PARTHENOS

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

D6.5 Report on the implementation of the Joint Resource Registry (Final)

AUTHORS: Luca Frosini
Massimiliano Assante
Leonardo Candela
Lucio Lelii
Francesco Mangiacrapa
Pasquale Pagano

DATE 15 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

Report on the implementation of the Joint Resource Registry (Final)

Deliverable Number	D6.5
Dissemination Level	Public
Delivery date	15 April 2019
Status	Final
Author(s)	Luca Frosini, CNR Massimiliano Assante, CNR Leonardo Candela, CNR Lucio Lelii, CNR Francesco Mangiacrapa, CNR Pasquale Pagano, CNR

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	D6.5
Document title	Report on the implementation of the Joint Resource Registry (final)
Author(s)	Luca Frosini, CNR Massimiliano Assante, CNR Leonardo Candela, CNR Lucio Lelii, CNR Francesco Mangiacrapa, CNR Pasquale Pagano, CNR
Dissemination level/distribution	Public

Document History

Version/date	Changes/approval	Author/Approved by
V 0.1 12.04.19	Version ready for approval	Luca Frosini
V 1.0	Reviewed	Sheena Bassett, PIN

Table of Contents

List of Tables

List of Figures

1 Executive Summary

Deliverable D6.5 “Report on the implementation of the Joint Resource Registry (final)” is the revised version of the deliverable D6.3 “Report on the implementation of the Joint Resource Registry (interim)”.

The D6.5 Report on the implementation of the Joint Resource Registry documents the final implementation of the Joint Resource Registry (JRR). It complements the D5.2 Report on the design of the Joint Resource Registry deliverable by providing details on how to interact with and exploit the functionalities it provides.

The JRR hosts the PARTHENOS entities represented according to the PARTHENOS Entities Model defined in WP5. As such, it represents an information system for the PARTHENOS community and the PARTHENOS universe of tools and services designed for and released in the PARTHENOS infrastructure.

The feedback obtained during the first reporting period has been used to improve the:

- quality of the design,
- the provided APIs of both services and clients,
- and the design of the Graphical User Interfaces (GUIs) provided to the PARTHENOS community.

The functional and non-functional requirements are now more detailed and the description have been improved.

The Resource Registry service REST APIs have been improved to strictly adhere to REST principles. The motivation for using REST architectural style has been added in section 4.1.1. The client’s APIs have been simplified and enriched. Two new Java clients have been released: Resource Registry Context Client and Resource Registry Schema Client (see sections 5.1 and 5.2 respectively).

This deliverable presents, in Section 2, the principles and guidelines that govern the implementation of the JRR which has been designed to support the persistence of the PARTHENOS Entities. The JRR is implemented as a tailored information system capable of satisfying the evolution of the model itself, the main features of which are described in Section 3 since the facet-based resource model is extensively referred to throughout this report. Entities, Resources, Facets and Relations are described in detail. Section 4 describes the set of technical components comprising the JRR and APIs, covering the architecture which includes the Resource Registry Service, Context Client, Schema Client, Publisher and Client. Section 5 provides information regarding how to interact with the Resource Registry Service by exploiting the Context and Schema Port Types. The REST APIs are also presented for each functionality. Section 6 covers the backend database, OrientDB, a Multi-Model Open Source NoSQL DBMS that brings together the power of graphs and the flexibility of documents into one scalable high-performance operational database. The final section provides information on the Studio GUI used by the Content Administrator for searching between the created types and inspection of their schema.

2 Introduction

The Joint Resource Registry (JRR) has been designed to support the persistence of the PARTHENOS Entities. It is implemented as a tailored information system capable of satisfying the evolution of the model itself. Moreover, it contributes to the large gCube open-source framework as presented in the deliverable D6.1 PARTHENOS Cloud Infrastructure. In this Section, the role of this tailored information system is first clarified and then the functional and non-functional requirements are illustrated.

2.1 Definition

Several definitions of an Information System (IS) exist. Each definition aims to capture either a specific role or a specific behaviour in systems managing some kind of information.
It is quite common to define an IS as "any organized system for the collection, organization, storage and communication of information". The Encyclopaedia Britannica defines an IS as "an integrated set of components for **collecting**, **storing**, and **processing data** and for **providing information**, knowledge, and digital products".

All the definitions convey the characteristics of Information. Information consists of data that:

- is accurate and timely,
- is specific and organized for a purpose,
- is presented ***within a context*** that gives it meaning and relevance,
- can increase understanding and ***decrease uncertainty.***

According to the Business Dictionary, an information system is "a combination of hardware, software, infrastructure and **trained personnel** organized to facilitate planning, control, coordination, and **decision making in an organization**". In this context, trained personnel are illustrated as:

- human resources;
- procedures for using, operating, and maintaining the information system;
- set of basic principles and associated guidelines, a.k.a. policies, formulated and enforced to direct and limit actions in pursuit of long-term goals.

Looking at the MIT Press, an information system is "a software system to capture, transmit, store, retrieve, and manipulate data **produced by software systems** to provide access to information, thereby supporting people, organizations, or **other software systems**". This definition makes it evident that software systems are both producers and consumers of the Information System making it the core of their business activities.

In the context of the Research Infrastructures[1] and system of systems, we can define an information system (IS) as a software system

- to capture, transmit, store, retrieve, and manipulate data **produced by software systems;**
- to provide access to information **-** ***organized for a purpose and within a contextual domain*** - that are used, accessed, and maintained according to **well-known procedures** operated under the limit of the (evolving) **organization policies;**
- to support people within an organization and **other software systems.**

2.2 Requirements

The analysis of the requirements of an information system capable of providing support for a Research Infrastructure led to identification of the functionality the system has to provide (functional requirements) and the constraints and performances it has to respect (non-functional requirements). Functional requirements define a function of a system or its components. Non-Functional requirements specify criteria that can be used to evaluate the operation of a system, rather than specific behaviour.

It is important to stress that there is an interdependence between these two types of requirements and the boundaries are not always explicit[2]. The provided implementation of a functional requirement could impact on the achievement of a non-functional requirement. Moreover, the practical achievement is a mix of architectural software design and deployment architecture. The design may enable more than one deployment architecture and this is important to accommodate the needs of the specific scenarios where the system has to be exploited.

2.2.1 Functional Requirements

IEEE has defined **Functional Requirements** as "*A requirement that specifies a function that a system or system component must be able to perform*"[3]According to this definition, the following requirements have been identified:

- **Contexts management**: the system must support the management of different context to allow segmentation and sharing of instances among the different context. A typical context is a Virtual Research Environment (VRE).
- **Types management**: The system must enable model definition (e.g., PARTHENOS Entities Model). This requires:
  - the definition of a Data Definition Language (DDL);
  - the design of dedicated APIs.
- **Instances management**: Create, Read, Update, Delete (CRUD) of any entity and relation types defined in the system. To support these requirements the system must:

[1] Research Infrastructures are facilities that provide resources and services for research communities to conduct research and foster innovation. They can be used beyond research for education or public services and they may be single-sited, distributed, or virtual. They include: major scientific equipment or sets of instruments; collections, archives or scientific data; computing systems and communication networks; any other research and innovation infrastructure of a unique nature which is open to external users.

[2] Jonas Eckhardt, Andreas Vogelsang, and Daniel Mèndez Fernández. Are "non-functional" requirements really non-functional? An investigation of non-functional requirements in practice. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE) , pages 832–842, May 2016

[3] IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

- provide a Data Manipulation Language (DML);
    - support univocal identification of any entities and relations;
    - support instances validation against the registered schema.
- **Referential Integrity** is a property of data stating references within it are valid[4]. A referential integrity constraint is defined as part of an association between two entity types. The purpose of referential integrity constraints is to ensure that valid associations always exist[5];
- **Propagation Constraints**: the system has to enforce the defined remove propagation constraints to the target entity if a client deletes either the source entity or the relation between that source and target entities;
- **Multi-tenancy support**: exploitation of resources in contexts, hereafter shortly referred as multi-context by offering:
    - APIs to share instances across contexts;
    - consistent views across contexts: entity and relation representations must be observable at the same time from any context the instances belong to;
    - context views as well as global view at any level of the context hierarchy;
    - propagation constraints enforcement to observe both add and remove propagation constraints to the target entity if a client adds or remove either the source entity or the relation between that source and target entities;[6][7]
- **Dynamic Query** (no pre-defined query): capabilities of a system allowing clients to build their own query and submit it to the system with no long-term impact on the JRR. With regard to relational databases, this characteristic seems obvious (provided by SQL). Unfortunately, especially with the new trend of NoSQL, this same functionality is not supported by some types of NoSQL databases;
- **Standard Abstraction** (desiderata) as far as the relational databases respect SQL standard dialect, it is a desiderata that the JRR supports a standard family of query language;
- **Subscription Notification** support allows *"full decoupling of the communicating entities in time, space, and synchronization"* [8] which reflect the nature of loosely coupled nature of distributed interaction in large-scale applications (such as a Research Infrastructure). By providing this functionality, the possibility to construct event-based services and to improve the scalability of the system will be ensured.

2.2.2 Non-Functional Requirements

Commonly Non-Functional Requirements are identified as *"requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviours"*[9]. Unfortunately, there is no consensus in the scientific community on a non-functional requirements definition. Martin Glinz[10] has defined taxonomy to identify non-functional requirements. In particular, a non-functional requirement can be:

[4] https://en.wikipedia.org/wiki/Referential_integrity

[5] https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/referential-integrity-constraint

[6] https://en.wikipedia.org/wiki/Referential_integrity

[7] https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/referential-integrity-constraint

[8] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. DOI=http://dx.doi.org/10.1145/857076.857078. ACM Comput. Surv. 35, 2 (June 2003), 114-131.

[9] https://en.wikipedia.org/wiki/Non-functional_requirement

[10] M. Glinz. On non-functional requirements. In Proc. 15th IEEE Int. Requirements Eng. Conf., 2007.

- An attribute: is a performance requirement or a specific quality requirement;
    - A performance requirement is a requirement that pertains to a performance concern;
    - A specific quality requirement is a requirement that pertains to a quality concern other than the quality of meeting the functional requirements.
- A constraint: is a requirement that constrains the solution space beyond what is necessary for meeting the given functional, performance, and specific quality requirements.

Under the above-mentioned definition and the taxonomy fall:

- **High Availability (HA)** is a characteristic of a system, which aims to ensure an agreed level of operational performance, usually uptime, for a higher than normal period [11];
- **Eventual Consistency** is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value [12] . The Consistency, Availability, Partitionability (CAP) theorem[13] states that it is impossible for a distributed computer system to provide more than two of the following three guarantees:
    - Consistency (C): every read receives the most recent write or an error;
    - Availability (A): every request receives a response, without a guarantee that it contains the most recent version of the information;
    - Partitionability (P): the system continues to operate despite arbitrary partitioning due to network failures.

Given the CAP theorem and the fact that Availability and Partitionability are mandatory requirements, we selected the Eventual Consistency requirement instead of the stronger Consistency;

- **Horizontal Scalability**. Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth [14]. Horizontally scalability (or scale out/in) means adding more nodes to (or remove nodes from) a system, such as adding a new computer to a distributed software application.
- **Multi-Tenancy**, i.e. a single instance of the technology should be able to serve many “independent” contexts (between the same Application Domain) [15];
- **EUPL** licence **compatibility** of all its components.

[11] https://en.wikipedia.org/wiki/High_availability

[12] Werner Vogels. 2009. Eventually consistent. Commun. ACM 52, 1 (January 2009), 40-44. DOI: https://doi.org/10.1145/1435417.1435432

[13] Eric A. Brewer. Towards robust distributed systems (abstract). In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC ’00, pages 7–, New York, NY, USA, 2000. ACM.

[14] André B. Bondi. 2000. Characteristics of scalability and their impact on performance. In Proceedings of the 2nd international workshop on Software and performance (WOSP '00). ACM, New York, NY, USA, 195-203. DOI=http://dx.doi.org/10.1145/350391.350432

[15] Please note that different Application domain must be managed by completely separated instances of the whole IS.

2.3 Architecture

The architecture of this information system comprises several components. It includes the software components dealing with the generic and the tailored entities models; the services components implementing the capabilities to interact with those entities; the backend database used to persist the entities; and finally, the graphical user interface oriented for human exploitation and visualization of the entities.

The architecture of the information system is, therefore, composed of the following software components:

- Facet Based Resource Model libraries:
    - Information System Model library;
    - gCube Model library;
    - PARTHENOS Model library.
    -
- Joint Resource Registry:
    - Resource Registry Service;
    - Resource Registry Context Client;
    - Resource Registry Schema Client;
    - Resource Registry Publisher;
    - Resource Registry Client.
    -
- Backend Database (i.e. OrientDB as Graph Database);
    -
- Information System Subscription Notification Service;
    -
- Graphical User Interface (GUI).

3 Facet Based Resource Model

The PARTHENOS Joint Resource Registry Data Model is extensively presented in Section 6 of the deliverable *D5.2 Design of the Joint Resource Registry*. In the following sections, some basic information about the Resource Model is reported since this is largely used in the remaining part of this document.

