VRE4EIC

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

Deliverable D3.3

Building Blocks

Document version: 1.1

VRE4EIC DELIVERABLE

Name, title and organisation of the scientific representative of the project's coordinator:
Mr Philippe Rohou t: +33 4 97 15 53 06 f: +33 4 92 38 78 22 e: philippe.rohou@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-M24
Deliverable number:	D3.1
Deliverable title	Building Blocks
Contractual Date of Delivery:	September 30, 2017
Actual Date of Delivery:	September 30, 2017
Editor (s):	Carlo Meghini (CNR ISTI)
Author(s) & Participant(s):	Keith Jeffery (ERCIM) Cesare Concordia (CNR ISTI) Francesco Furfari (CNR ISTI) Theodore Patkos (FORTH ICS) Nikos Minadakis (FORTH ICS) Yannis Marketakis (FORTH ICS) Vangelis Kritsotakis (FORTH ICS)
Reviewer (s):	Keith Jeffery (ERCIM) Philippe Rohou (ERCIM)
Work package no.:	3
Work package title:	Architecture, VRE development, integration and scalability
Work package leader:	Carlo Meghini (CNR)
Version/Revision:	1.1
Draft/Final:	Final
Total number of pages:	40

What is VRE4EIC?

VRE4EIC develops a reference architecture and software components for VREs (Virtual Research Environments). This e-VRE 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 e-VRE provides a comfortable homogeneous interface for users by virtualising access to the heterogeneous datasets, software services, 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

This document contains description of the VRE4EIC project work and findings.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any responsibility for actions that might occur as a result of using its content.

This publication has been produced with the assistance of the European Union. The content of this publication is the sole responsibility of the VRE4EIC consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 28 Member States of the Union. It is based on the European Communities and the Member States cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors (http://europa.eu/).

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

The present report accompanies deliverable D3.3 “Software Building Blocks”, resulting from Task T3.3 “Development of Building Blocks to Fill Gaps” of the VRE4EIC project.

As the name of the tasks says, Task T3.3 has received as inputs the Reference Architecture (D3.1), specifying the components of the enhanced VRE (e-VRE for short), and the Gap Analysis (D3.2), indicating which components needs to be implemented to fill the gaps in existing VREs. Based on these inputs, as well on the non-functional requirements collected in D2.1, Task T3.3 has designed a Technical Architecture for implementing the Reference Architecture, and has implemented the components of the Technical Architecture indicated by the Gap Analysis. These components will be taken as input by Task T3.4 “Integration of Reference VRE and Enhanced Existing VREs” to be integrated into the architectures of the EPOS and the ENVRIplus VREs (Task 3.4), thereby enhancing these VREs.

D3.3 delivers the implemented components of the Technical Architecture, named “Building Blocks” (BBs from now on for short) for their role in the enhancement of the VREs. These BBs are:

- the Node service,
- the Metadata service, and
- the AAAI service.

Technically, each BB is a microservice of the VRE4EIC Technical Architecture and implements a major component of the Reference Architecture. The present report provides useful information about the three BBs that make up D3.3. In particular:

- Section 2 indicates where the source code of the BBs is to be found, and how such code is licensed.
- Section 3 recaps the VRE4EIC Reference Architecture.
- Section 4 describes the VRE4EIC Technical Architecture, highlighting how such a Technical Architecture has been derived from the Reference Architecture in terms of the principles that have been followed and the results obtained.
- Section 5 describes the design of the three microservices corresponding to the three BBs selected by the Gap Analysis.

In addition, Section 6 describes the design of the Graphic User Interface (GUI) of e-VRE. The GUI is a cross-service BB of the Technical Architecture, whose development has been included to specifically address usability aspects and to support the usage of the BBs for teaching and demonstration purposes. The design of the GUI is reported in this document, while its development will be part of D3.4, resulting from T3.4 (mentioned above) and due at Month 30.

Section 7 illustrates the development environment.

Section 8 concludes.

As it will be clear from Section 4, the three microservices included in D3.3 have been designed with two main goals in mind:

- As parts of the Technical Architecture, they interoperate by design.
- At the same time, their inclusion in a “foreign” architecture will create the minimal possible impact, each one of them being functionally self-contained by design.

Both these features will facilitate the follow-up activity in WP3, that is the integration of the BBs into the EPOS and the ENVRIplus architectures to be carried out by Task 3.4.

2 The source code

The GitHub repository of VRE4EIC project is at the the following URL:

https://github.com/vre4eic

This repository contains a number of projects; the source code of the three microservices described in this document can be found at the following links:

- Metadata Service: https://github.com/vre4eic/E-VREMetadataServices
- Node Service: https://github.com/vre4eic/NodeService
- AAAI Service: https://github.com/vre4eic/AAAI
- TelegramBot: https://github.com/vre4eic/TelegramBots

3 The Reference Architecture: an overview

At the general level, the VRE4EIC Reference Architecture 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 e-VRE:

- 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 e-VRE.
- The *Interoperability* tier, which deals with interoperability aspects by providing functionalities for: i) enabling application components to discover, access and use e-VRE resources independently from their location, data model and interaction protocol; ii) publishing e-VRE functionalities via a Web Service API; and iii) enabling e-VRE applications to interact with each other.
- The *Resource Access* tier, which implements functionalities that enable e-VRE components to interact with eRIs resources. It provides synchronous and asynchronous communication facilities

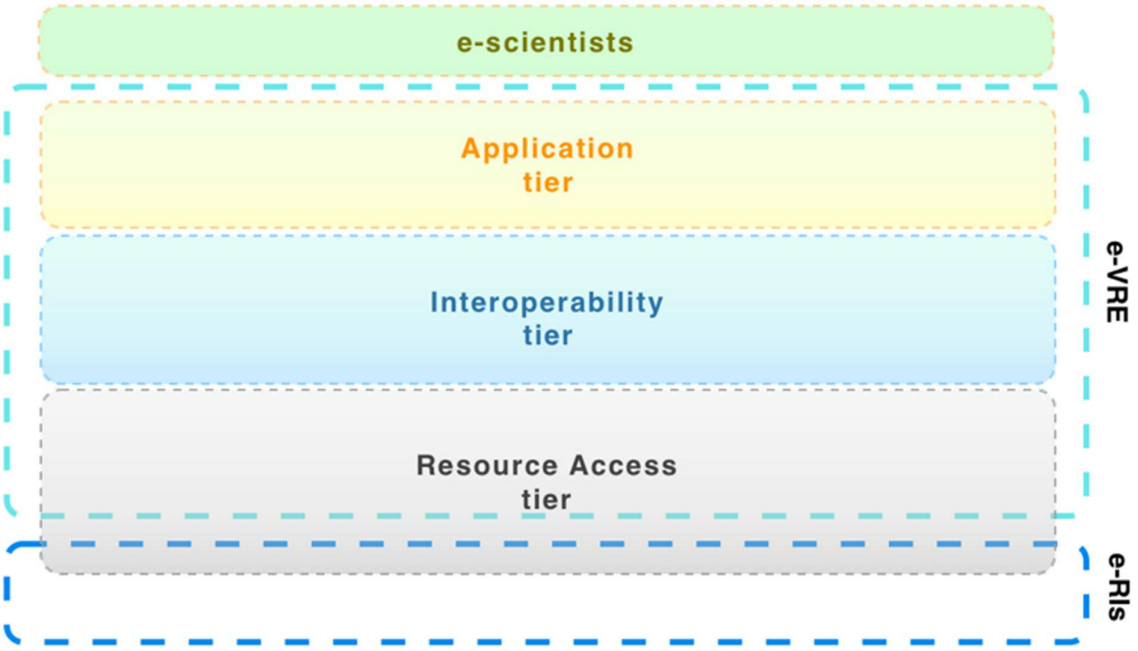


Figure 1 Architectural tiers in a VRE

Figure 1 depicts the logical tiers of e-VRE and shows their placement in an ideal space between the e-scientists that use the e-VRE and the e-RIs that provide the basic resources to the e-VRE.

Based on the analysis of the requirements, a set of basic functionalities have been individuated and grouped into six *conceptual components*:

- The e-VRE 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 e-VRE: access, create and manage resource descriptions, configure the e-VRE, plug and deploy new tools in the e-VRE 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 e-

VRE information space - i.e. the metadata concerning the e-VRE 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 e-VRE towards e-RIs. It implements services and algorithms to enable e-VRE to: communicate synchronously or asynchronously with e-RIs resources, query the e-RIs catalogues and storages, map the data models etc
- The **Authentication, Authorization, Accounting Infrastructure (AAAI**) component is the responsible for managing the security issues of the e-VRE 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.

Figure 2 shows how these six components are distributed on the 3-tier space introduced above.

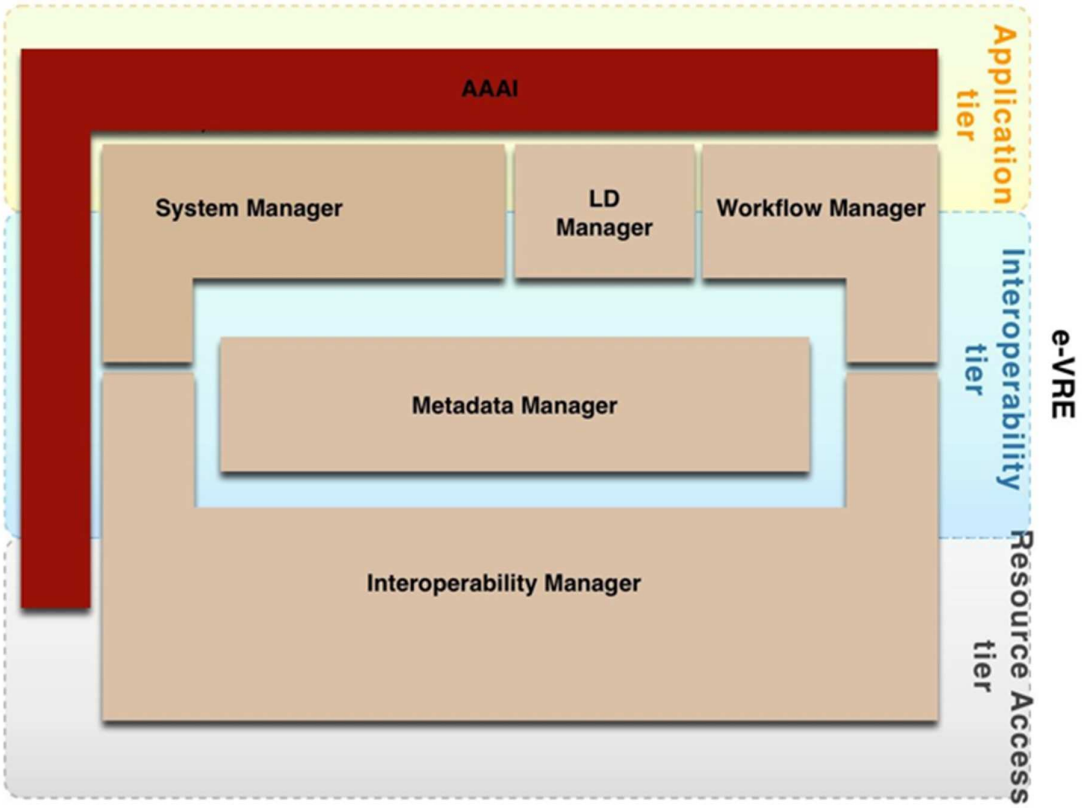


Figure 2 Conceptual components and logical tiers

4 The e-VRE technical architecture

In order to formally describe the architecture of the e-VRE we have created an UML component diagram describing the interfaces of every Conceptual component and indicating components relationships.

