VRE4EIC

A Europe-wide Interoperable Virtual Research Environment to Empower Multidisciplinary Research Communities and Accelerate Innovation and Collaboration

Deliverable D3.5

Final Architecture Design

Document version: 0.1

VRE4EIC DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:
Mr Peter Kunz t: +33 4 92 38 50 10 f: +33 4 92 38 78 22 e: peter.kunz@ercim.eu
GEIE ERCIM, 2004, route des Lucioles, Sophia Antipolis, F-06410 Biot, France
Project website address: http://www.vre4eic.eu/

Project	
Grant Agreement number	676247
Project acronym:	VRE4EIC
Project title:	A Europe-wide Interoperable Virtual Research Environment to Empower Multidisciplinary Research Communities and Accelerate Innovation and Collaboration
Funding Scheme:	Research & Innovation Action (RIA)
Date of latest version of DoW against which the assessment will be made:	31 May 2017 Amended Grant Agreement through amendment n°AMD-676247-8
Document	
Period covered:	M1-36
Deliverable number:	D3.5
Deliverable title	Final Architecture design
Contractual Date of Delivery:	30/09/2018
Actual Date of Delivery:	29/09/2018
Editor (s):	Peter Kunz (ERCIM)
Author (s):	Carlo Meghini (CNR ISTI)
Reviewer (s):	Keith Jeffery (ERCIM) Cesare Concordia (CNR ISTI) Luca Trupiano (CNR ISTI) Theodore Patkos (FORTH ICS) Nikos Minadakis (FORTH ICS) Yannis Marketakis (FORTH ICS) Vangelis Kritsotakis (FORTH ICS) Daniele Bailo (INGV) Zhiming Zhao (UvA)
Participant(s):	All project partners
Work package no.:	3
Work package title:	Enhanced VREs
Work package leader:	Carlo Meghini (CNR)
Distribution:	PU
Version/Revision:	0.1
Draft/Final:	Final
Total number of pages (including cover):	125

What is VRE4EIC?

VRE4EIC develops a reference architecture and software components for VREs (Virtual Research Environments). This eVRE bridges across existing e-RIs (e-Research Infrastructures) such as EPOS and ENVRI+, both represented in the project, themselves supported by e-Is (e-Infrastructures) such as GEANT, EUDAT, PRACE, EGI, OpenAIRE. The eVRE provides a comfortable homogeneous interface for users by virtualising access to the heterogeneous datasets, software services, and resources of the e-RIs and also provides collaboration/communication facilities for users to improve research communication. Finally it provides access to research management /administrative facilities so that the end-user has a complete research environment.

Disclaimer

VRE4EIC has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676247.

Table of Contents

1 Introduction

This is the second and final deliverable on architecture in VRE4EIC. The first (D3.1) explained the methodology, approach and overall architecture. This deliverable provides details of the architectural components, interfaces and related considerations leading to the technical architecture and the canonical reference prototype. Components of the architecture have been utilised in EPOS and ENVRIplus and validated there; the reference architecture has been demonstrated with the Canonical Reference Prototype.

1.1 Differences with D3.1

In the section "The VRE4EIC Reference Architecture" of this document there are 5 component diagrams that were not defined in the Deliverable 3.1, namely:

- The App Manager component diagram
- The Node Manager component diagram
- The Workflow Executor component diagram
- The Workflow Configurator component diagram
- The Workflow Repository component diagram

As described in the related sections, these are very important components and their design has required an accurate investigation.

1.2 Structure of this deliverable and terminology

The deliverable is structured as follows:

- Section 2 presents the methodological approach that has been followed to design the VRE4EIC *Reference Architecture* (RA). A Reference Architecture is a high level description of a system in terms of its components and their relationships; essentially a RA provides a template that can be used to implement systems in a family of applicative domains. The VRE4EIC RA is designed to be used to build systems in the Virtual Research Environment (VRE) domain. A VRE built using the VRE4EIC RA is called an *eVRE*.

- Section 3 presents the RA conceptual components, using UML Component Diagrams and Package Diagrams. A conceptual component is a module that provides a set of homogenous functionalities. The conceptual components of the RA are derived in the first part of this section, following a vertical decomposition of functionalities in a multi-tiers view and then a horizontal distribution of functionalities within tiers.

- Section 4 describes the development of a technical architecture for implementing the RA. A technical architecture is a coordinated set of software modules, called *building blocks,* each of which implements a conceptual component so that the component can be integrated into an existing VRE (to enhance it to a eVRE) in the simplest possible way. In order to achieve this goal, the Microservice paradigm is followed in deriving the technical architecture.

- Section 5 describes the eVRE developed in VRE4EIC, called the *Canonical Reference Prototype* (CRP). It first introduces the general approach followed in the implementation (the choreography approach) and then presents the implementation of the building blocks defined in the Technical Architecture.

- Section 6 presents one GUI of the CRP, called the *Metadata Portal*.

- Section 7 describes the two enhanced VREs, EPOS and ENVRIplus.

- Section 8 concludes.

- The Annex documents the design of the RA by giving the list of Generalized Functions and the tables containing the signatures of the functions designed for the Conceptual Components. These structures are taken from D3.1 for self-containedness.

2 Deriving the VRE4EIC Reference Architecture

2.1 Introduction

In literature, software architecture development targets the definition of a structured solution able to satisfy the functional and non-functional requirements of an application domain. Thus the development of a software architecture involves different points of views: the user, the system (the IT infrastructure), and the business goals. For each of these areas, the key scenarios/requirements should be identified as well as quality attributes. Then requirements should be refined into architectural functions and mapped into specific architectural components or modules.
This process raises a series of criticalities and issues and should be carefully developed so as to avoid architectural failures or incompleteness. For this reason, in the last 10 years different approaches for architecture specification have been proposed, largely inspired from the Software Development Life Cycle, SDLC [ISO/IEC 12207]. This is a well-defined, structured sequence of stages targeting the specification and the successive development of the intended software product. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.
A typical SDLC includes different activities such as: understanding of business needs and constraints; elicitation and collection of requirements; functional architecture design; architecture design; implementation; testing; deployment; maintenance.
The order in which these activities are to be executed is usually defined into a specific software development process such as Waterfall model, incremental model, RUP, V-model, iterative model, RAD model, Agile model, Spiral model, Prototype model, to mention just a few [SE].
In the development of the Reference Architecture reported by the present deliverable, an incremental software development process largely inspired by the RUP process [RUP, UP] has been followed. In this Section, the main characterization of the process to the specific exigencies of the project constraints and activities are schematized. In particular, the Section provides details about activities concerning the software architecture specification and design that are: elicitation and collection of requirements; functional architecture design; architecture design.
The process characterization has been performed in collaboration with the different partners, considering the output of the other deliverables (see previous Section) and analyzing the current available proposals of VREs. As a consequence, the Reference Architecture specification provided in this deliverable will document two views:

1. Component diagram: it describes the components necessary to implement the eVRE functionalities. It visualizes the physical components in a system as well as the interfaces among them.
2. Interaction diagrams: they describe the type of interactions among the different components of the architecture and represent the part of dynamic behavior. We consider in particular sequence diagrams that emphasize the sequence of the message exchanges in time.

In the remaining of this Section, the main activities of the SDLC specifically characterized for the eVRE architecture specification are presented.

2.2 Elicitation and collection of requirements

Elicitation and collection of requirements is a fundamental stage in the eVRE architecture specification. In the context of the VRE4EIC project, this stage has been carried out by task 2.1. From a technical point of view, it involves different stages such as [ISO/IEC/IEEE 29148]:

1. High-level use case definition: A representative of the stakeholder community presents the business/mission drivers for the eVRE considering also quality attributes, security aspects.
2. Use case definition: Stakeholders express scenarios representing their concerns about the system prioritizing when possible the main ones.
3. Specification eVRE requirements: the main use cases are analysed and refined in more detail. The focus is on specifying what a system should do (the functional requirements) and on how the system should function (the non-functional, or quality, requirements) [ISO/IEC 25010, ISO/IEC 25030].

Figure 1 presents a UML class diagram where these entities are related to one another and to the entities subsequently derived in analyzing the requirements to derive the Reference Architecture. The analysis of requirements is explained in the next Section, with reference to the Figure 1.

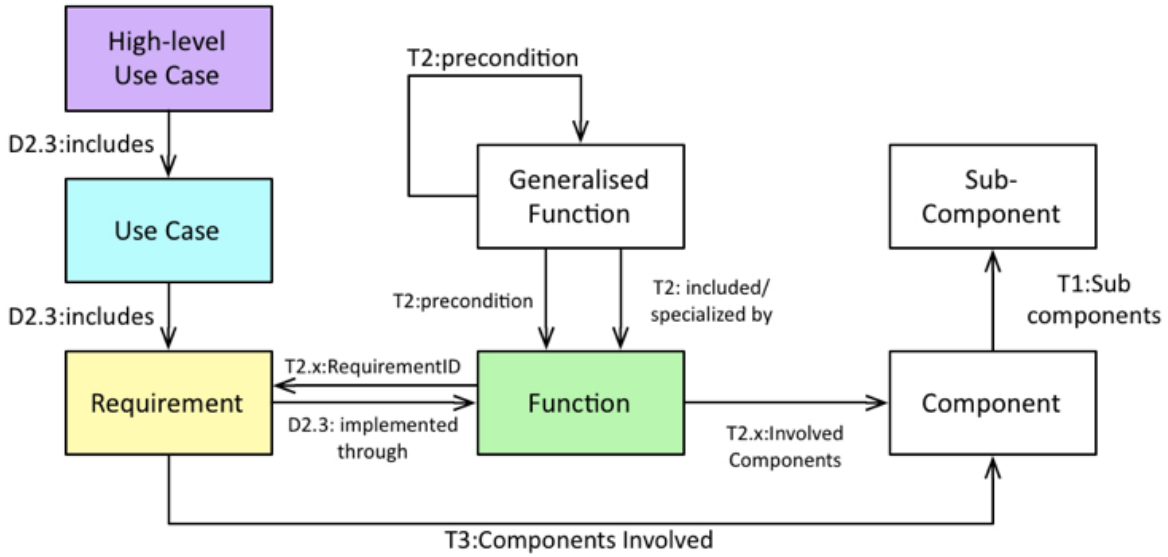


Figure 1 Entities involved in the Reference Architecture derivation process and their relationships.

2.3 Functional architecture design

The analysis started from the **Requirements** (yellow box). Each requirement has been considered individually, and the **Functions** (green box) required for its implementation have been derived. In the context of the VRE4EIC project, this stage has been carried out by Task 3.1. In order to ease the specification of functions, a set of **Generalised Functions** has also been derived, which are included or specialised by functions, or which may be used as preconditions by functions. Generalised functions are reported in Table 2 in Appendix, along with their relations to functions.
During the execution of this phase, several design guidelines have been followed [RUP, https://msdn.microsoft.com/en-us/library/ee658124.aspx]. A synthesis of the most important ones is provided below.

- *Separation of functions*: Isolate from the requirements different functions with as little overlap in functionality as possible. The important factor is minimization of interaction points to achieve high cohesion and low coupling.
- *Aggregation of functions:* Identify possible generalization or composition relations between the isolated functions to improve the organization and readability of the functional architecture design.
- *Learn from similar projects:* analyze similar projects and documentation so to derive an high conceptual-level architecture description focusing mainly on high view of modules/components and communications and interactions between them.
- *Reduce Responsibility:* Assign to each component or module the responsibility for only a specific functionality or aggregation of cohesive functionality.
- *Minimal Knowledge:* Each component or module should be unaware of the internal details of other components.

The association between requirements and functions is documented in Tables 2.1 and 2.2 given in Appendix. As the Figure also shows, this analysis of requirements into functions and components connects to the **Use Cases** (turquoise box) and the **High-level Use Cases** (purple box) derived in parallel by Task T2.2 and documented in deliverable D2.3. The connection is realized through requirements, and has been used for the assessment of the Reference Architecture.
Overall, the approach has allowed maintaining the relationship between use cases, requirements and components, thereby realizing the traceability of the Reference Architecture.

2.4 Architecture design

The **Components** that are required for the implementation of functions have finally been derived by Task 3.1. These components provide the functional architectural design and are refined and further decomposed into **Sub-Components** so that identified functional and nonfunctional requirements are completely satisfied. During this activity the interfaces of the components have also been defined and documented.
Also for carrying out this step several design guidelines have been followed [RUP, https://msdn.microsoft.com/en-us/library/ee658124.aspx]. Here below a synthesis of the most important ones is provided.

- Assess minimal knowledge: verify that each component does not rely on internal details of other components. In particular check that each component method is called from at least another object or component. Verify also that the method has information about how to process the request and, if appropriate, how to route it to appropriate subcomponents or other components.
- Avoid overloading of the functionality of a component: Avoid to overloaded components with many functions and applying the single responsibility and separation of concerns principles.
- Focus on communication between components: Understand the deployment scenarios and determine if all components will run within the same process, or if communication across physical or process boundaries must be supported—perhaps by implementing message-based interfaces.
- Define a clear contract for components: Components and modules should define a contract or interface specification that describes their usage and behavior clearly. The contract should describe how other components can access the internal functionality of the component,

module, or function; and the behavior of that functionality in terms of preconditions, postconditions, side effects, exceptions, performance characteristics, and other factors.

Components are detailed in Table 1, also given in the Appendix; Table 1 also documents the **Sub-Components** derived for each component.

2.5 REFERENCES

[ISO/IEC 12207] International Organization for Standardization, "ISO/IEC/IEEE 12207:2008 - Systems and software engineering -- Software life cycle processes," ISO/IEC, Mar. 2008.
[ISO/IEC/IEEE 29148] International Organization for Standardization, "ISO/IEC /IEEE 29148:2011 - Systems and software engineering — Life cycle processes — Requirements engineering," ISO/IEC/IEEE, Nov. 2011.
[SE] Ian Sommerville. Software Engineering. 9th Edition, Addison-Wesley 2011
[RUP] Rational Unified Process Best Practices for Software Development Teams, Rational Software White Paper TP026B, Rev 11/01, July 2003 https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
[UP] Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). The unified software development process (Vol. 1). Reading: Addison-Wesley.
[ISO/IEC 25010] International Organization for Standardization, "ISO/IEC 25010 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," ISO/IEC, Mar. 2011.
[ISO/IEC 25030] International Organization for Standardization, "ISO/IEC 25030 - Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality requirements," ISO/IEC, June 2007.

3 The VRE4EIC Reference Architecture

At the general level, the VRE4EIC RA conforms to the multi-tiers view paradigm used in the design of distributed information systems. Following this paradigm, we can individuate three logical tiers in eVRE:

- The *Application* tier, which provides functionalities to manage the system, to operate on it, and to *expand* it, by enabling administrators to plug new tools and services into the eVRE.
- The *Interoperability* tier, which deals with interoperability aspects by providing functionalities for: i) enabling application components to discover, access and use eVRE resources independently from their location, data model and interaction protocol; ii) publishing eVRE functionalities via a Web Service API; and iii) enabling eVRE applications to interact each other's.
- The *Resource Access* tier, which implements functionalities that enable eVRE components to interact with eRIs resources. It provides synchronous and asynchronous communication facilities

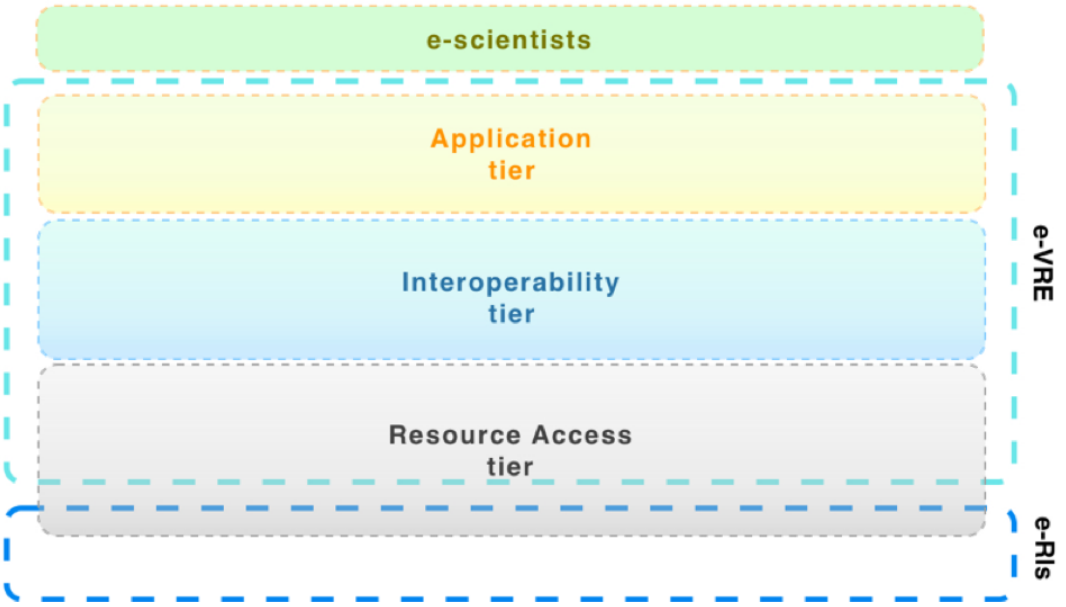


Figure 2 Architectural tiers in a VRE

The Figure 2 depicts the logical tiers of eVRE and shows their placement in an ideal space between the e-scientists that use the eVRE and the e-RIs that provide the basic resources to the eVRE.
According to this approach a system is composed by an integration infrastructure where a set of software components can be deployed, these components implement the system functionalities and potentially can be specified, developed and deployed independently of one another.
Based on these considerations and on the analysis of the requirements, for the basic integration infrastructure of eVRE we have individuated a set of basic functionalities grouped into six *conceptual components*:

- The eVRE management is implemented in the **System Manager** component. The System Manager can be viewed as the component enabling Users to use the *core* functionalities of

the eVRE: access, create and manage resource descriptions, query the eVRE information space, configure the eVRE, plug and deploy new tools in the eVRE and more.

- The **Workflow Manager** enables users to create, execute and store business processes and scientific workflows.
- The **Linked Data (LD) Manager** is the component that uses the LOD (Linked Open Data) paradigm, based on the RDF (Resource Description Framework) data model, to publish the eVRE information space - i.e. the metadata concerning the eVRE and the e-RIs in a form suitable for end-user browsing in a SM (Semantic Web)-enabled ecosystem.
- The **Metadata Manager** (MM) is the component responsible for storing and managing resource catalogues, user profiles, provenance information, preservation metadata used by all the components using extended entity-relational conceptual and object-relational logical representation for efficiency.
- The **Interoperability Manager** provides functionalities to implement interactions with e-RIs resources in a transparent way. It can be viewed as the interface of eVRE towards e-RIs. It implements services and algorithms to enable eVRE to: communicate synchronously or asynchronously with e-RIs resources, query the e-RIs catalogues and storages, map the data models.
- The **Authentication, Authorization, Accounting Infrastructure (AAAI**) component is the responsible for managing the security issues of the eVRE system. It provides user authentication for the VRE and connected e-RIs, authorisation and accounting services, and data encryption layers for components that are accessible over potentially insecure networks.

In order to improve modularity in design and development of the architecture, the functionalities of every conceptual component are partitioned, and each group of functionalities is assigned to a sub-component, the Table 1 indicates for each conceptual components its subcomponents and their role and main functionalities.

Table 1 VRE4EIC: sub components

Component	**Sub-components**	**Description**
AAAI Component		Manages security, privacy and trust aspects of the eVRE and its connections to the e-RIs
	Authentication	Manages user authentication for the eVRE and connected e-RIs (single sign on), interfaces with external identity provider services.

	Authorization	Manages user authorisations (rolebased access) based on (CERIF) metadata provided by the Metadata Manager.
	Accounting	Manages accounting and billing of resources for which payment is required, based on (CERIF) metadata provided by the Metadata Manager.
	Encryption	Provides encryption facilities.
Metadata Manager (MM)		Manages metadata about eVRE entities: resource descriptions, user descriptions, provenance information, preservation metadata etc. (CERIF)
Interoperability Manager (IM)		Manages interactions with e-RIs
	Query Manager (QM)	Manages local and distributed queries, collects result sets
	Data Model Mapper (DMM)	Manages data and query format conversion
	Adapters	Components that synchronously interact with e-RIs resources
	Message-Oriented Middleware (MOM)	Manages asynchronous interactions with eRIs resources using messaging protocols
	eVRE Web Services (eVRE WS)	Enable external applications to interact with eVRE
Workflow Manager (WM)		Manages business processes and scientific workflows, using the Metadata Manager for storing information on workflows

	Workflow configurator	Provides functionalities to build/edit/store *execution plans,* to control and monitor processing flows execution.
	Workflow executor	Manages workflow execution, including data staging
	Workflow repository	Provide functionalities to store and retrieve workflows, workflows will be published using LD manager
Linked Data Manager (LDM)		Manages the publication of information inas Linked Open Data
	SPARQL Endpoint	Allows retrieving resources and services published by eVRE as RDF documents
	LD API	The LD API maps CERIF metadata records in RDF, implements metadata enrichment of RDF records, i.e. adds to records typed links to vocabularies and thesaurus entries, Implements content negotiation
System Manager (SM)		Implements functionalities to *install* and manage an eVRE, e.g. specify the resources, specify the apps.
	Node Manager (NM)	Implements the functionalities to deploy, manage and run an instance of eVRE on a specific hardware
	User Manager (UM)	Manages user profiles and provides collaboration/ communication functionalities for users. It provides the functionalities to add/update/remove user profiles, to set up users permissions, to manage users preferences, to configure users working environments

	Resource manager (RM)	Manages resource information implementing add/update/remove operations on resource descriptions, associating resources to security policies, etc.
	App Manager (AM)	Provides functionalities to deploy and manage applications that operate on eVRE resources. It can be used also to embed applications such as Wiki or forums etc.

3.1 VRE4EIC conceptual components

This Section describes the Conceptual components introduced in the previous section, for every component is shown a UML Package Diagram containing the sub-components and one or more component diagrams showing the interactions of the component with other eVRE components. Interfaces and method signatures are reported in the section 10.3 of this Document.

3.2 The System Manager component

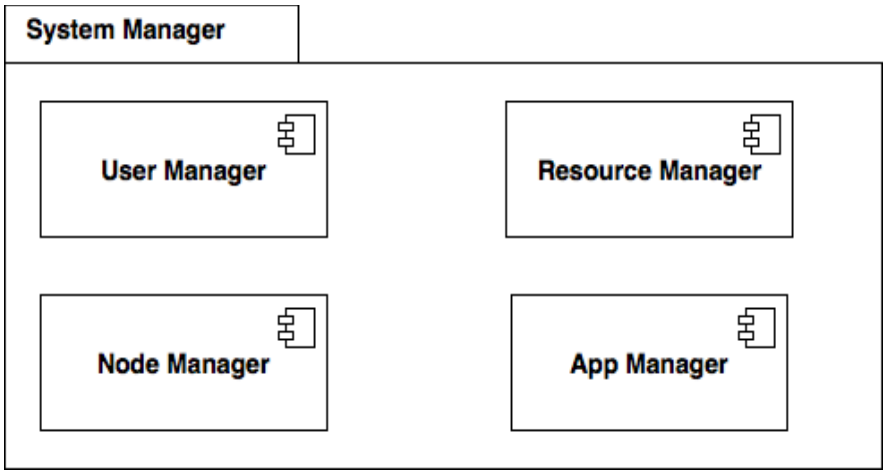


Figure 3 The System Manager

The System Manager is composed by 4 sub-components.

The **User Manager** is the component responsible for managing User Profiles, providing Authentication mechanisms and enabling users to receive Notifications for events they have subscribed. To perform its activities the User Manager interacts with:

- the Metadata Manager to store/retrieve/update User Profiles
- the AAAI to implement authentication, encryption, authorization and accounting tasks

- the AuthenticatorApp to implement login via external authenticator

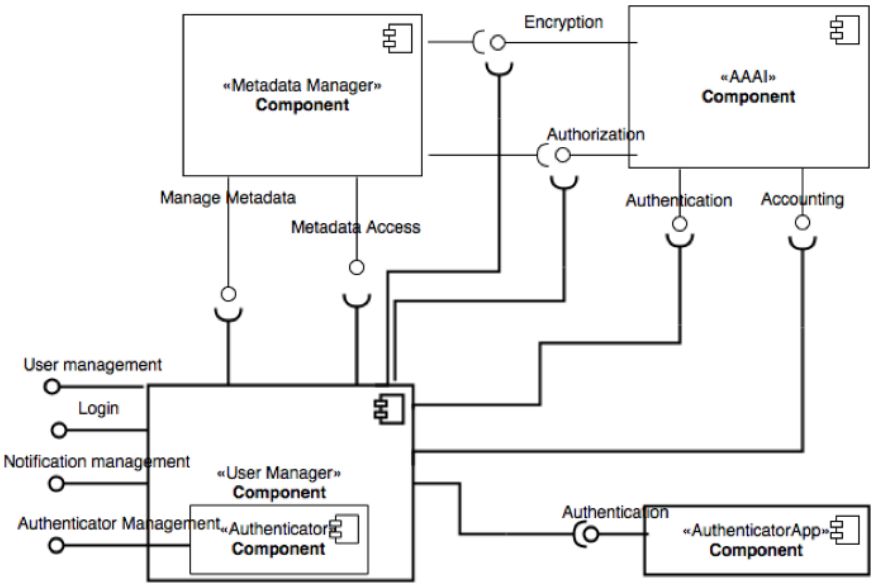


Figure 4 The User Manager Component Diagram

Interface provided by User Manager component are described in details in Table 14 of the document Annex.

The **Resource Manager** is the component responsible for managing information about resources provided by RIs and other infrastructures, it communicates with remote resources via Adapters or asynchronous messaging. The Resource Manager interacts with:

- Metadata Manager: to store, manage and retrieve information about resources
- AAAI component: to check permissions when interacting with external resources and to use encryption functionalities if needed
- Model Mapper Component: to map data when interacting with external resources
- RI Resource Adapter: for synchronous interactions with the external resource provided by a RI
- MOM component for asynchronous interactions with the external resource

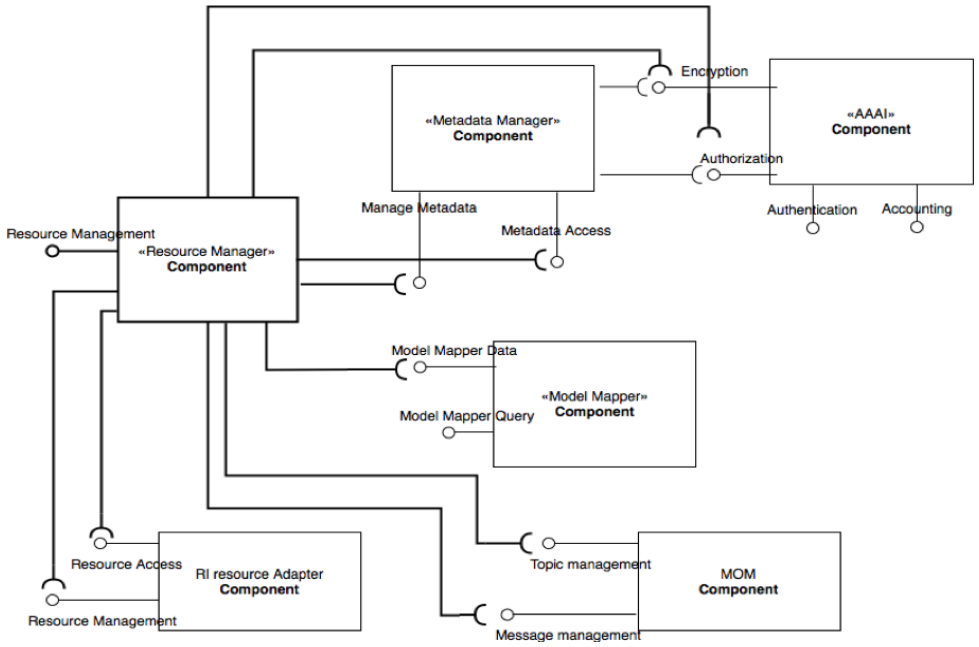


Figure 5 The Resource Manager Component Diagram

The goal of the **App Manager** is to provide functionalities to enable external applications to be embedded and used into the EVRE system. Generally speaking this means that such a component should implement a set of facilities to manage:

1. the deployment of external applications in EVRE
2. the life-cycle of these applications (install/start/stop/update/uninstall)
3. the publication and the discovery of these applications

The idea is to build the App Manager as a lightweight, unobtrusive component that will interact with external applications to track their lifecycle and their usage.
However, creating an App Manager able to automatically manage lifecycle and usage for every possible external application embedded in the EVRE is not feasible: we'll individuate a set of standards and technologies and implement the App Manager for applications adopting those standards , applications not implementing the selected technologies will be embedded by extending the App manager with *ad hoc* sub components.

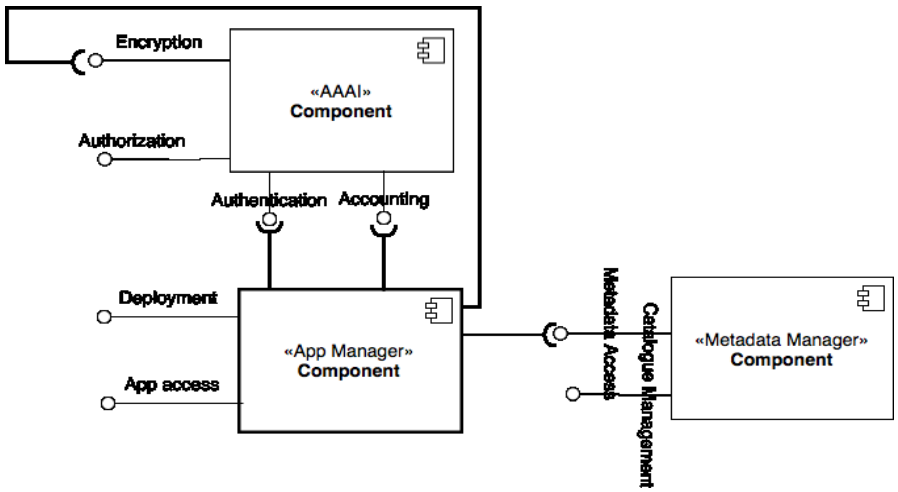


Figure 6 The App Manager Component Diagram

The **Node Manager** is responsible for managing the infrastructure, it implements the functionalities to deploy, manage and run an instance of eVRE on a specific hardware.