3.1 Information System Model

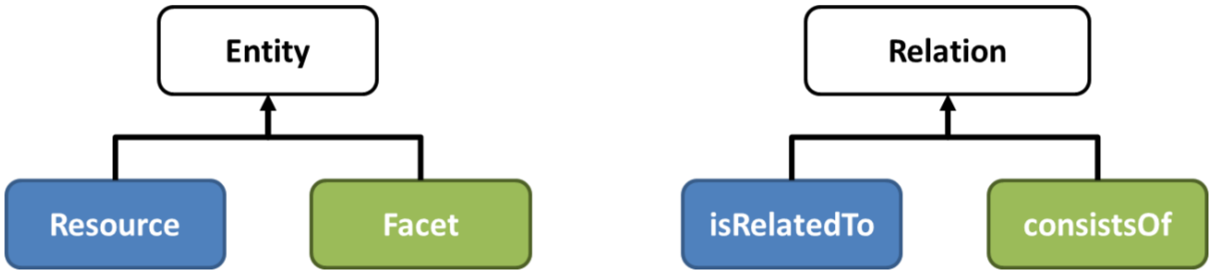
3.1.1 Basic Concept

Two typologies of **Entities** are envisaged:

- **Resources**, i.e. entities representing a description of "thing" to be managed;

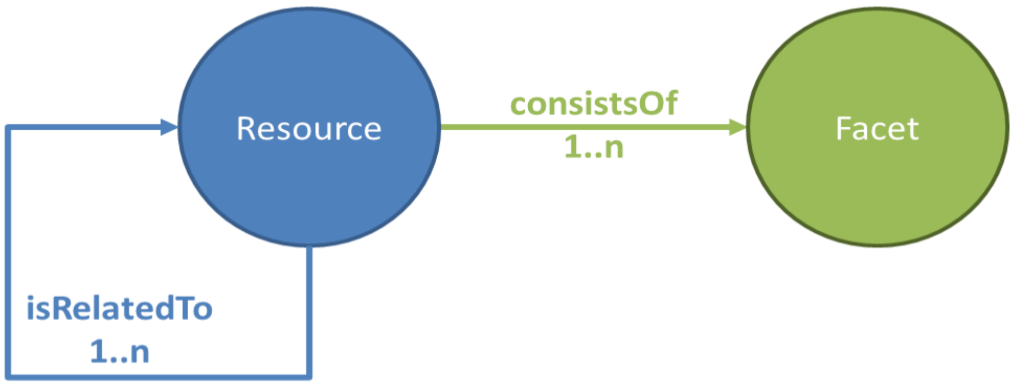
Every Resource is characterized by a number of Facets;

- Facets, i.e. entities contributing to "build" a description of a Resource. Every facet, once attached to a Resource profile, captures a certain aspect / characterization of the resource. Every facet is characterized by a number of properties.



Two typologies of **Relations** are envisaged:

- **isRelatedTo**, i.e. a relation linking any two Resources.
- **consistsOf**, i.e. a relation connecting each Resource with one of the Facets characterizing it;



Each Entity and Relation

- has a **header** automatically generated for the sake of identification and provenance of the specific information;
- can be specialized
  - A number of specializations are identified below. Such specialisations are managed by the gCube Core services, i.e. Core services builds upon these specialisations to realise its management tasks;
  - Other specialisations can be defined by clients, the system make it possible to store these additional typologies of relations and facets and to discover them.

Facet and **Relation** instances can have additional properties which are not defined in the schema (henceforth schema-mixed mode).

Relation properties:

- Any relation has a direction, i.e. a "source" (**out** bound of the relation) and a "target" (**in** bound of the relation). Anyway, the relation can be also navigated in the opposite direction;
- It is not permitted to define a **Relation** having a **Facet** as "source"**.** In other words:
  - It is not permitted to define a **Relation** connecting a **Facet** with another one;
  - It is not permitted to define a **Relation** connecting a **Facet** with a **Resource** (as target);
- A **Facet** instance can be linked (by **consistsOf** or any specialization of it) from different **Resources**.

Any Property can be enriched with the following attributes:

- **name*** (String): the property name;
- **type***: the type of the property (e.g. String, Integer, ...);
- **description** (String, default=null): the description of the property.
- **mandatory** (Boolean, default=false): indicate if the property is mandatory or not;
- **readOnly** (Boolean, default=false): the property cannot change its value;
- **notNull** (Boolean, default=false): whether the property must assume a value diverse from 'null' or not;
- **max** (Integer, default=null): whether the property can be limited to a maximum value;
- **min** (Integer, default=null): whether the property can be limited to a minimum value;
- **regexpr** (String, default=null): a regular expression[16] to validate the property.

[16] https://en.wikipedia.org/wiki/Regular_expression

Table 1: Basic Property Types

Type	Java type	Description
Boolean	java.lang.Boolean or boolean	Handles only the values True or False.
Integer	java.lang.Integer or int or java.math.BigInteger	32-bit signed Integers.
Short	java.lang.Short or short	Small 16-bit signed integers.
Long	java.lang.Long or long	Big 64-bit signed integers.
Float	java.lang.Float or float	Decimal numbers.
Double	java.lang.Double or double	Decimal numbers with high precision.
Date	java.util.Date	Any date with the precision up to milliseconds.
String	java.lang.String	Any string as alphanumeric sequence of chars.
Embedded	?extends org.gcube.informationsystem.model.embedded.Embedded	This is an Object contained inside the owner Entity and has no Header. It is reachable only by navigating the owner Entity.
Embedded list	List<? extends org.gcube.informationsystem.model.embedded.Embedded>	List of Objects contained inside the owner Entity and have no Header. They are reachable only by navigating the owner Entity.
Embedded set	Set<? extends org.gcube.informationsystem.model.embedded.Embedded>	Set (no duplicates) of Objects contained inside the owner Entity and have no Header. They are reachable only by navigating the owner Entity.
Embedded map	Map<String, ? extends org.gcube.informationsystem.model.embedded.Embedded>	Map of Objects contained inside the owner Entity and have no Header. They are reachable only by navigating the owner Entity.
Byte	java.lang.Byte or byte	Single byte. Useful to store small 8-bit signed integers.
Binary	java.lang.Byte[] or byte[]	Can contain any value as byte array.

Table 2: Derived Property Types

Type	Java type	Description
Enum	java.lang.Enum or enum	By default, it is represented using the String representation of the Enum so that the primitive type used will be String. The enumeration is checked by setting the Regexpr property. The Regular Expression is auto-generated and it will be something like ^(FIRST-ENUM-STRING_REPRESENTATION|SECOND-ENUM-STRING_REPRESENTATION|...|LAST_ENUM_STRING_REPRESENTATION)$. Otherwise (if indicated using an annotation), it can be represented using the Integer value of the Enum so that the primitive type used will be Integer. The enumeration is checked using Max and Min properties.
UUID	java.util.UUID	String representation of the UUID. The check is obtained using the regular expression ^([a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}){1}$
URL	java.net.URL	String representation of the URL. No check actually.
URI	java.net.URI	String representation of the URI. No check actually.

Table 3: Header

Name	Type	Attributes	Description

uuid	UUID	Mandatory=true
		NotNull=true
		ReadOnly=true
			This uuid can be used to univocally identify the Entity or the Relation

creator	String	Mandatory=true
		NotNull=true
		ReadOnly=true
			Filled at creation time. The creator is retrieved using the authorization token

creationTime	Date	Mandatory=true
		NotNull=true
		ReadOnly=true
			Creation time in milliseconds. Represent the difference, measured in milliseconds, between the creation time and midnight, January 1, 1970 UTC

lastUpdateTime	Date	Mandatory=true
		NotNull=true
			Last Update time in milliseconds. Represent the difference, measured in milliseconds, between the last update time and midnight, January 1, 1970 UTC

Table 4: Propagation Constraints

Name	Type	Attributes	Description

remove	Enum	Mandatory=true
		NotNull=true
		Regex=(cascadeWhenOrphan|cascade|keep)
			Indicate the behaviour to Resource Registry to be applied to the target Entity when the source Entity is remove from context or deleted

add	Enum	Mandatory=true
		NotNull=true
		Regex=(propagate|unpropagate)
			Indicate the behaviour to Resource Registry to be applied to the target Entity when the source Entity is added to Context

Any Relation contains such a property. If the values are not specified at creation time, the system will initialize it following the following rules:

- IsRelatedTo Relation: remove=keep, add=unpropagate
- ConsistsOf Relation: remove=cascadeWhenOrphan, add=propagate

3.1.2 Entity

The resource entity is conceived to describe every "main thing" to be registered in and discovered through the Joint Resource Registry.

Table 5: Resource

Resource				
Source	**Relation**	**Multiplicity**	**Target**	**Description**
Facets				
Resource	isIdentifiedBy	1..n	Facet	Any Resource has at least one Facet which in some way allow to identify the Resource per se.
Resource	consistsOf	0..n	Facet	Any Resource consist of zero or more Facets which describes the different aspects of the facet.
Relations				
Resource	isRelatedTo	0..n	Resource	Any Resource can be related to any other resource.

3.1.3 Facet

Facets are collections of attributes conceived to capture a certain feature / aspect of the resource they are associated with.

Every Facet has:

- A Header automatically generated to capture identification- and provenance-related aspects of the facet once it is instantiated;
- Zero or more properties. Besides the per-facet envisaged properties, clients can add new ones.

3.1.4 Relation

Every relation has:

- A Header
- A PropagationConstraint
- Zero or More properties (not necessarily predefined, similarly to Facets).

Table 6: isRelatedTo

Source	Relation	Multiplicity	Target	Description
Resource	isRelatedTo	0..n	Resource	A relation linking any two Resources.

Default PropagationConstraint has the following values: remove=keep, add=unpropagated.

Table 7: consistOf

Source	Relation	Multiplicity	Target	Description
Resource	consistsOf	1..n	Facet	A relation connecting each Resource with one of the Facet characterizing it.

Default PropagationConstraint has the following values: remove=cascadeWhenOrphan, add=propagate.

Table 8: isIdentifiedBy

Source	Relation	Multiplicity	Target	Description
Resource	isIdentifiedBy	1..n	Facet	A relation connecting each Resource with one of the Facet which can be used to identify the Resource.

4 Joint Resource Registry

The **Joint Resource Registry** is designed to support the following operations:

- To capture, transmit, store, retrieve and manipulate data from any software system enabled on the infrastructure, including:
    - Location and properties;
    - Status, load, exploitation usage, and accounting data.
- To provide access to information, organized to enable:
    - Monitoring, validation, and reporting;
    - Elasticity and pooling of resources;
- To support any software system to:
    - Discover services and infrastructure resources.

The Joint Resource Registry enables:

- a set of resource management functions
    - enabling functions:
        - publication, discovery;
        - monitoring, deployment;
        - contextualization, security, execution.
    - data management functions:
        - access, store;
        - index, search;
        - transfer, transform.
- a set of applications
    - built around those functions;
- an abstract view over functions
    - defined by specifications;
    - multiple implementations, over time / concurrently.
- secure and consistent entities evolution
    - tailored support for facet and resource definition;
    - implementations produce/consume different facets, independently.
- dynamic resource semantics
    - no longer predefined in class hierarchies;
    - implicitly captured by current facets;
    - changes over time / across “similar” resources.

4.1 Architecture

The constituent software components are:

- Resource Registry Service,
- Resource Registry Context Client,
- Resource Registry Schema Client,
- Resource Registry Publisher,
- Resource Registry Client.

4.1.1 Resource Registry Service

The Resource Registry Service is a web service running on SmartGears responsible for storing information, in particular the global and partial view of:

- the resources (e.g. computing, storage, services, software, datasets);
- their current status (e.g. up and running, available);
- their relationships with other resources;
- the policies governing their exploitation.

The Resource Registry is developed only by using the concepts defined in the Information System Model and it provides the capabilities to enrich its knowledge by creating new types of entities and relations and their schemas. The Resource Registry is capable of serving different applications domains (i.e. Context). To achieve this goal, the Resource Registry provides capabilities for managing Contexts (the contexts are hierarchical) and associating the entities and relations to one or more of the Contexts as requested by the different clients. The Resource Registry is also responsible for notifying any update to or creation of any entity or relation to **Information System Subscription Notification Service**.

To reach its goals, the Resource Registry offers four port types:

- **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 as REST[17] API. [18]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 such a 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 provides any concrete guideline or architecture. An example of concrete architecture for REST is ROA (Resource Oriented Architecture) which is based on HTTP 1.1. The design of the Resource Registry service

[17] https://en.wikipedia.org/wiki/Representational_state_transfer
[18] https://en.wikipedia.org/wiki/JSON

follows the ROA guidelines. In particular, every REST API is JSON[19] based. This means that any content present in an HTTP request is formatted using the JSON standard.

ROA uses standards HTTP methods applied to URI to realise Create, Read, Update, Delete (CRUD) operations. Most used HTTP methods in ROA are POST, GET, PUT and DELETE. According to HTTP specification [20] [21]:

- POST is used to create a new resource without providing the URI of creating resource. The representation of the resource is sent, as part of the HTTP body, via POST to the collection that will contain the resource. The server determines its appropriate location, and it provides the resulting URI to the client. Also, PUT can be used to create a new resource if the client provides the URI where the resource will become available;
- GET is used to get the information about a resource;
- PUT is used to update an existing resource. This operation instructs the server to apply a new representation as a replacement of the previous one;
- DELETE is used to delete an existing resource.

GET, PUT and DELETE must be idempotent, i.e., the same operation repeated multiple times has the same side effect than using it one time. Request For Comments (RFC) 7231 clarifies that “repeating the request will have the same intended effect, even if the original request succeeded, though the response might differ” [15]. GET must have no side effect, and this is also known as safe operation. “This does not prevent an implementation from including behaviour that is potentially harmful, that is not entirely read-only, or that causes side effects while invoking a safe method” [15].

Table 9: Mapping between HTTP methods and CRUD operations

CRUD Operation	HTTP Method	Safe	Idempotent
Create	POST	No	No
Read	GET	Yes	Yes
Update[22]	PUT	No	Yes
Delete	DELETE	No	Yes[23]

The table shows the mapping between HTTP methods and CRUD operations. Moreover, it shows the property of safety and idempotency the methods must satisfy.

