

PARTHENOS

Pooling Activities, Resources and Tools
for Heritage E-research Networking,
Optimization and Synergies

PARTHENOS Cloud Infrastructure (final)

PARTNER(s): CNR

DATE: 30 April 2019



PARTHENOS is a Horizon 2020 project funded by the European Commission. The views and opinions expressed in this publication are the sole responsibility of the author and do not necessarily reflect the views of the European Commission.





HORIZON 2020 - INFRADEV-4-2014/2015:

Grant Agreement No. 654119

PARTHENOS

Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization
and Synergies

NAME OF THE DELIVERABLE

Deliverable Number D6.6

Dissemination Level Public

Delivery date 30 April 2019

Status Final

Author(s) Pasquale Pagano
Massimiliano Assante
Luca Frosini
Paolo Manghi
Alessia Bardi
Fabio Sinibaldi
Roberto Cirillo
Giancarlo Panichi



Project Acronym	PARTHENOS
Project Full title	Pooling Activities, Resources and Tools for Heritage E-research Networking, Optimization and Synergies
Grant Agreement nr.	654119

Deliverable/Document Information

Deliverable nr./title	D6.6 PARTHENOS Cloud Infrastructure (final)
Document title	PARTHENOS Cloud Infrastructure (final)
Author(s)	Massimiliano Assante, Alessia Bardi, Roberto Cirillo, Luca Frosini, Paolo Manghi, Pasquale Pagano, Giancarlo Panichi and Fabio Sinibaldi.
Dissemination level/distribution	Public

Document History

Version/date	Changes/approval	Author/Approved by
V 0.1 15.01.19	Revised Structure, table of contents, introduction	Massimiliano Assante
V 0.2 05.03.19	Updated 2.6.3.2. Shared Workspace System and 2.1.2.1 Infra Monitoring and Alerting tools	Massimiliano Assante
V 0.3 25.03.19	Updated 2.3.1.2. Resource Registry	Luca Frosini and Pasquale Pagano
V 0.4 11.04.19	Updated Section 2.5 Content Cloud Framework	Alessia Bardi
V. 0.5 17.04.19	Minor Update to Section 2.5 Content Cloud Framework	Alessia Bardi
V. 0.6 18.04.19	Updated Summary, ToC and List of Figures	Massimiliano Assante
V. 0.7 24.04.2019	Review	Sheena Bassett

This work is licensed under the Creative Commons CC-BY Licence. To view a copy of the licence, visit <https://creativecommons.org/licenses/by/4.0/>



Table of contents

Executive summary	1
1. Introduction	3
1.1. Structure of this report	4
2. Cloud Infrastructure	5
2.1. Hardware Layer	5
2.1.1. Enabling Technology.....	5
2.1.2. Supporting Technology.....	6
2.1.2.1. Monitoring and Alerting System.....	6
2.1.2.2. Provisioning System.....	9
2.2. Enabling Framework	10
2.2.1. Overview	10
2.2.2. Key Features	12
2.2.3. Subsystems.....	12
2.2.3.1. Resource Registry	12
2.2.3.2. Resource Manager.....	15
2.2.3.3. Virtual Research Environment Manager.....	16
2.2.3.4. Authentication and Authorization	17
2.2.3.5. Accounting.....	22
2.3. Storage Framework	25
2.3.1. Overview	25
2.3.2. Key Features	26
2.3.3. Subsystems.....	26
2.3.3.1. File-Based Store System.....	26
2.3.3.2. Metadata Store System	28
2.3.3.3. Spatial Data Repositories.....	28
2.4. Analytics Framework	31
2.4.1. Overview	31
2.4.2. Key Features	31
2.4.3. Subsystems.....	31
2.4.3.1. Data Miner System.....	31
2.4.3.2. Smart Executor System.....	33
2.5. Content Cloud Framework	35
2.5.1. Overview	35
2.5.2. Key Features	36
2.5.3. Subsystems.....	36



2.5.3.1.	Workflow Management.....	38
2.5.3.2.	Data Source Manager.....	40
2.5.3.3.	Metadata Collector Service.....	42
2.5.3.4.	Transformator.....	46
2.5.3.5.	Provision services.....	47
2.6.	Collaborative Framework.....	49
2.6.1.	Overview.....	49
2.6.2.	Key Features.....	50
2.6.3.	Subsystems.....	50
2.6.3.1.	Social Networking System.....	50
2.6.3.2.	Shared Workspace System.....	51
2.6.3.3.	User Management System.....	53



Table of Figures

Figure 1. High-level architecture of the PARTHENOS infrastructure.....	4
Figure 2. Nagios Status Report and Availability Report for the Accounting Cluster	7
Figure 3. Prometheus Aggregated View via Grafana for the Accounting Cluster	9
Figure 4. Enabling Framework Architecture.....	11
Figure 5. Resource Registry Architecture.....	14
Figure 6. Resource Manager Architecture	16
Figure 7. VRE Manager Architecture	17
Figure 8. Authorization Architecture.....	20
Figure 9. Accounting Architecture.....	23
Figure 10. File-Based System Architecture	27
Figure 11. Spatial Data Repositories Architecture	30
Figure 12. Data Miner System Architecture.....	32
Figure 13. Smart Executor System Architecture.....	34
Figure 14. D-NET Aggregative Infrastructure Architecture	37
Figure 15. The data flow devised for the PARTHENOS aggregator	39
Figure 16. D-Net workflows for PARTHENOS.....	40
Figure 17. GUI for Data Source Management	40
Figure 18. Modify connection parameter to a remote data source	41
Figure 19. Parameters section: configuration of the aggregation workflow.....	41
Figure 20. History section: view status of past executions of the same workflow	41
Figure 21. Other settings section: configure scheduling and email notifications	42
Figure 22. Screenshot of the D-Net Metadata Inspector	48
Figure 23. Screenshot of the PARTHENOS Joint Resource Registry	49



Tables

Table 1. Analysis of the APIs of PARTHENOS research infrastructure with respect to the collection plugins natively available in D-NET.....	46
--	----



Executive Summary

“D6.6 PARTHENOS cloud infrastructure” is the revised and final version of “D6.1 PARTHENOS cloud infrastructure”. This deliverable reports the PARTHENOS e-infrastructure architecture: the hardware and the services. Hardware is organized as a dynamic cloud of virtual machines, supporting computation and storage, while the services are organized into e-infrastructure middleware, storage, and end user services.

This revised version of the document covers the whole period of the project, including the up to date information of the D6.1 deliverable and the improvements implemented for the management of the hardware as well as for the performance of the services developed through the project’s lifetime. Specifically, Section 2.1 is updated with information about the technology change that has occurred in the infrastructure regarding the Monitoring and Alerting tools, Section 2.3 is updated for the part concerning the enhancements delivered in the Resource Registry, and Section 2.5 is updated with information about the customization, in terms of workflows and services, of the D-Net instance for the implementation of the PARTHENOS aggregative infrastructure. Finally, Section 2.6 has been updated with information about the technology change that occurred in the Shared Workspace Service, for which the previous core component (Home Library) was replaced by a new one named StorageHub.



Abbreviations

ABAC	Attribute Based Access Control
API	Application Program Interface
BLOB	Binary Large Objects
CCF	Content Cloud Framework
CRUD	Create Read Update Delete
GIS	Geographic Information System
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting
RBAC	Role Based Access Control
RI	Research Infrastructure
REST	Representational State Transfer
ROA	Resource Oriented Architecture
SAML	Security Assertion Markup Language
SDI	Spatial Data Infrastructure
THREDDS	Thematic Real-time Environmental Distributed Data Services
VRE	Virtual Research Environment
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
XACML	eXtensible Access Control Markup Language
XSLT	Extensible Stylesheet Language Transformations



1. Introduction

The PARTHENOS e-infrastructure architecture consists of a hardware layer and a service layer. The hardware layer is organized as a dynamic pool of virtual machines, supporting computation and storage, while the services layer is organized into e-infrastructure middleware, storage, and end user services. The hardware layer consists of an OpenStack installation, supporting the deployment of services in the upper layer by provision of computational and storage resources. The service layer, illustrated in Figure 1, consists of five service frameworks, which can be summarized as follows:

- *Enabling Framework*: the enabling framework includes services required to support the operation of all services and the VREs supported by such services. As such it includes: a *resource registry* service, to which all e-infrastructure resources (data sources, services, computational nodes, etc.) can be dynamically (de)registered and discovered by user and other services; *Authentication and Authorization* services, as well as *Accounting Services*, capable of both granting and tracking access and usage actions from users; and a VRE manager, capable of deploying in the collaborative framework VREs inclusive of a selected number of “applications”, generally intended as sets of interacting services;
- *Storage Framework*: the storage framework includes services for efficient, advanced, and on-demand management of digital data, encoded as: files in a distributed file system, collection of metadata records, and time series in spatial databases; such services are used by all other services in the architecture, exception made for the enabling framework;
- *Content Cloud Framework*: the content cloud framework includes all services required to collect, transform, harmonize, and provide via APIs of different kinds all metadata records of interest to the PARTHENOS community and provided by data sources managed by the organisations in the PARTHENOS consortium. The data collection and provision activity are ruled by workflows, configured by data curators (e.g. transformation rules) and orchestrated by a local enabling layer, in order to keep the content cloud up to date with respect to the content available in the aggregated data sources;
- *Analytics Framework*: the analytics framework includes the services required for running methods provided by scientists taking advantage, in transparent way, of the power of the underlying computation cloud (e.g. parallel computation) and of a plethora of standard statistics methods, provided out of the box to compute over given input data;
- *Collaborative framework*: the collaborative framework includes all VREs deployed by the scientists and for each of them provides *social networking* services, *user management* services, *shared workspace* services, and WebUI access to the information cloud and to the analytics framework, via *analytics laboratory services*.

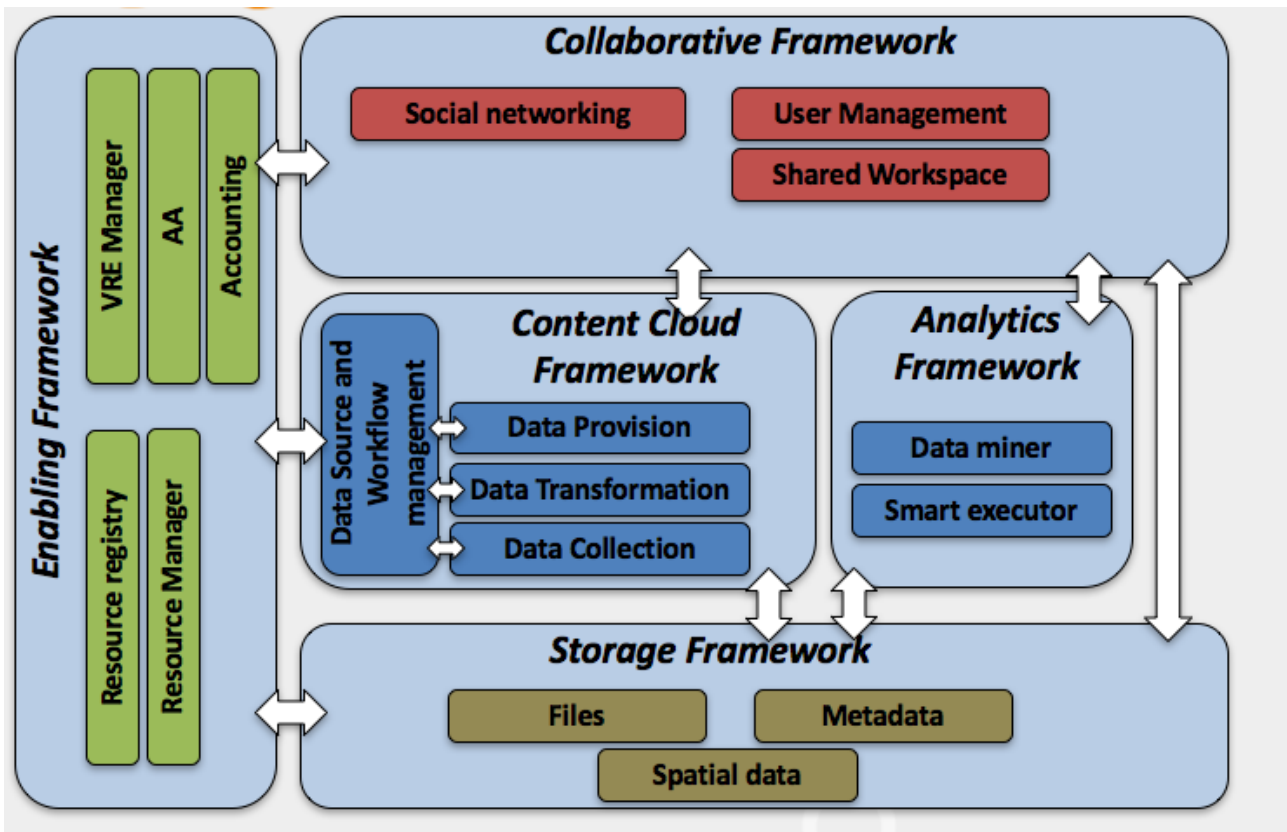


Figure 1. High-level architecture of the PARTHENOS infrastructure

1.1. Structure of this report

This report is structured into two sections: Section 1 is the introduction and Section 2 is about the Cloud Infrastructure, which concludes the report. In more detail, Section 2 is organized into six subsections: section Enabling Technology, for the hardware (storage and computation) layer, and sections Enabling Framework, Storage Framework, Analytics Framework, and Content Cloud Framework, and Collaborative Framework for the service layer.



2. Cloud Infrastructure

The PARTHENOS e-infrastructure architecture consists of a hardware layer and a service layer.

The hardware layer is organized as a dynamic pool of virtual machines, supporting computation and storage. The operations and management of those resources is performed via a set of **enabling technologies** selected to ensure availability and reliability of the infrastructure while guaranteeing reduction of costs of ownership and a set of **supporting technologies** selected to ensure secure monitoring, alerting and provisioning.

The services layer is organized into layered software frameworks that increasingly hide the complexity of the cloud-based infrastructure.

2.1. Hardware Layer

2.1.1. Enabling Technology

The following well-known technologies have been selected to manage the PARTHENOS e-infrastructure hardware resources:

- a. Ceph, <http://www.ceph.com>, has been selected as block storage since it is Amazon S3 compatible and OpenStack Swift compatible, it is completely distributed, and it may even use disposable server hardware;
- b. Openstack, <http://www.openstack.org>, has been selected as cloud-computing software platform. It uses Ceph as storage;
- c. ManageIQ, <http://manageiq.org>, has been selected to manage quotas, permissions, production vs. development environments.

The **Ceph Storage** offers object, block, and file storage under a unified system. It has been designed to provide excellent performance, reliability and scalability. It supports rapid provisioning of massively scalable cloud storage and enables computationally intensive workloads. It provides access to the storage via application written in Java, Python, Ruby, C, etc. It scales to Petabytes and it offers linear scaling with linear performance increase.



The Openstack, open source cloud computing platform, provides Infrastructure-as-a-Service (IaaS). OpenStack lets the PARTHENOS Enabling Framework deploy virtual machines and other instances that handle different tasks on the fly. It makes horizontal scaling affordable, which means that services that benefit from running concurrently can easily serve more or fewer tasks – issued either by users or by other services - on the fly by just spinning up more service instances.

The **ManagelQ** open source platform is a management framework for infrastructure integrating resources from several data centres. It has been designed to manage small and large infrastructure, and supports private data centres exploiting virtual machines and even public clouds. ManagelQ supports continuous monitoring of the latest state of the infrastructure, simplifies the enforcement of policies across the environment, and it optimizes the performance and utilization of the hardware resources.

