Architecture requirements analysis and design document

Daniele Bailo and the EPOS IT Team
June 2021

Table of Contents

1. Glossary

AAAI	Authentication, Authorisation, and Accounting Infrastructure
Cohesion	Cohesion refers to how well the individual elements within an application work together. As a general rule, developers should aim to build loosely-coupled, highly-cohesive software systems. The reason being, highly-cohesive systems tend to be more robust, reliable, and reusable than those with low cohesion. (https://www.tiempodev.com/blog/microservices/#:~:text=Microservice%20architecture%20is%20often%20achieved,language%2Dagnostic%20APIs%20like%20REST)
csv	Comma Separated Value
DDSS	Data, Data products, Software and Services
Decoupling	Microservice architecture is often achieved by decoupling a monolithic application into independent modules that each contain the components necessary to execute a single business function. (https://www.tiempodev.com/blog/microservices/#:~:text=Microservice%20architecture%20is%20often%20achieved,language%2Dagnostic%20APIs%20like%20REST)
DOI	Digital Object Identifier
EAB	External Advisory Board
EIDA	European Integrated Data Archive
EPOS-IP	European Plate Observing System – Implementation Phase
EPOS-N	European Plate Observing System – Norway
EUREF	Reference Frame Sub Commission for Europe
FAIR	Data principles: Findable, Accessible, Interoperable, Reusable
GLASS	Software/portal for WP10
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
ICS	Integrated Core Services
ICS-C	Integrated Core Services - Central Hub
ICS-D	Integrated Core Services - Distributed
MoU	Memorandum of Understanding
MSEED	MiniSEED (data format)
ORFEUS	Observatories and Research Facilities for European Seismology
PMB	Project Management Board
RCN	Research Council of Norway (Forskningsrådet)
RINEX	Receiver Independent Exchange Format
SAC	Seismic Analysis Code (storage/file format)
SeedLink	Protocol for seismic data exchange

SEISAN	Earthquake analysis software
TCS	Thematic Core Services
WMS	Web Map Service
WP	Work Package

2. Requirements analysis & processes definition

Requirements and processes are described form a User (agent or human) perspective. We identified and analysed 3 main Users Stories:

a. *Access and use of scientific (meta)data*
b. *Metadata Management*
c. *Agent interoperating with specific metadata formats endpoints*

For each User Story we defined:

a. *User Story Description* - User stories generically describe the expectations of the users and how a generic user wants to interact with the system. Users can be human or agents. User stores are written together with the users (customers).
b. *Macro processes* describe the system usage from a user or agent perspective. Compared to the user stories, this is a more formal descriptions of user/system interactions. Each step in the macro processes is a process itself, and is described in the “Process Description” section.
c. *Use Cases Definition* - For each step (user processes) in the macro processes, we describe the required process in terms of UML diagrams, preconditions etc.
d. *Flat Marker Diagram* – Represent a sketch mock up view of the system
e. *Process Definition* – A detail description of each use case using BPNM notation

2.1.. Access and use of scientific (meta)data

User Story description

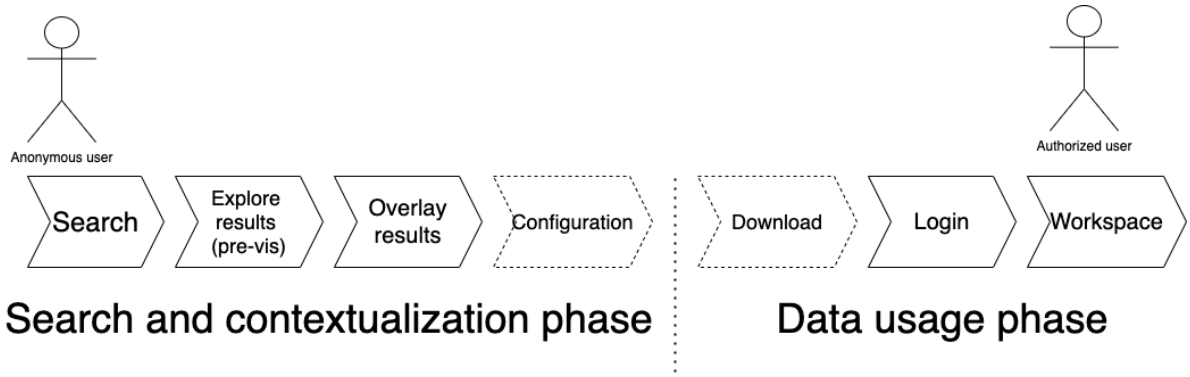
As a domain expert, I want to investigate a major fault zone that is monitored by a range of scientific instruments, including seismic, geodetic, and geochemical stations, to better understand the tectonic setup and geodynamics of the area. In this instance, I am going to look at Alto Tiberina and datasets associated to that region.

Table I. User story details

User Type	Functionality	User story process
	Search	● Perform basic search (spatio-temporal parameters) ● Perform Advanced search (keyword, organizations, free text on facets tree)

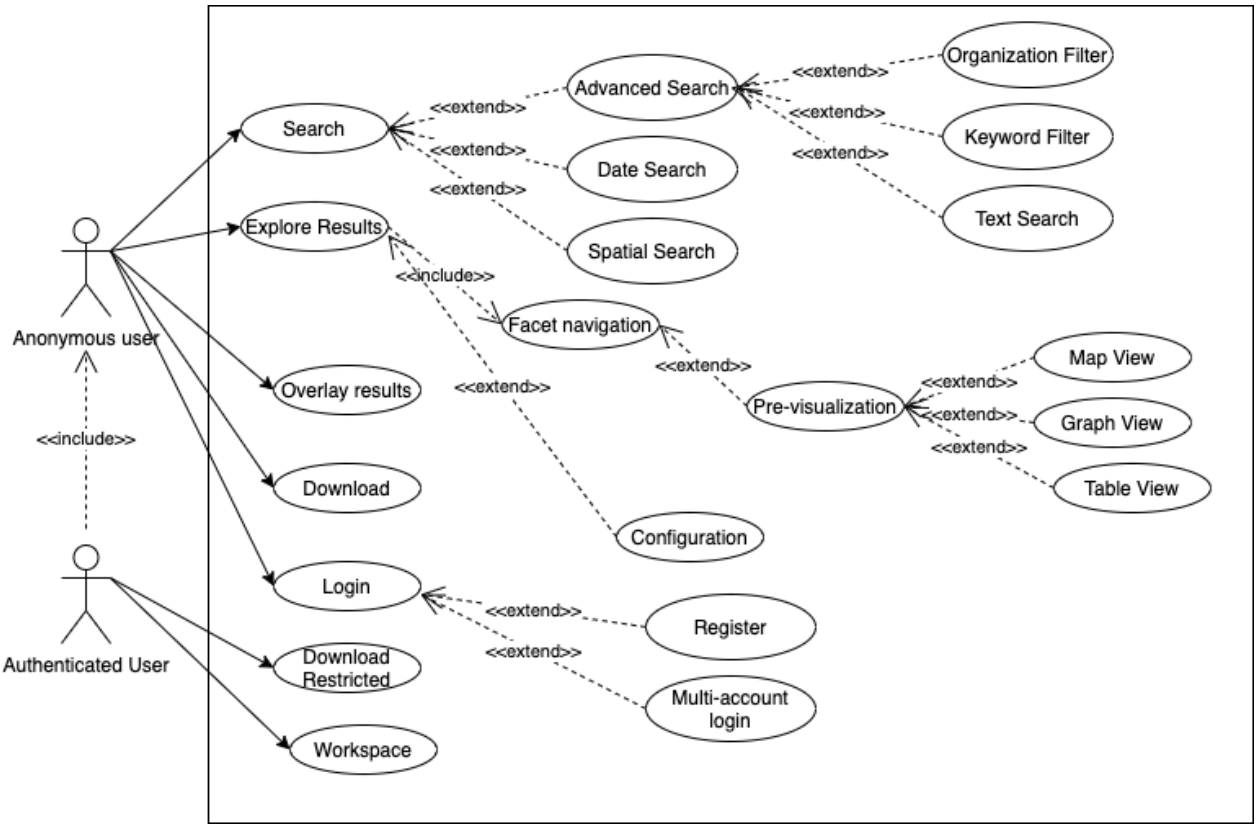
Scientific User	Explore result	• Browse the facets tree • Select the DDSS of interest • Previsualize the DDSS results on: Map pane, Graph pane, Table pane • Switch to detail pane • Switch to configuration pane • Configure available parameters to improve the initial search • Apply the new parameters to perform a new query and show the filtered results on Map pane, Graph pane, Table pane
	Overlay results	• On the facets tree pin a DDSS item in order to overlay the results on the map together with other DDSS already selected
	Download	• Identify the DDSS of interest • Click on the download icon • Choose the output file format if available
	Login	• Specify username and password or use a different authentication provider (Google account, EGI account, TCS specific integrations) • Register a new account (may need to have permission from EPOS-ERIC users admins, or need to have acceptance of terms and conditions, dislaimers, consents etc)
Authenticated Scientific user	Download restricted	• Identify the DDSS of interest • Click on the download icon • Choose the output file format if available • Login before perform download
	Workspace	• Identify the DDSS of interest • Click on the add to workspace icon • Login • Select an existing workspace or create a new one • Open workspace pane to check

Macroprocess



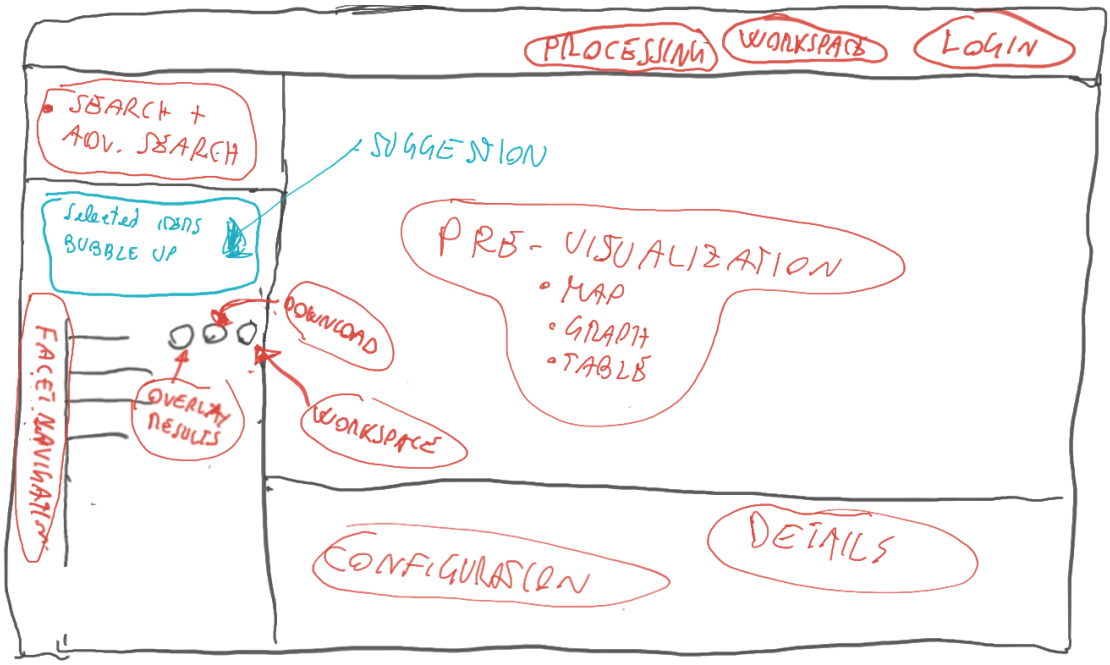
https://app.diagrams.net/#G1RhsKbAG7fGIJadxjWpVOFyQ3ZQShUU_G

Use Cases



Flat Marker Diagram

The following picture captures use cases into a GUI, Use cases are red-circled.
A potential future GUI revision and redesign will address further details.



https://jamboard.google.com/d/1PzsrdKth2wOMiwdi3BXHN4zm7B46LGucbOmGde7d-fE/viewer?f=0

