# D7.2 Second (final) Report on EPOS-ICS Architecture

**Document information Summary**

| | |
|---|---|
| **Date** | 17 September 2018 |
| **Document title:** | Second (Final) Report on EPOS-ICS Architecture |
| **Leader Partner** | UKRI (BGS) |
| **Main Author(s):** | Keith Jeffery, Matt Harrison, Kuvvet Atakan, Daniele Bailo |
| **Contributing author(s):** | Chris Card, Jan Michalek, Valerio Vinciarelli |
| **Reviewer(s):** | - |
| **Approved by:** | PMO; Implementation Phase Council |
| **Target audiences:** | Project partners, European Commission |
| **Keywords:** | EPOS-ICS; Architecture |
| **Deliverable nature:** | Key Deliverable; Report |
| **Dissemination level:** | Confidential |
| **Delivery date:** | M36 |
| **Version:** | 1 |

# Contents

# SUMMARY

This deliverable describes the ICS-C final architecture. Based on user satisfaction with the architectural design and simple prototype of EPOS-PP (Preparatory Phase) the initial architecture was defined. During the period M1-M18 of EPOS-IP (Implementation Phase) the architecture was refined based on interactions with the TCS and presented at EPOS project meetings. During the period M19-M36 progressive iterative prototypes driven by evolving user requirements and aspirations have been developed allowing the architecture to be specified in much more detail and the components refined and implemented. For some components (ICS-D, CES) implementation is continuing because this requires especially close working with the TCS.

Detailed work has been undertaken validating the ICS-C against the evolving and increasingly ambitious user requirements and – in particular – collecting the metadata describing the assets in the TCS to populate the catalog.

The architecture has been designed using the latest advances in metadata (for the catalog) and architectural approach (microservices). A consistent spiral, agile systems development method has been used.

As part of this work the teams of WP6 and WP7 of EPOS – each spread across several organisations – have been integrated into a functioning unit with appropriate skills and abilities for the tasks. There has been some delay in recruitment to provide the human resources required but this has been overcome and the work is on schedule.

# 1. Introduction

The purpose of this deliverable is to document the final architecture of the EPOS-ICS (EPOS Integrated Core Services) now being developed and implemented within EPOS-IP. It is based upon the demonstrated architecture developed as a first prototype in EPOS-PP and refined through subsequent iterations within EPOS-IP.

# 2. The Requirements

## 2.1 Introduction

The EPOS project may be considered a journey. During EPOS-PP disparate TCS (Thematic Core Services) communities discovered their commonality and differences and – particularly their digital assets. These were documented in the RIDE database which demonstrated clearly (a) that considerable assets existed; (b) that the organisations represented within the TCSs were willing to make them available (sometimes subject to conditions); (c) that there was overlap of assets between some TCSs; (d) that multidisciplinary geoscience could be achieved by providing appropriate interoperation mechanisms to make the assets available to all. The task of EPOS-IP is to build a geoscience environment (including governance, legal, financial, training and social aspects as well as technical ICT contributions) for the community. This deliverable describes the core IT component of EPOS-IP.

## 2.2 Method of Collection

The requirements were collected by use cases and discussions with the TCSs. In parallel, their DDSS (Data, Data Products, Software and Services) assets were collected, together with organisational and contact information and any conditions of use.

As a part of the requirements and use cases collection (RUC) from the TCS WPs, a specific list was prepared to include all data, data products, software and services (DDSS). This DDSS Master Table was used as a mechanism to update the RUC information as well as providing a mechanism for accessing more detailed IT technical information for the development of the ICS Central Hub (ICS-C). The DDSS Master Table was also used for extracting the level of maturity of the various DDSS elements in each TCS as well as providing a summary of the status of the TCS preparations for the ICS integration and interoperability. The current version of the DDSS Master Table (M34, 2018-08-01) consists of 363 DDSS elements (comparing to 454 in M12), where 160 of these already exists and are declared by TCSs to be ready for implementation during the first 24 months from M12 (i.e. until M36). The remaining DDSS elements required more time to harmonize the internal standards, prepare an adequate metadata structure and so are available for implementation from M36. In total, 21 different harmonization groups (HGs) are established to help organizing the harmonization issues in a structured way. TCS WPs are preparing individual TCS Roadmaps which will describe the development and implementation plans of the remaining DDSS elements including a time-line and resource allocations. In addition, user feedback groups (UFGs) are being established in order to give constant and structured feedback during the implementation process of the TCS-ICS integration and the development of the ICS.

The DDSS Master Table was constantly being updated as new information from the TCS WPs arrive. The most updated version is uploaded to the EPOS-IP Intranet pages (EPOS-IP ownCloud archiving system - https://owncloud.epos-ip.org/) under the shared folder (8 Transversal WPs/WP06 & 07/DDSS-Master-Table). The older versions are also kept in the archive for future reference. During the period M30-M36 the DDSS master table was transformed to the granularity database [https://epos-no.uib.no/granudb/loginUI] because of the problems of referential and functional integrity using a spreadsheet; relational technology provides appropriate constraints to ensure integrity.

The TCS requirements and use cases (RUC) collection process was designed carefully, taking into account the amount and complexity of the information involved in all 10 different TCS WPs. An increasingly detailed RUC collection process is formulated and explained through dedicated guidelines and interview templates. A roadmap for the ICS-TCS interactions for the RUC collection process was prepared for this purpose and distributed to all TCS WPs. The same document is also uploaded to the shared folder of the EPOS-IP intranet area (see also Deliverable D6.1 Appendix-1).

In this approach, a five-step procedure is applied involving the following:

- Step 1: First round of RUC collection

- Step 2: Second round of RUC collection
- Step 3: ICS-TCS Integration Workshop
- Step 4: Third round of RUC collection
- Step 5: Implementation of RUC to the CERIF metadata

Planning for the requirements and use cases (RUC) elicitation process started with the pre-project meeting held during the period July 8-9 2015 at the BGS facilities in Nottingham, UK. The first version of the guidelines level-1 for the ICS-TCS integration (Deliverable D6.1 Appendix-2) was prepared soon after this meeting and was distributed to the TCS WP leaders and the relevant IT-contacts. A second, more detailed guidelines level-2 (TCS-Handbook for ICS Integration) was prepared in September 2015 and distributed in the EPOS-IP project kick-off meeting held in Rome, Italy, during the period October 5-7 2015 (Deliverable D6.1 Appendix-3). Prior to the kick-off meeting, a preliminary collection of the RUC was requested from each TCS WP, which was then presented during the meeting. This first preliminary version of the RUC collection is given in Deliverable D6.1 Appendix-4.

In parallel with the guidelines for the ICS-TCS Integration, a dedicated RUC interview template level-1 was prepared (Deliverable D6.1 Appendix-5 to be used during the first site visits to the TCS WPs. The site visits were conducted during the time period between November 2015 and March 2016. All four steps are now completed, whereas step 5 with metadata implementation has started in January 2017 and is ongoing.

Since March 2017 work in WP6&7 was focused on two main tasks – development of the ICS system and implementation of metadata from TCS WPs. The fastest way of showing any data in the ICS system allowing some testing was foreseen through webservices. Top-priority DDSS elements (covering some of the webservices) were selected for the first round of implementation. The metadata model structure and its implementation via EPOS-DCAT-AP was agreed during the WP6&7 IT-team meeting in Bergen (9-11 May, 2017). The main tasks were to introduce the metadata model and implementation of metadata for first three entities (Person, Organization and Webservices) for 3-4 top-priority DDSS elements. The introduction of metadata implementation to TCS WPs was done via dedicated meetings and videoconferences in June-July 2017. Metadata implementation was facilitated via GitHub repository (https://github.com/epos-eu/EPOS-DCAT-AP) where one metadata example (XML file) was prepared and adapted by each TCS WP.
The ICS development succeeded in implementation of the core parts of the ICS architecture. The GUI prototype is searchable for 95 TCS webservices currently. Further ICS development and implementation of webservices (namely OGC (Open Geospatial Consortium) services) was facilitated during the WP6&7 IT-team meeting (with participation of WP15) in Keyworth, UK (11-15 December, 2017). The complete metadata schema representation as RDF including all metadata entities was ICS-internally introduced during video conference in the beginning of February 2018 and officially introduced to the EPOS IT community during the EPOS Implementation and Validation Workshop in Lisbon, Portugal (12-15 March, 2018).
The second round of metadata implementation started in August 2018 and should be finalized by mid-September 2018. The developed Web Metadata editor is making the metadata implementation more transparent for the TCS groups.

## 2.3 Summary of Requirements

The key requirements are as follows:

1. minimal interference with existing TCS operations and developments including IT;
2. easy-to-use user interface;
3. access to assets through a catalog: initially datasets but progressively also services, workflows, software modules; computational facilities, instruments/sensors all with associated organisational information including experts and service managers;
4. progressively assistance in composing workflows of services, software and data to deploy on e-Infrastructures to achieve research infrastructure user objectives.

Process of requirements and use cases collection resulted in a comprehensive set of information about the DDSS elements that are declared by the TCS WPs. These are summarized below through simple statistics (Figures 1-4). Implementation statuses at ICS level by M23 and M33 are shown in Figures 5-7.