[19] https://en.wikipedia.org/wiki/JSON
[20] Henrik Frystyk Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.
[21] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, June 2014.
[22] Also Create to the URI where the resource will be available.
[23] Allamaraju (Subbu Allamaraju. RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity. O’Reilly, first edition, Nov 2010) argues that DELETE idempotency should be accomplished client-side. The server should inform the client if a delete succeeded because the resource was really deleted or it was not found i.e., 404 Not Found error is suggested instead of 204 No Content. The latter situation should be treated as idempotent by the client.

ROA gives particular emphasis on "make it a resource" paradigm and proposes descriptive and predictable URI as technology to satisfy the resource identification constraint. Hence, any resource in ROA has a URI.

4.1.2 Resource Registry Context Client

The Resource Registry Context Client is a java library providing RPC facilities to interact with the **Context Management** port type. The library hides all the complexity of marshalling and un-marshalling of requests and results. By using this library, any client can manage java classes instead of JSON objects.

4.1.3 Resource Registry Types Client

The Resource Registry Schema Client is a java library providing RPC facilities to interact with the **Types Management** port type. The library hides all the complexity of marshalling and un-marshalling of requests and results. By using this library, any client can manage java classes instead of JSON objects.

4.1.4 Resource Registry Publisher

The Resource Registry Publisher is a java library providing RPC facilities to interact with the **Instances Management** port type. The library hides all the complexity of marshalling and un-marshalling of requests and result. By using this library any client can manage java classes instead of JSON objects.

4.1.5 Resource Registry Client

The Resource Registry Client is a java library providing RPC facilities to interact with the **Query & Access** port type. The library hides all the complexity of marshalling and un-marshalling of requests and result. By using this library any client manages java classes instead of JSON objects.

5 Interacting with Resource Registry Service

This section provides information regarding how to interact with the Resource Registry Service by exploiting the Context and Schema Port Types. The REST APIs are also presented for each functionality. Please note that the provided examples may intentionally hide some details in the response to avoid unneeded complexity.

Resource Registry is a web service which represents the core component of the JRR. It is designed to comply with Resource Oriented Architecture (ROA) by grouping the required management APIs logically and making them "as a resource". The Resource Registry service exposes five port-type. The term port type is used to indicate the first level Uniform Resource Locator (URL) path starting from the service base path, i.e., if the service base path is / then the URLs /a and /b are two different port types. Each port type exposes RESTful APIs to satisfy one or more of the functional requirements identified.

- **Contexts Management:** manages hierarchical contexts. A VRE is a typical context managed by the Resource Registry;
- **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**: manages entities and relations instances;
- **Sharing Management**: manages instances sharing across different contexts;
- **Query and Access**: supports the discovery of instances through access patterns and queries.

The rest of this chapter presents these five port type by describing the exposed REST APIs.

Every port type uses the standard HTTP statuses to provide information regarding the execution of the requested operations. In particular used status code for successful operation are:

- **`200 OK`** (used with `PUT` (update) and `GET`) https://tools.ietf.org/html/rfc7231#section-6.3.1;
- **`201 Created`** (used with `PUT` or `POST` when a resource is created) https://tools.ietf.org/html/rfc7231#section-6.3.2;
- **`204 No Content`** (used with `HEAD` and `DELETE`) https://tools.ietf.org/html/rfc7231#section-6.3.5.

The most common error status a client can obtain are

- **`400 Bad Request`** used to indicate a clients error https://tools.ietf.org/html/rfc7231#section-6.5.1;
- **`401 Unauthorized`** used to indicate that the client has not enough right to perform such request https://tools.ietf.org/html/rfc7235#section-3.1;

- `404 Not Found` used to indicate that the requested instance does not exists https://tools.ietf.org/html/rfc7231#section-6.5.4;
- `405 Method Not Allowed` the used HTTP method is not supported for the requested URL https://tools.ietf.org/html/rfc7231#section-6.5.5. The response contains the Allow HTTP Header indicating the supported HTTP method for such URL https://tools.ietf.org/html/rfc7231#section-7.4.1;
- `409 Conflict` the request could not be completed due to a conflict with the current state of the target resource. https://tools.ietf.org/html/rfc7231#section-6.5.8.];
- `500 Internal Server Error` indicate a server failure. https://tools.ietf.org/html/rfc7231#section-6.6.1.

5.1 Context Management

It is responsible for managing Context belonging to the same Application Domain.
The security configuration based on the Authorization Framework makes this port type accessible only from the Resource Manager. In other words, no others client is allowed to manage Context other than the Resource Manager. See *D6.1 PARTHENOS Cloud infrastructure* for details about the Resource Manager and the Authorization Framework.

Context requirements are:

- No predefined number of levels;
- Possibility to change the name of the Context with no impact for any component;
- Possibility to move a Context from a parent Context to another.

Available Methods:

- **Listing**: allows to enumerate the contexts;
- **Create**: allows to create a new context as a child of another context (if any). The context has a name;
- **Exists**: allows to check if a Context exists;
- **Read**: allows to read a Context;
- **Update**: allows to rename a context or to move a context as a child of another Context;
- **Delete**: allows to delete a Context.

Table 10: Context Management Operations, Methods and URLs

Operation	HTTP Method	URL
Listing	GET	/contexts
Create	PUT	/contexts/{CONTEXT_UUID}
Read	GET	/contexts/{CONTEXT_UUID}
Exists	HEAD	/contexts/{CONTEXT_UUID}
Update	PUT	/contexts/{CONTEXT_UUID}
Delete	DELETE	/contexts/{CONTEXT_UUID}

Any request to this port type has success if the following guarantees are satisfied:

- the hierarchy of contexts is a tree with an arbitrary number of levels;
- two contexts with the same name can only exist if they have different parents;
- any update to a context does not have any side effect on the instances belonging to the context;
- it is not possible to delete a context if it contains instances. It is a responsibility of the clients to remove the instances from the context (or delete them) before trying to delete the context.

This section provides information regarding how to interact with Resource Registry Service for Context Management. Apart from the REST API, this port type can be used also by using Resource Registry Context Client Java library. Both REST and Java APIs are presented. The provided examples can intentionally hide some details to avoid unneeded complexity.

Resource Registry Context Client has the following Maven coordinates:

```
<dependency>
	<groupId>org.gcube.information-system</groupId>
	<artifactId>resource-registry-context-client</artifactId>
	<version>[1.0.0-SNAPSHOT, 2.0.0-SNAPSHOT)</version>
</dependency>
```

To use the client, you need first get a `ResourceRegistryPublisher` instance.
By using `ResourceRegistryPublisherFactory.create()` method the library discovers the correct endpoint to interact with the Resource Registry for the current context.

```
SecurityTokenProvider.instance.set("Your-Token-Here");
ResourceRegistryContextClient resourceRegistryContextClient =
	ResourceRegistryContextClientFactory.create();
```

5.1.1 Contexts Listing

GET /contexts

Return the list of existing contexts.

5.1.2 Create Context

Create new Context as child of another Context (if any).

PUT /contexts/{CONTEXT_UUID}

5.1.2.1 Create Context Example 1

Create a new Context with named **ParthenosInfrastructure** with no parent. It is a ROOT Context.

Request URL

PUT /contexts/2705dd32-c857-444b-818a-3ec69e339e5d

Request Body

```
{
	"@class": "Context",
	"name": "ParthenosInfrastructure",
	"header": {
		"@class": "Header",
		"uuid": "2705dd32-c857-444b-818a-3ec69e339e5d"
	}
}
```

Response Body

```
{
	"@class": "Context",
	"name": "ParthenosInfrastructure",
	"header": {
		"@class": "Header",
		"uuid": "2705dd32-c857-444b-818a-3ec69e339e5d",
		"creator": "luca.frosini",
		"modifiedBy": "luca.frosini",
		"creationTime": "2017-03-1711:47:55",
		"lastUpdateTime": "2017-03-17 11:47:55"
	}
}
```

Java API

```
Context parthenosInfrastructure = new ContextImpl("ParthenosInfrastructure");
resourceRegistryContextClient.create(parthenosInfrastructure);
```

5.1.2.2 Create Context Example 2

Create a new Context with named ParthenosVO as child of Context with UUID 2705dd32-c857-444b-818a-3ec69e339e5d (ParthenosInfrastructure)

Request URL

```
PUT /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a
```

Request Body

```
{
	"@class": "Context",
	"name": "ParthenosVO",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a"
	},
	"parent" : "2705dd32-c857-444b-818a-3ec69e339e5d"
}
```

Response Body

```
{
	"@class": "Context",
	"name": "ParthenosVO",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a",
		"creator": "luca.frosini",
		"modifiedBy": "luca.frosini",
		"creationTime": "2017-03-17 11:47:56",
		"lastUpdateTime": "2017-03-17 11:47:56"
	},
	"parent" : "2705dd32-c857-444b-818a-3ec69e339e5d"
}
```

Java API

```
Context parthenosVO = new ContextImpl("ParthenosVO");
parthenosVO.setParent(parthenosInfrastructure);
esourceRegistryContextClient.create(parthenosVO);
```

5.1.3 Read Context

Return the definition of the Context identified by the UUID provided as path parameter.

Request URL

```
GET /contexts/{CONTEXT_UUID}
```

5.1.3.1 *Read Context Example*

Read the Context having UUID 30f6254c-c87a-451e-bc0f-7cfcbd94a84a.

Request URL

```
GET /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a
```

Response Body

```
{
        "@class": "Context",
        "name": "ParthenosVO",
        "header": {
                "@class": "Header",
                "uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a",
                "creator": "luca.frosini",
                "modifiedBy": "luca.frosini",
                "creationTime": "2017-03-17 11:47:56",
                "lastUpdateTime": "2017-03-17 11:47:56"
        },
        "parent" : "2705dd32-c857-444b-818a-3ec69e339e5d"
}
```

Java API

```
resourceRegistryContextClient.read("30f6254c-c87a-451e-bc0f-7cfcbd94a84a");
```

5.1.4 Verify Context

Check if the Context identified by the UUID provided as path parameter exists.

Request URL

```
HEAD /contexts/{CONTEXT_UUID}
```

5.1.4.1 Verify Context Examples

Check the Context having UUID 30f6254c-c87a-451e-bc0f-7cfcbd94a84a.

Request URL

```
HEAD /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a
```

If the context exist the response HTTP status code is `204 No Content`, otherwise is `404 Not Found`.

Java API

```
resourceRegistryContextClient.exists("30f6254c-c87a-451e-bc0f-7cfcbd94a84a");
```

5.1.5 Update Context

Rename or move a Context identified by the UUID provided as path parameter.

Request URL

```
PUT /contexts/{CONTEXT_UUID}
```

5.1.5.1 Rename Context Example

Rename a Context 30f6254c-c87a-451e-bc0f-7cfcbd94a84a (was ParthenosVO) to the new name ParthenosCommunity.

Request URL

```
PUT /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a
```

Request Body

```
{
	"@class": "Context",
	"name": "ParthenosCommunity",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a"
	},
	"parent" : "2705dd32-c857-444b-818a-3ec69e339e5d"
}
```

Response Body

```
{
	"@class": "Context",
	"name": "ParthenosCommunity",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a",
		"creator": "luca.frosini",
		"modifiedBy": "luca.frosini",
		"creationTime": "2017-03-17 11:47:56",
		"lastUpdateTime": "2017-03-17 11:57:36"
	},
	"parent" : "2705dd32-c857-444b-818a-3ec69e339e5d"
}
```

Java API

```
parthenosVO.setName("ParthenosCommunity");
Context parthenosCommunity =   resourceRegistryContextClient.update(parthenosVO);
```

5.1.5.2 Move Context Example

Move the Context 30f6254c-c87a-451e-bc0f-7cfcbd94a84a as ROOT Context.

Request URL

PUT /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a

Request Body
```
{
	"@class": "Context",
	"name": "ParthenosCommunity",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a"
	},
	"parent" : null
}
```

Response Body
```
{
	"@class": "Context",
	"name": "ParthenosCommunity",
	"header": {
		"@class": "Header",
		"uuid": "30f6254c-c87a-451e-bc0f-7cfcbd94a84a",
		"creator": "luca.frosini",
		"modifiedBy": "luca.frosini",
		"creationTime": "2017-03-17 11:47:56",
		"lastUpdateTime": "2017-03-17 11:58:21"
	}
}
```

Java API

```
parthenosCommunity.setParent(null);
resourceRegistryContextClient.update(parthenosCommunity);
```

5.1.6 Delete

Delete the Context identified by the UUID provided as path parameter.

Request URL

DELETE /contexts/{CONTEXT_UUID}

5.1.6.1Delete Context Example

Delete the Context having UUID 30f6254c-c87a-451e-bc0f-7cfcbd94a84a.

Request URL

PUT /contexts/30f6254c-c87a-451e-bc0f-7cfcbd94a84a

The returned HTTP status is `204 No Content`.

Java API

```
resourceRegistryContextClient.delete(parthenosCommunity);
```

5.2 Types Management

Types Management is responsible for managing the instantiation of the IS Model by allowing the definition of entities, relations and embedded types and their schema. Giving the REST principle Manipulation Of Resources Through Representations the defined Data Definition Language (DDL) (see functional requirement) is a specification of the representation of a type.

This port type exposes the following APIs:

- **Listing**: allows to enumerate the types;
- **Create**: allows to create a new type;
- **Exists**: allows to check if a type exists;
- **Read**: allows to read a type definition;
- **Delete**: allows to delete a type.

Table 11: Types Management Operations, Methods and URLs

Operation	HTTP Method	URL
Listing	GET	/types
Create	PUT	/types/{TYPE_NAME}
Read	GET	/types/{TYPE_NAME}
Exists	HEAD	/types/{TYPE_NAME}
Delete	DELETE	/types/{TYPE_NAME}

Types Management does not provide the capability to update the specification of a type. No one could know what the impact on changing the schema of a type would be because potentially any client could create them. The IS Model provides by design the support for evolution via inheritance and schema-mixed mode.

