H2020-SC6-GOVERNANCE-2018-2019-2020

DT-GOVERNANCE-05-2018-2019-2020

D4.6 Micro Proxies and services catalogue - Final

Project Reference No	959157 — ACROSS — H2020-SC6-GOVERNANCE-2018-2019-2020
Deliverable	D4.6 Micro Proxies and services catalogue – Final
Work package	WP4 - ACROSS Modules Set-Up
Nature	OTHER
Dissemination Level	PU = Public
Date	31/07/2023
Status	Final 1.0
Editor(s)	Vincenzo Savarino (ENG)
Contributor(s)	Petros Christopoulos (Grnet), Enrique Areizaga (TEC), Valentín Sánchez (TEC), David Britnell (DATAPORT)
Reviewer(s)	David Britnell (DATAPORT), Thilo Ernst (FHG)
Document description	This report provides the final report of design and implementation of components of *Harmonization and connectors layer* of ACROSS architecture. It documents the requirements, functional specification, design, description of modules, and description of components APIs in line with the updated requirements and feedbacks of previous validation stage.

About

The project is co-funded by the European Commission's Horizon 2020 research and innovation framework programme. Spanning through three years, ACROSS consists of a consortium of 10 partners from 7 countries: Athens Technology Center (coordinator), Tecnalia, Dataport, Engineering, Fraunhofer, GRNET, TimeLex, The Lisbon Council, Waag and VARAM. The project kicked off its activities in February 2021, with an energising online meeting, where all partners took the floor to present their plans to make the project a great success.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
V0.1	22/02/2023	Extension of ToC with the planned feature extension	ENG
V0.2	03/05/2023	Contribution on Service Catalogue model extension. Specification of Service Adapter structure and data models	TECNALIA, GRNET, DATAPORT, FHG
V0.3	23/06/2023	Federated Catalogue section	ENG

V0.4	20/07/2023	Pre-release ready for internal review	ENG
V0.5	26/07/2023	Feedback and revision	DATAPORT, FHR
V1.0	31/07/2023	Final release	ENG

Executive Summary

The main objective of the ACROSS project is to provide the means (tools, methods and techniques) to enable user-centric design and implementation of interoperable cross-border (digital) public services compliant with the current European regulations (e.g. the Single Digital Gateway (SDG) and Once-Only principle (OOP), European Interoperability Framework (EIF)) where the private sector can also interconnect their services **while ensuring the data sovereignty of the citizens, who can set the privacy level that will allow the public and private sector to access to their data based on their requirements**.

This report documents the overall result of activities performed in Task 4.2 " Public & Private sector offerings management tool" describing the current version of Service Catalogue components and related service proxy adapters. This report is seen as the final version of a living document, started from the deliverable *D4.4 Micro Proxies and services catalogue-Initial* and updated in the following *D4.5 Micro Proxies and services catalogue-Intermediate,* describing the finalization of the design and implementation of Service Catalogue and Adapters, by addressing the refinement of technical and user requirements of ACROSS platform complemented by a detailed technical and usage documentation.

Table of Contents

LIST OF FIGURES

List of Tables

List of Terms and Abbreviations

Abbreviation	Definition
API	Application Programming Interface
CH	Clearing House
CPSV-AP	Core Public Service Vocabulary Application Profile
DAPS	Dynamic Attribute Provisioning Service
DCAT	Data Catalog Vocabulary
DPV	Data Privacy Vocabulary
ECC	Execution Core Container
eIDAS	electronic Identification, Authentication and trust Services
EIF	European Interoperability Framework
ELI	European Legislation Identifier
EU	European Union
GDPR	General Data Protection Regulation
HTTPS	Hypertext Transfer Protocol Secure
ID	Identification
IDS	International Data Spaces
IDSCP	International Data Spaces Communication Protocol

ISA2	Interoperability solutions for public administrations, businesses and citizens
ISO	International Organization for Standardization
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation Linked Data
JWT	JSON Web Token
OAUTH2	Open Authorization 2.0
ODRL	Open Digital Rights Language
OOP	Once Only Principle
PA	Public Administration
PoC	Proof of Concept
RAM	Reference Architecture Model
REST	Representational state transfer
SDG	Single Digital Gateway
SME	Small Medium Enterprise
SSO	Single Sign On
ToC	Table of Content
UC	Usage Control
URI	Universal Resource Identifier
URL	Universal Resource Locator
WP	Work Package
WS	Web Socket

1 Introduction

1.1 Purpose and Scope

The main goal of ACROSS is provide a holistic solution that allows public administrations to deliver a user-centric interoperable cross-border mobility service compliant with the current European regulations where the private sector can also interconnect their services while ensuring the data sovereignty of the citizens. To this end one of the ACROSS objectives is to provide a **set of connectors and data harmonization tools** that will facilitate the actual interoperability of the cross-border mobility services through the connection of public services so that they can interoperate with services from other countries as well as with those of the private sector also to include their services and offerings.

The ACROSS solution aims to provide a common semantic and functional stratum enabling cross border access to data and processes backed up by functionalities and capabilities for data collection and harmonisation according to a set of common and shared data models based on European standards such as Core Public Service Vocabulary[1], Core Person Vocabulary[2] and DCAT for metadata[3].

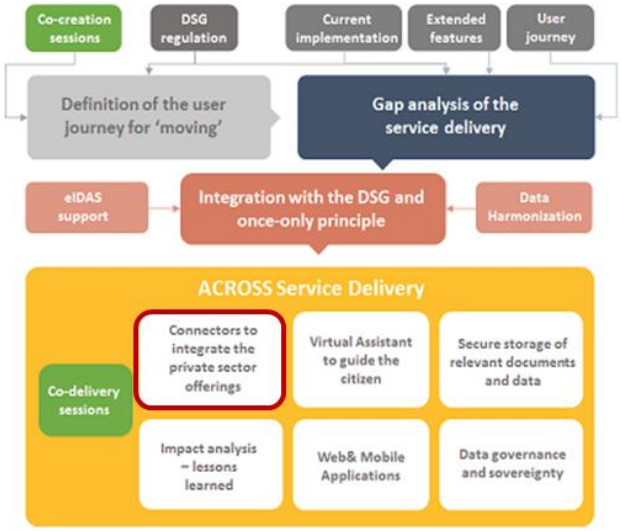

Figure 1 - ACROSS Conceptual approach and main components supporting user centric cross-border mobility service delivery.

The adoption of customizable connectors will provide proxy functionalities to connect and access public/private services offered by both the Public Administration and third parties and to get data from heterogeneous sources such as repositories, existing systems (e.g. owned by PA), etc. that could expose different interfaces.

To this aim, these customizable adapters will offer a set of software components ready to be used or tailored according to specific needs and requirements to allow the connection of data

[1]https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/core-public-service-vocabulary

[2]https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/core-person-vocabulary/release/100

[3] https://www.w3.org/TR/vocab-dcat-2/

sources and public/third-party private services to ACROSS Platform for the provisioning of cross-border mobility service delivery (Figure 1).

1.2 Approach for Work Package and Relation to other Work Packages and Deliverables

WP4 aims to provide a set of tools and technological solutions that implements the borders of ACROSS Platform; in details, these tools and solutions will concern authentication aspects compliant with eIDAS, user support tools to facilitate both the interaction of the citizens with User Journey Services and connection to public and private sector services.

This report is seen as a living document, as evolution of the first one (D4.4) and the outcomes and the implementation of components presented in this deliverable are a subject to continuous refinements and modifications, based on the progress of all technical work packages (WP3, WP4, WP5), as well as the validation and evaluation phases performed in WP6 activities. In fact, the services and tools developed in this WP are integrated into the platform created in WP5, in line with the architecture described in [1] and adopted in the use cases in WP6.