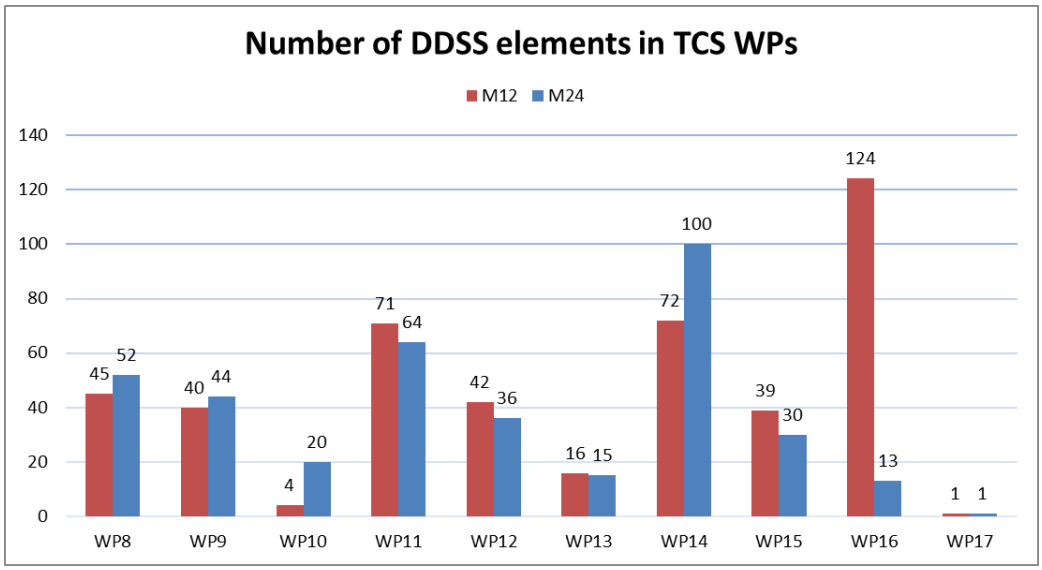



***Figure 1*** *Number of DDSS elements in TCS WPs in M24 compared to status from M12.*

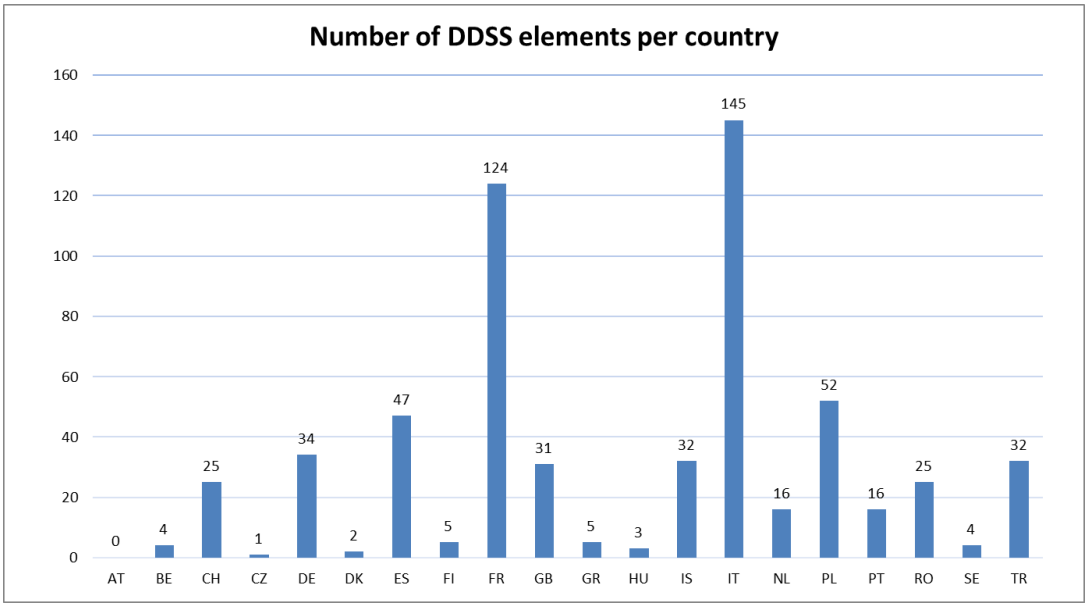



***Figure 2*** *Number of DDSS elements per country.*

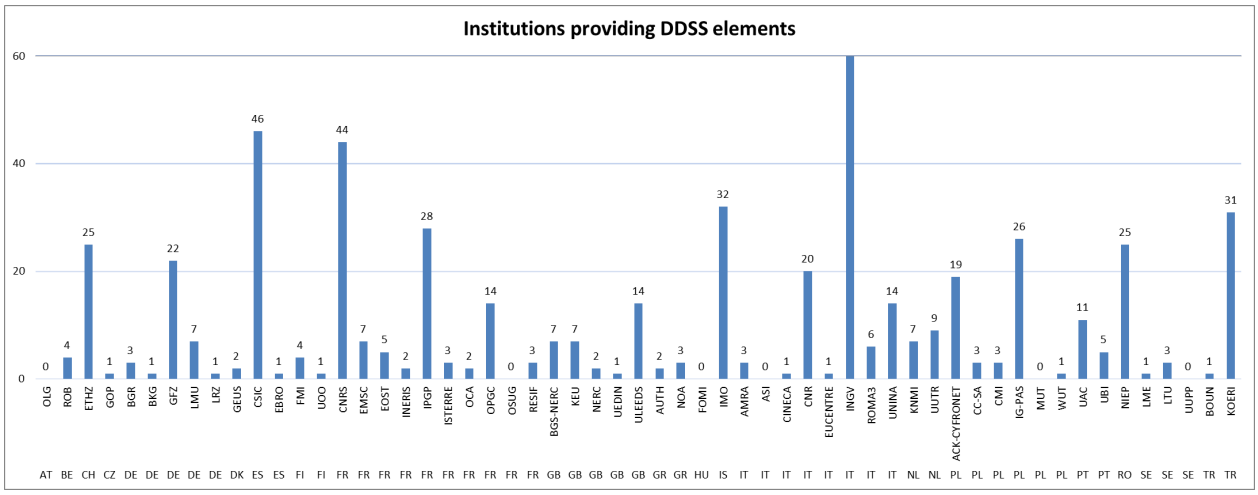



***Figure 3*** *Distribution of institutions providing DDSS per country.*

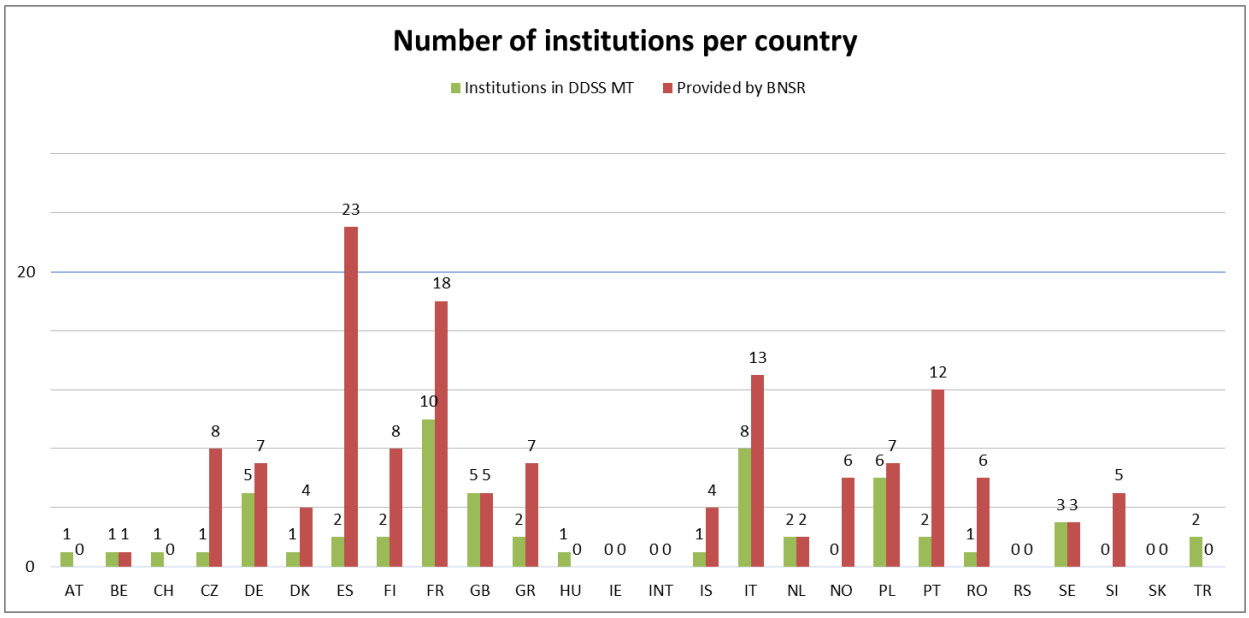



***Figure 4*** *Number of institutions contributing to individual DDSS elements as in DDSS Master Table which is not covering granularity in such details as information provided by Board of National Scientific Representatives (BNSR).*