2.1.2. Supporting Technology

2.1.2.1. Monitoring and Alerting System

The PARTHENOS e-infrastructure currently comprises 212 servers. This means that neither all of them are exploited at the same time nor that all of them have to be active concurrently to deliver specific service capabilities. Servers are allocated dynamically in accordance with the Cloud-computing approach and are activated/deactivated in response to load, failures, changes in policies and deployment strategies. This complexity requires a proper monitoring infrastructure to check the servers and the services running on the servers and to issue alerts when failures are identified. The PARTHENOS e-infrastructure exploits two well-known technologies to perform this task: Nagios and Prometheus.

Nagios is an enterprise-class monitoring and alerting solution that provides extended insight of the infrastructure, enabling quick identification and resolution of problems before they may affect critical business processes. It provides monitoring of all mission-critical infrastructure components including applications, services, operating systems, network protocols, systems metrics, and network infrastructure. Nagios provides a central view of operations, network, and business processes running on the infrastructure. Powerful dashboards provide at-a-glance access to powerful monitoring information and third-party data. Views provide users with quick access to the information they find most useful, and help them spot problems easily with advanced data visualization reports. Moreover, alerts



are sent to infrastructure managers and the Parthenos quality assurance task force via email or mobile text messages, providing them with outage details so they can start resolving issues immediately. Finally, multiple APIs provide for simple integration with in-house and third-party applications. In particular, for well-known technologies exploited in the PARTHENOS e-infrastructure, e.g. MongoDB, Cassandra, Couchbase, PostgreSQL, etc., existing add-ons have been installed to extend monitoring and native alerting functionality; for technologies developed both by PARTHENOS and by the exploited framework, i.e. gCube and D-Net, specific add-ons have been designed, implemented, and installed to extend monitoring and native alerting functionality in order to have a fully-complete and always up-to-date image of the status of the PARTHENOS e-infrastructure. Overall 2,194 service checks have been added and continuously executed to the monitoring and alerting infrastructure.

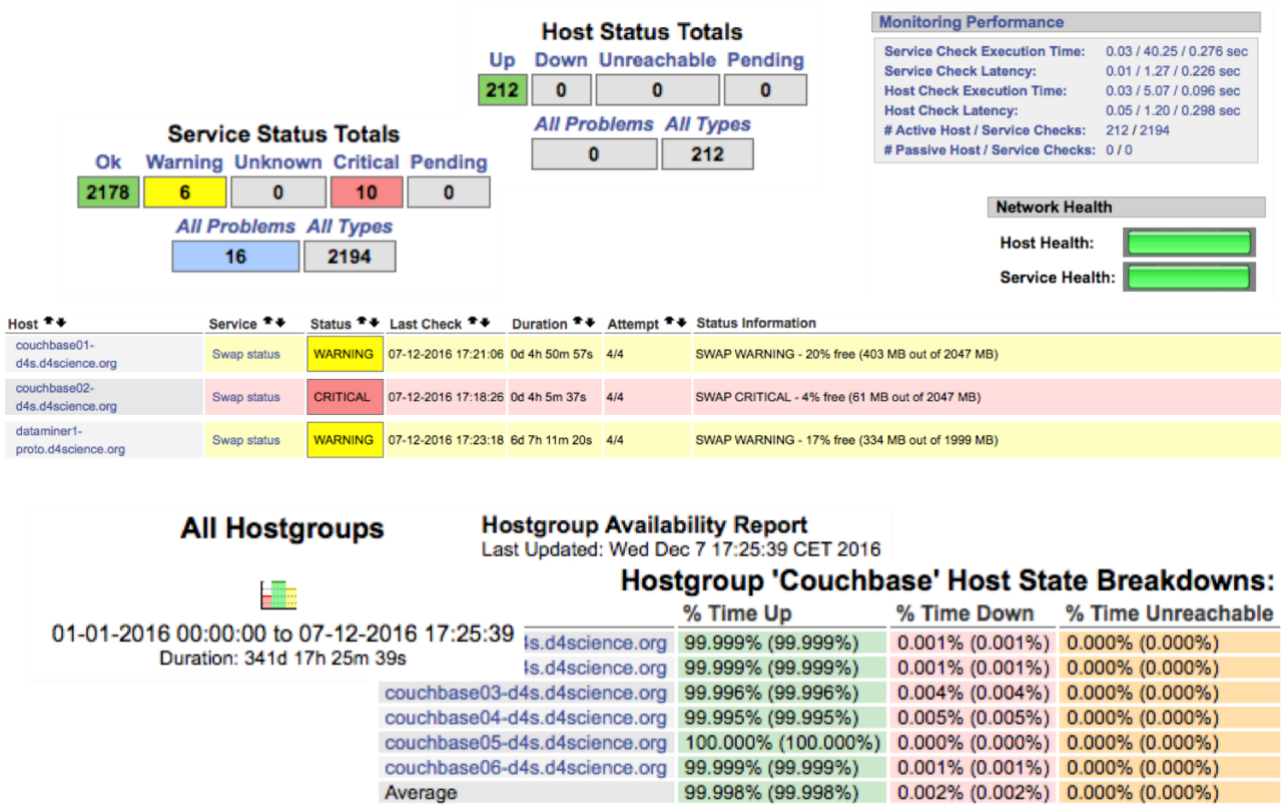


Figure 2. Nagios Status Report and Availability Report for the Accounting Cluster



Prometheus is an open-source system monitoring and alerting toolkit originally built at SoundCloud¹. It is a standalone open source project and maintained independently of any company. Prometheus's main features are:

- a multi-dimensional data model with time series data identified by metric name and key/value pairs;
- time series collection happens via a pull model over HTTP;
- PromQL, a flexible query language to leverage this dimensionality;
- no reliance on distributed storage, single server nodes are autonomous;
- pushing time series is supported via an intermediary gateway;
- targets are discovered via service discovery or static configuration;
- multiple modes of graphing and dashboarding support.

The multiple modes of graphing and dashboarding support feature has been exploited by adopting Grafana², which allowed to query, visualise, alert on and understand Prometheus data on metrics.

Grafana allows dashboards to be virtually defined as a set of servers that collectively perform a specific task. In the PARTHENOS e-infrastructure we defined a catch-all dashboard to include all servers and then specific virtual clusters to monitor the performance and the exploitation of physical resources for the key enabling software frameworks exploited in the infrastructure and reported in Section 2.2 and subsequent sections.

¹ <http://soundcloud.com/>

² <https://grafana.com/grafana>

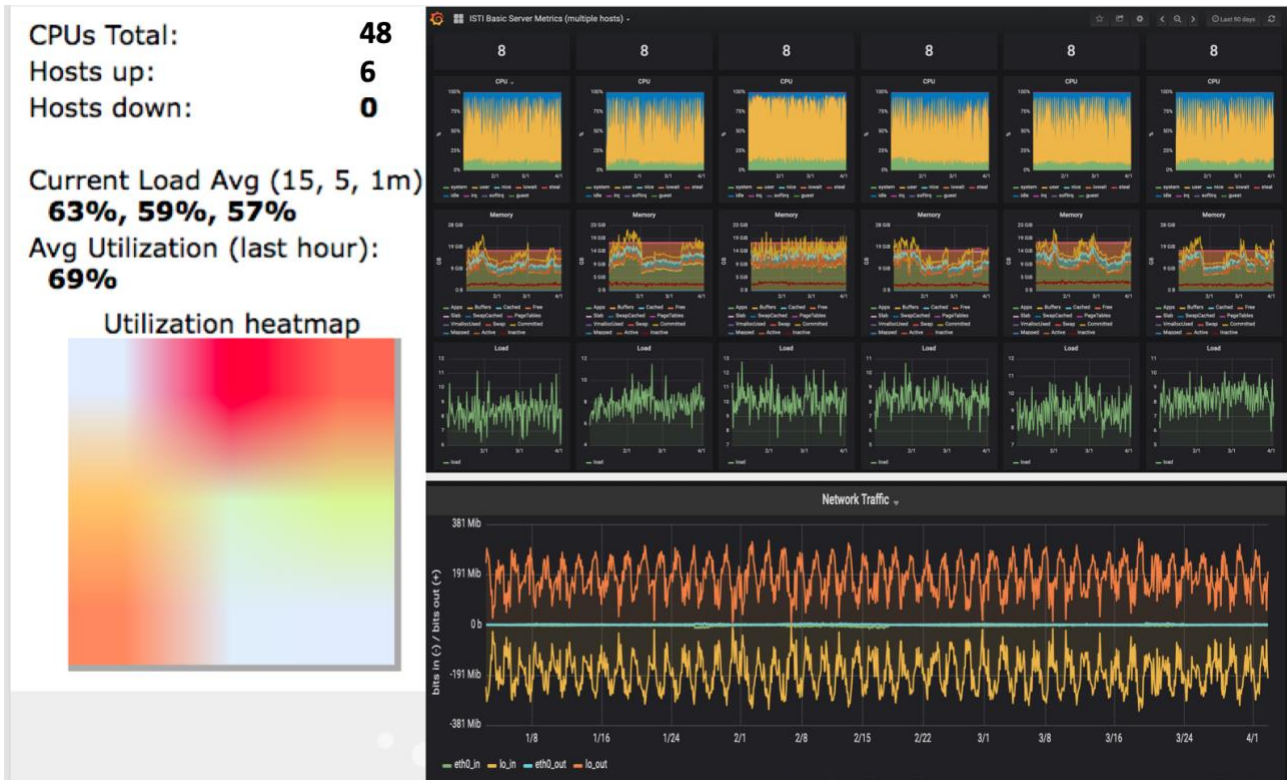


Figure 3. Prometheus Aggregated View via Grafana for the Accounting Cluster

2.1.2.2. Provisioning System

The PARTHENOS e-infrastructure currently comprises 212 servers and one of the core ambitions in designing it was the reduction of the deployment, operation, and maintenance costs. To achieve this ambition a key aspect was to automatize the configuration and management of servers, combining multi-servers software deployment, supporting ad hoc task execution, and configuration management.

Ansible is a free-software platform allowing configuration of servers according to *idempotence*. Idempotency is basically based on the description of what state is required on a server and Ansible figures out how to get to that state. This approach is opposite to other approaches that require specification of what to run on a server and how to run it. This allows drastic reduction of the costs of operations since it becomes possible to run Ansible plays over and over and it does the right thing according to the status of the server instead of repeating commands and configurations. Ansible is really useful for repeatedly setting up servers in the Cloud which need to be set up the same way.



In order to exploit Ansible in the PARTHENOS e-infrastructure, definition was needed of a number of resources and configuration scripts that are used by Ansible to perform the activities:

- inventories - list of servers to configure and maintain;
- playbooks - collection of plays, or simply a collection of roles for a 1-play playbook;
- plays - a collection of roles;
- roles - generally, one service (like postgres or nginx);
- tasks - a command that Ansible runs via its modules, like a task for installing a package via apt-get;
- handlers - like tasks that get called when other tasks request them via notifications. Typically used to restart services;
- host vars - variables that apply to one collection of hosts;
- modules - provided by Ansible to do things like configure MySQL (mysql module), install via apt-get (apt module), copy over files (file module), add users (user module).

Overall, to manage the PARTHENOS e-infrastructure, 197 roles have been defined.

2.2. Enabling Framework

2.2.1. Overview

The Enabling Framework is realized by a combination of services and libraries powered by the gCube System open-source project. Those services promote the optimal exploitation of the resources available in the PARTHENOS Cloud Infrastructure and the integration of technology external to it. They insulate, as much as possible, the management of the infrastructure from the data and the data management services that are hosted in or accessible through the infrastructure itself.

The motto at the heart of the management facilities is *'less dependencies for more management'* meaning that the requirements posed to resources (even independent resources) to be managed are minimal, close to zero in some cases. All the implemented solutions are prioritized in order to pursue this goal.

In order to comply with the new directions of openness and interoperability called for by our growing community, management facilities are implementing:

- adoption of standards;
- support for new software platforms by implementing a zero-dependency approach to software management.

The Enabling framework is composed of three main systems: Resource Management System, Information System, and Security System. These are complex ICT systems that exploit tailored persistence technologies managed via web services.

- The **Resource Management System** supports the creation of a Virtual Research Environment and its exploitation via the registration, management, and utilization of the resources assigned to it.
- The **Information System** supports the registration, discovery, and access of the resources profile.
- The **Security System** ensures the correct exploitation, auditing, and accounting of the resources under the policies defined at registration time and customized at VRE definition time. It is orthogonal to all services operating in the infrastructure and its components are deployed on all computing nodes.

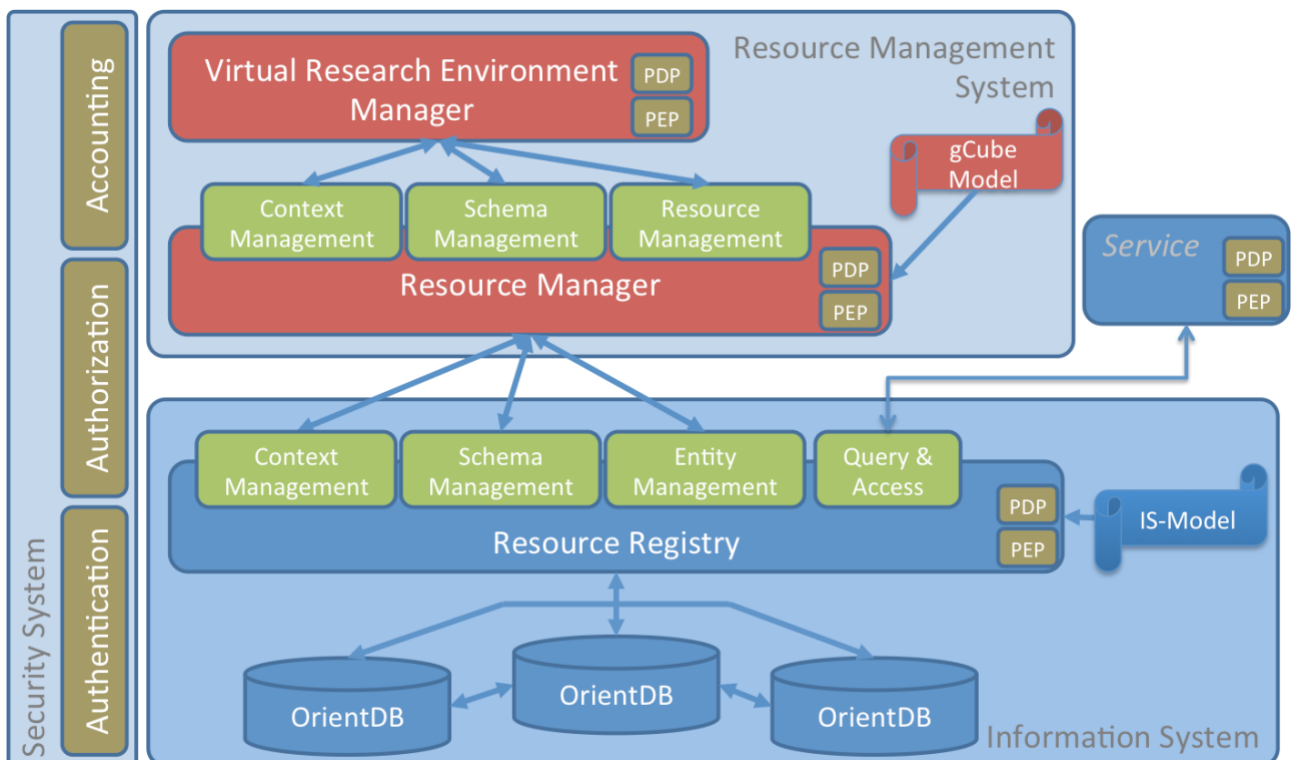


Figure 4. Enabling Framework Architecture



2.2.2. Key Features

Extensible notion of resource	A resource model which is open to modular extensions at runtime by arbitrary third parties.
Transparent software resource management	Nearly zero-dependency requested to managed resources for being part of the infrastructure.
Environment propagation	Operational information among services are transparently propagated over a range of protocols (SOAP, HTTP/S, and more).
Dynamic Deployment and Optimal Resource (re)Allocation	Remote deployment and (re-)configuration of resources across the infrastructure.
Resource lifetime management	Complete running of the entire lifetime of resources ranging from creation and publication to discovery, access and consumption.
Self-elastic management	Dynamic resource provisioning to meet peaks and lows in demand.
Interoperability, openness and integration at software level	Third-party software can be added to the infrastructure at runtime.
Support to standards	Crucial functionalities are accessible via recognized standards in order to enhance interoperability.