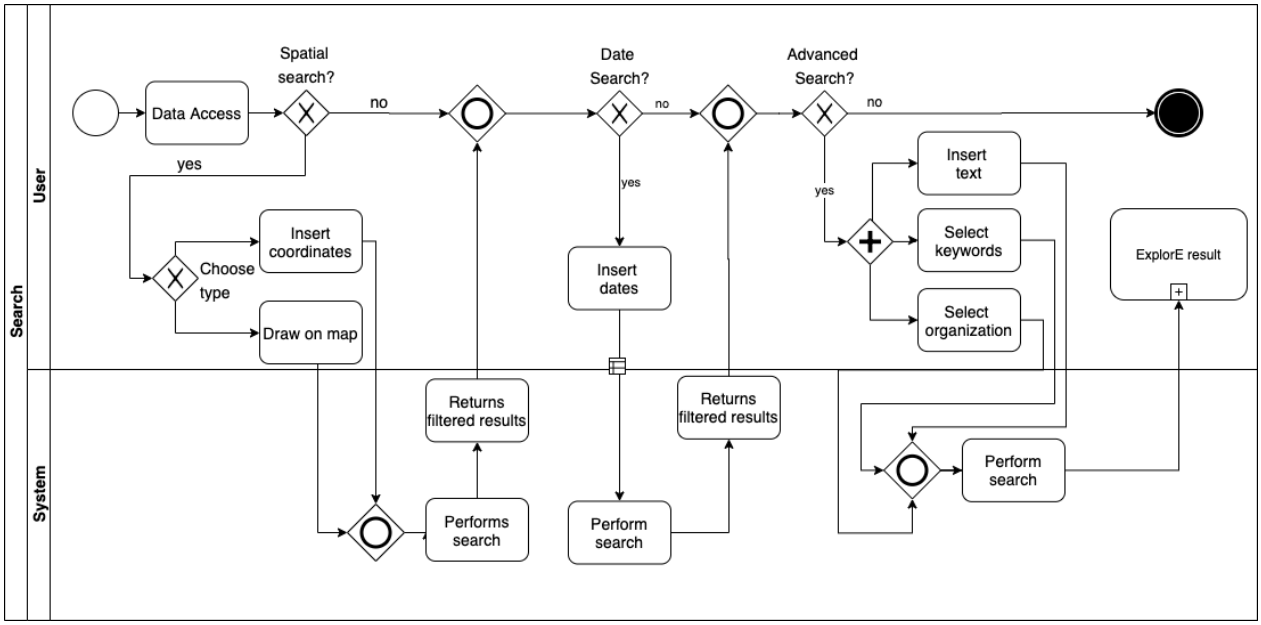
Process Definition

a. Search

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

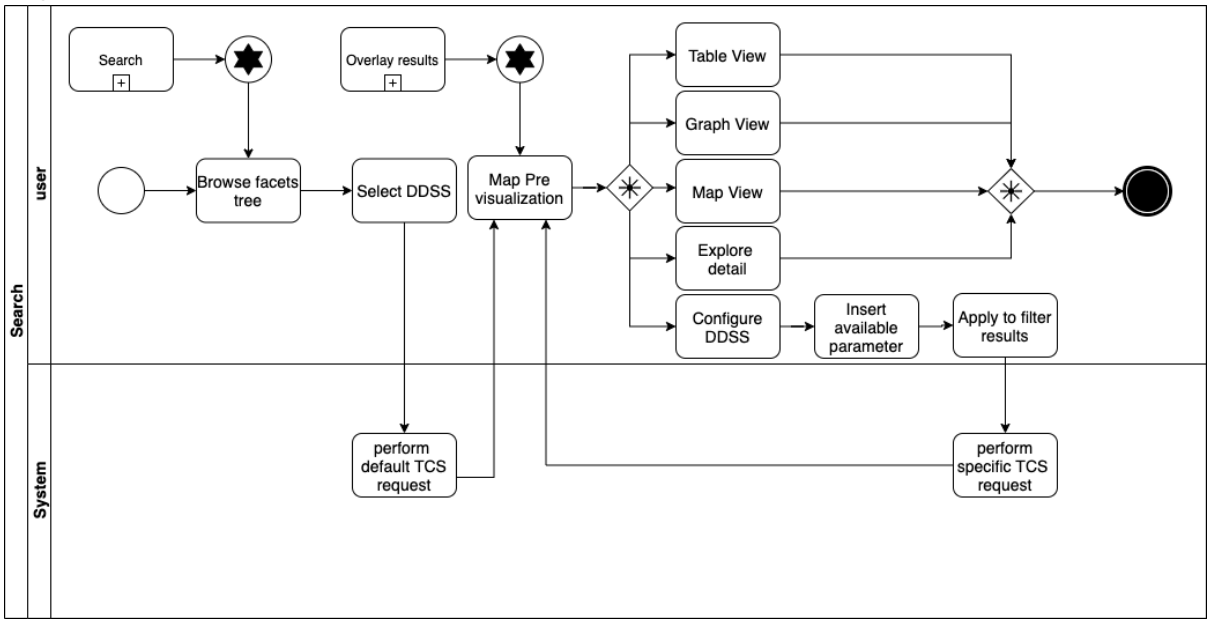
b. Explore results

Prerequisite

Actor	Precondition
System	Should be running

System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



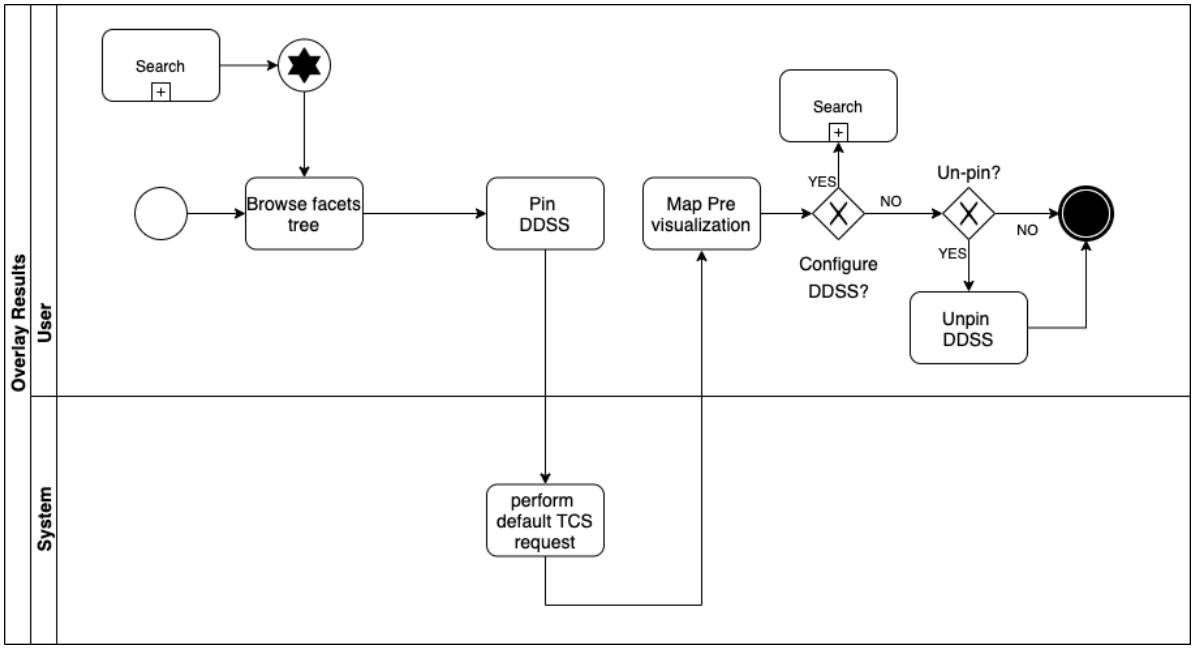
https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

c. Overlay results

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



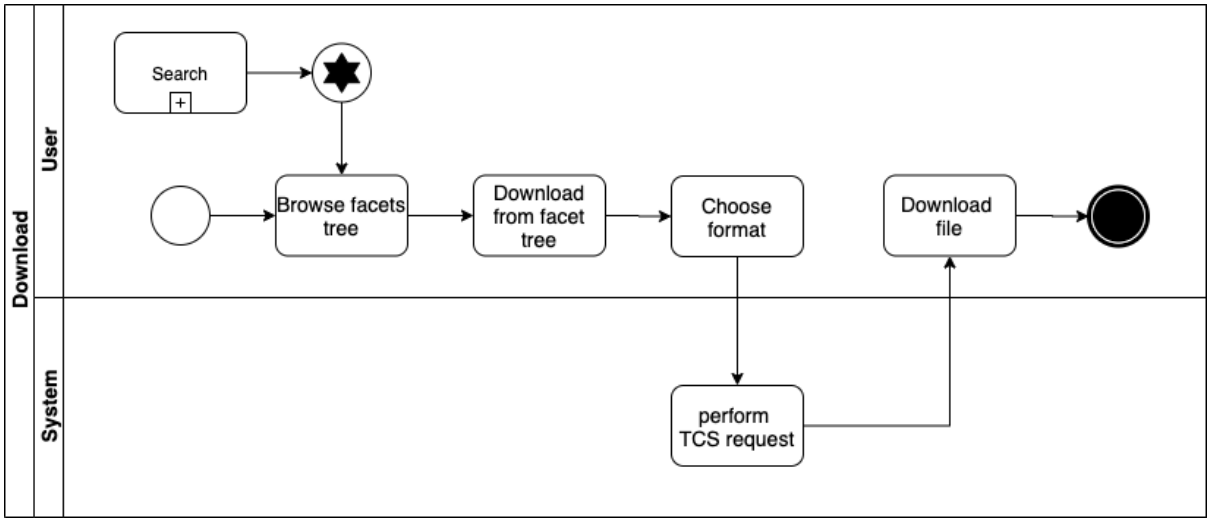
https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

d. Download

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



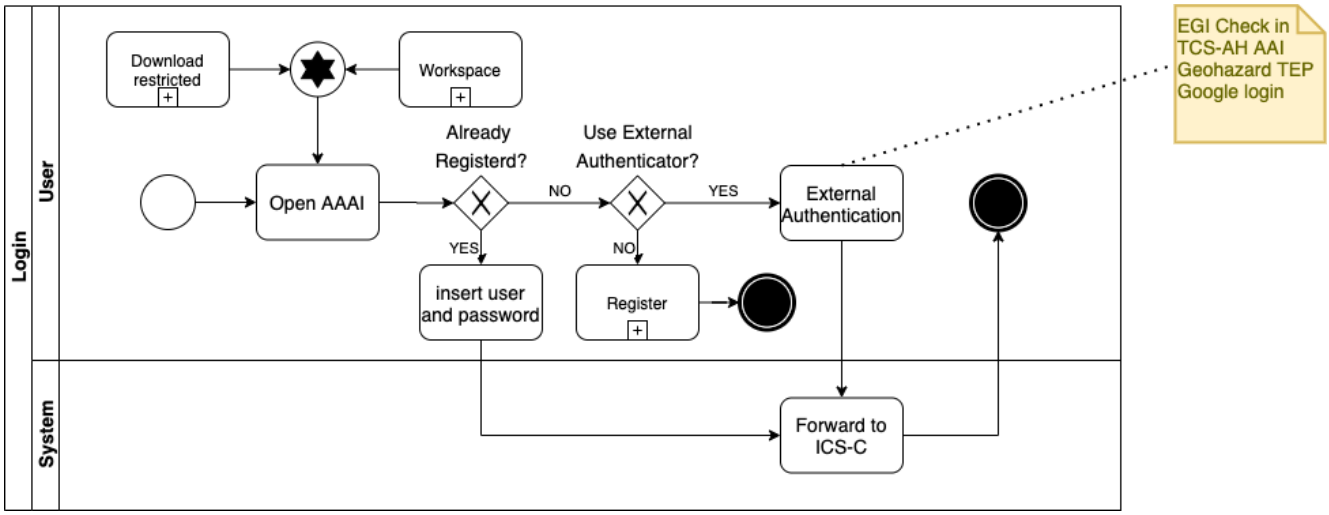
https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

e. Login

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



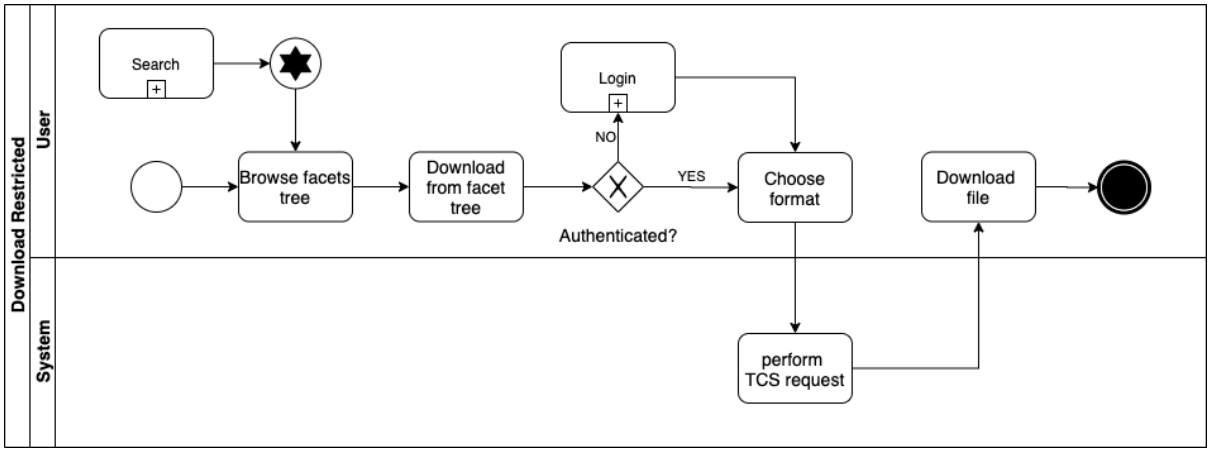
https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

f. Download restricted

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



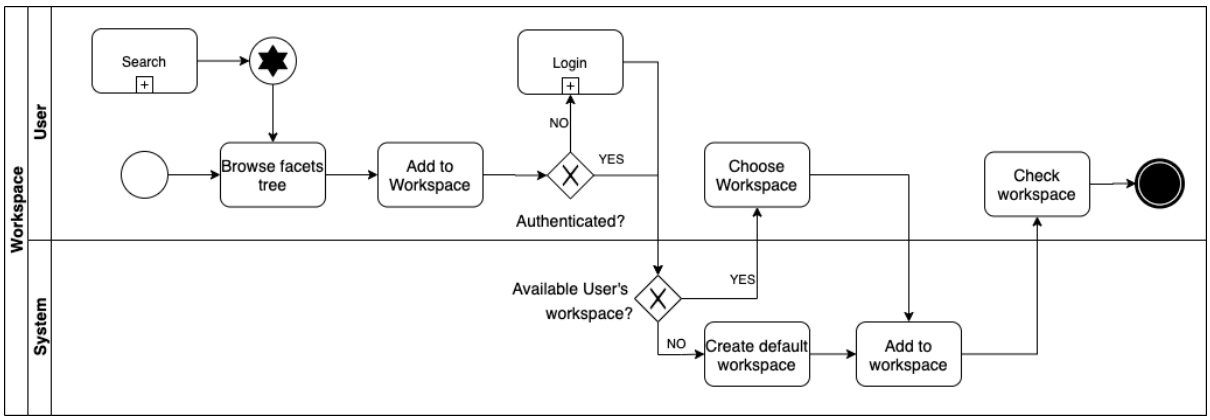
https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

g. Workspace

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
User	Should open the browser and insert URL https://www.ics-c.epos-eu.org/
User	Should accept terms and conditions and cookie policy

Logical flow



https://app.diagrams.net/#G1UGkyn7mjgaCOHqcTCESa1x7LS-LstFO2

2.2.. Metadata Management

User Story description

Dealing with scientific and administrative metadata feeding the interoperability layer.
Here users should be able to input/update new metadata (TCS data manager) and accept/reject proposed metadata input/updates (metadata curators).
As an authorized metadata editor person, I would like to have a GUI to manage metadata, that we may call “backoffice”.
The backoffice should have the following features:

1. Allow me to upload a DCAT file
2. Allow me to create/update/delete metadata manually through a GUI with forms
3. Know what is the status of acceptance of the metadata revision.