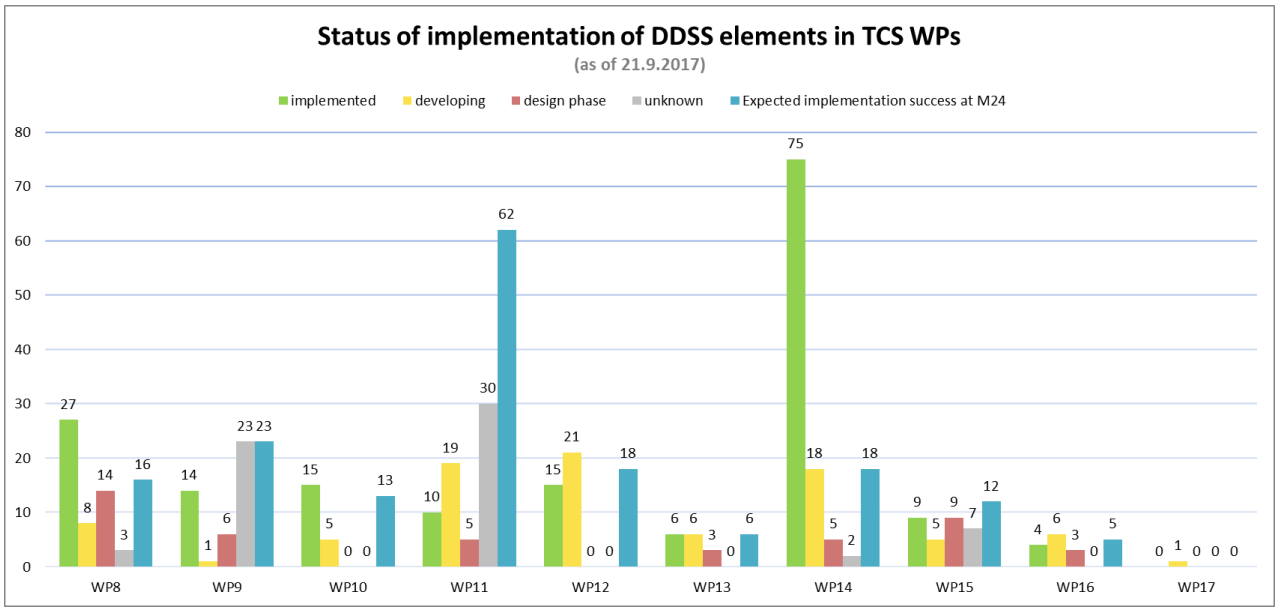



***Figure 5*** *Status of maturity of DDSS elements in TCS WPs before implementation to ICS at M23.*

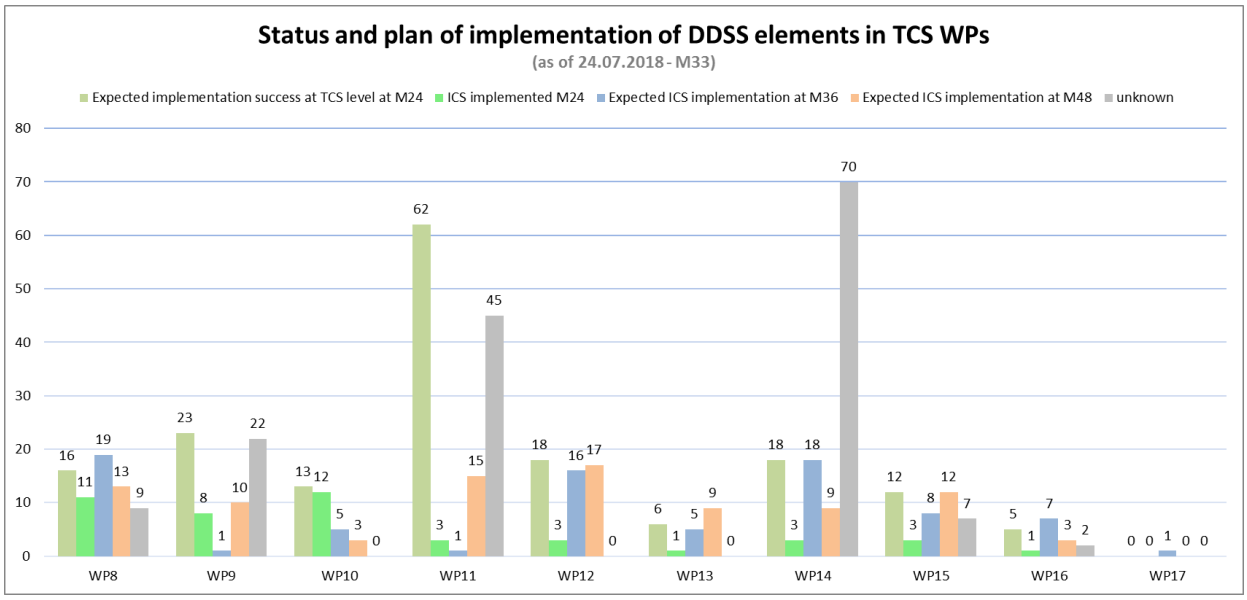



***Figure 6*** *Status and plan of implementation of DDSS elements in TCS WPs at M33.*

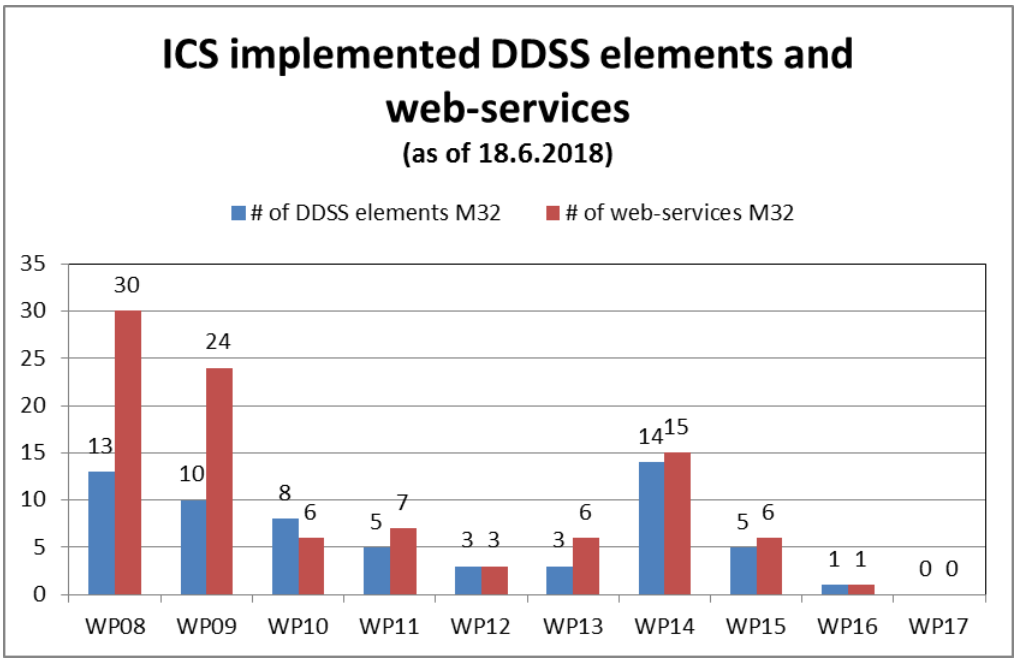



***Figure** 7 Number of implemented DDSS elements and web-services by M32.*

From the above it is clear that progress has been made. We have also cleared up confusion between DDSS elements and services: there is a n:m relationship between them: one service may use many DDSS elements and one DDSS element may be utilised in many services. The DDSS tables in Excel are being migrated to a GDB (Granularity Database) to represent these relationships and associated information.

# 3. State of the Art

## 3.1 e-Infrastructures

e-Is (e-Infrastructures) continue to provide a level of services common to – and used by - many RIs (and other research environments). The major e-Infrastructures of relevance to EPOS-IP are:

1. GEANT: the academic network in Europe which brings together the national research networks http://www.geant.org/
2. EGI: a foundation and organisation providing infrastructure computing and data facilities for research https://www.egi.eu/
3. EUDAT an EC-funded project to provide infrastructure services for datasets including curation, discovery https://eudat.eu/
4. PRACE: a network providing resources on supercomputers throughout Europe http://www.prace-ri.eu/
5. EOSC: the European Open Science Cloud which aims to provide infrastructure services for research with the first pilot project starting in January 2017 https://eoscpilot.eu/ and subsequently the EOSC-Hub which is soliciting services;
6. OpenAIRE: an EC-funded project to provide metadata to access research publications and – started recently – related datasets https://www.openaire.eu/

Participant organisations in EPOS have been involved to a greater or lesser extent in all of these activities. In particular EPOS TCSs (with support from the ICS team) have been conducting pilot projects with EGI, PRACE and EUDAT and EPOS is involved in the EOSC pilot.

## 3.2 Research Infrastructures

EPOS is one of the ~50 ESFRI roadmapped RIs (research infrastructures). The other RIs are developing similar facilities for their end-users as EPOS. Thus, there is potential to learn from them and even to cooperate in developments. RIs generally have their own computing and sensor/detector facilities within organisations cooperating in the RI. However, they may also utilise the e-Is – in particular GRID or CLOUD computing, EUDAT for data storage, OpenAIRE for open access provision of scholarly publications and – of course – rely on GEANT for networking.

The RIs with relevance to EPOS are:

1. ENVRIplus (related environmental science RIs, some have technology relevant to EPOS for example in sensors, asset catalogs, portal software, interoperation, metadata standards)
2. ELIXIR/EXCELERATE (techniques for metadata catalogs, ontologies and workflows)

## 3.3 Virtual Research Environments

Some RIs have developed user interfaces that claim to be VREs (Virtual research Environments). An example is BlueBridge which accesses a few RIs in oceanography relying on EGI to supply the underlying e-I in a monolithic 'silo' structure. VREs not only provide access to assets but also assist in workflow construction and deployment. They provide mechanisms for researcher intercommunication and research output production (such as scholarly papers). They provide assistance with researcher management and administration such as writing proposals, evaluating research, discovering teams and individuals as co-workers or competitors or viewing the 'landscape of research' of interest to the researcher or research manager.

VREs in Europe have some characteristics in common with SGs (Science Gateways) in North America and VLs (Virtual Laboratories) in Australia although neither cover as wide a range of facilities as VREs. Under the auspices of RDA, representatives of each of the three meet to discuss characterisation and interoperation within the VRE Interest Group. Within Europe a call for proposals yielded 6 funded VRE projects all now cloming to an end. EPOS participates in VRE4EIC: primarily with requirements and evaluation of the architecture and its usefulness but also contributing to the technical IT aspects. Furthermore, joint work with VRE4EIC has led to products for use in both projects: the metadata mapping tool 3M from FORTH (GR) and the AAAI (Authentication, Authorisation, Accounting Infrastructure) working with CWI (NL) and workflow management working with CNR.

There is an open question in the community over whether a VRE is (a) a user interface which accesses RIs (which in turn access e-Is), provides researcher communication, administrative and management support; (b) all of (a) plus all the assets of the RIs and capabilities of the e-Is – i.e. the complete system as seen by the end-user.

## 3.4 EPOS-ICS Positioning

The EPOS-ICS provides the entrypoint to the EPOS environment. It provides an inventory of, and access to, the assets of the TCSs. It also provides access to e-Is upon which (parts of) workflows are deployed (other parts may be deployed within the computing capabilities of RIs within EPOS). EPOS has been involved in projects with e-Is to gain joint understanding of the interfaces and capabilities ready for deployment from ICS-C. EPOS has also been involved in the VRE4EIC project (and cooperating with EVER-EST) to ensure convergent evolution of the EPOS ICS-C user interface and APIs for programmatic access with the developing VREs. EPOS partners are also participating in is the recently approved ENVRIFAIR project which will assist in building linkages between EPOS ICS-C and EOSC (***Figure 8***).

The linkage between ICS-C on the one hand and the e-Is and TCS local computing resources and assets on the other (ICS-D) will be constructed in the ICS-C and managed in the deployment phase. The workflow for the deployment (which may be a simple file download or a complex set of services including analytics and visualisation) will be generated within the ICS-C by interaction with the users. The workflow will be checked by the end-user before deployment. However, the detailed content/capability of the assets might not be known e.g. the dataset may not contain the relevant information despite its metadata description or the software may not execute as the user expects despite the metadata description. The execution of the deployment is monitored and execution information is returned to the end-user. The workflow may be deployed in one of two ways: (a) directly with no user interaction during execution of the deployment; (b) step-by-step with user interaction (so-called computational steering) between each step. Deployments

of type (a) will have better optimisation (for performance) and security but could possibly execute a workflow the components of which do not behave as the user expects. Deployments of type (b) lack optimisation but allow the user to stop the workflow deployment at any step, examine the results and – if not as expected – reorganise the workflow (by changing components) to meet more closely the requirement.

