-mongodb, document-store-lib-accounting-service).
- accounting-analytics-persistence-BACKEND (implementations available: accounting-analytics-persistence-couchdb, accounting-analytics-persistence-couchbase)
- accounting-aggregator-persistence-BACKEND (implementation available: accounting-aggregator-persistence-couchbase)

Each component in this layer has been explicitly developed over a certain storage technology. Each component MUST NOT rely on IS to discover the information needed to connect to the underling storage. In other words, the component MUST NOT have a hard-cabled connection information or a local configuration files. To retrieve the storage connection information, the following parameter must be part of the query:

- underlying storage technology
- enabling component

The first constraint allows to switch to different storage avoiding to switch all nodes together (two underling storage can co-exist). The second allow to keep separated the connection information for each component. This allow to provide create different access policy for different component (e.g. write only for accounting-lib connection and read only for accounting-analytics).

4.2.2.5 ACCOUNTING-SERVICE

Accounting-service is a RESTFull stateless service. The service receives accounting data from gCube nodes and uses the accounting stack to persist the records in CouchBase cluster.

Nodes which does not have access to CouchBase are equipped with document-store-lib-accounting-service. Document-store-lib-accounting-service library send records via HTTPS connection to one of the accounting-service instances. Accounting-service instances are equipped for clear reason with document-store-lib-couchbase to store the received records to CouchBase. The service avoids to expose the database directly to all infrastructure nodes, which can cause security issues.

4.2.2.6 ACCOUNTING CONSUMER LAYER

In this layer we find not only the accounting consumer which retrieve information related to collected data (e.g. Accounting Portlet, Quota Manager, and Accounting Dashboard) but also all the components that collect data and use accounting-lib to persist in a common pattern.

4.2.3 USE CASES

The Accounting framework collects all operations performed by the users and services accessing other infrastructure services. It is used mainly to implement the following use cases:

- Monitoring of the exploitation of the resources across VREs
- Identification of anomalies and incorrect exploitation of the resources
- Optimization of the resources allocated to the VREs
- Identification of the limits to associate to the resources to enable quota management

4.2.4 DEPLOYMENT

In gCube nodes all needed libraries (e. accounting-lib, document-store-lib, document-store-lib-accounting-service or document-store-lib-couchbase) are available giving that they are contained in the node distribution (I.e. smartgears-bundle, portal-bundle).

Accounting-service and CouchBase cluster are expected to be deployed and registered in the infrastructure.

Accounting-service is deployed to achieve High Availability by using HAProxy in front of the service instances. This deployment configuration provides also load balancing between service instances, actually 2 (two).

CouchBase is deployed in cluster with shards and replicas of data and indexes to achieve High Availability. Currently the cluster is composed of 8 (eight) server with a total amount of 36 GB of RAM and 2.7 TB of storage.

4.3 GCUBE AUTHORIZATION FRAMEWORK

The gCube Authorization framework is a token-based authorization system compliant with the Attribute-Based Access Control (ABAC[56]) and with the OASIS XACML[57] reference architecture. The token is a string generated on request by the Authorization service for identification purposes and associated with every entity belonging to a gCube infrastructure (users or services).

The gCube authorization framework has been completely developed in BlueBRIDGE project with token-based state-of-the-art technologies to allow a secure communication channel inside a gCube infrastructure and between gCube and external infrastructures.

The token is passed in every call and is automatically propagated in the call stack.

4.3.1 KEY FEATURES

- Self-service credential management
- Highly scalable with high availability support
- Provides access control across domains and application technologies in the d4science infrastructure

4.3.2 DESIGN

4.3.2.1 TOKEN BASED AUTHORIZATION

The token is a string generated on request by the Authorization service for identification purposes and associated with every entity belonging to a gCube-based infrastructure (users or services). The token is passed in every call and is automatically propagated in the lower layers. The token can be passed to a SmartGears service in 3 ways:

[56] https://en.wikipedia.org/wiki/Attribute-based_access_control

[57] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

- using the HTTP-header: adding the value ("gcube-token","{your-token}") to the header parameters.
- using the query-string: adding gcube-token={your-token} to the existing query-string
- logging via the default authentication widget showed by the browser using your gCube username as username and your token as password.

The personal token can be retrieved using the token widget deployed on every environment of the portal.