The Types Management implements the following policies:

- it ignores all the properties a client tries to define for a resource;
- it deletes a type only if the type has no instances and no sub-types;
- it supports multiple inheritance;
- it enforces inheritance rules. A type can have only one ancestor between Resource, Facet, isRelatedTo, consistsOf. The specialisations of any relation must have source and target entity types hierarchically compatibles with the parent relations.

This section provides information regarding how to interact with Resource Registry Service for Types Management. Apart from the REST API, this port type can be used also by using Resource Registry Schema Client Java library. Both REST and Java APIs are presented. The provided examples can intentionally hide some details to avoid unneeded complexity.

Resource Registry Schema Client has the following Maven coordinates:

```
<dependency>
        <groupId>org.gcube.information-system</groupId>
        <artifactId>resource-registry-schema-client</artifactId>
        <version>[1.0.0-SNAPSHOT, 2.0.0-SNAPSHOT)</version>
</dependency>
```

To use the client, you need first get a `ResourceRegistrySchemaClient` instance.
By using `ResourceRegistrySchemaClientFactory.create()` method the library discovers the correct endpoint to interact with the Resource Registry for the current context.

```
SecurityTokenProvider.instance.set("Your-Token-Here");
ResourceRegistrySchemaClient resourceRegistrySchemaClient =
        ResourceRegistrySchemaClientFactory.create();
```

5.2.1 Type Definition

Any Type is described by the following attributes:

- **name*** (String): the type name;
- **description** (String, default=null): the description of the type;
- **abstract** (Boolean, default=false): indicate if the type is an abstract or concrete. It is not possible to instantiate an abstract type;
- **superclasses*** (List<String>): the list of all super types of this type. Multiple Inheritance is supported.
- **properties**: zero or more properties. Any property is described by the following attributes:
    - **name*** (String): the property name;
    - **type***: the type of the property (e.g. String, Integer, ...);
    - **description** (String, default=null): the description of the property.
    - **mandatory** (Boolean, default=false): indicate if the property is mandatory or not;
    - **readOnly** (Boolean, default=false): the property cannot change its value;
    - **notNull** (Boolean, default=false): whether the property must assume a value diverse from 'null' or not;
    - **max** (Integer, default=null): whether the property can be limited to a maximum value;
    - **min** (Integer, default=null): whether the property can be limited to a minimum value;
    - **regexpr** (String, default=null): a regular expression[24] to validate the property.

Each Relation has two additional mandatory attributes:

- **source**: indicates the required source type to instantiate such a relation;
- **target**: indicates the required target type to instantiate such a relation.

Each Resource has two arrays (instead of properties):

[24] https://en.wikipedia.org/wiki/Regular_expression

- **facets**: this array defines which Facet types describe the resource and which consistsOf relation type must be used to connect the facet instance;
- **resources**: this array defines which other Resource types could be related to the defined type and which isRelatedTo relation type could be used to connect the instances of them. The array contains only the outbound relations.

Each element of the 'facets' and 'resources' arrays contained in the Resource definition is composed of six attributes (* indicates a mandatory attribute):

- **source*** (String): it is always the name of the defined target type;
- **relation*** (String): the relation type name to be used to connect the source type to the target type;
- **target*** (String): the target type name of the relation [String];
- **description** (String, default=null): the description of the reason why the source and the target should be related;
- **max** (Integer, default=null): the upper bound number of relations between the source and target types (null means unbounded);
- **min** (Integer, default=null): the lower bound number of relations between the source and target types (null is the same as zero which means that the relation is optional). Optional relations are specified as to provide suggestion to whom is interested in instantiating the resource type.

5.2.2 Type Creation

Allow to create new Entity or Relation or Embedded Type.

Request URL

```
PUT /types/{TYPE_NAME}
```

5.2.2.1 Resource Type Creation Example

```
PUT /types/Actor
```

Request Body

```
{
	"name": "Actor",
	"description": "Any entity (human or machine) playing an active role.",
	"abstractType": true,
	"superclasses": ["Resource"],
	"facets": [
		...
	],
	"resources": [
		...
	]
}
```

Java API

```
public interface Actor extends Resource {

	public static final String NAME = "Actor";
	public static final String DESCRIPTION = "Any entity (human or machine) playing an active role.";
	public static final String VERSION = "1.0.0";

	...
}

resourceRegistrySchemaClient.create(Actor.class);
```

5.2.2.2 Facet Type Creation Example

PUT /types/ContactFacet

Request Body

```
{
	"name": "ContactFacet",
	"description": "This facet is expected to capture contact information",
	"abstractType": false,
	"superclasses":["Facet"],
	"properties":[
		{
			"name": "name",
			"description": "First Name",
			"mandatory": true,
			"readonly": false,
			"notnull": true,
			"max": null,
			"min": null,
			"regexpr": null,
			"linkedType": null,
			"linkedClass": null,
			"type": 7 /* String*/
		},
		...,
		{
			"name": "eMail",
			"description": "A restricted range of RFC 822 compliant email address. ... ",
			"mandatory": true,
			"readonly": false,
			"notnull": true,
			"max": null,
			"min": null,
			"regexpr":"^[a-z0-9._%+-]{1,128}@[a-z0-9.-]{1,128}$",
			"linkedType": null,
			"linkedClass": null,
			"type":7 /* String */
		}
	]
}
```

Java API

```
public interface ContactFacet extends Facet {

        public static final String NAME = "ContactFacet";
        public static final String DESCRIPTION = "This facet is expected to capture contact information";
        public static final String VERSION = "1.0.0";

        public static final String EMAIL_PROPERTY = "eMail";
        public static final String EMAIL_PATTERN = "^[a-z0-9._%+-]{1,128}@[a-z0-9.-]{1,128}$";

        @ISProperty(mandatory=true, nullable=false)
        public String getName();

        public void setName(String name);

        @ISProperty
        public String getTitle();

        public void setTitle(String title);

        @ISProperty
        public String getMiddleName();

        public void setMiddleName(String middleName);

        @ISProperty(mandatory=true, nullable=false)
        public String getSurname();

        public void setSurname(String surname);

        @ISProperty(name=EMAIL_PROPERTY, mandatory=true, nullable=false, regexpr=EMAIL_PATTERN)
        public String getEMail();

        public void setEMail(String eMail);

}
```

5.2.2.3 IsRelatedTo Type Creation Example

PUT /types/Hosts

Request Body

```
{
        "name": "Hosts",
        "description": "...",
        "abstractType": false,
        "superclasses": ["IsRelatedTo"],
        "properties": null,
        "source": "Site",
        "target": "Service"
}
```

Java API

```
public interface Hosts<Out extends Site, In extends Service>
        extends IsRelatedTo<Out, In> {

        public static final String NAME = "Hosts";

}

resourceRegistrySchemaClient.create(Hosts.class);
```

5.2.2.4 ConsistsOf Type Creation Example

PUT /types/HasContact

Request Body

```
{
        "name": "HasContact",
        "description": "...",
        "abstractType": false,
        "superclasses": ["ConsistsOf"],
        "properties": null,
        "source": "Resource",
        "target": "ContactFacet"
}
```

Java API

```
public interface HasContact
                                        <Out extends Resource, In extends ContactFacet>
        extends ConsistsOf<Out, In> {

        public static final String NAME = "HasContact";

}

resourceRegistrySchemaClient.create(HasContact.class);
```

5.2.3 Embedded Type Creation Example

PUT /types/AccessPolicy

Request Body

```
{
        "name": "AccessPolicy",
        "description": "...",
        "abstractType": false,
        "superclasses": ["Embedded"],
        "properties":[
                {
                        "name": "policy",
                        "description": "...",
                        "mandatory": false,
                        "readonly": false,
                        "notnull": false,
                        "max": null,
                        "min": null,
                        "regexpr": null,
                        "linkedType": null,
                        "linkedClass": "ValueSchema",
                        "type": 9 /* Embedded */
                },
                {
                        "name": "note",
                        "description": "...",
                        "mandatory": false,
                        "readonly": false,
                        "notnull": false,
                        "max": null,
                        "min": null,
                        "regexpr": null,
                        "linkedType": null,
                        "linkedClass": null,
                        "type":7 /* String */
                }
        ]
}
```

Java API

```
public interface AccessPolicy extends Embedded {

        public static final String NAME = "AccessPolicy";

        @ISProperty
        public ValueSchema getPolicy();

        public void setPolicy(ValueSchema policy);

        @ISProperty
        public String getNote();

        public void setNote(String note);
}

resourceRegistrySchemaClient.create(AccessPolicy.class);
```

5.2.4 Read Type Definition

It allows to read Type Definition.

Request URL

```
GET /types/{TYPE_NAME}
```

5.2.5 Read a Resource Definition Example

```
GET /types/Actor
```

Response Body
```
{
	"name": "Actor",
	"description": "Any entity (human or machine) playing an active role.",
	"abstractType": true,
	"superclasses": ["Resource"],
	"facets": [
		...
	],
	"resources": [
		...
	]
}
```

Java API

```
resourceRegistrySchemaClient.read("Actor");
```

5.3 Instances Management

The Instances Management port type is responsible for the management of entities and relation instances. It offers the following APIs:

- **Create**: it allows to create a new entity or relation instance in a certain context;
- **Exists**: it allows to check if an instance exists in a certain context;
- **Read**: it allows to get the representation of the requested instance in a certain context;
- **Update**: it allows to update an instance in a certain context;
- **Delete**: it allows to delete an instance.

Table 12: Instances Management Operations, Methods and URLs

Operation	**HTTP Method**	**URL**
Create	PUT	/instances/{TYPE_NAME}/{UUID}
Read	GET	/instances/{TYPE_NAME}/{UUID}
Exists	HEAD	/instances/{TYPE_NAME}/{UUID}
Update	PUT	/instances/{TYPE_NAME}/{UUID}
Delete	DELETE	/instances/{TYPE_NAME}/{UUID}

The Instances Management implements the following policies:

- it manages the Header automatically;
- it allows to identify an instance via the Universally Unique Identifier (UUID) specified in the Header;

- it allows the creation of an instance only if the declared type is already present in the system (previously registered via the Type Management port type);
- it validates the instance against the schema of the defined type;
- it imposes the default values of propagation constraints when the client does not specify their values;
- it guarantees propagation constraints.

The gCube framework uses an authorisation token in the HTTP header to identify the user and the context of each request. The authorisation framework equips the container running the web services. It intercepts any requests, resolves the token and forwards the request to the service if authorised, along with the user and the operating context.

The Resource Registry uses the context to identify the belonging instances and the user to manage the Header properly:

- at creation time to initialise **creator** and **modifiedBy** properties;
- at updated time to updated **modifiedBy** property.

This section provides information regarding how to interact with Resource Registry Service for Instances Management. Apart from the REST API, this port type can be used also by using Resource Registry Publisher Java library. Both REST and Java APIs are presented. The provided examples can intentionally hide some details to avoid unneeded complexity.

Resource Registry Publisher has the following Maven coordinates:

```
<dependency>
	<groupId>org.gcube.information-system</groupId>
	<artifactId>resource-registry-publisher</artifactId>
	<version>[1.0.0-SNAPSHOT, 2.0.0-SNAPSHOT)</version>
</dependency>
```

To use the client, you need first get a `ResourceRegistryPublisher` instance.
By using `ResourceRegistryPublisherFactory.create()` method the library discovers the correct endpoint to interact with the Resource Registry for the current context.

```
SecurityTokenProvider.instance.set("Your-Token-Here");
ResourceRegistryPublisher resourceRegistryPublisher =
	ResourceRegistryPublisherFactory.create();
```

5.3.1 Create Facet Instance Example

PUT /instances/CPUFacet/69f0b376-38d2-4a85-bc63-37f9fa323f82

Request Body

```
{
	"@class": "CPUFacet",
	"header": {
		"@class": "Header",
		"uuid": "69f0b376-38d2-4a85-bc63-37f9fa323f82"
```

```
            },
            "model": "Opteron",
            "vendor": "AMD",
            "clockSpeed": "1 GHz"
}
```

Response Body

```
{
            "@class": "CPUFacet",
            "header": {
                        "@class": "Header",
                        "uuid": "69f0b376-38d2-4a85-bc63-37f9fa323f82",
                        "creator": "luca.frosini",
                        "modifiedBy": "luca.frosini",
                        "creationTime": "2016-10-05 11:16:24",
                        "lastUpdateTime": "2016-10-05 11:16:24"
            },
            "model": "Opteron",
            "vendor": "AMD",
            "clockSpeed": "1 GHz"
}
```

Java API

```
CPUFacet cpuFacet = new CPUFacetImpl();
cpuFacet.setClockSpeed("1 GHz");
cpuFacet.setModel("Opteron");
cpuFacet.setVendor("AMD");

resourceRegistryPublisher.create(cpuFacet);
```

5.3.2 Update Facet Instance Example

PUT /instances/CPUFacet/69f0b376-38d2-4a85-bc63-37f9fa323f82

Request Body

```
{
            "@class": "CPUFacet",
            "header": { "uuid":"69f0b376-38d2-4a85-bc63-37f9fa323f82" },
            /* only the UUID is checked and must be the same of the UUID provided in the URL */
            "model": "Opteron",
            "vendor": "AMD",
            "clockSpeed":"2 GHz"
}
```