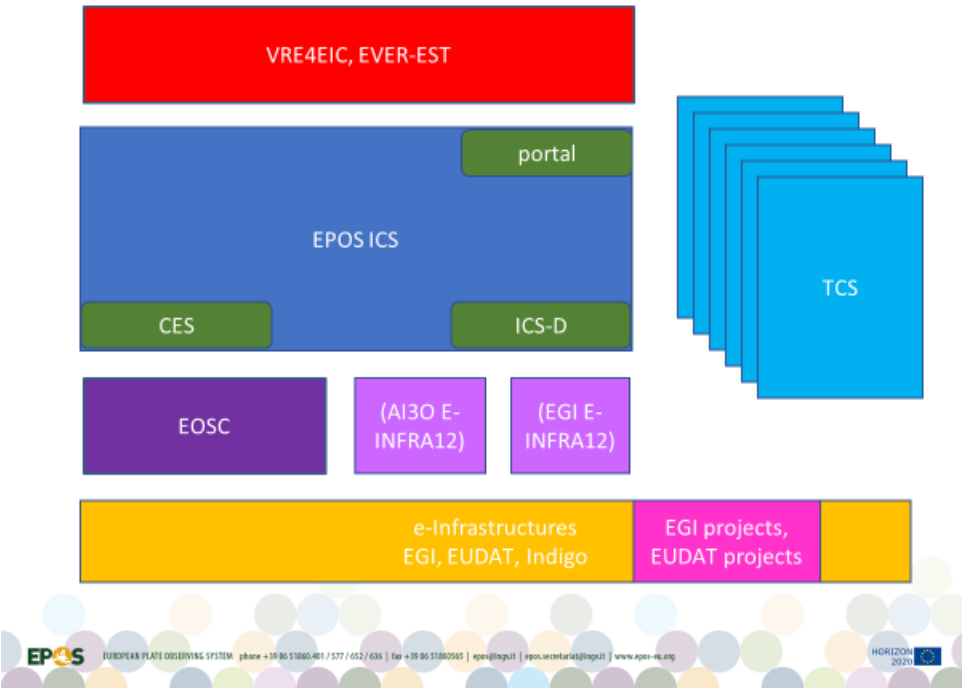



***Figure 8*** *The Positioning of ICS-C*

# 4. The Architecture

## 4.1 Introduction: ICS and TCS

The Integrated Core Services (ICS) represents the infrastructure consisting of services that will allow access to multidisciplinary resources provided by the TCS. These will include data and data products as well as synthetic data from simulations, processing, and visualization tools. The ICS consists of of the ICS-Central Hub (ICS-C) and distributed computational resources including also processing and visualisation services (ICS-D) of which a specialization is Computational Earth Science (CES).

The ICS-C consists of multiple logical areas of functionality, these include the Graphic User Interface (GUI), web-API, metadata catalogue, user management etc. A micro-service architecture has been adopted of the ICS-C, where each (micro) services is atomic and dedicated to a specific class of tasks. The ICS-C is where the integration of other services from ICS-D and TCS takes place. The architectural constraints for the ICS-D are elaborated as a metadata model within the ICS-C CERIF catalog and are being implemented.
The Thematic Core Services (TCS) enable the integration of data and services from specific scientific communities. The architecture of the services provided by the individual communities is not prescribed, what is required is that the metadata describing the data and services available is in a form that can be consumed by the ICS, allowing the ICS to integrate with those services and data (***Figure 9***).

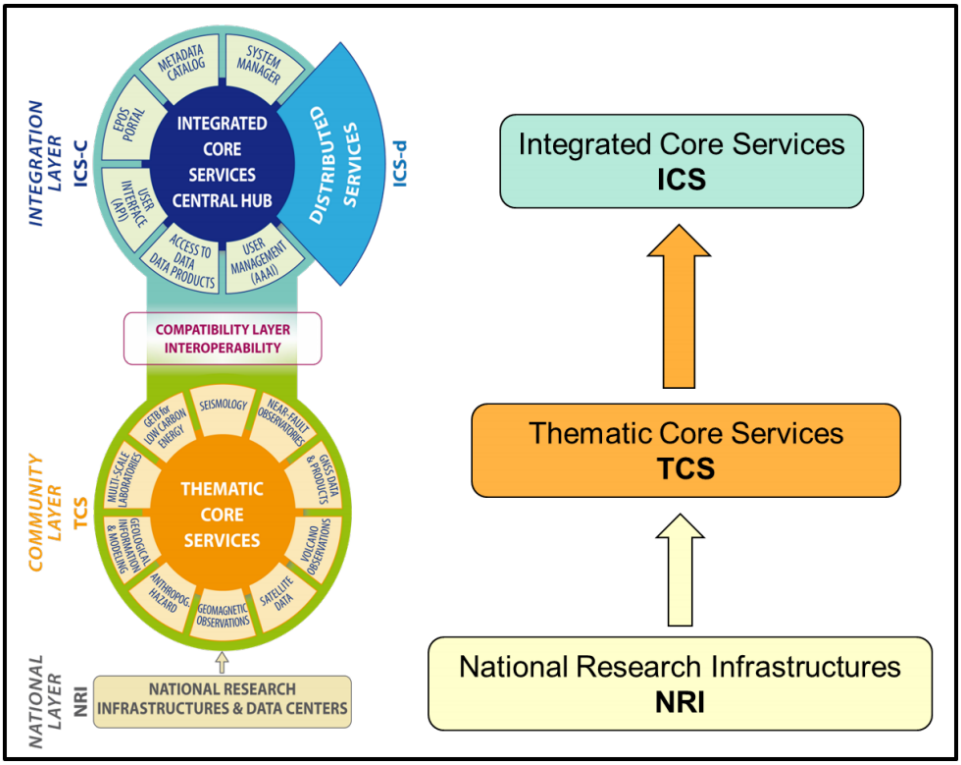



***Figure 9*** *The ICS-TCS Architecture*

## 4.2 The ICS System

The ICS-C System is the main system that manages the integration of DDSS from the communities. On top of such system, a Graphic User Interface (GUI) enable the user to search, discover, integrate data in a user-friendly way.
The EPOS ICS-C system architecture (***Figure 10Figure 9*** *The ICS-TCS Architecture*) was designed and developed with the aim of integrating data and services provided by communities through community specific data centers called Thematic Core Services (TCS). In order to a) enable the system to run in a distributed environment, b) guarantee up-to-date technological upgrades by adopting a software-independent approach, c) proper scaling of specific system functionalities, the chosen architecture followed a microservices paradigm.
The Microservices architecture approach envisages **small atomic services dedicated to the execution of a specific class of tasks**, which have high reliability[1] [2]. Such architecture replaces the monolith with a distributed system of lightweight, narrowly focused, independent services. In order to implement the microservices paradigm, Docker Containers technology was used. It enables complete isolation of independent software applications running in a shared environment. In particular, each microservice is developed in Java language and performs a simple task, as atomic as possible. The communication between microservices is done via messages received and sent on a queueing system, in this case RabbitMQ. As a result, a chain of microservices processes the requests.

[1] Newman, “Sam. Building Microservices”, O'Reilly Media, Inc., 2015
[2] International Journal of Open Information Technologies ISSN: 2307- 8162

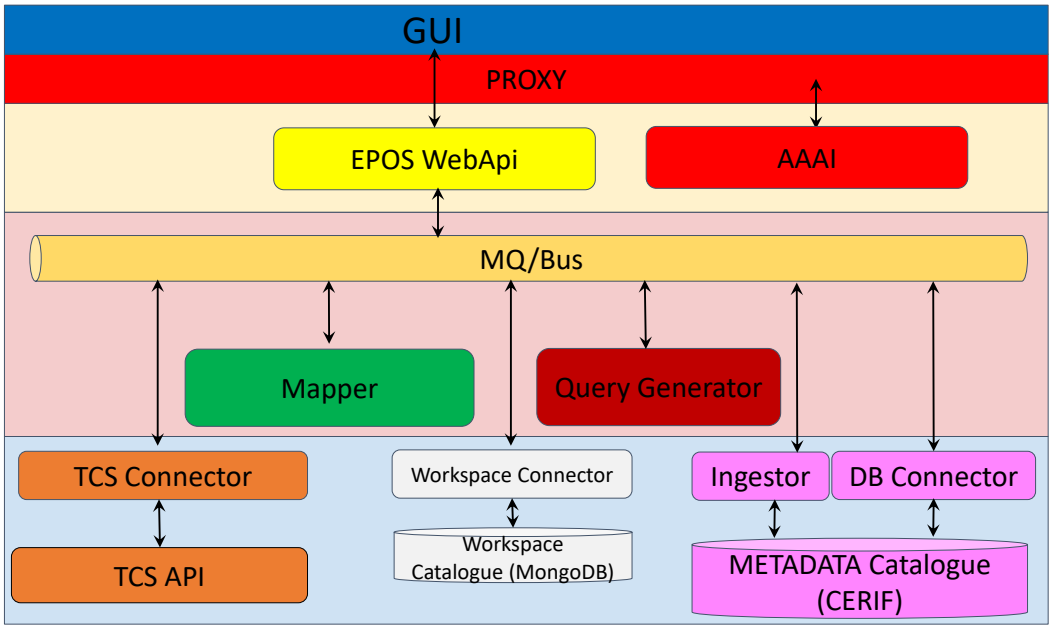



***Figure 10*** *Functional diagram of the system, its Components and Microservices*

A description of components and microservices will be provided in the following sections.

## 4.3 The Catalog

The metadata catalogue, is the key technology that enables the system to manage and orchestrate all resources required to satisfy a user request. By using metadata, the ICS-C can discover data or other digital objects requested by a user, contextualise them (for relevance and quality) access them, send them to a processing facility (or move the code to facility holding the data) depending on the constructed workflow, and perform other tasks. The catalogue contains: (i) technical specification to enable autonomic ICS access to TCS discovery and access services, (ii) metadata associated with the digital object with direct link to it, (iii) information about users, resources, software, and services other than data services (e.g. rock mechanics, geochemical analysis, visualization, processing). The data model used for the catalogue is CERIF (Common European Research Information Format).[3]
Metadata describing the TCS DDSS are stored using the CERIF data model which differs from most metadata standards in that it (1) separates base entities from linking entities thus providing a fully connected graph structure; (2) using the same syntax, stores the semantics associated with values of attributes both for base entities and (for role of the relationship) for linking entities, which also store the temporal duration of the validity of the linkage. This provides great power and flexibility. CERIF also (as a superset) can interoperate with widely adopted metadata formats such as DC (Dublin Core), DCAT (Data Catalogue Vocabulary), CKAN (Comprehensive Knowledge Archive Framework), INSPIRE (the EC version of ISO 19115 for geospatial data) and others using convertors developed as required to meet the metadata mappings achieved between each of the above standards and CERIF The metadata catalogue also manages the semantics, in order to provide the meaning of the attribute values.