1.3 Methodology and Structure of the Deliverable

This deliverable aims to report the design and implementation of data harmonization and connectors layer of ACROSS platform. To this end the deliverable has been structured in the following sections:

Section 2 describes the final release of components of Data harmonization and connector layer of ACROSS solution and its relationship with ACROSS architecture components. The defined service model to support interoperability is described and updated to address Use case requirements and the current interaction with the other components of ACROSS platform. Besides, a detailed description of Service Catalogue is provided. Finally, the section provides an update of baseline technologies that can be used as needed connector implementation and some examples on how to adopt them.

Section 3 conclusions and overall outcomes.

Annexes provides information about the defined service model and the APIs exposed by the Service Catalogue. Finally, Annex III provides a mapping of covered requirements as identified in D5.2.

2 Data harmonization and connectors

The following sections provide the updated technical context and details of the design and implementation of the main components of Data harmonization and connectors layer in line with the updated ACROSS architecture as documented in D5.2.

2.1 ACROSS context and architecture overview

The following Figure 2 provides a conceptual view of ACROSS architecture by identifying the related layers and actors.

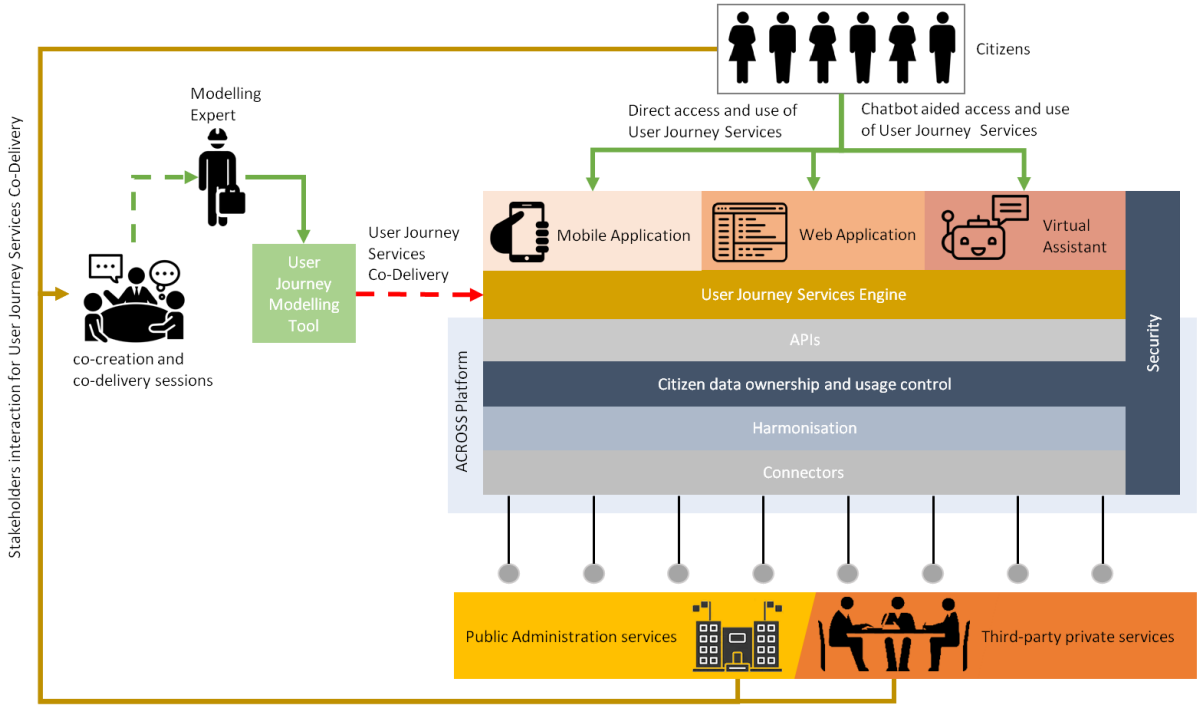

Figure 2 - Conceptual architecture of ACROSS

In particular, the **Harmonization and Connectors layers** provide the south bound connection with external systems and services from public and private sectors to provide a uniform description of the services and the related invocation. To this end these layers should provide for each identified service a service adapter (Figure 3), that is a ***single instance of adaptation*** of that service (public & private) for its use in ACROSS Platform from several points of view for its use in the upper layers: ***informational, data governance and service invocation***. To support that adaptation, Harmonization and Connectors layers should include a service catalogue with the aim to provides all functionality to register, model, map and publish and manage a uniform and harmonized machine-readable description of public and private services, according to the three

above points of view, needed to support the uses of each service by the upper layers of ACROSS Platform.

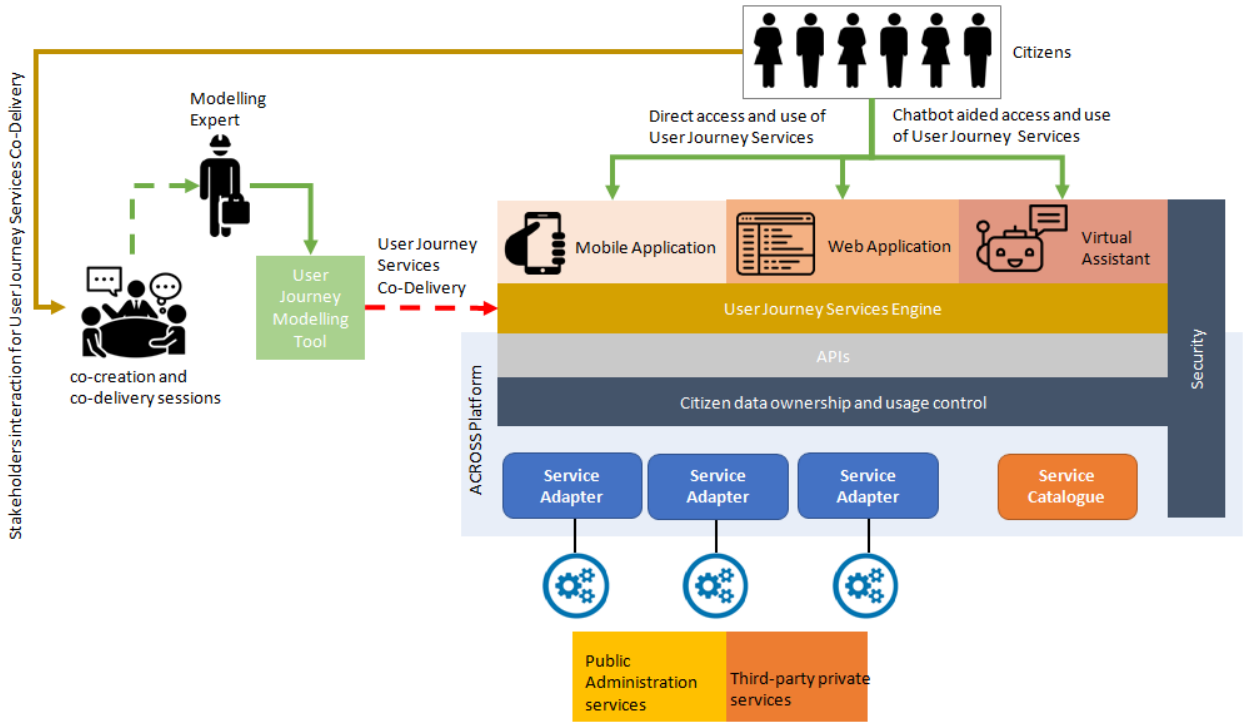

Figure 3 - Service Adapters and Catalogue. For each service used by ACROSS Platform a service adapter is defined and related information is stored in the Service Catalogue.