2.2.3. Subsystems

2.2.3.1. Resource Registry

The **gCube Resource Registry** is the core subsystem connecting producers and consumers of resources. It acts as a registry of the infrastructure by offering global and partial views of its resources and their current status and notification instruments. The approach provided by the Resource Registry is of great support for the dynamic allocation of resources and the interoperability solutions offered by the Resource Manager system. The feedback obtained during the first reporting period has been used to improve the quality of the design, the APIs of both services and clients, the design of the Graphical User Interfaces (GUIs), and the REST APIs (to strictly adhere to REST principles). Furthermore, the client's APIs has been simplified and enriched: two new Java clients have been released: Resource Registry Context Client and Resource Registry Schema Client which now makes a total of four Java Clients:



- Resource Registry Context Client;
- Resource Registry Schema Client;
- Resource Registry Publisher;
- Resource Registry Client.

Key Features

Resource Publication, Access and Discovery	The Resource Registry is functionally complete offering Java and WEB APIs to register new resources, to discover, and access them.
Consistency with the new Resource Model	The Resource Registry grants publication and access to resources compliant with the Resource Model.
Production level QoS - Responsiveness	Each query served in milliseconds, thousands of queries served each hour.
Production level QoS - Scalability	Infrastructures with more than 100K of resources successfully powered.
Production level QoS - Permanent and Uninterrupted Functioning	The Resource Registry instances have been continuously up for more than one year without human intervention.
Flexible deployment scenarios	The Resource Registry components can be deployed in several ways, to best fit the needs of the infrastructure or a specific community.

Architecture

The design of the Resource Registry supports distribution and replication wherever it is possible while abstracting clients from the deployment scenario. It exploits HAProxy for proxying requests to the deployed instances of the Resource Registry web service. **HAProxy** is a free, very fast and reliable solution offering high availability and load balancing for very high traffic web applications. Over the years it has become the de-facto standard open-source load balancer and it is now shipped with most mainstream Linux distributions. For these reasons, it is deployed by default in the PARTHENOS Cloud Infrastructure.

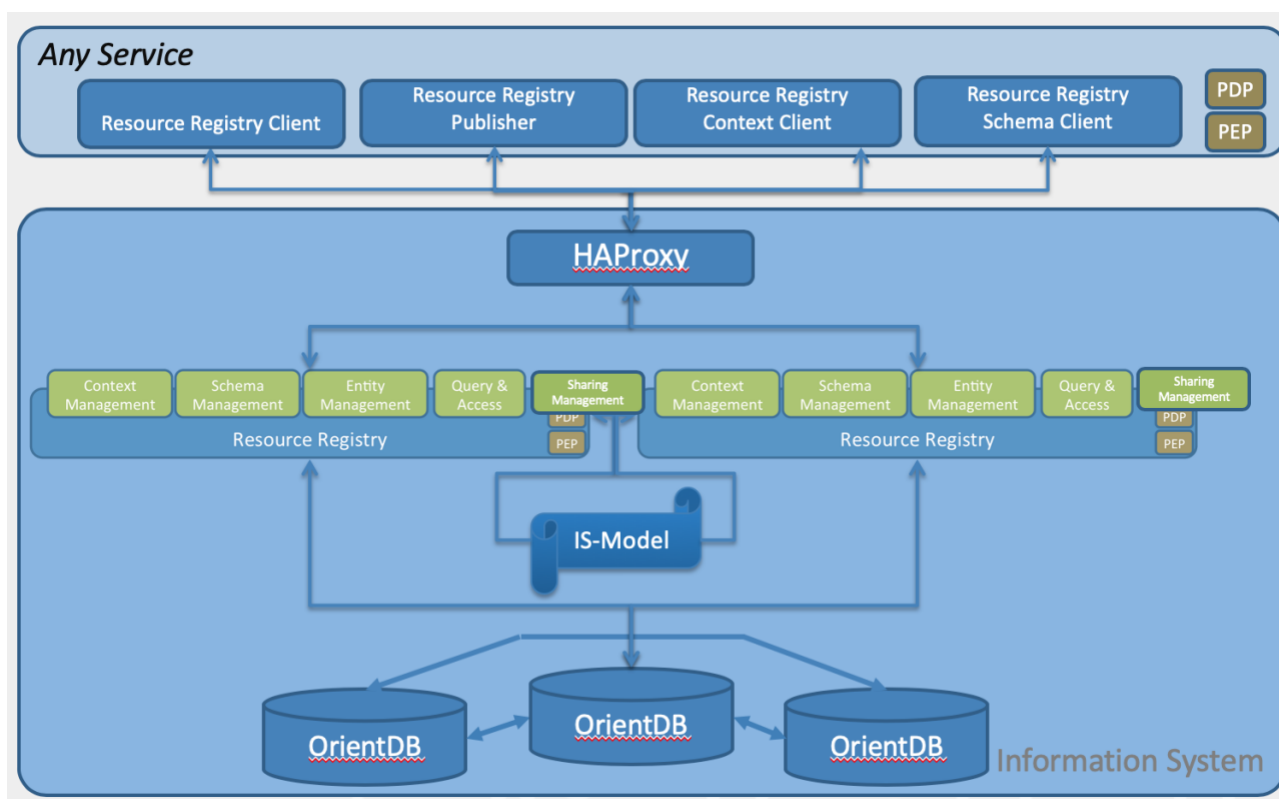


Figure 5. Resource Registry Architecture

The Resource Registry web service has now five port-types, each responsible for:

- **Context Management:** manage hierarchical contexts;
- **Types Management:** manages the definition of entities and relations types and their schema. This choice allows for easy extension and support modification to the resource model. This is the key factor for the sustainability of the service and infrastructure that have to last for several years;
- **Instances Management:** manage instances of registered Entity and Relation type;
- **Sharing Management:** manages instances sharing across different contexts;
- **Query & Access:** query instances and get the schema definition of registered types.

Every port-type is exposed with a REST³ API. **REST** is an excellent architectural style to support scalability of service while keeping the complexity of design, implementation, and deployment at very affordable costs. During the last decade, REST has emerged as a best practice to design web services. For this reason, REST has guided the design of the JRR. REST is an architectural style defined in 2000 by Roy Thomas Fielding. REST defines six principles and four constraints but it does not provide any concrete guidelines or architecture. An example of concrete architecture for REST is ROA (Resource Oriented

³ <https://en.wikipedia.org/wiki/JSON>



Architecture) which is based on HTTP 1.1. The design of the Resource Registry service follows the ROA guidelines. In particular, every REST API is JSON⁴ based. This means that any content present in an HTTP request is formatted using the JSON standard. The Resource Registry web service is stateless making it possible to replicate it horizontally.

2.2.3.2. Resource Manager

The Resource Manager is responsible for providing Resources compliant with the gCube-Model. In fact, this service is the only one entitled to perform operations on the Resource Registry. It does so by exposing three port types:

1. Context Management enables Resource Registry context management by checking if the requester has the proper role/rights to do the requested action.
2. Schema Management enables schema management on the Resource Registry by checking if the requester has the proper role/rights to do it;
3. Resource Management: enables management of Resource instances by checking if:
 - the requester has the proper role/rights to do the requested action;
 - the action can be performed looking at the policies attached to the entities and relation instances;
 - the action involves other entities or relations.

When all these checks are performed, and if and only if the action is feasible, the Resource Manager translates the incoming request in one or more outgoing requests to the Resource Registry service.

Key Features

Resource Publication, Access and Discovery	The Resource Manager offers Java and WEB APIs to register new resource types and instances.
Consistency with the gCube Model	The Resource Registry grants publication and access to resources compliant with the gCube Model at Resource level.

⁴ <https://en.wikipedia.org/wiki/JSON>

Architecture

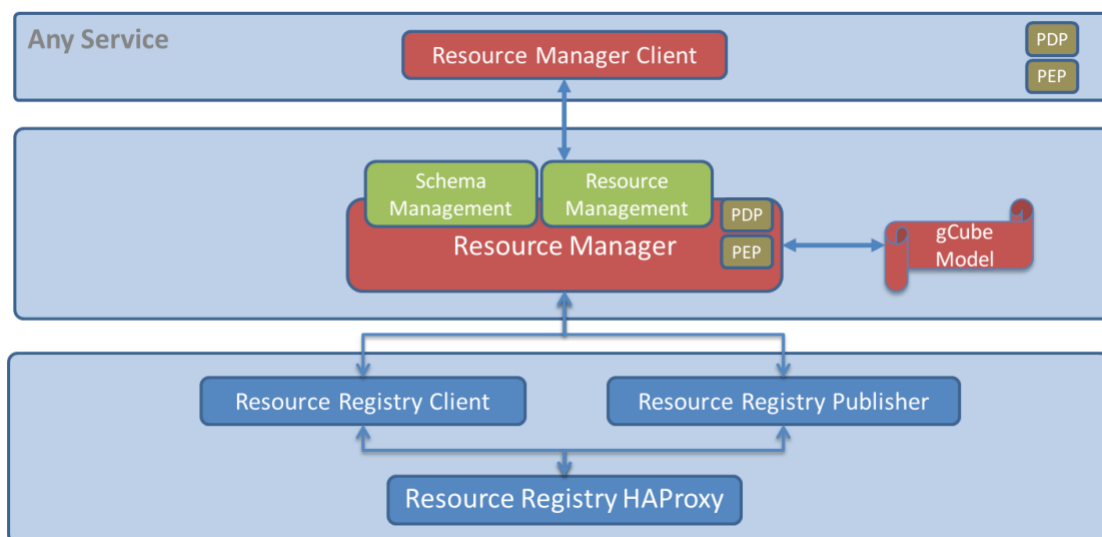


Figure 6. Resource Manager Architecture

As depicted in Figure 6. Resource Manager Architecture, the Resource Manager uses the Resource Registry Client to query the Resource registry and get the actual knowledge of the infrastructure.

When the Resource Manager receives a request, once it has performed the proper checks, it uses the Resource Registry publisher to effect the request.

Both Resource Registry Client and Publisher interact with one of the instances of Resource Registry through HA-Proxy.

2.2.3.3. Virtual Research Environment Manager

The VRE Manager is responsible for providing context guarantees based on the gCube-Model.

The VRE Manager operates on the PARTHENOS Cloud Infrastructure by using components of:

- the enabling technologies such as Resource Manager;
- supporting technologies such as Provisioning System.

The VRE Manager contacts the Resource Registry to get a current view of the infrastructure; uses the provisioning system to deploy/remove services and data; asks the Resource Manager to update the infrastructure state consistently.

Key Features

Context Management	The VRE Manager offers Java and WEB APIs to create, edit, and delete security context, i.e. Virtual Research Environment.
Consistency with the gCube Model	The VRE Manager grants publication and access to resources compliant with the gCube Model at context level.

Architecture

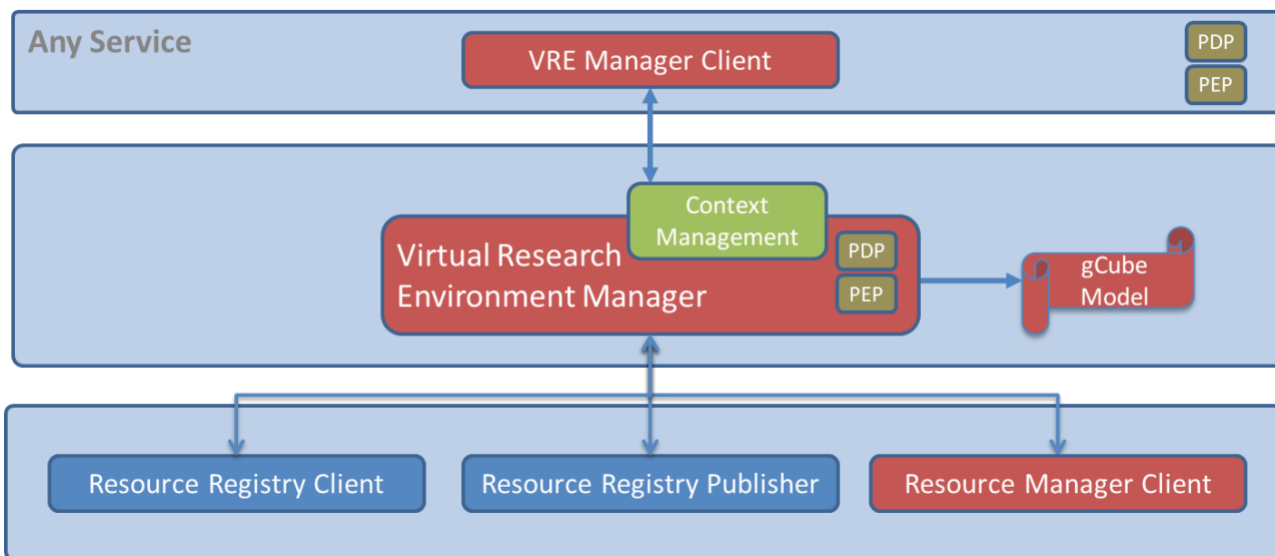


Figure 7. VRE Manager Architecture

As shown in Figure 7, the VRE Manager uses the Resource Registry Client to query the Resource Registry and get the actual knowledge of the infrastructure resources. By exploiting this information, the VRE Manager provides the support for the creation of the VRE. It creates a new security context and registers it in the Resource Registry. Then, it creates a secure symmetric key to enable encrypted conversion in the newly created security context. Finally, it interacts with the Resource Manager to allocate infrastructure resources to the newly created security context.

During the VRE lifetime, when the VRE Manager receives requests for VRE modifications, once it has performed the proper checks, it interacts with the Resource Manager to either edit, modify, or delete a Virtual Research Environment.

2.2.3.4. Authentication and Authorization

The goal of the Policy-oriented Security Facilities is to protect PARTHENOS Cloud Infrastructure resources from unauthorized accesses.



Service Oriented Authorization and Authentication is a security framework providing "security services" as web services, according to the "Security as a Service" ("SecaaS") research topic. It is based on standard protocols and technologies, providing:

- an open and extensible architecture;
- interoperability with external infrastructures and domains, obtaining, if required, also so-called "Identity Federation";
- total isolation from the enabling framework and technologies: zero dependencies in both the directions.

The Policy-oriented Security Facilities are powered by the gCube Authorization framework. The **gCube Authorization framework** is a token-based authorization system. The token is a string generated on request by the Authorization service for identification purposes and associated with every entity interacting with the 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 service in three ways:

- 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 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.

This framework is compliant with the Attribute-based Access Control (ABAC) that defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together.

ABAC defines access control based on attributes that describe:

- the requesting entity (either the user or the service);
- the targeted resource (either the service or the resource);
- the desired action (read, write, delete, execute);
- and environmental or contextual information (either the VRE or the VO where the operation is executed).

ABAC is a logical access control model that is distinguishable because it controls access to objects by evaluating rules against the attributes of the entities (requesting entity or target resource) actions and the environment relevant to a request. ABAC relies upon the



evaluation of attributes of the requesting entity, attributes of the targeted resource, environment conditions, and a formal relationship or access control rule defining the allowable operations for entity-resource attribute and environment condition combinations.

The Authorization framework is compliant with the XACML reference architecture. **XACML** is the OASIS standard for fine-grained authorization management based on the concept of Attribute-based access control (ABAC), where access control decisions are made based on attributes associated with relevant entities while operating in a given operational context, a natural evolution from Role Based Access Control (RBAC).