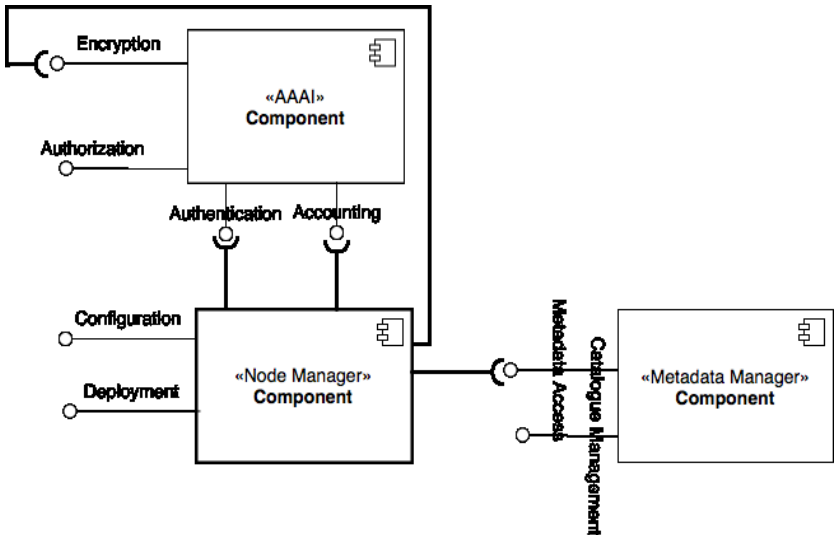


Figure 7The Node Manager Component Diagram

3.3 The Workflow manager Component

The Workflow Manager is responsible for managing both Business and Scientific workflows[1].In our vision Scientific Workflows represent experiments conducted by scientists, therefore the WF component will provide a workflow repository and interoperate with other workflow repositories to facilitate the reuse and reproducibility of scientific experiments. Information about workflows are stored in the Metadata Manager and will be published also as Linked Open Data via LD Manager.

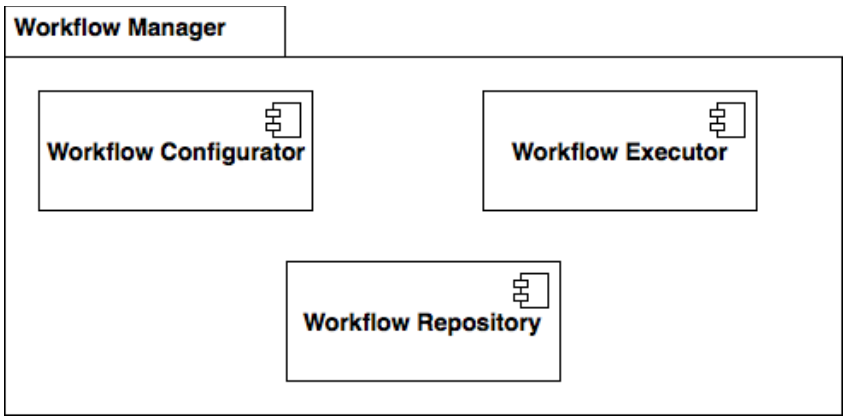


Figure 8 The Workflow Manager

The Workflow Manager is composed by three subcomponents: the Workflow Configurator, the Workflow Executor and the Workflow repository.
The Figure 9shows the overall component diagram for the Workflow Manager.

[1] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. Scientific workflows: Business as usual?,7th Intl. Conf. on Business Process Management (BPM), LNCS 5701, Ulm, Germany, 2009

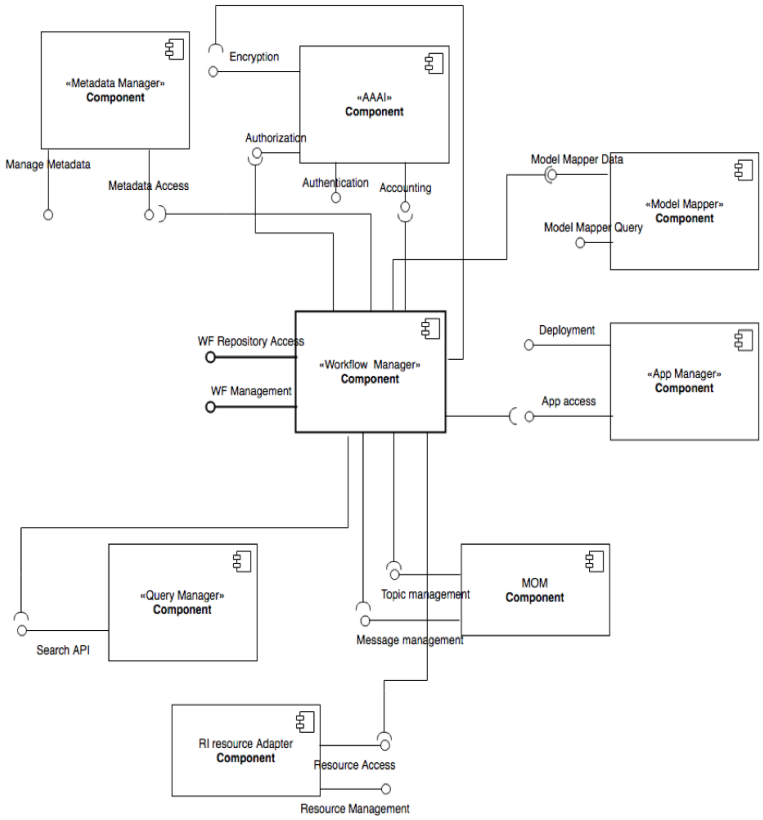


Figure 9 The Workflow Manager component

Interfaces and method signatures are described in details in Table 19 and Table 20 of this document Annex section. The functionalities of the Workflow Manager have been partitioned in three categories: functionalities to create, update and manage workflows, functionalities to execute workflow, and functionalities to manage the repository of the workflows. Therefore, we have defined three sub components for the Workflow Manager.

The **Workflow Configurator**, is the sub component that provide functionalities to create and manage workflows, it mainly interacts with the Metadata Manager and with The Query Manager to access resource the component diagram is reported below.

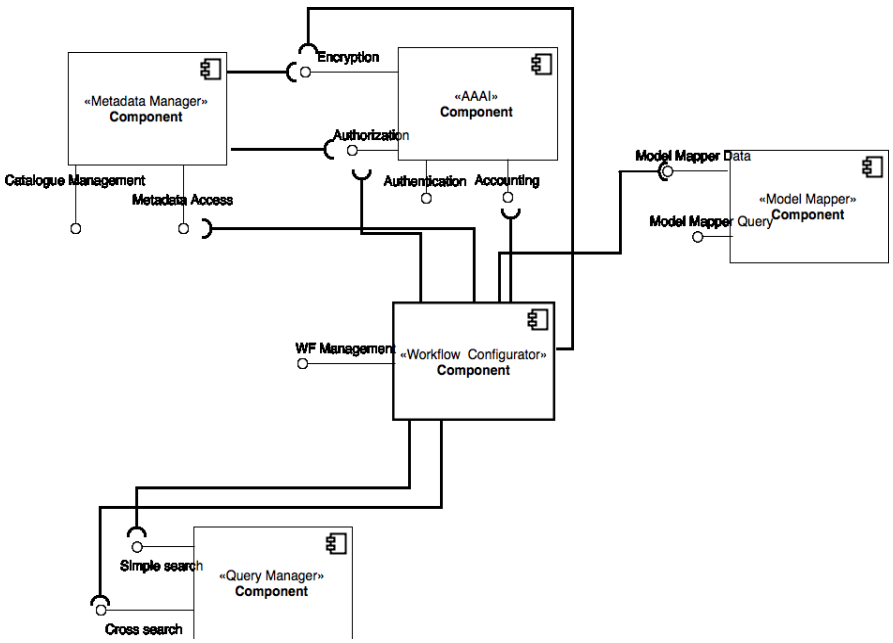


Figure 10 Workflow Configurator Component Diagram

The **Workflow Executor** provides functionalities to execute workflows. To implement its functionalitiesit may interact with many other components, the Component diagram is below

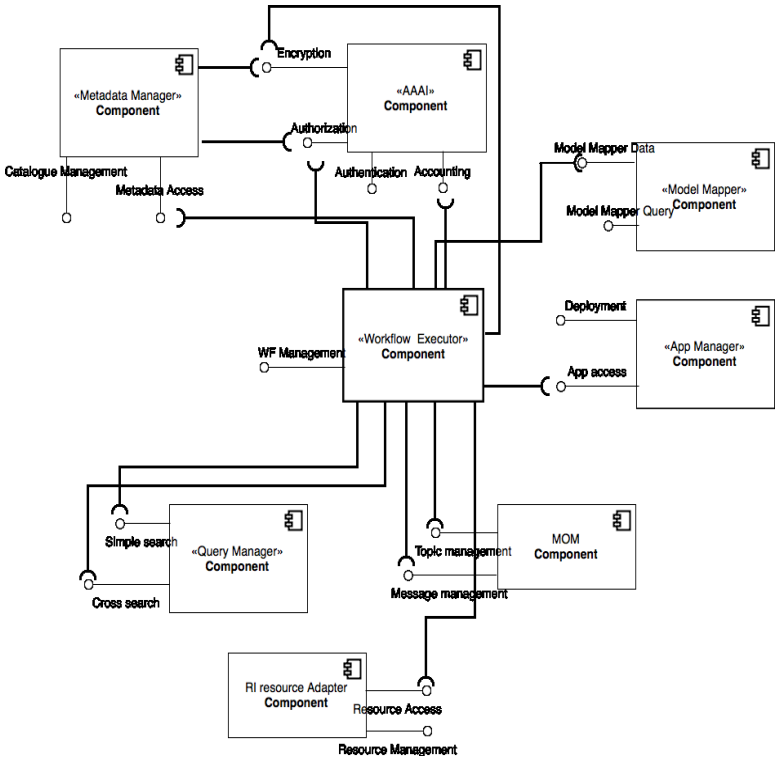


Figure 11Workflow Executor Component Diagram

The **Workflow Repository** manages the storage where workflow descriptions are stored and provides functionalities to access these descriptions.

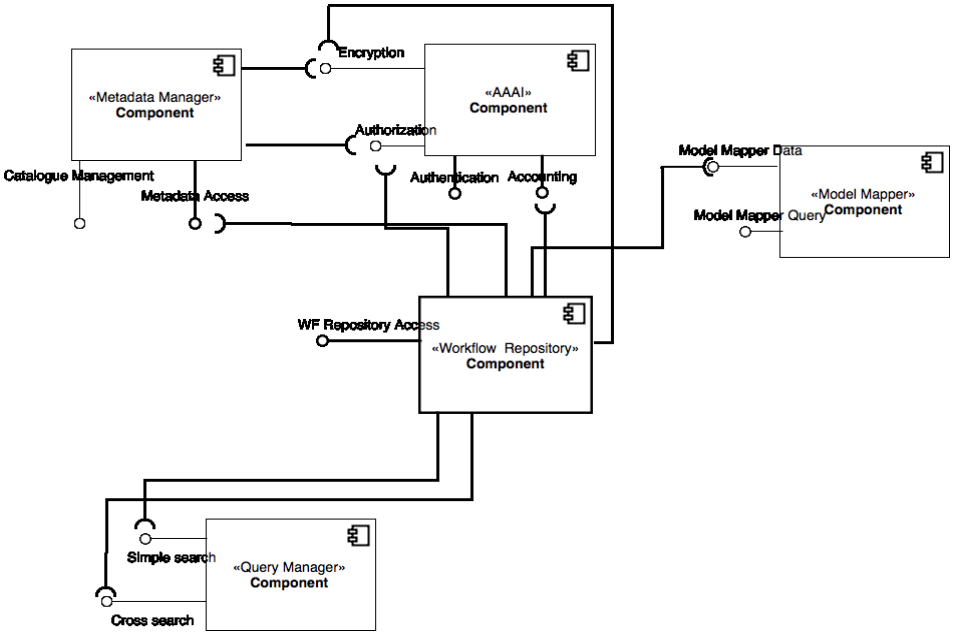


Figure 12 Workflow Repository Component Diagram

3.4 The Metadata Manager

The Metadata Manager is responsible for storing, manipulating and exposing metadata information about various resources. It contains a set of catalogues and repositories and stores information with respect to a set of predefined schemas.

The Metadata Manager component contains a set of subcomponents that deal with particular functionalities: the Thesaurus, the Provenance Manager, the Preservation Manager etc.

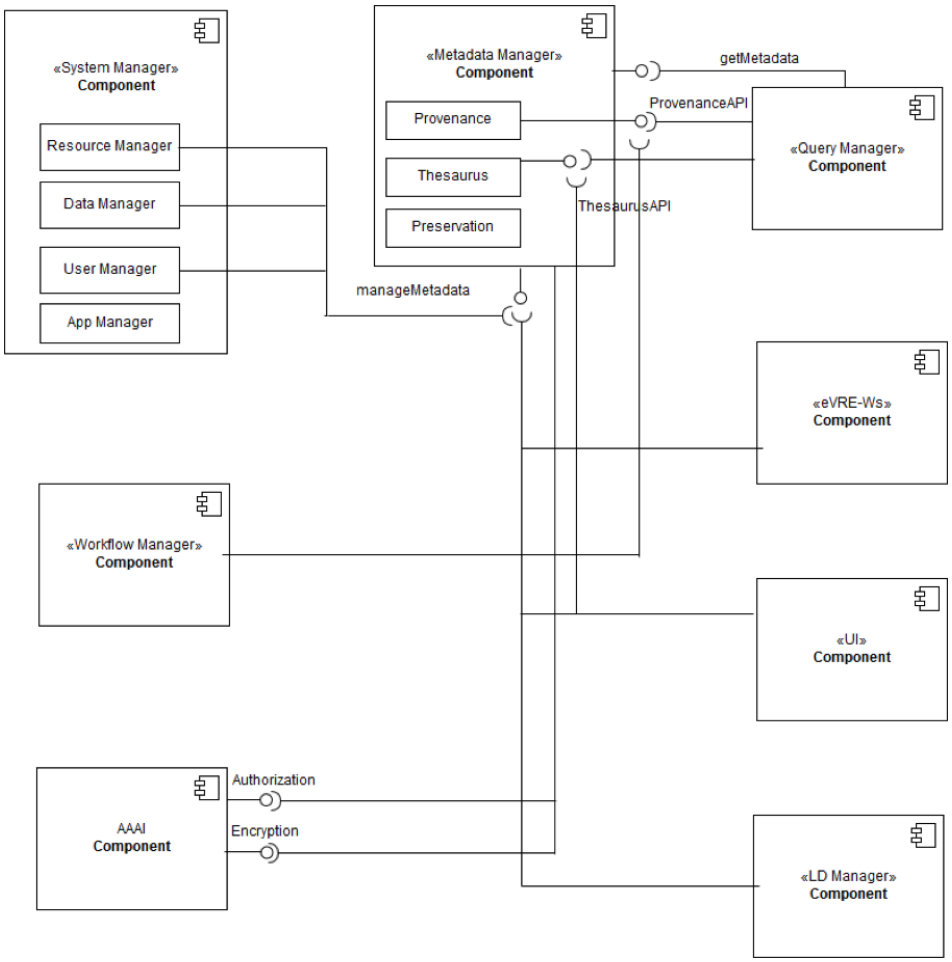


Figure 13 The Metadata Manager Component Diagram

Interfaces and signatures of methods are described in section 10.3.5 of the Annex section.

3.5 The Interoperability Manager

This conceptual component provides functionalities to implement interactions with external applications and frameworks. In particular it is used to interact with e-RIs, for instance to access their resources . Essentially it must provide functionalities to enable eVRE to: communicate synchronously or asynchronously with external agents. This is a key component: for instance, its functionalities will be used by existing VREs in order to use the enhancing services provided by eVRE.

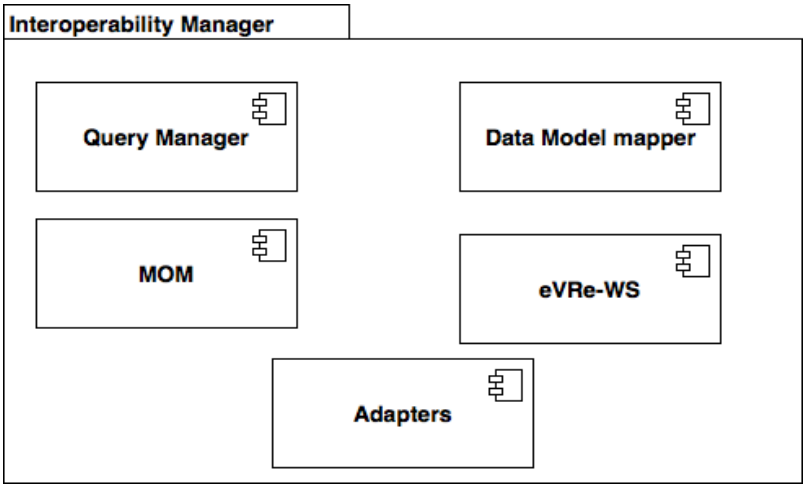


Figure 14 The Interoperability Manager

The **Query Manager** is the component responsible for managing the querying capabilities of the infrastructure. This component receives as input a set of query requirements (in terms of keywords, query preferences, dataset catalogues, etc.) and manages the entire process of

- preparing the query,
- splitting it into subqueries,
- submitting the subqueries into the proper systems,
- receiving the results, and
- integrating them in order to send them to the user as a unified set.

The QueryManager is a core component (a super-component) that aggregates and exposes the functionalities of various subcomponents (i.e., Query Analyzer, Query Mediator, Query Integrator, Query Publisher, etc.).

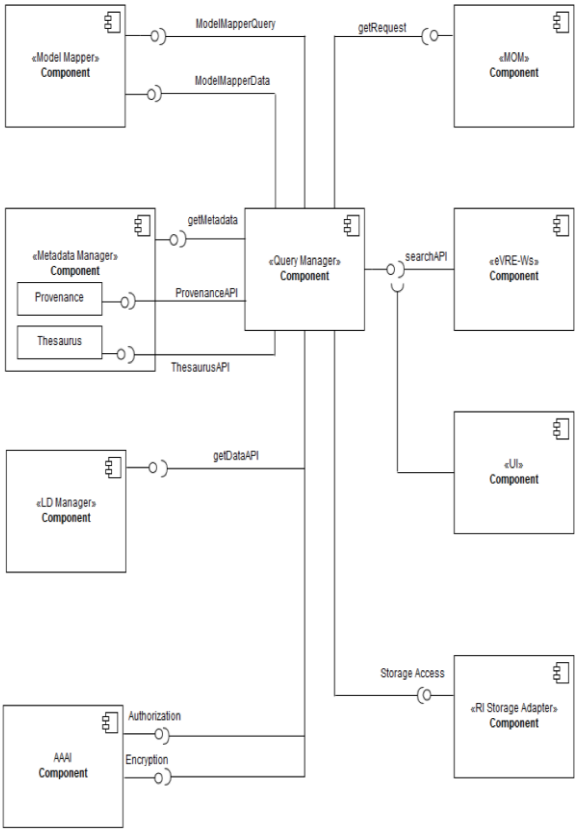


Figure 15 The Query Manager Component Diagram

The **Data Model Mapper** is responsible for performing the required mappings and storing the particular information (the mappings themselves) in its repository.

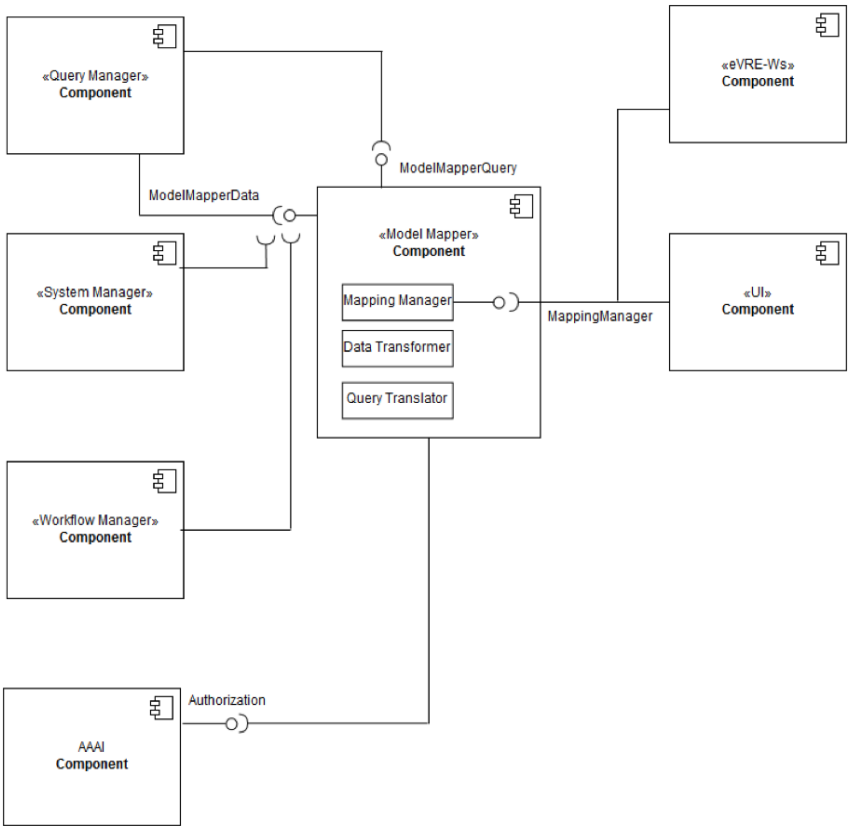


Figure 16 The Data Model Mapper Component Diagram

The **Resource Adapter** (or Adapter) indicates a set of components: in eVRE an Adapter is a software module that wraps an external eRI resource. The Adapter acts as a middleware between an eVRE component and an eRI resource, its role is to reduce dependency of eVRE from eRI resources. Adapters are specific for e-RI resources and use synchronous standard protocols to interact with the resource. The adapter could be deployed in the eVRE system or in the e-RI environment; it could also be split in subcomponents. Interfaces of Adapters depend on the resource they wraps.
For asynchronous communications the Interoperability manager uses a **MOM** component, which implements a message exchange protocol based on publish/subscribe paradigm.

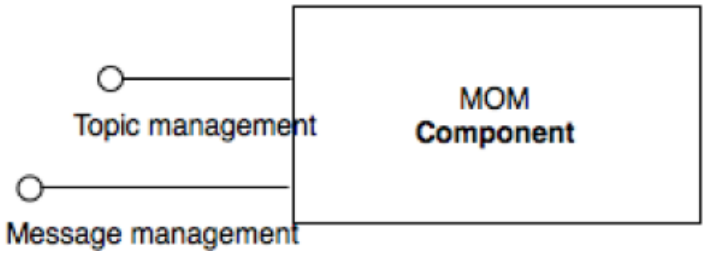


Figure 17 The Message Oriented Middleware Component

The **eVRE WS** component implements the Web Service API for eVRE architecture. It is a crucial component, it is used by external agents (applications) to access the functionalities of the eVRE. It could be also used as *service level integration middleware* for expanding the eVRE with new functionalities.

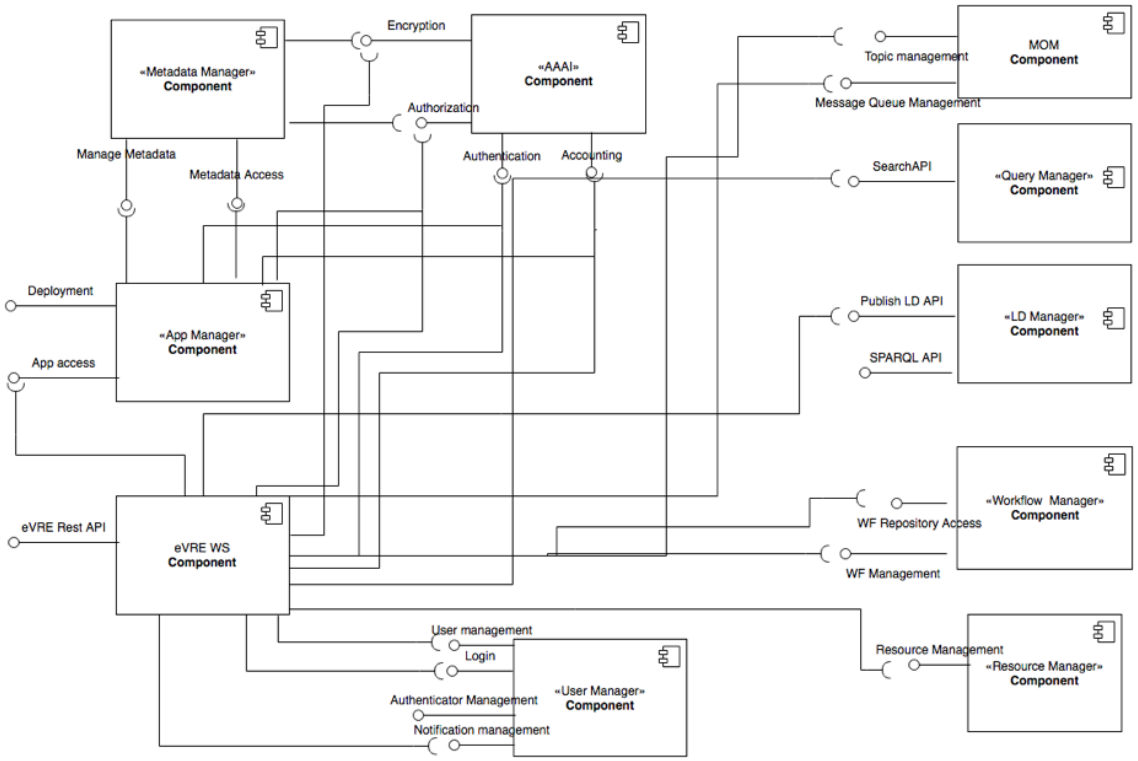


Figure 18 eVRE WS Component Diagram

3.6 The Linked Data Manager

The **LD Manager** is the component responsible for publishing resource descriptions with respect to the principles of Linked Open Data. The process of publishing contains the transformation of resources, the generation of (resolvable) URIs, their linking, etc.

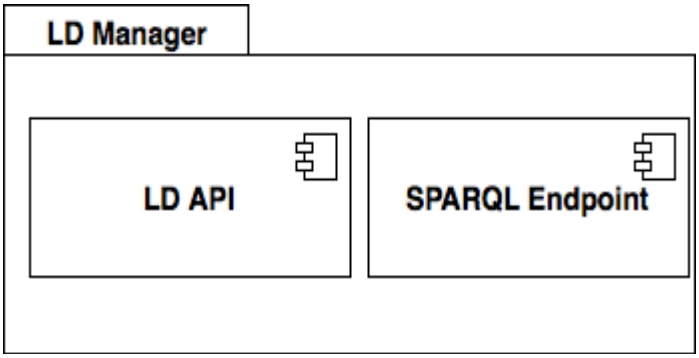


Figure 19 The LD Manager

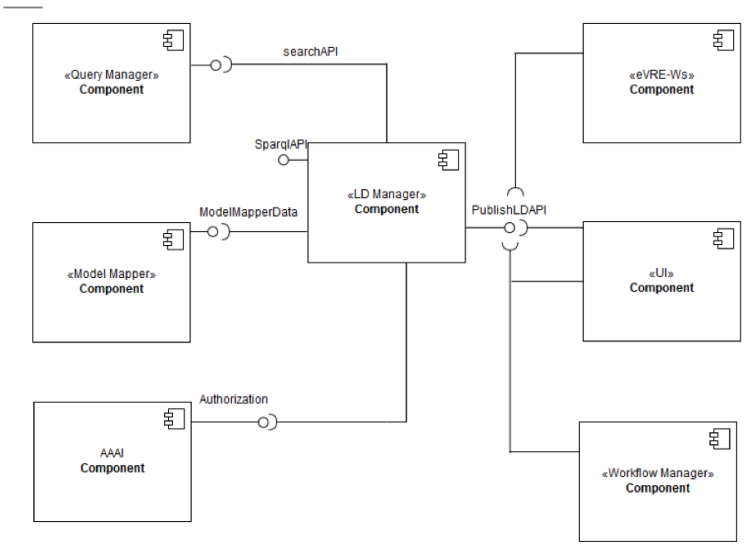


Figure 20 The LD Manager Component Diagram

3.7 The AAAI component

This component implements the authentication, authorisation, accounting and data encryption backend services for the eVRE architecture. It interacts with external identity providers (IdP) to enable single sign on across the various connected infrastructures. For any authenticated user, it provides authorization services by using attributes provided by the external identity provider (if any). These will be extended by using (CERIF) information by interfacing with the Metadata Manager component.