To create a readable Component diagram that clearly shows the architectural landscape of the e-VRE system, we have partitioned the functionalities of many components and assigned them to subcomponents for modularity. For instance, the Workflow manager's functionality has been partitioned into three areas, each assigned to a different subcomponent: the WF Configurator implements workflows definition functionalities, the WF executor implements execution functionalities and the WF repository component implements storage management for workflows.

In this document we will use the name of a Conceptual components to indicate the set of functionalities provided by the interfaces of its subcomponents (in case it has been partitioned).

The table below gives the names of the Conceptual components and subcomponents, and for each one indicates its role in the e-VRE system:

Component	**Sub-components**	**Description**
AAAI Component		Manages security, privacy and trust aspects of the e-VRE and its connections to the e-RIs
	Authentication	Manages user authentication for the e-VRE and connected e-RIs (single sign on), interfaces with external identity provider services.
	Authorisation	Manages user authorisations (role based 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 e-VRE 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
	e-VRE Web Services (e-VRE WS)	Enable external applications to interact with e-VRE
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 in e-VRE as Linked Open Data

	SPARQL Endpoint	Allows retrieving resources and services published by e-VRE 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 *define* and manage an e-VRE, e.g. specify the resources, specify the apps, and to deploy the defined VRE in the available resources.
	Node Manager (NM)	Implements the functionalities to deploy, manage and run an instance of e-VRE 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 e-VRE resources. It can be used also to embed applications such as Wiki or forums etc.

A key point in the definition of the technical architecture, has been the e-VRE 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* [vre4eic].
2. The e-VRE should be *applicable to different multidisciplinary domains* [vre4eic], i.e. it can be potentially used in every research domain.
3. The e-VRE functionalities should be exposed as services in a standardized way to enable developers to easily use them to develop new applications.
4. The e-VRE must provide *innovative standard software services to be retro-fitted to existing VREs to enhance them for their own domain purposes and for interoperability* [vre4eic].

From the architectural point of view the above requirements mean that the e-VRE must be easily expandable (by adding or replacing software components), highly modular (every 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] perfectly respond to 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 specific scientific domains
- *high cohesion:* components with related behaviour stand together *(i.e.,* related logic is kept in one service); changing the technology used to implement a microservice does not affect other building blocks.

In the following we describe the design principles we followed in defining the technical architecture:

- *Use Asynchronous communications.* We adopted an event-driven communication model, for Microservices interactions. The e-VRE 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 e-VRE (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[1]" [Newman] of the system when dealing with distributed processes.

[1] "*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].

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 e-VRE, 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 e-VRE). 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 e-VRE 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 e-VRE.

- *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 e-VRE 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: 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 e-VRE, the system provides functionalities to integrate external agents at *data level* (using the Metadata Manager functionalities), at *application level* (via e-VRE 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 **e-VRE WS** subcomponent: every microservice implements those Web Services endpoints enabling to access its functionalities. Then the e-VRE WS component is the composition of all microservices WS endpoints.

The following table 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.

Microservice	**Main component(s)**	**Auxiliary components**
Node Service	Node Manager, User Manager	- e-VRE WS (to enable *coexistence of different endpoints)*
App Service	App Manager	- e-VRE WS (to enable *coexistence of different endpoints)* - MOM, Adapters (to *avoid service coupling, to facilitate Choreography approach)*
Metadata Service	Metadata Manager	- e-VRE WS (to enable *coexistence of different endpoints)*, - Resource Manager, Data Model Mapper (*to facilitate Choreography approach)* - MOM, Adapter (to *avoid service coupling)*
LD Service	LD manager	e-VRE WS (to enable *coexistence of different endpoints)*
Workflow Service	Workflow Manager	- e-VRE WS (to enable *coexistence of different endpoints)* - MOM, Adapter (to *avoid service coupling)*
AAAI Service	AAAI Manager	e-VRE WS (to enable *coexistence of different endpoints)*
Query Service	Query Manager	- e-VRE WS (to enable *coexistence of different endpoints)*

		- Data Model Mapper (*to facilitate Choreography approach)* - MOM, Adapter (to *avoid service coupling)*

Figure 3 Microservices and e-VRE components

Note that, mainly to avoid coupled microservices, we broke the *don't repeat yourself* (DRY) rule by replicating components in different microservices, we are aware that this could create issues in developing and maintaining source code, but we think that an accurate development policy supported by a good infrastructure can help to manage this issue.

5 Implementing the technical architecture

In order to identify the components of the Reference Architecture, also named **building blocks,** 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 e-VRE 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

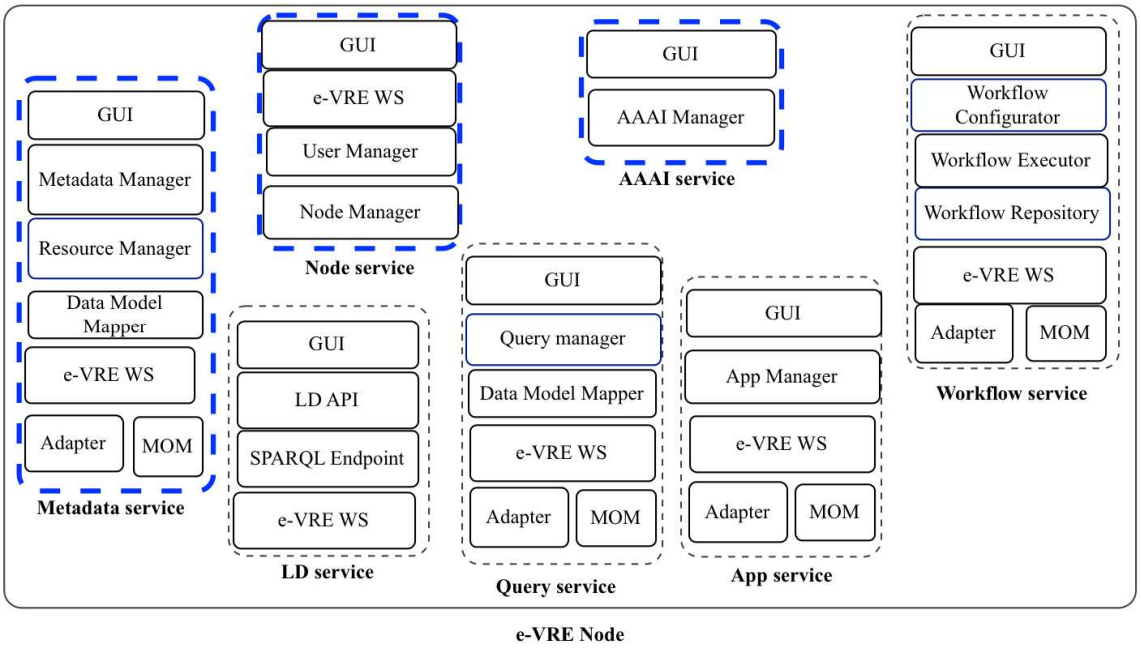


Figure 4 the e-VRE microservices

As the Figure 4 shows the two Building Blocks with highest priority belong to two microservices: the AAAI service and the Metadata service. The Node service, that implements the functionalities to deploy, manage and run an instance of e-VRE on a specific hardware, needs also to be implemented in order to be able to create and manage an installation of the architecture.

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 implementation details.

5.1 The Node Service

As described in table shown in Figure 3 the Node Service includes three components:

- Node manager
- User Manager
- e-VRE WS

The role of this service is to provide the basic functionalities to deploy, manage, run and possibly extend an instance of e-VRE.

5.1.1 The Node Manager

The Node Manager implements communication capabilities, coordination and configuration of the distributed services, and provides helper classes to the 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 e-VRE 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.

ZooKeeper is a high-performance coordination service for applications distributed over Internet. It exposes common services — such as naming, configuration management, synchronization, and group services — in a simple interface. *ZooKeeper* is used by several big companies, for example Yahoo uses it for doing leader election, configuration management, sharing, locking, etc.

Specifically, we used *Curator* [Curator] a library built on *Zookeeper* and developed by Netflix to simplify common distributed patterns called recipes i.e. leader election, distributed lock, barriers.

The system bootstrap process is based on *Curator* [Curator] which retrieves the e-VRE configuration. Administrators can use *Curator* interfaces to change system properties and to configure service availability.

Every node service gets access to the communication layer of Node Manager implementing the asynchronous communication facilities. The current implementation uses *Apache ActiveMQ* as codebase. It is a Message-Oriented Middleware (MOM) that supports well-defined JMS [JMS][JMS Brokers] interfaces and it is suitable to be used with different communication protocol like AMPQ [AMPQ] and MQTT [MQTT]. In particular, MQTT availability is useful for VRE enclosing many IoT (Internet of Things)devices producing real time dataset.

The Node manager is the component whose implementation abstracts from such legacy software by defining a communication layer that isolates concrete implementations, thus making relatively easily to change the message oriented middleware. It is organized in two packages: core.comm, core.messages.

The core.comm package contains a set of factory classes used to instantiate clients communicating each other with a publisher-subscribe pattern. Each factory is implemented with generic types representing the exchanged messages. Valid VRE messages are defined in the core.messages package. Currently there are specific messages for *authentication* functionalities, *metadata* exchange, *control* and *monitoring* messages. They are extensions of the base interface Message which extends Serializable interface.