Metadata Management involves 2 main actors:

- TCS DATA MANAGER: is the person in charge to manage the data for the Thematic Community. On the system will be configured many TCS DATA MANAGER as many Tematic Community are involved on EPOS
- METADATA CURATOR: is the person in charge to check the Tematic Community data and deploy the new insert and updates in production

Below is listed all users' stories processes for every actor involved in Metadata Management.

User Type	Functionality	User Story Process
TCS DATA MANAGER	Metadata Import	Backoffice Login Open Metada import function Upload a new TTL file Ask for Metadata Revision
	Online Metadata Management	Backoffice Login Browse published metadata Manage metadata (Update, Delete) Insert new metadata (Insert)
	Online Integrity Check	Backoffice Login Browse system version link Open system version link Check for data on the system
METADATA CURATOR	Metadata Revision	Backoffice Login Browse system version link Open system version link Check for data on the system Approve changes and start Deploy procedure

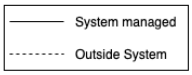
Macroprocess

For a complete detailed description of these processes, please refer to the original pitch’s results following this link

We described 2 different macro processes involved in Metadata management procedure:

- EPOS DCAT-AP METADATA IMPORT:
    - Metadata identification: it is an activity outside the system performed by the TCS DATA MANAGER who select the metadata to be inserted on the system
    - DCAT-AP mapping: using EPOS DCAT-AP vocabulary and definitions the TCS DATA MANAGER describes the previously identified metadata
    - Using a software that helps to validate the files, the TCS DATA MANAGER create the Turtle (TTL) file
    - The TCS DATA MANAGER access to the system and import the TTL file
    - After importing the data into the system, the TCS DATA MANAGER access to the new system version to check the Metadata
    - After checking the metadata on the new system version, the TCS DATA MANAGER ask for the Metadata Revision to the METADATA CURATOR
    - The METADATA CURATOR performs the revision on the system and start the Deploy to production environment. If the service is new this may require validation by those managing the Service Activation Roadmap (ICS-TCS interaction leaders and/or Executive Committee)
- METADATA CREATION AND EDITING
    - After accessing to the backoffice the TCS DATA MANAGER access to the metadata to insert, update or delete metadata
    - Afte changing the data the TCS DATA MANAGER access to the new system version to check the data
    - After checking the metadata on the new system version the TCS DATA MANAGER ask for the Metadata Revision to the METADATA CURATOR
    - The METADATA CURATOR perform the revision on the system and start the Deploy to production environment

EPOS DCAT-AP METADATA IMPORT

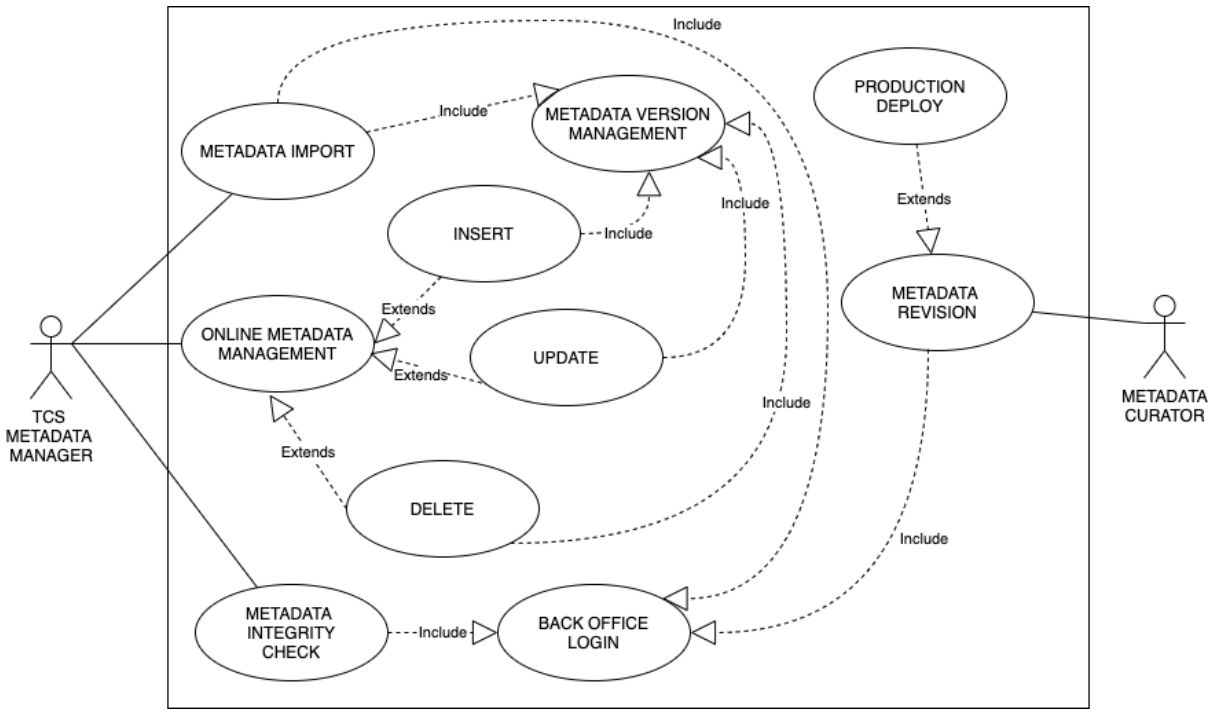


ONLINE METADATA CREATION AND EDITING

Use Cases

For a complete detailed description of these processes, please refer to the original pitch's results following this link

Below are described the use case



Flat Marker Diagrams

Not yet implemented

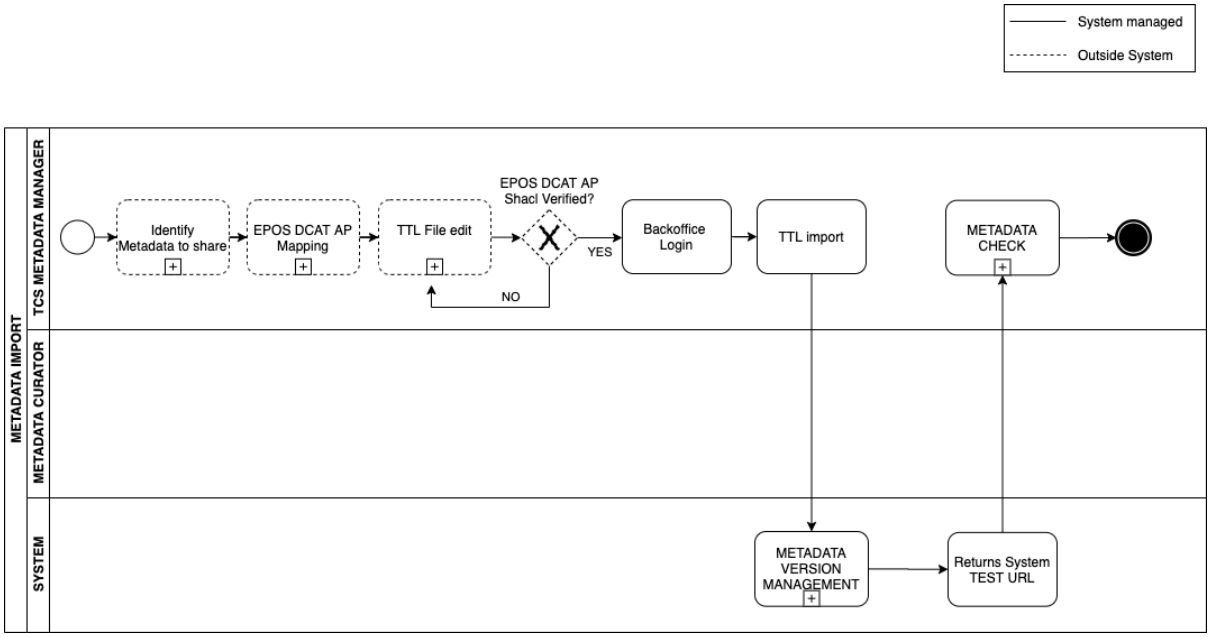
Process Definition

a. Metadata import

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
TCS Metadata Manager	Should open the browser and insert the backoffice URL
TCS Metadata Manager	Should be a registered user with metadata management authorizations

Logical flow

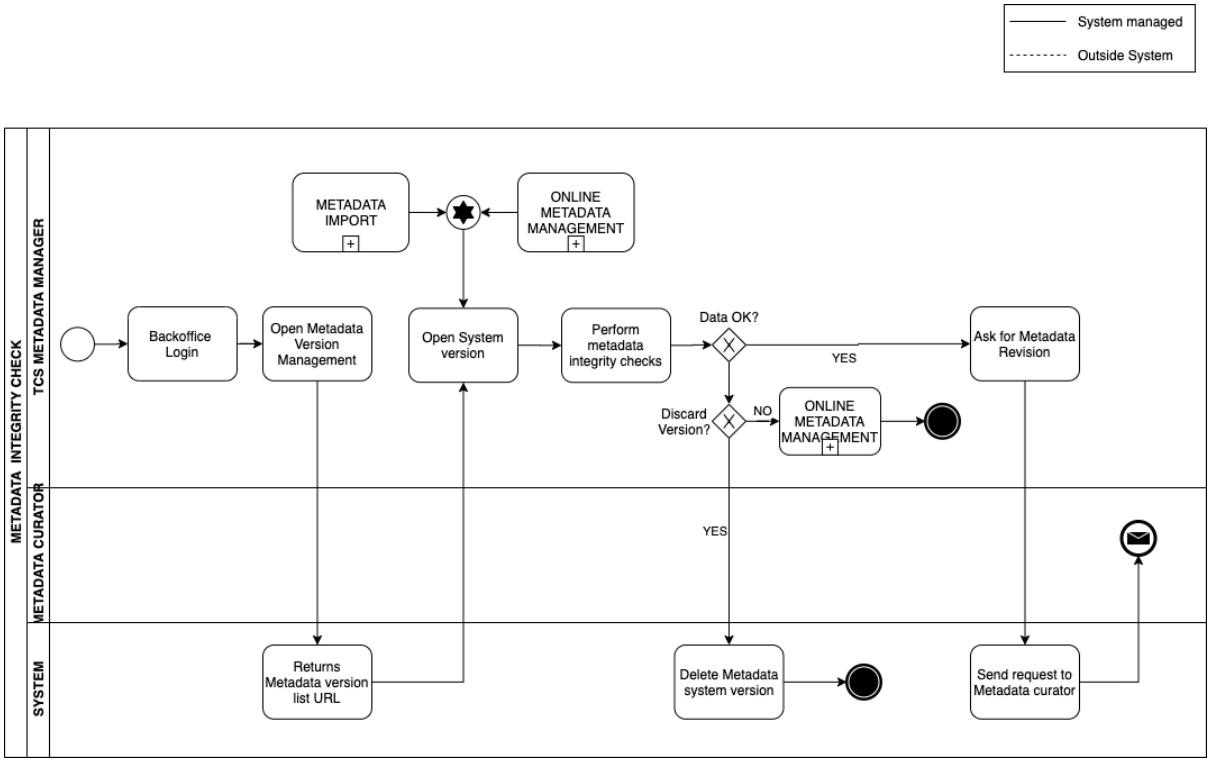


b. Metadata integrity check

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
TCS Metadata Manager	Should open the browser and insert the backoffice URL
TCS Metadata Manager	Should be a registered user with metadata management authorizations

Logical flow

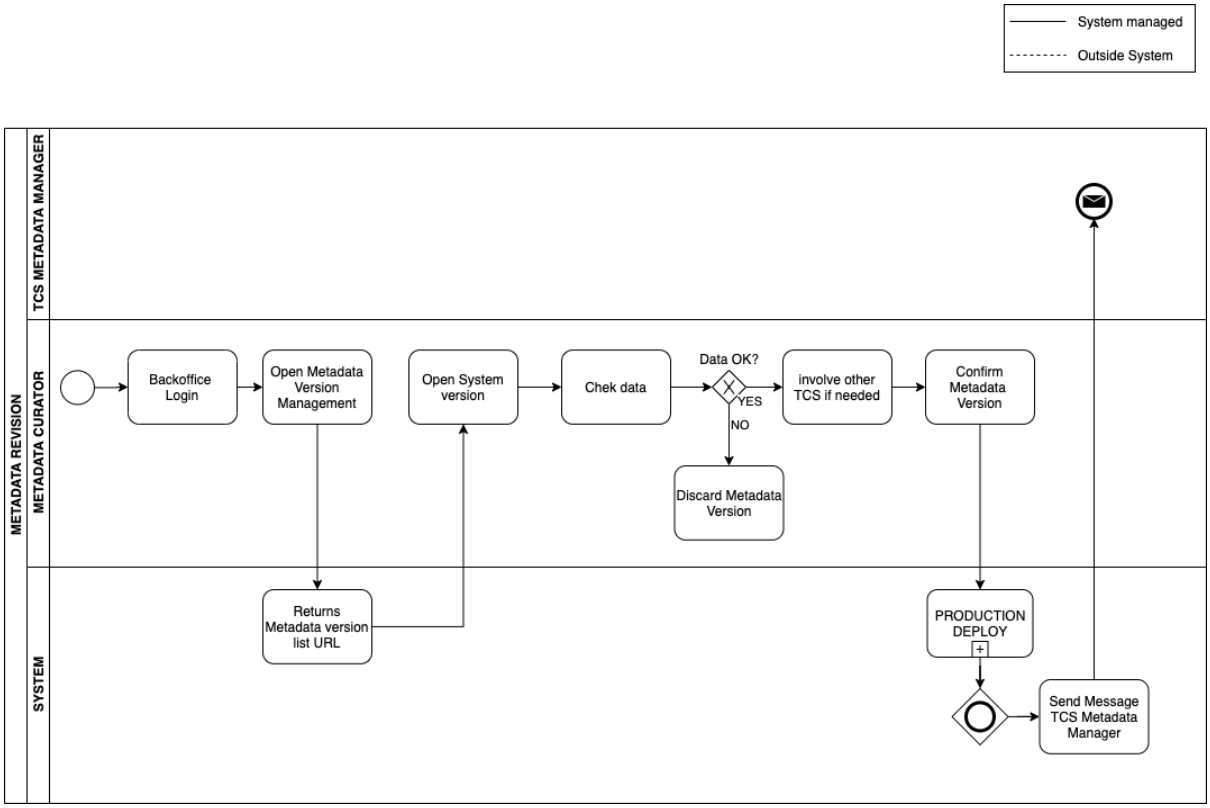


c. Metadata revision

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
TCS Metadata Manager	Should open the browser and insert the backoffice URL
TCS Metadata Manager	Should be a registered user with metadata management authorizations
Metadata Curator	Should open the browser and insert the backoffice URL
Metadata Curator	Should be a registered user with metadata curations authorizations