Key Features

Security as a Service	Authentication and Authorization provided by web services called by resource management modules.
Flexible authentication model	The user is not required to have personal digital certificates.
Attribute-based Access Control	A generic way to manage access: access control decisions are based on one or more <i>attributes</i> .
Support for different categories of attributes	User related attributes (e.g. roles) and environment related attributes (e.g. context).
Modularity	Composed of different modules: each module has a well-defined scope and provides well-defined services.
Support to standards	All the operations delivered by the facilities are built atop of recognized standards.
High performance	The design and architectural choices have been made paying great attention to performances.
Resource Usage Tracking	Administrators and users can monitor applications resources usage.

Architecture

The XACML standard proposes a reference architecture with commonly accepted names for the various entities involved in the architecture. The nomenclature is not new (SAML uses similar names to describe entities in its ecosystem), nor is the architecture complicated, allowing for easier common base of understanding of the standard. XACML is composed of five 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 five modules' capabilities are implemented by gCube as follow.

- Policy Administration Point (PAP) is implemented by 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 Resource Registry (Information System);
- Policy Retrieval Point (PRP) is implemented by a database controlled exclusively by the gCube Authorization Service.

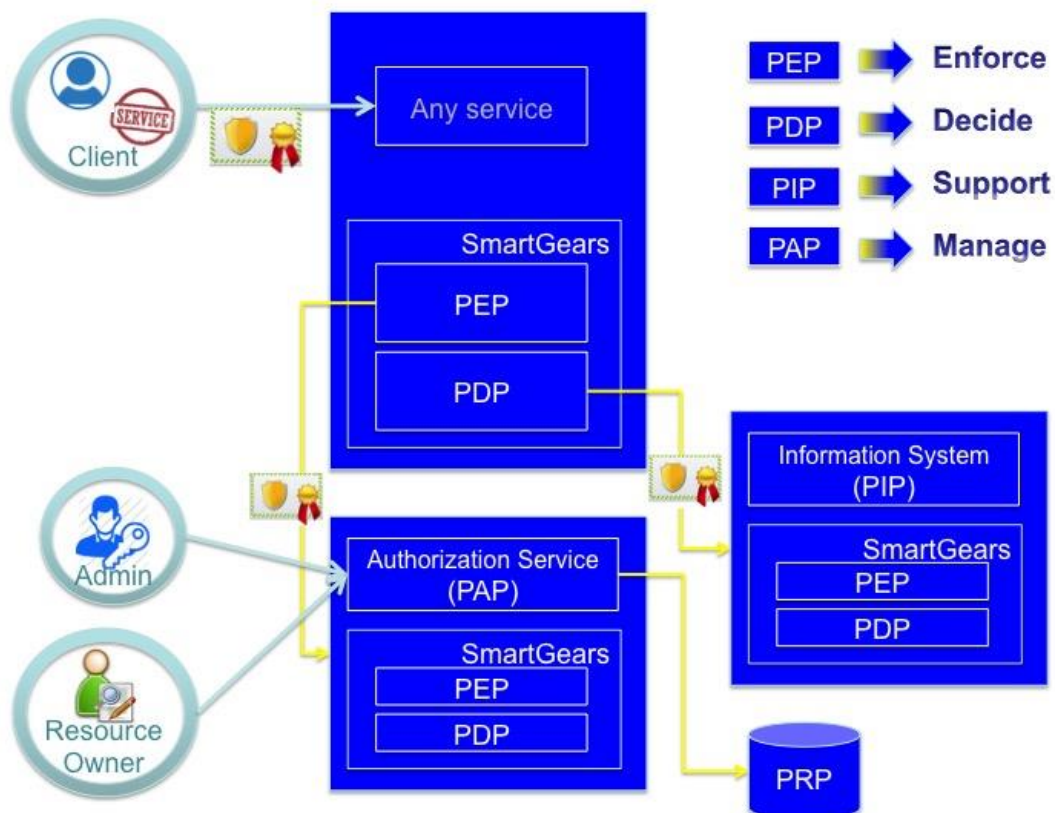


Figure 8. Authorization Architecture



The gCube Authorization Framework controls access to applications to allow or prevent the clients from performing various operations in the applications. This is controlled by the Authorization Service with the help of authorization policies. The purpose of authorization policies is to control clients' access. The authorization policies determine at runtime whether or not a particular action is allowed or denied.

All the policies are used to permit or 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. By default, users are permitted access to the operation in a specific context.

The S2S policies are used to deny to a service or a class of services, the access to a specific service or class of services. To make it easier to allow access only to few clients, an *except* restriction clause can be added to the policies.

For every policy, a specific ACTION, i.e. Access, Write, Delete, and Execute, can be specified (if supported by the service), otherwise all the ACTION will be denied.



The resource owner uses the policy-authoring tool (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 getting back then the business request is passed to the service, otherwise a Deny response is issued.

Highlight 1: flow of control governing the authorization flow.

2.2.3.5. Accounting

Accounting is defined as the recording, summarizing, and classifying of service invocations and other events, e.g. storage of data, systematically. Accountancy, in a simpler sense, is the procedure of communicating and translating raw data from the infrastructure operation to its managers and stakeholders.

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 comprise a large number of components clearly separating the functional constituents.
Multiple options for storage	The subsystem can rely on an array of diverse solutions for actually storing records.

Architecture

The gCube Accounting architecture is logically divided in four different layers:

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

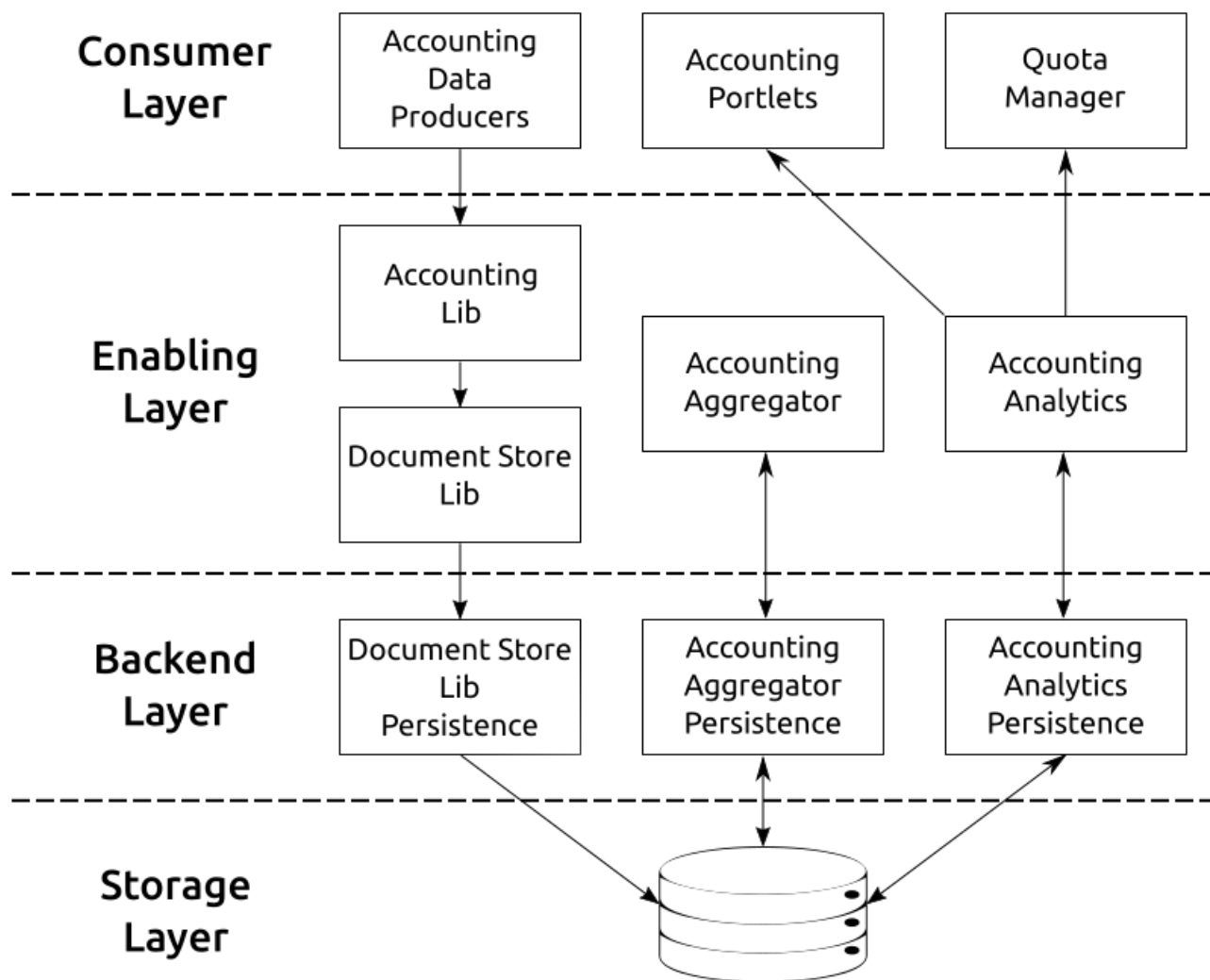


Figure 9. Accounting Architecture



All the component respect a set of common rules adopted to ensure high-availability, fully-distributed operations, low-operation costs:

- Each enabling layer has its own correspondent back-end implementation;
- Each back-end implementation is dynamically discovered at run-time. This allows to decouple the deployment of a different back-end from the development of the enabling layer. In other words, each component on the enabling layer must not have any dependency on a certain back-end implementation.

Accounting Enabling Layer

- The **Accounting Lib** collects, harmonizes and stores accounting data. It is mainly based and developed exploiting the facilities provided by the Document Store Lib.
- The **Accounting Analytics** exposes a common access point interface to query the collected accounting data.
- The **Accounting Aggregator** aggregates the collected Accounting data according to dynamically defined policies. The PARTHENOS e-infrastructure accounting policies have been defined to incrementally aggregate past accounting data without loss of information

Accounting Storage Layer

This layer is not developed by gCube. Rather it relies on technologies guaranteeing HA (High Availability). In the current settings, it is implemented by relying on **CouchBase**. Other supported backend technologies are CouchDB and MongoDB.

Accounting Backend Layer

Each component in this layer has been explicitly developed over a certain storage technology. They rely on the Resource Registry to discover the information needed to connect to the underlying storage. In other words, each component does not have hard-coded connection information or local configuration files. This approach allows retrieval of the storage connection information by specifying the underlying storage technology and the enabling component to use.

The first filter allows switching to a different storage backend at runtime and it supports the co-existence of different storage backends – particularly useful to migrate from one storage type to another without any downtime.



The second filter allows the connection information for each component to be kept separate. This allows the support of tailored access policies for each component, e.g. write-only for accounting-lib connection and read-only for accounting-analytics.

The **document-store-lib-BACKEND** supports the connection to and the storage of accounting data to the technology selected as the persistence layer. It has been implemented to support the three underlying technologies: document-store-lib-couchdb, document-store-lib-couchbase, document-store-lib-mongodb;

The **accounting-analytics-persistence-BACKEND** supports the connection to and the discovery and access of accounting data to the technology selected as the persistence layer. It has been implemented to support the three underlying technologies: accounting-analytics-persistence-couchdb, accounting-analytics-persistence-couchbase;

The **accounting-aggregator-persistence-BACKEND** supports the connection to and the aggregation of accounting data to the technology selected as the persistence layer. It has been implemented to support the three underlying technologies: accounting-aggregator-persistence-couchdb, accounting-aggregator-persistence-couchbase;

Accounting Consumer Layer

Each component in this layer allows either producing or consuming accounting information. It does not include only a graphical interface designed for managers, i.e. **Accounting Portlet**. Rather, it includes all the components that collect accounting data as the Quota Manager, currently in development stage.

2.3. Storage Framework

2.3.1. Overview

The Storage framework is realized by a combination of services and libraries powered by the gCube System open-source project. It is composed of three main systems: File-Based System, Metadata Store System, and Spatial Data Repository System. These act as main drivers for clients that interface the storage resources managed by the system or accessible through facilities available within the system.



- The File-Based System supports functions for **standards-based** and **structured access** and **storage of files** of arbitrary size.
- The Metadata Store System supports functions for the **storage of metadata** objects in XML format.
- The Spatial Data Repository System is composed by a number of different spatial data repositories for storing **spatial data** in different (standard) formats (e.g. NetCDF, vector data, raster data etc.).

2.3.2. Key Features

Standards compliancy	Support for standard communication protocols / interfaces and data / metadata formats.
Economy of scale	Services constituting one aggregative infrastructure may be hosted over servers maintained at different sites
Failover Management	Automatically transfers control to a duplicate computational node when faults or failures are detected
Support of geospatial dataset lifecycle	Support for generation, revision, publishing, access, visualization and sharing of geospatial data.
Support for analysis and processing	Support for high performance operations over datasets
Geo-referencing datasets	Provide analysis tools to create standard spatial representation of datasets

2.3.3. Subsystems

2.3.3.1. File-Based Store System

The File-Based Store system includes services providing clients functions for standards-based and structured access and storage of files of arbitrary size. This is a fundamental requirement for a wide range of system processes, including indexing, transfer, transformation, and presentation. Equally, it is a main driver for clients that interface the resources managed by the PARTHENOS infrastructure or accessible through facilities available within the same infrastructure.

The File-Based System is composed of a service abstracting over the physical storage and capable of mounting several different store implementations, (by default clients can make

use of the MongoDB store) presenting a unified interface to the clients and allowing them to download, upload, remove, add and list files or unstructured byte-streams (binary objects). The binary objects must have owners and owners may define access rights to files, allowing private, public, or shared (group-based) access.

All the operations of this service are provided through a standards-based, POSIX-like API which supports the organization and operations normally associated with local file systems whilst offering scalable and fault-tolerant remote storage

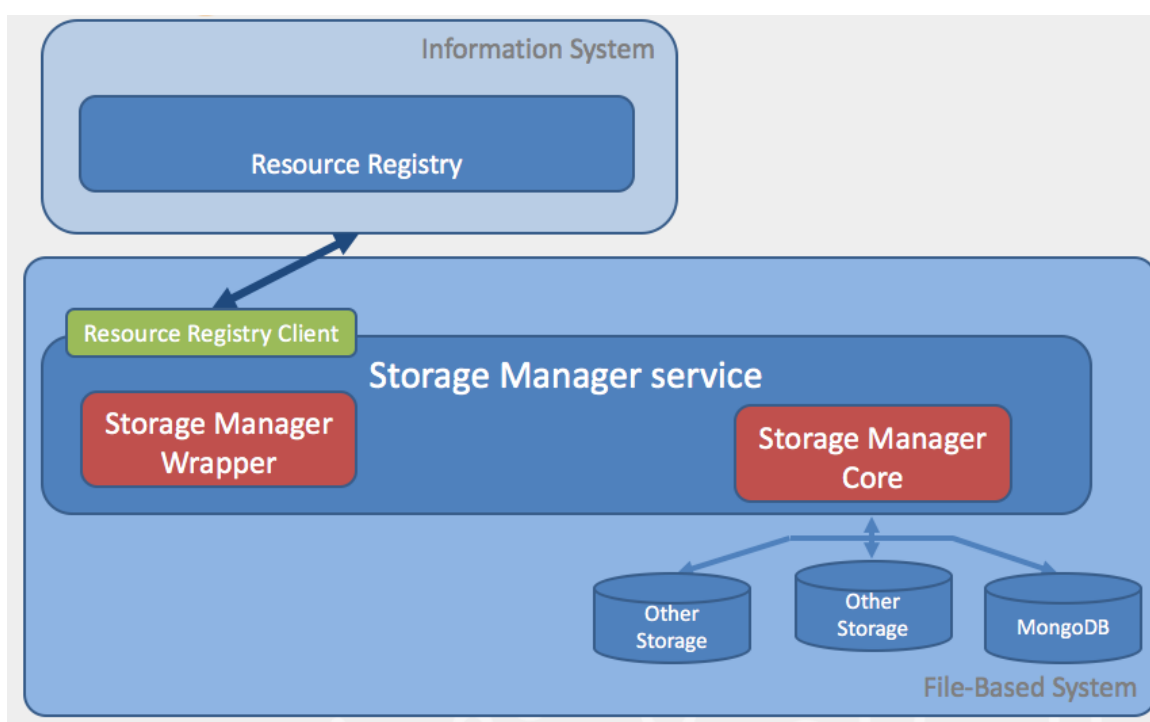


Figure 10. File-Based System Architecture

As shown in Figure 10, the core of the Storage Manager service is a software component named Storage Manager Core that offers APIs allowing abstraction over the physical storage. The Storage Manager Wrapper, instead, is a software component used to discover back-end information from the Resource Registry service of the PARTHENOS Infrastructure. The separation between these two components is necessary to allow the usage of the service in different contexts other than the PARTHENOS Infrastructure.