Response Body

```
{
            "@class": "CPUFacet",
            "header": {
                        "@class": "Header",

                        "uuid": "69f0b376-38d2-4a85-bc63-37f9fa323f82",
                        "creator": "luca.frosini",
                        "modifiedBy": "luca.frosini",
                        "creationTime": "2016-10-05 11:16:24",
                        "lastUpdateTime": "2016-10-05 11:18:33"
            },
            "model": "Opteron",
            "vendor": "AMD",
            "clockSpeed": "2 GHz"
}
```

Java API

```
createdCpuFacet.setClockSpeed("2 GHz");
resourceRegistryPublisher.update(createdCpuFacet);
```

5.3.3 Read Facet Instance Example

GET /instances/CPUFacet/69f0b376-38d2-4a85-bc63-37f9fa323f82

Response Body

```
{
        "@class": "CPUFacet",
        "header": {
                "@class": "Header",

                "uuid": "69f0b376-38d2-4a85-bc63-37f9fa323f82",
                "creator": "luca.frosini",
                "modifiedBy": "luca.frosini",
                "creationTime": "2016-10-05 11:16:24",
                "lastUpdateTime": "2016-10-05 11:18:33"
        },
        "model": "Opteron",
        "vendor": "AMD",
        "clockSpeed": "2 GHz"
}
```

Java API

```
resourceRegistryPublisher.read("69f0b376-38d2-4a85-bc63-37f9fa323f82");
```

5.3.4 Delete Facet Instance

DELETE /instances/CPUFacet/69f0b376-38d2-4a85-bc63-37f9fa323f82

Java API

```
boolean deleted = resourceRegistryPublisher.delete(createdCpuFacet);
```

5.3.5 Create Resource Instance

PUT /instances/HostingNode/670eeabf-76c7-493f-a449-4e6e139a2e84

Request Body

```
{
        "@class": "HostingNode",
        "header": {
                "uuid": "670eeabf-76c7-493f-a449-4e6e139a2e84",
                ...
        }"consistsOf": [
                {
```

```
				"@class": "ConsistsOf",
				"target": {
					"@class": "CPUFacet",
					"model": "Opteron",
					"vendor": "AMD",
					"clockSpeed": "2 GHz"
				}
			},
			{
				"@class": "IsIdentifiedBy",
				"target": {
					"@class": "NetworkingFacet",
					"ipAddress": "146.48.87.183",
					"hostName": "pc-frosini.isti.cnr.it",
					"domainName": "isti.cnr.it",
					"mask": "255.255.248.0",
					"broadcastAddress": "146.48.87.255"
				}
			}
		],
		"isRelatedTo": [
			{
				"@class": "Hosts",
				"propagationConstraint": {
					"add": "unpropagate",
					"remove": "cascade"
				},
				"target": {
					"@class": " EService",
					"header": {
						"uuid": "9bff49c8-c0a7-45de-827c-accb71defbd3"
					}
					/* The EService was already created, so the UUID is enough to attach it by
using Hosts relation */

				}
			}
		]
}
```

Response

```
{
		"@class": "HostingNode",
		"header": {
			"uuid": "670eeabf-76c7-493f-a449-4e6e139a2e84",
			...
		},
		"consistsOf": [
			{
				"@class": "ConsistsOf",
				"header": {
					"uuid": "9d0b1b2b-ac4e-40a9-8dea-bec90076e0ca",
					...
				},
				"target": {
					"@class": "CPUFacet",
					"header": {
						"uuid": "1daef6a8-5ca4-4700-844b-2a2d784e17b0",
						...
					},
					"model": "Opteron",
					"vendor": "AMD",
					"clockSpeed": "2 GHz"
				}
			},
			{
				"@class": "IsIdentifiedBy",
				"header": {
					"uuid": "02a7072c-4f72-4568-945b-9ddccc881e9f",
					...
				},
				"target": {
					"@class": "NetworkingFacet",
					"header": {
						"uuid": "59617b01-5856-4d8e-b85c-590a42039933",
						...
					},
					"ipAddress": "146.48.87.183",
					"hostName": "pc-frosini.isti.cnr.it",
					"domainName": "isti.cnr.it",
					"mask": "255.255.248.0",
					"broadcastAddress": "146.48.87.255"
				}
			}
		],
		"isRelatedTo": [
			{
				"@class": "Hosts",
				"header": {
					"uuid": "47494ad0-e606-4630-9def-4c607761ae14",
					...
				},
				"propagationConstraint": {
					"add": "unpropagate",
					"remove": "cascade"
				},
				"target": {
					"@class": "EService",
					"header": {
						"uuid": "9bff49c8-c0a7-45de-827c-accb71defbd3",
						...
					}
				}
			}
		]
}
```

Java API

```
NetworkingFacet networkingFacet = new NetworkingFacetImpl();
networkingFacet.setIPAddress("146.48.87.183");
networkingFacet.setHostName("pc-frosini.isti.cnr.it");
networkingFacet.setDomainName("isti.cnr.it");
networkingFacet.setMask("255.255.248.0");
networkingFacet.setBroadcastAddress("146.48.87.255");

networkingFacet = 	resourceRegistryPublisher.createFacet(networkingFacet);

HostingNode hostingNode = new HostingNodeImpl();
```

```
CPUFacet cpuFacet = new CPUFacetImpl();
cpuFacet.setClockSpeed("2 GHz");
cpuFacet.setModel("Opteron");
cpuFacet.setVendor("AMD");
hostingNode.addFacet(cpuFacet);

IsIdentifiedByImpl<Resource,Facet> isIdentifiedBy = new IsIdentifiedByImpl<Resource, Facet>(hostingNode,
networkingFacet, null);
hostingNode.addFacet(isIdentifiedBy);

PropagationConstraint propagationConstraint = new          PropagationConstraintImpl();
                        propagationConstraint.setRemoveConstraint(RemoveConstraint.cascade);
propagationConstraint.setAddConstraint(AddConstraint.unpropagate);

Activates<HostingNode, EService> hosts = new ActivatesImpl<HostingNode,           EService>(hostingNode,
eService, propagationConstraint);
hostingNode.attachResource(hosts);

hostingNode = resourceRegistryPublisher.createResource(hostingNode);
```

5.3.6 Update Resource Instance

PUT /instances/HostingNode/670eeabf-76c7-493f-a449-4e6e139a2e84

Request Body

```
{
        "@class": "HostingNode",
        "header": {
                "uuid": "670eeabf-76c7-493f-a449-4e6e139a2e84",
                ...
        },
        "consistsOf": [
                {
                        "@class": "ConsistsOf",
                        "header": {
                                "uuid": "9d0b1b2b-ac4e-40a9-8dea-bec90076e0ca",
                                ...
                        },
                        "target": {
                                "@class": "CPUFacet",
                                "header": {
                                        "uuid": "1daef6a8-5ca4-4700-844b-2a2d784e17b0",
                                        ...
                                },
                                "model": "Opteron",
                                "vendor": "AMD",
                                /* Updated the following property */
                                "clockSpeed": "1 GHz"
                        }
                },
                {
                        "@class": "IsIdentifiedBy",
                        "header": {
                                "uuid": "02a7072c-4f72-4568-945b-9ddccc881e9f",
                                ...
                        },
                        "target": {
                                "@class": "NetworkingFacet",
                                "header": {
                                        "uuid": "59617b01-5856-4d8e-b85c-590a42039933",
                                        ...
                                },
                                "ipAddress": "146.48.87.183",
                                "hostName": "pc-frosini.isti.cnr.it",
                                "domainName": "isti.cnr.it",
                                "mask": "255.255.248.0",
                                "broadcastAddress": "146.48.87.255",
                                /* Added the following property */
                                "username": "luca.frosini"
                        }
                }
        ]
}
```

Java API

```
networkingFacet = (NetworkingFacet) hostingNode.getIdentificationFacets().get(0);
networkingFacet.setAdditionalProperty("username", "luca.frosini");

cpuFacet = hostingNode.getFacets(CPUFacet.class).get(0);
cpuFacet.setClockSpeed("1 GHz");

hostingNode = resourceRegistryPublisher.updateResource(hostingNode);
```

5.3.7 Delete Resource Instance

```
PUT /instances/HostingNode/670eeabf-76c7-493f-a449-4e6e139a2e84
```

Java API

```
boolean deleted = resourceRegistryPublisher.deleteResource(hostingNode);
```

Similarly to the examples provided for Facets and Resources is possible to operate on consistsOf and isRelatedTo relations. Here we just provide an example of consistsOf creation.

5.3.8 Create ConsistsOf Instance

```
PUT /instances/IsIdentifiedBy/02a7072c-4f72-4568-945b-9ddccc881e9f
```

In this example the target Facet already exists. The Service set automatically the propagation constraint to default values (i.e. remove=cascadeWhenOrphan, add=propagate)

Request Body

```
{
        "@class": "IsIdentifiedBy",
        "header": {
                "uuid": "02a7072c-4f72-4568-945b-9ddccc881e9f",
                ...
        },
        "source": {
                "@class": "HostingNode",
                // The HostingNode must be already created. The header with UUID is enough.
                "header": {
                    "uuid": "670eeabf-76c7-493f-a449-4e6e139a2e84"
                }
        },
        "target": {
                "@class": "NetworkingFacet",
                /* The NetworkingFacet already exists, so the UUID is enough to attach it by using
IsIdentifiedBy relation */
                "header": {
                        "uuid": "59617b01-5856-4d8e-b85c-590a42039933"
                },
        }
}
```

Response

```
{
        "@class": "IsIdentifiedBy",
        "header": {
                "uuid": "02a7072c-4f72-4568-945b-9ddccc881e9f",
                ...
        },
        "propagationConstraint": {
                "add": "propagate",
                "remove": "cascadeWhenOrphan"
        },
        "source": {
                "@class": "HostingNode",
                "header": {
                        "uuid": "670eeabf-76c7-493f-a449-4e6e139a2e84"
                }
        },
        "target": {
                "@class": "NetworkingFacet",
                "header": {
                        "uuid": "59617b01-5856-4d8e-b85c-590a42039933",
                        ...
                },
                "ipAddress": "146.48.87.183",
                "hostName": "pc-frosini.isti.cnr.it",
                "domainName": "isti.cnr.it",
                "mask": "255.255.248.0",
                "broadcastAddress": "146.48.87.255"
        }
}
```

5.4 Query and Access

Query and Access port type allows the performing of queries on instances in a specific context. It exposes some of the safe methods already available through dedicated port types in addition to query APIs.

Table 13: Query and Access Management Operations (grouped by inherited port type), Methods and URLs.

Group	Operation	HTTP Method	URL
Contexts	Listing	GET	/access/contexts

Group	Operation	HTTP Method	URL
	Existence	HEAD	/access/contexts/{CONTEXT_UUID}
	Read	GET	/access/contexts/{CONTEXT_UUID}
Types	Listing	GET	/access/types
	Existence	HEAD	/access/types/{TYPE_NAME}
	Read	GET	/access/types/{TYPE_NAME} [?polymorphic=false]
Instances	Existence	HEAD	/access/instances/{TYPE_NAME}/{UUID}
	Read	GET	/access/instances/{TYPE_NAME}/{UUID}
Query	Query all instances of a type	GET	/access/query/{TYPE_NAME} [?polymorphic=true]
	Get filtered entities	GET	/access/query/{ENTITY_TYPE_NAME} /{RELATION_TYPE_NAME} /{REFERENCE_ENTITY_TYPE_NAME} [?polymorphic=true&direction=(in|out|both) [&reference={REFERENCE_ENTITY_UUID} &name1=value1&name2=value2&...]
Raw Query	Gremlin Query to Graph	GET	/query?q={query}

Table 13 shows the exposed APIs grouped by base URL. The APIs, which get context Information:

- list all existent contexts;
- check if a context with a certain `CONTEXT_UUID` exists;
- read the representation of the context identified by the `CONTEXT_UUID`.

The APIs to retrieve types information:

- list all the registered types;
- check if a certain type exists;
- read the schema for the specified `TYPE_NAME`.

Using the parameter polymorphic=true, apart from the schema for the specified `TYPE_NAME`, it returns also the schema of all the sub-types.

The APIs to get instances information:

- check if a certain instance exists;
- read the representation of a certain instance.

The above mentioned APIs has been already presented in the dedicated sections, hence they will not be presented again. This section, instead, provides information regarding how to interact with Resource Registry Service for Query and Raw Query parts. Apart from the REST API, this port type can be used also by using Resource Registry Client Java library. Both REST and Java APIs are presented. The provided examples can intentionally hide some details to avoid unneeded complexity.

Resource Registry Client has the following Maven coordinates:

```
<dependency>
	<groupId>org.gcube.information-system</groupId>
	<artifactId>resource-registry-client</artifactId>
	<version>[1.0.0-SNAPSHOT, 2.0.0-SNAPSHOT)</version>
</dependency>
```

To use the client, you need first get a `ResourceRegistryClient` instance.
By using `ResourceRegistryClientFactory.create()` method the library discovers the correct endpoint to interact with the Resource Registry for the current context.

```
SecurityTokenProvider.instance.set("Your-Token-Here");
ResourceRegistryClient                    resourceRegistryClient                    =
	ResourceRegistryClientFactory.create();
```

5.4.1 Get All Instances of a Type

GET /access/query/{TYPE_NAME}[?polymorphic=true]

This APIs returns the list of all instances of a certain `TYPE_NAME` provided as a path parameter. This API allows a client to indicate if it is interested in getting all the instances of the specified type (all the instances of sub-types) or it requires only the instances of the indicated type using `polymorphic=false` (default `true`).