4.3.2.2 POLICY LANGUAGE

The gCube Authorization Framework controls access to applications to allow or prevent the clients to perform various operations in the application. This is controlled by the Authorization Service embedded in the SmartGears framework with the help of authorization policies. The purpose of authorization policies is to control client access. The authorization policies determine at runtime whether or not a particular action is denied. You can define authorization policies that satisfy the authorization requirements using the policy language.

All the policies created in the system are used to DENY to a client an operation in a specific context. Two types of policy are supported:

- User2Service (U2S)
- Service2Service (S2S)

The U2S policies are used to deny to a user or a role the access to specific service or class of services. The S2S policies are used to deny to a service or a class of services the access to specific service or class of services. To make easier the possibility to allow access only to few clients an “except” restriction is defined in the policies.

For every policy a specific ACTION to prevent can be added (if supported from the service) otherwise all the ACTION will be denied:

- ALL (default)
- ACCESS
- WRITE
- DELETE
- EXECUTE

In the following some examples:

- U2S(context, User('myUserName'), Service(ServiceClass, ServiceName, ServiceIdentifier))
- U2S(context, Role('role'), Service(ServiceClass, ServiceName, ServiceIdentifier))
- U2S(context, allExcept[Role('role1'), Role('role2')], Service(ServiceClass, ServiceName, ServiceIdentifier))
- S2S(context, Service(ServiceClass1, ServiceName1, ServiceIdentifier), Service(ServiceClass2, ServiceName2, ServiceIdentifier2))
- S2S(context, Service(ServiceClass1, ServiceName1, ServiceIdentifier), Service(ServiceClass2, *))
- S2S(context, allExcept[Service(ServiceClass1, ServiceName1,*)], Service(*))

4.3.2.3 ARCHITECTURE

The Authorization framework is compliant with the XACML reference architecture. The XACML standard proposes a reference architecture with commonly accepted names for the various entities involved in the architecture. It is composed by 5 modules

- Policy Administration Point (PAP) - Point which manages access authorization policies
- Policy Decision Point (PDP) - Point which evaluates access requests against authorization policies before issuing access decisions
- Policy Enforcement Point (PEP) - Point which intercepts user's access request to a resource, makes a decision request to the PDP to obtain the access decision (i.e. access to the resource is approved or rejected), and acts on the received decision
- Policy Information Point (PIP) - The system entity that acts as a source of attribute values (i.e. a resource, subject, environment)
- Policy Retrieval Point (PRP) - Point where the XACML access authorization policies are stored, typically a database or the filesystem.

The 5 modules' capabilities are implemented by gCube as follow.

- Policy Administration Point (PAP) is the gCube Authorization Service
- Policy Decision Point (PDP) is implemented by a PDP library distributed with gCube SmartGears
- Policy Enforcement Point (PEP) is implemented by a PEP library distributed with gCube SmartGears
- Policy Information Point (PIP) is implemented by the gCube Information System
- Policy Retrieval Point (PRP) is implemented by a database controlled exclusively by the gCube Authorization Service