In the definition of ACROSS architecture (Figure 4), the upper conceptual layers have been included in the Data Harmonization and connectors main components blocks, namely **Service adapter** and **Service Catalogue** (Figure 5), to provide the following features:

- Secure connection to internal system of the PAs and services.
- Secure connection to private services to support cross border services.
- Uniform API for data access.
- Exposure of sub-set of existing functionalities/services of PAs and private
- Data access management based on authentication, authorization and privacy management for the requested information.
- Semantic adaption to support interoperability according to common vocabularies
- Catalogue of services instances, models and metadata registry

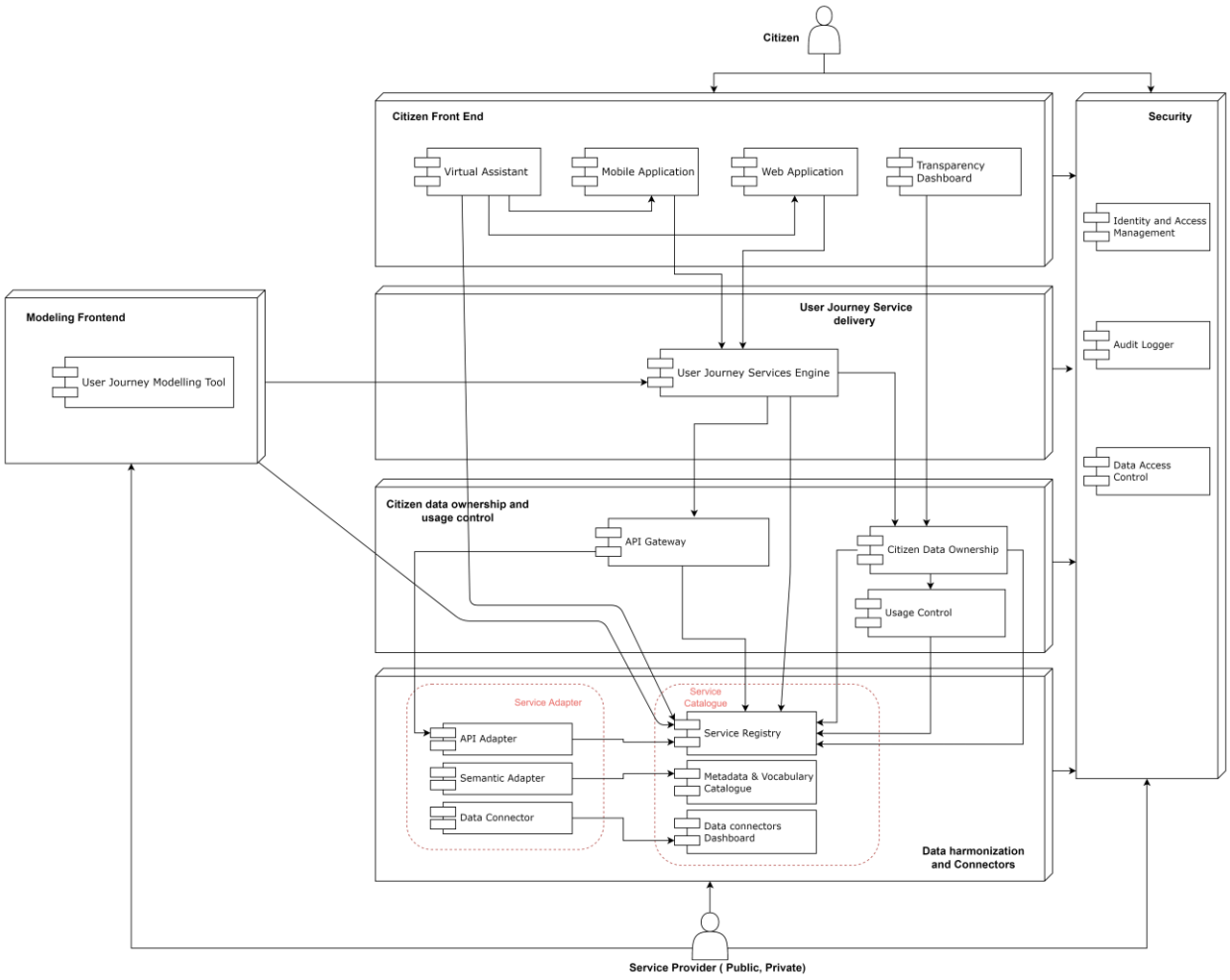

Figure 4 - Architecture of ACROSS Platform from [1]

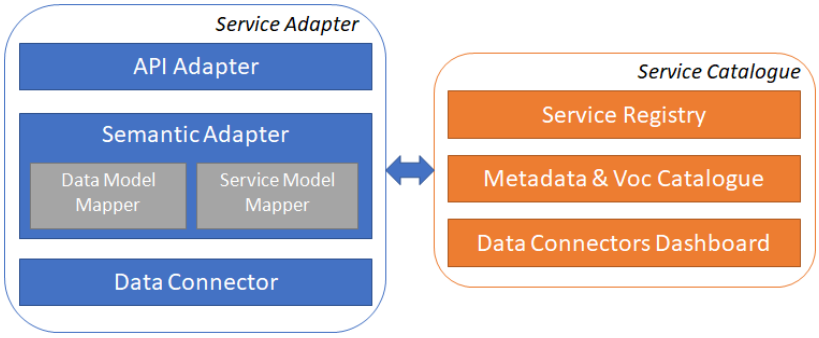

Figure 5 - Main components of Data harmonization and connectors. Each component is involved in the flow according to the type of adaptation performed.

The **API Adapter** module is responsible to expose a standard set of APIs for data access (by means of data connectors) or functionalities supplied by internal or legacy systems of PAs and private

service providers. API adapter is used by the API gateway to invoke all services included in the User Journey Service Engine. Moreover, it supports the enforcement of each request verifying, for instance, if a valid token is provided and if the token grants access to the requested resource. The **Semantic adapter** contributes in the service adaptation, through the Data Model Mapper, in order to supply data gathered from Data Connector in the expected format or, through the Service Model Mapper, to the semantic adaptation of service descriptions according to the standard and shared model adopted by the Service Catalogue. The **Data Connector** is actually the module that performs the technical integration of Legacy/Proprietary systems. Each instance of service adaptation has an ad-hoc developed data connector accordingly to the type of integration (e.g. read from API, read csv or json file or other files, read from SQL or NoSQL databases, etc.), covering also all the security aspects of authentication and authorization.

The above Service adapter modules are supported by the Service Catalogue by storing all the needed information or produced by the service adaptation modules to be consumed by the upper layers. The front-end and business layer of the Service Catalogue is the **Service Registry,** responsible to provide APIs for programmatically interaction and dashboards and graphical editors. The backend part is performed by the **Metadata and Vocabulary Catalogue.** Finally, the Data Connector Dashboards module provides a management cockpit of all available data connectors instances.