2.3.3.2. Metadata Store System

The Metadata Store system includes services for the storage of metadata objects in XML format. The core service is the MDStore Service, a web service that implements the factory pattern for the management of MDStore units.

A **MDStore** unit is a metadata store capable of storing metadata objects of a given metadata data model. Consumers can create and delete units, and add, remove, update, fetch, get statistics on metadata objects from/to a given unit via the MDStore Service. The Service is implemented as an abstraction over the document-oriented storage MongoDB in order to exploit its high-scalability and replica management features, but also to take advantage of out-of-the-box support with the Hadoop Map-Reduce framework, if necessary.

2.3.3.3. Spatial Data Repositories

The Spatial Data Repositories include services, technology, policies and practices designed in order to provide the following features:

- **Data Discovery:** the ability to browse, query and access metadata files about accessible geospatial datasets. This feature is obtained exploiting **GeoNetwork** webservice, the Open Source catalogue for geospatial metadata compliant with standards mandated by Open Geospatial Consortium (OGC).
- **External Repository Federation:** transparently extend the Data Discovery capabilities by including output from registered external catalogues and repositories in order to give users global result from a single point.
- **Data Access & Storage:** provide users and applications to access/store geospatial data in standard formats. Due to the heterogeneity of spatial data representation and formats, the following technologies have been adopted:
 - **PostGIS:** Geospatial extension of **PostgreSQL** relational DBMS;
 - **GeoServer:** Open Source application for management and dissemination of geospatial data through standards mandated by OGC;
 - **Thredds Data Server:** Unidata's Thematic Real-time Environmental Distributed Data Services (THREDDS) is a web server that provides data access for scientific geospatial dataset formats (i.e. NetCDF).
- **Data Processing:** the Data Processing framework includes services designed to perform analysis and transformations over geospatial datasets. The adoption of



52°North Web Processing Service (WPS) grants users a standardized way to interact with Data Processing facilities. This framework is fully described in section 2.4 of this document.

- **Data Visualization:** Web application, named GeoExplorer, designed to render views of overlapped geospatial datasets on a specific Earth projection, with the ability to query / inspect and export selected data and rendered images.

The set of spatial data repositories and the comprehensive set of integrated technologies for their management, discovery, and exploitation is fully integrated with both infrastructure's enabling technology and layers (c.f. Sections 2.1 and 2.2) in order to exploits provisioning, monitoring, accounting, authentication and storage facilities of the infrastructure.

Key Features

Support of geospatial dataset lifecycle	Support for generation, revision, publishing, access, visualization and sharing of geospatial data.
Support for analysis and processing	Support for high performance operations over geospatial datasets
Dataset enrichment and harmonization	Provide tools to harmonize and add information on existing data
Georeferencing datasets	Provide analysis tools to create standard spatial representation of datasets
Standards compliancy	Support for standard communication protocols / interfaces and data / metadata formats.
External repository federation	Gather all available information in one single point.
Policies adoption assurance over third-party technologies	Configuration/orchestration of third party technologies in order to ensure access policy compliancy.
Horizontal scalability	Ability to expand / contract the SDI resource pool in order to accommodate heavier or lighter loads.

Architecture

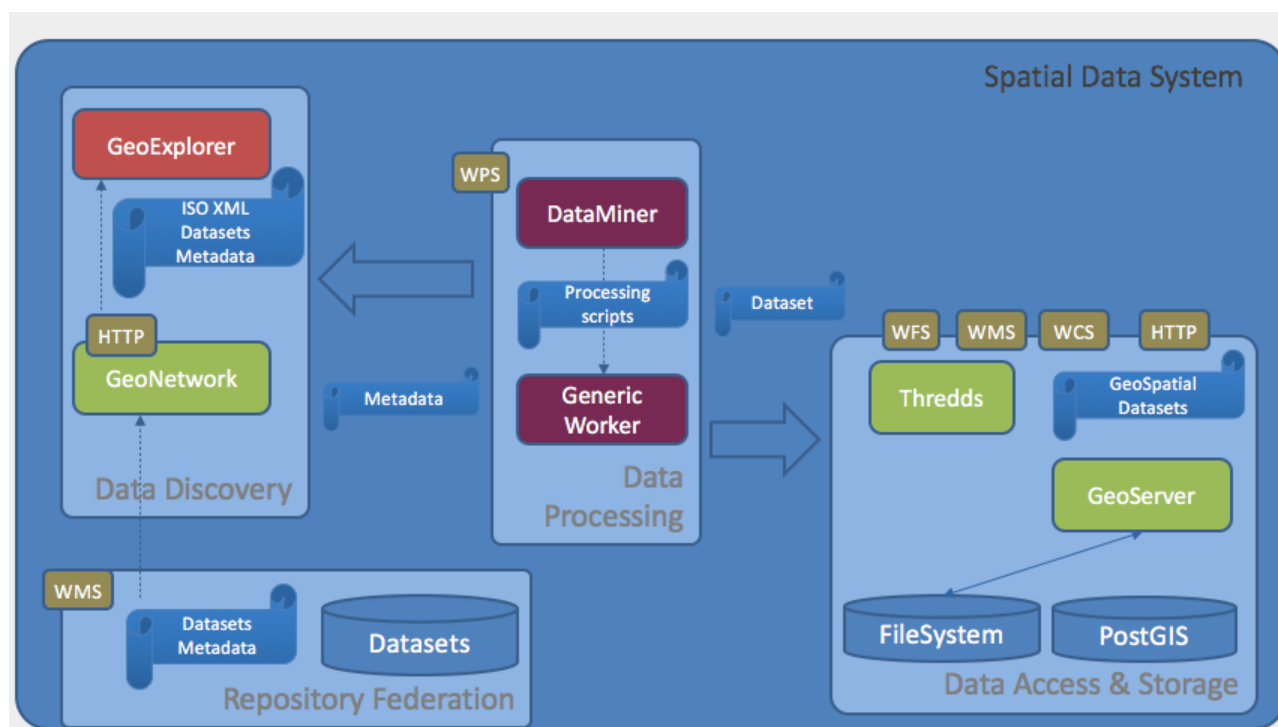


Figure 11. Spatial Data Repositories Architecture

The set of Spatial Data Repositories technologies selected and integrated are not only fully compliant with the Open Geospatial Consortium (OGC) standards, i.e. Web Map Service (WMS), Web Coverage Service (WCS), and Web Feature Service (WFS). Rather, specific mediators and validators have been designed and implemented to respect the INSPIRE Directive, the EU initiative geared to help to make spatial or geographical information more accessible and interoperable for a wide range of purposes supporting sustainable development.

The Data Discovery components allow the discovery at runtime of all available datasets independently of their locations and it is indifferent to the technology used to persist them. It also indifferent to the fact that the data are maintained by a repository managed by the PARTHENOS Cloud Infrastructure or by an independent provider. The same applies also to the Data Processing components that first discover the datasets to process and then are able to process them independently of the technology used to persist them and the provider entitled to manage them.

2.4. Analytics Framework

2.4.1. Overview

The Analytics Framework includes a set of features, services and methods for performing data processing and mining on information sets. These features face several aspects of data processing ranging from modelling to clustering, from identification of anomalies to detection of hidden series. This set of services and libraries is used by the e-infrastructure to manage data mining problems even from a computational complexity point of view. Algorithms are executed in parallel and possibly distributed fashion, using the same e-infrastructure nodes as working nodes. Furthermore, Services performing Data Mining operations are deployed according to a distributed architecture, in order to balance the load of those procedures requiring local resources.

2.4.2. Key Features

Parallel Processing	Support for the execution of algorithms on multi-cores computational nodes.
Distributed Processing	Transparent distribution of the execution on sets of computational nodes.
Failover Management	Automatically transfers control to a duplicate computational node when faults or failures are detected.
State-of-the-art data mining algorithms	General purpose algorithms (e.g. Clustering, Principal Component Analysis, Artificial Neural Networks, Maximum Entropy, etc.) supplied as-a-service.
Data trends generation and analysis	Identification of trends; inspection of time series to automatically identify anomalies; basic signal processing techniques to explore periodicities in trends.

2.4.3. Subsystems

2.4.3.1. Data Miner System

The Data Miner System's goal is to offer a unique access for performing data mining or statistical operations on heterogeneous data. These data can reside on the client side in the form of CSV files or they can be remotely hosted, as SDMX documents or, furthermore, they can be stored in a database.



The Data Miner System is composed by a service, namely Data Miner service, able to take such inputs and execute the operation requested by a client interface, by invoking the most suited computational infrastructure, choosing among a set of available possibilities: executions can run on multi-core machines, or on different computational infrastructures, like the PARTHENOS, Windows Azure, CompSs and other options. Algorithms are implemented as plug-ins which makes the injection mechanism of new functionalities easy to deploy.

Key Features

Openness	Interaction with external software supporting Standard protocols.
Parallel Processing	Support for the execution of algorithms on multi-core computational nodes.
Distributed Processing	Transparent distribution of the execution on sets of computational nodes .
State-of-the-art data mining algorithms	General purpose algorithms (e.g. Clustering, Principal Component Analysis, Artificial Neural Networks, Maximum Entropy, etc.) supplied as-a-service.

Architecture

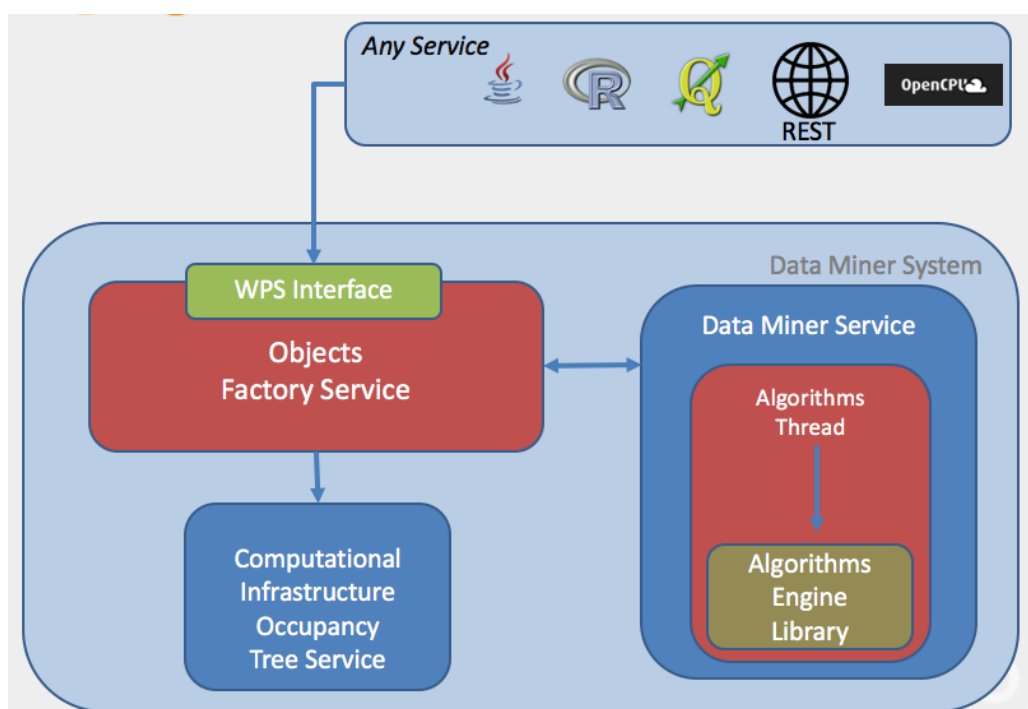


Figure 12. Data Miner System Architecture



According to Figure 12, the Data Miner System comprises the following components:

- **Computational Infrastructure Occupancy Tree service:** a service which monitors the occupancy of the resources to choose among when launching an algorithm;
- **Data Miner Service:** a service executing all the computations asked by a single user/service. It is composed of two components:
 1. **Algorithms Thread:** an internal process which puts in connection the algorithm to execute with the most unloaded infrastructure resource which is able to execute it. Infrastructures are weighted even according to the computational speed; the internal logic will choose the fastest available;
 2. **Algorithms Engine Library:** a container for several data mining algorithms as well as evaluation procedures for the quality assessment of the modelling procedures. Algorithms follow a plug-in implementation and deployment;
- **Object Factory Service:** a service acting as a broker for Data Miner Service and a link between the users' computations and the Occupancy Tree service;

It is worth noticing that, thanks to the support of HTTP-REST and WPS protocols, the Data Miner System is capable of interacting with different external software supporting such standards (e.g. QGIS, OpenCPU) and different programming languages, in particular Javascript, R, and Java.

2.4.3.2. Smart Executor System

The SmartExecutor service allows execution of "gCube Tasks" and monitoring of their execution status. Each instance of the SmartExecutor service can run the "gCube Tasks" related to the plugins available on such an instance. Each instance of the SmartExecutor service publishes descriptive information about the co-deployed plugins.

Key Features

Repetitive Tasks	Task can be scheduled to be periodically repeated.
Tasks take over	Task can be taken in charge from new instances in case of instance failure or instance overload.



Architecture

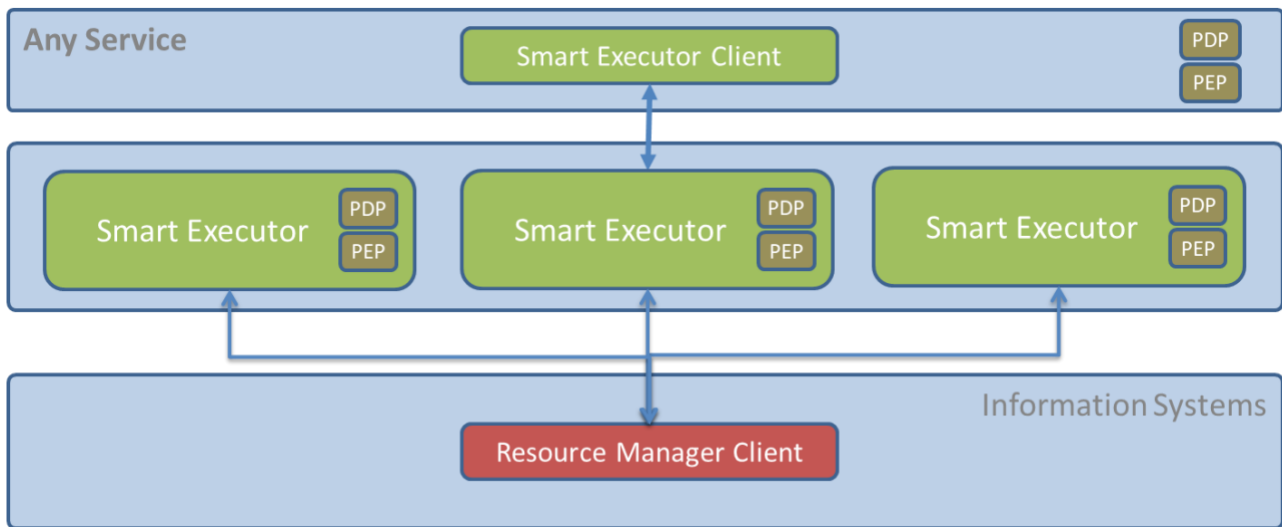


Figure 13. Smart Executor System Architecture

Clients may interact with the SmartExecutor service through a library (SmartExecutor Client) of high-level facilities to simplify the discovery of available plugins in those instances. Each client can request to execute a "gCube Tasks" or getting information about the state of their execution.