There are other helper classes defined in the nodeservice.modules package used to ease the correct usage of communication pattern. From point of view of the components using modules, the actual communication pattern is completely hidden. For instance, in order to check if one user has been properly authenticated, components can instantiate AuthModule class and verify if the the user client

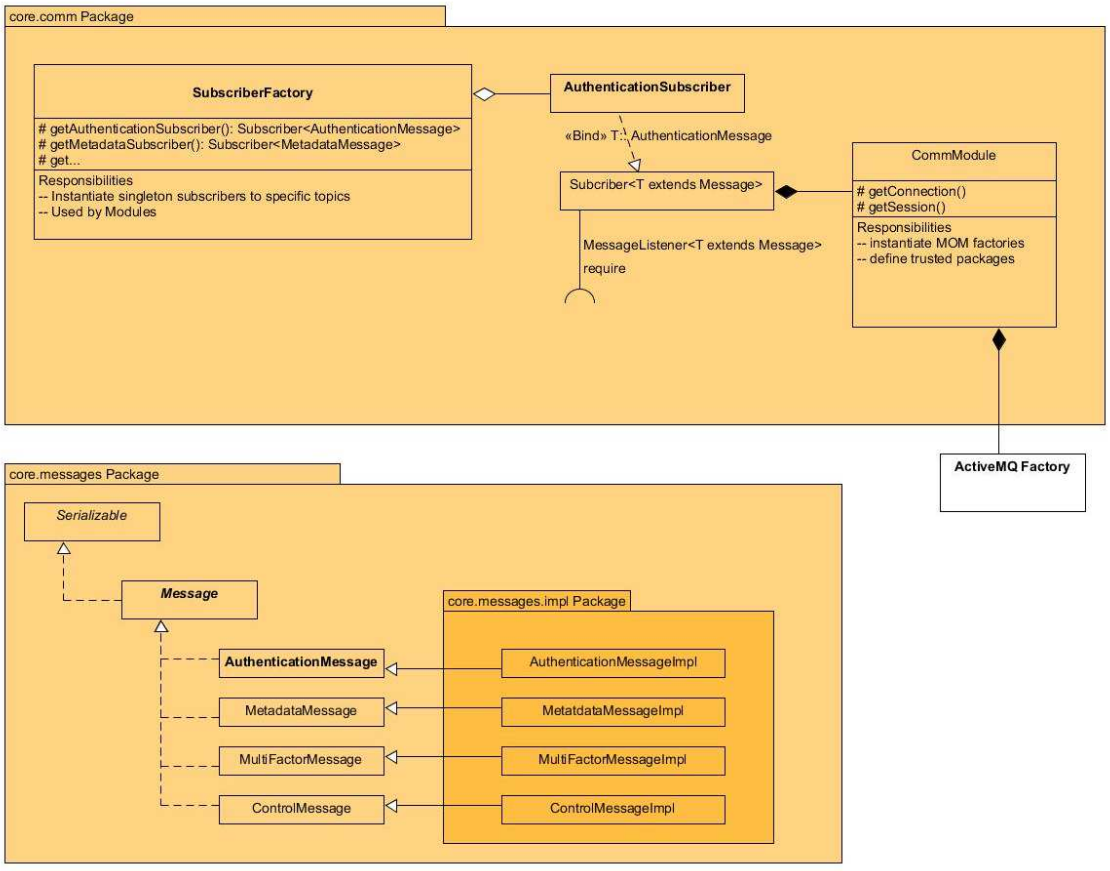
authentication token is valid by invoking the checkToken() method from the authentication module. Currently there are four modules developed for *authentication*, *metadata* exchange, *control* and *monitoring* of the resource usage.

In order to extend the functionalities of the node manager a three step process can be adopted:

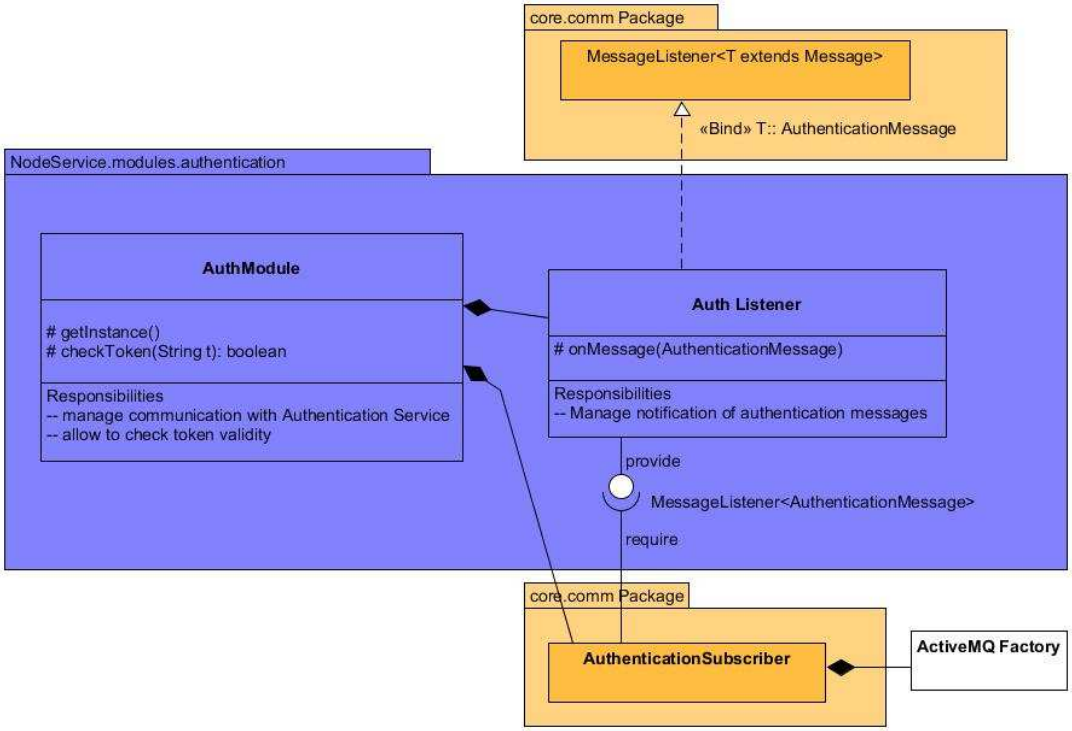
1. Message type definition
2. Factory class definition
3. Module design

Specific message types can be defined by extending the Message interface reported in the class diagram below (core.messages package). The Message interface implements the Serializable interface allowing data to be transferred as java bytecode, other data serialization protocol can be used (e.g. XML) however it is always convenient to derive from Message interface because many useful classes are based on the generic type extension <T extends Message> . If needed, a new communication channel can be defined by adding it to the list of available topics in core.Common class. For security reasons every MOM software allows to define the data type and serialization version that can be exchanged, by exporting the list of granted packages. During the development phase a TrustedAll marker has been used to speedup prototyping and testing (see CommModule class for details).

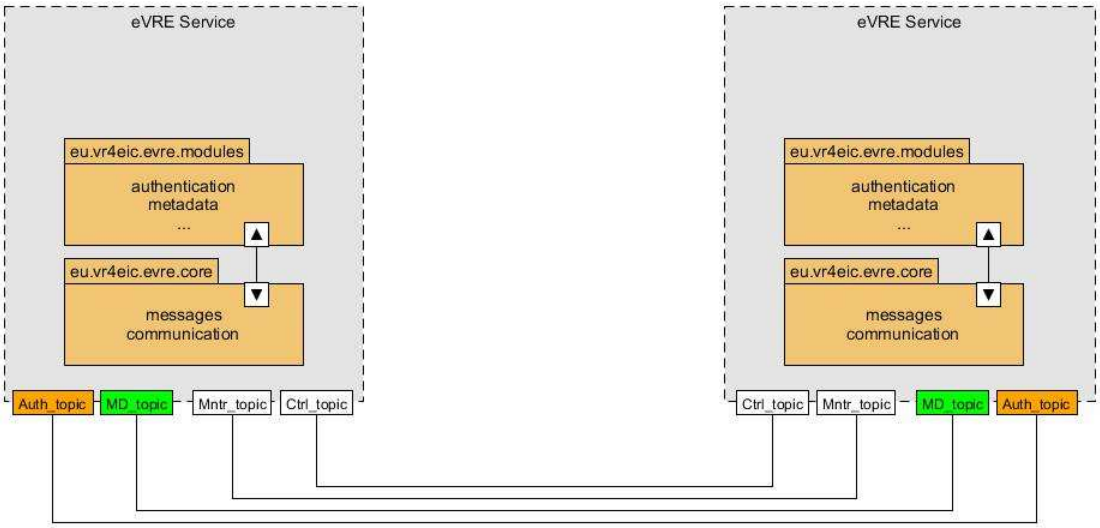
The second step consists of defining the factory class that instantiates and initializes the right communication pattern. As example in the class diagram we report the instantiation of a subscriber factory for the exchange of authentication messages. The generic class Subscriber<T> can be specialized binding it with the AuthenticationMessage interface. Subscriber<T> provides to retrieve the connection and session instance by means of the communication module that in turn delegates the legacy code, that's the ActiveMQ framework.



The third step is the definition of the appropriate module. In this case, the factories defined in the core package are used in the modules package to hide details of the communication. The authentication module implements a specialised authentication listener binding the generic class MessageListener<T> with the AuthenticationMessage type. The onMessage() callback of the listener stores the status of authenticated users and the module class provides housekeeping of the storage infrastructure. The only method exposed by the authentication module is the non-blocking checkToken() operation that returns the validity of the token.



The following image shows the packages that should be imported by generic e-VRE services to collaborate with each other. Communication channels are transparently created by the libraries.



5.1.2 The User manager

The main User Manager responsibilities are to manage User Profiles (add/update/remove user information, to set up users permissions, to manage users preferences, to configure users working environments) and to implement authentication interaction methods.

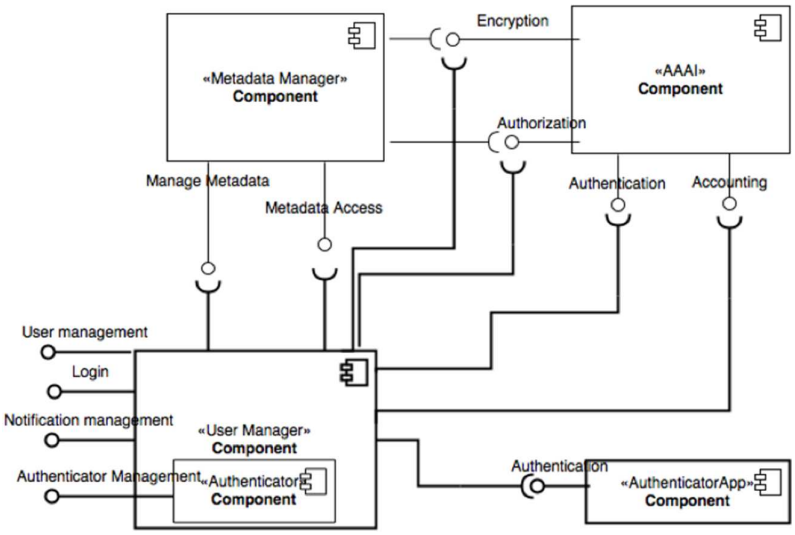


Figure 5 The User Manager UML Component diagram

This component has been developed from scratch, using Java technology. When a new user is created in the e-VRE 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.

The User Manager needs to interact with the AAAI service in order to execute two crucial operations: i) store credentials of users created in e-VRE 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 adopted for them a synchronous communication: the User Manager contacts the AAAI using an encrypted channel and waits until an answer has been received.