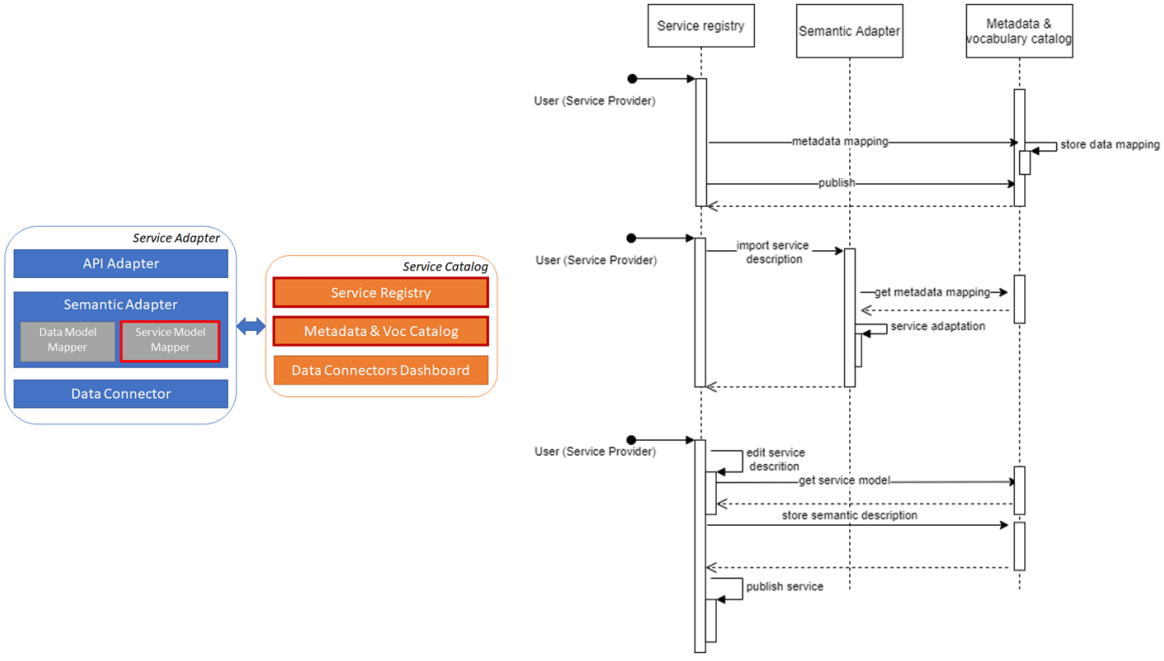

Figure 6 - Modules involved (and related flow) in Service model adaptation

Each component and related sub modules are invoked in relation to the type of service adaptation and the interaction with the upper layers. For example, the Semantic Adapter component can be invoked in order to provide functionalities for the provisioning of a common information model of the service registered to the platform. (i.e CPSV, see sections 2.2 and 2.3) (Figure 6), or used to support the interaction of ACROSS platform with an external service by defining a data connector and related data model adaptation (Figure 7).

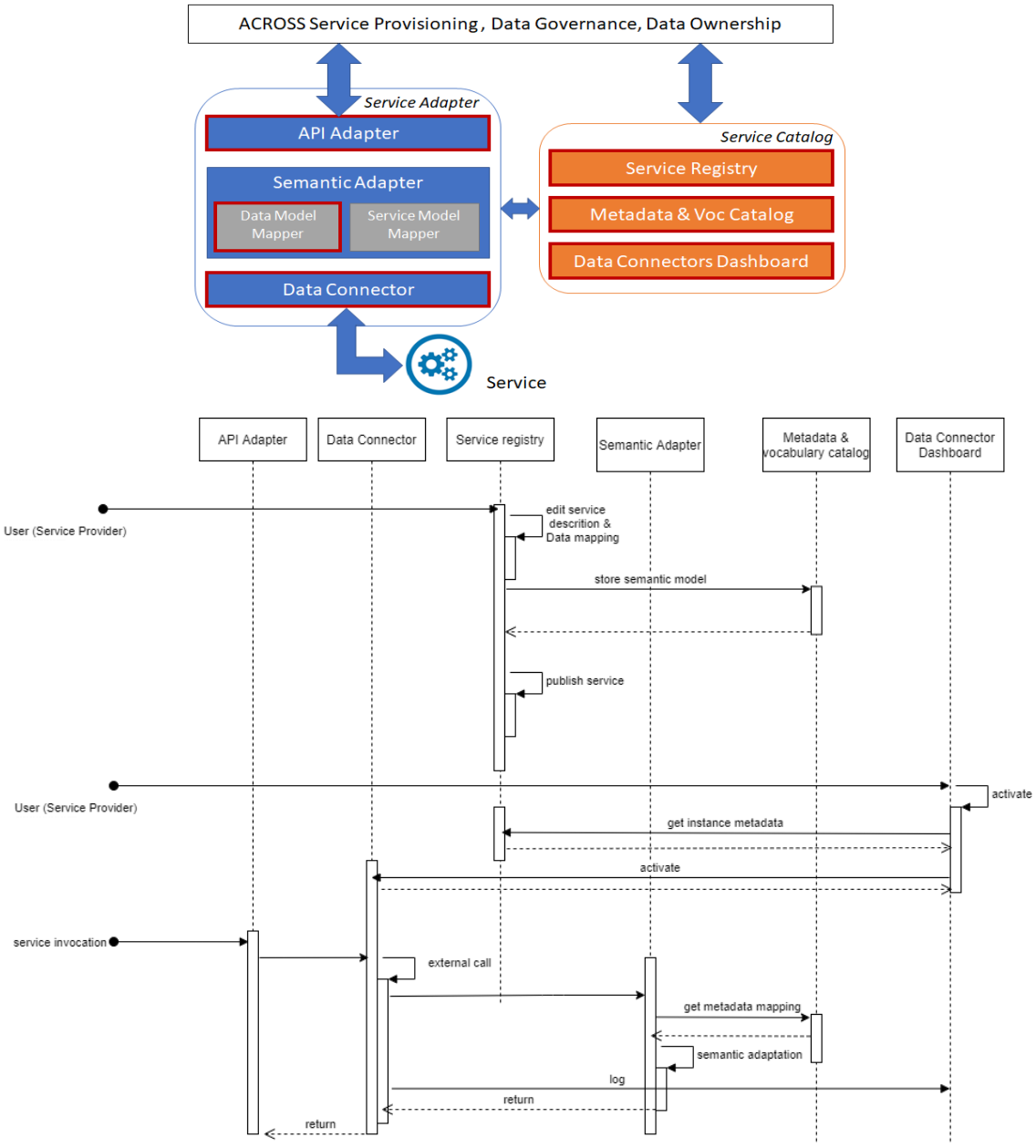

Figure 7 - Modules involved (and related flow) in Service invocation adaptation

In general, as depicted in the following Figure 8, each module interacts or it is invoked in order to provide a specific functionality or a piece of information, to be used also externally to ACROSS platform, for example to export service model description in a standard format.