The SmartExecutor service allows tasks execution through the use of co-deployed plugins. The service allows inputs parameter to be passed to the plugin requested to run. The execution is invoked every time it matches the scheduling parameters. The way to schedule the plugin execution is indicated by the scheduling parameter. There are two different ways to schedule an execution:

- **run and die:** the plugin is launched just for one time and after this execution it won't be repeated;
- **scheduled:** the plugin repeats its execution over time according to a delay interval or to a "cron" expression.

SmartExecutor instances can take care of a scheduled run when the node where it was previously allocated crashes or is overloaded. To achieve this goal, a scheduled task description is registered in the Information System through the Resource Manager.



2.5. Content Cloud Framework

2.5.1. Overview

The PARTHENOS Content Cloud is a digital space that acts as container of resources and metadata of resources aggregated from RI registries registered in the PARTHENOS registry.

The Content Cloud Framework (CCF) groups the services needed to (i) populate the Content Cloud, and (ii) make the Content Cloud accessible to human and machines according to different standard protocols. The CCF is based on the *D-Net Software toolkit*, a service-oriented framework specifically designed to support developers at constructing custom aggregative infrastructures in a cost-effective way.

The D-Net Software Toolkit⁵ is an open source (Apache licence) service-oriented framework, fully developed in Java. Its first software release was designed and developed within the DRIVER and DRIVER-II EC projects (2006-2008). The scenario motivating its realization was that for constructing the European repository infrastructure for Open Access repositories. The infrastructure had to harvest (tens of) millions of Dublin Core metadata records from hundreds of OAI-PMH repository data sources, harmonize the structure and values of such records to form a uniform information space.

A D-Net data infrastructure is a run-time distributed environment, inspired by Service-Oriented Architecture paradigms, where administrators can dynamically construct, refine, and monitor aggregation and data management workflows for information space construction.

⁵ D-Net Software Toolkit: <http://www.d-net.research-infrastructures.eu>



Key Features

Economy of scale	Services constituting one aggregative infrastructure may be hosted over servers maintained at different sites.
Robustness	Service replicas, i.e. clones of functionality and content, can be kept at different sites. This strategy, in combination with dynamic discovery of resources, makes the system more robust to network failures and system crashes (availability of service) as well as to concurrent accesses (scalability by workload distribution).
Autonomy	Manager Services can autonomously orchestrate and monitor the status of services in the aggregative infrastructure.
Elasticity	Thanks to dynamic discovery, services can join or leave the infrastructure anytime without administrators having to re-configure application workflows.
Modularity	Services provide “functionality in isolation”, that is functionality “factored out as much as possible”, in order to maximize re-use in different workflows.
Customizability	Services managing metadata objects should be customizable at run-time to operate according to a given data model. This feature makes service instances dynamically programmable and promotes their re-use to serve different goals.
Metadata Interoperability	Services are able to manage metadata records in different formats. Different standard exchange protocols are supported both in import and export phases. In the import phase (data collection) idiosyncratic protocols can also be supported by integrating dedicated plugins.

2.5.2. Subsystems

The D-Net framework is composed of services that can be grouped in layers and categories based on the functionality they provide. The enabling layer includes component for the correct operation of the aggregative infrastructure and are available in every instance of D-Net. Components and services in the data management layer are, instead, selected and

configured to address specific requirements. It is possible to group services and components in four main categories, as depicted in Figure 14:

- Mediation area: services and components that support and implement data collection processes.
- Storage area: services and components for storing data and metadata. Different storage options are available out-of-the-box. For PARTHENOS, the selected services are the MDStore, a storage service based on the MongoDB technology, for the storage of metadata records, and the Index, used by the D-Net curation tool called “Metadata Inspector” (see Figure 22 and Deliverable 6.4 “Report on Services and Tools” for details on the curation tool).
- Manipulation area: services and components that implement data manipulation such as transformation, harmonisation, de-duplication, and validation. For PARTHENOS the Transformator service was selected and enhanced to support the execution of mappings in X3ML format.
- Provision area: services and components that interface external applications, e.g. end-user portals, third-party services to the aggregated content. For PARTHENOS, the SPARQL endpoint offered by a Virtuoso server, and the OAI-PMH Publisher, based on the MongoDB technology, have been selected and configured.

In the next subsections, some of the most D-Net relevant services and components are described, highlighting the configuration and extensions that have been applied during the PARTHENOS project.

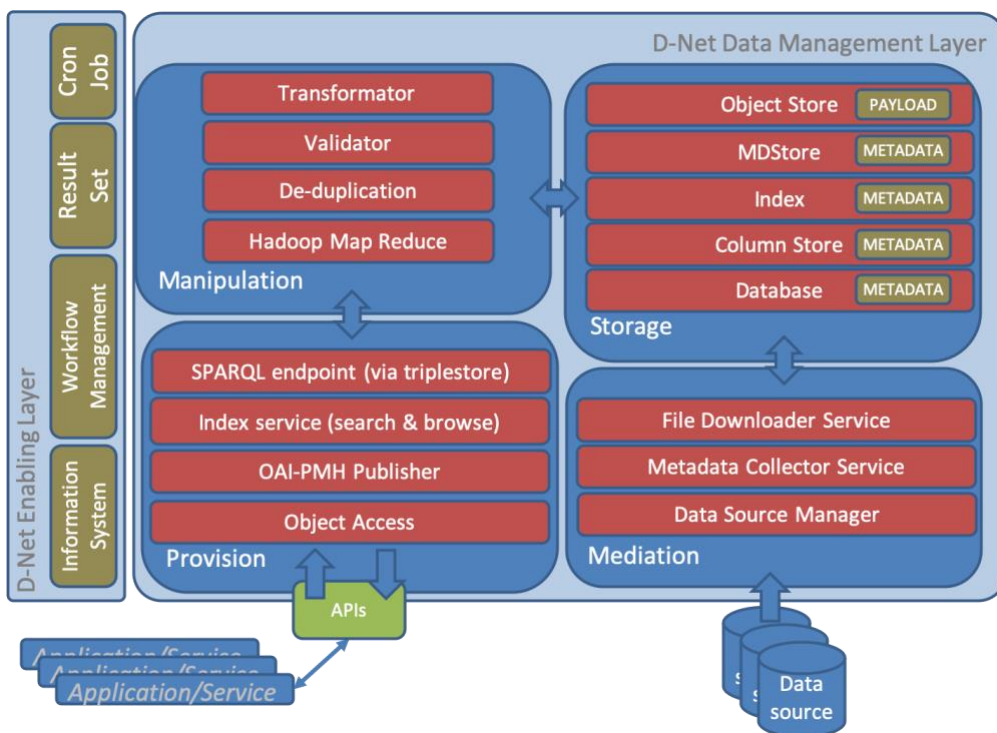


Figure 14. D-NET Aggregative Infrastructure Architecture



2.5.2.1. Workflow Management

The Workflow Management System (WMS) addresses service orchestration and monitoring, hence “autonomic behaviour”. One or more WMSs can be configured by developers to autonomously execute *workflows*. D-Net workflows are resources describing sequences of steps, where each step may consist of business logic (i.e. Java code), remote service invocations, workflow forks (i.e. parallel sub-workflows), and workflow conjunctions (confluence of parallel workflows). Typically, service invocations are preceded by a look-up into the Information System (IS) to discover the “best” service of the needed kind and available to execute the call. Workflows can be fired manually or as a consequence of the notification of a resource-related event from the IS or because of time-events, i.e. cron jobs. Workflows are commonly used to automatically schedule data aggregation (i.e. collection and transformation of metadata records into a common format) from data sources.

Workflows can implement long-term transactions by exploiting subscription and notification of events in the IS. When a time-consuming step is to be fired (e.g. indexing a large set of metadata objects), the invocation is accompanied by a subscription request to the event “conclusion of the step”. The WMS waits for the relative notification before moving on to the next step. Workflows can also be used as monitoring threads, checking for consistency and availability of resources or consistency and Quality of Service of the aggregative infrastructure. For example, aggregative infrastructure policies may require that a given collection of information objects be replicated K times; monitoring workflows may, at given time intervals, check that this is really the case and possibly take corrective actions, e.g. creating a missing replica. When corrective actions are not possible, warning emails can be sent to administrators.

The WMS user interface offers a graphical overview of the ongoing workflows and allows administrators to interact with such workflows, for example to manually re-execute them or to redefine their configuration parameters. In the current D-Net implementation, workflows are not treated as infrastructure resources, i.e. cannot be shared by different instances of the Manager Service, and are preserved in the local status of the service.

The data flow devised for PARTHENOS is depicted in Figure 15. For each data source, the aggregation workflow automatically collects input metadata records, transforms them according to a defined X3ML mapping, and makes the resulting RDF records available to

metadata experts for inspection. The publishing workflow is executed in order to officially publish the records, making them available via an OAI-PMH Publisher (in RDF/XML according to the PARTHENOS Entities model and Dublin Core) a SPARQL endpoint and in the PARTHENOS Joint Resource Registry.

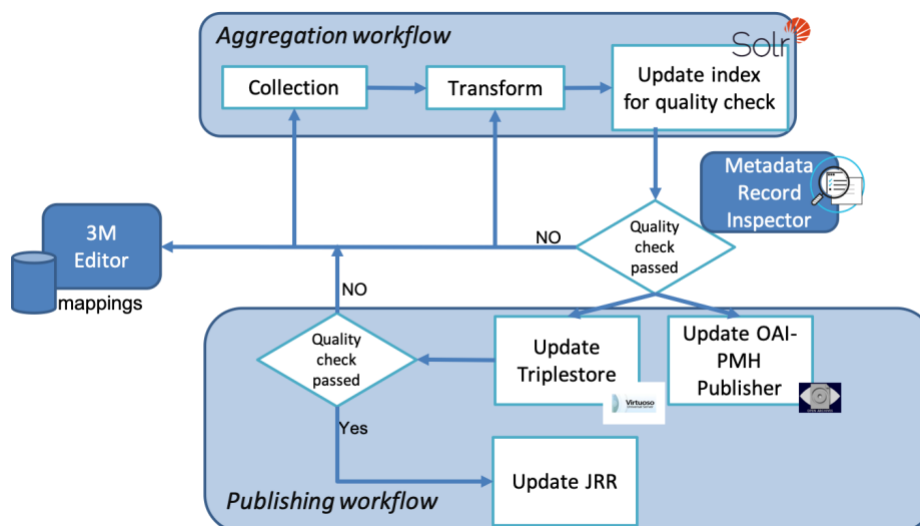


Figure 15. The data flow devised for the PARTHENOS aggregator

The data flow is implemented by a set of D-Net workflows:

- One aggregation workflow per data source endpoint, composed of three steps as depicted in Figure 16: (a) collection, transform and index. The last step makes the transformed records available for inspection in the Metadata Record Inspector, one of the tools used by aggregator administrators and data experts to check the quality of the aggregated records (more details available in deliverable D6.4 “Report on services and tools”).
- One workflow per data source endpoint that makes the aggregated records available via OAI-PMH (Figure 16, item (b)).
- One workflow per data source endpoint that pushes the aggregated records into the triple store (Virtuoso), making them available via a SPARQL endpoint (Figure 16, item (c))
- One workflow per data source endpoint for publishing resources into the PARTHENOS JRR (Figure 16, item (d)). RDF resources are read from Virtuoso and mapped according to the JRR model. They are then pushed to the JRR and therefore available via the JRR GUI. Thanks to this workflow, the PARTHENOS URI that were generated in the transformation step of the aggregation workflow can be resolved to the relative entry of the JRR.

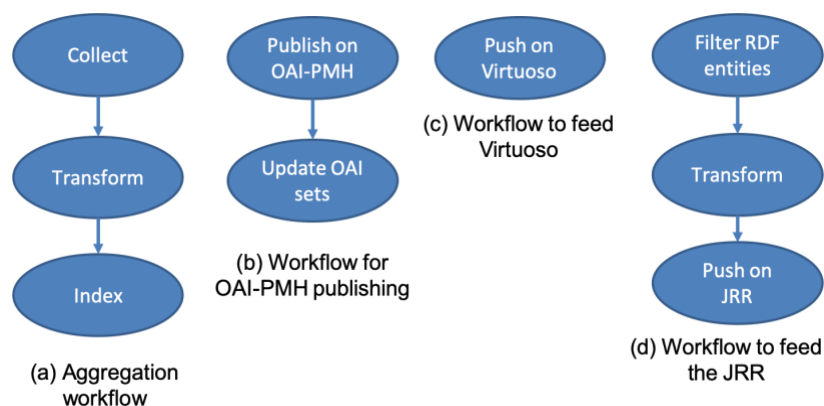


Figure 16. D-Net workflows for PARTHENOS

2.5.2.2. Data Source Manager

The Data Source Manager provides services and graphical user interfaces (GUIs) for the registration and administration of data sources to the aggregative infrastructure, meaning the organization and scheduling of the respective data collection and processing workflows. Figure 17 illustrates the administrative user interface used to start and monitor the execution of an aggregation workflow for the metadata records available from the OAI-PMH endpoint of CulturalItalia. The given workflow collects the metadata records from the remote OAI-PMH endpoint (sub workflow “collection”), transforms them according to the PARTHENOS Entities model by applying a dedicated transformation rule (sub workflow “transform”) and then pushes the transformed records into a Solr index (sub workflow “index”). From the same interface, the aggregation manager can modify the parameters to use when connecting to the data source (Figure 18), the mappings to be applied (Figure 19), check the history of past executions of the workflows (Figure 20), set an automated scheduling of the workflows and customize notification settings (Figure 21).