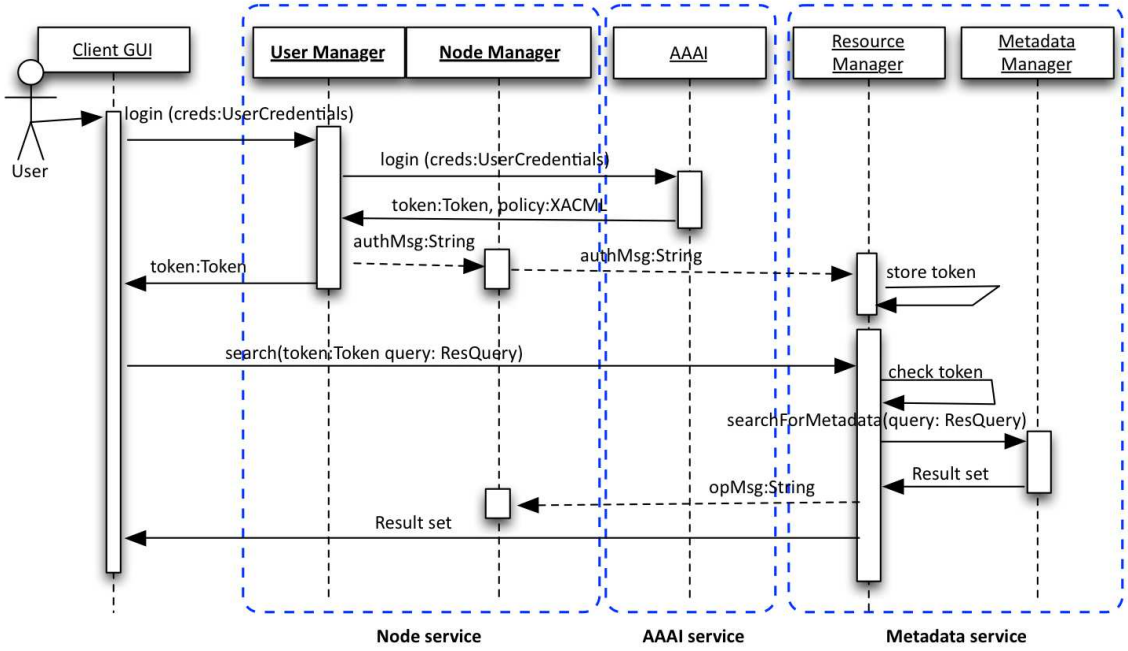


Figure 6 Example: authenticate and search metadata, the sequence diagram

The UML sequence diagram describes a simple Use Case where an external client authenticate a user and then executes a query on the metadata manager:

1. An external client asks the User Manager (UM) to authenticate a user by sending the user's credentials
2. The UM forwards the credentials to AAAI and wait for the answer
3. If the credentials are valid the AAAI returns a valid token (i.e. a temporary identifier that is assigned to a user in a session) and the policy associated to the valid user
4. The UM generates a message (in particular an AuthenticationMessage)and sends the message to the Node Manager
5. The Node Manager asynchronously sends the message to all registered services. It will be received by the Metadata Service which stores the message locally.
6. The UM sends back to the client an answer containing the token, this token will be used for every interaction with the e-VRE
7. The client executes the query by calling directly the Metadata service, in particular the Resource manager. It sends the token in the call.
8. The Resource Manager uses the local storage to check if the token is valid and if the operation is permitted for that token.
9. The Resource Manager executes the search by interacting with the Metadata Manager and, before sending the client the result set, publishes a message informing other services of the operation executed
10. The client receives the result set

Steps 1-6 of the sequence diagram shows how the *traditional* login/password based authentication method works in e-VRE.

However the User Manager implements also a two-factor authentication (2FA) method i.e. a method using a combination of two different factors to authenticate a user. In our implementation the User Manager doesn't know what channel is used to provide the second factor to the user; when a client requests a 2FA the User Manager checks if the user exists then publishes a message with the request,

the component that will manage the second authentication channel generates the second factor and sends it to the requesting user and to the User Manager. When the user provides the second factor to the User Manager it checks if it is correct and then validates credentials.

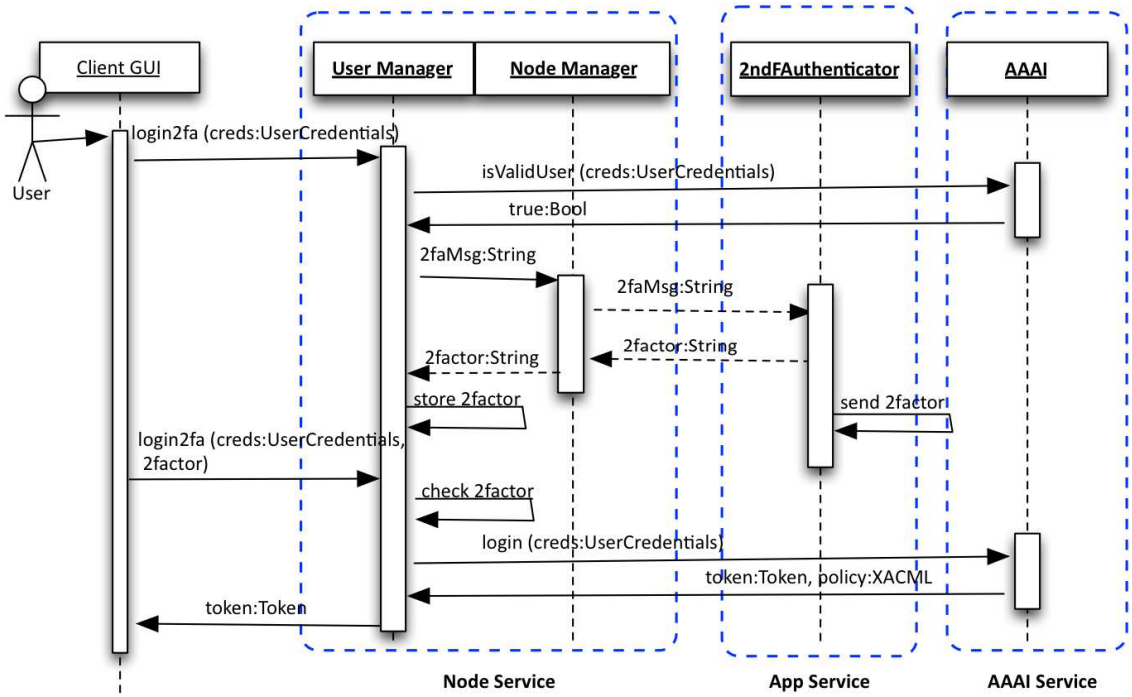


Figure 7 Example: 2FA sequence diagram

Note that using synchronous (request/response) connection to interact with AAAI, we have broken the *'avoid service coupling'* principle, this means that in some particular *retro-fitting* scenarios, the Node Service may be deployed along with other microservices in the hosting VRE.

5.1.3 The e-VRE WS

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

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

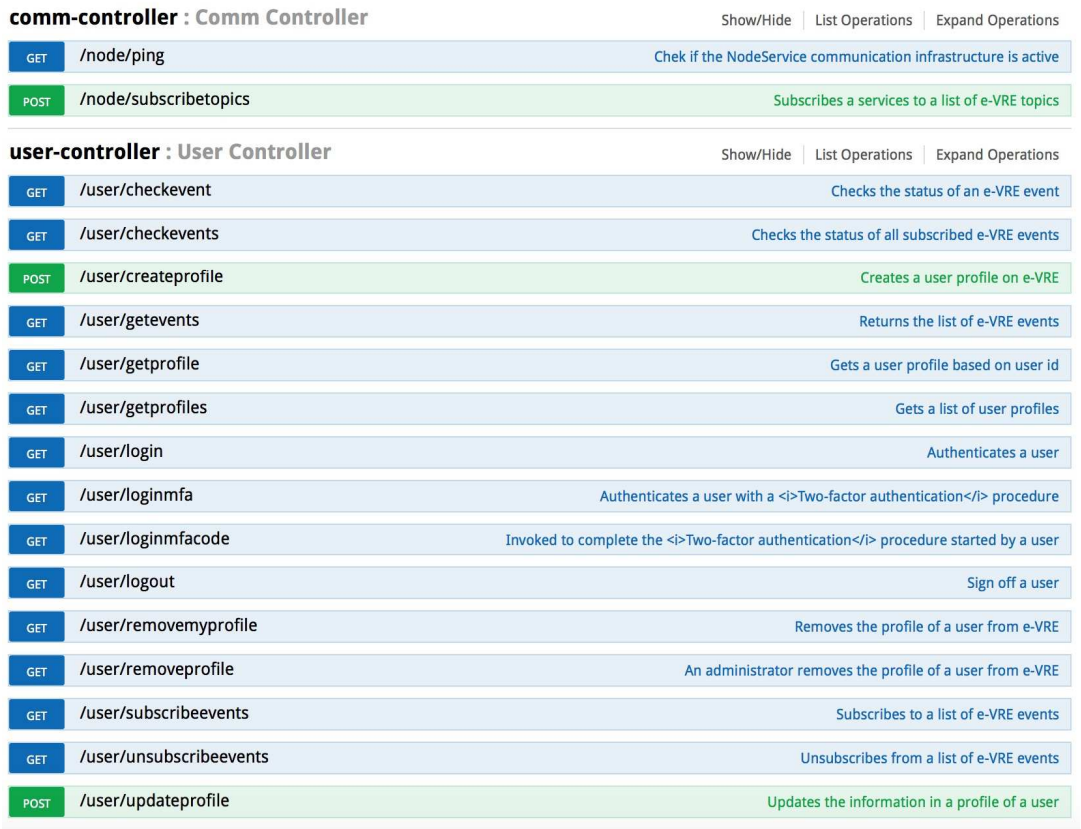


Figure 8 Example: 2FA sequence diagram

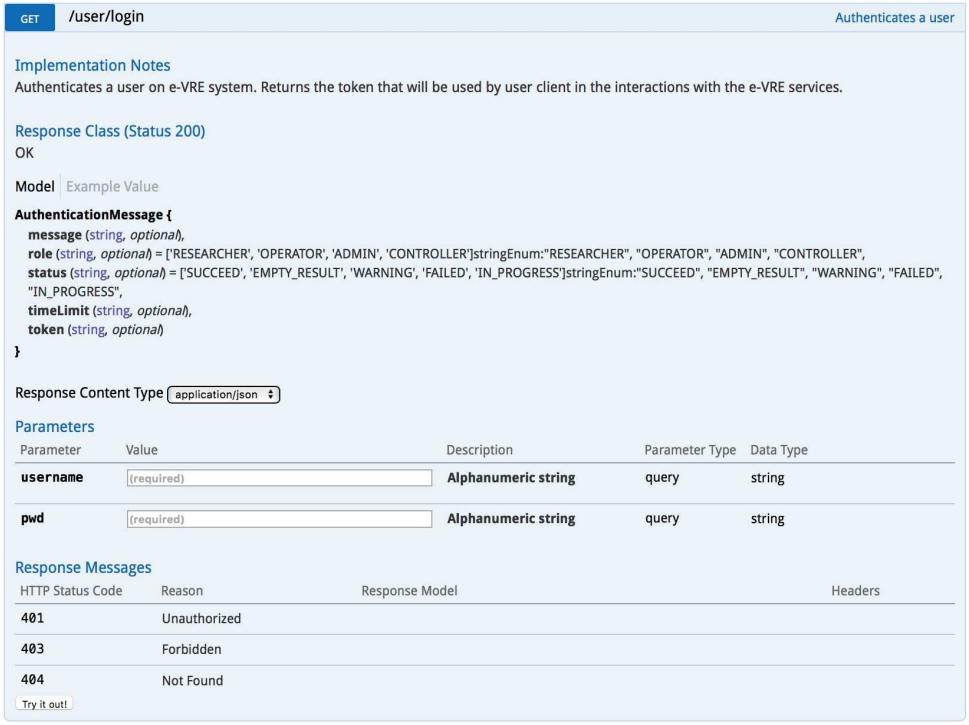


Figure 9 The documentation of the technical details for the Authenticator web service entry point

The figure 8 shows the list of entry points and as an example Figure 9 shows the technical details to use the e-VRE WS entry point for authentication.