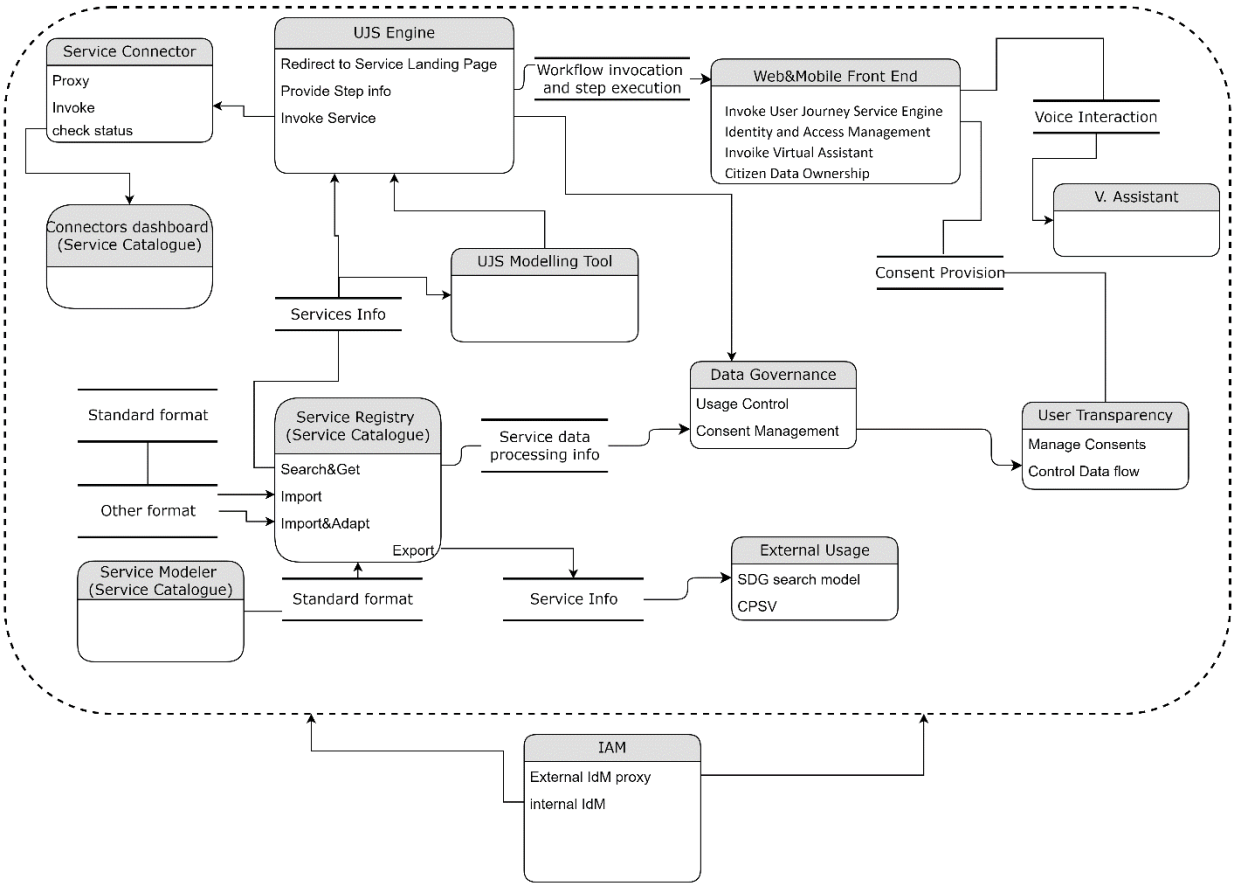

Figure 8 - Service Catalogue and Service Adapter modules interaction map with other modules of ACROSS platform.

2.2 Service Model

As described in the previous section, to support a uniform and harmonized machine-readable description of public and private services a service model has been defined to collect all information from the three point of view (Informational, Service invocation, Data Governance&Ownership) and managed in the Service Catalogue. The idea is to define this model by including and extending existing common models to describe each view (Figure 9).

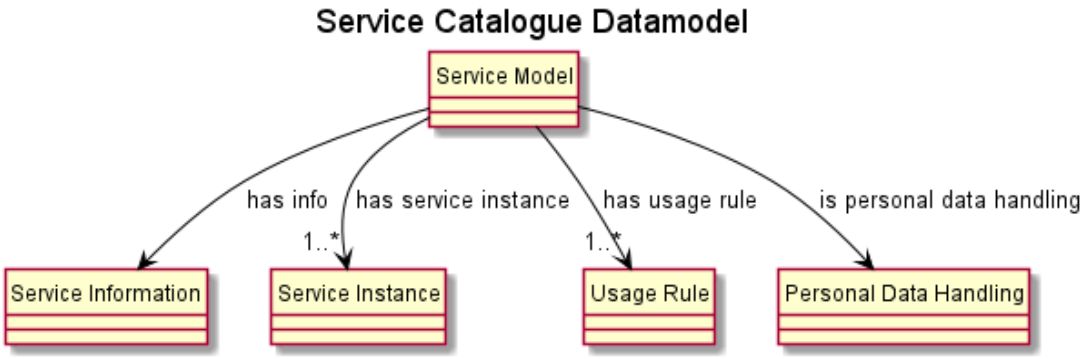

Figure 9 - Graphical representation of the Service Model main classes adopted in ACROSS

2.2.1 Information view

This section provides all information metadata of a service. This section follows the CPSV-AP, the Core Public Service Vocabulary Application Profile [2], a data model provided by the ISA2 Programme that is the result of a joint effort from different public administrations to reduce interoperability barriers. The CPSV-AP provides public administrations with a common data model for describing public services related to business and life events and to facilitate the set-up of catalogues of services oriented to businesses and citizens. With the CPSV-AP, public administrations can (i) provide information on public services in a user-centric way, grouped logically around business or life events and other ways of classifying; (ii) map different data models to a common model requiring only one single description and (iii) federate and publish information on Points of Single Contact and eGovernment portals in a more efficient and interoperable way.

The definition of CPSV-AP model has followed the main objective to provide a lightweight and modular standard that can be reused. The CPSV-AP specifies only 2 mandatory classes (Public Service and Public Organisation) (Figure 10) and the data model itself is based on other standards such as the Core Public Organisation[4], the Core Criterion and Evidence[5] and the ELI Vocabulary[6] .

The CPSV-AP enables the description of public services and the associated life and business events, by standardising the semantics of personal milestones, including having a child, beginning

[4] https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic/solution/core-public-organisation-vocabulary/about

[5] https://joinup.ec.europa.eu/solution/core-criterion-and-core-evidence-vocabulary/about

[6] https://publications.europa.eu/en/web/eu-vocabularies/model/-/resource/dataset/eli

education, looking for a new job, as well as professional changes such as starting or financing a company, hiring an employee. CPSV-AP model will make information about life and business events structured, easier to capture and machine-readable. Public administrations and service providers can use this to guarantee a degree of cross-domain and cross-border interoperability between public service catalogues.