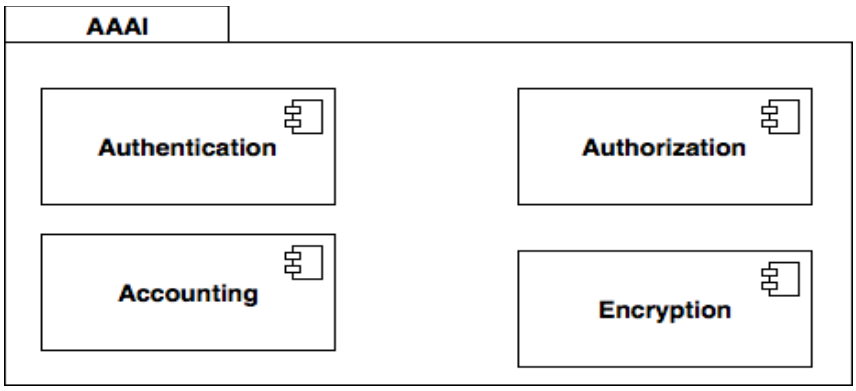


Figure 21 The AAAI Component

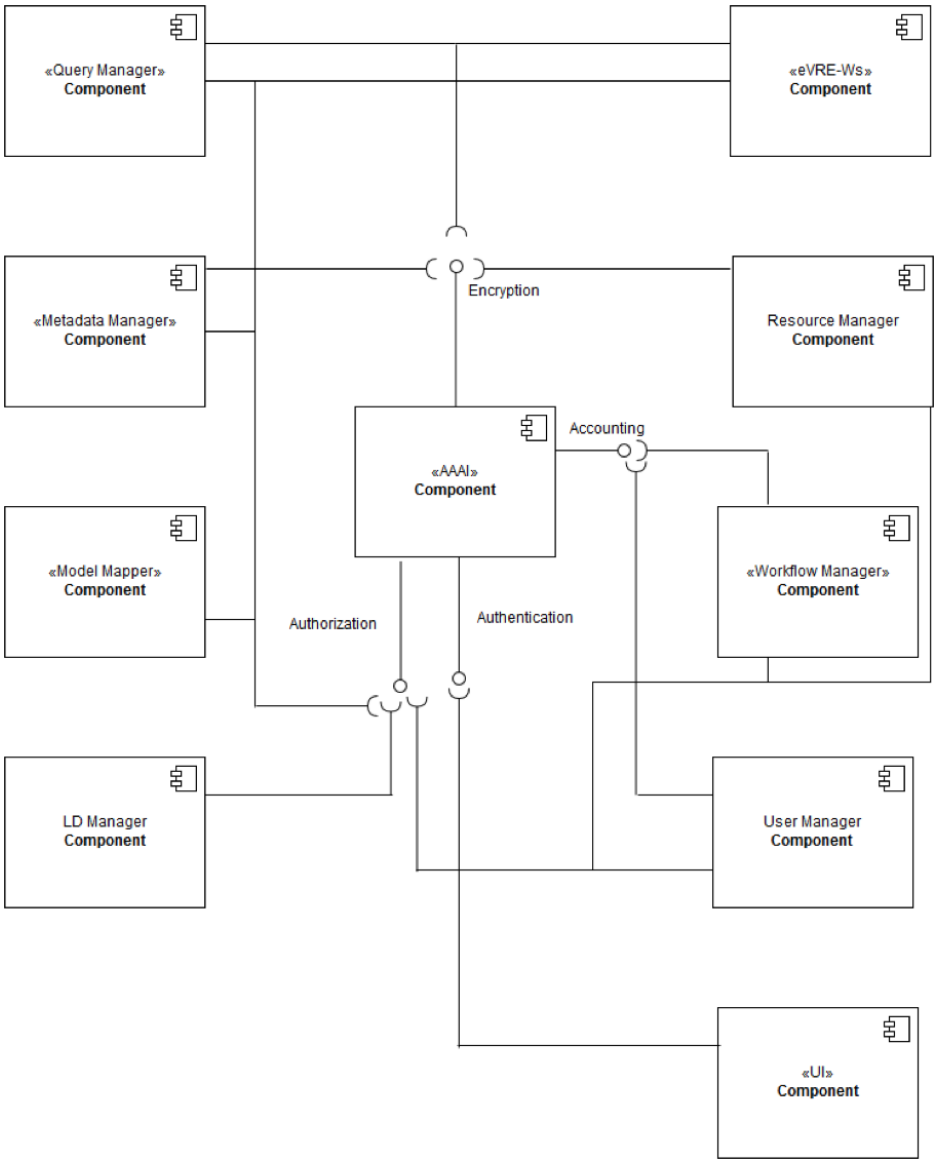


Figure 22 The AAAI Component diagram

4 The VRE4EIC technical architecture

A Technical Architecture has been defined in order to implement a specific eVRE called Canonical Reference Prototype (CRP) which complements the Reference Architecture, by selecting a set of suitable implementation techniques and open-source components.
The key point in the derivation of the technical architecture, has been the VRE4EIC non-functional requirements defined in the project proposal:

1. The developed Virtual Research Environments must be a dynamic system: it should reuse and integrate existing VRE tools, services, standardized building blocks and workflows where appropriate [vre4eic], and develop new innovative ones where needed .
2. The eVRE should be applicable to different multidisciplinary domains, i.e. it can be potentially used in every research domain.
3. The eVRE functionalities should be exposed as services in a standardized way to enable developers to easily use them to develop new applications.
4. The eVRE must provide innovative standard software services to be retro-fitted to existing VREs to enhance them for their own domain purposes and for interoperability.

From the architectural point of view the above requirements mean that the eVRE system must be easily expandable (by adding or replacing software components), highly modular (every architectural component should be independently deployable) and capable of supporting technology heterogeneity. We decided to adopt the Microservices approach for our technical architecture, since the two key concepts of Microservices architecture [Newman] fits the above requirements:

- loose coupling: every service knows as little as it needs to about the components with which it cooperates; this enables the microservices to be independently deployable on existing VREs or replaceable in different domains
- high cohesion: components with related behavior stand together (i.e., related logic is kept in one service); changing the technology used to implement a microservice does not affect other microservices.

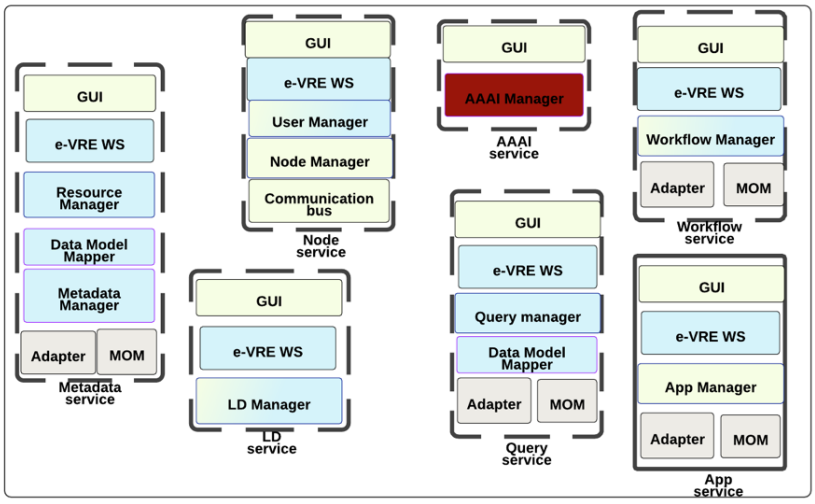


Figure 23 The eVRE Technical Architecture

The set of conceptual components functionalities, defined in the VRE4EIC RA, have been partitioned according to the non-functional prerequisites, and the resulting subsets have been implemented as *microservices*.

We adopted a number of design principles to define the microservices in our architecture:

- *Use Asynchronous communications*. We adopted an event-driven communication model, for Microservices interactions. The eVRE Microservices interacts asynchronously by exchanging *messages*. According to this model, a publisher generates a message whenever an event occurs, containing information about the event that has fired the message. The message is conferred to a third party, which will asynchronously deliver it to one or more consumers. Upon receiving of a message, consumers react according to the type of message received. The event-driven interaction model is not blocking: the microservice initiating the communication does not wait for answer. Additionally, it is highly decoupled: a producer does not have any way of knowing who is going to react to its messages. From an architectural point of view an event-driven interaction model reduces communication latency and improves scalability and flexibility of eVRE (Requirements 1, 2, 4 above): new publishers or subscribers can be added to (or can be removed from) an event without the other publishing/consumer microservices need to know it.

- *Distributed processes management.* When creating a Microservices-based architecture it is important to choose how to deal with the problem of managing processes that stretch across the boundary of individual services. The two possible approaches are: to have a central service that guides such kind of processes (called Orchestration) or to implement the logic to monitor and track processes in each involved microservice (called Choreography). Considering our requirements, we decided to favour distribution and avoid a central point of control. Our choice therefore went for the Choreography approach. A significant issue when adopting Choreography approach is to implement a strategy to obtain the so-called "eventual consistency[2]" [Newman] of the system when dealing with distributed processes. We tried, as [Newman] suggests, to reduce the possibility of having distributed processes by individuating at design time those operations that can involve multiple microservices and trying to 'keep' the execution of these operations inside a single microservice. Checking the UML Component diagram of eVRE, we noticed that a significant distributed process is the management of resource descriptions (resource descriptions are metadata records containing information about resources used by the eVRE). The **Resource Manager** is responsible for this task; it may use the **Metadata Manager** as repository, the **Data Model Mapper** for metadata conversion, **Adapters** to interact with remote services. The Resource Manager provides functionalities to implement several crucial features of the eVRE system (see Deliverable 2.1 for the list of features): PVF1 Data provenance information, CF3 Data storage and preservation, IF1 Data identification, PF2 Metadata harvesting etc. For this reason we decided to put the components cited above in a single microservice called the Metadata service. For those processes that cannot be managed at design phase, a number of technical solutions can be adopted to implement consistency, some of them really sophisticated and we are investigating a framework able to deal with this issue in eVRE.

- *Avoid service coupling because of component dependencies*. This is a crucial issue in defining a Microservices-based architecture. In a number of significant Use Cases individuated it is required that eVRE interacts with external resources, for instance, when the Resource Manager want to update catalogues contained in the Metadata Manager or the Workflow Manager executes tasks involving remote datasets. These interactions are mediated by **Interoperability Manager** (IM) via Adapters (for communication protocols) and Data Model Mapper (for data conversion). Implementing these components in separate microservices introduces a dependency that can result in an inefficient implementation of Requirement 4:

[2] "*Rather than ensuring that the system is in a consistent state all the time, instead we can accept that the system will get it at some point in the future*" [Newman].

for instance when the microservice implementing the Workflow Manager or the Resource Manager will be retro-fitted into an existing VRE we need to retrofit also the microservice implementing the IM, thus deploying in the hosting VRE a number of software modules that are possibly never used. Additionally, if a change is required in a subcomponent of the IM because a new technology is required by a specific microservice (Requirement 1 could cause this) we need to be sure that this change doesn't have side effects affecting interactions with other microservices. To avoid coupling between microservices we decided to *embed* the IM sub-components used to interact with external resources into the microservices using them: each microservice will have its own implementation so it can be easily retrofitted and possible changes won't affect other microservices.

- *Efficiently manage integration with third-party software*. Integration is a key feature of eVRE, the system provides functionalities to integrate external agents at *data level* (using the Metadata Manager functionalities), at *application level* (via eVRE WS and ad-hoc Adapters) and it provides also *asynchronous* integration via MOM component. In the design phase, we have assigned to the **App Manager** component the role of manager and monitor of life-cycles of integrated applications; due to its crucial role we decided to create a specific microservice for it, called App Service.

- *Efficiently manage coexistence of different endpoints*. An important advantage provided by Microservices architectures is that different services can be installed on different nodes and also that different version of the same service can coexist on the same node. To implement this, we decided to partition the **eVRE WS** subcomponent: every microservice implements those Web Services endpoints enabling to access its functionalities. Then the eVRE WS component is the composition of all microservices WS endpoints.

The Table 2 lists the Microservices resulting from the application of the above principles to the Reference Architecture. Each microservice is described by listing the main component(s) that it implements and the auxiliary components that are part of it, as a consequence of the application of the above principles.

Table 2 VRE4EIC Microservices

Microservice	**Main component(s)**	**Auxiliary components**
Node Service	Node Manager, User Manager	eVRE WS
App Service	App Manager	eVRE WS MOM, Adapters (to *avoid service coupling, to implement Choreography)*

Metadata Service	Metadata Manager	eVRE WS Resource Manager, Data Model Mapper MOM, Adapter (to *avoid service coupling and implement Choreography)*
LD Service	LD manager	eVRE WS
Workflow Service	Workflow Manager	eVRE WS MOM, Adapter (to *avoid service coupling)*
AAAI Service	AAAI Manager	eVRE WS
Query Service	Query Manager	eVRE WS Data Model Mapper MOM, Adapter (to *avoid service coupling and implement Choreography)*

4.1 Deriving the technical architecture

4.1.1 The GAP analysis

In order to identify the components of the Reference Architecture to be implemented by the project, a Gap Analysis (GA) has been carried out. The main goal of the GA is to investigate the functionalities provided by a number of VREs and to identify the gaps between such VREs and the eVRE defined in VRE4EIC. The Gap Analysis [Deliverable 3.2] has derived the following ranking (in decreasing order of priority):

1. AAAI - Authentication, Authorization, Accounting Infrastructure, MM - Metadata Manager
2. IM - Interoperability Manager, LDM - Linked Data Manager
3. QM - Query Manager, DMM - Data Model Mapper
4. AM - App Manager
5. WM - Workflow Manager
6. RM - Resource Manager, UM - User Manager

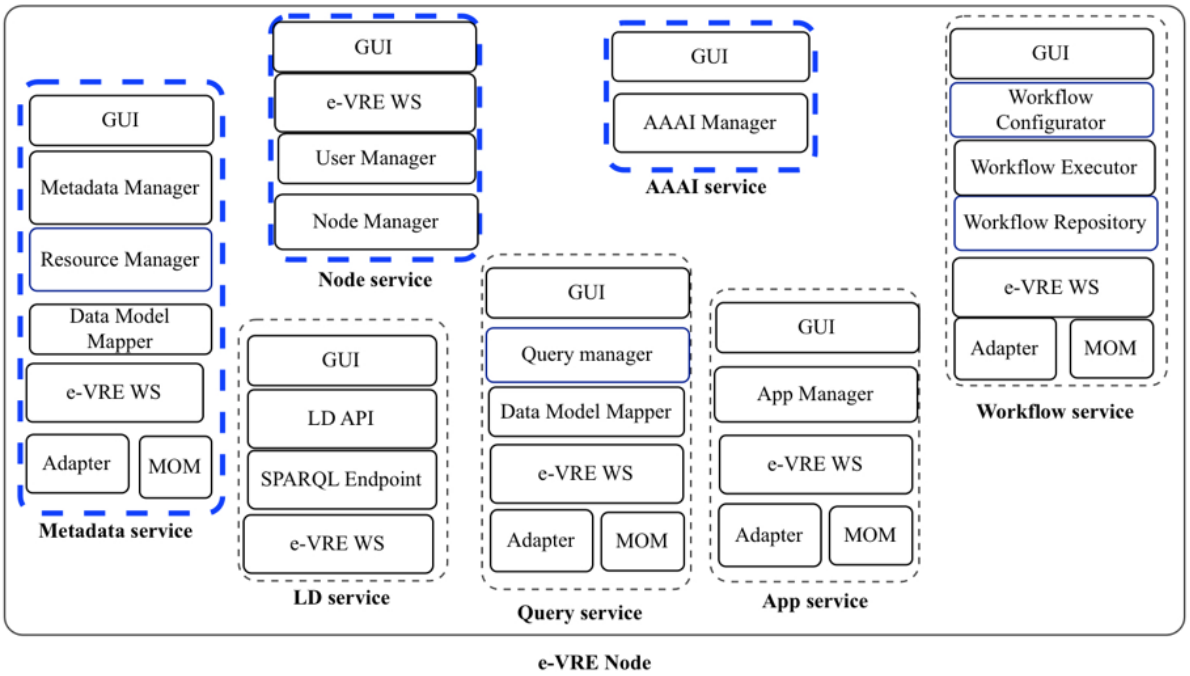


Figure 24 Development priorities

As shown in Figure 24 the two components with highest priority are implemented by two microservices: the AAAI service and the Metadata service. The Node service, that implements the functionalities to deploy, manage and run an instance of eVRE on a specific hardware, needs also to be implemented in order to be able to create and manage an installation of the RA.
Due to the nature of Microservices architecture, microservices can be developed by independent teams which only need to share the API contracts of what their services can do and how others can use them. We set up three development teams, each one developing a Microservice.
The following sections describe the overall implementation choices, detailed information can be found in the Deliverable 3.3 [D 3.3].

4.1.2 Node Service technological choices

The Node Service includes three components: the Node manager, the User Manager and the eVRE WS.

The **Node Manager** implements the internal communication infrastructure, coordination and configuration of the distributed services, and provides helper classes to other components of the Node Service in order to ease the development of new functionalities.
Several functionalities have been developed from scratch in components whose libraries can be easily extended and reused in generic eVRE services, other functionalities have been delegated to components that make use of de facto standard libraries and services. Design goal for these components has been the isolation of legacy code.
The legacy code included in the current implementation, consists of Apache Zookeeper [ZooKeeper] and Apache ActiveMQ [ActiveMQ] open source software.

The **User Manager** has been developed from scratch, using Java technology. When a new user is created in the eVRE system user manager stores the User Profile in its own repository and creates a reference to the record, the reference is stored in the Metadata Manager. An important choice here

has been to introduce a *synchronous* interaction between this component and the AAAI: indeed the User Manager needs to interact with the AAAI service in order to execute two crucial operations: i) store credentials of users created in eVRE and ii) check credentials validity when a user logs in.
Both interactions could be implemented using the asynchronous communication model adopted in our system, however, in order to improve security we decided to remove any mediation in these operations and implement for them a synchronous encrypted communication: the User Manager contacts the AAAI using an encrypted channel and waits until an answer has been received. The Figure 25 shows the UML sequence diagram for the authentication operation.

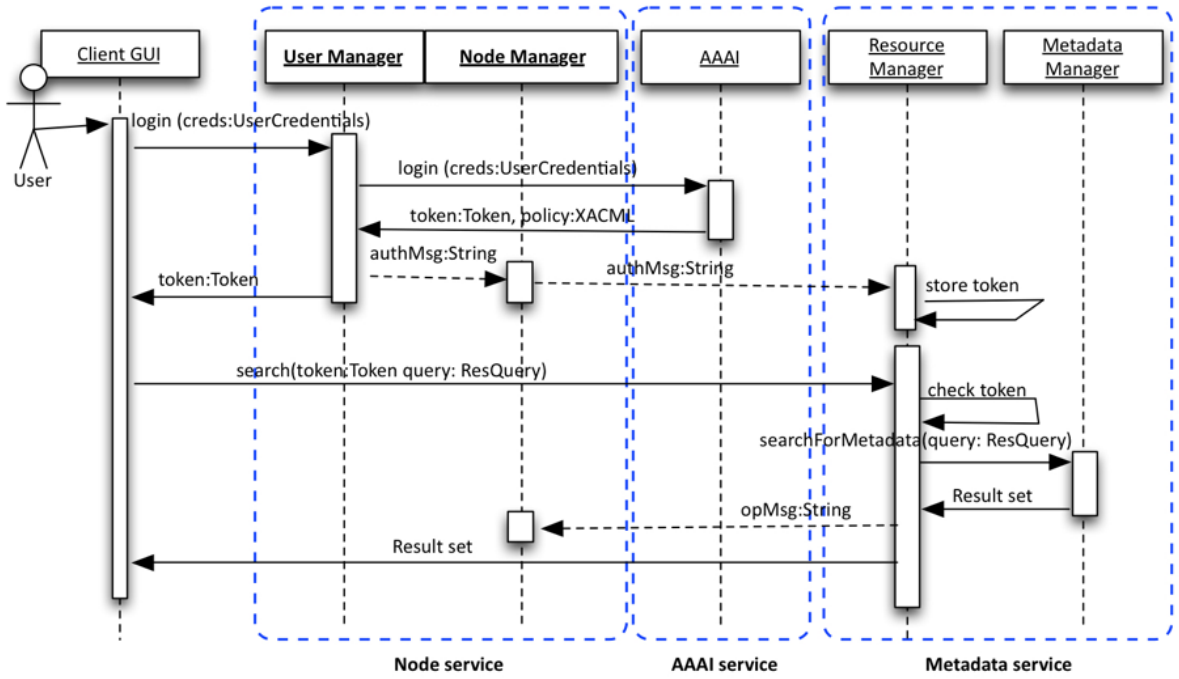


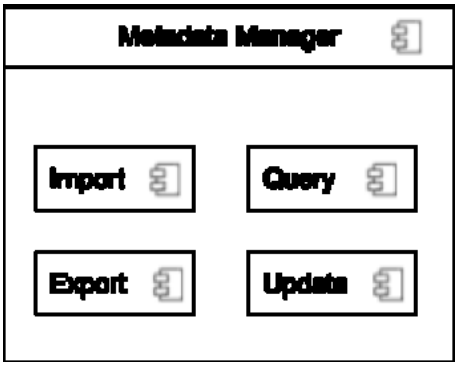
Figure 25 Sequence Diagram for the Authentication Use Case

The **eVRE WS** is composed of a set of restful Web Services. As documentation for the eVRE WS, we have created a publicly available application programming interface (OpenAPI) that provides developers with programmatic access to the eVRE WS entry points; it can be found at the following link:

http://v4e-lab.isti.cnr.it:8080/NodeService/swagger-ui.html#/

4.1.3 Metadata Service technological choices

The **Metadata Manager** implements all the methods that can be used for the metadata management namely, import, export, query and update. Moreover, it contains the required methods for the mapping and transformation of eVRE data provided by pilots into the CERIF data model expressed in RDF. Taking into account that said data are expressed in RDF, it is necessary to use an RDF Triplestore for the exploitation of the corresponding metadata catalogues.



The **eVRE WS** are documented via OpenAPI at the following link:
https://app.swaggerhub.com/apis/rousakis/Id-services/1.0.0

The **Resource Manager** is responsible for exposing various functionalities about the resources of the research infrastructure. More specifically it provides the means to retrieve, update and remove resources in the infrastructure. The component does not have any prerequisites in terms of persistent storage, and only relies on other components and APIs within the infrastructure to carry out the aforementioned tasks. To this end, it communicates (both synchronously and asynchronously) with other components like the metadata manager for storing and retrieving information about resources the AAAI component for checking permissions, etc. By design the resource manager is not meant to interact with public users, therefore it will be delivered as an API, that supports the corresponding functionalities as part of the metadata service component.

The **Data Model Mapper** has been built using the 3M set of tools that consist of 3M editor[3], the web application which is the GUI for the definition of the schema mappings, 3M manager[4] that exploits an eXist database and x3ml engine[5] which is the services that executes the transformation of the data.
This set of tools, which are implemented mainly using JAVA, is available in GitHub, and the whole 3M framework can be installed in a linux machine by using the Linux Installer that is currently published[6].

The **graphical user interface** is provided through the "VRE4EIC Metadata Portal". This portal aims at capturing all user requirements through a generic use case scenario by offering an easy to use UI environment, allowing end users to take advantage of the Node and Metadata Service.

4.1.4 AAAI standards and technologies

The AAAI consists of three components, the identity provider, the access control unit and the policy information point (PIP). The outlined design is standard practice. The interfaces between the components are standardised.
Federated identity management is realised by three major protocols: OpenID, oauth2 and SAML. The eVRE identity provider is implemented with an instance of Unity, it maintains its own user and user profile database. In general though it will act as a broker to other identity providers, such as the user's affiliation EDUGain provider. The user (profile) metadata is maintained in a CERIF database. The AAAI component, however, must maintain the mapping from the identity it has for a specific user to the CERIF person identifier.
As described in a previous section (The Node Service implementation), we choose to implement a synchronous communication: for user authentication the User Manager interacts with AAAI using a

[3] https://github.com/isl/3MEditor
[4] https://github.com/isl/Mapping-Memory-Manager
[5] https://github.com/isl/x3ml
[6] https://www.dropbox.com/s/566q2dhmbzbk662/3M_installer-1.1.tar.gz?dl=0

synchronous, encrypted channel. The *token* obtained by the User Client must be used in every interaction with eVRE building blocks for instance when executing queries on the catalogue via the Metadata Service.

5 The Canonical Reference Prototype

The following sections describe in details the overall approach followed to implement the Canonical Reference Prototype, the technologies and standards adopted in each case and the main instructions to install and run the building blocks[7].

5.1 The choreography approach in eVRE: defining events and messages

As described in Deliverable 3.3, the analysis of non-functional requirements has lead us to adopt *distribution of control* avoiding a central point of control in eVRE. Our choice therefore went for the Choreography approach [D 33].
To implement this approach, an *event-driven* communication model has been adopted for interactions of building blocks. The first step for the implementation of this principle is the definition of the set of *events* that occurs in a system. According to [RUSS] "Events represent full, complete, self-describing and immutable *Facts* about the system", and when creating a microservices architecture it is important to clearly define "which events should a service process" and "which events will a service emit".

For every eVRE microservice (building block) we have individuated the set of events it can emit by considering the list of the Generalized Functions (GF) extracted from the use cases [D 31], every completed GF produce a specific event emitted by the building block that has executed it; consequently, the building blocks that process the event will be those building blocks that could be affected by the GF executed. Since we tried to implement the *maintain the distributed process management in a single service* principle by design [D 33], most of the events don't have any side

[7] The complete installation guide for the e-VRE canonical prototype will be published on the VRE4EIC site and in the deliverable of Work Package 7.

effect to other building blocks and are emitted just to be processed for log reasons. However there are some special events that need to be processed by other building blocks: the most important of this events is the event generated by Fun21: Agent Authentication that must be processed by every other building blocks to guarantee the implementation of the Fun22: Continuous Access.
Events are communicated by microservices exchanging *messages*. To guarantee modularity, a microservice should not require any additional context, or dependencies on the in-memory session state to process a message representing an event; it must process the message and reacts accordingly if needed, whiteout further information. The messages defined in eVRE will be described in the section related to Node Service.
The event driven model in eVRE is implemented using asynchronous communication. An asynchronous communication interaction is not blocking (the microservice initiating the communication does not wait for answer) and is highly decoupled (a producer of a message does not know who is going to react to its message). From an architectural point of view an event-driven interaction model reduces communication latency and improves scalability and flexibility of eVRE (see Requirements 1, 2, 4 in the section "The VRE4EIC technical architecture"): for instance, new publishers or subscribers can be added to (or can be removed from) processing an event message without other Microservices need to know it. In eVRE architecture the stream of events uses a Communication Bus that is implemented as a set of asynchronous communication channels, one for every category of events, managed by a specific component, the bus manager checks the producers/consumers credentials to permit a building block to execute produce/consume operations.

5.1.1 The development environment

Typically, geographically distributed teams whose goals are to design and develop a large Information System need several kinds of collaborative tools to manage code versioning, to track development activities and to distribute and deploy releases of the system.
We decided to use Git as Version Control System for the documents produced during the architecture design, and for the source code produced during the implementation of eVRE.
The Jenkins framework (https://jenkins-ci.org) is used as Continuous Integration framework. Jenkins is a server-based system running in Apache Tomcat (and other servlet containers) and it is installed on a server hosted at ISTI-CNR:
http://v4e-hub.isti.cnr.it

The source code can be downloaded from the VRE4EIC repository defined on GitHub:
https://github.com/vre4eic.

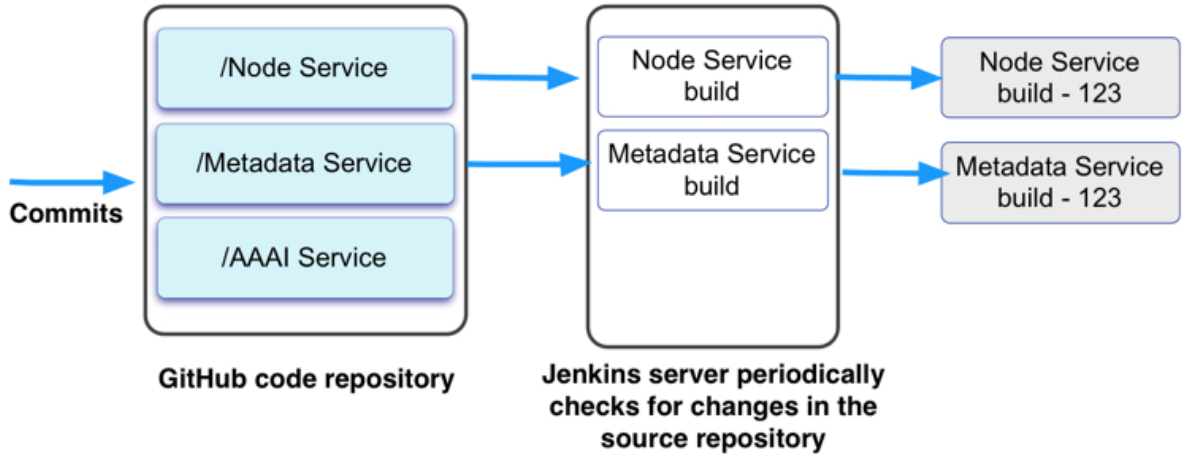


Figure 26 Development Environment

To distribute the eVRE system we decided to adopt a container-based virtualization approach. We are currently using Docker [Docker] as framework for this task.

5.2 The Node Service in the CRP

The Node Service implements all functionalities related to the User Profile management and eVRE system administration. The UML component diagram is shown in section 3.2.