5.2 The Metadata Service

The metadata service implementation contains 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 e-VRE 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.

5.2.1 Requirements

Triplestore The metadata service relies on the existence of a specific RDF triplestore used to store and manage the data which are converted into CERIF model expressed in RDF. For that purpose, we rely on Blazegraph[2], which is considered as an ultra-scalable, high-performance graph database with support for RDF/SPARQL APIs. It is available under the GPLv2 open source license.

Apart from the exceptional performance it can offer, Blazegraph is platform independent, supporting an executable jar deployment for getting started very quickly in an system which runs Java 7 or higher. A more detailed list of its features, provided APIs etc. can be found in the official wiki page[3] of Blazegraph.

3M As described below the 3M set of tools that is being utilized by the Data Model Mapper component, can be installed by using the public Linux Installer. Alternatively, the code has to be downloaded from github and the installation to be performed manually. In this case Java 7 or higher, Tomcat 7 and eXist db need to be installed before deploying the 3M wars.

5.2.2 Metadata Manager

As aforementioned, metadata manager implements all the functionalities, required for the management of the metadata repository. These functionalities are implemented as Restful web services illustrated in the component diagram of Figure 6 namely import, export, query and update.

Deployment Process When someone wants to deploy the services of this module on his own application server and infrastructure the first obvious step is the deployment of the WAR file. However, this is not sufficient as there are some configuration parameters which have to be set by the user in order the services to work correctly. These parameters are defined and set in a file which is essentially a properties file. This type of files is usually used as a good-practise way to initialize parameters which remain constant in the whole life-cycle of any application. Next, we analyse the parameters which are contained in the properties file:

- **triplestore.url:** the url of Blazegraph triplestore which is used for the metadata management
- **triplestore.namespace:** the name of the dedicated repository (i.e., namespace) used to store the metadata

[2] https://www.blazegraph.com
[3] https://wiki.blazegraph.com

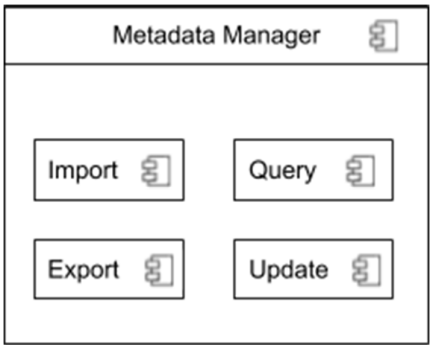


Figure 10 The Metadata manager component

5.2.3 The e-VRE WS

Each one of the components is implemented from the proper restful Web Services forming the corresponding Metadata restful API. The overview of the services is illustrated in Figure 7 and a more detailed documentation can be found in this link: https://app.swaggerhub.com/apis/rousakis/ld-services/1.0.0. For instance, Figure 8 shows the technical details to use the e-VRE query entry point along with the required parameters along with the expected returned responses.

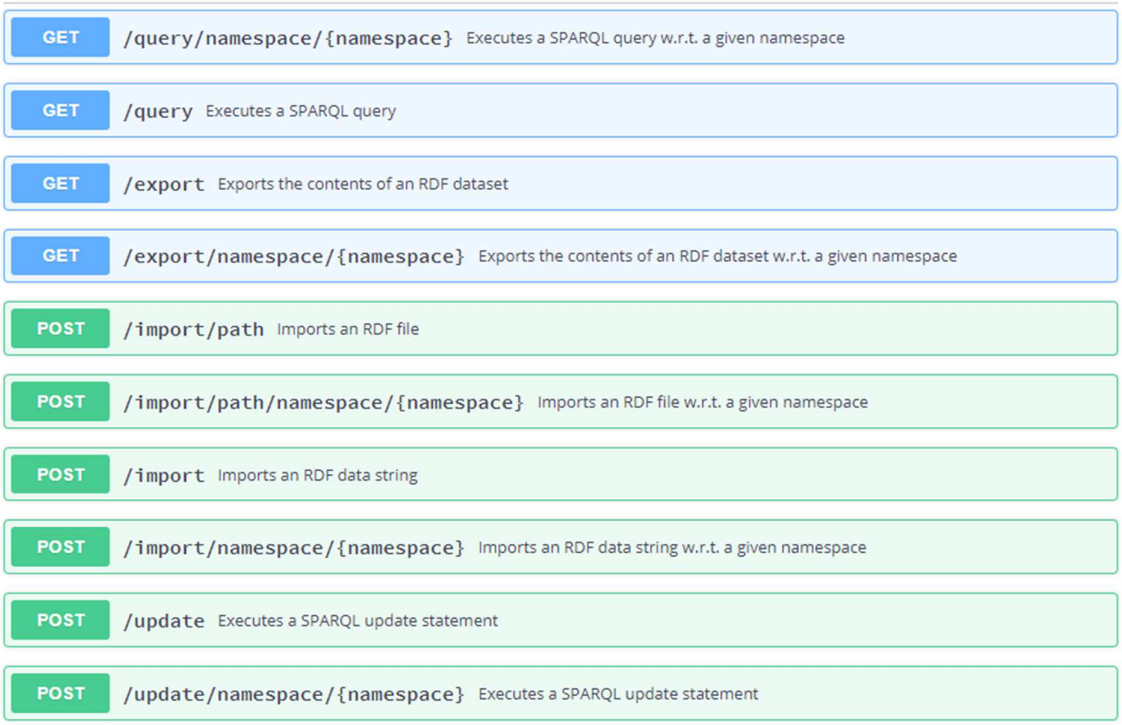


Figure 11 The list of entry points and descriptions of Metadata Service e-VRE WS.

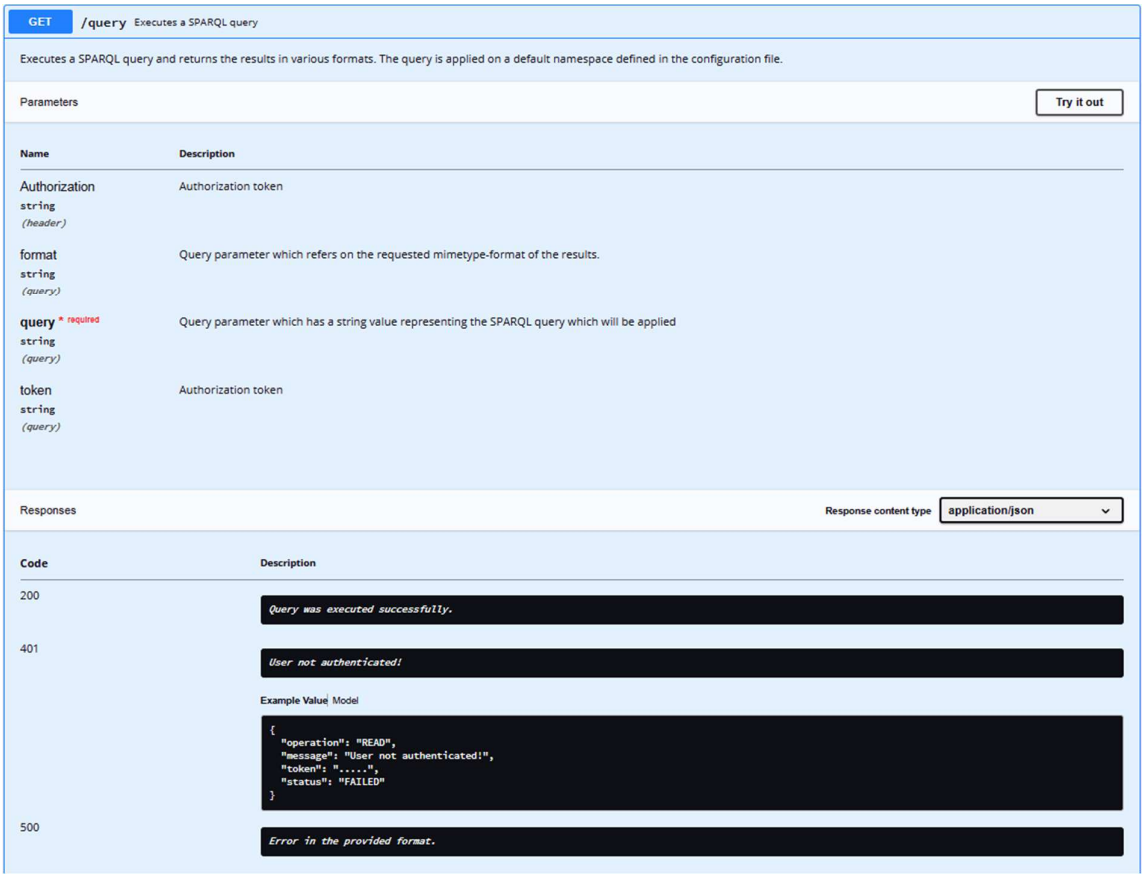


Figure 12 The documentation of the technical details for the Query web service entry point

5.2.4 The Resource Manager

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 pre-requisites 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.