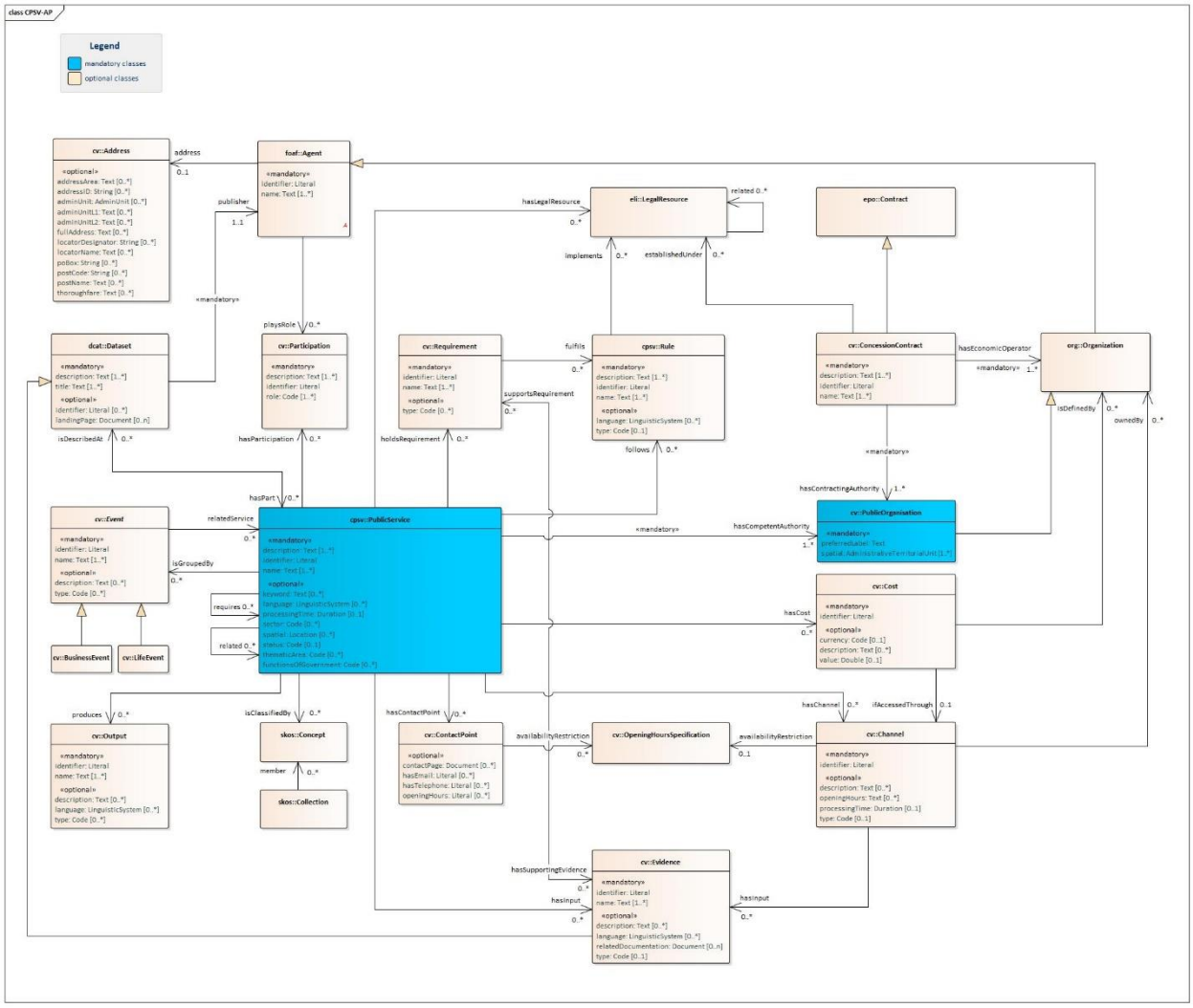

Figure 10 - Graphical representation of the relationships between the classes and properties of the full Core Public Service Vocabulary Application Profile (figure from [2])

These aspects contribute to address ACROSS requirements (section 7) for a Semantic and technical interoperability with SDG (single digital gateway) in order to "*Easy and user-friendly access to information means enabling the users to easily find the information, to easily identify which parts of the information are relevant for their particular situation and to easily understand the relevant information*"[4] and to address the recommendation to *"...use the Core Public Services Vocabulary (CPSV) to facilitate interoperability with national service catalogues and semantics. Member States should be encouraged to use the CPSV, but are free to decide to use national solutions. The information included in the repository for links should be made publicly available in open, commonly used and machine-readable format, for example by application programming interfaces (APIs), in order to enable its reuse."*

In the context of Public Services, the CPSV-AP data model brings in the concepts of input (class Evidence) and output (actual result of executing a given Public Service) that might link to other Public Service's input and output. Furthermore, the CPSV-AP data model provides properties (such as "required" and "related") which allow to explicitly indicate other required and/or related public services. This information will be used to support the user journey modelling and service engine (see Figure 8).

2.2.2 Usage Rule and Personal Data Handling views

Usage Rule and Personal Data Handling views capture information about for handling data access rights. In particular Usage Rule view provides an interoperable information model, vocabulary, and encoding mechanisms for representing statements about the usage of content and services. This model, as described in [3] is based on IDS Usage Control Model [9] a specialization of the Open Digital Rights Language (ODRL)[7] to enforce usage restrictions for data, namely contract agreements, after access has been granted. Therefore, the purpose of usage control is to bind policies to data being exchanged and to continuously control the way how messages may be processed, aggregated, or forwarded to other endpoints. In the defined Service Model, a one-to-many relationship is defined (see section 5.2) by mapping a service description with one or more contract agreements defined in a separated model and not stored and managed by the Service Catalogue but managed by the components of Data Ownership and Usage Control layer.

[7] https://www.w3.org/TR/odrl-model/

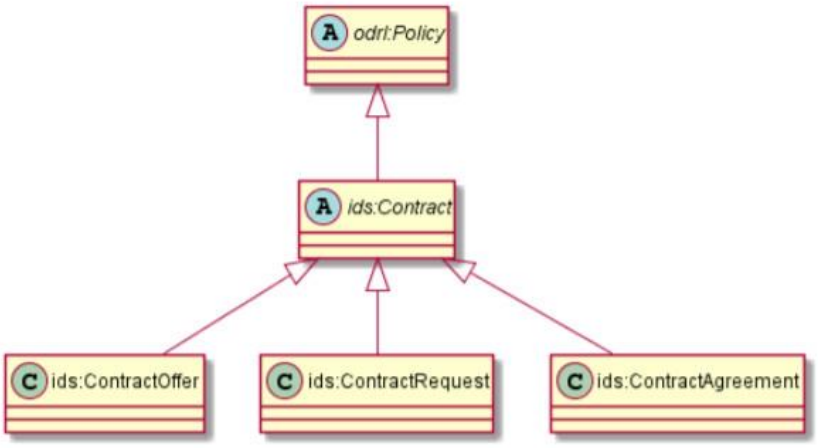

Figure 11 - IDS Contract Agreement

The personal Data Handling section is a specialized profile of The Data Privacy Vocabulary (DPV) [5] providing terms (classes and properties) to describe and represent information about personal data handling. In particular, the vocabulary provides extensible taxonomies of terms to describe the following components:

- Personal Data Categories
- Purposes
- Processing Categories
- Technical and Organisational Measures
- Legal Basis such as Consent
- Entities such as Recipients, Data Controllers, Data Subjects
- Rights
- Risks