5.4.1.1 Get All Instances of EService

GET /access/query/EService?polymorphic=false

Response

```
[
	{
		"@class": "EService",
		"header": {
			"uuid": "0717b450-a698-11e2-900a-a46c6ff57f05",
			…
		},
		"consistsOf": [
			…
		],
		"isRelatedTo": [
			…
		],
	},
	…,
	{
		"@class": "EService",
		"header": {
			"uuid": "3b6061f9-e2ab-4c01-b3b2-48b470a5b8a",
			…
		},
		"consistsOf": [
			…
		],
                "isRelatedTo": [
                        ...
                ],
        }
]
```

5.4.1.2 Get All Instances of EService and subtypes

GET /access/query/EService?polymorphic=true

Response

```
[
        {
                "@class": "RunningPlugin",
                "header": {
                        "uuid": "66d69dab-203e-45ff-b49e-a8fa4126a392",
                        ...
                },
                ,
                "consistsOf": [
                        ...
                ],
                "isRelatedTo": [
                        ...
                ],
        },
        ...,
        {
                "@class": "EService",
                "header": {
                        "uuid": "0717b450-a698-11e2-900a-a46c6ff57f05",
                        ...
                },
                ,
                "consistsOf": [
                        ...
                ],
                "isRelatedTo": [
                        ...
                ],
        }
]
```

5.4.2 Get Filtered Entities

```
GET     /access/query/{ENTITY_TYPE_NAME}
        /{RELATION_TYPE_NAME}/{REFERENCE_ENTITY_TYPE_NAME}
        [?polymorphic=true&direction=(in|out|both)
        [&reference={REFERENCE_ENTITY_UUID}
        &name1=value1&name2=value2&...]
```

This API returns the list of instances of a specific entity type (indicated by the first path parameter i.e., `ENTITY_TYPE_NAME`) and filters them according to the following criteria:

- The second path parameter indicates which relation type (i.e., `RELATION_TYPE_NAME`) must be related to the obtained instances;

- The direction of the relation can be specified by using direction query variable which by default is both. The reference of the direction is the `ENTITY_TYPE_NAME`. Allowed values for `direction` are (`in|out|both`);
- The third path parameter (i.e., `REFERENCE_ENTITY_TYPE_NAME`) defines the types of the entity in the opposite side of the relation;
- The `reference` query parameter enforces a specific instance of referenced type by specifying the `UUID` (i.e., `REFERENCE_ENTITY_UUID`);
- When the client does not indicate `reference` query parameter, it is possible to filter between the reference entities by specifying an arbitrary number of name-value couples. The API evaluates in **AND** the set of couple specified as query parameters (i.e., `&name1=value1&name2=value2`);
- The polymorphic query parameter (`default=true`) indicates if the client is requesting only instances of the indicated type (`polymorphic=false`) or also the instances of any extension of the indicated type (`polymorphic=true`).

The service will return `400 Bad Request` in case the indicated types are not compliant with the model (e.g., specifying an isRelatedTo relation between a facet type and a resource type).

This API is very versatile. Examples of use are:

- `/access/query/Eservice/IsIdentifiedBy/SoftwareFacet?polymorphic=false&direction=out`
  this invocation allows to retrieve all the EService instances having a `SoftwareFacet` related with `IsIdentifiedBy`. The only allowed direction is out because `IsIdentifiedBy` is a specialisation of `ConsistsOf` which by definition 'exists' (`out`) from a resource and 'enters' (`in`) into a facet;
- /access/query/EService/IsIdentifiedBy/SoftwareFacet?polymorphic=true&direction=out&reference=7bc997c3-d005-40ff-b9ed-c4b6a35851f1

This invocation allows retrieval of the `EService` instance identified by a `SoftwareFacet` with `UUID 7bc997c3-d005-40ff-b9ed-c4b6a35851f1`. The URL has `polymorphic=true` query parameter (it could be omitted because it is the default) this means that the result could be for example a `RunningPlugin` (which is a specialisation of EService) identified by the `SoftwareFacet` with such `UUID`. The `direction=out` query parameter is the only valid value for the request to avoid to get a `400 Bad Request` response;

- `/access/query/Resource/IsIdentifiedBy/ContactFacet?polymorphic=true&direction=out`
  This invocation allows retrieval of all the Resource instances (any type of Resource instance giving the query parameter polymorphic=true) which is identified by a `ContactFacet`;
- /access/query/Resource/ConsistsOf/ContactFacet?polymorphic=true&direction=out
  This invocation allows retrieval of all the Resource instances (any type of Resource instance giving the query parameter `polymorphic=true`) having a ContactFacet related by any type of `ConsistsOf` relation (giving the query parameter `polymorphic=true`);

- `/access/query/Service/Hosts/Site?polymorphic=true&direction=in`
  This invocation allows retrieval of all the Service instances having an incoming Hosts relation (a specialisation of `IsRelatedTo`) from `Site` resource instances;
- /access/query/Service/Hosts/Site?polymorphic=true&direction=in&reference=16032d09-3823-444e-a1ff-a67de4f350a8
  This invocation allows retrieval of all the Service instances hosted by (having an incoming Hosts relation from) the Site with UUID 16032d09-3823-444e-a1ff-a67de4f350a;
- /access/query/Eservice/ConsistsOf/SoftwareFacet?polymorphic=true&direction=out&group=accounting&name=accounting-service
  This invocation allows to retrieve the `EService` instances identified by a `SoftwareFacet` with `group=accounting` and `name=accounting-service` (i.e., the running Accounting Service instances).

5.4.3 Raw Query

GET /query?q={query}

This API provides a way to query the underlying database persistence by using the persistence query language dialect. This API does not provide any consistency with the Information System Model concepts. The result is related to how the service decides to represent the Information System Model concepts on persistence data model. At the time of writing, the underlying database persistence is OrientDB. It should be used only for development purposes because the way to represent the Information System Model concepts can change at any time or can change the database persistence. At time of writing the query language supported is OrientDB SQL Dialect[25]

This API is accessible only to infrastructure managers and administrators. The base path of this API differs from the others to facilitate the definition of networking and authorisation policies to restrict the access.

GET /query?q=SELECT FROM SoftwareFacet LIMIT 2

Response Body

```
{
	"result": [
		{
			"@type": "d",
			"@rid": "#99:5",
			"@version": 12,
			"@class": "SoftwareFacet",
			"header": {
				"@type": "d",
				"@version": 0,
				"@class": "Header",
				"uuid": "6b724a7c-9f51-4a4e-8e8e-1636ca2e9d29",
				"creator": "VREManagement:WhnManager:pc-frosini.isti.cnr.it_8080",
				"creationTime": "2017-10-05 16:09:02.618 +0200",
				"lastUpdateTime": "2017-10-05 17:23:44.191 +0200",
				"@fieldTypes": "creationTime=t,lastUpdateTime=t"
			},
```

[25] http://orientdb.com/docs/last/SQL.html

```
            "name": "WhnManager",
            "description": "Web Hosting Node Service",
            "optional": false,
            "version": "2.0.0-SNAPSHOT",
            "group": "VREManagement",
            "in_IsIdentifiedBy": [
                "#168:5"
            ]
        },
        {
            "@type": "d",
            "@rid": "#99:6",
            "@version": 5,
            "@class": "SoftwareFacet",
            "header": {
                "@type": "d",
                "@version": 0,
                "@class": "Header",
                "uuid": "bc98eec4-4365-49fd-83b3-2cacaf17f8bf",
                "creator": "VREManagement:SmartExecutor:pc-frosini.isti.cnr.it_8080",
                "creationTime": "2017-10-05 17:22:06.351 +0200",
                "lastUpdateTime": "2017-10-05 17:23:44.206 +0200",
                "@fieldTypes": "creationTime=t,lastUpdateTime=t"
            },
            "name": "SmartExecutor",
            "description": "Smart Executor Service",
            "optional": false,
            "version": "1.7.0-SNAPSHOT",
            "group": "VREManagement",
            "in_IsIdentifiedBy": [
                "#168:6"
            ]
        }
    ],
    "notification": "Query executed in 0.147 sec. Returned 2 record(s)"
}
```

6 Backend Database (i.e. OrientDB as Graph Database)

OrientDB is a Multi-Model Open Source NoSQL DBMS that brings together the power of graphs and the flexibility of documents into one scalable high-performance operational database[26]. OrientDB engine supports **Graph**, **Document**, **Key/Value**, and **Object** models. A graph represents a network-like structure consisting of Vertices (also known as Nodes) interconnected by Edges (also known as Arcs).

OrientDB's graph model is represented by the concept of a property graph, which defines the following:

- **Vertex** - an entity that can be linked with other Vertices and has the following mandatory properties:
    - unique identifier,
    - set of incoming Edges,
    - set of outgoing Edges.
- **Edge** - an entity that links two Vertices and has the following mandatory properties:
    - unique identifier,
    - link to an incoming Vertex (also known as head),
    - link to an outgoing Vertex (also known as tail).

In addition to mandatory properties, each vertex or edge can also hold a set of custom properties. These properties can be defined by users, which can make vertices and edges appear similar to documents.[27] Given that, we can say that OrientDB, used as graph database, is de-facto a graph-document database. This peculiarity provides an excellent support for the Information System model which has been mapped on OrientDB concepts as following:

- Entities are modelled as Vertexes;
- Relations are modelled as Edges.

In both cases, the OrientDB internal ID has been hidden and, instead, the header property (embedded) is created which provides, among others, the ID to uniquely identify the Entity or Relation.

Another important characteristic is the native support of embedded properties. Embedded properties are structured properties inside a vertex or an edge. The Header is the only properties of resources.

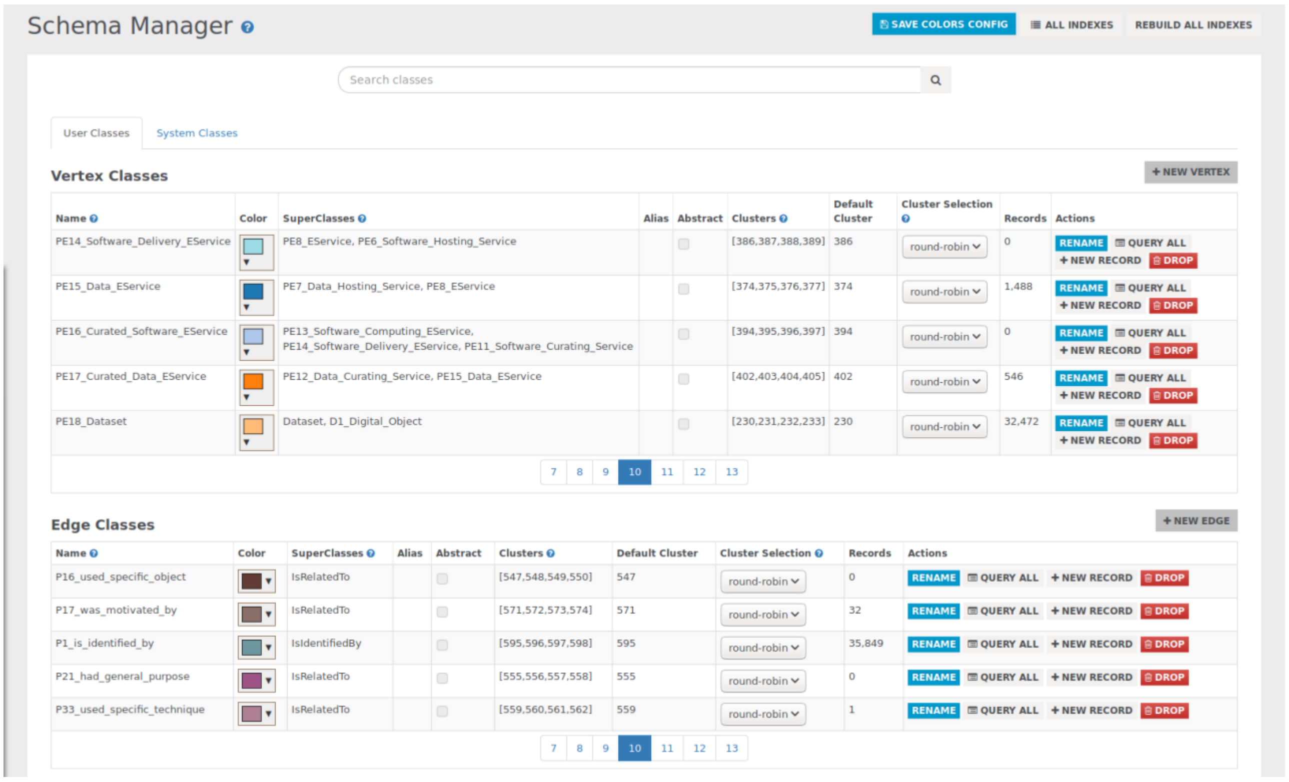
OrientDB provides a simple referential integrity support guaranteeing that if a vertex is deleted then every attached edge (incoming or outgoing) is also deleted. The Resource Registry provides additional referential integrity support by using directives contained in each PropagationConstraint property attached to edges. It is responsibility of the Resource Registry to provide support for this.

[26] http://orientdb.com/docs/2.2.x/
[27] http://orientdb.com/docs/2.2.x/Tutorial-Introduction-to-the-NoSQL-world.html

7 The Studio GUI

The Content administrator is allowed to use the Web Graphical User Interface (GUI) provided with OrientDB called Studio.

Figure 1 shows the interface allowing browsing and searching of the content of the Joint Resource Registry. It also allows the inspection of the schema of the resources defined in the PARTHENOS Entity Model. At the top of the page the search bar is presented. The browsing is paginated and the types are divided into vertex and edge types.