The use of CERIF provides automatically:

(a) The ability for discovery, contextualization and (re-)use of assets according to the FAIR principles[4];
(b) A clear separation of base entities (things) from link entities (relationships);
(c) Formal syntax and declared semantics;
(d) A semantic layer also with the base/link structure allowing crosswalks between semantic terminology spaces;
(e) Conversion to/from other common metadata formats;
(f) Built-in provenance information because of the timestamped role-based links;
(g) Curation facilities because of being able to manage versions, replicates and partitions of digital objects using the base/link structure;

[3] CERIF is maintained by euroCRIS http://www.eurocris.org/cerif/main-features-cerif
[4] https://www.force11.org/group/fairgroup/fairprinciples

The catalog is constantly evolving with the additional of new assets (such as services, datasets) but also increasingly rich metadata as the TCSs improve their metadata collection to enable more autonomic processing.

# 5. Implementation

## 5.1 Populating the Catalog

The process of populating the catalog is crucial in the EPOS vision. Indeed, populating the catalog means to make available all the information needed by and end user to perform queries, data integration, visualisation and other functionalities provided by the system.

In order to manage all the information needed to satisfy user requests, all metadata describing the Thematic Core Services (TCS) Data, Datasets, Software and Services (DDSS) will be stored into the EPOS ICS, internal catalog. Such a catalog, based on the CERIF model, differs from most metadata standards used by various scientific communities in that it is much richer in syntax (structure) and semantics (meaning).
For this reason, EPOS ICS has sought to communicate to the TCSs the core elements of metadata required to facilitate the ICS through the EPOS Metadata Baseline. This baseline can be considered as an intermediate layer that facilitates the conversion from the community metadata standards such as ISO19115/19, DCAT, Dublin Core, INSPIRE etc. describing the DDSS elements and not the index or detailed scientific data (***Figure 11***).

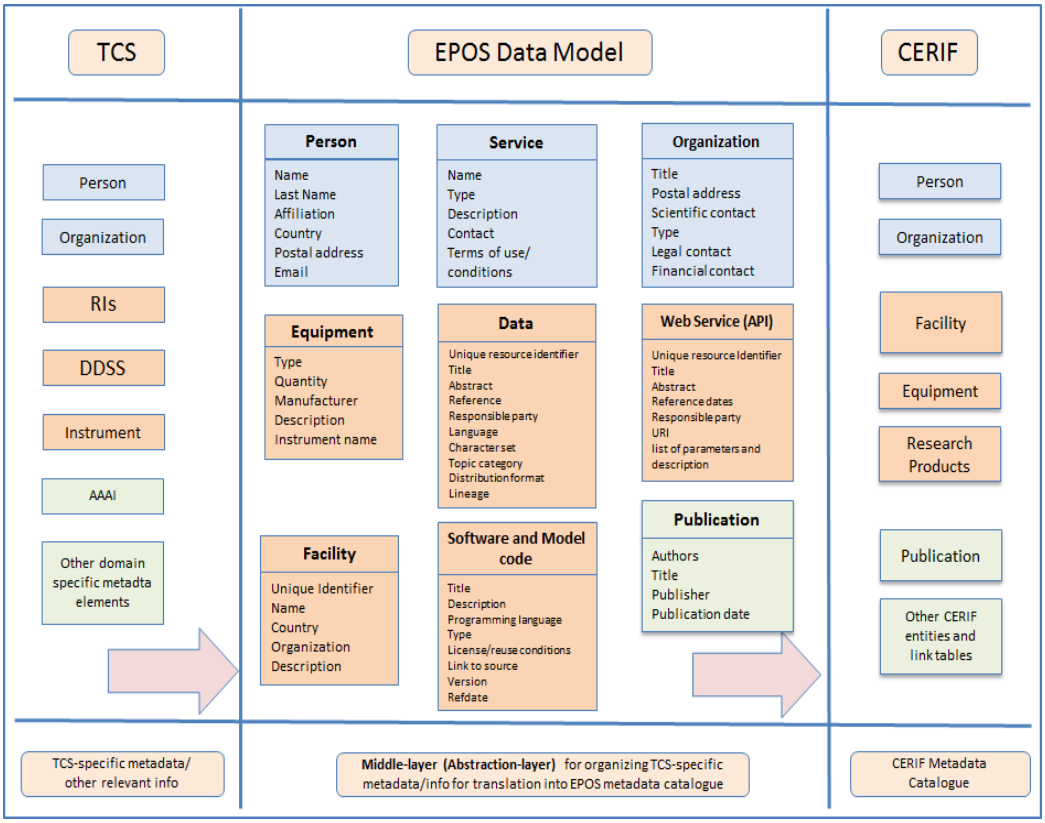



***Figure 11*** *Snapshot of the EPOS baseline concept*

The EPOS baseline, presents a minimum set of common metadata elements required to operate the ICS taking into consideration the heterogeneity of the many TCSs involved in EPOS. It has been implemented as an application profile using an extension of the DCAT standard, namely the EPOS-DCAT-AP. It is possible to extend this baseline to accommodate extra metadata elements where it is deemed that those metadata elements are critical in describing and

delivering the data services for any given community. Indeed, this has happened M24-M36 when the original DCAT-AP was found to be inadequate and a new version with richer metadata was designed and implemented.

The metadata to be obtained from the EPOS TCS as described in the baseline document (and any other agreed elements) will be mapped to the EPOS ICS CERIF catalog. The process of converting metadata acquired from the EPOS TCS to CERIF will be done by EPOS WP07 &06 but in consultation with each TCS as to what metadata they have available and harvesting mechanisms.
The various TCS nodes have APIs or other mechanisms to expose the metadata describing the available DDSS in a TCS specific metadata standard that contains the elements outlined in the EPOS baseline documents better described in the following sections. It also requires ICS APIs (wrappers) to map and store this in the ICS metadata catalogue, CERIF. These TCS APIs and the corresponding ICS convertors collectively form the "interoperability layer" in EPOS, which is the link between the TCSs and the ICS.

## 5.2 Building the ICS

The development of the ICS-C was carried out by multiple teams / working groups each concentrating on a specific aspect of the system.

### *5.2.1 User Interface*

The user interface will be developed as a single page web application (SPA) that is served statically but contains dynamic content (AJAX) that runs in the user's browser – thereby reducing the load on the server(s) serving the user interface.
The user interface's primary connection to the core ICS-C system will be via the Web API (below), which will expose a rest-like API (over HTTP) for locating metadata held within the catalogue, accessing data sources described in the metadata, and manage workspaces created by the user to organise the catalogue of interest. The user interface will not only offer the user the ability to perform a "faceted" search for available data, but also offer search categories for example to search for organisational entities such as organisations, people, etc.
An important aspect of the user interface is the visualisation of data, the primary use case is be able to visualise the geographic extents of the data represented by the metadata entries in the catalogue. The user interface will need to support the visualisation of a subset of supported data formats (spatial, temporal, binary-downloads); it will be impractical for it support all data types used by all scientific communities.
As mentioned above, the user can create workspaces, in which they can group metadata entries (typically those represent data-sources) of interest – the long-term persistence of these workspace is handled by the Web API (below). Workspaces give the user the ability to carry out further work with the item of interest, or refine their configuration (where applicable), all of the visualisation available to the "discovery" pages of the user interface will also be available for the contents of the workspaces.
Workspaces provide the access point to working with distributed process environment – usually accessed and managed through a workflow (e.g. Taverna, Kepler) or notebook (e.g. Jupyter) type system. It is planned that a processing environment can be spawned from a workspace. A processing environment is expected to consist of a Docker container running remotely, pre-installed with a relevant technology stack (such as Jupyter or a workflow engine), and "seeded" with access to the data source include in the originating workspace. A running processing environment will serve its own user interface that will be included in an appropriate page/location in the EPOS user interface.

The following diagram (***Figure 12***) illustrates the system boundary of the User Interface in the context of the ICS-C.

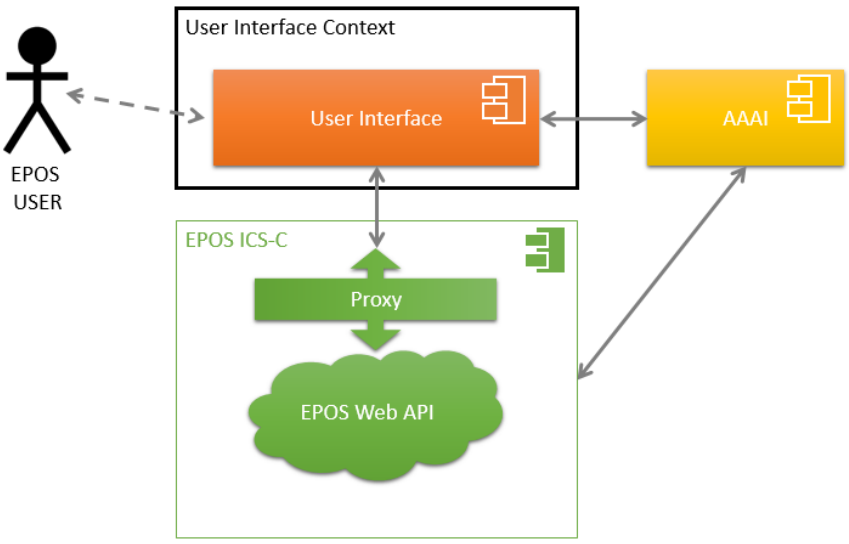



***Figure 12*** *the system boundary of the User Interface in the context of the ICS-C.*

The overall EPOS system is comprised of a number of components and artefacts, as can be seen in earlier sections, the remainder of this section will detail the architecturally significant elements of the User Interface.

## Constraints

Any component of a system is developed within constraints; usually the interfaces to other components or constraints associated with non-functional requirements (performance, security…). These constraints shape the architectural decisions which in turn provide the context for the development of any component. In the case of the GUI, the primary constraint on the functionality of the User Interface is the integration with ICS-C via the EPOS web API, the implication being that user interface will only be able to offer functionality that (when required) has adequate support from the EPOS web API.
A secondary constraint is that the authentication part of AAAI, is offered by a 3rd party, hence the act of authentication and what that facilitates within the wider EPOS system is largely outside the control of the user interface.

## Quality Attributes

The following quality attributes have been identified as architecturally significant for the user interface.

**Usability** usability is the dominating quality that user interface will be judged on.

**Performance** performance is also essential and is one of the driving reasons why a stateless SPA architecture has been adopted.

**Security** security, in particular authentication and authorisation in terms of access, from the user interface, to the resources exposed by the API is of particular importance to the owners of assets exposed via EPOS.

**Adaptability** adaptability of chosen technologies and designs will be crucial for such a project such as EPOS where requirements are fluid and evolve throughout development.

**Maintainability** the components (and the source-code) delivered during development must take into consideration maintainability, as the teams responsible for maintaining the EPOS system may not be comprised of the original development teams.