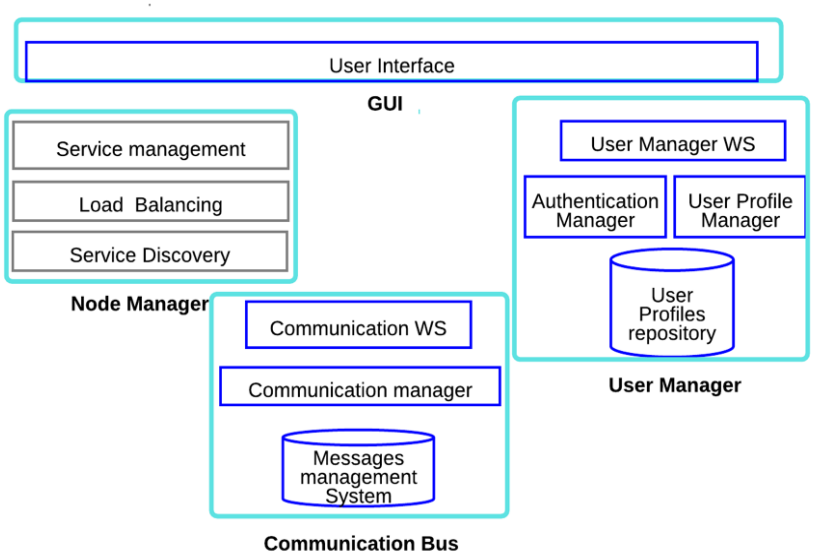


Figure 27 The Node Service: a technical view

The eVRE Node Service functionalities are implemented by 4 main software components:

- a GUI that enables the administrator to monitor the system and define the configuration
- the User Manager that manages user profiles and provide authentication facilities
- the Communication Bus that is used as communication infrastructure for the eVRE Microservices interaction.
- the Node Manager which implements the functionalities to manage the eVRE system

5.2.1 The Node Manager

In a microservice architecture, services are typically distributed in a server infrastructure (created using containers and VM images). In such an infrastructure microservices may scale up and down based upon certain predefined conditions and the address of a microservice may not be known until the service is deployed and ready to be used. The Node Manager implements the functionalities to manage a distributed configuration service, a synchronization service, and a naming registry.

Essentially, the Node Manager role is to enable the set of autonomous eVRE microservices to act as a coherent single system.

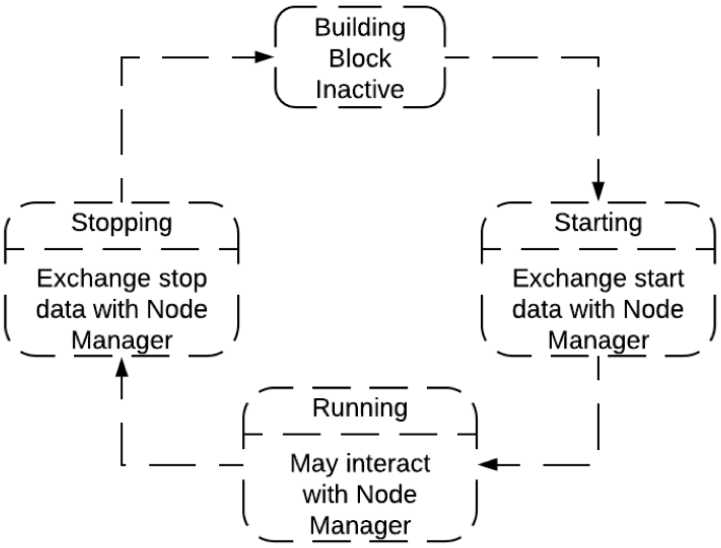


Figure 28 Partial state diagram of an eVRE building block life-cycle

Each eVRE *building block*, knows the address of the Node Service and during the start phase of its life-cycle registers with the Node Manager component. During the registration, the building block:

1. provides information about itself, such as the endpoint address.
2. gets information it may need to execute its business logic: the address of other building blocks, credentials to access remote resources, certificates to implement encrypted communications etc

At runtime, a building block can interact with the Node Manager, for instance to find out the location of other eVRE building blocks or to communicate significant changes in its state. When the building block stops, it communicates with Node Manager to inform that it will be no longer available.
The Node Manager stores locally the information about the status of the eVRE system and implements a policy for load balancing.
According to our architecture design, an *eVRE system* is made by the set of running building blocks coordinated by a specific Node Manager.

5.2.2 The User Manager

The *User Manager* building block implements the management of User profiles containing information about the users that registers on eVRE and *wraps* the authentication functionalities of the AAAI building block.

Figure 29 The User Manager class

The functionalities of the User Manager building block are mainly used by external agents to register/authenticate in the eVRE systems.

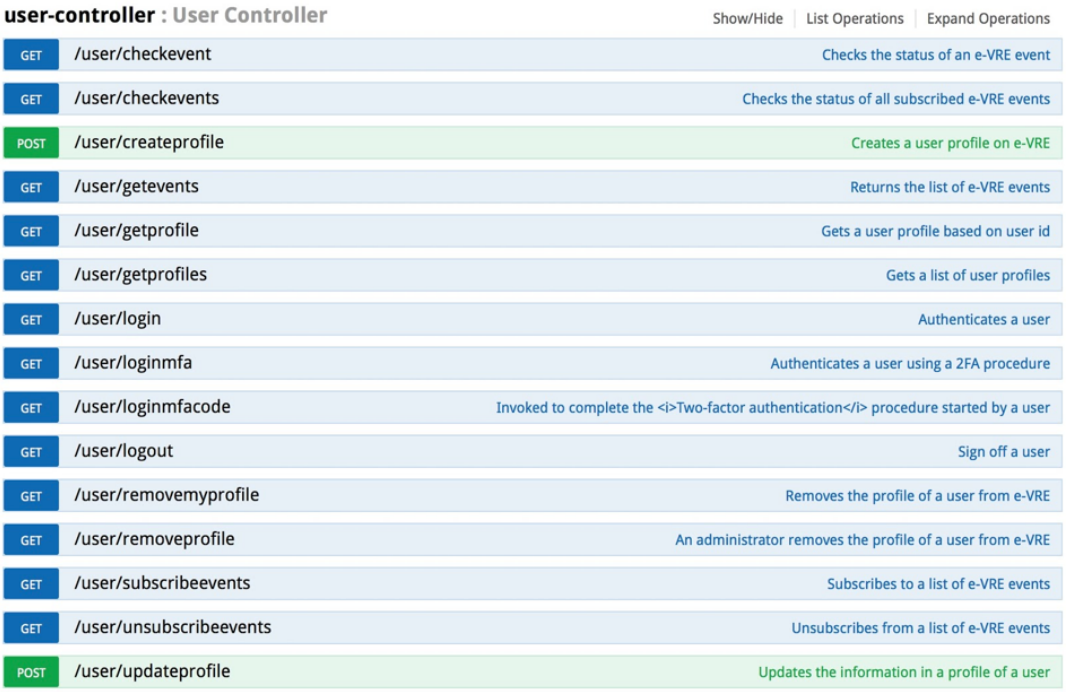


Figure 30 eVRE WS entry points for eVRE User Manager

5.2.3 The Communication bus

The Communication Bus is responsible for managing the asynchronous interactions implementing the event driven model in eVRE. It acts as a Message Oriented Middleware (MOM) and provides to eVRE building blocks functionalities to exchange messages.

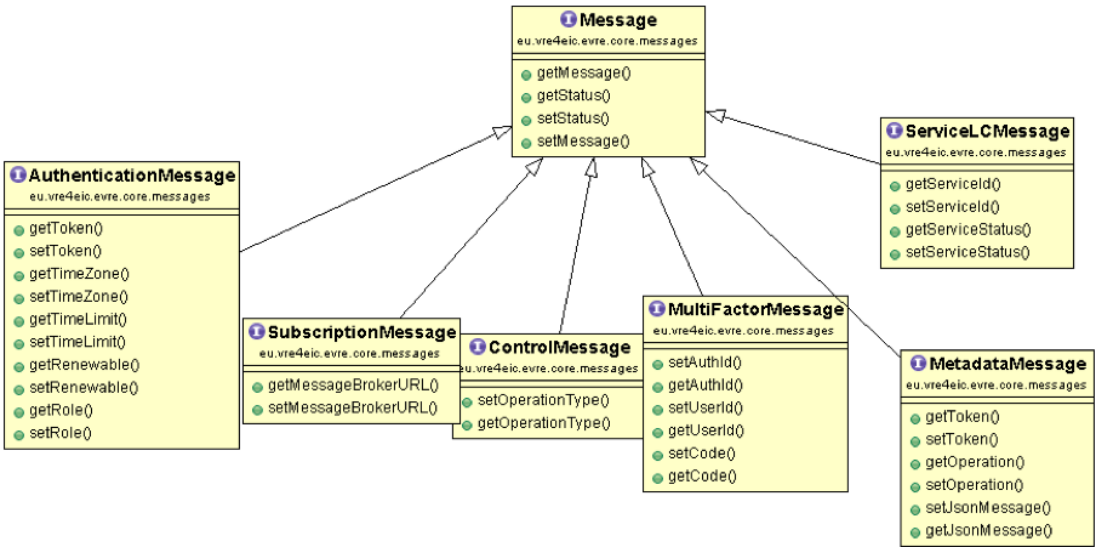


Figure 31 Class Diagram of messages

Messages describe events that are relevant for the implementation of the business logic of the system, the image above shows a subset of the messages implemented in the canonical prototype.
Following the MOM principles, the eVRE Communication Bus is used to create a number of communication channels, each one containing a specific kind of messages.
When a building block starts, it gets information about the system configuration from the Node Manager, including the list of active communication channels. It may *subscribe* to one or more channels, and may also ask the Communication Bus to create some channels. During its life-cyde the building block will be able to produce and consume (according to its permissions) messages on the channels it has subscribed.

5.2.4 Implementation choices and adopted technologies for the Node Service

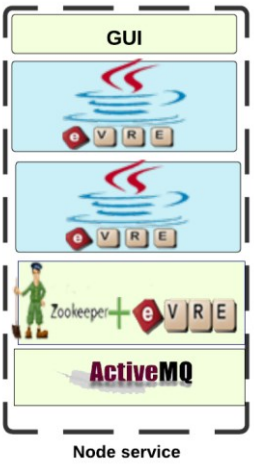


Figure 32adopted for the Node Service The main technologies

The *eVRE Web Services* component has been implemented in Java, the entry points of this component are described and documented in details in the prototype site.
The same approach has been adopted to develop the *User Manager* component, a Java library has been built using MongoDB as persistence layer.
To implement the *Node Manager* in the eVRE prototype the Apache ZooKeeper framework has been used. Zookeeper provides functionalities to maintains status type information in memory and keeps a copy of the state of the entire system and persists this information in local log files. In eVRE, every building block creates a znode (a file that persists in memory on the Node Manager server). The znode can be updated by building blocks that have permissions to do it, and any other building blocks in the eVRE can register with the Node Manager to be informed of changes to that znode (i.e. to "watch" a specific znode).
In order to help developers to implement the logic described for an eVRE building block we have created an helper class (a sort of Node Manager client), called NodeLinker class. The current release of this class is shown in Figure 33:

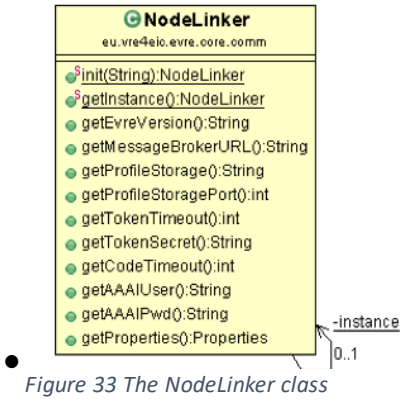


Figure 33 The NodeLinker class

The Node Linker class needs to be initialized with the address of the Node Manager and the credentials, when instantiated it automatically sends to the Node Manager the address and the name of the building block and downloads a number of properties that can be used by the building block.
This class will be upgraded in the next release to include also the exchange of security certificates.
Finally, the *Communication Bus* in the eVRE prototype is based on Apache ActiveMQ, a framework implementing MOM principles; it is used as communication layer by a Java API developed by VRE4EIC team. In particular the code developed for Communication Bus add to ActiveMQ a security layer to encrypt and digitally sign messages exchanged, details of this security layer have been reported in the Deliverable 5.4 of the VRE4EIC project and will be also explained in the section related to AAAI building block.

5.2.5 Source Code, documentation, set up

The java code is published on GitHub, and can be downloaded at the following URL:

https://github.com/vre4eic/NodeService

The Node Service has been developed as a Java Maven project, the code is separate in two main packages:

- *eu.vre4eic.eVRE.nodeservice* where there is the code that implements the functionalities of the Node Service building block.

- *eu.vre4eic.eVRE.core* where we implement the code implementing functionalities commons to all building blocks. In particular this package contains the Java classes implementing messages and the classes implementing Node Manager clients. This development choice has been taken in order to create a common API that can be used by all building blocks when interacting each others or with the infrastructure. These API are distributed as Java archive (jar)

The java classes are documented using Javadoc, the complete documentation can be read here:
http://v4e-lab.isti.cnr.it:8080/NodeService/doc/index.html
For the documentation of eVRE Web Services we adopted Swagger, a software framework that enables developer to describe the API. The swagger documentation of the eVRE Web Services is here:

http://v4e-lab.isti.cnr.it:8080/NodeService/swagger-ui.html#

To set up the Node Service MongoDB and ActiveMQ are required in your environment.
At the moment this deliverable the only way to install a Node Service is to clone or download the source code, then manually change property values in the file:
[your_dir]/NodeManager/src/main/resources/Settings.properties
In particular the MongoDB and ActiveMQ address and credentials must be set. When the file *Settings.properties* has been updated a Web ARchive (WAR) must be created using maven and deployed on a application container.

5.3 The eVRE AAAI implementation in the CRP

To implement the functionalities related to authorization/authentication in eVRE prototype we have delegate to the User Manager the role of partial wrapper of the AAAI functionalities; this implementation pattern enables eVRE prototype to be independent from the framework used to implement the AAAI. In particular the User Manager wraps the interactions between the AAAI framework and the external agents to execute two crucial operations: i) store credentials of user profiles created in eVRE prototype and ii) check credentials validity when a user logs in the system. As explained in Deliverable 3.3, in order to improve security, we decided to implement here a synchronous communication: the User Manager interacts with AAAI using a synchronous, encrypted channel. Essentially, when an external agent authenticates using the User Manager (Figure 34):

1. The client sends the credentials using a eVRE WS entry point
2. the User Manager forwards the credentials to AAAI and waits for answer
3. the AAAI verify that credentials are valid and returns a *token*
4. the User Manager creates an eVRE Authentication Message (explained in next sections of this deliverable) that contains the token and sends this message asynchronously via the Communication Bus
5. the User manager sends an answer to the client containing the token.

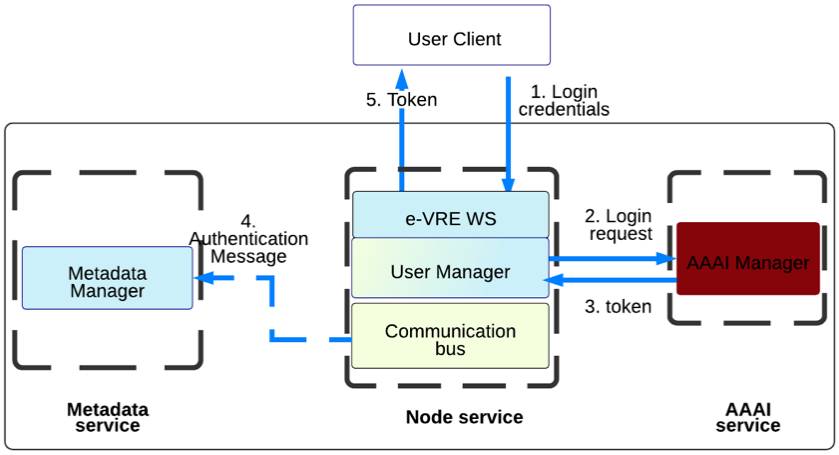


Figure 34 User authentication

The client will use the token in every interaction with eVRE building blocks for instance when executing queries on the catalogue via the Metadata Service. The Figure 34 below shows the UML *sequence diagram* describing this use case.

The main security/trust issue in this use case is that the Metadata Manager needs to know that the *token* it receives asynchronously has been created by an eVRE building block that has the authority of creating it, and that it has not been tampered with. The solution adopted to solve this issue has been to sign and encrypt tokens and messages exchanged by eVRE building blocks. In the use case the AAAI creates a token, encrypts it, and then returns it to the Node Service (as explained this happens on a secure encrypted channel), the Node Service creates an *AuthenticationMessage*, signs it and send the message asynchronously to eVRE building blocks. When the Metadata Service receives the token: checks the signature, if it is a valid signature, decrypts the token and stores it locally.
Every eVRE building block signs messages before publishing them in the Communication Bus channels and subscribers validate messages signature before consuming them.

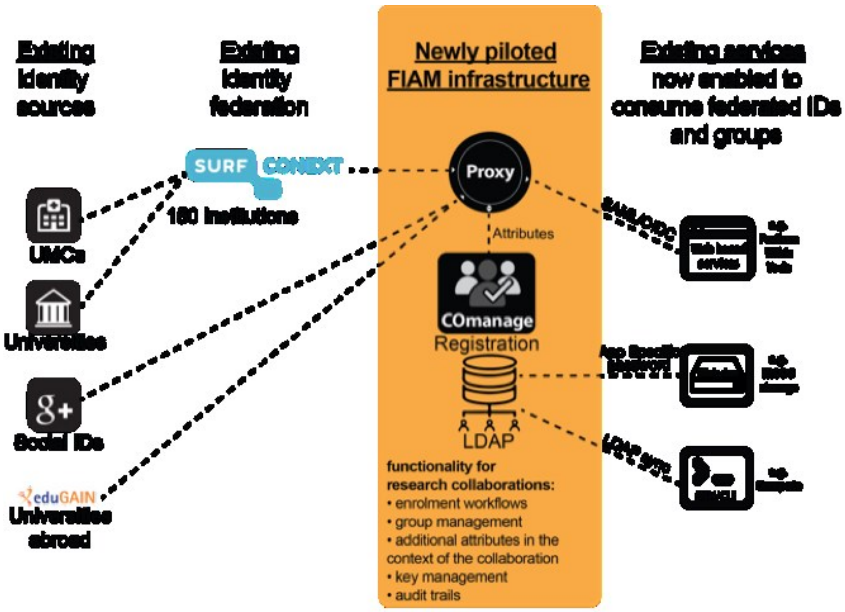
To implement token encryptions and message signing, we have used the JSON Web Token (JWT) standard8. In the eVRE prototype, JWT encryption and signature are based on shared private keys, exchanged by building blocks via Node Manager, a more secure (and efficient) Public Key Infrastructure will be implemented in next release.

5.3.1 Security and trust components

Some key issues discussed in Deliverable D5.3 and D5.4 around Security and Trust require solutions that are either too domain- or platform-specific, or insufficiently standardized to be incorporated in the reference architecture or the canonical reference prototype. As a use case in dealing with security and trust issues in a digital humanities use case, we explored user interface components to support a collaborative, explorative and interactive web front-end that can still produce transparent and reproducible results on (privacy) sensitive data. These components are also used to gain practical experience in connecting eVREs to other (inter)federated authorization infrastructures that currently

[8] The JWT is "an open standard [...] that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA". Info: https://jwt.io/introduction/

under development by the VRE4EIC project participating in a pilot[9] managed by the Dutch National Research and Education Network (NREN) organization SURFnet.



The VRE4EIC project participation in the pilot is to gain practical experiences with existing international standards for federated authentication (e.g. identity management) and (inter) federations such as eduGAIN, and to explore the landscape around federated authorization that is currently developing in projects such as AARC2. We collaborate since May 2018 in a specific pilot by SURFnet where several services are bundled in a virtual Science Collaboration Zone, and are particularly interested in the federated attribute management functionality. It allows us to explore the distributed management of user attributes stored at the Identity Provider, the federated infrastructure and/or the eVRE AAAI service. The Pilot has just started and a first working prototype has been demonstrated at an internal project meeting. We will fully report on this in deliverable D3.5

5.3.1.1 Collaborative executable notebook for transparent data science

To demonstrate trust-specific components a public demonstrator has been developed and made available as a web service. This web service demonstrates the use of the SWISH datalab as a prototype of a transparent and collaborative executable notebook user interface for e-RIs and eVREs platforms. The demo contains all the code necessary to reproduce the tables, figures and other computational results of an open access Web Science 2018 paper: “Using the Web of Data to Study Gender Differences in Online Knowledge Sources: the Case of the European Parliament”. It uses an collaborative notebook interfaces integrating the complete statistical analysis code (in R) for all results presented in the paper, and the complete declarative data pre-processing and data modeling code (in Prolog). This means that all computational steps leading to the resulted on in the paper are open for inspection and review. In addition, it offers permalinks for all interactive results obtained from the system, which means that all final and intermediary steps can be downloaded in the future, even if the underlying code in the collaborative notebook is changed by the researcher or her colleagues.

5.3.1.2 Technical details of the release

All software and code needed to reproduce the public web demonstrator has been archived for long term storage at Zenodo under DOI https://doi.org/10.5281/zenodo.1232929. To project this code against dependencies on operating-specific details and changes in third-party software, a self-

[9] https://wiki.surfnet.nl/display/SCZ/Pilot+partners

contained virtual machine image has been published for long term storage at https://doi.org/10.5281/zenodo.1237673. All source code of the software is available on the VRE4EIC GitHub account (https://github.com/vre4eic/websci2018-reproducibility-pack) and as docker containers (https://hub.docker.com/u/vre4eic/).

5.3.2 The Two-Factor Authentication (2FA) CRP

The eVRE CRP User Manager implements a two-factor authentication (2FA) method i.e. a method using a combination of two different factors to authenticate a user. The following sequence diagram shows the building blocks involved to implement this functionality.

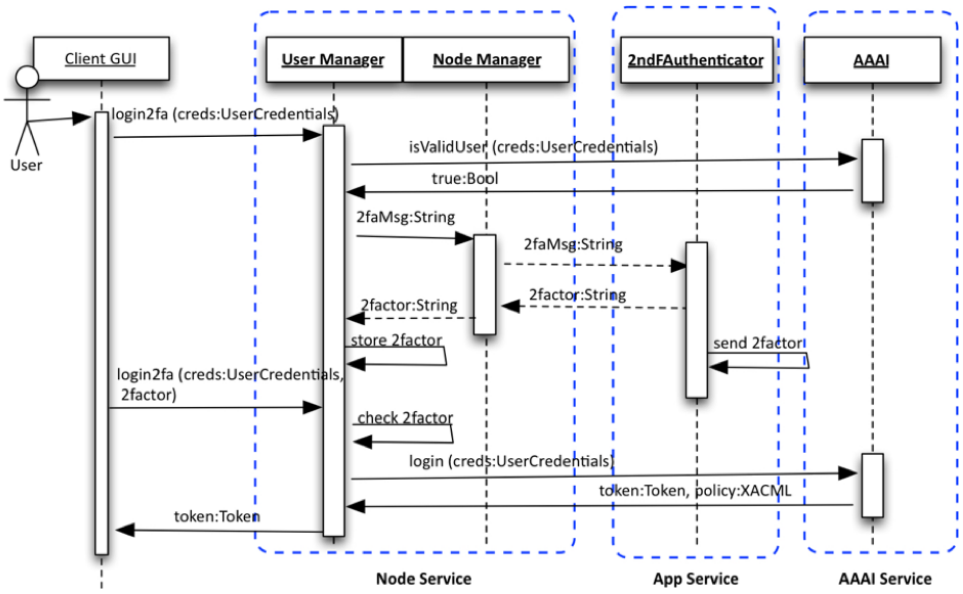


Figure 35 2FA Use Case Sequence Diagram

In eVRE prototype the channel used to communicate to the user the second *factor* is a Telegram Channel, a detailed description of how a user can use the 2FA feature to authenticate on eVRE prototype is in VRE4EIC Deliverable 3.3 [D 33], in particular in sections 6.1 and 5.4.
The component integrating the telegram Framework is called *eVRE-TGBotAuthenticator*, the source code of this component is available here:
https://github.com/vre4eic/TelegramBots

5.4 Implementing The Metadata Manager in the CRP

It is the eVRE building block responsible for storing and managing resource catalogues. These functionalities are provided by exploiting various components (i.e. MetadataManager, DataModelMapper, etc.).

5.4.1 Implementation choices and technologies adopted

During the design phase of the project it has been decided that this building block should have used a triple-store as repository. Initially, Blazegraph[10] was installed as the Metadata Service repository. However, during the eVRE prototype development, it was noticed that in many cases Blazegraph had performance issues. In addition to that, Blazegraph's development itself, seems to be inactive for the last couple years. For those reasons it was decided to seek for an alternative triple-store, ending up

[10] https://www.blazegraph.com

with adopting the Virtuoso Universal Server[11]. The Virtuoso Universal Server merges the capabilities offered by a hybrid database engine and by a middleware that combines the functionality of a traditional RDBMS, ORDBMS, RDFStore, virtual database, web application server and file server. It is in fact a single threaded server process that supports multiple protocols.
The following standard Web and Internet protocols have been implemented in Virtuoso: HTTP, HTTPs, WebDav, SOAP, UDDI, SPARQL and SPARUL. In addition, concerning the development of database-based applications and the integration of systems, Virtuoso has implemented a wide variety of industry standard data access APIs, such as ODBC, JDBC, OLE DB and ADO .NET. Virtuoso is made up of various server and client components, which enable the communication of a local or remote Virtuoso server, such as the Conductor, which is Web based Database Administration User Interface, and ISQL and ISQLO utilities. To this end, the Virtuoso Universal Server can be exploited to produce a clustered-based system of RDFStores. In addition, there is a specific version of this server, which can be deployed on the Amazon Cloud. Concerning the specific characteristics of Virtuoso that are of interest for eVRE prototype, it can be highlighted that Virtuoso not only allows the management of RDF (linked-)data but it enables their querying through the SPARQL language. The same language (actually SPARUL) can be exploited for the updating of the linked-data.

5.4.2 Metadata Service: the source code

MetadataService has been developed as a JAVA maven project. It consists of different components, therefore it is important to separate the development spaces for these components. All the different components are placed under the eu.vre4eic.eVRE package. After that follows another package that groups together the resources of a particular component (i.e. eu.vre4eic.eVRE.metadatamanager contains all the resources of the corresponding component).
For each component we use different packages for grouping together resources that are used for a particular reason. More specifically we use the following packages:

- api: for adding the interfaces of the components. These are the contractual interfaces that are also visible from other components.
- impl: for adding the actual implementation (or different implementations) of the interfaces that are found under the api package.
- model: for adding the structural components that are required (i.e. POJOs)
- exceptions: for adding the corresponding exceptions-related resources
- test: for adding the resources that are needed for testing the components

Furthermore for grouping the resources that are exploited throughout all the components we have created a commons package (found under eu.vre4eic.eVRE.commons)

The Javadoc of the code developed can be found here:
http://139.91.183.70/apidocs/
The swagger documentation of the eVRE Web Services is here:

https://app.swaggerhub.com/apis/rousakis/ld-services/1.0.0

5.5 The Workflow Service in the CRP

In the eVRE Reference Architecture, the Workflow Manager (WM) component is responsible for the management of business processes and scientific workflows. To do this, the WM interacts with other

[11] https://virtuoso.openlinksw.com/universal-server/

components, for instance it uses the Metadata Manager to get information to build workflows and to store them, and the Query Manager to execute distributed queries. In the technical architecture, the functionalities of the Workflow Manager are implemented by the *Workflow Service*. These functionalities have been partitioned in three groups, each one implemented by a different sub-component:

- The Workflow Configurator provides functionalities to build/edit/store workflows, and to control and monitor their executions
- The Workflow Executor manages the execution of workflows, including data staging, it may interact with the App Manager to execute tasks.
- The Workflow Repository provides functionalities to store and retrieve workflows descriptions to/from the Metadata Manager, and to publish them using the LD Manager.

5.5.1 The Workflow Configurator

The Workflow Configurator implemented in the Canonical prototype access the VRE4EIC catalogue to:

- search for resources that may be used in building a Workflow, for instance Web services descriptions or references to external datasets, etc.
- save descriptions of web services, so these become eVRE resources and can be searched and reused

The UML sequence diagram in Figure 36, describes the interactions between eVRE components in a use case where the Workflow Configurator search for WADL descriptions of Web Services.

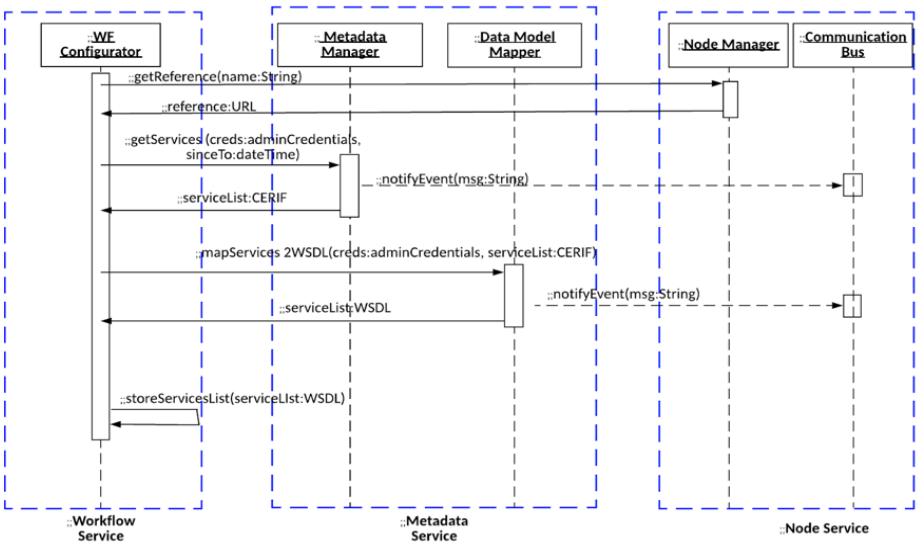


Figure 36 Search WADL resources Use Case

The Workflow Configurator interacts directly with the Metadata Manager (via the Metadata Service eVRE WS, not shown in the diagram), and to do this it needs to know the reference (the URL) of the Metadata Service. As explained in the section describing the Node Service, the Workflow Service obtains the reference of the Metadata Service by querying the Node Manager during its start-up phase. However, the reference to Metadata Service can be obtained also at runtime (the Node

Manager may implement a policy for load balancing when more Metadata Services are part of an eVRE system).