Vertex Classes

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
PE14_Software_Delivery_EService	🟦	PE8_EService, PE6_Software_Hosting_Service		☐	[386,387,388,389]	386	round-robin	0	RENAME QUERY ALL + NEW RECORD DROP
PE15_Data_EService	🟦	PE7_Data_Hosting_Service, PE8_EService		☐	[374,375,376,377]	374	round-robin	1,488	RENAME QUERY ALL + NEW RECORD DROP
PE16_Curated_Software_EService	🟦	PE13_Software_Computing_EService, PE14_Software_Delivery_EService, PE11_Software_Curating_Service		☐	[394,395,396,397]	394	round-robin	0	RENAME QUERY ALL + NEW RECORD DROP
PE17_Curated_Data_EService	🟧	PE12_Data_Curating_Service, PE15_Data_EService		☐	[402,403,404,405]	402	round-robin	546	RENAME QUERY ALL + NEW RECORD DROP
PE18_Dataset	🟧	Dataset, D1_Digital_Object		☐	[230,231,232,233]	230	round-robin	32,472	RENAME QUERY ALL + NEW RECORD DROP

7 8 9 10 11 12 13

Edge Classes

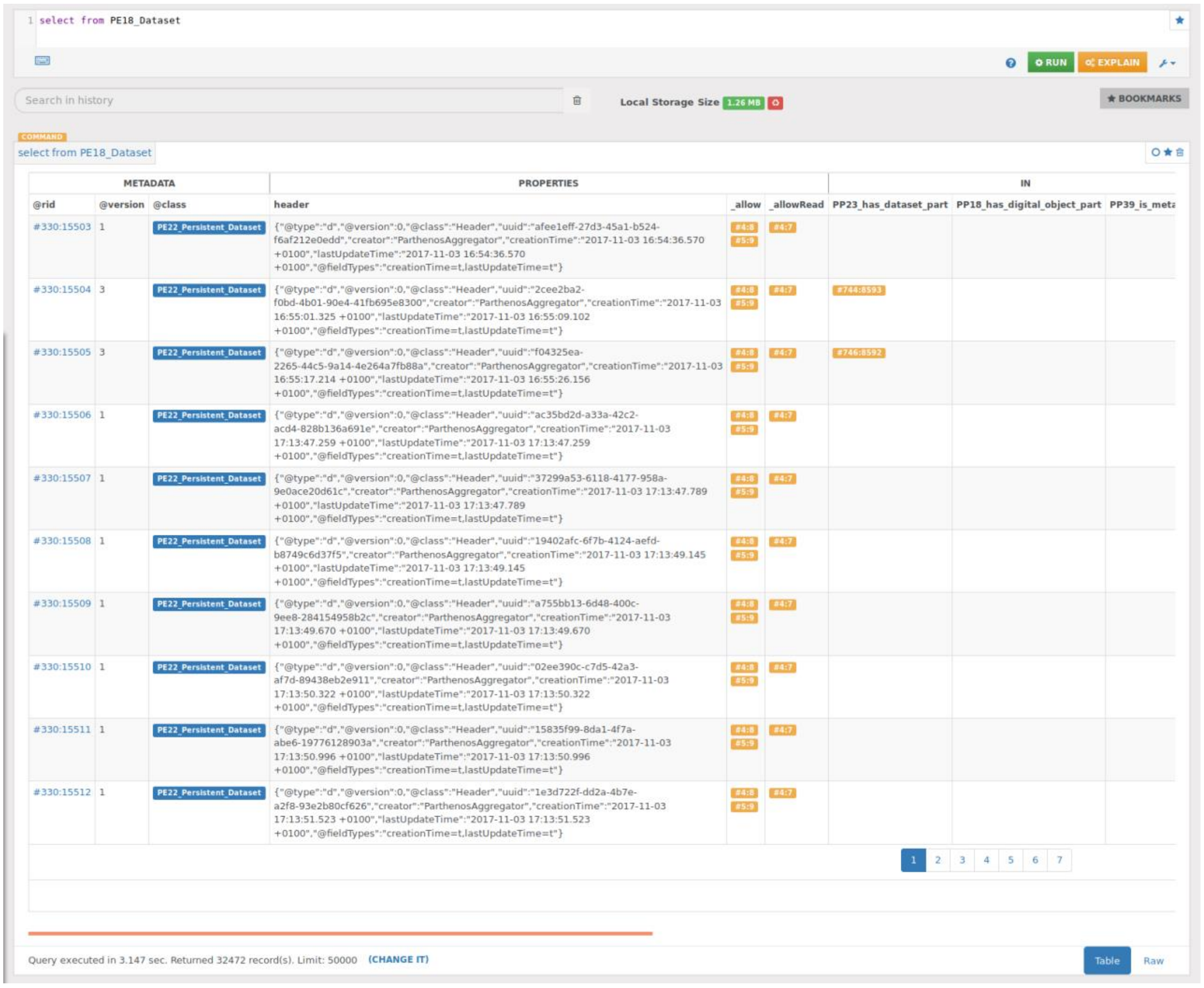
Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
P16_used_specific_object	🟫	IsRelatedTo		☐	[547,548,549,550]	547	round-robin	0	RENAME QUERY ALL + NEW RECORD DROP
P17_was_motivated_by	🟫	IsRelatedTo		☐	[571,572,573,574]	571	round-robin	32	RENAME QUERY ALL + NEW RECORD DROP
P1_is_identified_by	🟦	IsIdentifiedBy		☐	[595,596,597,598]	595	round-robin	35,849	RENAME QUERY ALL + NEW RECORD DROP
P21_had_general_purpose	🟪	IsRelatedTo		☐	[555,556,557,558]	555	round-robin	0	RENAME QUERY ALL + NEW RECORD DROP
P33_used_specific_technique	🟪	IsRelatedTo		☐	[559,560,561,562]	559	round-robin	1	RENAME QUERY ALL + NEW RECORD DROP

7 8 9 10 11 12 13

Figure 1: Schema Manager

Moreover, two different interfaces to get the results of a query are provided. The first one, see Figure 2, provides textual results, while the second one, see Figures 3 and 4, provides a graphical representation of the graph results of the query.

The interface providing textual results also allows editing of any of the presented instances by clicking on the resulting row.



```
1 select from PE18_Dataset
```

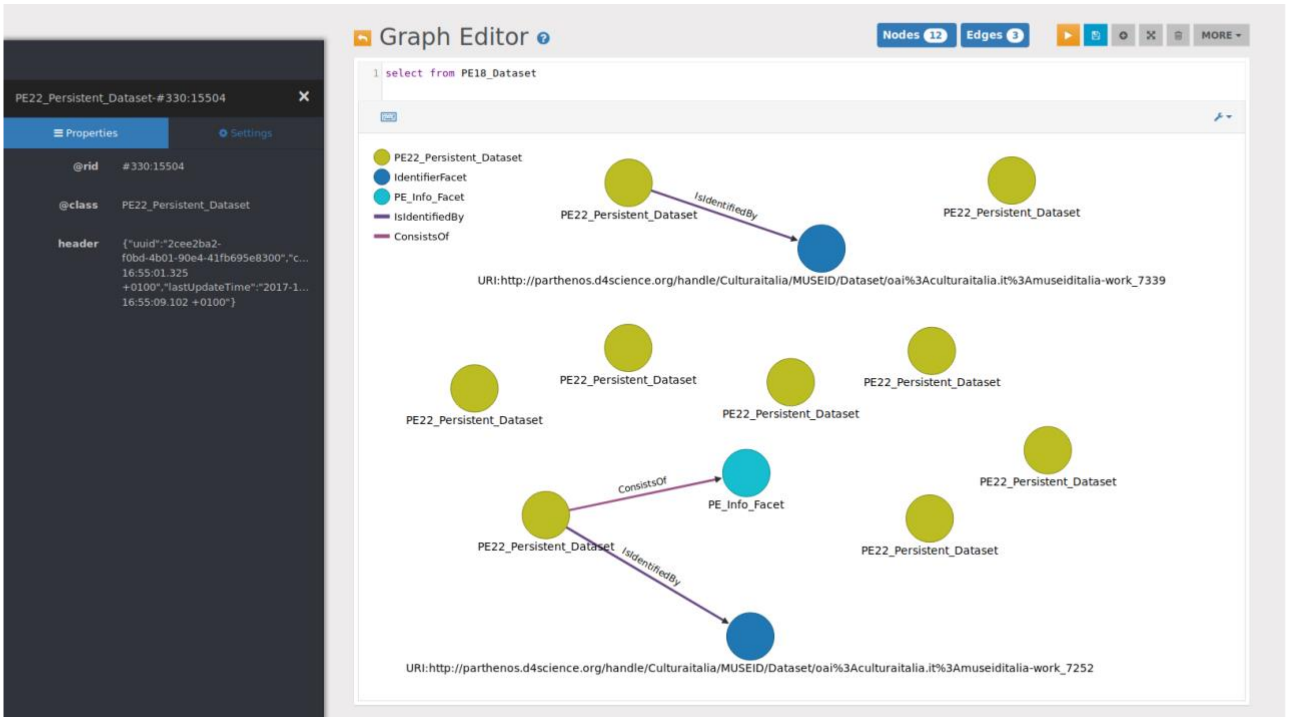
select from PE18_Dataset

METADATA			PROPERTIES			IN		
@rid	@version	@class	header	_allow	_allowRead	PP23_has_dataset_part	PP18_has_digital_object_part	PP39_is_meta
#330:15503	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"afee1eff-27d3-45a1-b524-f6af212e0edd","creator":"ParthenosAggregator","creationTime":"2017-11-03 16:54:36.570 +0100","lastUpdateTime":"2017-11-03 16:54:36.570 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15504	3	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"2cee2ba2-f0bd-4b01-90e4-41fb695e8300","creator":"ParthenosAggregator","creationTime":"2017-11-03 16:55:01.325 +0100","lastUpdateTime":"2017-11-03 16:55:09.102 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7	#744:8593		
#330:15505	3	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"f04325ea-2265-44c5-9a14-4e264a7fb88a","creator":"ParthenosAggregator","creationTime":"2017-11-03 16:55:17.214 +0100","lastUpdateTime":"2017-11-03 16:55:26.156 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7	#746:8592		
#330:15506	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"ac35bd2d-a33a-42c2-acd4-828b136a691e","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:47.259 +0100","lastUpdateTime":"2017-11-03 17:13:47.259 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15507	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"37299a53-6118-4177-958a-9e0ace20d61c","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:47.789 +0100","lastUpdateTime":"2017-11-03 17:13:47.789 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15508	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"19402afc-6f7b-4124-aefd-b8749c6d37f5","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:49.145 +0100","lastUpdateTime":"2017-11-03 17:13:49.145 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15509	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"a755bb13-6d48-400c-9ee8-284154958b2c","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:49.670 +0100","lastUpdateTime":"2017-11-03 17:13:49.670 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15510	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"02ee390c-c7d5-42a3-af7d-89438eb2e911","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:50.322 +0100","lastUpdateTime":"2017-11-03 17:13:50.322 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15511	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"15835f99-8da1-4f7a-abe6-19776128903a","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:50.996 +0100","lastUpdateTime":"2017-11-03 17:13:50.996 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			
#330:15512	1	PE22_Persistent_Dataset	{"@type":"d","@version":0,"@class":"Header","uuid":"1e3d722f-dd2a-4b7e-a2f8-93e2b80cf626","creator":"ParthenosAggregator","creationTime":"2017-11-03 17:13:51.523 +0100","lastUpdateTime":"2017-11-03 17:13:51.523 +0100","@fieldTypes":"creationTime=t,lastUpdateTime=t"}	#4:8 #5:9	#4:7			

Query executed in 3.147 sec. Returned 32472 record(s). Limit: 50000 (CHANGE IT)

Figure 2: Textual Query Inspector

The interface providing the representation of the graph instead allows inspection of the content of the vertexes and edges by clicking on any one of them. The information is provided in the side panel on the left. The side panel has two tabs: the first shows the properties of the selected element; the second tab is used to change the presentation information of the element such as the colour of the circle for vertexes, and the attached label for edges and vertexes. The label can be either one of the attributes of the element or OrientDB internal information such as the internal id.