Logical flow

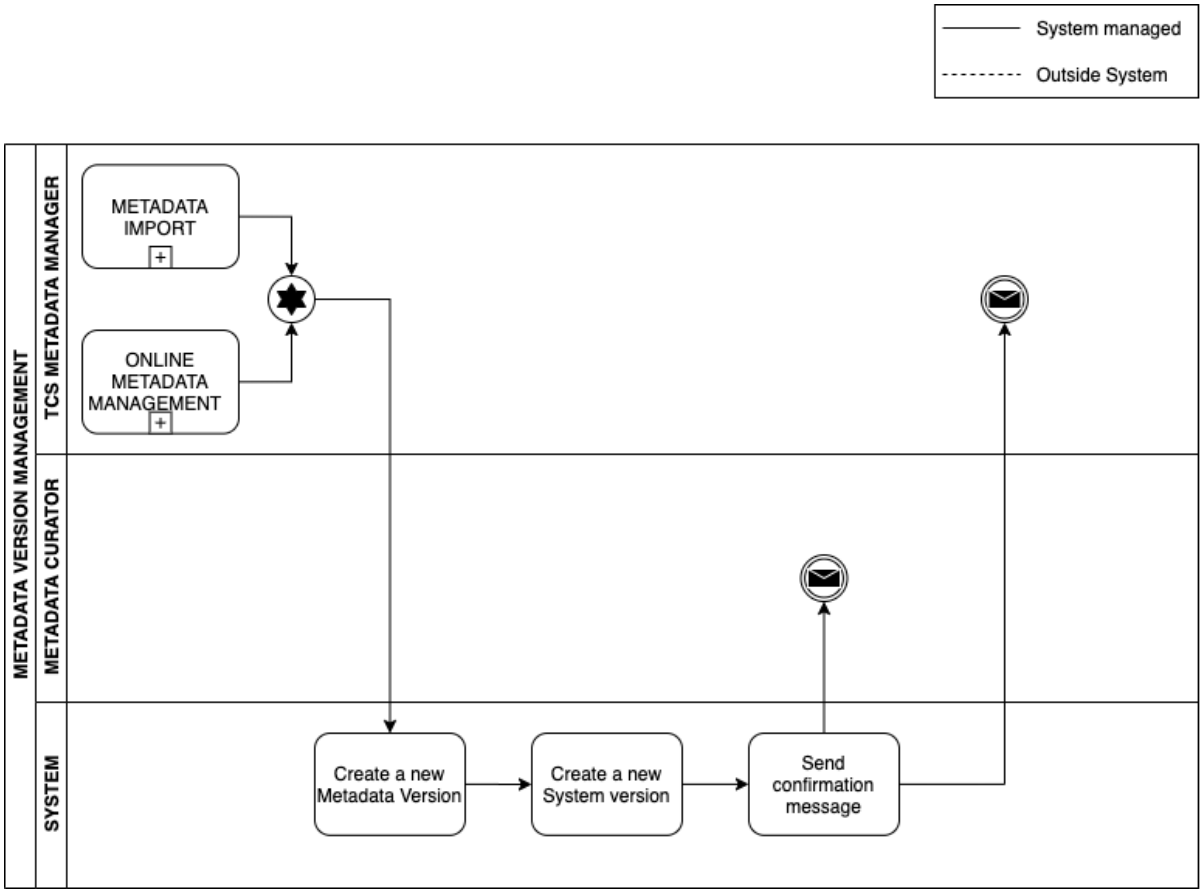


d. Metadata version management

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running

Logical flow

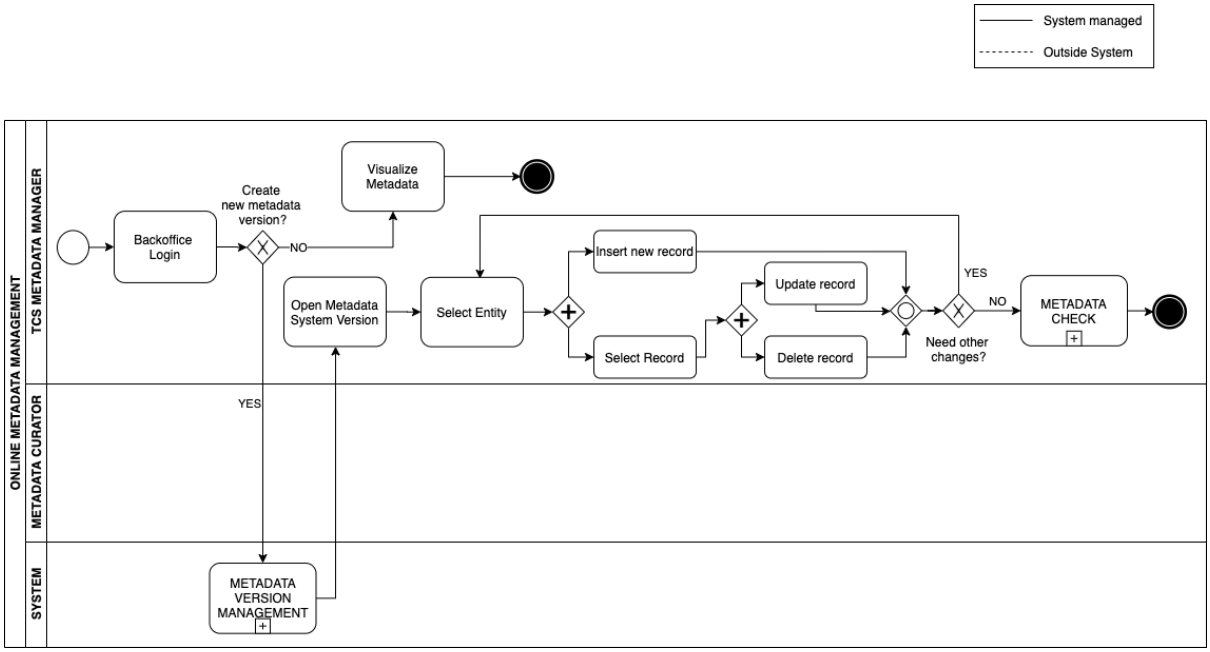


e. Online metadata management

Prerequisite

Actor	Precondition
System	Should be running
System	All components should be up and running
TCS Metadata Manager	Should open the browser and insert the backoffice URL
TCS Metadata Manager	Should be a registered user with metadata management authorizations

Logical flow



2.3.. Agent interoperating with specific metadata formats endpoints

User Story description

As a software agent, I want to be able to query an endpoint and receive the metadata in different metadata formats, e.g.:

- DCAT-AP
- CERIF-XML

--> TECHNICAL WORK TO BE DONE:

- WEB-API DESIGN
- IMPLEMENTATION OF CERIF--> DCAT CONVERTORS

Macroprocess

This process will be investigated in the future.

Use Cases

This use cases will be investigated in the future

Flat Marker Diagrams

To be defined

Process Definition

The related processes will be investigated in the future

3. Architectural Considerations

The previous chapter clearly highlights the requirements of the system, whether in terms of User expectations and in terms of user processes and interaction with the system.
Such a requirement collection provides information about the architecture of the system used to implement the listed descried requirements.
The goal of this section is to establish the basis for a clear and detailed system architecture design (discussed in section 4) by defining the core concepts and costraints that descend from the requirements in chapter 2.

3.1. Architectural requirements definition

Chapter 2 ”Requirements analysis & processes definition defines requirements from a User perspective.
What is immediately apparent is that the system needs to satisfy different users stories, that include a number of different functionalities. These functionalities are likely to evolve in time, and new ones are going to be added. This leads to the definition of two main architectural requirements (AR):

AR #1: The system needs to support many different functionalities
AR #2: the system needs to be able to easily support slight refactoring of existing functionalities and easy inclusion of new functionalities

AR #1 and AR #2 are common to many current IT systems, and one of the best approaches to satisfy them is adopt a modular architecture design

As a result, we can define an additional Architectural Requirement:

AR #3: The system should be implemented with a modular architecture

In the EPOS case, the team is composed by an international team where: different roles are assigned, and in particular a Dev Team is responsible for developments while a Dev Team, represented by Hosting Organizations, provides the e-infrastructure. This leads to another key requirement related to the collaborative approach:

AR #4: the system should be implemented by using modern DevOps techniques that ensure performances and other key attributes (described later).

Being the team distributed over different countries, and given the extreme flexibility of some teams where an Agile approach is adopted, it is vital that the modules of the system can be easy maintained, which leads to AR#5:

AR #5: the system should guarantee maintainability of modules ensuring loose coupling as much as possible.

What also emerges from the use cases in chapter 2, is that the system needs to ensure a strong management of the data and information in order to serve the users who should be able to browse information (metadata) about TCS resources and access the datasets.
In order to manage this complexity, two additional Key architectural requirements raise:

AR #6: The sytem should be based on the EPOS data model (described in section " The EPOS Data Model**")**
AR #7: The system needs to implement a rich metadata catalogue in order to store all the information required by the EPOS data model.

The key system requirements are here reported in tabular form of readers convenience:

Requirement #	Requirement description
#1	The system needs to support many different functionalities
#2	the system needs to be able to easily support slight refactoring of existing functionalities and easy inclusion of new functionalities
#3	The system should be implemented with a modular architecture
#4	The system should be implemented by using modern DevOps techniques that ensure performances by and other key attributes (described later).
#5	The system should guarantee maintainability of modules ensuring loose coupling as much as possible.
#6	The system needs to implement a rich metadata catalogue in order to store all the required information to manage the integrated assets

The above requirements are here summarized in a tabular way but they are the results of years of study, discussion and considerations condensed in key deliverables like "Second (Final) Report on EPOS-ICS Architecture" (10.5281/zenodo.1470257)[1]

As a consequence of these Architectural Requirements, we can define Goals and Constraints to be considered in the architectural design.

3.2. The EPOS Data Model

In order to guarantee that the concepts of interest in EPOS are well dealt with within the architecture, and in order to ensure that appropriate communication occurs among different modules, it was necessary to define an EPOS data model.

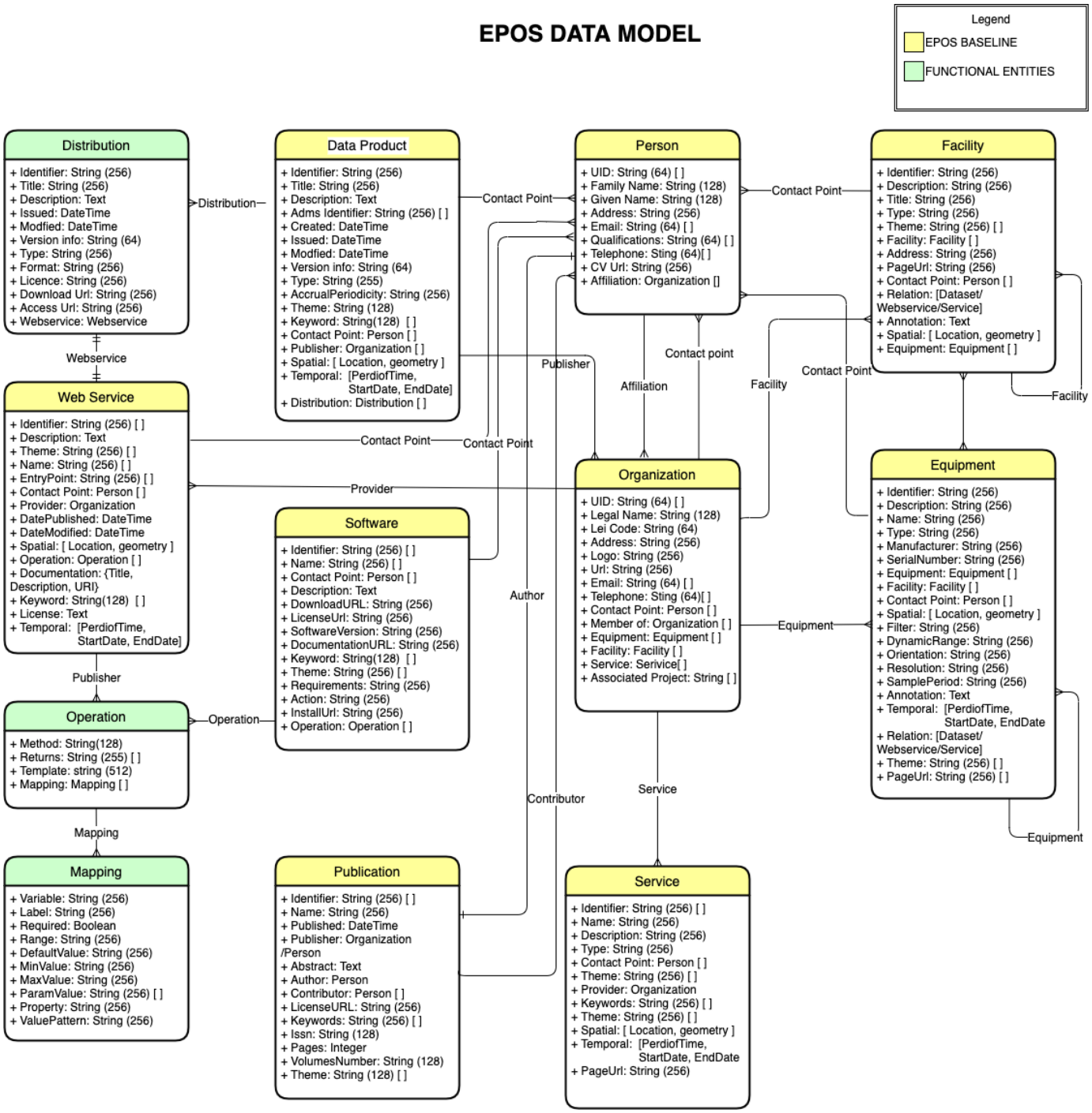
Such model is mostly conceptual, and serves for different purposes:

1. To define the structure of the information that the system provides to the end users by means of appropriate interfaces (e.g. web APIs)
2. To define the components of the system, that have the role of implementing specific functionalities

[1] Second (Final) Report on EPOS-ICS Architecture
https://zenodo.org/record/1470257#.YDO_QmpKhqt (last accessed 22/02/2021)

3. as a guideline to define inputs and outputs of the different system components

The EPOS data model was discussed and defined in EPOS-IP as of 2018 in the document "Second (Final) Report on EPOS-ICS Architecture"[2] and is here reported for reader's convenience in the following class diagram.