## Assumptions

Primary requirements of the user interface revolve around the ability of "visualise" the response of web-services; a significant assumption on the behalf of the user interface is that the EPOS web API will be able to reliably and

unambiguously identify (ahead of time) to the user interface what the expected output of a web-service will by. Thereby allowing the user interface to make informed judgments about the type(s) of visualisation that can be offered.

## Risks

Developmental risks exist in the following forms:

- The constant evolution of requirements can invalidate design decisions if care is not taken to design the software in way that allows for flexible refactoring. Designing a flexible solution is often more time consuming in its own right, but having said that it is highly unlikely that a development team will be able to anticipate all possible scenarios. In short, constant changes in product direction will affect what can be achieved in terms of: time, quality & scope.
- The user interface has a fundamental dependency on the EPOS web API, which is also in active development, hence the user interface is subject to any of the risks that EPOS web API is itself exposed to. During development this has to some degree been mitigated by the creation of a mock API with the same expected "interface" as the real API, however once in the testing and productions phases the user interface will be inherently reliant on the EPOS web API.
- The EPOS system as a whole has a dependency on the presence of sufficient and appropriate metadata, with which the core system, API, and user interface will be driven and populated. Insufficient metadata won't necessarily affect the architecture of any of the system components, but it will affect the potential of testing and demonstrating the system as whole.

The recognition and analysis of these risks, and their management by contingency/amelioration measures is an important aspect of developing the pre-production system.

## Technical Infrastructure

**Development Practices:**

- Agile / Scrum — project methodology
- CI/CD — Continuous Integration / Continuous Deployment (dev-ops)
- Peer review — all code merged into production branches is peer reviewed

**Development Infrastructure:**

- GitLab — version control software, automated pipelines for build/test/deploy
- Node & NPM — to compile and manage dependencies
- Docker — to run GitLab CI pipelines, host UI server

**Technologies & Frameworks:**

- Web Application
  - HTML
  - JavaScript & Type Script
  - CSS & SASS
  - Angular
  - Angular component libraries
- Visualisation
  - Leaflet (visualisation of, supported, map based data)
  - Charting (to be confirmed)

**Deployment Infrastructure:**

- GitLab — automated pipelines for build/test/deploy
- Docker — to run GitLab CI pipelines, host UI web-server
- Kubernetes — to host and orchestrate deployed Docker containers
- Nginx — web-server , for serving interface pages

## Logical Overview

The diagram below (***Figure 13***) gives an overview of the design of the user interfaces and how it can be broken down in to logical units; logical units may comprise of: components, files, APIs between units.

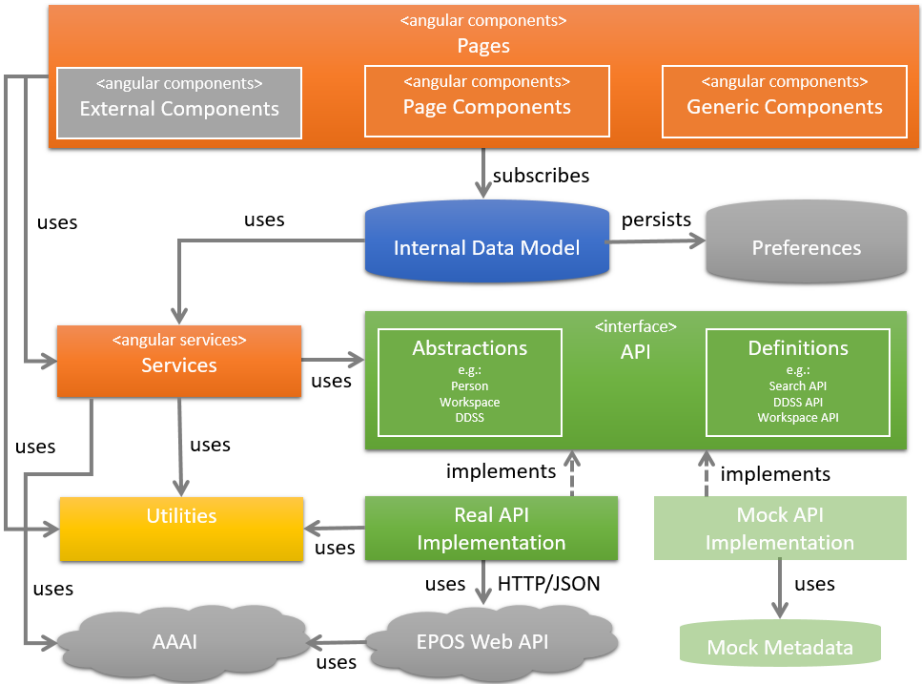



***Figure 13*** *Graphical User Interface*

## Angular Components

Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data. Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient. Every Angular application has at least one component, the root component that connects a component hierarchy with the page DOM – in the case of the EPOS user interface it is the `app.component`. Within Angular a page is a component that will act as an entire view, and may itself contain nested components.

Below is an inexhaustive sample of the core angular components created for the EPOS user interface:

- app.component — main application component
- landing.component — landing page component
- temporal.component — temporal workspace (charting) page component
- spatial.component — spatial workspace (map) page component
- processing.component — processing workspace (workflow) page component
- rightbar.component — right hand pop out bar component, used during data discovery
- sidenav.component — navigation bar component, used on majority of pages
- table.component — reusable table components
- workspace.component — workspace contents page component
- login.component — reusable login component
- header.component — reusable header component, used on all pages
- footer.component — reusable footer component, used on all pages

## Services

For data or logic that is not associated with a specific component, that is to be shared across components, the user interface uses services. Services are made available to components via dependency injection.

Below is an inexhaustive sample of the core angular services created for the EPOS user interface:

- api.service — service for accessing the API abstraction (see below)
- model.service — service for accessing the internal data model (see below)
- aaai.service — service for accessing third party authentication providers
- search.service — service that encapsulates the process of carrying out a "search"
- execution.service — service that encapsulates the process of executing a web-service
- workspace.service — service for interacting and managing user workspaces

### Internal Data Model

The EPOS user interface, as noted before is stateless single-page-application, it does however maintain its own internal state for the duration whilst it is active with in the users browser. This internal state model allows otherwise unrelated angular components to publish and subscribe to data events.
For example if the data model is updated with the facets selected by the user, another component can subscribe to the relevant event(s) to be notified, and in turn update the model with the list of results to display, which in turn is listened to by the component that renders the results in the browser.
In addition the data model can be used for storing transient preferences such as whether a side bar is expanded or not. These transient preferences are not essential for the user interface to function, but may offer the user an enhanced user experience – therefore these may in the future be considered for short term / volatile persistence via an API.

### API

The user interface has a collection of classes and interfaces that act as an abstraction of the EPOS web API, enabling the remainder of the user interface code to be relatively insulated from minor changes in the web API, however large changes to the web API will result in refactoring of the API abstractions.
The full breadth of functionality offered by the web API is broken down into smaller interfaces (interface - segregation) targeted at a specific areas of functionality, allowing a full API implementation to be built up (via composition) from individual classes – this allows for the easy replacement of parts of the API with either mock functionality or real functionality depending on the availability and progress of the web API.

There are two full implementation of the API within the user interface codebase:

- Development — implemented using calls to the real EPOS web API.
- Mock — implemented using mock date held as assets within the user interface.

The API component also includes abstractions for the datatypes returned by the EPOS web API, for example: DDSS, Facet, Workspace, etc. This helps ensure type safety throughout the user interface code, reducing the potential for bugs that ensue if arbitrary blobs of JSON were passed around.
The user interfaces' API code communicate with EPOS web API over HTTP in a rest –like approach, with responses being returned as JSON structures.

### Utilities

The utilities component represents the collection of code that offers useful functionality to multiple and disparate parts of the code-base. It is also a catch all for common or global configuration such as CSS styling for the entire user interface.

## *5.2.2 Web API*

The Web APIs are developed as the primary entry point to the core ICS system for client systems (including EPOS GUI described above), they are implemented as a restful-like APIs that will allow consumers to search for metadata, and return that metadata in the as JSON-LD in a form that corresponds to that of the proposed "metadata baseline"
As new requirements are identified, either the Web APIs will be enhanced or additional APIs will be created. For example user management and workspace management.

**Development Practices:**

- Agile / Scrum
- Continuous Integration / Continuous Deployment (CI/CD)

**Development Infrastructure:**

- GitLab
- Docker for hosting the WebApi component.

**Technologies & Frameworks:**

- Java
- Swagger.io – API building / testing framework.
- Framework: Spring
- Java Library: Gson

A summary description of the current WEB APIs endpoint is represented in the following (***Figure 14***). Endpoints are available at http://epos.cineca.it/webapi/v0/swagger-ui.html#.

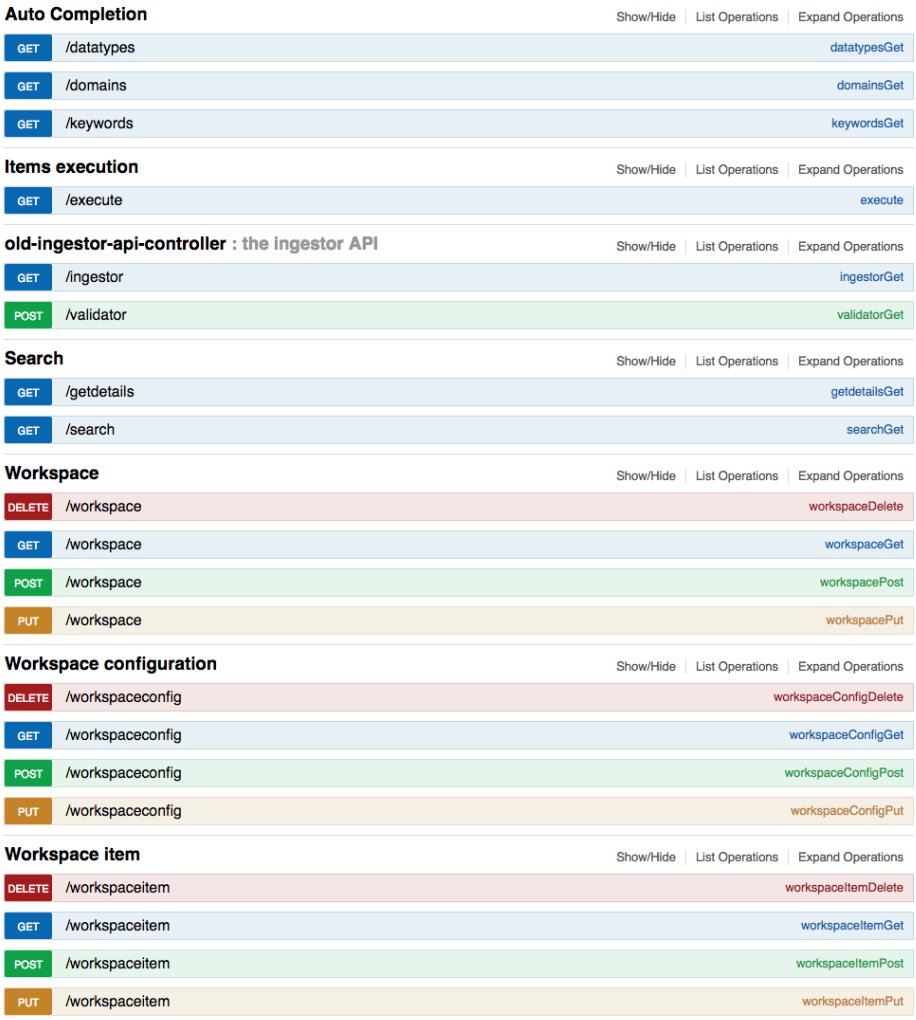



***Figure 14*** *Current WebAPI Endpoints*

## *5.2.3 Core System*

As prescribed in the microservice framework, different components (services) are attached to one single 'merger' node, or bus. In our case the bus is a queueing system that enable the different components to send and receive messages.