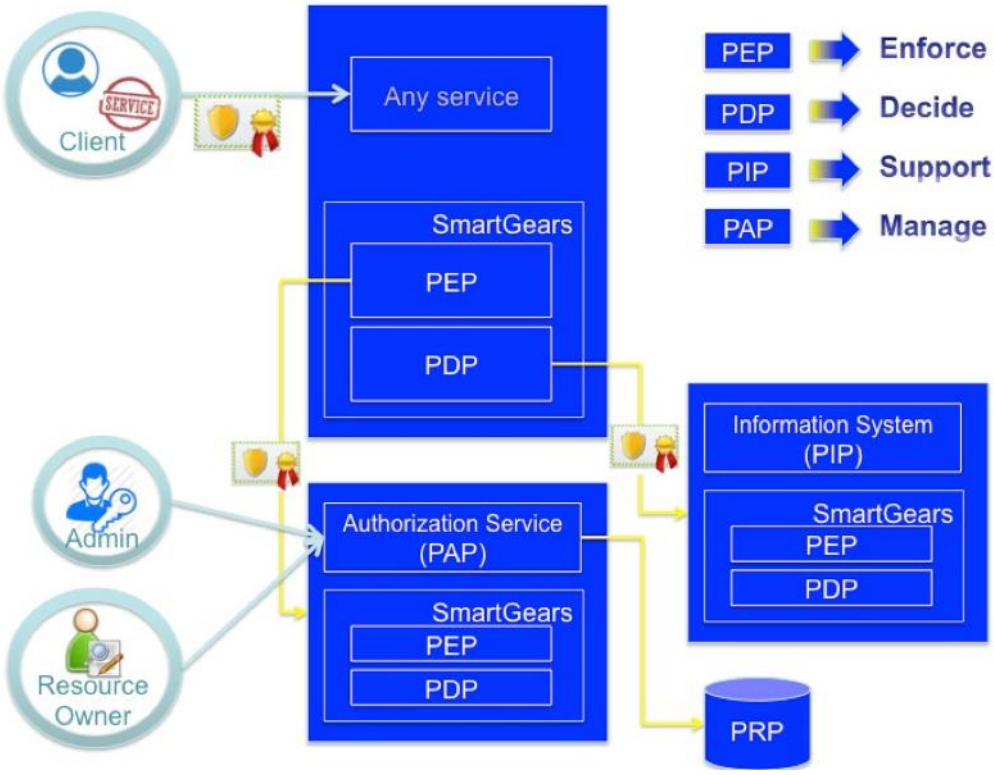


Figure 14. Authorization Framework Architecture

The following flow of control governs the authorization flow.

- The policy author uses policy authoring tools (GUI) (part of the PAP) to write policies governing access and exploitation of his/her own resources.

- The policy administrator then uses the PAP GUI to administer the policies. Please note that policies are not distributed to PDPs upon their creation but at first request referring access/exploitation of a given resource. PDPs use a cache with TTL to avoid the exchange of too many requests.

- The PEP intercepts the business level request to access the resource decorated with a token. It resolves the token by sending a request to the PAP and gets back information about the validity of the token to operate in the specific operational context. If the access is denied (invalid token) a Deny Response is immediately issued. If the access is permitted the request to the PAP allows to populate the PDP cache with the appropriate policies. Then it produces a request out of it and sends it to the PDP for actual decision making.

- The PDP, on receiving the request, looks up the policies deployed on it and figures out the ones which are pertinent to the specific request. It may, if necessary, query the PIP for additional attributes that are needed to evaluate the policies. By exploiting the attributes contained in the request, the attributes obtained from the PIP and attributes that are generic to the operational context, the PDP decides whether the request can be allowed (Permit response), denied (Deny response), is not applicable since none of the policies deployed on it can be used to evaluate the request (NotApplicable response) or there was some issue with evaluating the response against the policy, for example due to lack of sufficient attributes available to the PDP (Indeterminate response).

- The response is then sent by the PDP to the PEP. The PEP parses the response from the PDP and handles each of the four possible response cases. If either a Permit or a NotApplicable response is get back then the business request is passed to the service, otherwise a Deny response is issued.

4.3.3 USE CASES

The Authorization framework allows to implement a policy-based allocation of resources to VREs. It supports the following use cases:

- Assignment of a service to a VRE
- Authorization of a person to access a VRE
- Either temporary or permanent ban of a person in the infrastructure
- Either temporary or permanent ban of a person in a specific VRE
- Either temporary or permanent ban of a person in accessing a specific service in a VRE

4.3.4 DEPLOYMENT

As described in the Architecture section, the Data Transfer suite is composed by three main components:

- **authorization-client-library** is a java library that can be used by any other java application to contact the authorization service.
- **authorization-model-library** is a java library that implements the gcube-authorization model.
- **authorization-service** is a REST Service working on a java servlet container.

4.4 DATA TRANSFER SERVICES

The implementation of a reliable data transfer mechanisms between the nodes of a gCube-based Hybrid Data Infrastructure is one of the main objectives when dealing with large set of multi-type datasets distributed across different repositories.

To promote an efficient and optimized consumption of these data resources, a number of components have been designed to meet the data transfer requirements.