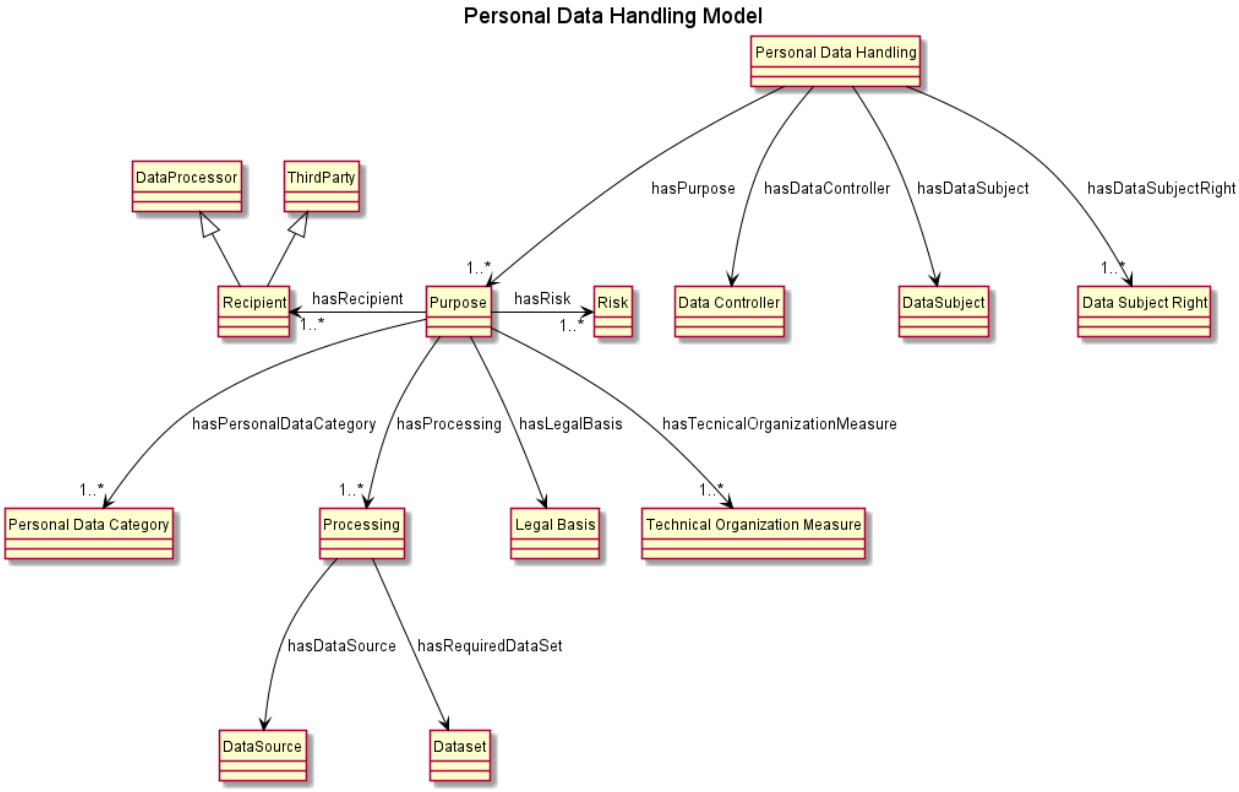

Figure 12 - Class diagram of Personal Data Handling model as profile of Data Privacy Vocabulary (DPV)

These terms are intended to represent personal data handling as machine-readable information by specifying personal data categories undergoing processing, its purpose(s), the data controller(s) involved, recipient(s) of this data, the legal bases or justifications used (e.g. consent or legitimate interest), involving technical and organisational measures and restrictions (e.g. storage location and storage duration), the applicable rights, and possibility of risks.

Examples of applications where the concepts provided by the DPV can be used are:

1. represent policies for personal data handling
2. represent information about consent e.g. provenance of consent
3. log/document personal data handling actions e.g. by a data controller
4. support automated checking of legal compliances of data handling ex ante (prior to processing), or ex post (i.e. check compliance after processing)

In accordance with the above application scenarios, the Personal Data Handling section uses the DPV vocabulary to collect all needed information that will consumed by the Consent Manager component of Data Ownership and Usage Control layer.

2.2.3 Service Instance view

This view provides all operational information to manage and invoke each service instance mediated by the related service adapter. It includes at least:

- Internal technical details to interact with internal components (e.g. data governance)
- Data/service connector invocation
- API Documentation (Open API/Swagger)
- Authentication and Authorization endpoints

The view will be extended with further information during the evolution of service adapter and all the component of ACROSS platform.

Detailed information about Service Model is provided in Annex I.

2.3 Service Catalogue

The following sections provides an overview of the current Service Catalogue implementation [6][7]. Service Catalogue provides all functionality to register, model, map and publish and manage all the information needed to support the uses of a service (public&private) according to the three points of view of Service Model described in the previous section:

- Informational
- Service Invocation
- Semantic interoperability & Personal Data Governance

The catalogue enables the storage and publishing of service by providing general, technical and data processing information based on standard models in particular based on CPSV-AP.

The Service Catalogue is a layered application implementing the Service Registry (front-end and backend) and Metadata Catalogue features, to provide APIs (see Appendix II) for programmatically interaction with other components of ACROSS platform and dashboards and a graphical editors supporting users to manage service descriptions and related model adaptation (Figure 13).

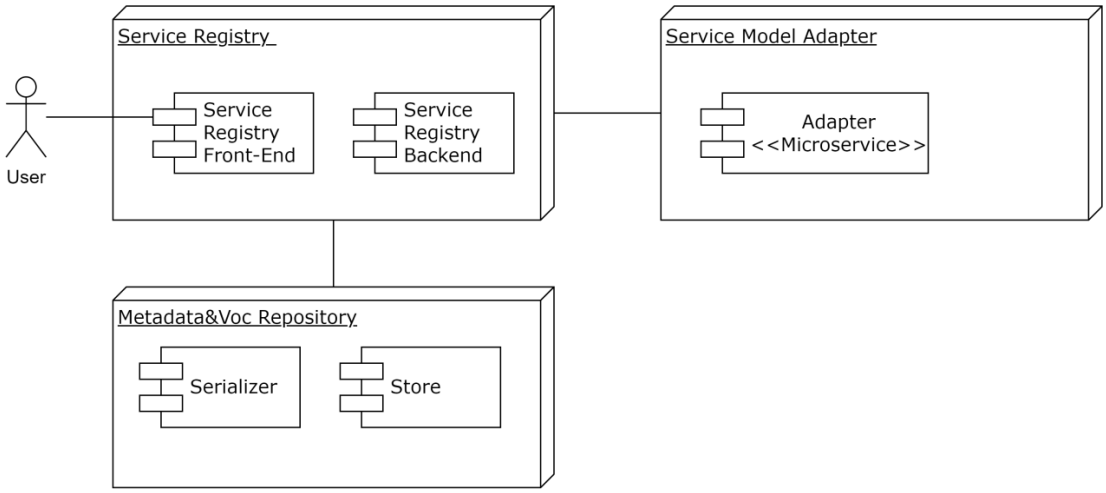

Figure 13 - Main modules implemented by the Service Catalogue

The Backend is implemented as Spring Boot[8] Java microservice, and will be deployed with a tightly coupled storage service (MongoDB[9] 4.2+). The Front-end, is an Angular portal based on Nebular[10] framework (Figure 14).

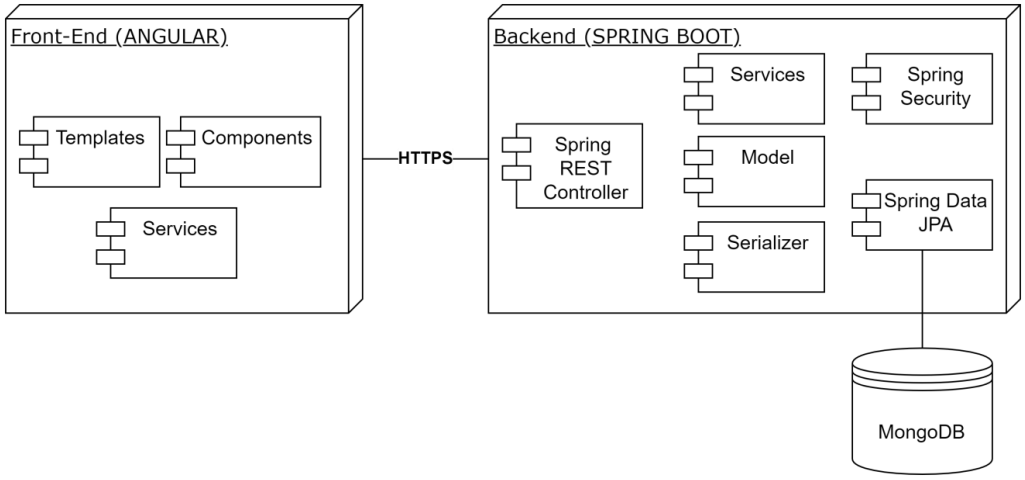

Figure 14 - Front-End and Backend of Service Catalogue implementation

[8] https://spring.io/projects/spring-boot
[9] https://www.mongodb.com/
[10] https://akveo.github.io/nebular/

The two layers can be deployed as Docker[11] containers, based on Tomcat Alpine image[12] and paired with a MongoDB container. This adoption of several reliable and production ready technologies (Figure 15) guarantees robustness and modularity of the solution.

Figure 15 - Adopted technologies in Service Catalogue implementation.

Service Catalogue architecture implementation is completed by integrating Spring Security and Keycloak[13] that supports OpenId Connect[14] and OAuth2[15] authorization framework. The Service Catalogue uses the Open Id Connect protocol upon the OAuth2 Authorization workflows, to perform User authentication and obtain an Access Token (JWT), which will be used to grant access.