## Internal Queues

The main idea behind Work Queues (aka: Task Queues) is to avoid doing a resource-intensive task immediately and having to wait for it to complete. Instead we schedule the task to be done later. We encapsulate a task as a message and send it to a queue. A worker process running in the background will pop the tasks and eventually execute the job. When you run many workers the tasks will be shared between them.
This concept is especially useful in web applications where it's impossible to handle a complex task during a short HTTP request window[5].
Within the EPOS system, the name of the queue is defined in this way:

*"To" + <component-name>*

Each component has two queues, a queue for input and a queue for output, a consumer associated with the input queue, and a producer associated with the output queue.
The input queue is a queue fixed at the time the component is started, instead the output queue is a queue that is defined by the logic of each individual component depending on the operation to be performed
The components and related input queues are shown below.

| *Component Name (Actual Name)* | *Input queue name* |
|---|---|
| WebApi | ToWebApi |
| QueryGenerator (QueryGen) | ToQueryGen |
| DBConnector | ToDBConnector |
| Ingestor | ToIngestor |
| Mapper | ToMapper |
| Workspace Connector (Workspace) | ToWorkspace |
| TCSConnector | ToTCSConnector |
| Validator | ToValidator |

## Framework

MQManager is a simple framework that allows in a few steps to implement the connection to the queue and to maintain a standard in the structure of the component. The latest version is 1.4.2.
Currently it is possible to find it in the following repository, but it will be moved to an official repository:
https://github.com/vvalerio/MQManager

This framework defines:

- the structure of the JSONMessage (which we will discuss in the next chapter "Message Exchange")
- the ID structure of a message

[5] See for instance http://www.rabbitmq.com/tutorials/tutorial-two-java.html

- a standard "EPOS Logger" logger to which an HTML formatter is associated to facilitate reading through a web page
- the connection to the queue
- the declaration of connection channels
- the declaration of queues

The framework in the next version (1.5) will also define:

- An interface for creating messages
- Beans for each JSONMessage payload of component for input and output
- Parser of the JSONmessage payload

### Message Exchange: JsonMessage

JsonMessage is the message sent among the components of a RabbitMQ net.
This is the actual format:

```
{
    "Header":  {
        "version":1,
         "id":"<creation-timestamp>_<random-number>",
         "serviceType":"<serviceType>",
         "type":"<type>",
         "updatedOn":"<message-update-timestamp>",
         "responseNode":"<responseComponent>"
    },
    "Payload":  {  // < dinamic - content >
    }
}
```

Header parameters description:

- **VERSION:** the components version to be used (defined through the WebAPI endpoints)
- **ID:** message identifier, used by RPC message exchange of RabbitMQ.
- **TYPE:** the type of service requested from the WebApi. It's used to define the *initial Queue***UPDATE ON:** timestamp of last update of the JsonMessage
- **RESPONSE NODE:** queue of the next component
- **SERVICE TYPE:** the type of "sub-service" requested from the WebApi, is used for select the right queue of RabbitMQ.

The following snippet shows the interaction between *type* and *initial Queue.*

```
private static HashMap < String, String > routingMap = new HashMap < String, String > ();
static {
    routingMap.put("ingestor", "ToIngestor");
    routingMap.put("search", "ToQueryGen");
    routingMap.put("keywords", "ToQueryGen");
    routingMap.put("domains", "ToQueryGen");
    routingMap.put("workspace", "ToWorkspace");
    routingMap.put("workspaceconfig", "ToQueryGen");
    routingMap.put("execute", "ToWorkspace");
    routingMap.put("validator", "ToValidator");
}
```

The following table shows the relationship between serviceType, RabbitMQ initial queue and following queues, and Service requested

| Service Type | RabbitMQ Queues (in bold the initial queue) | Service/s Requested |
|---|---|---|
| Keywords | **ToQueryGen**,ToDBConnector,ToMapper | Metadata Service |
| Domains | **ToQueryGen**,ToDBConnector,ToMapper | Metadata Service |
| Datatypes | **ToQueryGen**,ToDBConnector,ToMapper | Metadata Service |
| Search | **ToQueryGen**,ToDBConnector,ToMapper | Metadata Service |
| getDetails | **ToQueryGen**,ToDBConnector,ToMapper | Metadata Service |
| wsPOST | **ToWorkspace** | Workspace Service |
| wsPUT | **ToWorkspace** | Workspace Service |
| wsGET | **ToWorkspace**,ToMapper | Workspace Service |
| wsDELETE | **ToWorkspace** | Workspace Service |
| wsitemPOST | **ToWorkspace** | Workspace Service |
| wsitemPUT | **ToWorkspace** | Workspace Service |
| wsitemGET | **ToWorkspace**,ToMapper | Workspace Service |
| wsitemDELETE | **ToWorkspace** | Workspace Service |
| wsconfPOST | **ToWorkspace** | Workspace Service |
| wsconfPUT | **ToWorkspace** | Workspace Service |
| wsconfGET | **ToWorkspace**,ToMapper | Workspace Service |
| wsconfDELETE | **ToWorkspace** | Workspace Service |
| wsconfVisualize | **ToWorkspace**,ToTCSConnector,ToMapper | Workspace Service, Execution Service |
| wsconfDownload | **ToWorkspace**,ToTCSConnector | Workspace Service, Execution Service |
| ingestor | **ToIngestor** | Ingestor Service |
| validator | **ToValidator** | Validator Service |

### Message Exchange: Response Message

The Response Message is the message sent back to WebAPI.
This is the actual format:

```
{
    "self": "<self url>",
    "message": "<message>",
    "response": "<response-type>",
    "data": "<result>"
}
```

**SELF:** the self-URL

**MESSAGE:** a human-readable message which describes the result of the operation

**RESPONSE:** a http-like response (*SUCCESS, BAD_REQUEST, INVALID_FORMAT, etc.*)

**DATA:** the result of the request

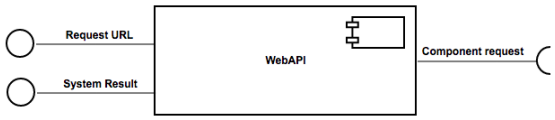
## *5.2.4 System Components*

The following components elaborate and explain the components of the Architecture diagram in(***Figure 10***) In order to define what a component is, we borrow Martin Fowler's definition[6].

> *"A component is a unit of software that is independently replaceable and upgradeable."*

Each component performs an atomic operation or a minimal set of operations. A short description of the components (Fig. 10) follows:

### WebAPI




#### Description

It represents the entry point to access the EPOS system and provides different RESTful services. It receives the REST request from GUI, creates the JSON message that contains the request parameters and sends it to the right component. It waits for a response and sends it to GUI component.

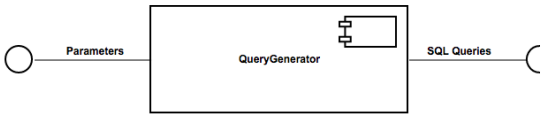
#### Input

A request from the endpoints

#### Output

A JSONMessage which payload is the representation of the input request

### QueryGenerator




#### Description

It receives a JSONMessage, generates SQL queries from the information inside the JSONMessage payload, updates the payload with the queries.

#### Input

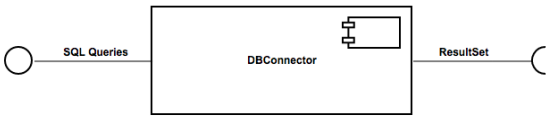
A JSONMessage which payload contains set of information to be able to generate a set of queries

#### Output

A JSONMessage which payload contains a set of queries

[6] [https://www.martinfowler.com/articles/microservices.html]

### DB Connector




#### Description

It receives a JSONMessage which contains a set of queries, execute them and updates the JSONMessage with the result set.

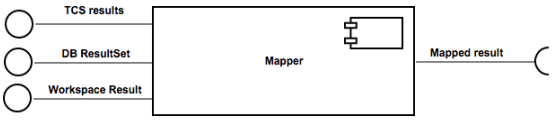
#### Input

A JSONMessage which payload contains a set of queries

#### Output

A JSONMessage which payload contains the result of database queries (CERIF)

### Mapper




#### Description

It receives a JSONMessage that contains information that can be mapped from a generic format to another format (currently GeoJSON, "GUI-JSON", in the future other formats)

#### Input

A JSONMessage which payload contains some mappable information

#### Output

A JSONMessage which payload contains the mapped result

### Workspace Connector

Description

Receives a JSONMessage that contains information to get, create, update, remove a workspace, a workspace object, or an object configuration.
The component is connected to a NoSQL database in MongoDB, which stores all the information concerning the workspaces

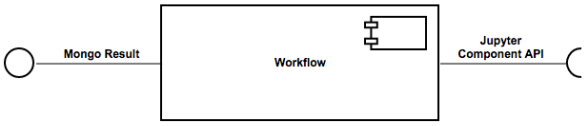
Input

A JSONMessage which payload contains information about workspace, workspace item, workspace item configurations

Output

A JSONMessage which payload contains a JSON which describes a workspace, a workspace item or a workspace item configuration

**Workflow**




Description

It creates a request to ICS-D to spawn a docker container with jupyter notebook and other tools to execute scientific workflows

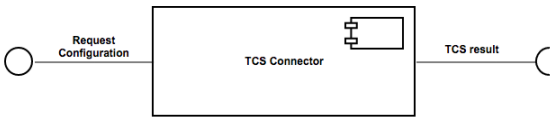
Input

A JSONMessage which payload contains information about a workspace

Output

A JSONMessage which payload contains the URL to retrieve the spawned container

**TCS Connector**




Description

It receives the JSON message with the configuration which contains information to build the request for the TCS API, executes the request and update the JSONMessage payload with the result.