5.2.5 Data Model Mapper (3M)

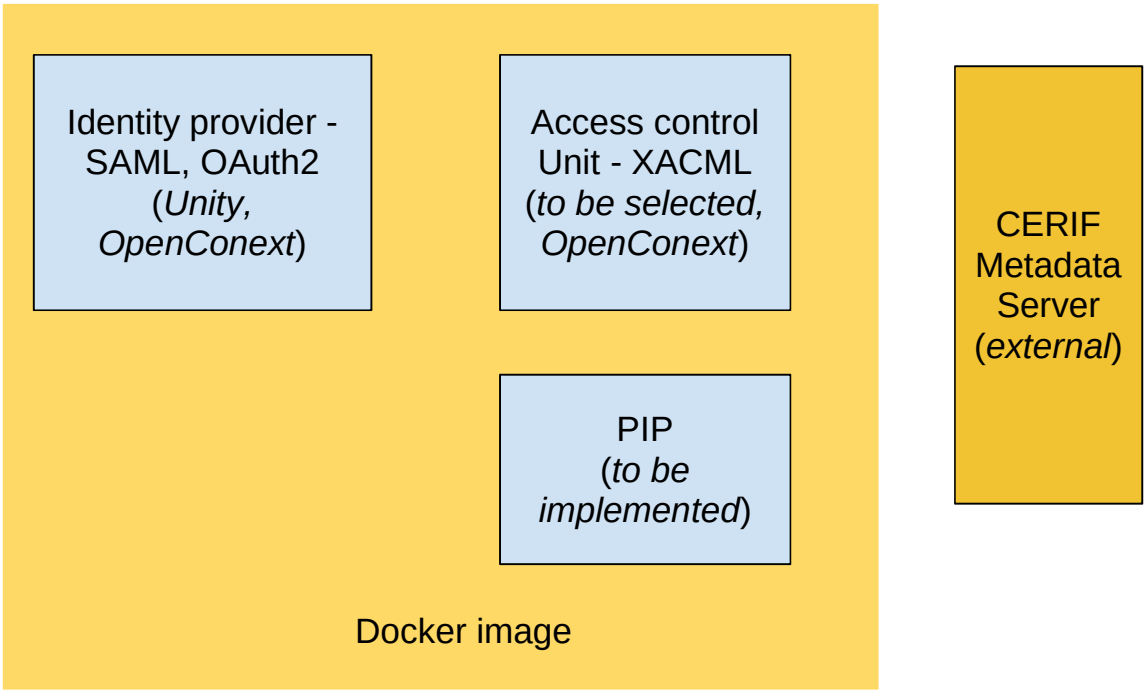
The Data Model Mapper implements all the functionalities, required for the mapping between data schemata and the transformation of data (or conversion of data formats) according to the target schema of the aggregator.

Data Model Mapper utilizes the 3M set of tools that consist of 3M editor[4], the web application which is the GUI for the definition of the schema mappings, 3M manager[5] that exploits an eXist database and x3ml engine[6] 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[7].

5.3 The AAAI

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.



5.3.1 Identity provider

Federated identity management is realised by three major protocols: OpenID, oauth2 and SAML. All these protocols follow the same pattern:

1. If a user wishes to log on the browser is *redirected* (HTTP 30x) to the identity provider (browser contacts node service (NS), NS then redirects to identity provider)
2. The identity provider establishes the identity of the user
3. The identity provider tells the user that a specific service wishes to use his/her identity and (optionally) wishes to have some profile properties (name, email,

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

picture). The identity provider asks for the consent of the user to proceed. Typically you can give that consent for one connection or *forever*.

1. The identity provider redirects back to the initiating service
2. Depending on the protocol, an additional HTTP request from the service to the identity provider may be needed.

After this sequence the service knows the identity of the user is verified by the identity provider. The service has an *identity* (classic OpenID) or an *authorization token* (e.g., oauth2) and optionally some properties about the identified user. In privacy aware settings there may not be any property.
From this moment on, the service must maintain a secure (HTTPS) connection with the user identified by a session cookie. The server must associate the session with the authorization token (or identity, we only refer to authorization token from now on). The authorization code is sent along with messages inside the VRE. **It must be transferred securely inside the VRE**.
If the VRE appears to the user as having multiple HTTP endpoints, the above is used for each endpoint. The user has to approve each endpoint once. When returning the user only has to identity with his/her own identity provider once. Unity can (and must) be configured to **hand the same authorization token to each configured endpoint**.

5.3.2 Federated identity

Our identity provider (in our case an instance of Unity, but a different choice does not affect the design) can maintain 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 VRE identity provider must complete user profile information to satisfy minimal requirements.

5.3.3 User profile

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.

5.3.4 Access control unit

Whenever a service wishes to know that a user has access to a resource, the server posts the authorization token together with its *policies* expressed as an XACML document to the access control unit to ultimately receive a yes/no response. The access control unit executes the following steps:

1. Query the identity provider with the authorization token to obtain the CERIF identity of the user.
2. Query the CERIF metadata service to obtain attributes about the user (e.g., affiliation, relation to research projects) (the PIP module)
3. Evaluate the XACML policy rules on the attributes and decide on access

Candidates for this component are AuthZForce and OpenAZ, assuming we want Open Source and the XACML REST profile.

5.3.5 Access standards

Login to AAAI is based on one of oauth2, SAML or OpenID. Different endpoints and different instantiations of the VRE architecture can make different choices. We recommend using oauth2 for the prototype because it is simple and widely supported.
The only **user profile** management provided by the Node Service component is binding an identity to a CERIF identity. This is achieved using REST calls on the Unity server.
Authorization should probably use the REST profile for XACML. This profile is the most widely supported.
PIP attribute fetching shall call the CERIF metadata service using SPARQL.

5.3.6 User interface

User interaction with the AAAI is:

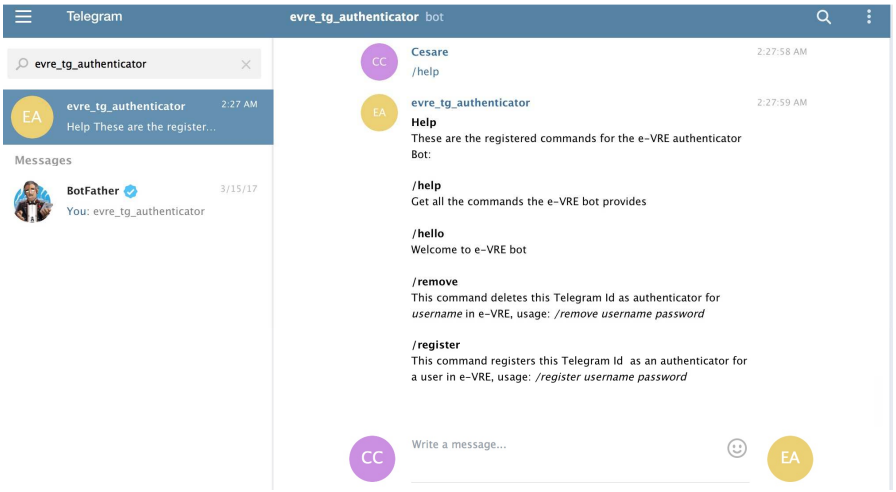
1. Register an account, typically using federation from an existing identity provider.
2. Bind the account to a CERIF person. If our AAAI component gets the identity of the user from an external identity provider (e.g., EDUGain), we have no true identity. One way to relate identities is by using verified email addresses.
3. Remove an account
4. Grant VRE endpoints access to the user's identity. Possibly we do not want this step and automatically assume that if you grant the VRE Unity server with access to your identity you implicitly grant access your identity to all services that are part of the VRE. Note this step is about the user willing to share his/her identity with the VRE, not about the user having access to the VRE or its components.

5.4 The App Service

In order to implement the two factors authentication method (described in the User Manager section) we have developed a software module implementing a *Telegram Bot*: a software that enables the e-VRE to interact with the media platform Telegram.
Telegram Bots are special Telegram accounts that do not require a phone number to set up, the VRE4EIC bot has been registered with the id:

evre-tg_authenticator

A user that wishes to interact with e-VRE using Telegram, can add this account to his/her contact list and can start sending message.



The list of commands accepted by evre-tg_authenticator is:

- **/help**: Get all the commands the e-VRE bot provides
- **/hello**: Welcome to e-VRE bot
- **/register**: This command registers this Telegram Id as an authenticator for a user in e-VRE, usage: */register username password*
- **/remove**: This command deletes this Telegram Id as authenticator for username in e-VRE, usage: */remove username password*

These commands are forwarded by telegram to the e-VRE software module **TgAuthenticator**, developed from scratch using Java technology, that executes them.

When an e-VRE user sends the */register username password* command to evre-tg_authenticator bot, the TgAuthenticator sets up Telegram has the channel that will be used in two factors authentication for that user, this means that the second factor code will be sent to him/her via Telegram (note that Telegram channels are encrypted). The command */remove username password* instead disables the two factor authentication. In the future new commands will be added to this bot.

From the architectural point of view he TgAuthenticator is an independent software module, distributed as jar, that enables the e-VRE to be integrated with an external application (Telegram); in order to manage its life-cycle a basic version of the App manager has been implemented, it is able to start/stop TgAuthenticator and sends messages to other services when the status change.

6 GUI Design

The graphical user interface is provided through the "VRE4EIC Metadata Portal", the design of which is still under development. 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.

6.1 Requirement analysis

The functionality to be provided fulfils the following user requirements:

Security	
R1	Login with two factors authentication;
R2	Role Based Access Control (RBAC);
R3	User profile management;
R4	User Registration;
Data Presentation & Query Construction	
R5	Data classification with respect to VREs and RIs;
R6	Building advanced use case query scenarios;
R7	Providing a simple and user-friendly Graphical User Interface (GUI);
R8	Presentation of results in both tabular and geospatial forms through suitable GUIs respectfully;
R9	Providing map interactivity functionalities for assisting users in constructing geospatial data queries;
R10	Providing SPARQL view of the constructed queries;
R11	Allowing users to store and load constructed queries, in such a way that all actions made through the GUI will be preserved (and shown when loading one);
R12	Stored queries will be associated to the user's profile and the potential of sharing them with other users or making them public (through the portal) will also be possible (future work);
R13	Displaying suitable statistics when available will be provided (future work);
R14	All input forms will be validated before the submission of any entries, preventing the end user from entering erroneous input;
R15	Appropriate recommendations will be available to the end user when necessary;