5.5.2 The GUI, the Workflow Executor, the Workflow Repository

The Workflow Executor and the Workflow Repository components of the Workflow Service must implement a number of user requirements, as described in Deliverable 3.1, in particular the requirements: PRQ28 Data Processing Control, ORQ2 Processing Parallelization, ORQ4 Data Compartmentalization [D 31]. The GUI should provide to users the possibilities of create workflows, save them in the Workflow Repository and execute workflows in the Workflow Executor.
The effort to obtain a complete implementation of these components exceeds the resources available in the project. For this reason, to implement the required functionalities we decided to investigate the possibility to integrate an existing workflow engine in eVRE.
This starting point of our investigation has been a detailed analysis [HOLL] on state of art in the field of workflow engines. The following table summarizes results of the work:

	Kepler	Triana	Wings/ Pegasus	Taverna	Pipeline Pilot	KNIME
Life Science application support	++	++	+	+++	++	+++
GUI	+++	+++	+++	+++	+++	+++
Heterogeneous resources	+++	++	++	+++	++	+
Provenance & Sharing	++	++	++	+++	++	+
Scalability	+++	++	+++	++	++	++
Extensible	+++	++	++	+++	-	+
Security	++	+	+	++	+	+
Active development	+++	++	+++	+++	+++	+++

Table 3.1: Evaluation of common scientific workflow management systems. Legend: '-' means not supported, '+++' means fully supported.

A number of tests has been made on the above frameworks and two possible choices have been considered: Kepler and Apache Taverna. The main features of these two frameworks are compared in Table 3.

Table 3 Workflow Engines features comparison

Apache Taverna	Kepler
Domain-independent	Domain-independent
GUI for workflow composition	GUI for workflow composition
Simple Conceptual Unified Flow Language (SCUFL2)	Uniform access to computational components through actor model.
Enable to share and manipulate workflows outside the editor.	Workflows are saved as XML files

Dataflow-oriented model of execution and support loops	Many different models of computation are possible, focus on actor-oriented
Support for RESTful web services & OGC service consumption	
Support for the use of Cloud in workflow execution	Support for the use of Cloud in workflow execution
https://taverna.incubator.apache.org	https://kepler-project.org/
LGPL license	BSD License

We decided to use Apache Taverna for the prototype implementation. From the implementation point of view the main reasons for this choice have been that it enables developers to implement the web services and local services integration using Java related technologies; additionally it provides monitoring tools that can be easily integrated in eVRE GUI, in particular the Taverna Workbench. The Taverna Workbench, a tool that enable users to create, manage and store workflows, has been designed and developed as a plug-in platform, this means that its functionalities can be easily extended by developing and installing new plug-ins. A specific plugin for eVRE has been developed, it enables Taverna Workbench to interact with eVRE, details of the eVRE plug-in are described in the following sections.

5.5.3 Implementation choices

This section describes the implementation of the Workflow Service in terms of the three main actions of the *Fun15: Workflow Enactment* general function that is implemented by the Workflow Service [D 31].

5.5.3.1 Workflow creation

The workflow is created using the Taverna Workbench which access the VRE4EIC catalogue to get Web Services descriptions and use these descriptions to create workflows. The main responsible for this activity in eVRE is the Workflow Configurator: it interacts with Metadata Manager to get the Web Services descriptions, with AAAI to check authentication/authorization, with the Data Model Mapper to implement the transformation of CERIF records into a format accepted by Taverna.
To enable the Taverna Workbench to interact with the eVRE we have developed a plug in module that uses the eVRE WS provided by the Workflow Configurator. The plug-in is published and can be installed by any user using Taverna Workbench (release 2.5).

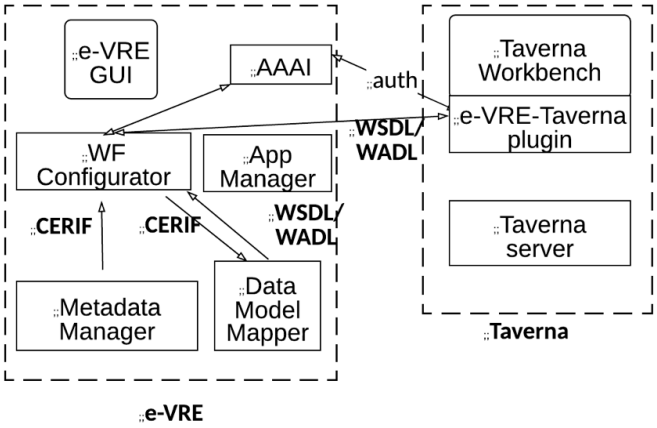


Figure 37 Workflow Creation

5.5.3.2 Workflow storage

When a workflow is created:

- It is stored in the local Apache Taverna repository,
- A WSDL document describing the workflow is created by eVRE-Taverna plugin.
- The WSDL document is consumed by the Workflow Configurator of eVRE, and passed onto the Data Model Mapper, which transforms it into a CERIF Service description. The so obtained description is stored in the Metadata Manager (see Figure 38).

From this point on, the workflow can be discovered and invoked as any other service whose description is stored in the eVRE Catalogue.

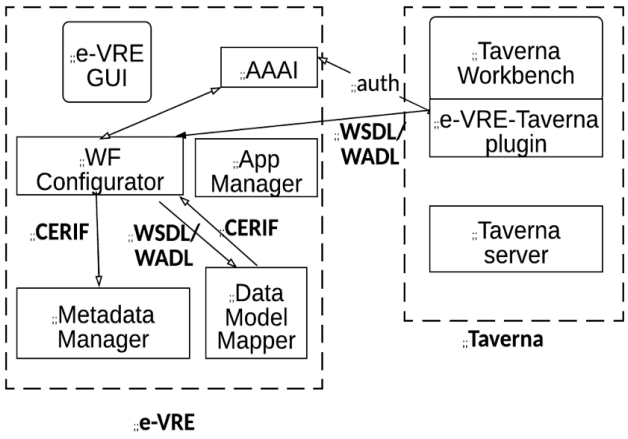


Figure 38 Workflow Storage

5.5.3.3 Workflow execution

The execution of a number of predefined workflows can be launched by the user from the eVRE GUI. The GUI captures the input parameters of the workflows (via an HTML form, or similar) and demands the execution to the App Manager of eVRE. The App Manager, in turn, will interact with a defined Taverna server to execute the workflow. The result of the workflow is returned by the Taverna server to the App Manager, and from the App Manager to the GUI.

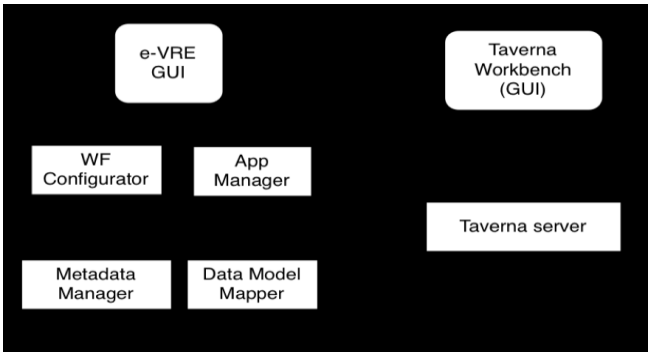


Figure 39 Workflow Execution

5.5.4 Source code, documentation and set up

The java code of the Workflow Service is published on GitHub, and can be downloaded at the following URL:

https://github.com/vre4eic/WorkflowService

The Workflow Service has been developed as a Java Maven project, at the moment it only contains the source code of the Workflow Configurator.
The documentation of eVRE Web Services is here:

http://v4e-lab.isti.cnr.it:8080/WorkflowService/swagger-ui.html#/

The eVRE-Taverna plugin source code is available here:

https://github.com/vre4eic/eVRETaverna

The Readme.md file on GitHub repository contains instructions for the set-up of this building block and of the eVRE Taverna plugin.

5.6 The App Service in CRP

The goal of this building block is to implement the functionalities of the App Manager conceptual component: enable external applications to be embedded and used into the eVRE system (see Deliverable 3.1 [D 31]). The App Manager is a crucial component for eVRE, it should be implemented as a lightweight, unobtrusive software module that interacts with external applications to track their lifecycle and their usage.
Creating a generic App Manager, able to automatically manage lifecycle and usage for every possible external application embedded in the eVRE, is not feasible: a plan has been defined in the design phase to consider the main standards for this purpose and to proceed implementing the App Service for these standards: the Servlet level 3 specification has been adopted as first candidate.
However, the eVRE canonical prototype embeds two external software frameworks with a different integration technologies: the Telegram framework for notifications and Two Factor Authentication (2FA) using the Telegram API and the Taverna Workflow Engine, for workflow management GUI and execution using the plug-in platform provided by Taverna Workbench.
As described in the correspondent sections we have created two specific components each one running inside the eVRE system and interacting with eVRE building blocks using the Communication Bus.

The main role of the App Manager in eVRE prototype is to consume messages sent by these components to keep a log of states transitions and to produce messages that are consumed by these two components, each one reacting according to its business logic.

6 The VRE4EIC Metadata Portal

This section describes the implementation of a GUI for the VRE4EIC Reference Architecture. In particular this GUI facilitates the exploration, discovery and management of the metadata describing resources contained in the VRE4EIC catalogue. It incorporates a multitude of features on top of an intuitive and user friendly environment, in order for both novice and expert users to execute complex queries. The platform is agnostic to the underlying conceptual model, yet it can be configured to take advantage of the main concepts designed.

6.1 Introduction to this section

The publishing of structured and semantically enriched data is changing traditional models of conducting business and research. Modern science in particular is becoming more collaborative and multidisciplinary, taking advantage of the plethora of data being produced by groups with diverse scientific backgrounds. So-called Virtual Research Environments (VREs) aim to promote this scheme, overcoming physical or semantic barriers, and facilitating researchers from diverse fields to exchange data and resources, decoupling science from ICT.
For such an interoperability to be achieved, and considering the heterogeneity in scope, features and technologies, various challenges are faced from the technical, semantic and legal standpoint. One major goal is the generation of high-level ontologies with rich metadata that are easily explored by researchers.

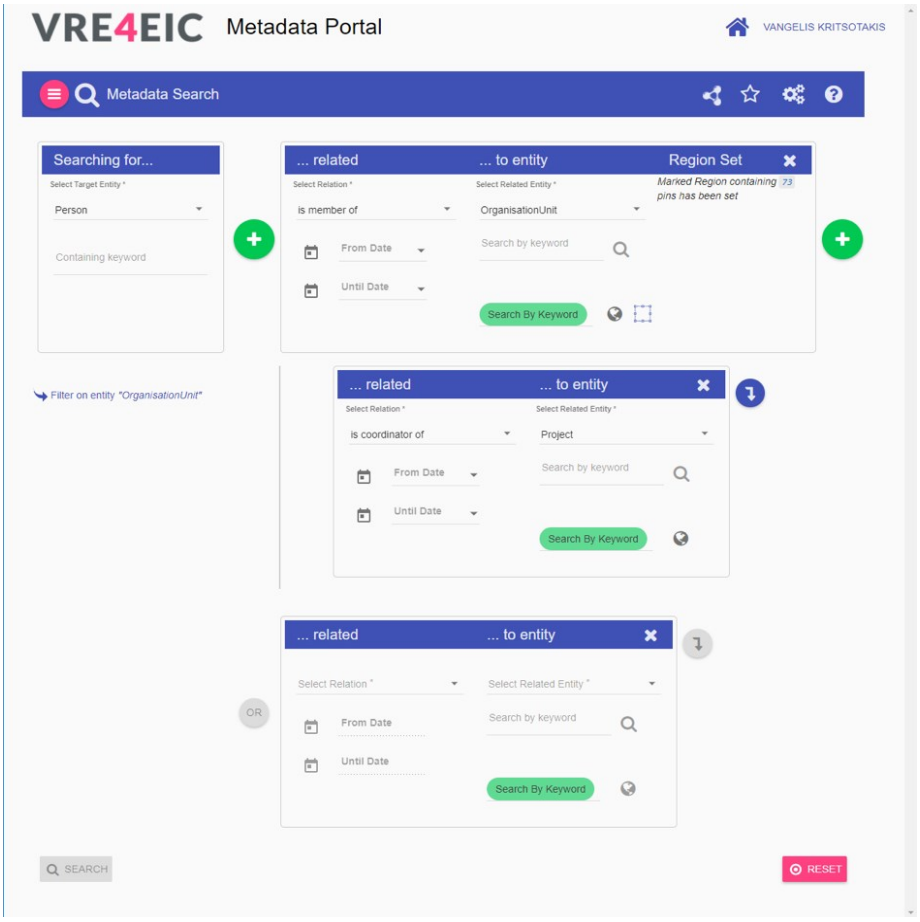
6.2 Targeted Objectives of the Design

This section describes the design, architecture and implementation of the VRE4EIC Metadata Portal, a fully functional platform that facilitates the exploration, discovery and management of semantic metadata for both novice and expert users who wish to execute complex queries. The platform, provides an intuitive, user friendly environment that is highly configurable, making it appropriate for various domains, still being agnostic of the underlying conceptual model.
The purpose of the VRE4EIC Metadata Portal is to provide a user-friendly environment to all VRE4EIC users for the search, management and import of Metadata, contained in certain VREs and RIs. This is achieved through the appropriate Graphical User Interface (GUI) through which end users can take advantage of the Node and Metadata Services.
The portal does not only aim at offering (another) simple query interface for providing access to the underlying metadata, hiding the complexity of writing expressive SPARQL queries. Although the look'n'feel resembles similar query building systems with the goal of reducing the learning curve for the novice user, the features incorporated are based on a thorough analysis of the requirements of existing VREs and Research Infrastructures (RIs), offering a repertoire of solutions for various beneficiaries. Overall, the goal is to support both

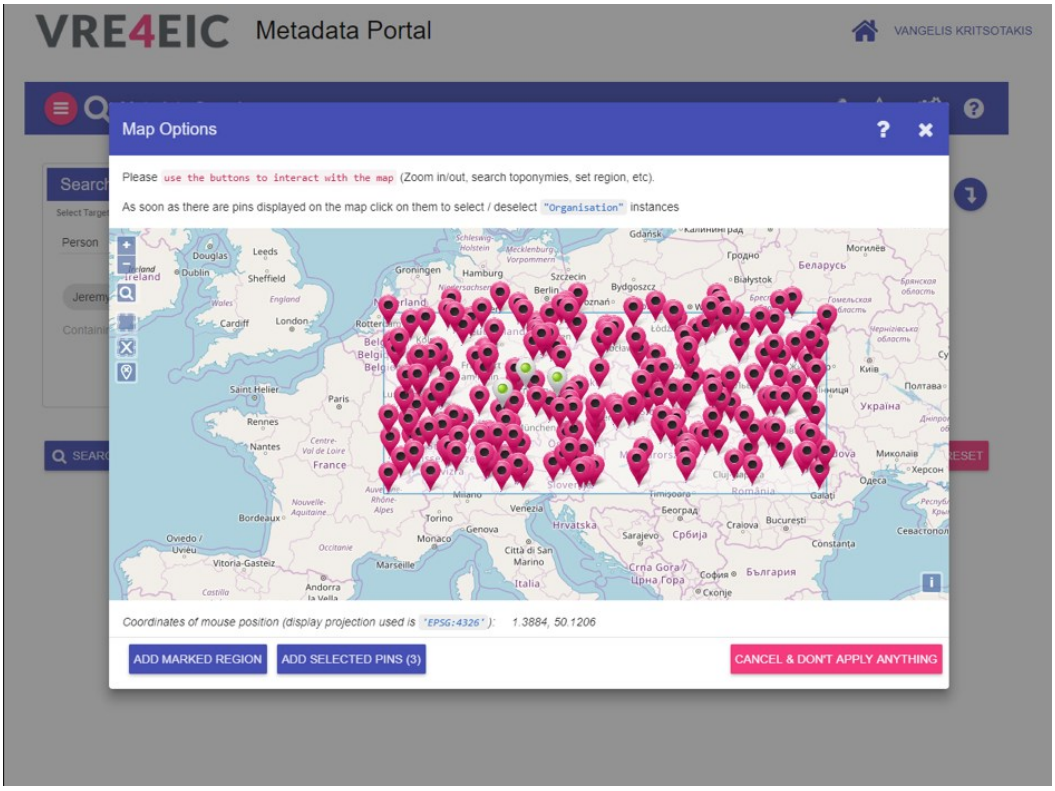
- Query writing for the expert, exploiting the capacity of the underlying conceptual models and gathering best practices from similar systems,
- Discovery of metadata (exploratory search) for the novice user, to help researchers search across domains and data that they are not familiar with, guiding them in the process of creating a query.

6.3 Functional Model

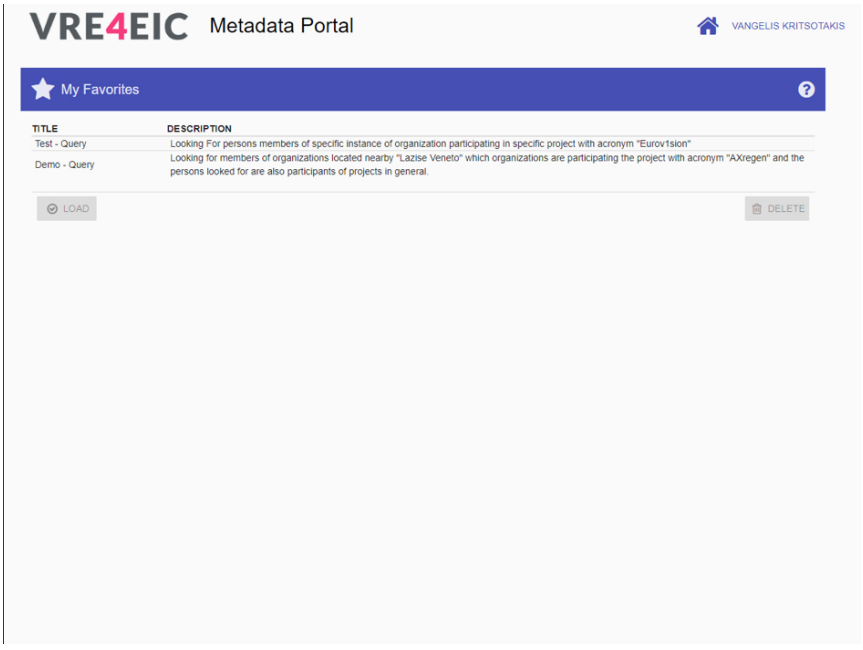
The current implementation offers a metadata catalogue containing metadata from affiliated RIs. In short, the main key features of the platform are described in the following paragraphs.
One key characteristic of the portal is that it dynamically executes sub-queries and presents partial results on-the-fly, while the user builds the query. The goal is twofold. On the one hand, it helps exploration by presenting results and allowing end users to limit the scope of the query and, on the other hand, it prevents from executing meaningless queries that return no results. This is achieved by transparently altering the options available during the query building process, eliminating choices that can lead to an empty search space.
The platform offers a combination of different ways for searching relevant metadata within the same query. It enables the user to compose queries using *keyword search*, *entity-based search*, *time range-queries*, *filter-based search* and *geo-spatial search* through an interactive map.
Filter-based search occurs based on an intuitive interaction model and may involve conjunctive and disjunctive nested queries. Options to limit the depth, the degree and the regular expression usage (AND/OR), are available, simplifying the excessive use of nested filters.



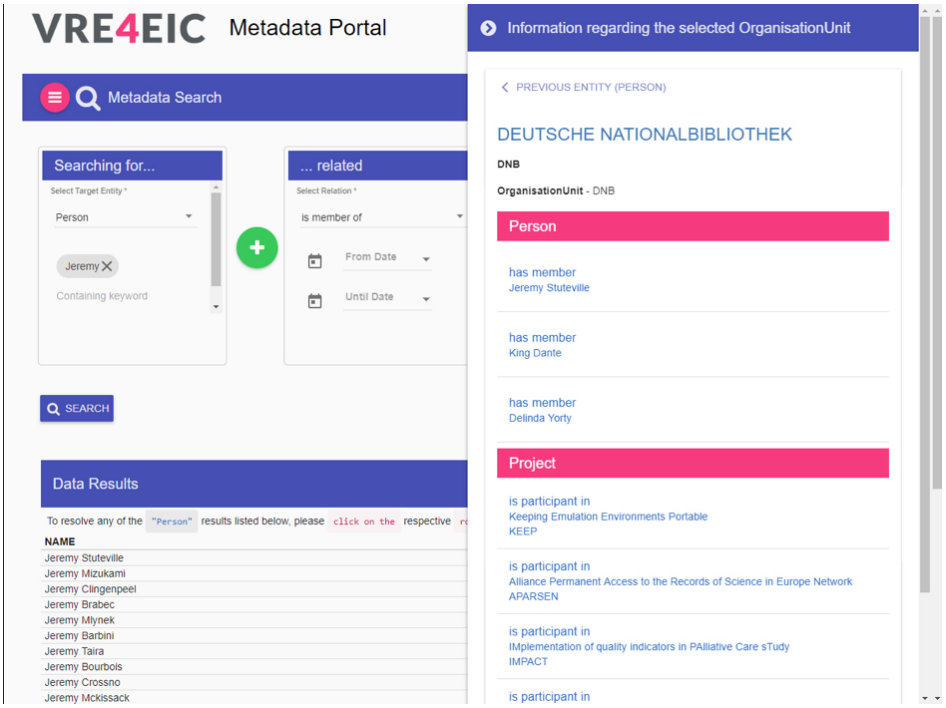
Geo-spatial queries are constructed with the help of an interactive map, offering most of the functionalities that one expects to find in similar systems, such as searching by toponyms or geographical regions or selecting specific instances to be included in the query etc.



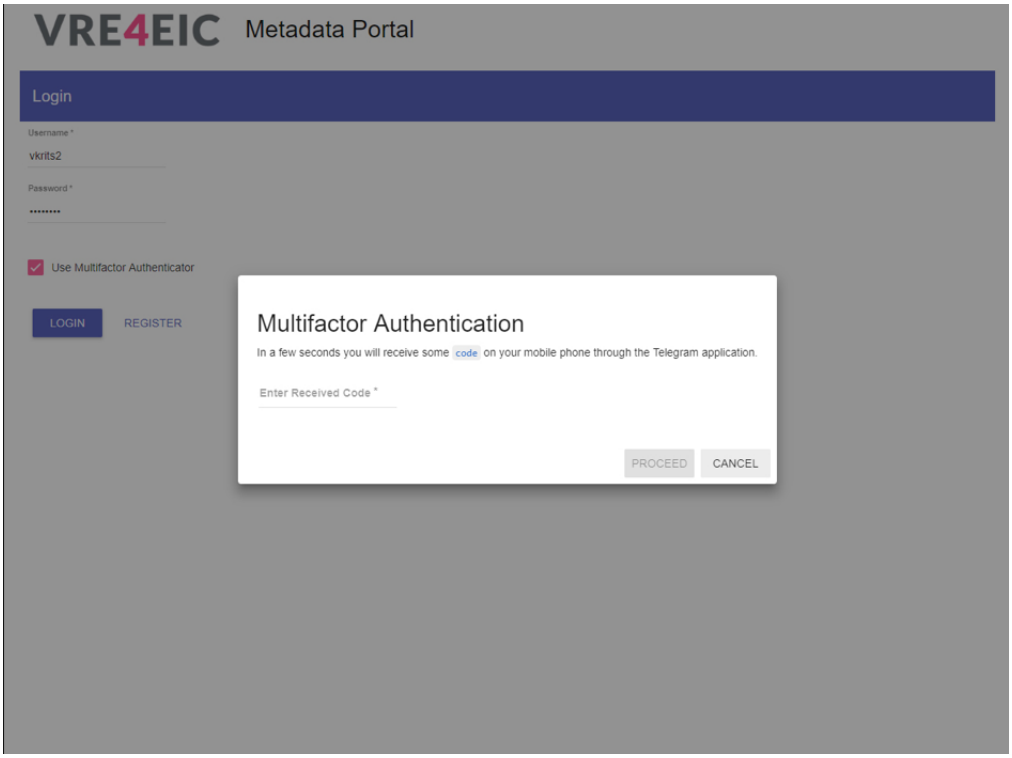
The platform also gives users the ability to store queries for later use. This is a handy option, enabling the retrieval of complex queries that can be used as templates for constructing competency queries with multiple filters or for making minor adaptations to complex exploration tasks.



After executing a query, the results can be examined through an internal resolver, allowing simple navigation from any instance of the targeted entity to any of the instances of the related entities. In that way, end users can even continue their discovery after the results are retrieved.

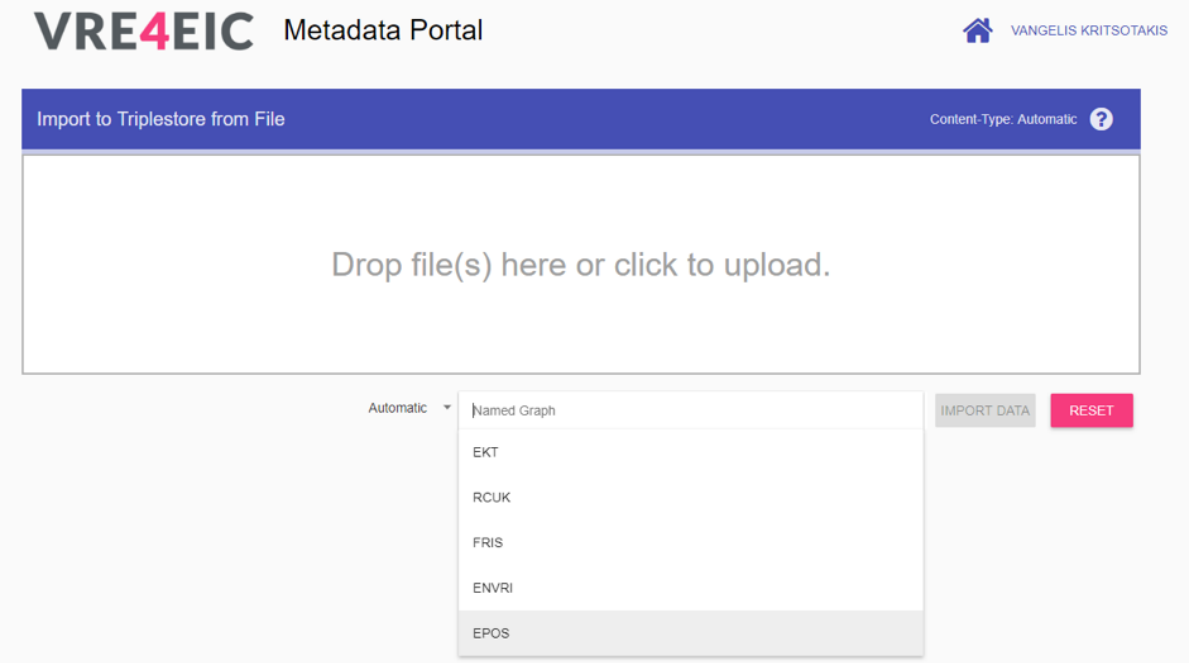


The platform provides a multi-factor authentication mechanism for granting access to the users. This mechanism requires users to present two pieces of evidence, which are their regular credentials and a code sent to the "Telegram Messenger" account, they possess. This compartment also provides Role Based Access Control (RBAC), ensuring that users actions are regulated according their knowledge background determined through their user roles.

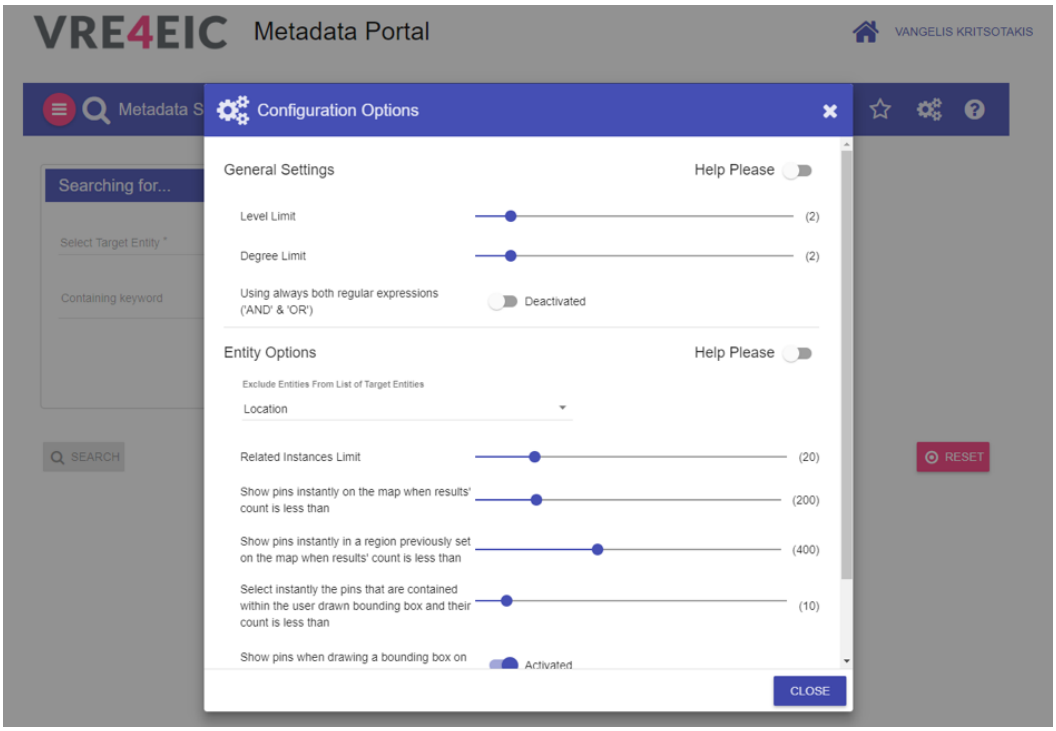


The platform allows data import from a variety of RDF file formats on either existing or new defined graphs. The process is as simple as a drag & drop and supports multiple file upload in a single step.