The EPOS Data model includes the following concepts:

[2] Second (Final) Report on EPOS-ICS Architecture https://zenodo.org/record/1470257#.YDO_QmpKhqt (last accessed 22/02/2021)

Person, Equipment, Facility, Service, Web service, Organization, Data, Software and mode code, Publication

The data model defines the objects that the user will have to deal with in the EPOS ecosystem. This has consequences, for instance, in the definition of the User experience and associated functionalities implemented by the System modules.

3.3. System Boundary

The following system boundary / context diagram represents external entities that may interact with the ICS-C, and to emphasise those that are outside of the scope of the system described in this document.

Source: https://app.diagrams.net/#G1NCdgQA0GumKjAS55PqTyZivK2mUznb_w

3.4. Architectural Goals and Constraints

Core Concepts and Patterns

a. Modular approach.

The system should be implemented with a modular architecture so that adding new functionalities and scaling current functionalities does not perturb the entire system. Modules should be decoupled and cohesive (implement one functionality). Microservice Architecture is the reference approach;

1. Each module should have a clearly defined functionality, and dependencies among modules should be avoided (e.g. decoupling of microservices).
2. Each module should be individually testable (e.g. rabbitmq endpoint should always answer with the same payload format)
3. Each module can be deployed (i.e. in any order)

b. Interaction between clients (GUI or Agent) and system

a. Web API should be designed to serve a specific category of consumers (e.g. GUIs clients, software agents harvesting metadata in specific formats, desktop applications etc.);
b. Web API should be decoupled from any specific client logic (e.g. GUI should not force constraints on the WEBAPI as information about the content to be downloadable or visualized);
c. Client should hide to the end users the potential complexity caused by a consistent number of negotiations needed to retrieve datasets; this may require for instance:
    i. Creation of complex agents within the client
    ii. Simplifying TCS services

iii. Using ad hoc metadata description

b) Web API should reuse existing patterns for facilitating adoption and reusage (FAIR)

c. Data and Metadata

a. Information used for satisfying user requirements are intrinsically interconnected and thus require to maintain referential integrity. (Scientific data, system (workspace) data, GRDB).
   Technically, they should be stored in a unique data source as much as possible (e.g. main CERIF metadata catalogue, including User Workspace);
b. Persistence within modules might be necessary for storing cache information (status, routing etc.): these are reused in case of scaling (e.g. container duplication); stateless modules usage is encouraged, so that there is no need to store internal state, which would ease scalability;
c. The main database containing information should be persistent over time and allow incremental updates (CRUD operations). Re-population, re-building, and total erasing are not sustainable (e.g. no full ingestion pipeline)

d. Authentication and Authorization

a. ICS-C system needs authentication, authorization, and profiling of users.

Quality Attributes[3]

According to the requirements in section XXX, the following quality attributes emerged:

1. Modularity
   The following requirements and constraints make this quality attribute relevant:
   a. needs to implement several functionalities each of which potentially implemented by different technologies.
   b. needs to be easy to maintain
   c. should enable different individuals or teams to work in parallel
   d. should be modular, so that modules can be replaced as long as the interfaces with the other modules remain the same
   e. Should support the use of several technological tools (e.g. AAAI system, database etc.)

2. Maintainability
   The following requirements and constraints make this quality attribute relevant:
   a. needs to implement several functionalities each of which potentially implemented by different technologies.
   b. needs to be easy to maintain
   c. should enable different individuals or teams to work in parallel
   d. modules can be replaced as long as the interfaces with the other modules remain the same

[3] https://en.wikipedia.org/wiki/List_of_system_quality_attributes

e. Should support the use of several technological tools (e.g. AAAI system, database etc.)

3. Deployability
   The following requirements and constraints make this quality attribute relevant:
   f. Should be easy to deploy
   g. Should guarantee a minimal level of performances, as defined in the technical Annex[4] (**click the note to check KPIs)**

4. Scalability
   The following requirements and constraints make this quality attribute relevant:
   a. Should be resilient to increasing amount of user or to incresing performance requirements

5. Securability
   The following requirements and constraints make this quality attribute relevant:
   a. Might provide security audit
   b. Must provide layered permission access (i.e.: Read vs Create, Write, Update)

6. Usability
   The following requirements and constraints make this quality attribute relevant:
   c. should be easy to interact with, for example reuse existing API where possible
   d. Should document API
   e. Might provide versioning of API message for old clients
   f. Should be resilient to invalid payloads

3.5. Architectural Assumptions and Decisions

Architectural Assumptions and Decisions are more concrete, may include a justification or reference ADRs, for example the use of a specific technology/ library/ format - DCAT, CERIF that influence the design, or inferences about implementation or the need of users

ttl/SHACL driven metadata ingestion and DCAT-AP as transfer format

The experience gained in managing EPOS metadata has taught that TCS communities wish to provide metadata either by a) autonomous systems, b) manually, but with a human understandable format.
For this reason, an extension of DCAT was done and is currently used to describe and transfer part of the metadata from the communities to the main system catalogue in CERIF.
EPOS-DCAT-AP is hence used as a metadata transfer layer.

[4] EPOS ERIC provided the following estimates of potential use of the system for the first year of operation:
- Average request per sec: 5/sec
- Max request per sec: 20/sec
- Maximum users per day: 1000

This requires a specific module of the system (the “ingestor”) to perform this action.

Proxying web services responses

Proxying web service is currently aimed at changing the changing the output format of a TCS: in order to harmonize the output standard of the ICS-C (usability) caching.

Error handling: how errors are managed (e.g. error gets logged)

Logging captures internal error handling; at an architectural level we need to know:

a. errors in one module that may prevent another from completing it role.
b. How/which errors are communicated to the user.
c. How an error is handled and rolled back if it leaves the system in an inconsistent state.

This can be done by means of Log files produced by the handling procedures within the code. A Log file needs to contain: the current timestamp, the java class, the row that raises the error and the session user (it can happen that an error occurs only for a certain group of users and not for other).

CERIF adoption

One of the Key aspects of the current architecture is the metadata. It is currently stored within the CERIF metadata catalogue.

The characteristics of CERIF that made it relevant for the current ICS-C implementation are the following:

a. it separates clearly base entities from relationships between them and thus represents the more flexible fully-connected graph rather than a hierarchy;
b. it has generalised base entities with instances specialised by role (for example <person> rather than <author>), the role specialisation is in the linking entities
c. it handles multilinguality and temporal information by design;
d. the temporal information in linking entities can provide provenance and versioning recording e.g. versions of datasets;
e. CERIF separates the semantics into a special ‘layer’ which is referenced from CERIF instances.
f. CERIF provides formal syntax and declared (multilingual) semantics. It also upholds referential and functional integrity in both syntactic and semantic layers.
g. Over the years euroCRIS[5] members have mapped other metadata standards to CERIF. Thus, CERIF has been used as a ‘switchboard’ for converting from one metadata standard to another thus permitting interoperability between sources with different metadata standards.

[5] EuroCRIS euroCRIS, founded in 2002, is an international not-for-profit association, that brings together experts on research information in general and research information systems (CRIS) in particular. The mission of euroCRIS is to promote cooperation within and share knowledge among the research information community and interoperability of research information through CERIF Website: https://eurocris.org/ (last viewed 24/02/2021)

h. The flexibility and formality of CERIF provides: (a) a canonical superset metadata standard that can be used to interoperate between other metadata standards; (b) a reliable representation of the world of interest for use in virtualising technologies.

Web Services approach (metadata with brokering architecture)

The other key for our system is the web-service based architecture. Two approaches have been considered at the beginning of EPOS:
A) collecting metadata describing ALL assets from the TCS, including *all* datasets, *all* stations, *all* software
B) assuming that access to assets information by TCS communities is provided by means of web services, and map only webservices into the CERIF catalogue.
Either approaches can be implemented thanks to the flexibility of our metadata model. However, it was immediately realized that TCS communities are so heterogenous that neither of the two approaches alone can be adopted. For instance, many communities already have web services that provide access to time series or list of stations, so it doesn't make sense to duplicate this metadata into CERIF and having to deal with all the issues related to synchronization and update of metadata between TCS and ICS-C systems.
A mix of the two approaches was needed.
As a result, *our architecture relies primarily on the web services by the TCS communities and the description of these into the metadata catalogue. Then, for the communities that are willing to do so, specific assets are mapped (e.g. equipment, facilities, software source code and others).*
The current architecture does not rely uniquely nor on web services nor on pure metadata mapping. For this reason, we called it the metadata with brokering approach.

Technological dependencies

- CERIF standard implementation is in PostgreSQL, so we use the same technology.

Per-component centralized routing

The architectural approach proposed in this document stems from discussions around architectural topics, one of these being how to best centralize/orchestrate the ICS-C business and routing logic.
The first version of the ICS-C system had the business and the routing logic widespread and distributed in the components. Each component indeed decided how to process the payload, which action performs and to whom to send next message in queue.
This hardcoded solution was not optimal, as at each change in the logic (routing) forced the developers to correct and update all components that handled the issue; in practice it meant to localize the java code point, to build and deploy the updates.
To avoid all these efforts which contradicts the principles of modularity and maintainability, a solution that centralized the business and routing logic of the ICS-C system was studied. As a result, it was decided to describe these logics in form of rules, which are easy to manage.

A first prototype with a central component was produced - the Event Manager dedicated to handle the business and the routing logic of all the other components.
The Event Manager used the Drools engine and it contained specific rules files for each component: through these rules the Event Manger decided how to process the payload, which action to perform and to whom to send next message in queue. The other ICS-C components are like slaves that perform actions upon notification by the Event Manager and communicate one with each other only through the queue.
The Drools rules files are easily updatable: indeed, each rule can be modified by the developer without build and deploy the Event Manger: just restart it.
With this prototype we have made the logic of the system centralized and easy maintainable.
A further problem then raised: centralized component means having a single point failure - the Event Manager, that becomes the strategic component of the ICS-C system.
To avoid this problem, the architecture was further refactored as described in this document.
Such refactoring keeps the Drools rules approach thus avoiding hardcoded logics, but it decentralized the Drool engines in each component or better to say in each module: indeed, the components become modules with specific job and dedicated api.

3.6. External Constraints

AAAI System

IP address authorization/URL process to authorize EPOS on the Cyfronet AAAI systemAuthentication protocol supported (and attributes)

TCS

We have to interoperate with external heterogeneous web services (pointers to documentation describing protocols and metadata standards)

3.7. Existing Architectures

We are reusing the EPOS ICS-C architecture as of 2018 version, here referenced:
- https://epos-ci.brgm.fr/epos/devops-documentation/-/blob/master/ics-c_architecture.md

3.8. Architectural Risks

1. Rule-engine was prototyped but not deployed in production yet; minor risk.
2. Adopting a single database brings the advantage of (referential) integrity, but also imply risks conneted to lower decoupling:

i. it might imply dependencies on teams working on specific components in an isolated way
ii. Future requirements might require more separation of components.

4. Architecture design

This section describes the architecture of the EPOS ICS-C system.
In order to differentiate concepts from implementation practices, we adopt a twofold approach: in the first sub-section, we present the conceptual design, that focuses on the concepts to be implemented at architectural level; each concept can be rendered in one or more modules; in the second sub-section, we present the implementational architecture, that is to say what is going to be implemented in practice.
The latter architecture only describes the modules and does not get into the details of the actual implementation (e.g. chosen programming language and other very technical details) that can however be found on the Gitlab environment hosting the codebase.

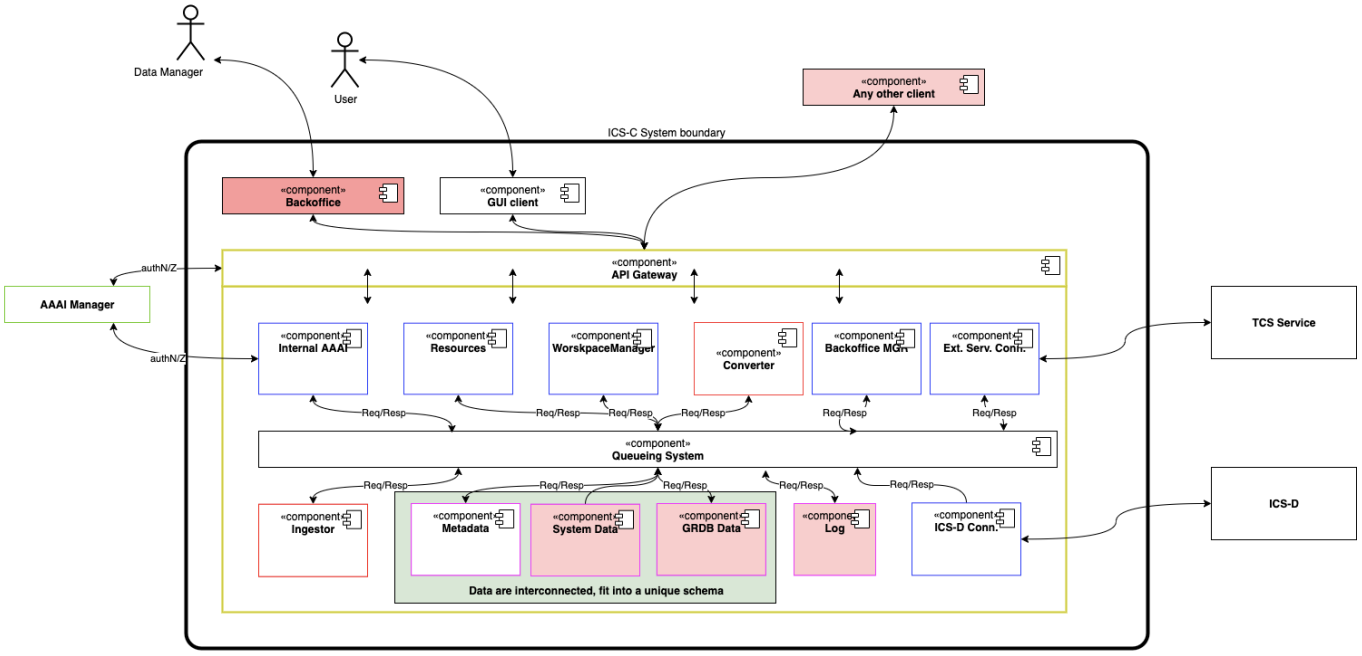
a. Conceptual design

i. Overview and Diagram

The following diagram describes the conceptual components of the architecture.
The driving concepts can be summarized by the following statements:

1. Each of the components is supposed to capture a well-defined functionality
2. Components interaction is guaranteed by one main component, namely the queueing system
3. Components interact through messages
4. A main “API gateway” component ensures interaction between the system and external users (humans or machines)
5. Components outside the ICS-C system boundary are external systems with which the ICS-C system is supposed to communicate and interoperate (AAI manager, ICS-D, TCS service)
6. Components in the blue squares are implement functionalities required by the user and actionable by means of a Web-API(proxied by the API gateway).
7. Currently, the following main functionalities emerged from the user requirements, here represented as Web-API endpoints”:
    7.1. “/resources”, for accessing assets stored in the ICS-C catalogue
    7.2. “/execute”, for executing external services (from TCS)
    7.3. “/workspace”, for managing the user workspace
    7.4. “/backoffice”, for managing metadata from the backoffice application
    7.5. “/conversion”, for managing payload or metadata conversion
    7.6. “/ICS-D”, for interacting with potential ICS-D (currently a placeholder for future work)

8. Components in the red squares implement “support” functionalities, that is to say functionalities needed by the system, internally. Although the converter is supposed to be an internal "support" component, this may implement Web-APIs to provide an additional service to end users.
9. Data elements were differentiated in three different kinds:
    9.1. “Metadata”, representing the “scientific metadata” that enable the system to manage the interaction with the TCS, and is mostly provided by the communities.
    9.2. “System data”, representing data needed by the users of the ICS-C system that should persist after the end of the session. It includes for instance information about workspaces, and may include workflow information.
    9.3. “GRDB data”, this represents the metadata currently stored in the Granularity Database, and partially overlapping with the “metadata” component
10. The three data element above represent the data that the ICS-C system is required to manage. Such data that is intrinsically related, and requires referential integrity. For instance, a workspace (system data) is related to a user and to services (metadata) selected by users, and each update to the services needs to be reflected into the workspace for integrity reasons.



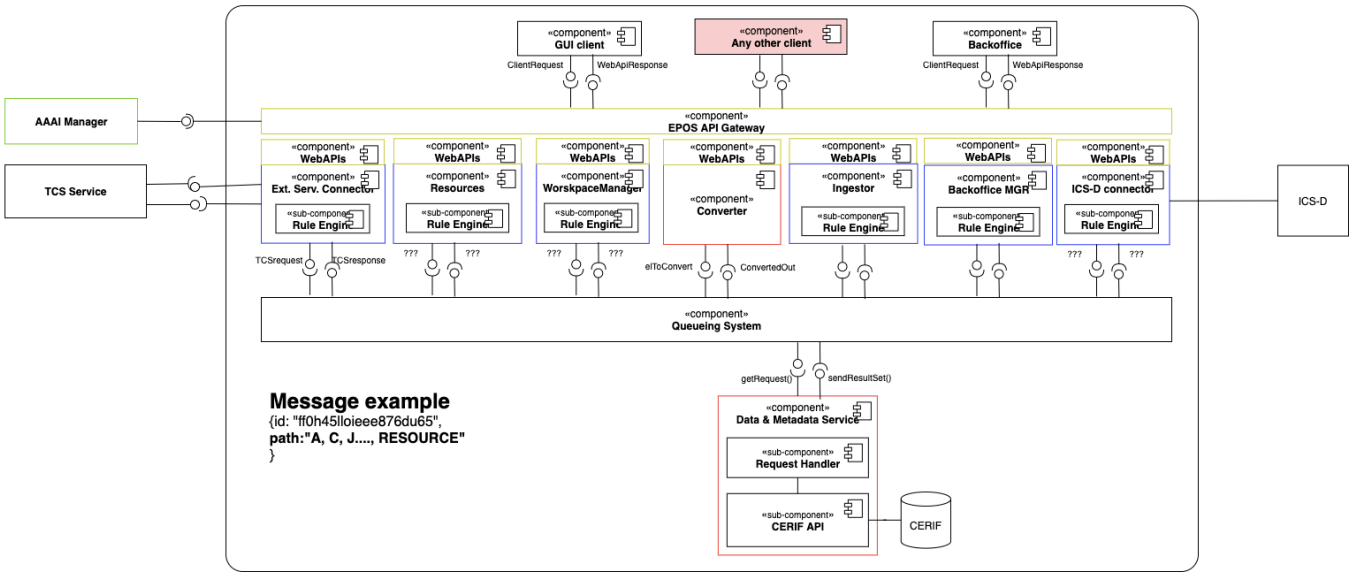
Source diagram: https://app.diagrams.net/#G1WobjN59cMpDOKzfHkGPmzh0ye4a8UEpI

b. Technical design

i. Overview and diagram

The diagram in this section represents the implementation of the conceptual architecture. The following key aspects need to be taken into account:

1. Each of the *components* in the conceptual design is rendered as one or more *modules* in the Technical Design. Modules are very close to the microservice concept, but do not match fully with it, so it is preferable to call these “modules”.
2. Blue modules represent functionalities that implement request by specific WEB-APIs endpoints, namely:
    a. “/resources” is implemented by the Resource module
    b. “/execute” is implemented by the External Service Connector module
    c. “/workspace” is implemented by the Workspace module“/backoffice” is implemented by the Backoffice MGR module
    d. “/ics-d” will be implemented by the ICS-D connector module (placeholder)
3. In addition, the following modules have been supplied with WEB-APIs:
    a. Converter: its task is to convert from one data format to another data format. This might be useful for external users that wish to convert metadata on demand
    b. Ingestor: this is required by the backoffice client
4. The selected standard to implement the above Web-API is RESTful because of the advantages connected to the stateless API architecture
5. Rule engines: some modules have a sub-module named “rule engine”. This is in charge of “collecting” the information from other modules or services and aggregate it in order to answer to the user request. This is done by defining the routing of the message that need to interact with the modules providing the information to be aggregated.
6. Messaging: the rule engine defines the routing of each message. The idea here is to embed the routing path into the message header in order to avoid messages to go back to the rule engine at each step (for being routed to the next module).
   This choice is motivated by the fact that a centralized management of the messaging (i.e. rule engine taking decision at each step of the path) increases the amount of messaging in the queuing system.



Source diagram: https://app.diagrams.net/#G1WobjN59cMpDOKzfHkGPmzh0ye4a8UEpI

ii. Modules Communication

Communications among modules occurs by means of a Queuing system.
Message exchange follows the principle of Saga-Pattern[6].
Each module uses two internal frameworks in order to take advantage of the Queueing system features:

- *RoutingFramework*: This framework deals with connecting the component to the queuing system, managing the connections; the generation and sending / receiving of messages.
- *Rule Engine*: This framework decides which path a message must follow to aggregate the necessary information. This decision is made on the basis of a rule.

In order to implement this, once a request has been made on a module, the following happen:

1. Inside the module:
    1.1. Once a request has been made to the component's RESTful API, the Rule Engine activates a request-specific rule
    1.2. The Rule Engine, through the rule, deploys a path that includes the module required to aggregate the information
    1.3. The RoutingFramework creates the message to be sent to the Queuing System and put the information about the routing path in the message header
2. On the queueing system, “across modules”:
    2.1. At each step, on the basis of the route written in the header, the message is properly sent to the right queue (hence right module).
    2.2. In case of conditionals or decisions, the deployed route will contain the originator module, so that appropriate decision can be made by the rule engine.

Message header example, originated by the resource microservice:

```
Node:          rabbit@rabbitmq
Connection:    ip:port -> ip:port
Virtual host: epos
User:          epos
Channel:       xyz
Exchange:      <<Resources>>
Routing keys: [<<...>>]
Routed queues: [<<...>>]
Properties:    [{<<"expiration">>,longstr,<<"10000000">>},
                {<<"correlation_id">>,longstr,
                 <<"214910843203799d19d4e-0cd4-45df-afa9-
73f3e5b1a89e149639689">>},
                {<<"headers">>, table,
                 [{<<"x-request-id">>,longstr,
                   <<"c70d3fcdfe9b33cf7e70c1922a7c1243">>},
                  {<<"referer">>,longstr,
                   <<"https://some-address/gateway/resources">>},
```

[6] https://microservices.io/patterns/data/saga.html

```
                {<<"route">>,array,["data&metadata_service",
"convertor", "resources"]},
...
```

iii. Module description template

Description: responsibility and rationale (include diagrams as necessary, including what logical unit it contributes to and how)
Interactions with other elements (include diagrams as necessary, APIs, data, messages)
Error conditions and handling
Process view
Deployment considerations (how it's deployed, how it's been designed to handle scaling thing like that)
Implementation detail: be selective, only include what adds benefit or affects other elements of the system (hopefully this is zero or minimal), or requires investigation/prototyping, or just needs to be documented and "imposed" upon the component implementation for some justifiable reason.

iv. Resources

Module name: Resources
Role: Discovery
Description: Discovery resources on metadata catalogue. It requires no authentication. It is used to provide a list of all available resources (services, software, organization etc.).
Interactions (with other parts of the system): queueing system, rule-based (Data & Metadata Service, Converter)
Inputs and outputs (system side): messages
Error handling:
Implementation detail (optional): it includes the rule engine framework
Process: see sequence diagrams
Deployment: n/a

Associated endpoints (for architectural purposes, full documentation may include others):

- **/resources:** returns a JSON with a list of all resources
- **/resources/{resource_type}** returns a JSON with all resources of a certain type
- **/resources/{resource_type}/{id}** returns a JSON with details about a specific resource identified by an {id}

Endpoint response structure:

-

v. External Services Connector

Module name: External Services Connector
Role: Access external metadata and data
Description: Access to external services from ICS-C in order to gather data and metadata to be used by clients
Interactions (with other parts of the system): queueing system, rule-based (Data & Metadata Service, Converter)
Inputs and outputs: messages, TCS Webservices API
Error handling:
Implementation detail (optional): it includes the rule engine framework
Process: see sequence diagrams
Deployment: n/a

Associated endpoints (for architectural purposes, full documentation may include others):

- **/access:** returns a Payload or a redirect from a TCS endpoint

Endpoint response structure:

-

vi. WorkspaceManager

Module name: Workspace Manager
Role: Workspaces
Description: It provide the service to store selected and/or customized metadata information by users
Interactions (with other parts of the system): queueing system, rule-based (Data & Metadata Service, Converter)
Inputs and outputs: messages
Error handling:
Implementation detail (optional): it includes the rule engine framework
Process: see sequence diagrams
Deployment: n/a

Associated endpoints (for architectural purposes, full documentation may include others):

- **/workspaces:** returns a list of workspaces from a user
- **/workspaces/{workspaceid}:** returns a workspace from a user with a list of items
- **/workspaces/{workspaceid}/item/:** returns a list of items for a workspace
- **/workspaces/{workspaceid}/item/{itemid}:** returns an item from a user workspace with a list of configurations
- **/workspaces/{workspaceid}/item/{itemid}/configurations:** returns a list of configurations for a workspace item
- **/workspaces/{workspaceid}/item/{itemid}/configurations/{configurationid}:** returns a configuration from a user workspace item

Endpoint response structure:

-

vii. Data&Metadata Service

Module name: Data&Metadata Handler
Role: Handling Database Requests
Description: This service handle request received from modules that need to be executed on the database
Interactions (with other parts of the system): queueing system, metadata catalogue
Inputs and outputs: messages
Error handling:
Implementation detail (optional): it includes the new CERIF API as internal library in order to easily query the metadata catalogue
Process: see sequence diagrams
Deployment: n/a

viii. Converter

Module name: Converter
Role: Conversions of payloads
Description: It provides a service which enable the conversion of metadata/data payloads, based on a plugin system
Interactions (with other parts of the system): queueing system, Data & Metadata Service
Inputs and outputs: messages
Error handling:
Implementation detail (optional): the application use a set of plugin, each plugin could convert on single kind of payload. Information about plugins are stored into the metadata catalogue
Process: see sequence diagrams
Deployment: n/a

Associated endpoints (for architectural purposes, full documentation may include others):

- **/convert:** returns a list of workspaces from a user
- **/plugins**: returns a list of available plugins
- **/plugins/{id}:** return the information about a single plugin

Endpoint response structure:

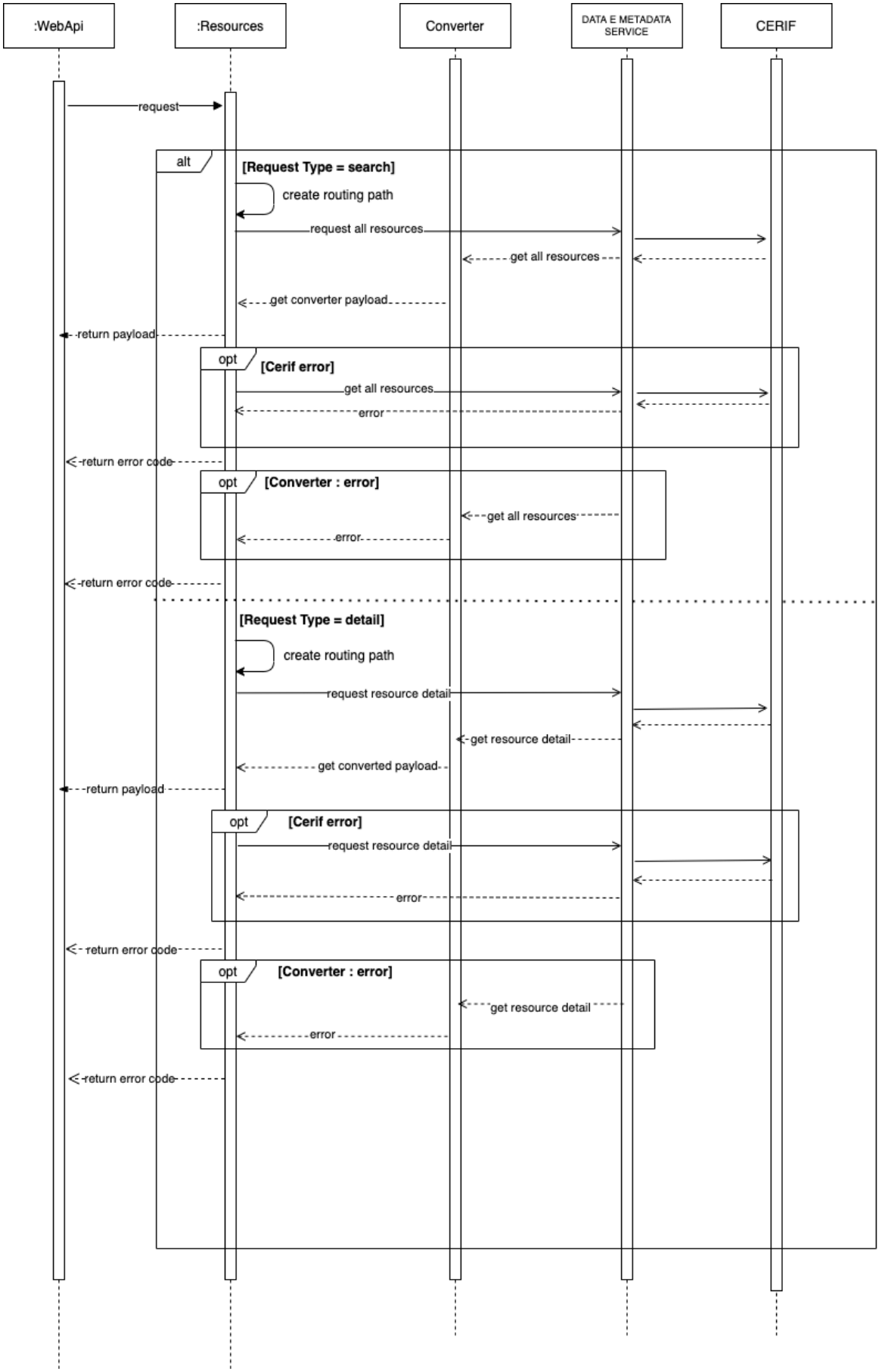
-

c. Sequence Diagrams

Sequence diagram represent the execution of specific API endpoint of each module.

i. Resource Sequence Diagram

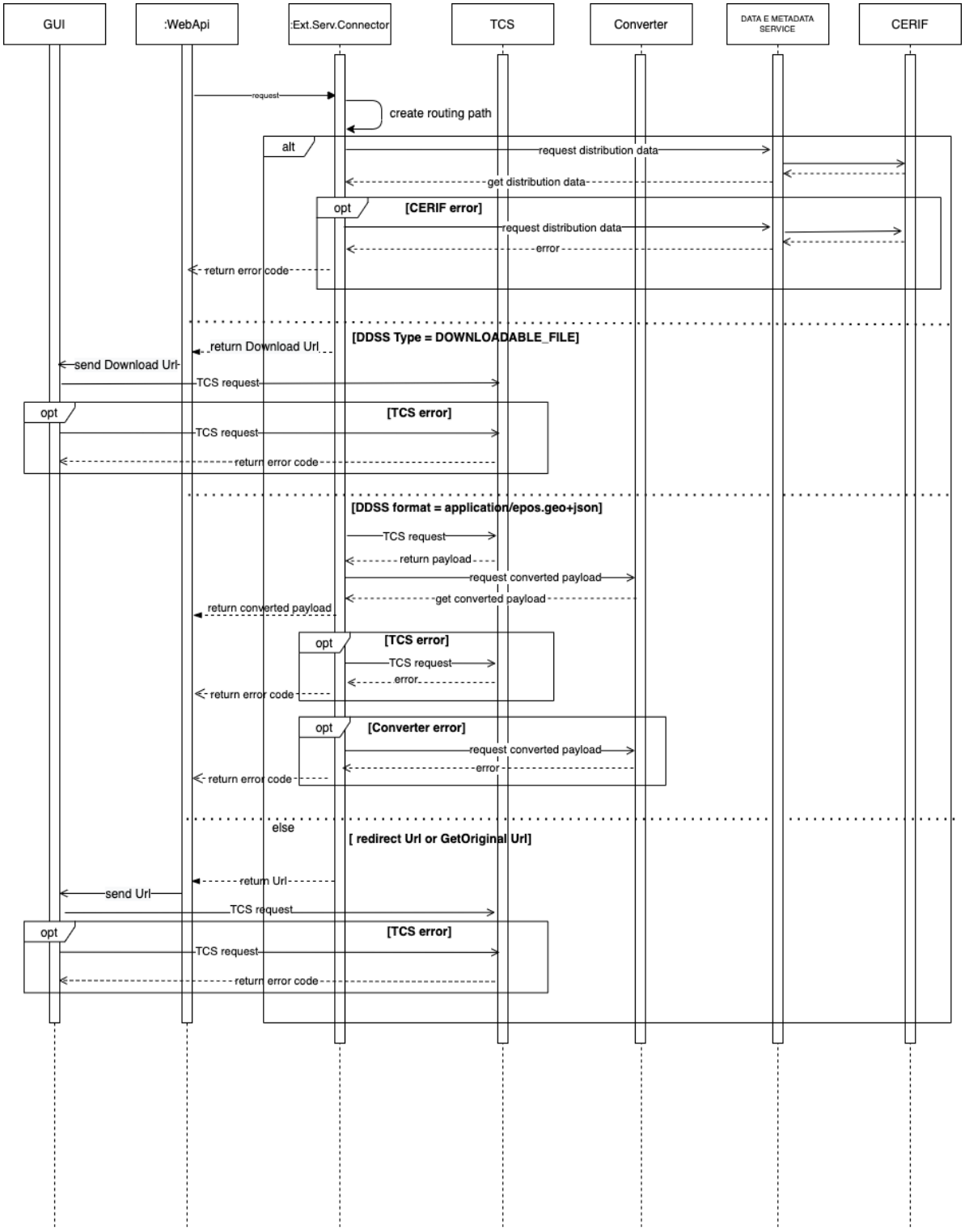
The Resource module has two endpoints: /search and /detail.
The /search endpoint returns the entire range of resources available and catalogued within the metadata catalogue (e.g. services, distributions etc.).
This information is extracted from the metadata catalogue in the form of *resultset;* resultsets are then sent to the Converter, where these are converted into the format used for the payload returned to the GUI.
The /detail endpoint returns the detailed metadata of a specific resource.
The process to provide the detailed metadata is similar to the route described above for the /resources endpoint.
The following diagram describes in detail such process.


:WebApi
:Resources
Converter
DATA E METADATA SERVICE
CERIF
request
alt
[Request Type = search]
create routing path
request all resources
get all resources
get converter payload
return payload
opt
[Cerif error]
get all resources
error
return error code
opt
[Converter : error]
get all resources
error
return error code
[Request Type = detail]
create routing path
request resource detail
get resource detail
get converted payload
return payload
opt
[Cerif error]
request resource detail
error
return error code
opt
[Converter : error]
get resource detail
error
return error code

Source:
https://app.diagrams.net/?mode=google&gfw=1#G1mNvAmFlqgBqa7oTzXBZnOAZhLEBgc_OE

ii. External Service Connector Sequence Diagram

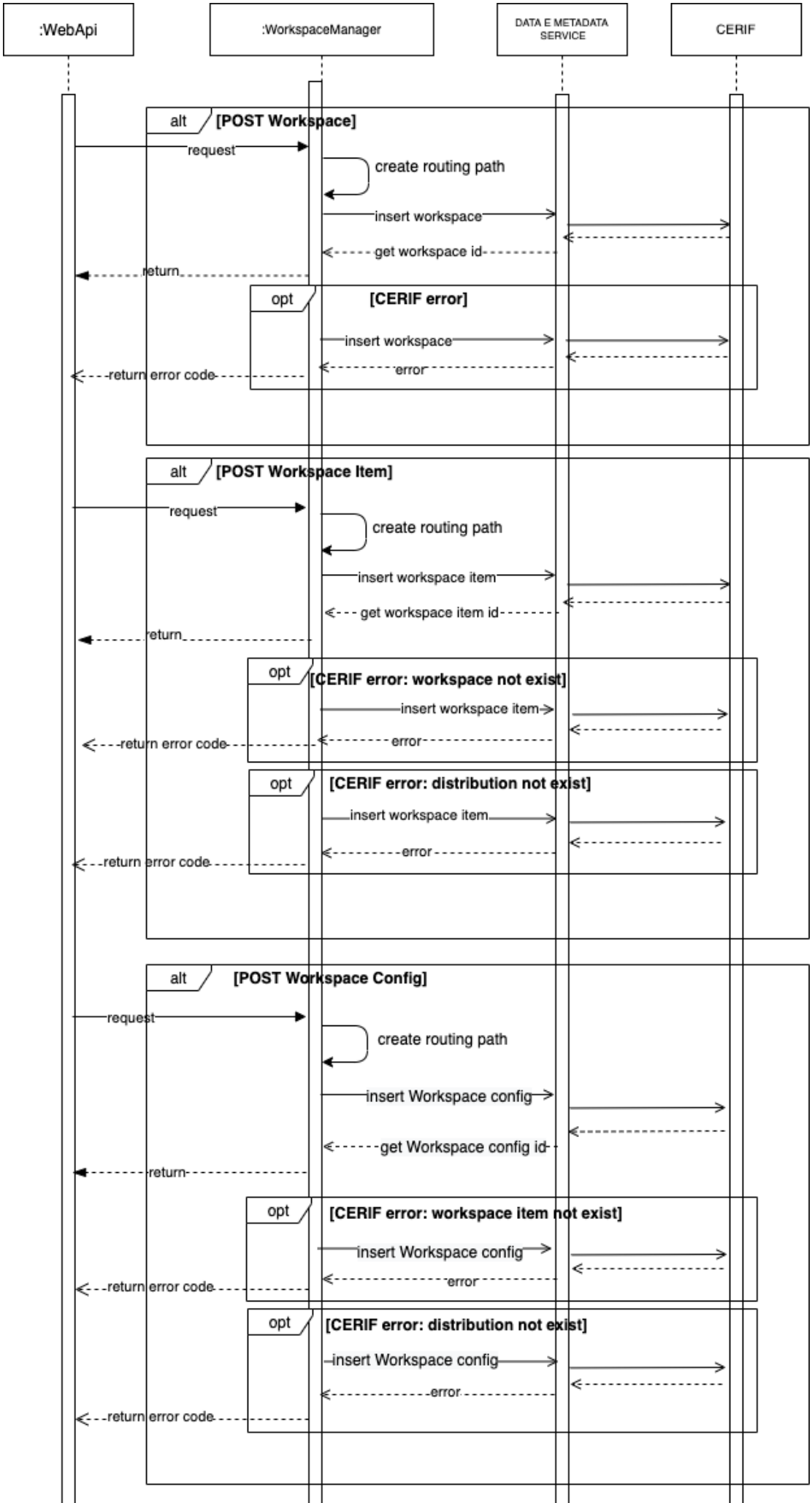
The External Service Connector has two endpoints: /execute and /getoriginalurl.
The /execute enpoint executes a specific query on the TCS endpoints and returns the payload.
In the execute endpoint a request to Cerif DB is first made to get the metadata about the requested distribution. Then the system behaviour depends on the DDSS returned.
If DDSS type is DOWNLOADABLE_FILE the External Service Connector returns to the GUI the url to download file, if DDSS format is application/geo+json the returned payload from TCS is processed by the Converter.
The /getoriginalurl edpoint returns the URL of a configured TCS.
In the /getoriginalurl endpoint the External Service Connector returns the redirect url to the GUI, and the GUI execute a call directly to TCS.