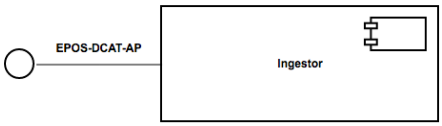
### Input

A JSONMessage which payload contains information about the workspace item configuration

### Output

A JSONMessage which payload contains the result of the request on the TCS API

## Ingestor (RDF Ingestor)




### Description

It receives the EPOS-DCAT-AP xml/turtle file, parse it and call the stored procedures in the CERIF database. Then update the materialized views on the database.

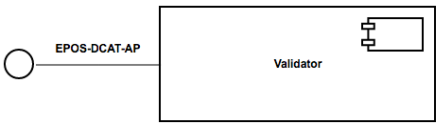
### Input

A single or a set of EPOS-DCAT-AP xml/turtle files

### Output

A JSONMessage which notify the result of the operation

## Validator




### Description

It receives an EPOS-DCAT-AP turtle string, parse it and validate it.

### Input

A JSONMessage which payload contains an EPOS-DCAT-AP turtle string

### Output

A JSONMessage which notify the result of the operation

### Metadata Catalogue

An internal metadata catalogue is required by the ICS to store, in the CERIF model, details of the data and services provided by the TCS and ICD-D.

A "metadata baseline" has been established and updated, this provides a profile of key metadata elements in the minimum which would be needed to meet the requirements of the ICS, taking into consideration the heterogeneity of the data and services provided by the TCS. This metadata baseline allows us to select key metadata elements from the metadata provided by the TCS's. The baseline will expand over time as we identify other key metadata elements from the communities not currently provided in the baseline and key elements required to offer the more complex assets from the TCSs to end-users (e.g. workflow or TNA (Trans-National Access)).

A mapping framework maps elements of the baseline to the CERIF Model. This mapping enabled the creation of appropriate convertors to convert metadata from the TCS to populate the CERIF model.

A series of database views have been created to aggregate and simply the CERIF Model as a middle layer in the database to enable the ICS Web API to retrieve data from the metadata catalogue as per the baseline concepts.

Technologies & Frameworks:

- SQL
- Postgresql – open source Object-Relational DBMS

### Technologies & Frameworks

The following technologies have been used to develop the ICS-C system

- Java
- Docker for hosting the individual (micro) services that make up the core system.
- Framework: Spring
- Queue Technology: RabbitMQ
- Java Library: Gson
- Java Library: Jsonld-java
- Java Library: apache jena

## 5.3 The Pre-Production Prototype

The notion of a prototype is the development of a product to evaluate a concept, but isn't intended for final release. The intention is to demonstrate progressively improved development versions (beyond a prototype) of the ICS-C. This has happened M19-M36 with demonstrations at EPOS meetings and at meetings with specific TCSs.

### *5.3.1 Demonstrator*

The goal of the demonstrator is to illustrate to stakeholders how user stories / use cases and requirements are going be satisfied.
The requirements, user stories and use cases drove the development of the ICS-C. The demonstrator includes as much real functionality as is possible at each stage leading to a complete pre-production version for handover to the EPOS production service team 20180930.

Functionality:

- User Interface with the ability to call the Web API and display search results.

- Web API with a working subset of query types.
- Core system forwarding requests from the Web API to the Metadata Catalogue.
- Metadata Catalogue with pertinent data.

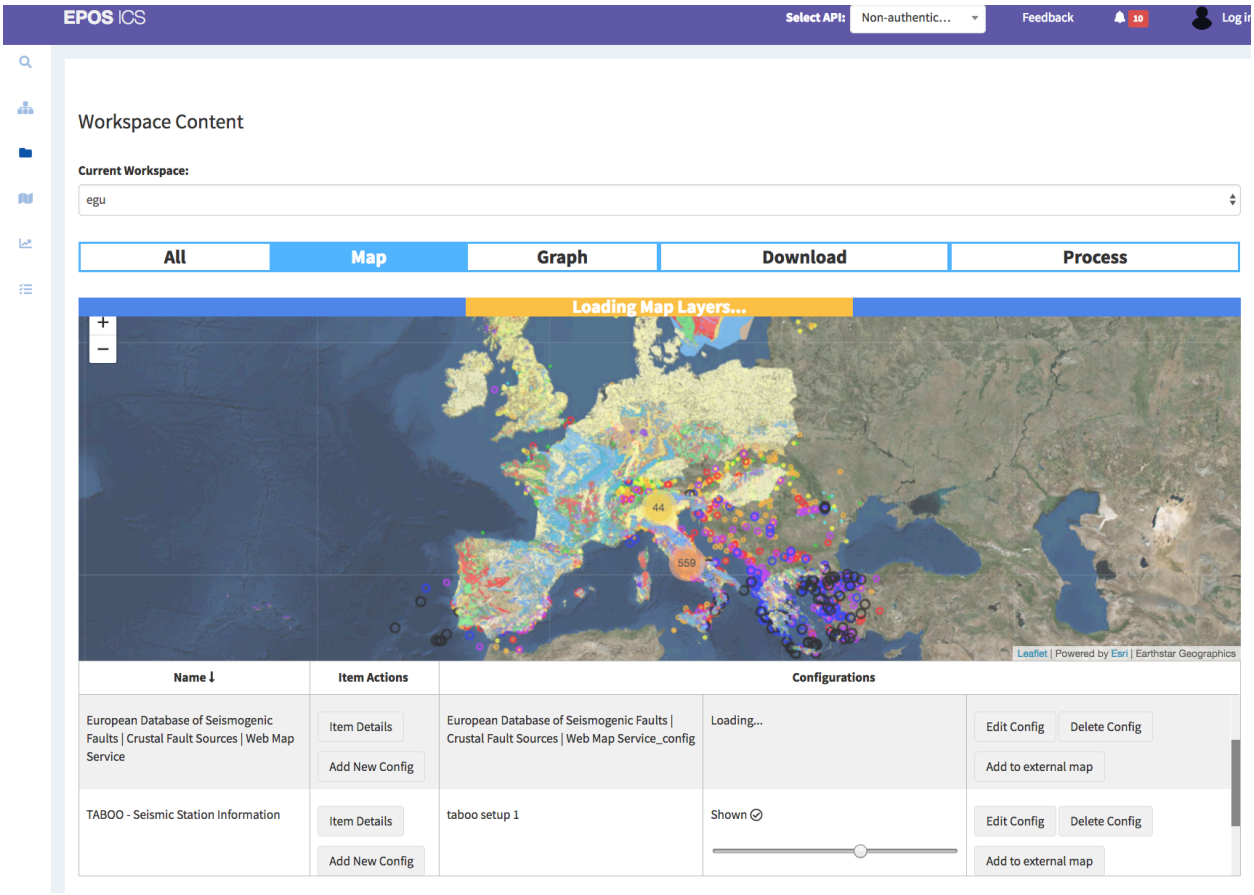



***Figure 15*** *Snapshot of the ICS-C demonstrator*

## *5.3.2 AAAI*

The current architecture includes AAAI. This has been implemented using UNITY[7] and has involved close cooperation with CYFRONET. However, in May 2018 (M32) an integrated authentication system for academic communities was announced and this has now been integrated. Authorisation is more complex and depends on rules agreed with the TCS (within the context of the financial, legal and governance traversal workpackages of EPOS-IP) for each of their assets and included further metadata elements into the CERIF catalog to control such authorisation. AAAI will be continuously evolved and updated to ensure appropriate security, privacy and governance.
Related to this, the GUI now provides a user notification pointing to a legal disclaimer for the EPOS system.

## *5.3.3 Workflow Management*

A major requirement of the system, after asset discovery, is the construction of workflows that can be used to access / process data. This has implications for the entire software stack; visually designing the workflows, managing and persisting inputs and outputs, scheduling and execution of processes, access to metadata, access to data and service from the TCS. The topic as whole required significant analysis of requirements and available technologies. Working in cooperation with VRE4EIC project we have the basic components for (a) a general workflow manager interface; (b) interfaces to specific workflow managers such as Taverna[8].

[7] http://www.unity-idm.eu
[8] https://taverna.incubator.apache.org/

### *5.3.4 ICS-D*

The distributed services offered by the ICS-D facet of the architecture ties-in with the workflow management, as the distributed services in question beyond just being discoverable are likely candidate for inclusion in processing workflows. A specification of the metadata elements required for ICS-D has been produced and forms part of the architecture. ICS-D will appear to the workflow, or to the end-user, as a service accessed through an API. However, the choice of which ICS-D to use and the deployment of a workflow across one or more ICS-Ds requires optimisation middleware. Results from the PaaSage project[9] are relevant and the concurrent MELODIC project[10] offers optimisation including that based on dataset placement and latency. Further refinement of requirements and the architectural interfaces took place M30-36

### *5.3.5 Visualisation*

Beyond simple map visualisations that consume web map services the ICS-C user interface may be required to support additional types of visualisation. This set of supported visualisation types and associated data formats needs confirmation as it will not be practical to support all formats of data for all types of visualisation.

### *5.3.6 TCS Integration & Harvesting*

Greater interaction with TCS to ensure that their metadata, data and services are available for harvesting in the appropriate format and to populate the CERIF data model has been achieved and will be continued.

### *5.3.7 Quality and Quality Assurance*

A process for

(1) Handing over ICS-C developments (modules) from the development team to the production team;
(2) Accepting developments (assets) from the TCS into the development environment (mainly provision of appropriate metadata);
(3) Accepting assets of ICS-D and CES into the development environment (mainly provision of appropriate metadata);

has been defined and is being refined to ensure appropriate quality.

Furthermore, KPIs (Key Performance Indicators) concerning the performance if the ICS-C environment are being defined during the pre-Production phase to be ready for monitoring during production.

# 6. CONCLUSION

The architecture outlined and demonstrated (in successive prototypes) in EPOS-IP has found favour (not without some criticism of course – leading to agile improvements) from the user community. Furthermore, the prototype system has passed TRA (Technological Readiness Assessment) procedures within the governance of the EPOS-IP project. Currently the ICS is undergoing validation tests. The architecture meets the requirements, it is state of the art and has a further development plan.

[9] https://paasage.ercim.eu/
[10] melodic.cloud/