Graph Editor
Nodes 12
Edges 3
MORE
PE22_Persistent_Dataset
IdentifierFacet
PE_Info_Facet
IsIdentifiedBy
ConsistsOf
IsIdentifiedBy
ConsistsOf
PE_Info_Facet
URI:http://parthenos.d4science.org/handle/Culturaitalia/MUSEID/Dataset/oai%3Aculturaitalia.it%3Amuseiditalia-work_7339
URI:http://parthenos.d4science.org/handle/Culturaitalia/MUSEID/Dataset/oai%3Aculturaitalia.it%3Amuseiditalia-work_7252

```
select from PE18_Dataset
```

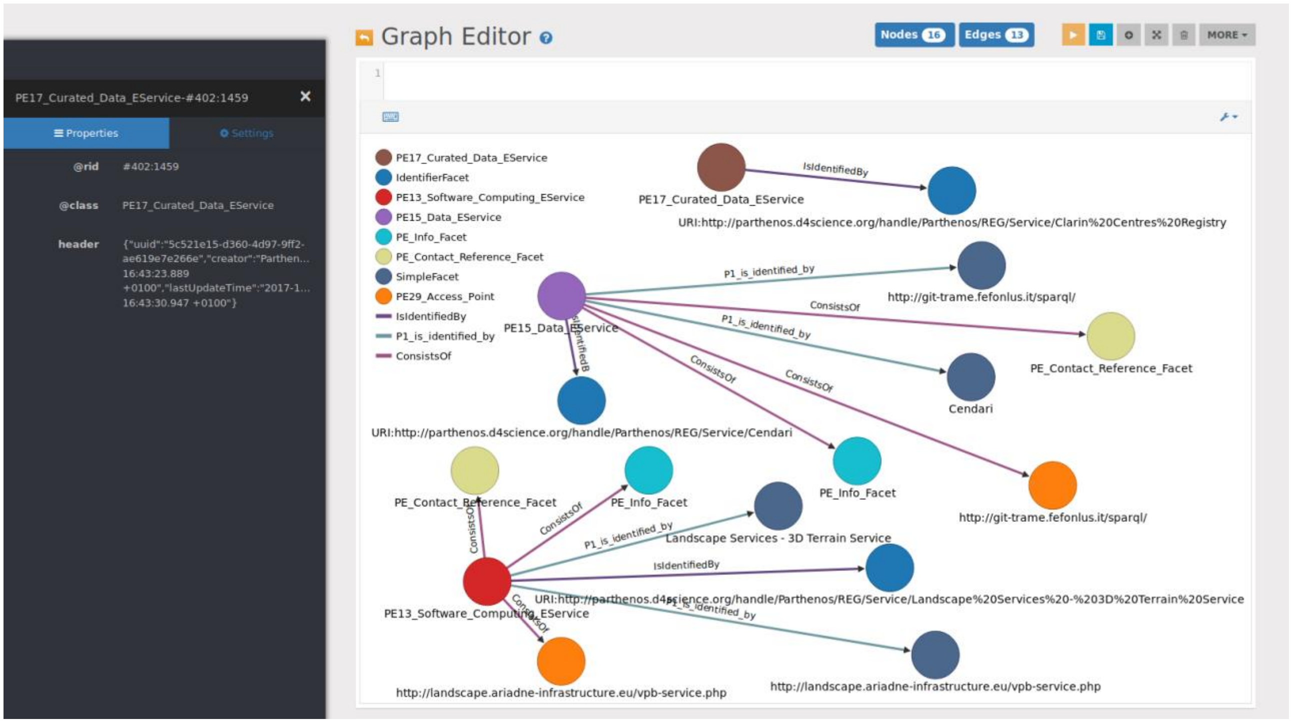
PE22_Persistent_Dataset-#330:15504

Properties | Settings

@rid	#330:15504
@class	PE22_Persistent_Dataset
header	{"uuid":"2cee2ba2-f0bd-4b01-90e4-41fb695e8300","c... 16:55:01.325 +0100","lastUpdateTime":"2017-1... 16:55:09.102 +0100"}

Figure 3: Graph Query Inspector

The Graph Query Inspector interface also allows iterative inspection by navigating the relations (edges) created between the entities (vertexes). By clicking on the element an overlay menu is presented. The menu directs the user to the valid options available for the navigation.



Graph Editor
Nodes 16
Edges 13
MORE
PE17_Curated_Data_EService
IdentifierFacet
PE13_Software_Computing_EService
PE15_Data_EService
PE_Info_Facet
PE_Contact_Reference_Facet
SimpleFacet
PE29_Access_Point
IsIdentifiedBy
P1_is_identified_by
ConsistsOf
URI:http://parthenos.d4science.org/handle/Parthenos/REG/Service/Clarin%20Centres%20Registry
http://git-trame.fefonlus.it/sparql/
Cendari
URI:http://parthenos.d4science.org/handle/Parthenos/REG/Service/Cendari
Landscape Services - 3D Terrain Service
URI:http://parthenos.d4science.org/handle/Parthenos/REG/Service/Landscape%20Services%20-%203D%20Terrain%20Service
http://landscape.ariadne-infrastructure.eu/vpb-service.php

PE17_Curated_Data_EService-#402:1459

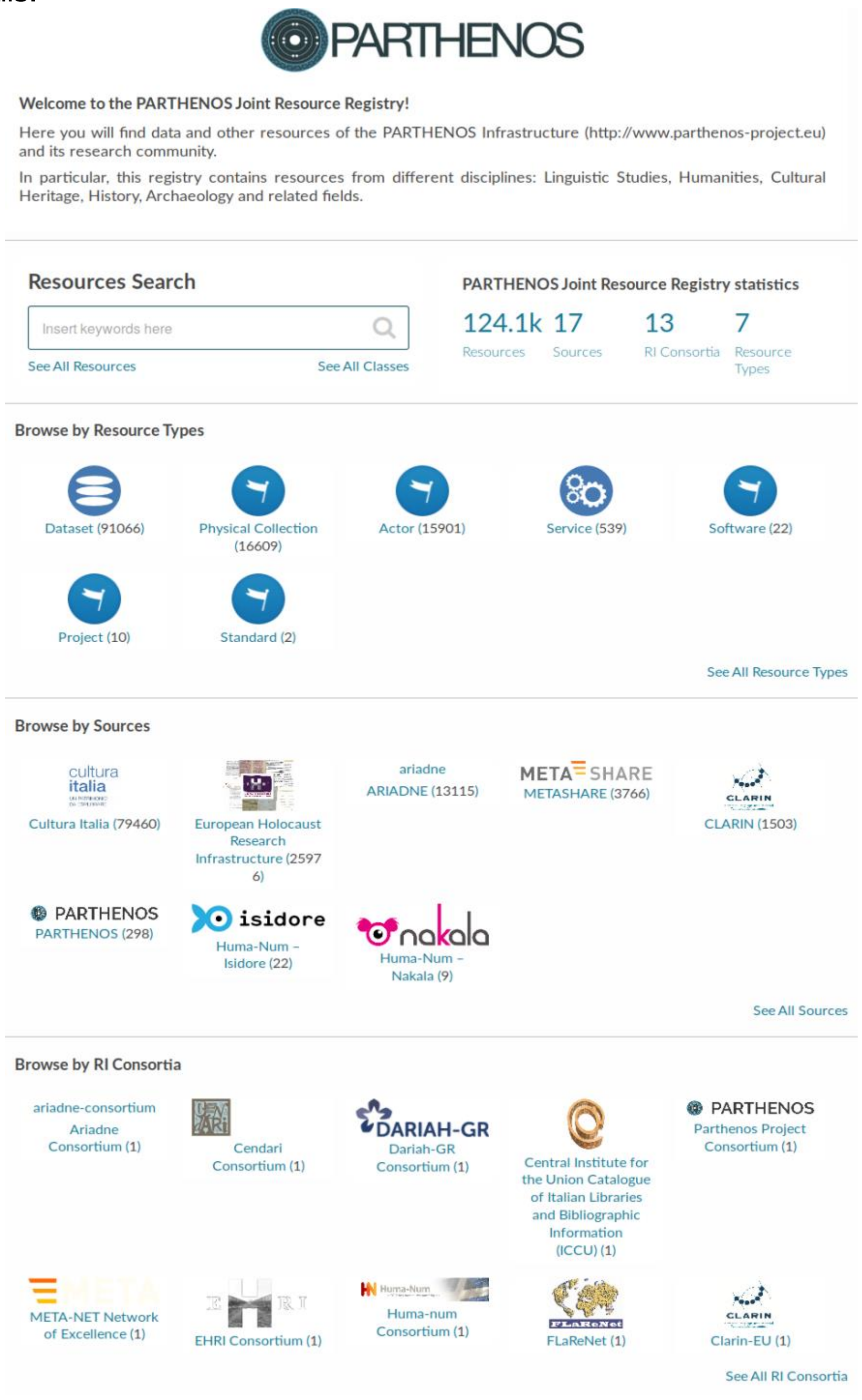
Properties | Settings

@rid	#402:1459
@class	PE17_Curated_Data_EService
header	{"uuid":"5c521e15-d360-4d97-9ff2-ae619e7e266e","creator":"Parthen... 16:43:23.889 +0100","lastUpdateTime":"2017-1... 16:43:30.947 +0100"}

Figure 4: Graph Editor

The Content administrator graphical user interface will be complemented by an additional interface designed for end-users. This additional tool will present the content of the Joint Resource Registry as a catalogue of resources. The catalogue will be searchable and browsable while faceted search will allow interactive inspections of the PARTHENOS entities.

The end-user graphical user interface is currently under testing and validation and its description will be added to the D6.5 Report on the Implementation of the Joint Resource Registry (final) deliverable due at month 48. A preliminary screenshot of this interface is shown in Figures 5 and 6. Figure 5 shows the welcome page allowing browsing between the types and the research infrastructures (i.e. groups), Figure 6 shows an example of a resource details.



PARTHENOS

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.

Resources Search

Insert keywords here

See All Resources — See All Classes

PARTHENOS Joint Resource Registry statistics

124.1k	17	13	7
Resources	Sources	RI Consortia	Resource Types

Browse by Resource Types

Dataset (91066) · Physical Collection (16609) · Actor (15901) · Service (539) · Software (22) · Project (10) · Standard (2)

See All Resource Types

Browse by Sources

Cultura Italia (79460) · European Holocaust Research Infrastructure (25976) · ARIADNE (13115) · METASHARE (3766) · CLARIN (1503) · PARTHENOS (298) · Huma-Num – Isidore (22) · Huma-Num – Nakala (9)

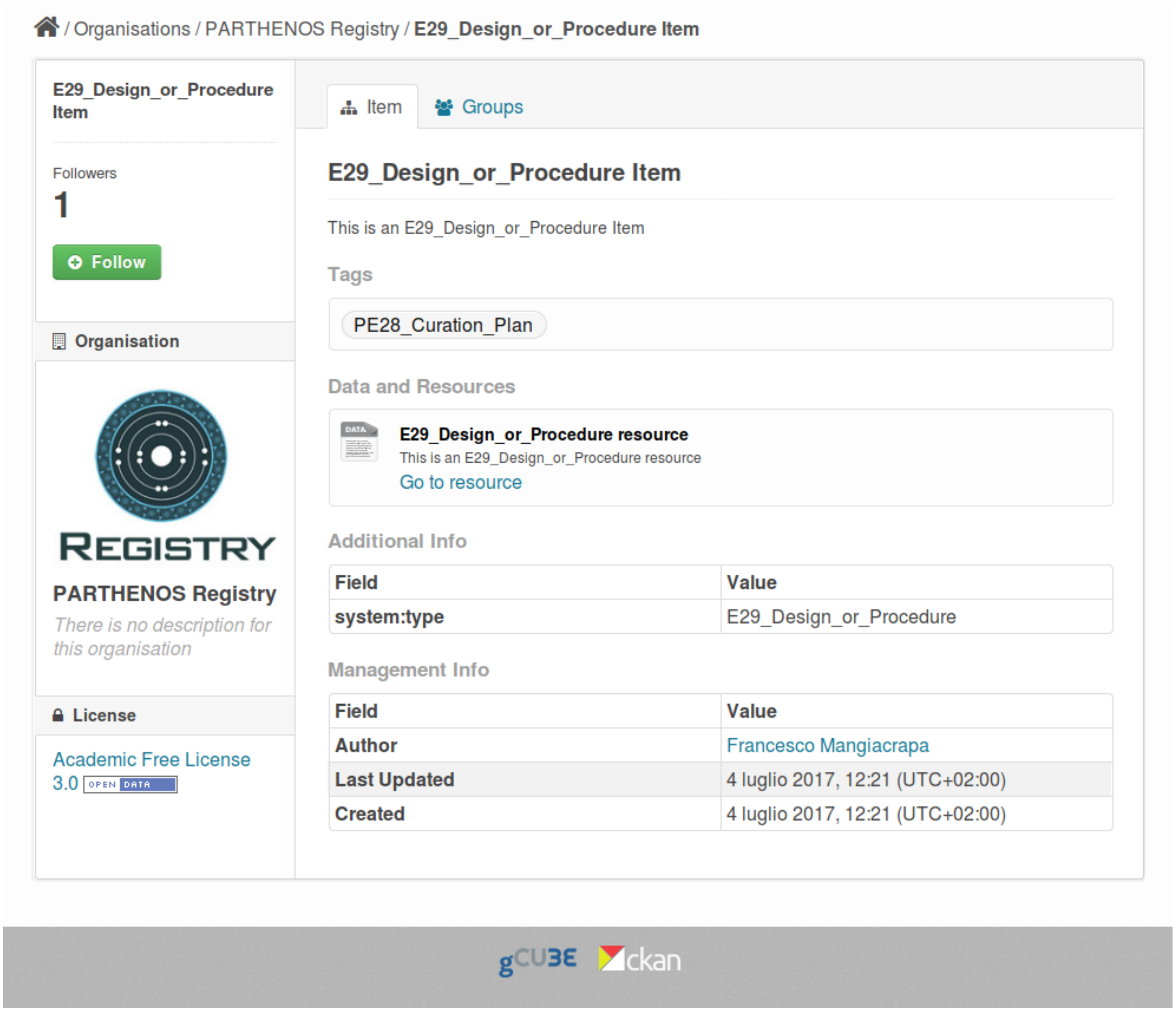
See All Sources

Browse by RI Consortia

ariadne-consortium Ariadne Consortium (1) · Cendari Consortium (1) · Dariah-GR Consortium (1) · Central Institute for the Union Catalogue of Italian Libraries and Bibliographic Information (ICCU) (1) · Parthenos Project Consortium (1) · META-NET Network of Excellence (1) · EHRI Consortium (1) · Huma-num Consortium (1) · FLaReNet (1) · Clarin-EU (1)

See All RI Consortia

Figure 5: End-user Graphical User Interface welcome page



/ Organisations / PARTHENOS Registry / **E29_Design_or_Procedure Item**

E29_Design_or_Procedure Item

Followers
1

Follow

Organisation

PARTHENOS Registry

There is no description for this organisation

License

Academic Free License 3.0

Item | Groups

E29_Design_or_Procedure Item

This is an E29_Design_or_Procedure Item

Tags

PE28_Curation_Plan

Data and Resources

E29_Design_or_Procedure resource
This is an E29_Design_or_Procedure resource
Go to resource

Additional Info

Field	Value
system:type	E29_Design_or_Procedure

Management Info

Field	Value
Author	Francesco Mangiacrapa
Last Updated	4 luglio 2017, 12:21 (UTC+02:00)
Created	4 luglio 2017, 12:21 (UTC+02:00)

Figure 6: End-user Graphical User Interface resource details