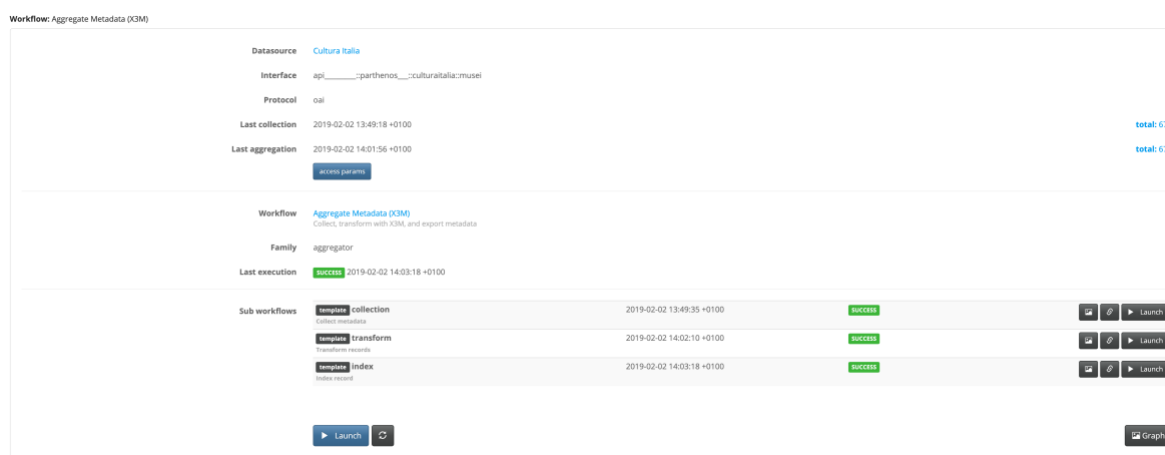


Figure 17. GUI for Data Source Management

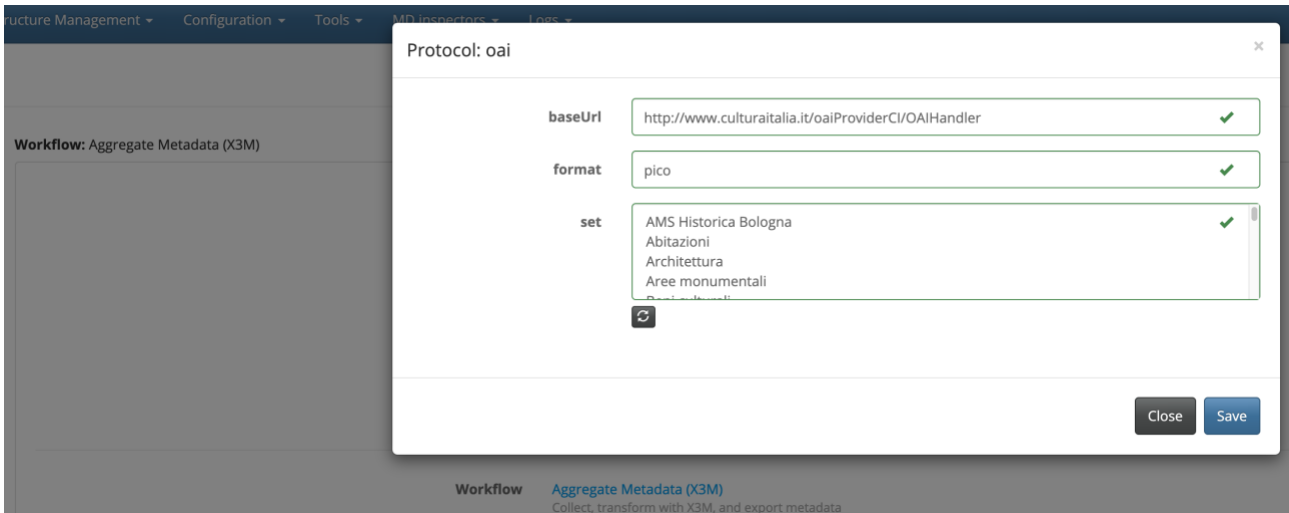


Figure 18. Modify connection parameter to a remote data source

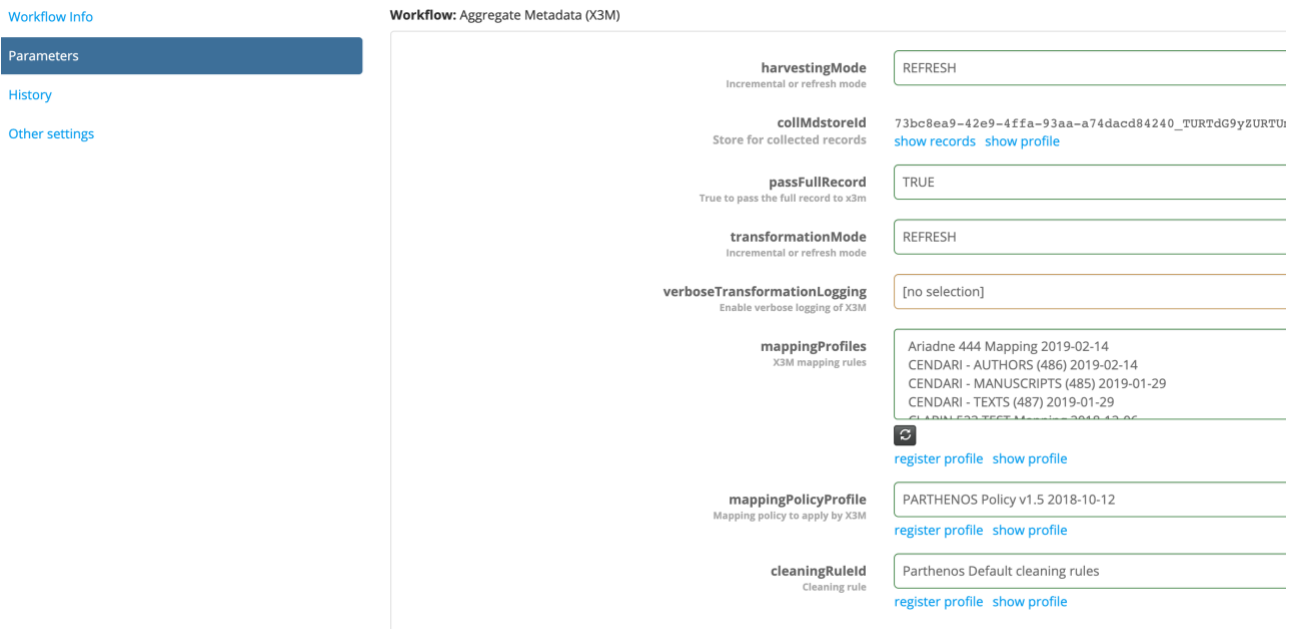


Figure 19. Parameters section: configuration of the aggregation workflow

Workflow: Aggregate Metadata (X3M)

Process Id	Workflow name	Workflow family	Status	Date
wf_20190202_140210_587	index	aggregator	Success	2019-02-02 14:03:20
wf_20190202_134559_343	Aggregate Metadata (X3M)	aggregator	Success	2019-02-02 14:03:19
wf_20190202_134935_979	transform	aggregator	Success	2019-02-02 14:02:10
wf_20190202_134603_677	collection	aggregator	Success	2019-02-02 13:49:35
wf_20180802_171943_391	collection	aggregator	Success	2018-08-02 17:19:47
wf_20180716_161859_153	index	aggregator	Success	2018-07-16 16:19:13
wf_20180412_180210_594	index	aggregator	Success	2018-04-12 18:02:41
wf_20180412_175830_333	Aggregate Metadata (X3M)	aggregator	Success	2018-04-12 18:02:41
wf_20180412_180040_106	transform	aggregator	Success	2018-04-12 18:02:10
wf_20180412_180019_309	collection	aggregator	Success	2018-04-12 18:00:40
wf_20180412_172648_721	index	aggregator	Success	2018-04-12 17:26:59
wf_20180412_172356_378	index	aggregator	Failure	2018-04-12 17:24:03
wf_20180409_135645_931	index	aggregator	Success	2018-04-09 13:56:53
wf_20180409_135555_678	Aggregate Metadata (X3M)	aggregator	Success	2018-04-09 13:56:53
wf_20180409_135608_449	transform	aggregator	Success	2018-04-09 13:56:45

Figure 20. History section: view status of past executions of the same workflow



Figure 21. Other settings section: configure scheduling and email notifications

2.5.2.3. Metadata Collector Service

The Metadata Collector Service is capable of fetching data from external data sources and imports them into the aggregative infrastructure as information objects conforming to a given data model. In order to be discovered and accessed for collection, data sources must be registered with a profile in the registry. The profile can specify one or more access point interfaces (APIs), that is different ways to access the content of the data source. For example, a publication repository may provide an OAI-PMH interface as well as an FTP interface to provide bulk-access to metadata and files of publications, respectively. D-Net provides so-called “collector plug-ins” that are able to collect files from data sources implementing the most common (de-facto) standard exchange protocols:

- *OAICollectorPlugin*: harvests metadata records in XML from an OAI-PMH Publisher. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
 - *format*: OAI metadataPrefix (mandatory);
 - *set*: list of sets to be harvested (optional). When not provided all records are harvested.
- *HttpCollectorPlugin*: collects metadata records from one XML file available at a remote location. The plugin will split the input file into several XML records based on the *splitOnElement* parameter. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
 - *splitOnElement*: name of the XML field that identifies the root of each records in the input file (mandatory).
- *HttpListCollectorPlugin*: Collects metadata records from a remote endpoint when the path of each record is provided via a text file (one by line). In other words, the data source has an "index file" where the paths to the metadata records are listed one per line. The plugin reads each line of the index file and downloads from the



URL provided in each line. The index line can also contain a partial URL as the baseURL is used as prefix to each line in the index file. Parameters:

- *baseUrl*: base URL to construct the final URLs to the metadata records to download (mandatory);
- *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
- *listUrl*: URL to the the index file listing the locations (mandatory).
- *FileCollectorPlugin*: Collects metadata records from one XML file available at a local location on the file system. The plugin will split the input file into several XML records based on the *splitOnElement* parameter. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory)
 - *splitOnElement*: name of the XML field that identifies the root of each records in the input file (mandatory).
- *FilesystemCollectorPlugin*: Collects metadata records from one folder the file system. Each file must be a single metadata record. In case of json files, they are collected and transformed in XML. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory). In case of json format, refer to the xpath that is valid after the conversion from json to xml via the org.json library;
 - *extensions*: comma separated list of extensions. (optional);
 - *fileFormat*: xml or json. Default to xml. (optional).
 - *setObjIdentifierFromFileName*: true if you want the D-NET identifier to be forged based on the file name instead of the ID XPath.
- *ClasspathCollectorPlugin*: Collects metadata records from one XML file available in the classpath of the web application. The plugin will split the input file into several XML records based on the *splitOnElement* parameter. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
 - *splitOnElement*: name of the XML field that identifies the root of each records in the input file (mandatory).
- *HttpCSVCollectorPlugin*: Reads one CSV file with header and generate one XML metadata record per line. Lines with invalid quotes are skipped. Parameters:
 - *baseUrl*: base URL of the endpoint (mandatory);
 - *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
 - *separator*: string used to separate columns (optional). When empty `t` is assumed.
 - *identifier*: name of the column where the value to be used to forge D-NET identifier for each record can be found (mandatory);



- *quote*: char used in the CSV for quoting (optional). When empty, no quotation char is considered.
- *ReadExcelPlugin*: Collects metadata records from one sheet of an excel file with headers. The data is converted from excel to csv;
- *FtpCollectorPlugin*: Collects XML metadata records with a given extension from an FTP site with login authentication. The plugin can be configured to: (i) go recursively into subfolders; (ii) collect files with multiple extensions. Parameters:
- *baseUrl*: base URL of the endpoint, must include the path on the SFTP site where to start the collection (mandatory);
- *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
- *username* and *password*: for SFTP authentication (mandatory);
- *recursive*: set to true to enable recursion. False to only visit the folder identified by *baseUrl*. (mandatory);
- *extensions*: comma separated list of extensions. (mandatory);
- *SftpCollectorPlugin*: Collects XML metadata records with a given extension from an SFTP site with login authentication. The plugin can be configured to: (i) go recursively into subfolders; (ii) collect files with multiple extensions. Parameters:
- *baseUrl*: base URL of the endpoint, must include the path on the FTP site where to start the collection (mandatory);
- *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);
- *username* and *password*: for FTP authentication (mandatory);
- *recursive*: set to true to enable recursion. False to only visit the folder identified by *baseUrl*. (mandatory);
- *extensions*: comma separated list of extensions. (mandatory).
- *RestCollectorPlugin*: Collects metadata records from a REST endpoint. It features a number of parameters to support its adoption for the collection for almost any REST API, regardless their peculiarities for handling pagination and returning json or XML.
- *TarGzCollectorPlugin* and *ZipCollectorPlugin*: Collect metadata records from a targz or zip archive located in the local file system. Each file in the archive is assumed to be one XML metadata record. Parameters:
- *baseUrl*: path to the archive in the local file system (mandatory);
- *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory).
- *FileGZipCollectorPlugin*: Collects metadata records from a gzipped file located in the local file system. The plugin will split the input file into several XML records based on the *splitOnElement* parameter. Parameters:
- *baseUrl*: path to the file in the local file system (mandatory);
- *ID XPath*: xpath where the value to be used to forge the D-NET identifier for each record can be found (mandatory);



- *splitOnElement*: name of the XML field that identifies the root of each records in the input file (mandatory).
- *SchemaOrgPlugin*: Collects schema.org/Dataset records published in accessible html pages. Supported microformats are JSON-LD. Supported Repository protocols:
 - sitemapindex - Given a sitemapindex.xml file, the plugin will access sequentially all sitemap files and extract the endpoints to later be processed by the transformation pipeline;
 - HTTP API listing - Given a GET URL, the plugin will sequentially retrieve page by page all listed entries, extract endpoints from the result and provide them for processing to the transformation pipeline.

In cases when a data source cannot be aggregated by one of the existing plug-ins, new plug-ins can be easily created to implement the proprietary/idiosyncratic protocol of the data source. In the context of PARTHENOS, an analysis of the APIs of the research infrastructures has been carried out to ensure that D-Net could collect metadata records from all the available sources. A summary of the analysis is available in Table 1. The analysis revealed that D-Net natively covers the majority of API protocols used by research infrastructures. The SFTP plugin has been extended to support public key authentication to support the collection of ARIADNE records, while new dedicated plugins have been developed for DARIAH-DE, EHRI and Huma-Num.

Plug-in for DARIAH-DE

Metadata records are available in XML from an HTTP endpoint. In order to collect one record, the base URL must be completed with the identifier of the record. The identifiers of the records can be found in the collection overview endpoint available at <https://colreg.de.dariah.eu/colreg-ui/api/collections/>.

Plug-in for EHRI

Similar to DARIAH-DE, the list of identifiers of EHRI XML records must be fetched from a specific endpoint. With the identifier it is possible to construct the HTTPS URL from which the record can be collected. The endpoint to discover the identifiers is, in this case a GraphQL API (<https://portal.ehri-project.eu/api/graphql>).

Plug-in for Huma-Num Isidore

A dedicated plug-in has been created as a simplification of the REST plugin, with the added-value of handling XML processing with a more updated Saxon HE library.



Table 2. Analysis of the APIs of PARTHENOS research infrastructure with respect to the collection plugins natively available in D-NET.

PARTHENOS data source	D-Net collector plug-in	Note
ARIADNE	SFTP plugin	Extended with public key authentication
CLARIN	HTTP plugin	
CENDARI	Filesystem plugin	
CulturalItalia	OAI-PMH plugin	
DARIAH-DE	DARIAH-DE plugin	Dedicated plug-in implemented
DARIAH GR/ΔΥΑΣ	OAI-PMH plugin	
EHRI		Dedicated plug-in implemented
Huma-Num Isidore collection level	HTTP plugin	
Huma-Num Isidore item level	Isidore-REST plugin	Dedicated plug-in implemented
Huma-Num Nakala collection level	HTTP plugin	
Huma-Num Nakala item level	OAI-PMH plugin	
LRE Map	Gzip plugin	
METASHARE	Targz plugin	

2.5.2.4. Transformator

The Transformator leverages services for transforming metadata objects of one metadata data model into objects of one output metadata data model. User interfaces allow data managers to specify the logic of the transformation, i.e. the mapping, which can be an XSLT, a D-Net script or an X3ML mappings generated with the X3M Mapping tool implemented by the project partner FORTH and available in the PARTHENOS infrastructure. The support of



X3ML mappings has been realized during the project via the integration into the Data Transformation System of the X3ML engine, the transformation engine developed by FORTH capable of processing X3ML mappings. In the specific case of PARTHENOS, mappings have been prepared with the X3M Mapping tool and registered in the registry in order to be discoverable by the D-Net Data Transformation System.

2.5.2.5. Provision services

Services in the provision area interface external applications, e.g. end-user portals, third-party services, with resources and metadata in the Content Cloud in its different manifestations. Specifically, metadata are made available to third-parties via the following services:

OAI-PMH Publisher Service An OAI-PMH Publisher Service offers OAI-PMH interfaces to third-party applications (i.e. harvesters) willing to access metadata objects. The service can be dynamically configured to expose sets grouping records that satisfy given criteria (e.g. original data source, value of the subject field). The service is implemented on MongoDB.

Index (search and browse) Service An Index (factory) Service manages a set of Index units capable of indexing metadata objects of a given data model. Consumers can feed units with metadata objects, remove objects or query the records. As indexing back-end, the service supports Apache Solr, a de-facto standard for full-text indexing and content retrieval. The schema of the Solr index is configured by the Index Service according to a configuration, that can be changed dynamically at run-time, in terms of indexable and browsable fields. In PARTHENOS, Solr is configured to index metadata records transformed into the PARTHENOS Entities model to serve the D-Net Metadata Inspector, a tool that support aggregation managers and data curators to perform analysis of the quality of the aggregated metadata records. Figure 22 shows a screenshot of the Metadata Inspector. More information about the tool are available in Deliverable 6.4 “Report on Services and Tools”.