During this process, the system acquires any available user-profiling material and uses it as extra provenance information to accompany the imported data. This provenance information is important for knowing who has import what and where.



Finally the platform is configurable on many different levels, based on a specialized dashboard, in order to enhance flexibility, robustness and simplicity. Configuration options are accessed directly through the GUI (an administrator user role is required) and can significantly affect functionality, system performance and the level of complexity. These parameters are mainly limitations, regulations and option exceptions to be set on queries, the logical expressions used, or the available entities.



Overall, a more detailed listing of functionalities supported by the platform is provided in Annex section of this deliverable. The functionality covers aspects related to security, data presentation and discovery, data import and export, system configuration and administration and system's robustness and fault tolerance.

6.4 Architecture of the VRE4EIC Metadata Portal GUI

The GUI is implemented by several sub-components which interact with external components (mainly using restful web services). A high-level diagram of the platform is presented in Figure 40

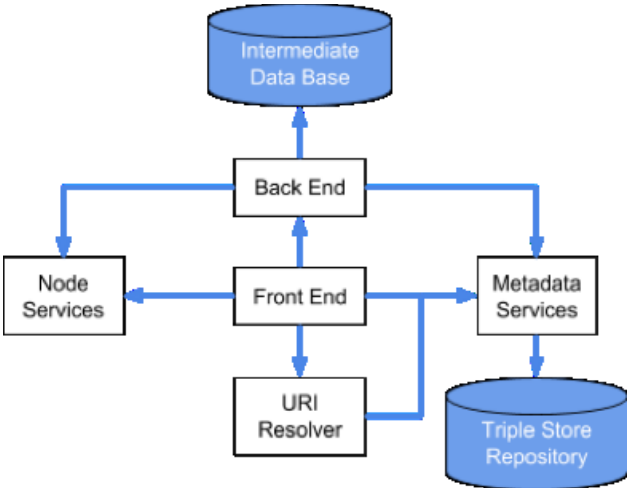


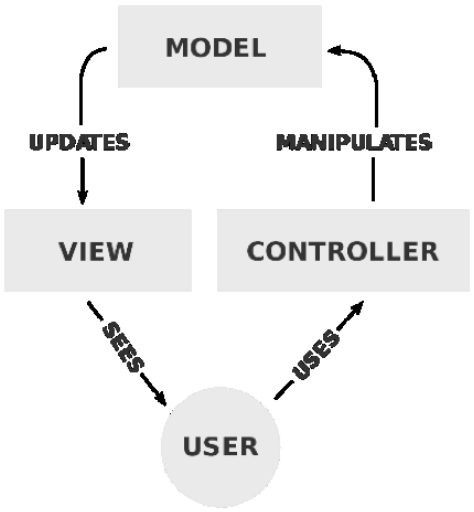
Figure 40 Portal GUI Platform

In the next sections, each of the components, constituting the portal, are described in more details along with their responsibilities and main functionality.

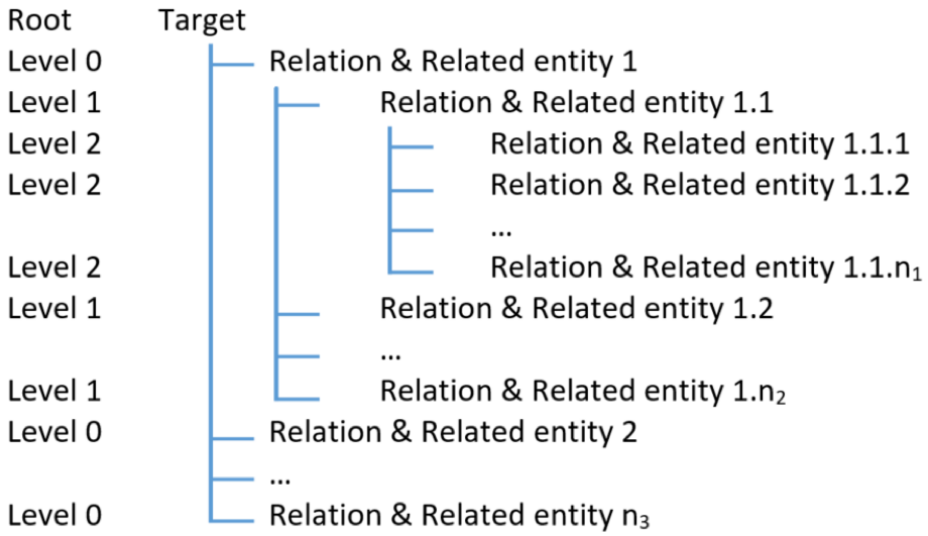
6.4.1 Front End

The front end is mainly responsible for facilitating human-computer interaction and provides the required features that constitute the GUI intuitive and usable. However, its functionality is further expanded, since it is also responsible for implementing the required logic for executing users' actions and deciding the proper services to be called. Moreover, this component is responsible for properly handling errors and informing end-users about them when they occur. Finally, tasks related with users' login or registration are directly handled by this component. The front end is based on the Model View Controller (MVC)[12] design, where the user interface layer is isolated from the application's logic. The available controllers receive http requests and dynamically build the required models for the data view. This view then uses the data prepared by the controller to generate a final presentable response.

[12] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller



The logic of the front-end regarding the dynamic construction of queries, relies on the general assumption that the user is looking for a "target entity" which is "related" to one or more "entities". That assumption is further expanded so that each of the "entities" can be "related" to other "entities", forming in this way a tree model on the fly, describing the constructed query. The picture below shows the tree model in a generic way.



Tree model describing the general concept under which relies the logic of dynamically constructing queries

All nodes under the root node are actual filters applied on target entity. The same applies all node under any node parent. Any nodes in the same level that have the same parent and are more than one are accompanied by a regular expression (OR/AND) that defines the way filters are applied together.

6.4.2 Back End

The back end is responsible for serving all front end needs by executing the required functions and calling the appropriate web services. The back end mainly consists of controllers, internal back-end services (not to be confused with web services) and an Intermediate database.
Each controller is responsible for calling the appropriate services or external web-services, which combined together can fulfill a task. Moreover, the responsibility of the controllers is to deliver the required output at the front end, or the appropriate error message, if some failure occurs. This sub-

component is responsible for checking the token's validity in any interaction with the front-end and acts respectively. On the other hand, services are responsible for accomplishing more specific tasks that usually aim on single targets. These services interact with web services to achieve their goal. The involved web services are the node services, related with user's profile and security, and the metadata services, related to query execution and metadata import / export, which are explained next. Since the platform interacts with users and is flexible enough to operate in a generic form, configuration options and users' structured queries, have to be stored to some place different than the Triple Store Repository itself, since this is only used for storing metadata. As such, an intermediate database is used for serving this purpose.

6.5 Interactions of the VRE4EIC Metadata Portal with the eVRE building blocks

6.5.1 Node Service

The VRE4EIC Metadata Portal interacts with the Node Service for:

- User authentication at login;
- User registration;
- Retrieving user's profile information;
- Retrieving User roles for applying RBAC;

It is important to mention that the portal itself does not hold any personal information related to end-users, since this is the responsibility of the Node Services to fulfill in some remote and secure repository.

6.5.2 Metadata Service

The Metadata Service is the building block, responsible for providing services related to metadata management. These services can be further classified into Query Services and Import/Export Services. Query services execute queries on the data stored into the Triple Store Repository and deliver the output back to the requester. Import services can insert data formed in a variety of formats into the Triple Store Repository. Finally, Export Services can extract data specified from the Triple Store repository and deliver it back to the requester in a variety of formats. The Metadata Restful API are documented using Swagger an documents can be accessed using this link:
https://app.swaggerhub.com/apis/rousakis/ld-services/1.0.0.
The set of specific eVRE WS entry points used in the VRE4EIC Portal appear in Figure 41.

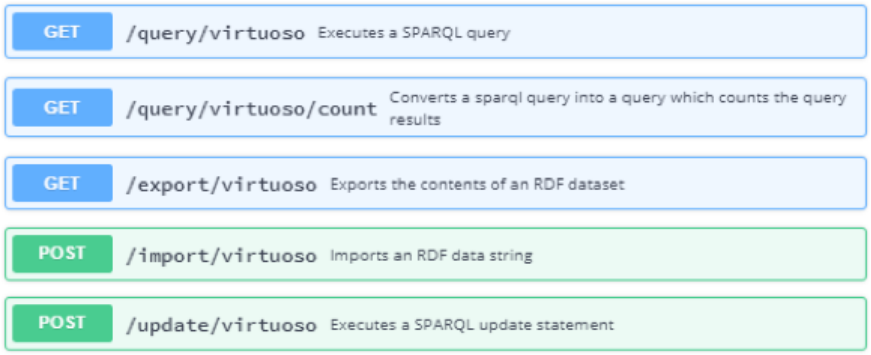


Figure 41 eVRE WS Entry Points for the Metadata Portal

6.6 Technologies used in implementing the VRE4EIC Metadata Portal

A powerful combination of technologies has been used to implement the VRE4EIC Graphical User Interface

- Spring Boot (a Web Application Using Spring MVC)
- AngularJS (a structural framework for dynamic web apps based on HTML and JavaScript)
- Bootstrap-UI & Material Design (UI component frameworks)

All the components can be executed from the command line as standalone Maven applications, since they include an embedded server container (Jetty by default, however Tomcat can also be embedded)

- No prior installed software is required (except of the JVM);
- No prior configuration is needed;
- Independent, portable and easy to be deployed

7 Validating the VRE4EIC Reference Architecture: the EPOS and ENVRIplus enhanced VREs

7.1 The enhanced EPOS VRE

EPOS, the European Plate Observing System, is a long-term plan to facilitate integrated use of data, data products, and facilities from distributed research infrastructures for solid Earth science in Europe. EPOS will bring together Earth scientists, national research infrastructures, ICT (Information & Communication Technology) experts, decision makers, and public to develop new concepts and tools for accurate, durable, and sustainable answers to societal questions concerning geo-hazards and those geodynamic phenomena (including geo-resources) relevant to the environment and human welfare.
EPOS vision is that the integration of the existing national and trans-national research infrastructures will increase access and use of the multidisciplinary data recorded by the solid Earth monitoring networks, acquired in laboratory experiments and/or produced by computational simulations. The establishment of EPOS will foster worldwide interoperability in the Earth sciences and services to a broad community of users.

The collaboration and interaction between EPOS and VRE4EIC was carried along two main dimension: EPOS integration and EPOS enhancements.
In the first one, EPOS contributed to make its assets (metadata, datasets etc) available in an integrated way through VRE4EIC system prototype. In the second one, EPOS took advantage of existing building blocks from VRE4EIC that implemented missing functionalities in EPOS. This interaction hence demonstrated both the feasibility of the VRE4EIC concept, that aims at integrating heterogeneous resources in a homogenous way, and the added value of the architecture and developments carried on in VRE4EIC, that can contribute to the enhancements (i.e. expanding functionalities) of existing VREs and Research Infrastructures.

7.1.1 EPOS Integration with VRE4IC: workflows

In the framework of integrating EPOS resources in VRE4EIC, EPOS provided access to to so called "scientific workflows". The Figure 42 shows how the integration was done:

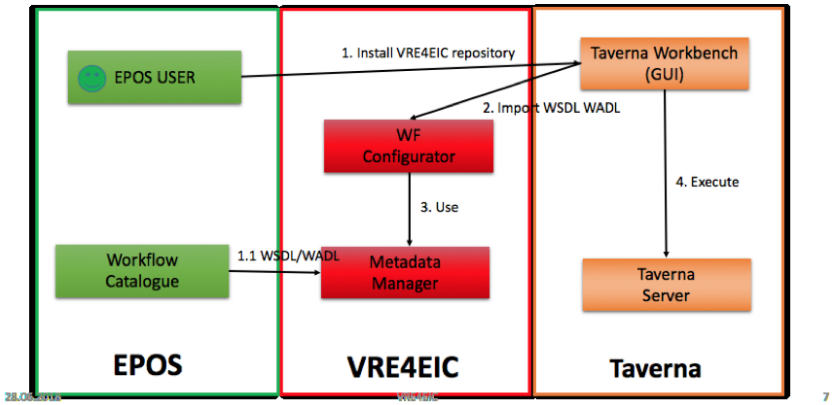
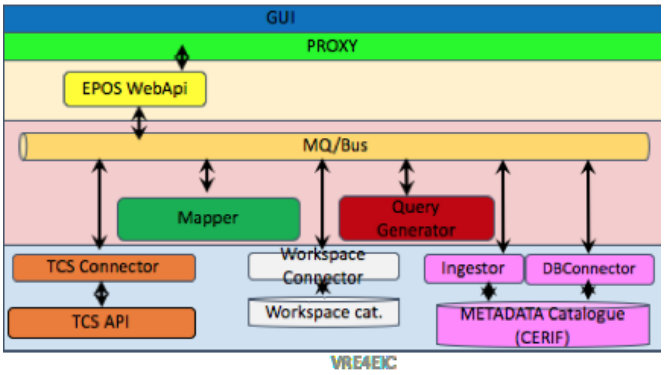


Figure 42 EPOS-VRE4EIC integration

The three boxes represent respectively EPOS system, VRE4EIC system and the "user system" (e.g. laptop).
Initially an EPOS user, but might be any user, launch on his/her own laptop the TAVERNA workbench application in order to execute some scientific workflow (step 1 in the picture). In order to access to workflows provided by the VRE4EIC system, the user install a plugin that automatically connects to the WF configurator component (in the VRE4EIC domain) and fetches web services descriptions stored ito VRE4EIC metadata manager (step 2 and step 3 in the picture). The metadata manager, in turn, access to webservices description from EPOS workflows catalogue (whether runtime or by ingesting information in advance).
This ensures that any non-skilled user can take advantage of workflows and webservices from EPOS domain (but potentially from any domain) just by installing a plugin on its workflow application (in this case taverna workbench).

7.1.2 EPOS architecture enhancement

First enhancement in EPOS was in terms of system architectural paradigms and functionalities. Leveraging on the studies, tests and experience from WP3, EPOS technical team started a fruitful collaboration. Both systems start from a common baseline, which can be summarised with the a) adoption of the same architectural paradigm (Microservices), b) adoption of the same metadataoriented approach, through the use of a CERIF based metadata catalogue.
EPOS technical staff and VRE4EIC WP3 had regular meetings and discussion, both remotely and face to face (at the Project's meetings) where the two architectures were compared.
As a result, EPOS architecture was enhanced by moving from a microservice centralized management, to a microservice choreography approach, both described in Deliverable 3.4
A snapshot of the EPOS Architecture follows.



7.1.3 EPOS functionality enhancement

A second enhancement of EPOS was in terms of additional functionalities provided by VRE4EIC.
As mentioned, the main EPOS system, called Integrated Core Services Central hub, use a microservice approach. It includes a central queuing system and several components that performs atomic tasks and are connected to the queuing system. Examples of microservices are reported in the above in the diagram: the queueing system, the connector to Thematic Core Services and the metadata catalogue and others.
As evident from the diagram, some components implementing functionalities as authorisation are missing. They are however implemented by components or "building blocks" developed in the context of VRE4EIC. Building blocks are basically the components of the VRE4EIC reference architecture. Technically each building block is a microservice implementing a major component of the Reference Architecture, so it can be easily "plugged" into other architectures.

In this case, the building block to consider is the «AAAI service». AAAI stands for Authentication, Authorization, Accounting Infrastructure, that is to say a system to manage user secure access to a system with authorization. The main characteristic of this service is that it integrates different authentication mechanisms in one single system, thus providing a user a single point of access to resources. In practice, it means that independently from his or her account credentials, user will type them in one single form. Integrating it into an existing Research Infrastructure like EPOS, means avoiding the effort of integrating many different authentication services into one Research Infrastructure, with all related technical and security issues.
EPOS tested the module, and the results were encouraging, as it enabled EPOS users with existing credentials to log in to EPOS in an easy way, without jumping from a website to another. It indeed integrated several heterogeneous Identity Providers, like Edugain, Google, but potentially – what we aim at doing in the future – also other providers, both academic and generic, for instance ORCID, GitHub, Facebook.

The Figure 43 shows in terms of functional blocks how the enhancement was achieved.

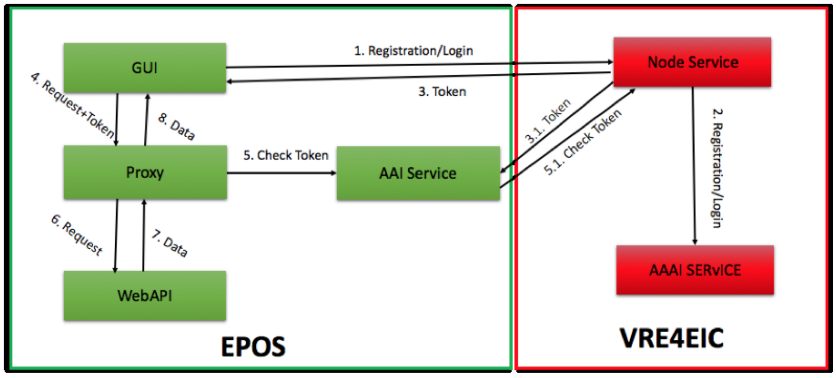


Figure 43 EPOS Functionality Enhancements

Green box represent the EPOS system, while the red box represents the VRE4EIC system prototype.
Users access to the EPOS GUI, and when they want to perform login they are redirected (step 1) to the VRE4EIC node services. This service in turn register/login the user by means of the VRE4EIC AAAI service (step 2). Response is enclosed into a token that can be used from the GUI (step 3). Such token will then be used by the GUI for communicating with the EPOS WEB APIs. Anytime the GUI makes a call the the EPOS WEB APIs, indeed, the token goes through a proxy that checks its validity by connecting to the VRE4EIC service (steps 4-5). If the token is recognised as valid, then the request proceeds (step 6) and WEB APIs respond with a data pyload that is then rendered or presented by the GUI to the user (steps 7-8).
These components interoperate in real time, thus providing the functionality of logging in to the EPOS Web Interface by using the services provided externally by VRE4EIC. All this complexity is hidden to the user, who just insert login and password in one simple login form.
A working demo of this integration can be found here:

http://nodedev.bgs.ac.uk/epos/epos-gui/otherAAIversion/search

Clicking on the "login" top right button, user can log in and is automatically recognised by the system.

7.2 Enhancement in ENVRI plus VRE

The ENVRI community represents a cluster of environmental and earth science research infrastructures, and thus represents a vital forum in which to promote the eVRE solutions developed

in the project. The Data for Science theme within the ENVRIplus project is concerned with providing common technical solutions and recommendations to many of the problems shared by the ENVRI community, for example with regard to metadata cataloguing, provenance, identification and citation of persistent resources and data processing. Thus the eVRE architecture and building blocks solutions have both been exploited to the ENVRIplus community as part of the ENVRI service portfolio of technologies, standards and recommendations. More specifically, eVRE developments have been applied to the problems of enhancing i) cross-RI data and service discovery, ii) cross-RI workflow composition, and iii) cross infrastructure workflow execution and provenance.

7.2.1 Community catalogue for cross-RI data and services

In the ENVRIplus community, data and other digital assets are catalogued using different metadata standards (e.g., CKAN, ISO 19139 and Dublin Core) and using different technical solutions to serve metadata (e.g., B2FIND, GeoNetwork and Get-IT). Such diversity makes cross-RI data and service discovery very difficult.

To address this, the eVRE solution has been used in ENVRIplus to build a contextual rich community catalogue for multiple research infrastructures in ENVRIplus. The CERIF standard was included in the ENVRIplus catalogue recommendation together with CKAN (used by EUDAT's B2FIND service). More specifically, the following two actions were taken to address the short term and long term challenges respectively.

7.2.1.1 Short term: manually setting up a CERIF-based data catalogue

In the short term, the CERIF database and catalogue software stack already implemented by EPOS (which is also a member of the ENVRI community) have been directly deployed in ENVRIplus by ENVRIplus project partner IFREMER. By setting the ENVRIplus instance, a small set of records from SeaDataNet (concerned with the marine domain), ICOS (concerned with the atmospheric domain) and ANAEE (concerned with ecosystems and biodiversity) are being manually ingested. Figure 44shows the basic scenario:

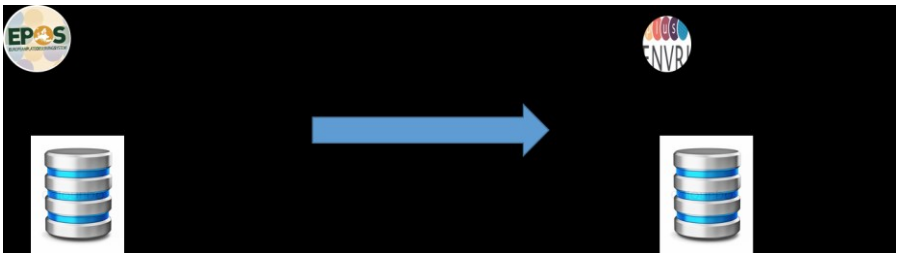


Figure 44: The CERIF catalogue solution provided by EPOS will be deployed more broadly in ENVRIplus.

7.2.1.2 Long term: automatically harvesting CERIF records from diverse ENVRI RI catalogues

In parallel to (a), an automated approach is also being prototyped by the University of Amsterdam within ENVRIplus. The basic idea is to take advantage of the metadata manager and metadata mappings developed in WP4 using the 3M environment, and to build upon that development by automating the manual pipeline currently used by FORTH to dynamically transform metadata records from ENVRI RI catalogues into a single CERIF database. Figure 45 shows the basic scenario:

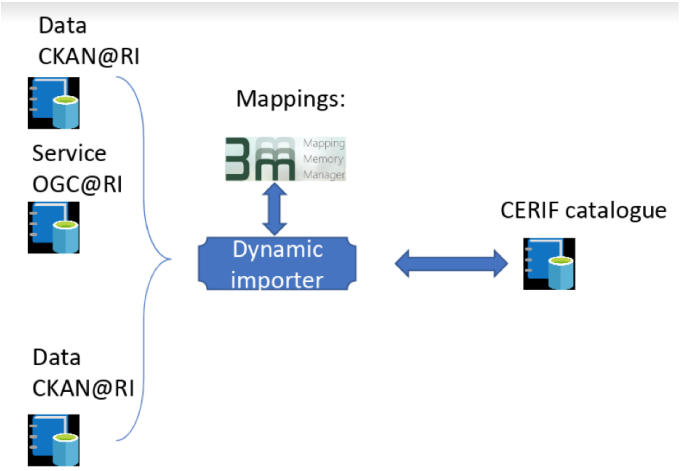


Figure 45: An automated pipeline for harvesting ENVRI RI catalogues into a single CERIF catalogue.

Based on the automated pipeline, we have developed as a service --named Metadata Catalogue Mapper (MetaCatMap)-- that implements a flexible metadata mapping pipeline using different metadata schemes to provide cross-RI metadata search and discovery.
Metadata Catalogue Mapper (MetaCatMap) consists of the following components:

- RESTCat: This is the REST frontend API that takes requests from users to perform the conversion from one standard to another. The parameters or the request are the source metadata catalogue URL and the target mapping standard
- CatTask: This component extracts individual entries from the source metadata catalogue and creates a mapping task and adds it in a task queue
- CatMap: This component pulls a mapping task from the task queue, and queries the X3ML mapping framework for the appropriate mapping that matches the source and target standard
- X3ML Engine: This component performs the actual transformation of an entry from the source standard to the target standard

Figure 46 shows the overall architecture.

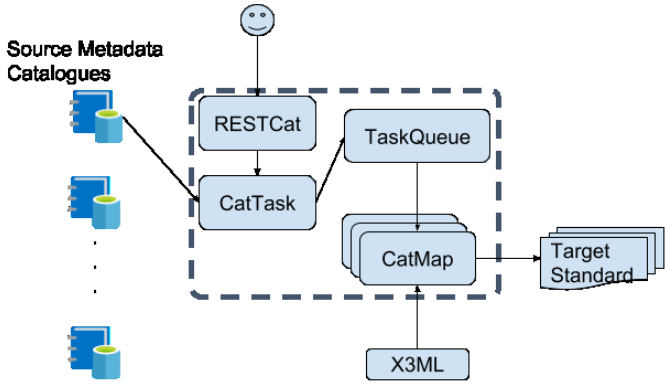


Figure 46

7.2.2 Architecture of the Metadata Catalogue Mapper (MetaCatMap)

Following the construction of the joint catalogue based on CERIF, the semantic search capabilities being developed as part of the metadata service building block of the eVRE will be in turn applied in ENVRIplus to enhance data and service discovery based on vocabulary linking activities now being conducted in ENVRIplus between RIs with similar but currently distinct vocabularies for describing environmental phenomena and observations. Since the metadata from ENVRIplus catalogues will be ingested and mapped onto CERIF, it is possible to exploit the semantic classification layer of CERIF to take advantage of the linked vocabularies produced by ENVRIplus and so provide enhanced semantic search based on the metadata service developed by FORTH, as shown in Figure 47

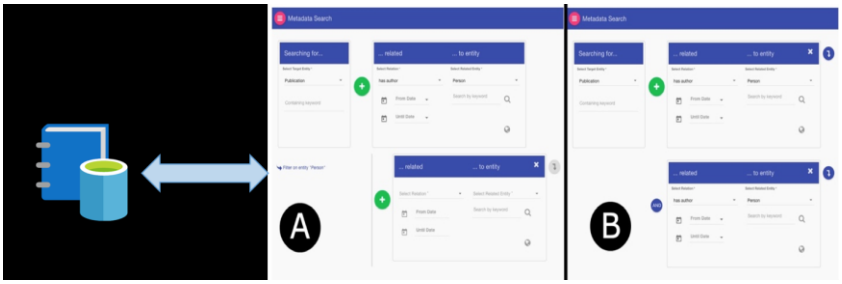


Figure 47 Using the semantic search support provided by the eVRE to enhance cross-RI data and service discovery.

7.2.3 Cross-RI workflow composition

Using the semantic search capabilities of the metadata service and the workflow manager building block provided by the eVRE, the ENVRIplus community is also able to enhance their cross-RI workflow composition capabilities. Figure 48 shows the basic scenario:

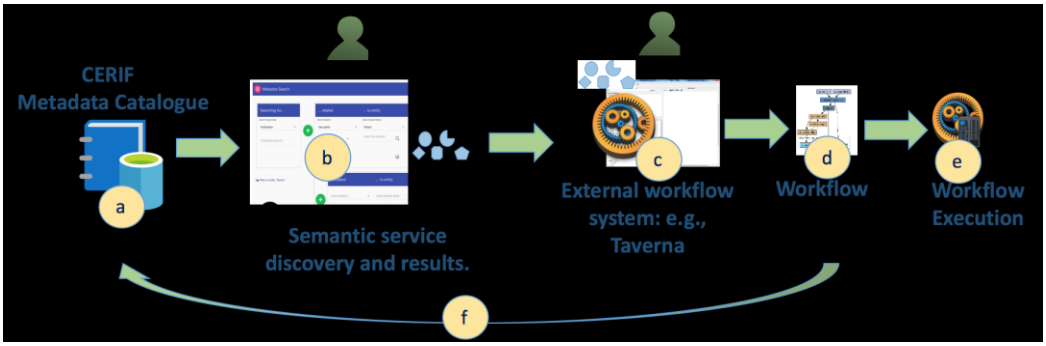


Figure 48 The workflow composition using eVRE building blocks

The process can be broken down into six distinct steps:

- **a.** Ingesting metadata from data and service catalogues (in ENVRIplus) into a CERIF catalogue, as described earlier.
- **b.** Semantically searching data and services using the eVRE metadata service with semantic search capabilities, likewise as described earlier.
- **c.** The workflow environment (in this case we use Taverna), will load the pre-selected services (identified in step **b**) into its own local service catalogue.
- **d.** Composing an executable workflow using Taverna.
- **e.** Executing the workflow using the Taverna engine.
- **f.** Storing the metadata of successful workflow compositions or executions back into the joint CERIF catalogue.

During the integration, the ENVRIplus team specifically compared two scenarios to show the added value of using semantic search, as shown in Figure 49:

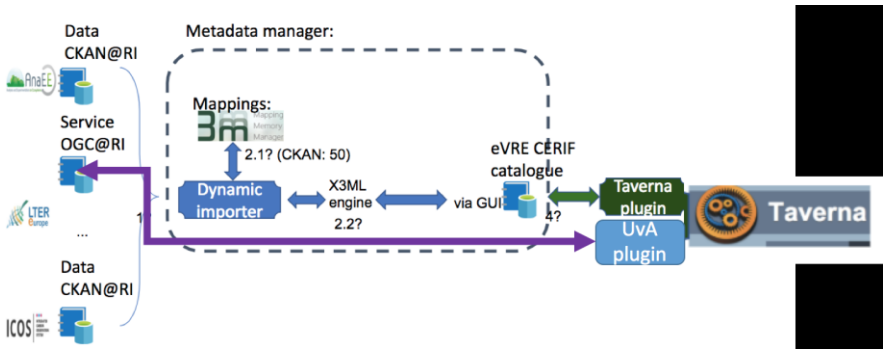


Figure 49 Loading the RI service catalogue into Taverna directly (scenario a, indicated by the pink line via a UvA-developed plugin) or via CERIF and semantic search (scenario b, provided by the CERIF pipeline).