Source: https://app.diagrams.net/?mode=google&gfw=1#G1Dde1Y0bWSuGOmqES8OLmng2bpjocqkjm

iii. Workspace Manager Sequence Diagram – POST requests

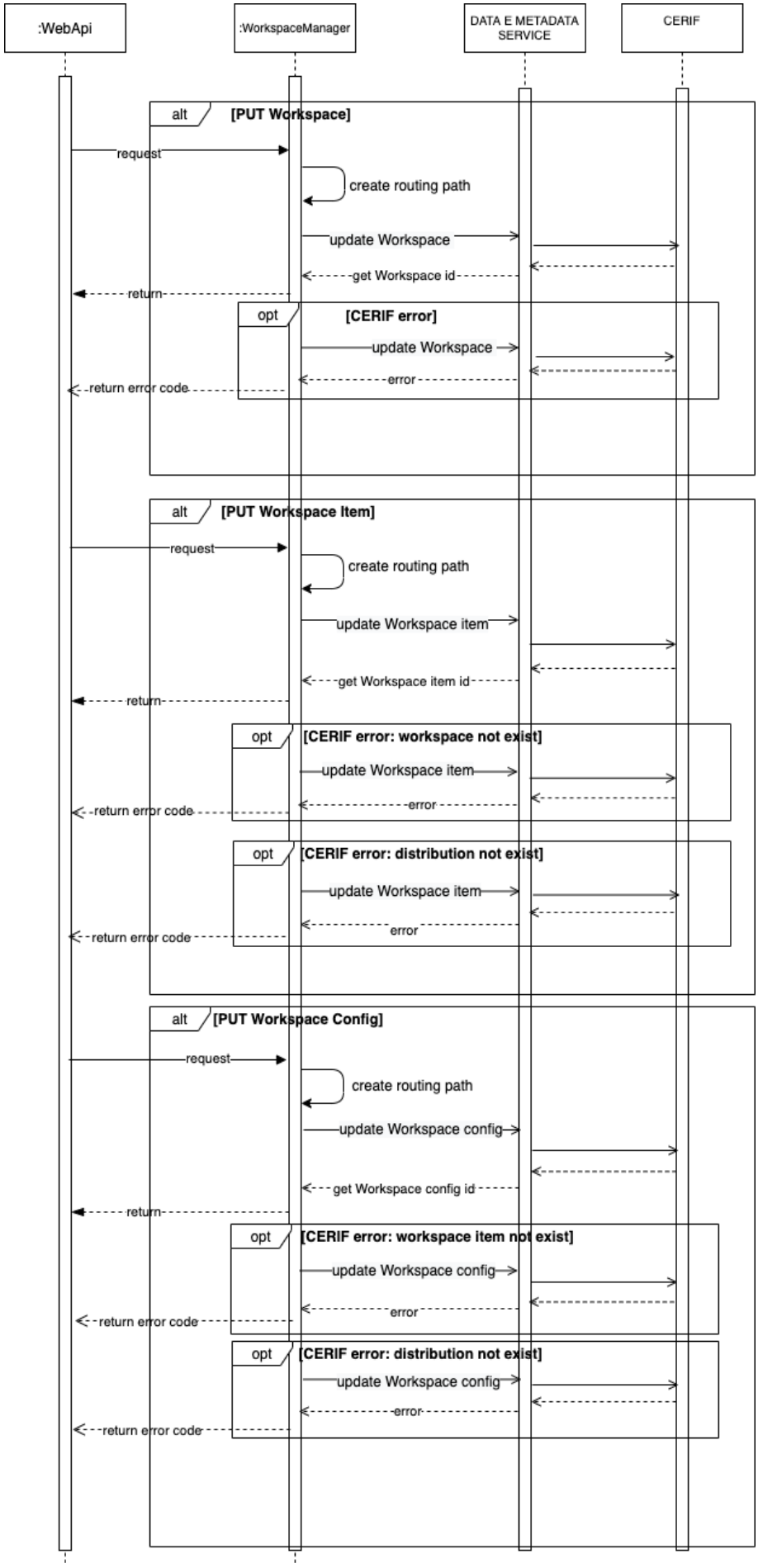
In the Workspace Manager the POST requests are dedicated to the creation of a new workspace, item and configuration. These diagrams assume that the user is correctly logged.
The POST request for the creation of a configuration assumes that both a workspace and an item exist and can be associated to the configuration.
The POST request for the creation of an item assumes that a workspace exist and can be associated to the item.


:WebApi
:WorkspaceManager
DATA E METADATA SERVICE
CERIF
alt
[POST Workspace]
request
create routing path
insert workspace
get workspace id
return
opt
[CERIF error]
insert workspace
error
return error code
alt
[POST Workspace Item]
request
create routing path
insert workspace item
get workspace item id
return
opt
[CERIF error: workspace not exist]
insert workspace item
error
return error code
opt
[CERIF error: distribution not exist]
insert workspace item
error
return error code
alt
[POST Workspace Config]
request
create routing path
insert Workspace config
get Workspace config id
return
opt
[CERIF error: workspace item not exist]
insert Workspace config
error
return error code
opt
[CERIF error: distribution not exist]
insert Workspace config
error
return error code

https://app.diagrams.net/?mode=google&gfw=1#G1hmdmcGBTdVl4foME_4Gh-T5eMG7ZHe6D

iv. Workspace Manager Sequence Diagram – PUT request

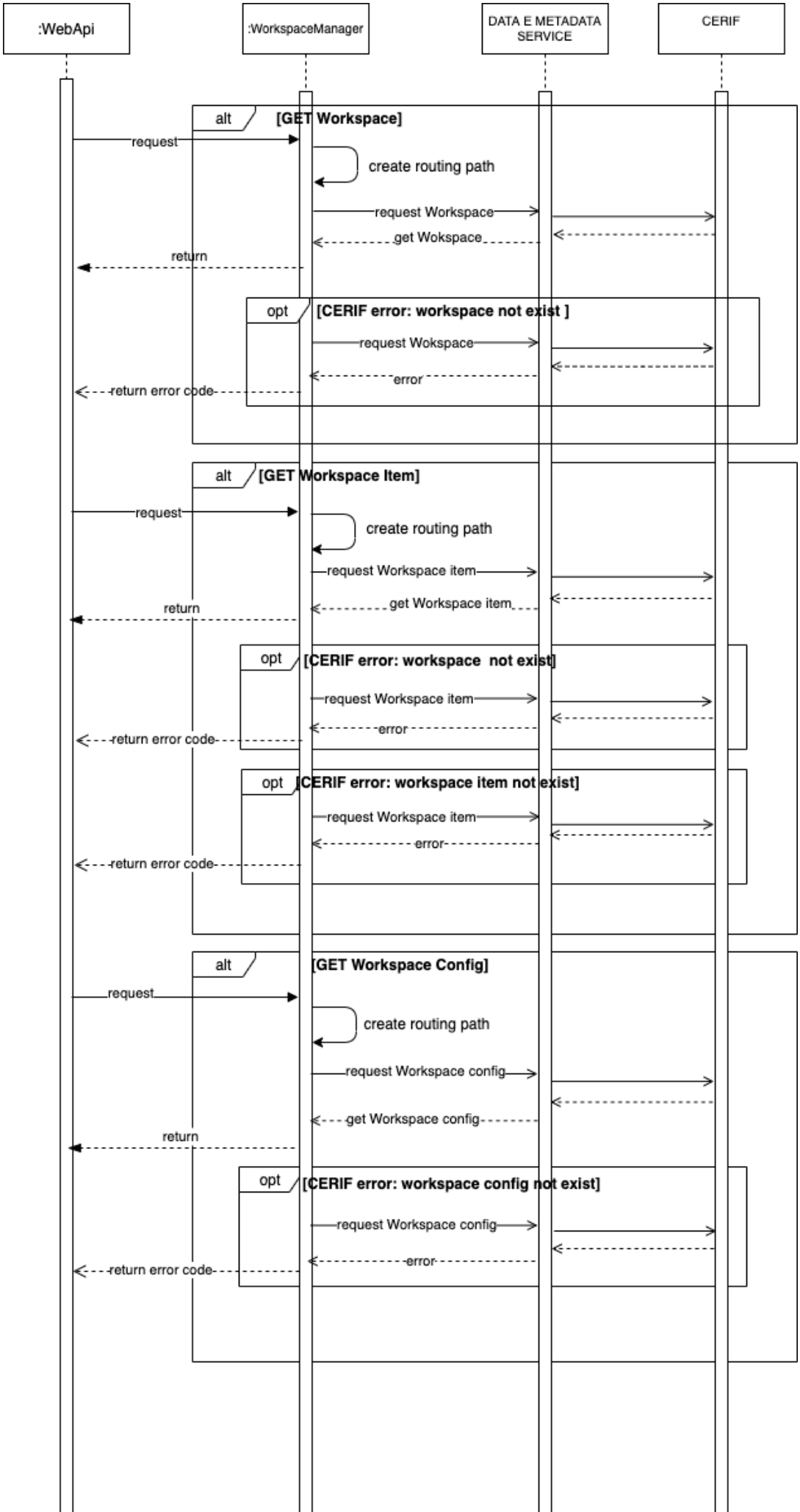
In the Workspace Manager PUT requests are dedicated to updating an existing workspace. These diagrams presume that the user is correctly logged.
The PUT doesn't follow a precise flow: you can update only a workspace data, or an item data or a config data.


:WebApi
:WorkspaceManager
DATA E METADATA SERVICE
CERIF
alt
[PUT Workspace]
request
create routing path
update Workspace
get Workspace id
return
opt
[CERIF error]
update Workspace
error
return error code
alt
[PUT Workspace Item]
request
create routing path
update Workspace item
get Workspace item id
return
opt
[CERIF error: workspace not exist]
update Workspace item
error
return error code
opt
[CERIF error: distribution not exist]
update Workspace item
error
return error code
alt
[PUT Workspace Config]
request
create routing path
update Workspace config
get Workspace config id
return
opt
[CERIF error: workspace item not exist]
update Workspace config
error
return error code
opt
[CERIF error: distribution not exist]
update Workspace config
error
return error code

Source: https://app.diagrams.net/?mode=google&gfw=1#G1juCBwogbh-IzAjlZObT_CM7f5T6jG8Bp

v. Workspace Manager Sequence Diagram – GET request

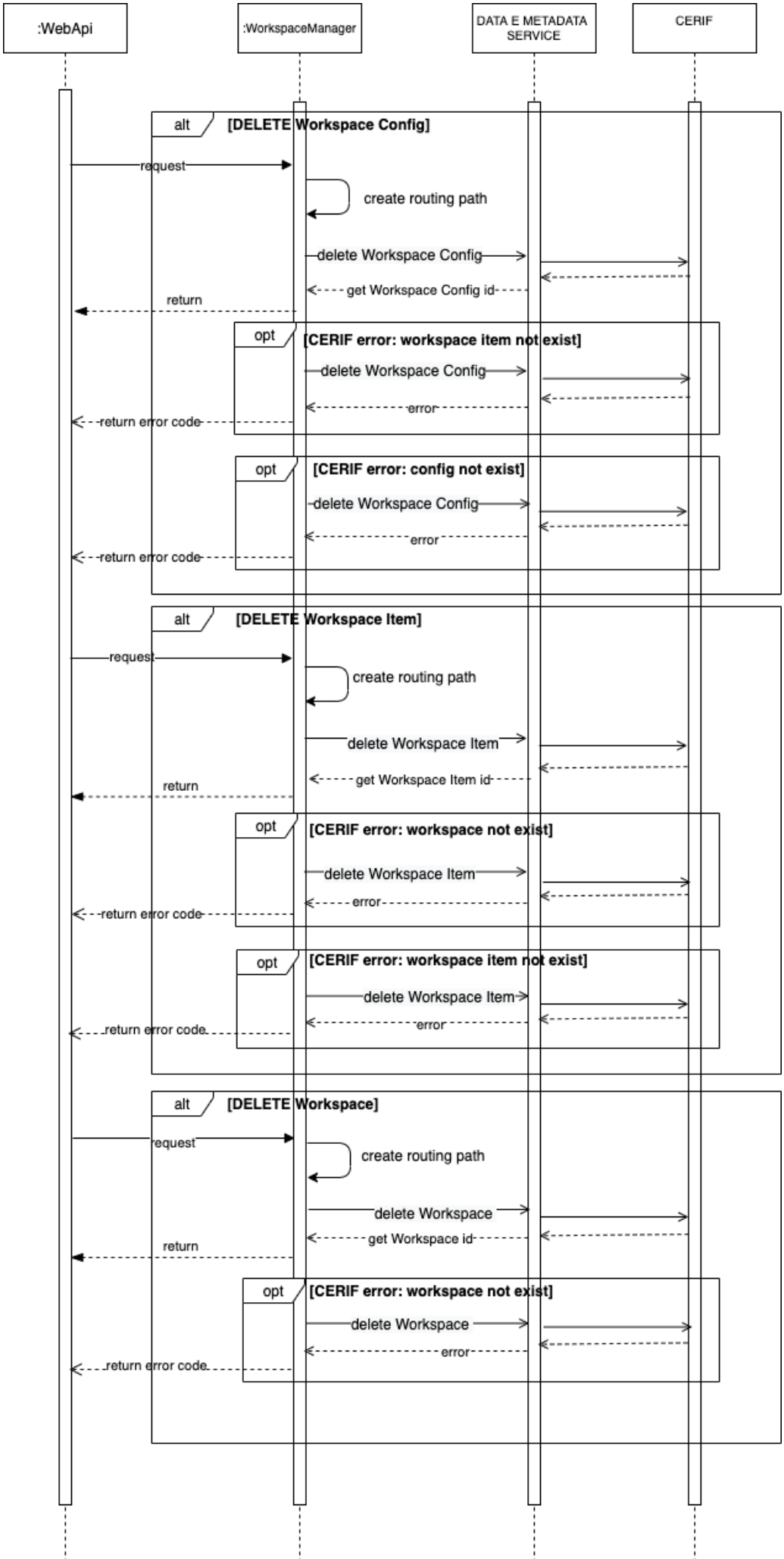
In the Workspace Manager GET requests are dedicated to reading an existing workspace information. These diagrams presume that the user is correctly logged.
The GET doesn’t follow a precise flow: it's possible read only a workspace data, or an item data or a config data.


:WebApi
:WorkspaceManager
DATA E METADATA SERVICE
CERIF
alt
[GET Workspace]
request
create routing path
request Workspace
get Wokspace
return
opt
[CERIF error: workspace not exist]
request Wokspace
error
return error code
alt
[GET Workspace Item]
request
create routing path
request Workspace item
get Workspace item
return
opt
[CERIF error: workspace not exist]
request Workspace item
error
return error code
opt
[CERIF error: workspace item not exist]
request Workspace item
error
return error code
alt
[GET Workspace Config]
request
create routing path
request Workspace config
get Workspace config
return
opt
[CERIF error: workspace config not exist]
request Workspace config
error
return error code

Source:
https://app.diagrams.net/?mode=google&gfw=1#G1TQLBC9D88shQyDEG3NaSAq2O1hWx6bek

vi. Workspace Menager Sequence Diagram – DELETE flow

In the Workspace Manager DELETE requests are dedicated to deleteing a workspace, an item or a configuration. These diagrams presume that the user is correctly logged.
Deleting a workspace requires the deletion of all associated items, and for deleting an item requires the deletion of associated configurations.


:WebApi
:WorkspaceManager
DATA E METADATA SERVICE
CERIF
alt
[DELETE Workspace Config]
request
create routing path
delete Workspace Config
get Workspace Config id
return
opt
[CERIF error: workspace item not exist]
delete Workspace Config
error
return error code
opt
[CERIF error: config not exist]
delete Workspace Config
error
return error code
alt
[DELETE Workspace Item]
request
create routing path
delete Workspace Item
get Workspace Item id
return
opt
[CERIF error: workspace not exist]
delete Workspace Item
error
return error code
opt
[CERIF error: workspace item not exist]
delete Workspace Item
error
return error code
alt
[DELETE Workspace]
request
create routing path
delete Workspace
get Workspace id
return
opt
[CERIF error: workspace not exist]
delete Workspace
error
return error code

Source:
https://app.diagrams.net/?mode=google&gfw=1#G1R2YYCBc7MtUwK5bRlNYiphGs0jT0q8pu