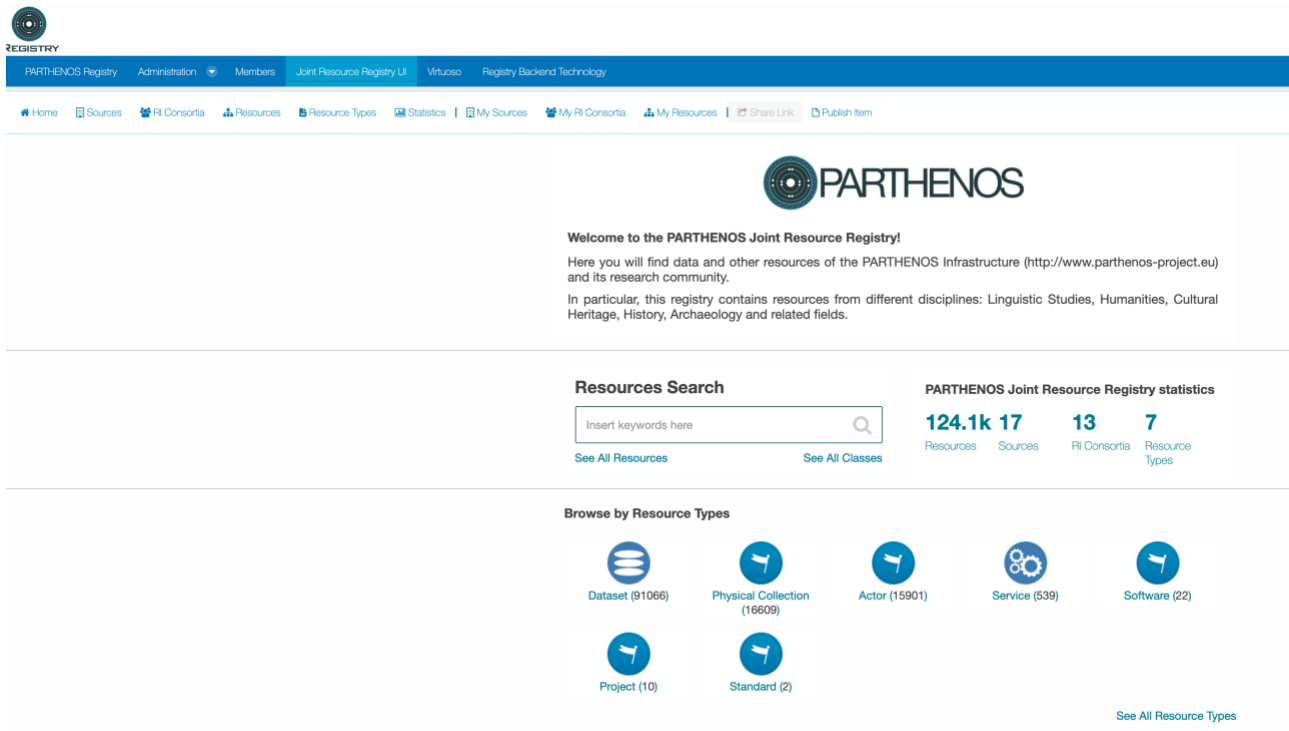
Metadata Record Inspector

XPPath	Value	Vocabulary
///rdf:Description[./rdf:type@iri="http://www.idoc-crm.org/idoc-crm/ES3_Place"]/rdfs:label	Eathing Copse, Lower Eathing, E of Peper Narow	PARTHENOS.Places

Figure 22. Screenshot of the D-Net Metadata Inspector

SPARQL Service: The SPARQL endpoint is offered by an OpenLink Virtuoso server deployed in the PARTHENOS infrastructure and fed by the PARTHENOS Aggregator. The service is freely available at <https://virtuoso.parthenos.d4science.org/sparql> and accessible via the PARTHENOS Registry VRE (https://parthenos.d4science.org/group/parthenos_registry/virtuoso).

In addition to the aforementioned endpoint, the PARTHENOS Aggregator also feeds the PARTHENOS Joint Resource Registry available via the PARTHENOS Registry VRE (https://parthenos.d4science.org/group/parthenos_registry/catalogue). Figure 23 shows a screenshot of the home page of the Joint Resource Registry, described in more detail in deliverable 6.5 “Report on the implementation of the Joint Resource Registry”.



The screenshot shows the PARthenos Joint Resource Registry website. The header includes the logo and navigation links: PARthenos Registry, Administration, Members, Joint Resource Registry UI, Virtuoso, and Registry Backend Technology. A secondary navigation bar contains links for Home, Sources, RI Consortia, Resources, Resource Types, Statistics, My Sources, My RI Consortia, My Resources, Share Link, and Publish Item. The main content area features a large PARthenos logo and a welcome message: "Welcome to the PARthenos Joint Resource Registry! Here you will find data and other resources of the PARthenos Infrastructure (http://www.parthenos-project.eu) and its research community. In particular, this registry contains resources from different disciplines: Linguistic Studies, Humanities, Cultural Heritage, History, Archaeology and related fields." Below this is a "Resources Search" section with a search input field and a search button. To the right, "PARthenos Joint Resource Registry statistics" are displayed: 124.1k Resources, 17 Sources, 13 RI Consortia, and 7 Resource Types. A "Browse by Resource Types" section follows, showing icons and counts for Dataset (91066), Physical Collection (16609), Actor (15901), Service (539), Software (22), Project (10), and Standard (2). A "See All Resource Types" link is at the bottom right.

Figure 23. Screenshot of the PARthenos Joint Resource Registry

2.6. Collaborative Framework

2.6.1. Overview

The Collaborative Framework is realized through a combination of software components (services and libraries) powered by the gCube System. Three main subsystems characterise the Collaborative Framework:

- Social Networking System;
- Shared workspace System;
- User Management System.

These systems provide consumers with a homogenous abstraction layer over different external technologies enabling to operation of the framework. The external technologies involved comprise Apache Cassandra, Apache Jackrabbit, Elastic Search, MongoDB, and Liferay Portal. In particular, the Social Networking System exploits an Apache Cassandra cluster and an Elastic Search cluster, the Shared Workspace System exploits an Apache Jackrabbit repository (metadata) and a MongoDB cluster (payload) for its backend, the User Management System exploits Liferay Portal for its backend and to allow users to login for personalized services or views.



2.6.2. Key Features

Collaboration	Users can share posts and have multiple discussions on the VRE homepage, adding comments and files in line with the discussion.
Folder Sharing	Folder sharing enables the reuse of content and the ease of creating multiple VREs for different audiences with shared content.
Custom Notifications	Important and personalized alerts appear in each user's notification area, and custom applications can add their own notifications.
Responsive Design support	Web Applications are based on Twitter Bootstrap, making it possible to create responsive pages that look great regardless of device.
Economy of scale	Services constituting one aggregative infrastructure may be hosted over servers maintained at different sites

2.6.3. Subsystems

2.6.3.1. Social Networking System

The Social Networking System comprises services conceptually close to the common ones promoted by social networks – e.g., posting news, commenting on posted news, likes, private messages and notifications; It is composed of two main services, the Social Networking Service and the Social Indexer Service. The Social Networking Service logic relies on the Social Networking Model, this Model is used also for the efficient storage of the Social Networking Data (Posts, Comments, Notifications etc.) in the underlying Apache Cassandra Cluster. This Cluster is queried by means of a Java client.

Architecture

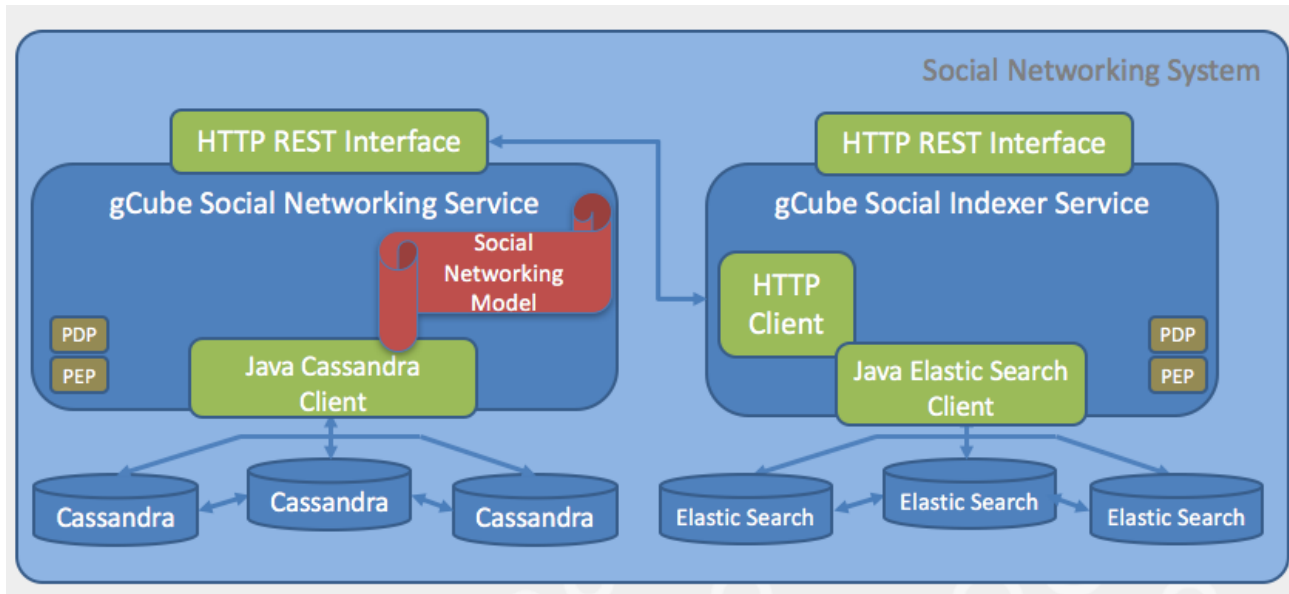


Figure 24. Social Networking System Architecture

The Social Networking Service exposes an HTTP REST Interface for the internal and external services of the infrastructure. The Social Indexer Service uses such interfaces for the retrieval of the Social Networking Data to index by means of an Elastic Search Cluster. The Social Indexer Service exposes an HTTP REST Interface for the internal and external services of the infrastructure needing to perform search operations over the Social Networking Data.

Both Services rely on the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP) to intercepts user's access request and evaluate these requests against authorization policies of the Authorization System of the Infrastructure.

2.6.3.2. Shared Workspace System

The Shared Workspace System provides a remote (Cloud) folder-based file system, supporting sharing of folders and different item types (ranging from binary files to information objects representing, for instance, tabular data, workflows, distribution maps, statistical algorithms).

Architecture

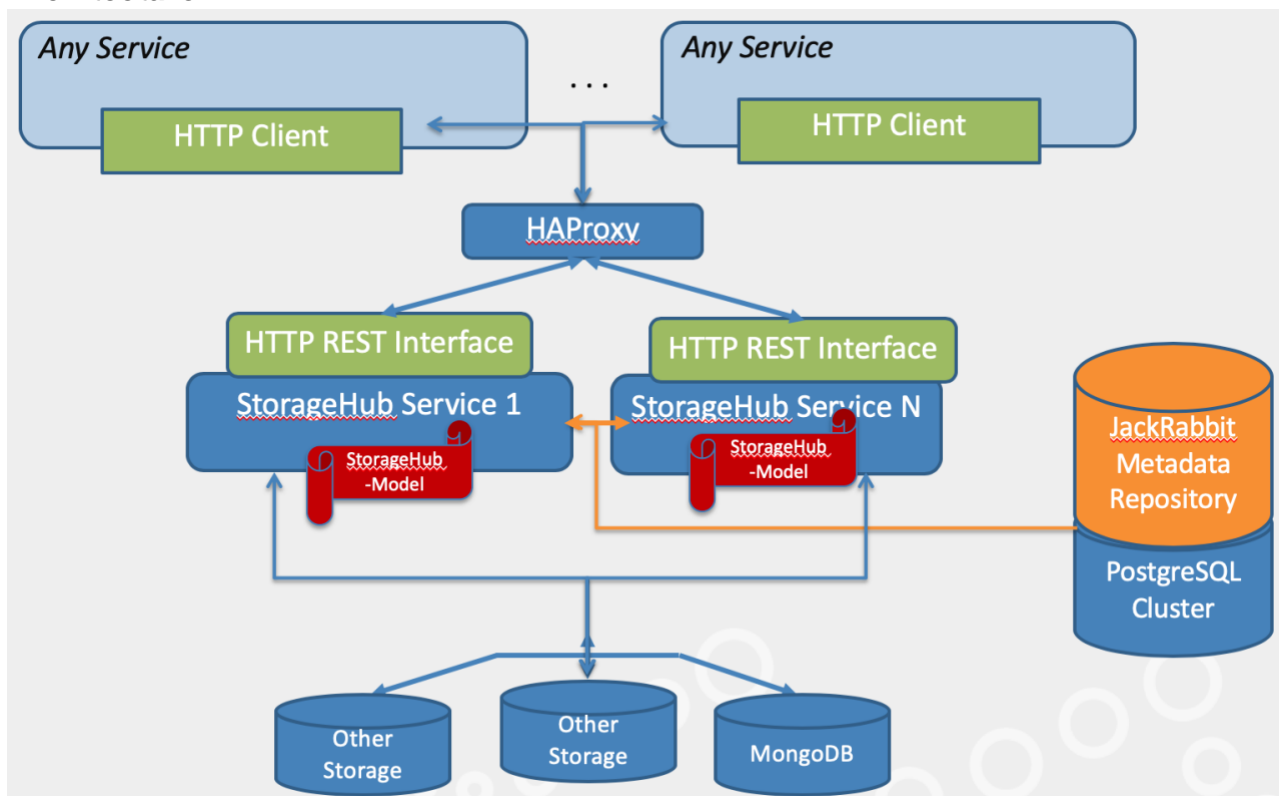


Figure 25. Shared Workspace System Architecture

With respect to the last reporting period (M36), the Shared Workspace System was heavily redesigned with the aim of increasing its scalability and overall performance. It consists of one gCube service, named StorageHub Service, relying on two different storage technologies for the metadata of the items being stored, namely Apache Jackrabbit as a metadata repository and PostgreSQL as the Apache Jackrabbit Back-end Database. The StorageHub Service is replicable and a HAProxy on top is used for proxying requests to the deployed instances of it. One other distinguished feature of the StorageHub Service is that the actual payload of the items can be stored on a number of in-house and commercial storage technologies, for instance in a MongoDB Cluster, but also on other types, including Cloud Storages solutions (e.g. Amazon S3).

The StorageHub Service identifies a core set of capabilities to work on JackRabbit content. Together with its model, named StorageHub model, it exposes content in the content repository as HTTP resources, fostering a RESTful style of application architecture. The Home RESTFUL interface processes HTTP requests coming from clients. The following operations are supported:



- retrieve content;
- create content;
- modify existing content;
- remove existing content;
- move existing content to a new location;
- copy existing content to a new location;

2.6.3.3. User Management System

Users are the fundamental entity managed by this System. As a matter of fact, the User is an entity that can sign into the PARTHENOS Portal and do something. Users are assigned a Role which defines the user's privileges. The User Management System provides functionality to manage personal profiles and users in the PARTHENOS infrastructure, supporting user groups (for the purpose of group specific privileges) and roles for application specific needs related to the user's role in PARTHENOS.

The following roles are supported:

- VRE Manager: this role is envisaged to manage the VREs Management. Users with this role can
 - create, edit, a VRE;
 - manage VRE users (e.g. approve / reject membership requests, assign roles, create and manage groups);
- VRE Designer: this role is envisaged to manage the VREs Management. Users with this role can
 - Define/request a VRE, also by asking for the list of required applications;
- VRE Member: this role grants basic privileges within a VRE, such as the ability to visit the VRE's private pages and use the VRE offering.