In the context of a single RI, scenario (a) can provide an effective, quick solution. In the case of workflows composed using services provided by multiple RIs however, it is judged that scenario (b) will provide better selection on data and services.

7.2.4 Cross-infrastructure workflow execution and provenance

Finally, the eVRE building blocks have also been used to enhance workflow execution and provenance collection across multiple e-infrastructures. Figure 50 shows the basic scenario:

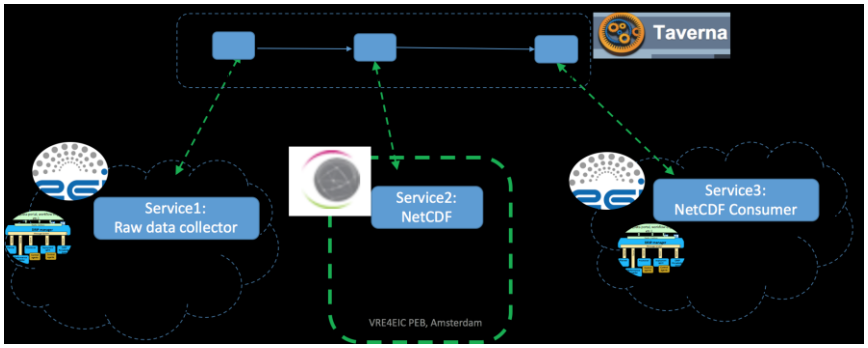


Figure 50 Cross-Infrastructure workflow execution and provenance.

During the integration phase, the workflow manager (the eVRE building block) will be used to perform execution, with services produced by the UvA team together with the ENVRIplus provenance working group used to harmonize the distributed provenance information generated by the different distributed e-infrastructures at different levels on the technology stack, including workflow level, service level and infrastructure logs. A CERIF record will be created to link the different contexts of provenance data from different sources.

8 Conclusions

VRE4EIC has successfully developed a reference architecture which has been demonstrated in the form of a canonical reference prototype and of which components have been used in EPOS and ENVRIplus. The architecture has met the original VRE4EIC objectives in providing a reference architecture for future VREs and components useful to existing VREs/RIs.

9 References

[vre4eic] VRE4EIC project proposal, section: 1.4.2 Innovation potential and advances beyond the state-of-the art

[Newman] S. Newman, Building Microservices, O'Reilly Media. February 2015

[D33] https://www.vre4eic.eu/images/Public_deliverables/D3.3_Building_Blocks.pdf

[D34] https://www.vre4eic.eu/images/Public_deliverables/D3.4_Enhanced_VREs.pdf

[D53]https://www.vre4eic.eu/images/Public_deliverables/D5.3_A_strategy_for_the_VRE4EIC_project_to_handle_security_privacy_and_trust_issues_V2.pdf

[D54] https://www.vre4eic.eu/images/Public_deliverables/D5.4_Strategies_for_the_VRE_end_users_to_handle_security_privacy_and_trust_issues-second_version.pdf

[ZooKeeper] P. Hunt, M. Konar, E Junqueira, B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems", *USENIX ATC*, vol. 10, 2010.

[ActiveMQ] Apache Software Foundation, “Apache ActiveMQ,” http://activemq.apache.org/ , Bruce Snyder, Dejan Bosanac and Rob Davies, “ActiveMQ in Action” 2011 by Published by Manning.

[HOLL] Holl S. Automated Optimization Methods for Scientific Workflows in e-Science Infrastructures. Forschungszentrum Jülich; 2014.

10 Annexes

10.1Generalised functions

The tables in this subsection clarify the connection between the different viewpoints already described in the document. Specifically, as identified in the use cases, there are a number of domain-independent, general tasks that a user of a VRE typically performs, such as querying for metadata, asserting information in the catalogues, and others. These are called *Generalized Functions*. Such Generalized Functions are usually composed of elementary *Functions,* e.g., cross-searching, which are structured based on specific series of steps. The elementary Functions have a direct connection with the *Requirements*, as they have been developed, in order to satisfy one or more of them. Notice that even elementary Functions can be further decomposed into other functions, according to the level of abstraction needed.

Error! Reference source not found. connects the Generalized Functions extracted from the use cases with the elementary Functions and the steps involved.
Then, Table 5 and Table 6 below elaborate on the elementary Functions, associating them with the relevant requirements of each. In addition, they correlate these functions with the corresponding components of the architecture which implements them.

Table 4 Generalized Functions

Generalized Functions	**Pre-condition**	**Steps Involved**	**Included/ Specialized by Fun**
GF1: Querying	**Fun21**: Agent Authentication (the agent has been succesfully authenticated)	The agent accesses the VRE Search UI (in case of a human user) or the corresponding Search API (in case of software entity) The agent prepares a query to be submitted, involving ○ keywords, and/or ○ topic filtering, and/or ○ target (internal or external) sources, and/or ○ target datasets, and/or ○ other filtering criteria During the query preparation process, the system offers spell checking and recommendations (Thesaurus - MM) The query is submitted to the QueryManager by the agent. The Query Manager receives a query and starts parsing it. Specifically: ○ The Query Manager (QM) examines the sources ○ The QueryManager breaks the query into subqueries ○ The QueryManager submits the queries (to MetadataManager and LD Manager) The QueryManager retrieves all results ○ The QM merges the results ○ The QM returns the integrated results The agent browses over the results (IDs of resources) (see GF2: Dataset/Metadata Exploration)	**Fun1**:Simple Search **Fun2**:Cross Search **Fun3**: Advanced Search (the next ones are neither primitive nor abstract; they are compound functions) **Fun11**: Data Collection **Fun12**: Data Sampling **Fun19**: Data Discovery
GF2: Dataset/ Metadata Exploration	**GF1**: Querying	The agent has a list with IDs of resources. The UI shows the results to the user (only for human users) ○ The UI creates the pages with the results ○ The UI shows some information about the query evaluation (might require communicating with other components for retrieving such information) The user selects to browse for more information about a specific result The above request is submitted to QueryManager, along with the selected source (from GF1) The QueryManager inspects the given sources and submits the corresponding requests to MetadataManager and LDManager	**Fun4**:Dataset Viewing **Fun5**: Dataset Preselection **Fun6**: Dataset Customization (the next ones are neither primitive nor abstract; they are compound functions)

Generalized Functions	Pre-condition	Steps Involved	Included/ Specialized by Fun
		The QueryManager returns the integrated results as a graph The UI shows the graph to the user (only for human users)	**Fun19**: Data Discovery **Fun11**: Data Collection **Fun12**: Data Sampling
GF3: eRISoftware	Access to a eRI service through VRE (e.g., instrument-related or other non-VRE service)	The agent accesses the menu or UI or APIs that contains the list of instruments/real-time sensor data/processes/third-party software etc The agent selects the one it desires. The system redirects the agent to the eRI interface The agent interacts directly with a dedicated UI (e.g., it accesses/calibrates/configures instruments) The system maintains a timestamped log with the user's actions	**Fun7**: Instrument Integration **Fun8**: Instrument Configuration **Fun9**: Instrument Calibration **Fun10**: Instrument Monitoring (the next ones are neither primitive nor abstract; they are compound functions) **Fun11**: Data Collection **Fun12**: Data Sampling
GF4: Data Cataloguing		The user wishes to register a new resource to the VRE catalogue or update an existing one She selects the type of resource, in order to be redirected to the appropriate UI She fills in forms asking for the necessary metadata. Some fields are already completed by the system. The system checks the metadata and returns the corresponding messages The user performs quality improvement The user selects to store the metadata to the catalogue (if needed) The system communicates with the underlying eRI to verify	**Fun13**: Resource Registration **Fun14**: Resource Update **Fun17**: Resources Annotation **Fun18**: Metadata Harvesting

Generalized Functions	Pre-condition	Steps Involved	Included/ Specialized by Fun
		that the dataset involved have been stored successfully If the previous step is successful, the system stores the metadata to the corresponding catalogue The system also updates the preservation and provenance catalogues as appropriate	
GF5: Workflow Enactment		The user builds the workflow using the UI of the Workflow Manager The workflow is deployed on the execution engine(s) The workflow is executed The WF Manager monitors the workflow tasks and notify the user for registered update The results are stored in a temporary area accessible by the user	**Fun15**: Workflow Enactment (the next ones are neither primitive nor abstract; they are compound functions) **Fun12**: Data Sampling

Table 5 Query Requirements

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun1: Simple Search	**PRQ10** Simple search **PRQ11** Multiple format support **PRQ14** Spelling checking **PRQ15**	**The user performs a simple search** 1. The user inserts in a form a list of keywords 2. The user submits the form 3. The system retrieves the query results and returns them to the user	**UI** **IM** Query Mediator Query Manager **MM** Resource Catalogue **LD**	**GF1**: Querying

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
	Query suggestion **IRQ5** Citation Tracking		**AAAI**	
Fun2: Cross Search	**PRQ12** Cross searching **PRQ14** Spelling checking **PRQ15** Query suggestion **PRQ20** Linking external resources **IRQ5** Citation Tracking	**The user performs a cross search** 1. The user ticks on the cross searching option 1.1. (optional): The user defines a list of external sources 2. The user inserts in a form a list of keywords 3. The user submits the form 4. The system retrieves the query results and returns them to the user	**UI** **IM** Query Manager **MM** Resource Catalogue **LDM** **AAAI**	**GF1**: Querying
Fun3: Advanced Search	**PRQ13** Advanced search **PRQ14** Spelling checking **PRQ15** Query suggestion	**The user performs an advanced search** 1. The user clicks on the advanced search option 2. The user selects/edits a number of search criteria 3. The user inserts in a form a list of keywords 4. The user submits the form 5. The system retrieves the query results and returns them to the user	**UI** **IM** Query Manager **MM** Resource Catalogue Provenance Catalogue	**GF1**:Querying

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
	PRQ16 Query Filter **IRQ5** Citation Tracking		Preservation Catalogue **LDM** **AAAI**	
Fun4: Dataset Viewing	**PRQ17** Datasets viewing	**The user views all the datasets metadata** 1. The user performs simple search without submitting any keywords 2. The system retrieves the full list of datasets' metadata	**UI** **IM** Query Mediator Query Manager **MM** Resource Catalogue **LDM** **AAAI**	**GF2**: Dataset/ Metadata Exploration
Fun5: Dataset Preselection	**PRQ18** Datasets pre-selection	**The user pre-selects datasets** 1. The user views all the datasets (PRQ17) 2. The user selects the datasets to search on	**UI** **IM** Query Mediator Query Manager **MM** Resource Catalogue	**GF2**: Dataset/ Metadata Exploration

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
			LDM **AAAI**	
Fun6: Dataset Customization	**PRQ19** Dataset customization	**The user selects a set datasets to be exploited during search** 1. The user selects his profile 2. The user views all the datasets (PRQ17) 3. The user selects the datasets to search on	**UI** **IM** Query Mediator Query Manager **MM** Resource Catalogue User Catalogue **LDM** **AAAI**	**GF2**: Dataset/ Metadata Exploration

Table 6 Data Requirements

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun7: Instrument Integration	**CLRQ1** Instrument Integration	**The user views many instruments** 1. The user selects to views all the available instruments 2. The system returns an integrated list of instrument description to the user	**UI** **IM** Query Mediator Query Manager	**GF3**: eRISoftware

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
			MM Resource Catalogue **LDM** **AAAI**	
Fun8: Instrument Configuration	**CLRQ2** Instrument Configuration **CLRQ4** Instrument Access **CLRQ5** Configuration Logging **CLRQ10** Process Control	**The user configures an instrument** 1. The user selects an instrument 2. The system redirects the user to the RI interface 3. The user configures the instrument (The RI creates the appropriate logs)	**UI** **SM** Resource Manager **MM** Resource Catalogue **IM** **AAAI**	**GF3**: eRISoftware
Fun9: Instrument Calibration	**CLRQ3** Instrument Calibration **CLRQ4** Instrument Access	**The user calibrates an instrument** 1. The user selects an instrument 2. The system redirects the user to the RI interface 3. The user calibrates the instrument	**UI** **SM** Resource Manager **MM** Resource Catalogue	**GF3**: eRISoftware

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
			IM **AAAI**	
Fun10: Instrument Monitoring	**CLRQ6** Instrument Monitoring **CLRQ7** (Parameter) Visualisation **CLRQ8** (Real-Time) (Parameter/Data) Visualisation	**The user monitors an instrument** 1. The user selects an instrument 2. The system redirects the user to the RI interface 3. The RI returns information to the user on the instrument's status/parameters/data	**UI** **SM** Resource Manager **MM** Resource Catalogue **IM** **AAAI**	**GF3**: eRISoftware
Fun11: Data Collection	**CLRQ11** Data Collection **CLRQ12** (Real-Time) Data Collection	**The user retrieves data from an instrument** 1. The user performs a cross searching advanced search 2. The system returns a list of results and the hosting RIs 3. The user retrieves/collects the data from the RIs	**UI** **IM** Query Manager **MM** Resource Catalogue User Catalogue Resource Catalogue **SM** Resource Manager **LD<** **AAAI**	**GF1**: Querying **GF2**: Dataset/ Metadata Exploration **GF3**: eRISoftware

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun12: Data Sampling	**CLRQ13** Data Sampling	**The user performs data analysis** 1. The user performs a cross searching advanced search 2. The system returns a list of results and the hosting RIs 3. The user retrieves/collects the data from the RIs 4. The user selects the analysis to be performed (workflow) 5. The system returns the analysis results	**UI** **SM** Resource Manager **MM** Resource Catalogue User Catalogue **IM** Query Manager **WM** **AAAI**	**GF1**: Querying **GF2**: Dataset/ Metadata Exploration **GF3**: eRISoftware **GF5**: Workflow Enactment
	CTRQ9 Online Dataset Editing **CLRQ14** Noise Reduction **CLRQ15** Data Transmission **CLRQ16** Data Transmission Monitoring **PVRQ1** Data Provenance **PVRQ2** Data Acquisition Information	**RIs responsibility**		

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
	PVRQ3 Data Curation Information **PVRQ4** Data Publication Information **IRQ1** Data Identification **IRQ3** Raw Data Identification **IRQ4** Data Citation **CRQ1** Data Product Generation **CRQ7** Data Replication **CRQ8** Replica Synchronisation **PRQ7** Data Compression **PRQ29** Data Processing Monitoring **PRQ35** Data Backup			
Fun13:	**CLRQ17**	**The user registers a resource performing (meta)**	**UI**	**GF4**: Data Cataloguing

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Resource Registration	Data Cataloguing **CRQ2** Data Quality Checking **CRQ3** Data Quality Verification **CRQ6** Data Storage & Preservation **PRQ4** Resource Registration **PRQ5** (Metadata) Registration **PRQ6** Data Conversion **CLRQ9** Experiment **CTRQ15** Funding body Information **CLRQ18** Data Publication **IRQ2** Data Provider **IRQ4** Data Citation **PRQ8** Semantic Harmonisation	**data quality checking** 1. The user selects to register/update a new resource in the catalogue 2. The system checks the metadata and returns the corresponding messages 3. The user performs the quality improvement 4. The user selects to store the metadata to the catalogue 5. The system stores the metadata to the corresponding catalogues	**SM** Resource Manager **MM** Whole **IM** DMM **LDM** **AAAI**	

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun14: Resource Update	**CRQ4** Data Versioning	**The user registers a new version of a resource** 1. The user selects to register a new resource version in the catalogue 2. The system updates the resource, preservation and provenance catalogues accordingly	**UI** **SM** Resource Manager **MM** Whole **IM** DMM **LDM** **AAAI**	**GF4**: Data Cataloguing
Fun15: Workflow Enactment	**CRQ5** Workflow Enactment **PRQ26** Scientific Workflow Enactment **PRQ28** Data Processing Control **ORQ2** Processing Parallelisation **ORQ4** Data Compartmentization	**The user creates a workflow** 1. The user creates a workflow 2. The user select to run the workflow 3. The system returns the workflow's results	**UI** **WM** **MM** **IM** **AAAI**	**GF5**: Workflow Enactment
Fun16: Access Control	**CRQ6** Data Storage & Preservation	**Fundamental Infrastructure's Functionality**	**MM** **AAAI**	

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun17: Resources Annotation	**PRQ1** Resources Annotation **PRQ2** (Data) Annotation **PRQ32** Quality Rating **PRQ34** Data Tag	**The user annotates data/resource** 1. The user selects a resource 2. The user annotates the resource 3. The system stores the appropriate information	**UI** **SM** Resource Manager **MM** Resource Catalogue Metadata Catalogue	**GF4**: Data Cataloguing
Fun18: Metadata Harvesting	**PRQ3** Metadata Harvesting	**The user enables metadata harvesting** 1. The user selects a source to be harvested 2. The user selects the frequency 3. The system harvests the metadata and either registers a new resource or updates an existing one's metadata	**UI** **SM** Resource Manager **MM** Whole **IM** DMM **LDM** **WM**	**GF4**: Data Cataloguing
Fun19: Data Discovery	**PRQ9** Data Discovery and Access **PRQ31** Dataset Download	**The user discovers data using the searching capabilities (see Query Requirements)**	**UI** **IM** Query Manager **MM** Resource Catalogue **LDM**	**GF1**: Querying **GF2**: Dataset/ Metadata Exploration
Fun20:	**CTRQ4** Interface	1. On the user device: User goes to eVRE portal in the	**UM**	For details on the authentication

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
User/ Agent Registration	Customization **CTRQ5** Wizard Configuration **CTRQ7** Multilingual Interface	browser, and creates her/his own user profile using functionalities provided by the User Manager. 2. The User Manager interacts with MM to register the profile and gets from the MM the configuration information needed to configure the user environment 3. Once the profile is created UM asks "Do you want to register this device with eVRE as authenticator?" User agrees 4. Depending on the device UM can ask the user to download and install a software implementing VRE4EIC authentication API (to enable the device to implement authenticator client functionalities) 5. UM registers the device in MM and AAAI as authenticator 6. UM shows message, "Registration complete."	**AAAI**	and registration framework adopted in eVRE see the section Notes on the EVRE authentication protocol above
Fun21: Agent Authentication	**CTRQ1** Login **CTRQ31** Accounting	**1) User Authentication (external authenticator mode):** - User access eVRE in browser, sees an option "*Sign in with your registered device.*" - User chooses this option and gets a message from the browser, "*Please complete this action on your phone.*" -**On the phone**: - User sees a notification similar to "Sign in to eVRE?" - User selects this notification. - User is shown a list of their VRE4EIC identities, e.g., "*Sign in as Alice / Sign in as Bob.*" - User picks an identity and provides it.	**UI** **UM** **AAAI** **MM (Fun)**	

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
		-**On the laptop**: - Web browser shows that the selected user is signed in, and navigates to the signed-in page/restore the user session etc. **2) Device authentication (<u>external</u> authenticator mode)** - User connects an external instrument to a device or a network connected to eVRE - The device sends a message to eVRE to notify that he wants to add a new device to the Research Environment - On the user laptop/smartphone signed to eVRE,user sees a prompt similar to "*Authorise instrument to sign in on eVRE, chose identity*" - User is shown a list of their eVRE identities, e.g., "*Sign in as Alice / Sign in as Bob"*. - User picks an identity and provides it. - Instrument is ready to operate on eVRE.		
Fun22: Continuous Access	**CTRQ2** Continuous Access	1)**Continuous access: save session Use Case (session explicitly closed)** - Agent authenticates on eVRE - if present, the previous session is restored - agent operate on eVRE - Agent invoke 'sign out' functionality in the UM GUI, or specific 'save session' functionality - UM saves all information related to the current session in a specific storage assigning an identifier to	**UI, UM, AAAI, MM**	This requirement has been interpreted as: 'maintain user's session across multiple connections'

Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
		the saved session - UM interacts with the MM manager in order to: i) associate the session identifier to the User Profile and ii) change the status of the User profile in case of sign out - If required UM interacts with AAAI, this could be required when we need to notify to external eRI that the connection is closed and the user is no longer operating on their resources **2) Continuous access: save session Use Case (session not explicitly closed)** **-** Agent authenticates on eVRE - if present, the previous session is restored - Agent operate on eVRE - For a defined amount of time no action occurs - UM automatically saves all information related to the current session in a specific storage assigning an identifier to the saved session - UM interacts with the MM manager in order to associate the session identifier to the User Profile **3) Continuous access: restore session Use Case** - Agent authenticates on eVRE - The User Manager interacts with Metadata Manager to discover if a session identifier is associated to the user profile - If an id is returned the information is retrieved from the specific storage - The UM interacts with the AAAI to verify that the user can access the content of the old session-is restored -Agent operates on eVRE		

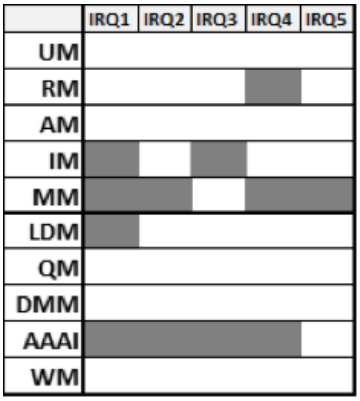
Function ID	Requirement ID	Function Description	Involved components	Related Generic Functions & Notes
Fun23: Update Alert	**CTRQ8** Update Alert **CTRQ10** Notification	**1) Update Alert: event subscription** - User authenticates on eVRE - The user uses User Manager to subscribe to events - The UM saves subscriptions on the User Profile (MM) and associate the id of the user to the list of users subscribed to that event in the event list **2) Update Alert: update notification** - User authenticates on eVRE - The user session is restored - The user uses the UM functionalities to read the alerts related to her/his subscriptions.	**UI, MM, UM, RM, IM, AAAI**	An user receives a notification when an event she is interested in occurs. Generally speaking there could be a large number of events in a eVRE, to name a few: changes in the lifecycle of processes, changes in the availability status of a resource published by a RI, updates of a shared documents etc.
Fun24: Resource Connection	**CTRQ28** Computing Resource Connection	**Access EGI computational resources via OCCI** The user has previously obtained a VOMS certificate (http://www.eu-emi.eu/training/cert-tutorial) via AAAI functionalities, this information is registered in the user profile and the actual certificate is managed by AAAI. - User authenticates on eVRE - The user session is restored - User selects the EGI entry point - The user sends a request, validated with the VOMS certificate, to the EGI entry point asking for computational resources - The user chose the references to the computational resources and use them	**UI, IM, AAAI, MM**	This requirement means the possibility to access computational or storage resources for an application installed in the eVRE. These resources can be provided by eVRE environment or by an external provider. This requires that the user has permission to use those resources and there is an Adapter in the IM that interacts with those resources,

10.2Requirements and components

The tables in this Section show the components involved in the implementation of user requirements. They have all the same structure, and each of them reflect a pillar of requirements. For every requirement pillar, the table highlights: the relationships with other requirements, the relationships with the metadata catalogue ("CERIF entities" column) and notes and comments that we have on the requirement.

Before each table, a diagram is given reporting the incidence matrix between the components in the Reference Architecture (row) and the requirement (column).

10.2.1 Data Identification and Citation



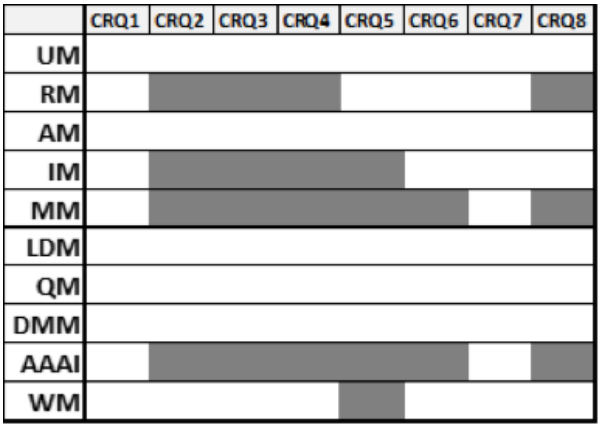
UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 7 Data Identification and Citation

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
IRQ1	Data Identification	CLRQ17	AAAI, MM, IM, LDM	cfResultProduct (dataset), associated cfFedId	
IRQ2	Data Provider		AAAI, MM	cfOrgUnit_ResultPriduct	
IRQ3	Raw Data Identification	IRQ1	AAAI, IM		

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
IRQ4	Data Citation	IRQ1	AAAI, RM, MM		
IRQ5	Citation Tracking		MM		Through the exploitation of the Provenance Catalogue

10.2.2 Data Curation



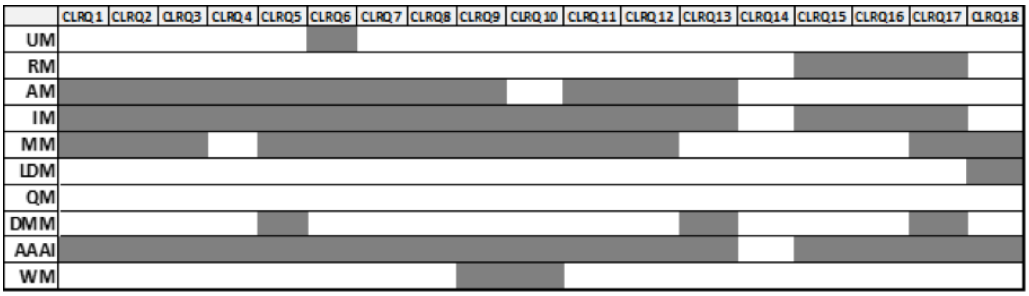
UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 8 Data Curation

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
CRQ1	Data Product Generation				eRI's responsibility
CRQ2	Data Quality Checking		RM, AAAI, MM, IM		Some checking and verification is needed before including data descriptions in our catalogues

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
CRQ3	Data Quality Verification	CRQ2	AAAI, RM, MM, IM		
CRQ4	Data Versioning		AAAI,RM, MM, IM, LDM		Only metadata update and versioning is our responsibility
CRQ5	Workflow Enactment		AAAI, WM, MM, IM		
CRQ6	Data Storage & Preservation		AAAI, MM		we only deal with the metadata here
CRQ7	Data Replication				eRIs responsibility
CRQ8	Replica Synchronisation		AAAI, RM, MM		eRIs responsibility

10.2.3 Data Cataloguing



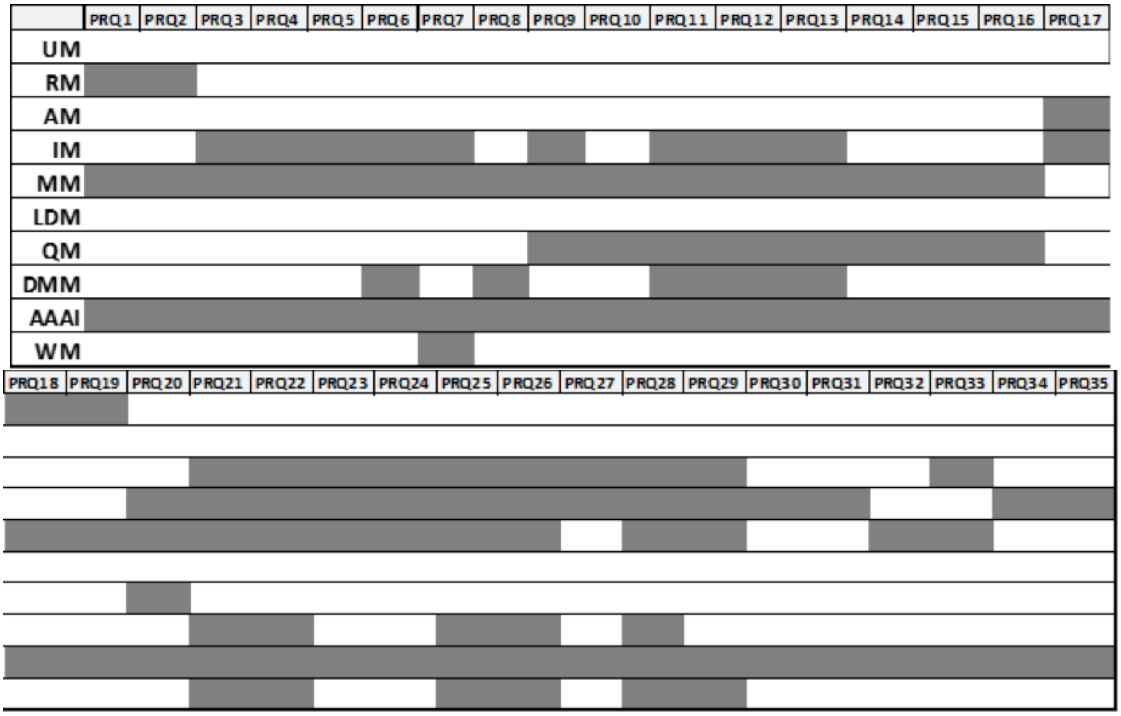
UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 9 Data Cataloguing

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
CLRQ1	Instrument Integration		AAAI,AM, MM, IM, LDM		CERIF has an equipment and a measurement entity For this set of requirements (CLRQ) we have to match them to the ENVRI 6 pillars. we assume we access the eRI functionalities, we only say to the users what they can do and instruct the eRIs to start their sensors etc. We act as mediators
CLRQ2	Instrument Configuration		AAAI,AM,MM, IM		
CLRQ3	Instrument Calibration	CLRQ2	AAAI,AM, MM, IM		linked with CLRQ2
CLRQ4	Instrument Access		AAAI,AM, IM		
CLRQ5	Configuration Logging		AAAI,AM, DMM, MM, IM		
CLRQ6	Instrument Monitoring	CLRQ4	UI,AAAI,AM, MM, IM		Uses CLRQ4
CLRQ7	(Parameter) Visualisation	Similar to CLRQ6	AAAI,AM, MM, IM		
CLRQ8	(Real-Time) (Parameter/Data) Visualisation		AAAI,AM, MM, IM		May require data streaming analysis
CLRQ9	Experiment		AAAI, AM, MM, IM, WM		