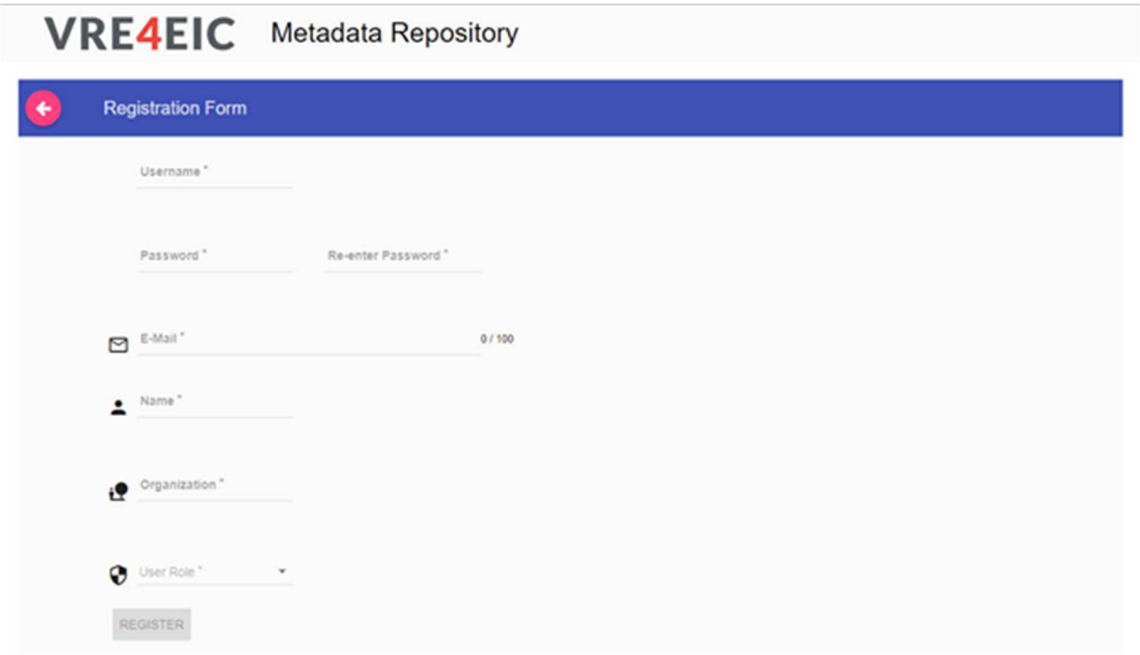
R16	Improper choices leading to dead ends (i.e. no results) will be hidden from the end user;
R17	Appropriate tooltips or guidelines will be available for better user assistance;
Data Import / Export	
R18	Allowing data import from different file formats on specified namegraphs (constructing a new namegraph if necessary) ;
R19	Supporting a variety of RDF based import file formats;
R20	Allowing data export capabilities from any view displaying data results;
Administration & Configuration	
R21	A variety of configuration options for customizing the default behaviour will be available;
Robustness and Fault Tolerance	
R22	All possible errors will be appropriately handled;
R23	All functionalities will be preserved after any error occurrence or recovery;;
R24	The end user will be notified in case of any error occurrence or recovery;

6.2 Generic Use Case Scenarios Through Mock-ups

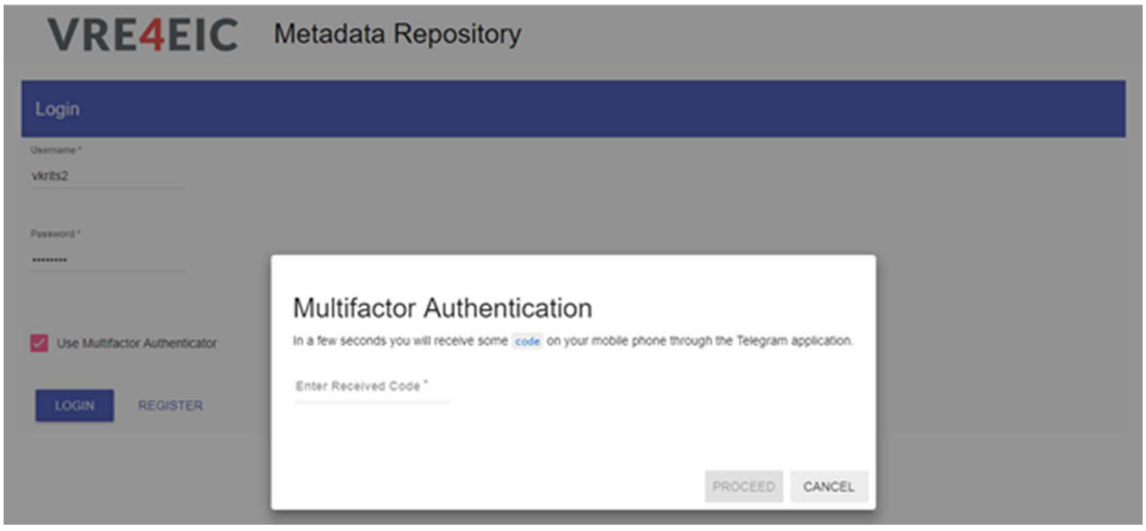
The following chapter describes the basic generic use case scenario while it tries to capture all the possible sub-cases. Graphical user interface mock-ups have been included along with the respective explanatory description in order to present the scenario in more detail. The portal will be accessed through the web using any modern web browser that supports HTML 5.

6.2.1 6.2.1 Login / Registration to the Portal

The user needs an account in order to be able to access the portal. This is achieved by clicking on the “Register” link from the login page and then filling in the respective form.

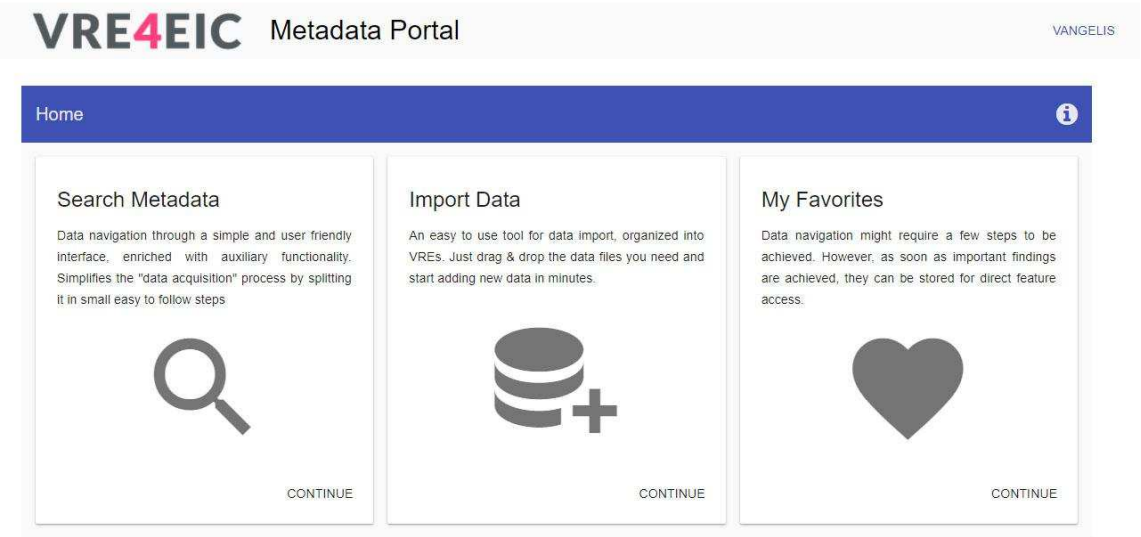


The user can log into the portal by using the two factors authentication method. In order to do so, a Telegram account is required. From the login web page, the user checks the 'Use Multifactor Authentication' box and presses the login button. A temporary code will be sent to the user's Telegram account, which has to be entered manually in the respective input-text of the login page as shown in the following screenshot. An informative message will notify the user when entering the wrong credentials.



6.2.2 Main Menu

After logging in, the perspective view will change and the user will be directed to the home page, where he / she will be asked to choose among 3 different tasks. In any case the system is regulated by user roles and thus these choices might vary according to the roles held.



The three tasks to choose are the following:

a. Construct metadata queries and run them;
b. Import Metadata through RDF files;
c. Accesses user stored queries;

6.2.3 Constructing Search Queries and Running them

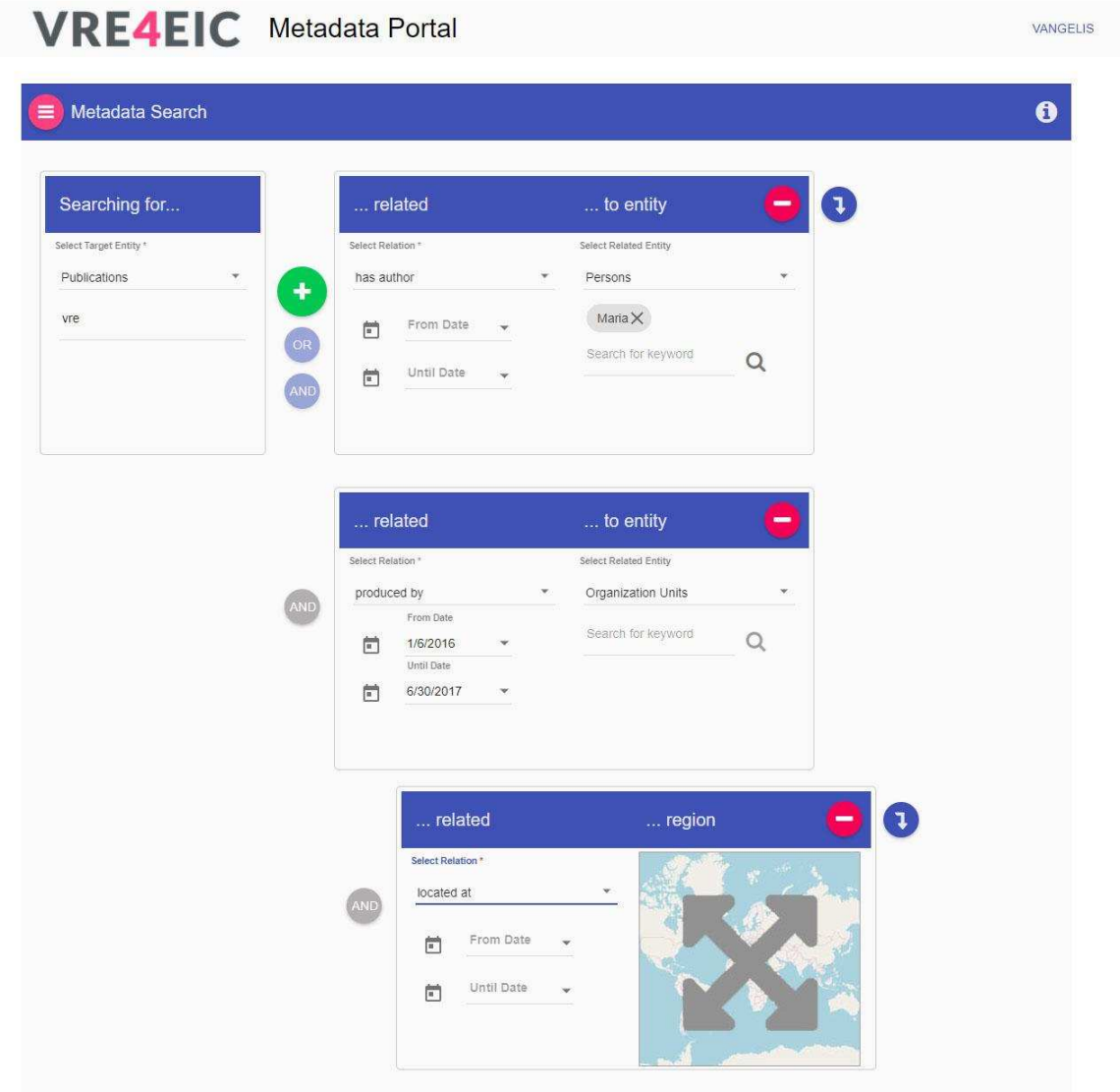
The first task from the list above, provides a simple, easy and generic way of constructing advanced queries for searching metadata. It provides certain UI components (i.e. autocomplete input text, drop down lists, chips, calendar, dialogs, buttons, tables, tooltips, etc), combined together to offer a user friendly experience without requiring any certain technical knowledge by the end user.