It is possible to distinguish two different main components:

- the Result Set components
- the Data Transfer 2

4.4.1 KEY FEATURES

The components belonging to this class are responsible for:

Reliable data transfer between Infrastructure Data Sources and Data Storages

- by exploiting the uniform access interfaces provided by gCube and standard transfer protocols

Structured and unstructured Data Transfer

- it guarantees both Tree based and File based transfer to cover all possible iMarine use-cases

Transfers to local nodes for data staging

- data staging for particular use cases can be enabled on each node of the infrastructure

Advanced transfer scheduling and transfer optimization

- a dedicated gCube service responsible for data transfer scheduling combined to transfer optimization at the level of protocols and Access interfaces.

Transfer statistics availability

- transfers are logged by the system and make available to interested consumers.

Transfer shares per scopes and users

- a management interface is used to configure transfer shares per scopes and users at the level of Data Sources and Storages.

4.4.2 DATA TRANSFER 2

The implementation of a reliable data transfer mechanisms between the nodes of a gCube-based Hybrid Data Infrastructure is one of the main objectives when dealing with large set of multi-type datasets distributed across different repositories.

To promote an efficient and optimized consumption of these data resources, a number of components have been designed to meet the data transfer requirements.

4.4.2.1 KEY FEATURES

The components belonging to this class are responsible for:

Point to Point transfer

- direct transfer invocation to a gCube Node

Reliable data transfer between Infrastructure Data Sources and Data Storages

- by exploiting the uniform access interfaces provided by gCube and standard transfer protocols

Automatic transfer optimization

- by exploiting best available transfer options between invoker and target nodes

Advanced and extensible post transfer processing

- plugin - oriented implementation to serve advanced use case

Controlled access to remote filesystems

- REST Interface towards configurable local paths

4.4.2.2 DESIGN

In a Hybrid Data e-Infrastructure, transferring data between nodes is a crucial feature. Given the heterogeneous nature of datasets, applications, networks capabilities and restrictions, efficient data-transfers can be problematic. Negotiation of best-efficient-available-transfer method is essential in order to achieve a global efficiency, while an extensible set of post operations lets application serve specific needs among common ones. We wanted to provide a light-weight solution that could fit the evolving nature of the infrastructure and its communities' needs.

ARCHITECTURE

The architecture of Data Transfer is a simple client server model, respectively implemented by *data-transfer-library* and *data-transfer-service*.

The service is a simple REST server, implemented as a gCube SmartGear web application. It is designed exploiting the plugin design pattern, so that it can be easily extended without introducing hard dependencies between the architecture's components. It also exposes interfaces aimed to:

- get node's capabilities in terms of available plugins and supported transfer method;
- submit and monitor transfer requests

The client is a java library that:

- contacts the server asking for its capabilities in order to decide the best efficient transfer way between the two sides;
- submits a transfer request through the selected transfer channel
- monitors its status/outcome.

Optionally, a transfer request can specify an ordered set of plugin invocation, that are executed after the transfer has been completed. These executions are monitored by the client as part of the atomic transfer process.

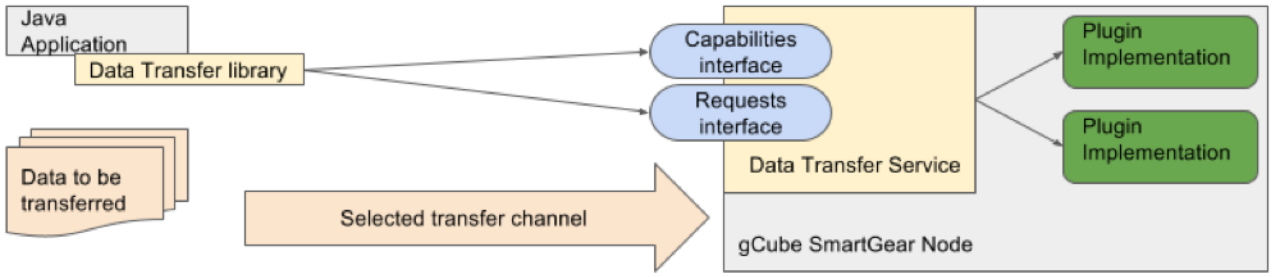


Figure 15. Data Transfer Architecture

4.4.2.3 DEPLOYMENT

As described in the Architecture section, the Data Transfer suite is composed by three main components:

- **data-transfer-library** is a java library that can be used by any other java application. Thus the two need to be in the same classpath.
- **data-transfer-service** is a gCube SmartGears REST service, thus it needs to be hosted in a Smart Gear installation.
- *plugin* implementations are discovered at runtime by the service, thus they need to be deployed along with the *data-transfer-service* itself.