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
CLRQ10	Process Control		AAAI, WM, MM, IM		
CLRQ11	Data Collection		AAAI, AM, MM, IM		A meta requirement. It contains a lot of features. It should be split into sub-requirements. Will affect provenance and preservation catalogues.
CLRQ12	(Real-Time) Data Collection	CLRQ11	AAAI, AM, MM, IM		Specialization of CLRQ11
CLRQ13	Data Sampling		AAAI, AM, DMM, IM		Can be modeled as real time integration with a statistical program. Other options could be: i) enable researchers to deploy and run their statistical programs on eVRE, ii) eVRE provides a set of statistical libraries Need to be defined. e-RIs responsible
CLRQ14	Noise Reduction				e-RIs responsible
CLRQ15	Data Transmission		AAAI, IM, RM		e-RIs responsible
CLRQ16	Data Transmission Monitoring		AAAI, IM, RM		e-RIs responsible
CLRQ17	Data Cataloguing		RI, AAAI, DMM, MM, IM		
CLRQ18	Data Publication		LDM, AAAI, MM		This requirement seems more generic

10.2.4 Data Processing



UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 10 Data Processing

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
PRQ1	Resources Annotation		AAAI, RM, MM		
PRQ2	(Data) Annotation		AAAI, RM, MM		
PRQ3	Metadata Harvesting		AAAI, MM, IM		
PRQ4	Resource Registration		AAAI, MM, IM		

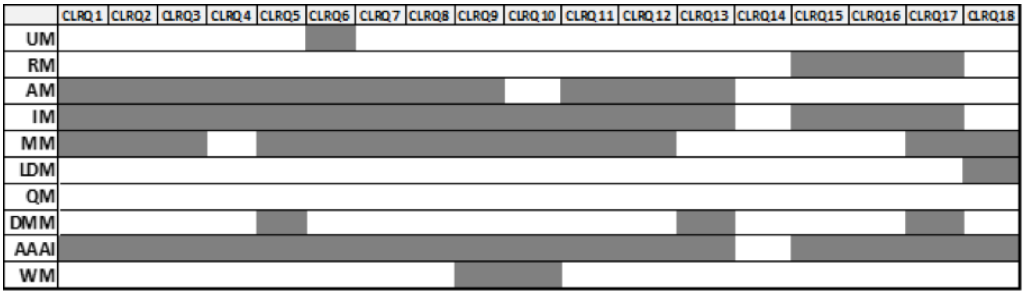
Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
PRQ5	(Metadata) Registration	PRQ4	AAAI, MM, IM		
PRQ6	Data Conversion		AAAI, MM, IM LDM		Affects provenance and preservation catalogues
PRQ7	Data Compression		AAAI, MM, IM, WM		
PRQ8	Semantic Harmonisation		AAAI, DMM, MM		Consider http://www.w3.org/2005/Incubator/ssn/ssnx/ssn which is being formalised at http://w3c.github.io/sdw/ssn/
PRQ9	Data Discovery and Access	PRQ10, PRQ12, PRQ13	AAAI, MM, IM (QM)		Access is not our responsibility. Discovery is accomplished through metadata and the catalogue
PRQ10	Simple search	Uses: PRQ12	QM, AAAI , MM	Search into all / given attributes of CERIF metadata	Use opensearch.org for the query mediator. It is an interface. We should also guide users how to write queries (e.g., suggest queries) Simple search typically addresses the catalogue, other queries focus on the data themselves. The e-RI will also have their own search engines. How this integration will take place. Merging the results is not easy! The user may be asked how to complete the merging process (steer the process)
PRQ11	Multiple format support	Uses: PRQ12	QM, AAAI, MM, IM(DMM)	Metadata for cfResPubl (doc), cfMedium,(any other media), cfResProd	

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
				(datasets)	
PRQ12	Cross searching		AAAI, MM, IM (DMM, QM)	Based on mappings CERIF - X	The query manager must know the query language of datasets involved and the mapper needs to know the result data formats, these info are stored on the Resource Manager
PRQ13	Advanced search	Uses: PRQ12	AAAI, MM, IM (DMM, QM)	Search into all / given attributes of CERIF metadata	There are 3 levels of queries, existential (generic), contextual (catalogue) and queries on the data. The latter is very difficult, we may decide that we only give connections to the underlying search facilities of eRIs.Or we can go further and suggest querying, transformation, merging etc facilities. This is an open question for now
PRQ14	Spelling checking		AAAI, QM, MM	Use the thesaurus stored in the semantic layer	Need to decide the best way to implement Language Vocabularies (consider opensearch.org)
PRQ15	Query suggestion		QM, AAAI, MM	Suggest metadata values 'similar' to xx from attributes stored in CERIf entities	similar searches to the ones the user places

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
PRQ16	Query filter		AAAI, QM, MM	NA	
PRQ17	Datasets viewing	PRQ18	AAAI, AM, IM		we focus only on metadata, not the actual data.We could offer an Amazon like interface, where the different facets (topics or subjects) of the data available are presented
PRQ18	Datasets pre-selection		AAAI, UM, MM		Only authorized-users can set up the list of datasets available in a department
PRQ19	Dataset customization		AAAI, UM, MM		The user (or administrator) narrows the search to specific types of datasets only (notice that we only focus on metadata here). This is different from pre-selection,as preselection is an one-time process, whereas customization is specific to each query.
PRQ20	Linking external resources		QM, AAAI, MM, IM		datasets given by sources not officially connected to the VRE, as long as they satisfy certain baseline requirements
PRQ21	Data Assimilation		AAAI, WM, AM, MM, IM(DMM)		
PRQ22	Data Analysis		AAAI, AM, MM,, IM(DMM), WM		
PRQ23	Data Mining		AAAI, AM, MM, IM		
PRQ24	Data Extraction		AAAI, AM, MM, IM		
PRQ25	Scientific Modelling and Simulation		AAAI, AM, MM, IM(DMM), WM		

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
PRQ26	(Scientific) Workflow Enactment		AAAI, WM, AM, MM, IM(DMM)		
PRQ27	(Scientific) Visualisation		AAAI, AM, IM		
PRQ28	Data Processing Control		AAAI, WM, AM, MM, IM(DMM)		
PRQ29	Data Processing Monitoring		AAAI, WM, AM, MM, IM		
PRQ30	API		AAAI, IM		
PRQ31	Dataset Download		AAAI, IM		
PRQ32	Quality Rating		AAAI, MM		
PRQ33	Peer Review		AAAI, MM, AM		
PRQ34	Data tag		AAAI, MM		
PRQ35	Data Backup		AAAI, IM		eRIs responsibility

10.2.5 Data Optimization

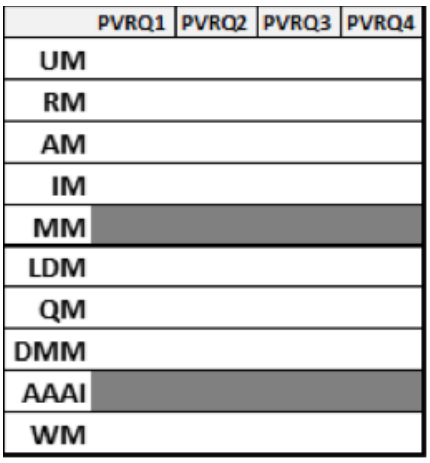


UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 11 Data Optimization

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
ORQ1	Large datasets processing		AAAI, RM, AM, IM, WM		
ORQ2	Processing parallelisation		AAAI, RM, AM, IM, WM		
ORQ3	Real time processing		AAAI, IM(Adapter), WM		
ORQ4	Data Compartmentalization		AAAI, RM, AM, IM, WM, MM		

10.2.6 Data Provenance

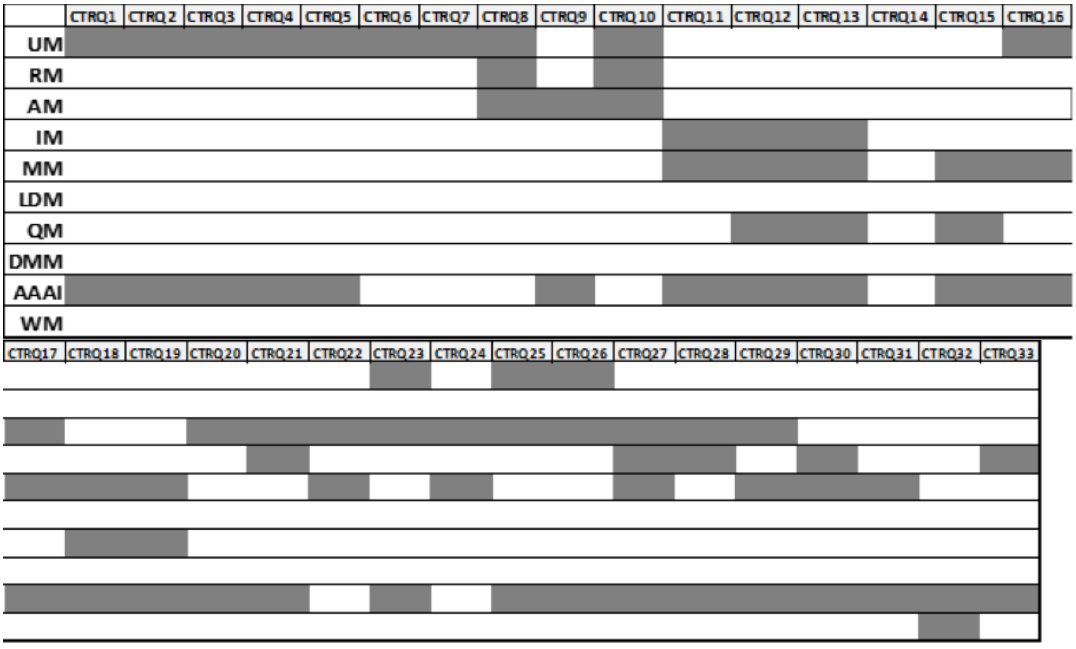


UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 12 Data Provenance

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
PVRQ1	Data Provenance		AAAI, MM		
PVRQ2	Data acquisition information		AAAI, MM		
PVRQ3	Data curation information		AAAI, MM		
PVRQ4	Data publication information		AAAI, MM		

10.2.7 Collaboration, Training and Support



UM - User Manager, RM - Resource Manager, AM - App Manager, IM - Interoperability Manager, MM - Metadata Manager, LDM - Linked Data Manager, QM - Query

Table 13Collaboration, Training and Support

Req. ID	Description	Rel. with other req.	Components involved	CERIF entities	Notes and Comments
CTRQ1	Login		UM, AAAI	cfPerson for user + some profile information as role in cfPers_Srv, cfPers_Lang	CTRQ2 and CTRQ3 seem special cases of CTRQ1
CTRQ2	Continuous access	CTRQ3	UM, AAAI	Keep information linked to cfPerson (datasets, services,…) with timestamp?	Ubiquity, availability (both non-functional requirements) and multi-channel (functional). Not across devices, but stay connected forever.For example, if connection fails, the session should be the same after we reconnect. This means we need to maintain in our servers all users' sessions (scalability issues)
CTRQ3	Single login	CTRQ1	UM, AAAI	cfPerson for user +profile information	Aka Single sign-on

CTRQ4	Interface customization		UM, AAAI	cfPerson for user + profile information	Customize the UI of each user, based on what activities she frequently performs. Customization, localization and personalization, Accessibility is also important
CTRQ5	Wizard configuration	GRQ4	UM, AAAI	cfPerson for user + profile information	
CTRQ6	User instruction		UM	cfPerson for user + cfPers_Medium , cfPers_ResPubl (doc)	This aspect will include not only tutorials at the VRE4EIC level, but also at the eRI level. This will give incentives for the researchers to generate and upload their own tutorials
CTRQ7	Multilingual interface		UM	cfLang / multilingual attributes of all entities	CTRQ7 could be included in CTRQ5
CTRQ8	Update alert	CTRQ10	SM(UM,...)		But “activity streams” is much more potent.And it is important for interoperability
CTRQ9	Online dataset editing		AM, AAAI	Link to cfFacility (the RI) that has the cfResProd (dataset), or to a given cfServ (service)	The tool has to be provided by some eRI and the users only access the datasets of that eRI. We send the VRE users there.

CTRQ10	Notification	CTRQ8	SM(UM,...)	cfPerson for user + profile information	There is a subscription mechanism. Consider "activity streams" for this
CTRQ11	Additional services interfaces		AAAI, MM, IM	cfServ	A use case for the VRE system administrator.
CTRQ12	Search for funding	Similar to: PRQ10, PRQ11, PRQ13	QM, AAAI, MM, IM		We assume there is a person populating our catalogue with metadata about project calls. In this case, calls are a resource as any other.
CTRQ13	Funding proposal	Similar to CTRQ12	AAAI, MM, QM, IM		Only the proposals accessible by the authors and the successful ones
CTRQ14	Electronic funding bid				This is irrelevant for us! There are already complex systems for this
CTRQ15	Funding body information		QM,AAAI , MM		Information about name and address of the body or more complex, such as policies? All probably, even though is difficult to implement
CTRQ16	Funding alert	Similar to: CTRQ8, CTRQ10	AAAI, UM, MM		A standard query executed periodically
CTRQ17	Research team setup		AAAI, MM, AM		

CTRQ18	Finding collaborators	Similar to: PRQ10, PRQ11, PRQ13	AAAI, QM, MM		
CTRQ19	Expertise finding	Similar to: PRQ10, PRQ11, PRQ13, related to CTRQ21	AAAI, QM, MM		These requirements are not only simple queries, they are like recommendation systems. Decision: no recommendation
CTRQ20	Forum tool		AAAI,AM		just a simple message exchange system suffices at this point, plug a tool for this
CTRQ21	SNS integration		AAAI, AM, IM		There are simple APIs we can integrate in our system that give twitter buttons, etc
CTRQ22	Group newsletter		AM, MM		
CTRQ23	Meeting organizer		AAAI, AM, UM		Integrate a tool
CTRQ24	Digest email		AM, MM		
CTRQ25	Teleconferencing		AAAI, AM, UM		Considering the integration with a teleconferencing suite
CTRQ26	Instant message		AAAI, AM, UM		Integrate a tool
CTRQ27	Project monitoring		AAAI, AM, IM, MM		A tracking/ticketing system for the project, like dates, deliverables,etc. Integrate a tool

CTRQ28	Computing resource connection		AM, AAAI, IM		
CTRQ29	Education support	CTRQ6	AM,MM,AAAI		Does this require integration with MOOC platforms (in the architecture)?
CTRQ30	Financial information		AAAI, IM, MM		A billing component is needed. Not everyone could use for free certain components
CTRQ31	Accounting		AAAI, MM		
CTRQ32	Workflow engine		AAAI, WM		
CTRQ33	API		AAAI, IM		

10.3Conceptual components: Interface Descriptions

10.3.1 User Manager Interfaces

10.3.1.1 User Manager: User management interface description

Table 14 User Manager Interface

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
createUserProfile	user:UserProfile	ReturnValue	UserProfile	Creates a user profile
updateUserProfile	userId:String, user:UserProfile	ReturnValue	UserProfile	Updates the profile userId
removeUserProfile	userId:String	ReturnValue		Deletes the profile userId

getUserProfile	userId:String	UserProfile	UserProfile	Retrieves the profile userId
getUserProfile	creds:UserCredentials	UserProfile	UserProfile	Gets the profile with provided credentials
getUserProfile	query:UserQuery	UserProfile[0..*]	UserProfile	Gets a list of profiles (wildcards allowed in query).

10.3.1.2 User Manager: Notification management interface description

Events related to RI resources are captured via Resource Manager and registered in Metadata Manager. The User Manager uses the Metadata Manager to check status changes.

Table 15 Notification Management Interface

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
subscribeEvent	userId:String, events:EVREEvent[0..*]	ReturnValue	EVREEvent	Subscribes to a list of events
checkEvent	userId:String, eventId:String	ReturnValue		Returns the status of the specific event
checkEvents	userId:String	ReturnValue[0..*]		Returns the status of all subscribed events
getSubscribedEvents	userId:String	EVREEvent[0..*]	EVREEvent	Returns list of events subscribed by userId

10.3.1.3 User Manager: login interface description

The idea is that VRE4EIC authentication services could be based on scoped credentials assigned to a User or an Entity and controlled by authenticators (https://en.wikipedia.org/wiki/Authenticator).
Please note that we are talking of the authentication process between an eVRE user and the eVRE system, it will 'wrap' the protocols adopted by VRE4EIC AAAI infrastructure.
The scoping of the credentials must be enforced jointly by a User Agent implementing the VRE4EIC authentication API and an authenticator that holds the credential, by constraining the availability and usage of credentials.
Scoped credentials must be located on authenticators, which can use them to perform operations subject to user consent.

According to outcome of D2.1 we should have two types of authenticator:

- Authenticators located in the same device (e.g., smart phone, tablet, desktop PC) as the user agent is running on.
- Authenticators operate autonomously from the device running the user agent, and accessed via network or other protocols. This last part is needed mainly to implement requirements about external instruments or devices (DRQ2, DRQ4, DRQ5…)

To implement this behavior we designed three interfaces: the *login* and the *Authenticator Management* interfaces provided by the User Manager component and the *Authentication* interface provided by a component called AuthenticatorApp that implements the client side functionalities of the authentication mechanism.

Table 16 Login Management Interface

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
login	creds:UserCredentials	ReturnValue		
login	authenticatorId:String, deviceId:string	ReturnValue		Uses external Authenticator (see: UC 21) for login
logout	userToken:String	ReturnValue		Signs out the user

10.3.1.4 User Manager: Authenticator Management interface description

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
registerAuthenticator	creds:UserCredentials, authenticatorId:String	ReturnValue		Register external authenticator
removeAuthenticator	creds:UserCredentials, authenticatorId:String	ReturnValue		Remove external authenticator

10.3.1.5 AuthenticatorApp: Authentication interface description

Table 17 Authenticator App Interface

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**

authenticate	reqInfo:info	ReturnValue		The reqInfo contains information about device requesting authentication and the eVRE service that has been requested.
synchCredentials	authenticatorId:String, creds:UserCredentials[1..*]	ReturnValue		Synchronize credentials with the eVRE

10.3.2 Resource Manager: *Resource management* interface description

Table 18 Resource Manager Interface

Operation	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
addResourceProfile	resource:ResProfile	ReturnValue	ResProfile	
updateResourceProfile	resourceId:String, resource:ResProfile	ReturnValue	ResProfile	
removeResourceProfile	resourceId:String	ReturnValue		
getResourceProfiles	query:ResQuery	ResourceProfile[0..*]	ResProfile	Wildcards can be specified in the query
getResourceProfile	resourceId:String	ResourceProfile	ResProfile	
isResourceAvailable	resourceId:String	ReturnValue		Checks if the resource is currently available in the RI, if RI provides this service. Depending on the RI service the return value can include info like:

				ETA for downtime, current use rate etc

10.3.3 Workflow Manager

10.3.3.1 Workflow Manager: Wf Repository Access interface

Table 19 WF Repository Access Interface

Operation	Parameter-list	Return type	Map to CERIF entity	Notes
getWF	idWF:String	WorkFlowDescription	WorkFlowDescription	
getWFs	idWFs:String[1..*]	WorkFlowDescription[0..*]	WorkFlowDescription	
getWFs	wfQuery	WorkFlowDescription[0..*]	WorkFlowDescription	Wildcards allowed

10.3.3.2 Workflow Manager: Wf management interface

Table 20Interface WF Config Manager

Operation	Parameter-list	Return type	Map to CERIF entity	Notes
createWF	wf: WorkFlowDescription	ReturnValue	WorkFlowDescription	
updateWF	idWF:String	ReturnValue		
deleteWF	idWF:String	ReturnValue		
getWFs	idWFs:String[1..*]	WorkFlowDescription[0..*]	WorkFlowDescription	
executeWF	idWF:String	ReturnValue		

stopWF	idWF:String	ReturnValue		
pauseWF	idWF:String	ReturnValue		
getWFStatus	idWF:String	ReturnValue		

10.3.4 MOM Component

10.3.4.1 MOM: Topic interface

Table 21 MOM Topic Interface

Operation	Parameter-list	Return type	Map to CERIF entity	Notes
getTopics	query:TopicQuery	Topic[0..*]	Topic	Wildcards allowed
createTopic	topic:Topic	ReturnValue	Topic	
updateTopic	topicId:String	ReturnValue		
removeTopic	topicId:String	ReturnValue		

10.3.4.2 MOM: Message Management interface

Table 22 MOM Message Interface

Operation	Parameter-list	Return type	Map to CERIF entity	Notes
addMessage	msgs:Message[1..*]	Return values		Routing info stored in messages
getMessages	topicIds:String[1..*]	Message[0..*]	Message	
getMessages	topicId:String[1..*], query:MessageQuery	Message[0..*]	Message	Wildcards allowed

getTopics	query:TopicQuery	Topic[0..*]	Topic	Wildcards allowed
subscribeTopic	topicIds:[1..*]	ReturnValue		
removeSubscription	topicIds:[1..*]	ReturnValue		
checkTopic	topicIds:[1..*]	ReturnValue		

10.3.5 Metadata Manager Interfaces

10.3.5.1 Metadata Manager Get Metadata Interfaces

Table 23Metadata Manager Get Metadata Interface

Operation Name	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
getResourceMetadata	String: resourceURI Collection<String> graphspaces	Collection<Triple> results		The method takes as input the URI of a resource, and the corresponding graphspaces and returns the contents of the metadata catalogue as a collection of triples (i.e. <S, P, O>, where S=subject, P=predicate and O=Object).
getResourceMetadataUsingType	String: resourceURI Collection<String> graphspaces Enum: metadataType	Collection<Triple> results		The method is similar in spirit with *getResourceMetadata* method, however it also contains a type parameter to specify the exact metadata type that is requested (i.e.. owner of a

				resource).
searchForMetadata	Collection<String> queryTerms Collection<String> graphspaces	Collection<Triple> results		The method takes as input a set of queryTerms and the graphspaces to search for and searches in the metadata catalogues for these terms. Finally it returns the results as a collection of triples.

The Metadata Manager also contains the ManageMetadata interface that allows users to add new information or to update existing information from the metadata catalogues. This interface is exploited from almost all the sub-components of the SystemManager (ResourceManager, UserManager, etc.). These components invoke the appropriate methods of the interface every time some entry needs to be updated; the rationale is that each Manager (from the System Manager) will update the appropriate metadata catalogue (i.e., the User Manager will be able to update only the metadata about users). However, in some cases the ManageMetadata can be used by particular users or agents (having an administrative role) for updating information in the metadata catalogues. Furthermore, the LDManager uses this interface for adding metadata information as regards to the publishing of Data also as Linked Open Data.

10.3.5.2 Metadata Manager: ManageMetadata interface

Table 24 Metadata Manager Manage Metadata Interface

Operation Name	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**

insertUpdateMetadata	String: resourceURI Enum: metadataType String: metadataValue String: graphspace	void		The method updates (or adds if such information does not exist) particular information about the metadata of a resource. The type of the metadata and the corresponding value are also given in the parameters list. The new triples is being added under the given graphspace.
deleteMetadata	String: resourceURI String: graphspace	void		This methods deletes information about a specific metadata resource from the given graphspace.

Apart from the interfaces of the Metadata Manager component, there is also the ThesaurusAPI interface, offered by the Thesaurus subcomponent, which is being used by the Query Manager and the UI components to retrieve suggested terms during search and spell checking. In addition, the ProvenanceAPI interface, offered by the Provenance Manager subcomponent, enables the storing of information about Queries and workflows and is being used by the Workflow Manager and the Query Manager.

10.3.5.3 Metadata manager: ThesaurusAPI interface

Table 25 Metadata Manager Thesaurus Interface

Operation Name	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
getSuggestedTerms	String: queryTerm	Collection<String> suggestedTerms		The method takes as input a queryTerm, and searches in the catalogue maintained by the Thesaurus component for suggested terms. For instance if the queryTerm is "inf" then it will return (among others) the terms "infrastructure", "inference", etc.

getSimilarTerms	String: queryTerm int: editDistance	Collection<String> similarTerms		The method takes as input a queryTerm and a value for the editDistance function to perform spell checking. It returns a ranked collection of potential corrections for the given query term.

10.3.5.4 Metadata Manager: ProvenanceAPI interface

Table 26 MetadataManager Provenance Interface

Operation Name	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
getProvenanceMetadata	String: resourceURI	Collection<Triple> results		The method takes as input the URI of a resource and returns all the available provenance metadata of the resource as a collection of triples.
getProvenanceMetadataUsingType	String resourceURI Enum: metadataType	Collection<Triple> results		The method is similar in spirit with *getProvenanceMetadata*, however the current one uses one more parameter for defining the type of the metadata that is requested, and returns the facts about provenance of the given resource as a collection of triples.

updateProvenanceMetadata	String: resourceURI Enum: metadataType String: metadataValue	void		The method updates (or adds if such information does not exist particular information about the provenance of a resource. The type of the metadata and the corresponding value are also given in the parameters list.

We should note that there is no direct connection between the Mapping Manager (subcomponent of the Model Mapper) and the Metadata Manager. The reason is that the Mapping Manager (as it is shown in the corresponding diagram) is being exploited by the Query Manager directly whenever it is required to perform mappings (over data or over a query).

10.3.6 Query Manager Component: SearchAPI interface

Table 27 Query Manager Interface

Operation Name	**Parameter-list**	**Return type**	**Map to CERIF entity**	**Notes**
searchSimple	Collection<String> queryTerms Collection<Pair<String,String>> preferences	Collection<Result> results		The method takes as input a set of query terms, and a set of preferences - expressed as key-value pairs (i.e. perform ranking, filtering, etc.)- and searches locally, and finally returns a collection of results.

searchSimpleWithinRange	Collection<String> queryTerms Collection<Pair<String,String>> preferences int: startOffset int: limit	Collection<Result> results		The functionality is similar in spirit with *searchSimple* method, with the only difference that it takes as input the startOffset and the upper limit, to support paging of results (i.e. starting from result 1 bring 100 results).
searchFederated	Collection<String> queryTerms Collection<String> dataSources Collection<Pair<String,String>> preferences	Collection<Result> results		The method takes as input a set of queryTerms, a set of data sources, and a set of preferences - expressed as key-value pairs- and performs a federated search over the given data sources, and finally returns a collection of results.

searchFederatedWithinRange	Collection<String> queryTerms Collection<String> dataSources Collection<Pair<String,String>> preferences int: startOffset int: limit	Collection<Result> results		The functionality is similar in spirit with *searchFederated* method, with the only difference that it takes as input the startOffset and the upper limit, to support paging of results (i.e. starting from result 1 bring 100 results).

10.3.7 Model Mapper Component

10.3.7.1 ModelMapperQuery interface

Table 28 Model Mapper Query Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes
transformQueryExpression	String: initialQuery String: targetSchema	String: queryExpr		This method takes as input a query and a description of the target schema and is responsible for transforming it so that it can be submitted to the target system. The method returns the expression of the query with respect to the target format.

It also has a **ModelMapperData** interface that exposes the functionalities as regards the data transformation, which is being used from the QueryManager component (whenever it is required to transform some data - i.e. results- and deliver them in an homogeneous way to the users/agents), the WorkflowManager and the SystemManager components.

10.3.7.2 Model Mapper: ModelMapperData interface

Table 29Model Mapper Data Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes
transformData	String: originalData String: targetSchema	String: transformedData		This method takes as input the textual description of some data a query and a description of the target schema and is responsible for transforming them with respect to the target schema. The method returns the textual description of the transformed data.

Finally the ModelMapper component has a MappingManager interface that contains all the functionalities as regards the manipulation of mappings.

10.3.7.3 Model Mapper Component: MappingManager interface

Table 30 Model Mapper Mapping Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes
getMapping	String: mappingID	String: mappingExpression		The method takes as input the ID of a mapping and returns the textual representation of the mappings (i.e. as an X3ML file)
addMapping	String: mappingID String: mappingExpression	void		The method takes as input a textual representation of a mapping (i.e. a X3ML [X3ML_Framework_IJDL_2016] file), and an ID and stores the mapping in the MappingManager catalogue.

updateMapping	String: mappingID String: mappingExpression	void		The method takes as input a textual representation of a mapping (i.e. a X3ML file), and an ID and updates an existing mapping in the MappingManager catalogue.
deleteMapping	String: mappingID	void		The method takes as input a textual representation of a mapping (i.e. a X3ML file), and removes the mapping from the MappingManager catalogue.

10.3.8 LDManager Component:

10.3.8.1 LD Manager PublishLDAPI interface

Table 31 LDManager Publish Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes
publishLinkedData	String: publicationSourceURI	void		This method takes as input the publicationSourceURI in which the data will be published (i.e a named graph uri)

Moreover, the published data are exposed through the **SPARQL-API** interface.

10.3.8.2 LDManager Component: SPARQL-API interface

Table 32 LD Manager Sparql Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes

querySPARQL	String: sparqlQuery Format: returnType	String: sparqlResults		The method takes as input a SPARQL query, and the type of the returned results (i.e. XML, JSON, HTML, etc.), it executes the query and returns the results with respect to the given return type.

10.3.9 AAAI Component interfaces

Table 33 AAAI Interface

Operation Name	Parameter-list	Return type	Map to CERIF entity	Notes
authenticateUser	creds:UserCredentials	UserProfile	UserProfile	Delegate to federated identity service
authorizedUser	creds:UserCredentials resourceId:String, operationType:String	Boolean		Return true if user in her current role is authorized to perform given operation on given resource
billUser	creds:UserCredentials resourceId:String, amount:Number	Invoice		Bill user for using amount units of a given resource
encryptData	encryption:Scheme plainData	EncryptedData		Encrypt data using given scheme. Note that in the actual implementation, we expect this function to be embedded in the components that need them