As soon as a query has been constructed, it can be saved for future usage or can be immediately be run, returning results. Certain guidelines will be embedded into this UI, assisting users in fulfilling their task (to construct the query).

The main idea is that the end user defines what he/she is looking for when there is no further direct information about it (i.e. looking for some publication about VREs when the title remains), but rather there is information about related entities (.i.e one of the authors is some Maria and Maria is a member of FORTH institute).

Such a query requires to define the target entity, the related entity(ies) and the relation between them. Then further filters can be applied. These can be further relations to different entities. Finally further filters could also be applied to the related entities as well.

In addition to the above, map UI components will also be offered for geographically restricting results in case of an entity that has geospatial nature.



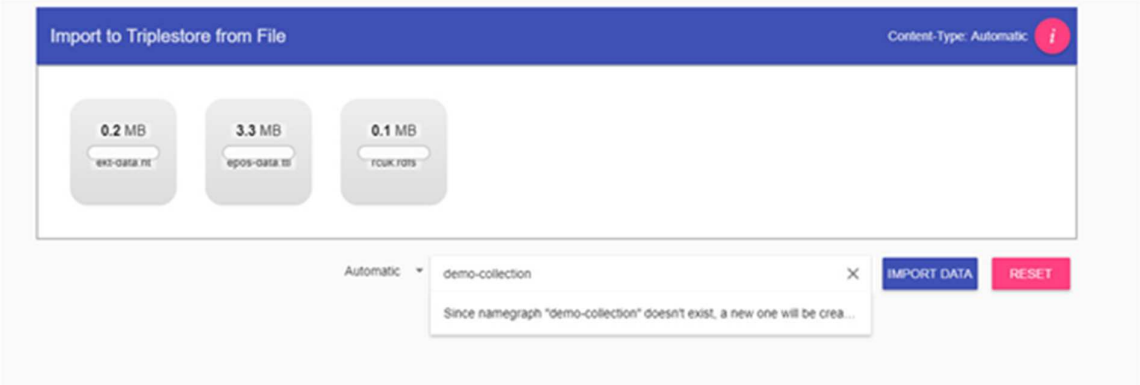
6.2.4 Import Metadata

The portal offers functionality for importing data to the main database. These functionalities are accessible through the home page described earlier.

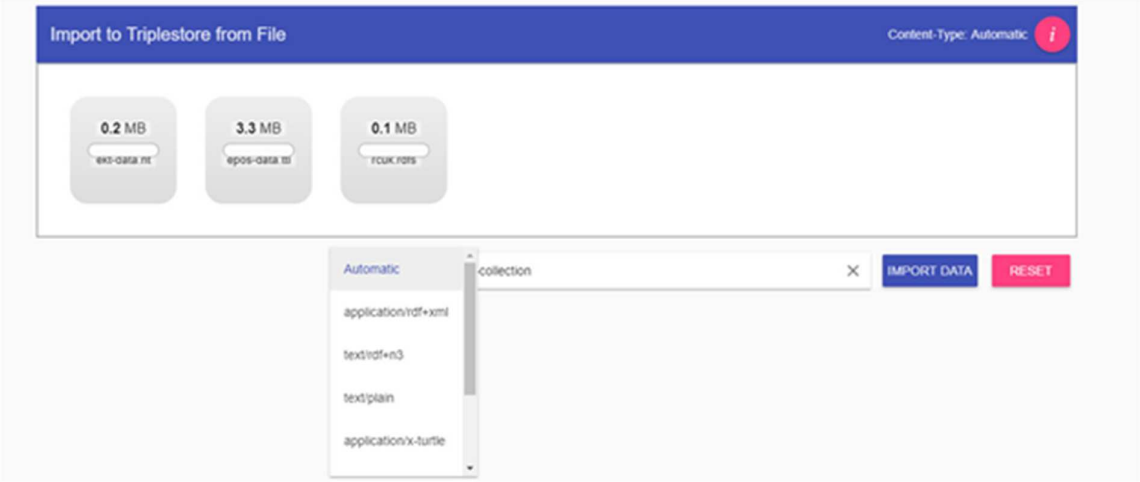
The import functionality allows administrator user to import data in the main database. This is achieved through the respective GUI that provides file drag & drop capabilities. The supported files are:

- rdf-xml,
- rdf-n3,
- text,
- turtle,
- json,
- trig,
- nquads

Initially the user drags and drops the data files in the provided area in the GUI. Then he/she selects the VRE where the data will be imported as shown in the picture below.

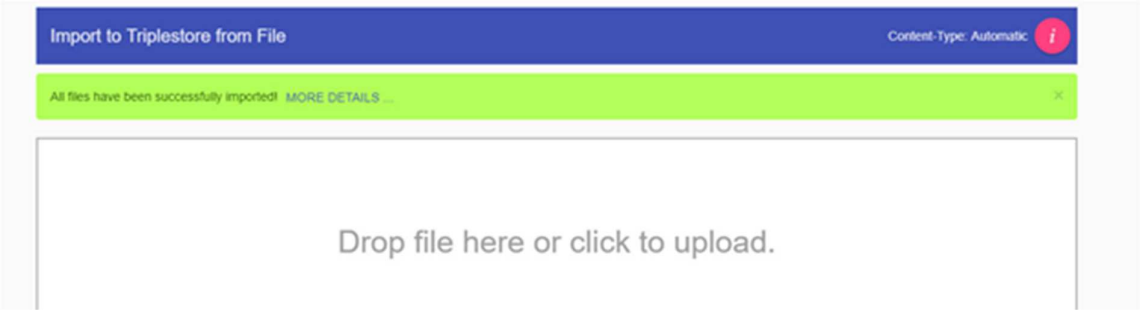


A new VRE one can also be created on the fly, if none is selected. The next thing for the user to decide, is selecting the file format. An automatic option is available which works perfect in case that the selected files vary in content-type.



Then the user clicks on the “Import Data” button to initiate the import functionality. The files will be processed sequentially while their contents are imported into the main database.

Finally, as soon as everything is finished, a success or fail message will notify user about the status. Since multiple files are imported in one step, the possibility that some of the file imports might fail while other imports might succeed, is possible. The user will be able to observe the status for all file imports, after the whole task is completed, by clicking on the provided link “More Details…”.



6.2.5 Accessing stored constructed queries

Each user has access to a list of stored queries constructed by him / her self. This list is reached from the home page as a 3d option in the available tasks. As soon as a query is loaded, the view will change to that one of the Query constructor with every steps made by the user preloaded as well.

6.3 Technologies used

Web GUI

Main technology used: AngularJS & Material Design.

GUI's Back End

Main technologies used: Spring Boot.

6.4 Technical Requirements

The following VRE4EIC Services are required for the portal to properly function:

- **Node Service** - Used for the User Registration, Authentication, Profile and RBAC
- **Metadata Service** - Used for constructing & running the queries, importing & exporting data
- **URI Resolver** - Used for resolving URIs from the search results

Finally, the only technology related requirement is that java 8.0 or higher has to be installed on the server running the VRE4EIC Portal.

6.5 Deployment Process

The portal is package in a jar file which is self deployed on a JETTY Web server and javax.servlet container by executing the respective command on terminal. Several properties have also to be configured on the respective property file. These settings are:

- File size limit for uploading RDF files when importing data
- The Metadata Service URL
- The Node Service URL

7 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.

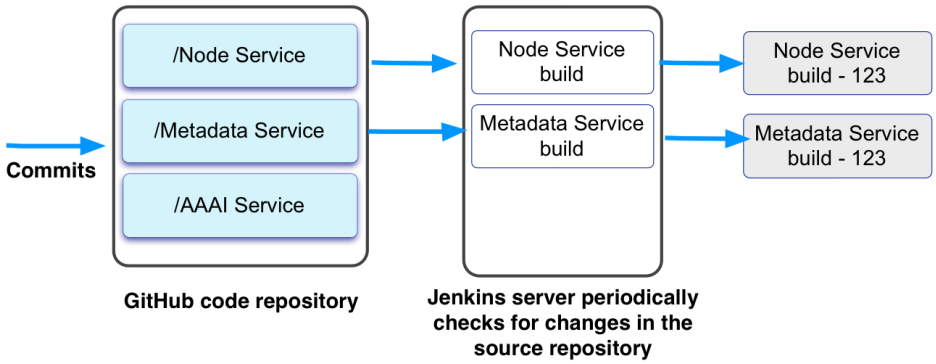
We decided to use Git as Version Control System for the documents produced during the architecture design, and for the source code produced during the implementation of e-VRE.

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.



To distribute the e-VRE system we decided to adopt a container-based virtualization approach. We are currently using Docker [Docker] as framework for this task. The current configuration is based on a multi-container Docker application that uses 3 containers:

- v4e-tomcat
- v4e-mongodb
- v4e-activemq

The first container hosts the VRE4EIC Java code under a Tomcat server, the other two containers deploy the mongodb and activemq services. However we have just started this task and we are now starting the test phase.

8 Conclusions

The present report accompanies the three Building Blocks delivered by VRE4EIC, namely the Node, the Metadata and the AAAI service.

The report describes the architectural work that has been done in order to transform the Reference Architecture (presented in D3.1 and briefly recapitulated in Section 3 above) into a Technical Architecture (Section 4) that takes into account the non-functional requirements while respecting the implementation strategy of the project, targeted at enhancing existing VREs. This work has resulted in a microservice-based architecture, which is a relatively recent technological development [Newman] perfectly fitting the project scenario.

Some technical details concerning the three microservices are provided in Section 5.

The report includes also the design of the GUI, which is a sort of cross-service Building Block developed to address usability aspects and to support the usage of the project results for teaching and demonstration purposes.

The activity of WP3 proceeds now on two parallel streams:

- integration of the implemented building blocks into existing VREs (Task T3.4)
- development of the GUI (Task 3.5)

The present deliverable has laid the necessary bases for both these streams, which will deliver their results together in D3.4, at Month 30.

9 References

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

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

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

[ALv2] http://www.apache.org/licenses/LICENSE-2.0

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

[Curator] Curator zookeeper recipes, http://curator.apache.org/ (Apache Software Foundation)

[JMS] N. Laranjeiro, et al., "Experimental Robustness Evaluation of JMS Middleware," *2008 IEEE International Conference on Services Computing*, Honolulu, HI, 2008, pp. 119-126.

[JMS Brokers] A. F. Klein, et al. , "An experimental comparison of ActiveMQ and OpenMQ brokers in asynchronous cloud environment," *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*, Sierre, 2015, pp. 24-30.

[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.

[AMPQ] J. L. Fernandes, et al., "Performance evaluation of RESTful web services and AMQP protocol," *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, Da Nang, 2013, pp. 810-815.

[MQTT] B. Aziz, "A Formal Model and Analysis of the MQ Telemetry Transport Protocol," *2014 Ninth International Conference on Availability, Reliability and Security*, Fribourg, 2014, pp. 59-68.

[Docker] https://www.docker.com