4.4.2.4 GENERAL PURPOSE PLUGINS

This section describes general purposes plugin, which are included in **default distributions**. This means that these plugins are always available on a SmartGears node.

DECOMPRESS ARCHIVE PLUGIN

The 'Decompress Archive' plugin extracts the content of an archive to a specified path. It can be useful in different applications (i.e. web applications deployment) thus it is considered general purpose.

4.4.2.5 SPECIFIC PLUGINS

This section lists plugins modules designed to address a particular installation (typically the management of third party applications). They will be available only on certain installation nodes, depending on needs.

THREDDS PLUGIN SUITE

THREDDS plugin suite contains a set of plugins aimed to manage a THREDDS installation in a gCube infrastructure. It is always present in a gCube-enabled THREDDS instance (see **3.2.3.1 GCUBE-Enabled Services**).

The suite contains the following plugins:

- **SIS/GEOTK plugin:** extracts ISO Metadata from **netcdf** files by exploiting Apache SIS[58] library, and publishes it to GeoNetwork Catalogue;
- **REGISTER CATALOG plugin:** register a THREDDS catalogue configuration file into the main one;

The suite also exposes a JSON Object which represents the current structure of THREDDS Catalogues for management purposes.

4.4.2.6 USE CASES

The Data Transfer is designed mainly for point to point transfers, so its use is most suited for moving datasets where they need to be stored and processed.

The automatic transfer means negotiation mostly helps in situations where the remote nodes are in heterogeneous networks and relative restrictions (i.e. cloud computing).

Implemented plugins can help to deploy and prepare data locally to those storage technologies that have particular needs (i.e. THREDDS, GeoServer).

The use of logic placeholders for actual persistence paths, allows for homogeneity of approach between different installations.

The REST interface allows controlled access to remote filesystems just by HTTP interactions, making it easy to integrate in every environment.

WELL SUITED USE CASES

The feature is best designed for the following use cases:

- Point to point transfer in heterogeneous networks
- REST access to services filesystems

LESS WELL SUITED USE CASES

More advanced use cases such as scheduled tasks and parallel transfers with strong requirements on data coherence between nodes are beyond the scope of the current implementation, and needs to be managed by the user application.

WORKSPACE SYNCHRONIZATION USE CASE

A particular use case is strongly based on the exploitation of DataTransfer2 capabilities and its Thredds plugin suite: Workspace synchronization (see **3.2.3.4 Workspace**).

[58] http://sis.apache.org/

4.4.2.7 DEPLOYMENT

Data Transfer 2 service is distributed as a SmartGears web service. It is included in SmartGears distribution; thus, it is automatically deployed within any SmartGears node.

Being a web service, its war can be deployed on any web application container, granted that SmartGears common libraries are available in the classpath.

Additional plugins are distributed as uber-jars containing all their needed dependencies, and they need to be placed in the service classpath in order to be discovered by it. Deployment details may vary depending on the container capabilities.

Data Transfer 2 is a REST service, thus no client implementation is strictly required to interact with it. However, a java client library is distributed along with the service to help exploit its logic. It needs to be in caller's classpath at runtime in order to be exploited.

5 SOFTWARE DISTRIBUTION

All the software produced for the gCube system follows a common integration, testing and release process and, therefore, all facilities share the same location for:

- **source code**: available on GitHub at **https://github.com/gcube-system/gcube-releases**;
- **binary packages**: available in the gCube download page on the gCube website at **https://www.gcube-system.org/software-releases**.

All the facilities described in this document are delivered with the **EUPL v1.1 software license**.

REFERENCES

[1] Candela, L., Coro, G., Frosini, L., Galante, N.A., Giammatteo, G., Kakaletris, G., Lelii, L., Marioli, V., and Sinibaldi, F. (2016) BlueBRIDGE Resources Federation Facilities. BlueDRIDGE Project Deliverable D10.1

[2] BlueBRIDGE project wiki - https://support.d4science.org/projects/bluebridge/wiki

[3] gCube wiki - https://wiki.gcube-system.org/gcube/About_gCube