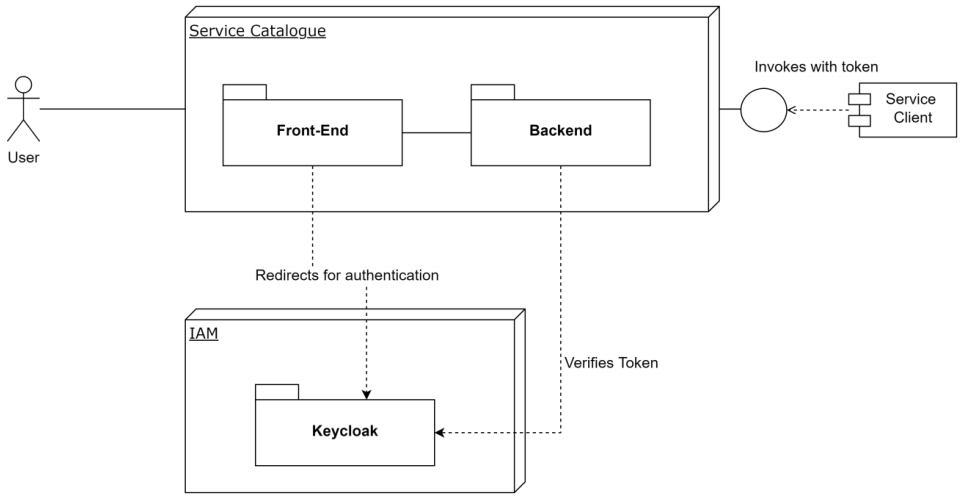

Figure 16 - Interaction of Service Catalogue with Keycloak for identity and access management

[11] https://www.docker.com/
[12] https://hub.docker.com/_/tomcat
[13] https://www.keycloak.org/
[14] https://openid.net/connect/
[15] https://oauth.net/2/

Similarly, a client application/service wanting to interact with the Service Catalogue, will perform OAuth2 Authorization, obtaining an Access Token to be used in the request to APIs (Figure 16).

The choice of Keycloak provides an out of box solution for a rapid security layer development of application with supporting features such as Single-Sign-On (SSO), Social Login, User Federation, Client Adapters, Admin Console and Account Management Console and finally Identity Brokering[16] (Figure 17).

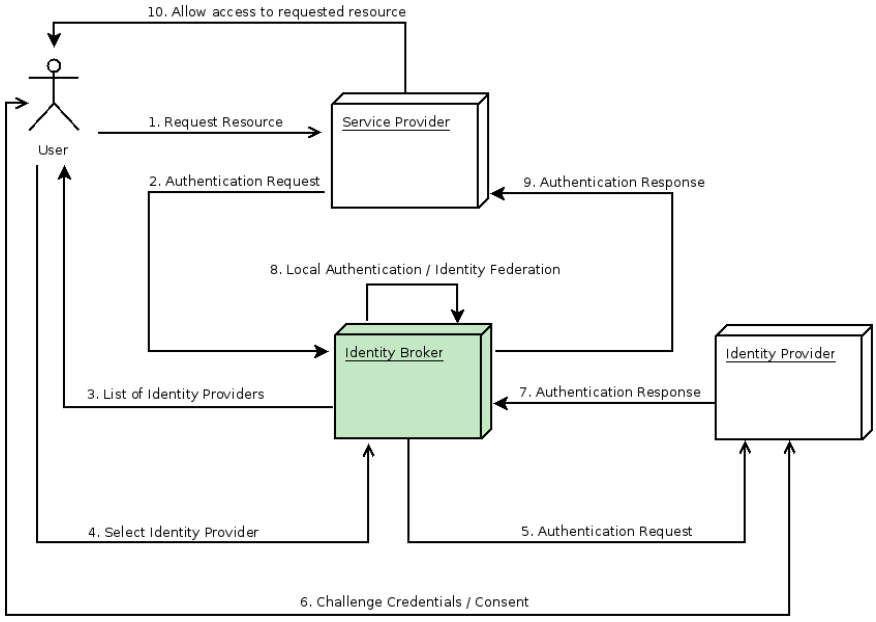

Figure 17 - Identity brokering flow supported by Keycloak

This last aspect will facilitate the integration of the Service Catalogue with multiple and specific identity Systems.

The following sections provide a summary of Service Catalogue frontend functionalities. Detailed information on the installation and usage of Service Catalogue is provided in [7]

2.3.1 Service Manager

The Service Manager is a multi-role, Angular[17] based admin dashboard implemented with the aim to include all module sections to interact with Service Catalogue but at the same time,

[16] https://www.keycloak.org/docs/latest/server_admin/#_identity_broker
[17] https://angular.io/

according to the role of authenticated user, to manage the consents registry as Data Controller (Figure 18).

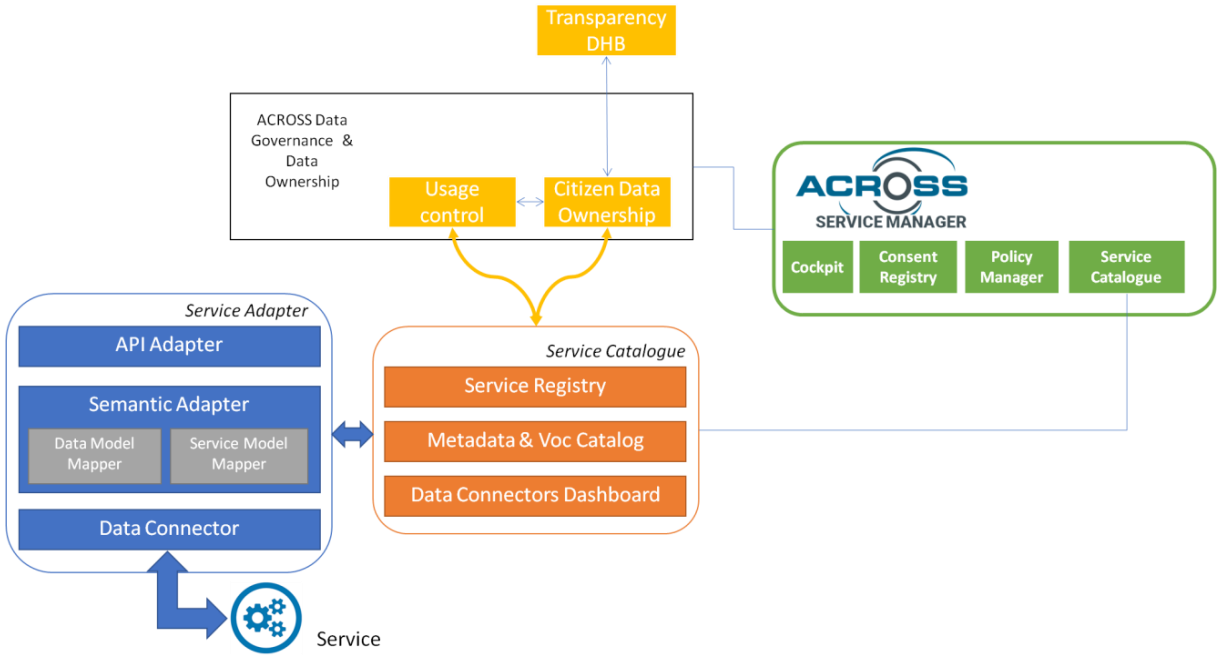

Figure 18 - The Service Manager admin dashboard provides a modular web interface to interact with several layers of ACROSS Platform.

The Service Manager uses Keycloak as identity broker providing at login phase an extensible page to select optional authentication systems (Figure 19).

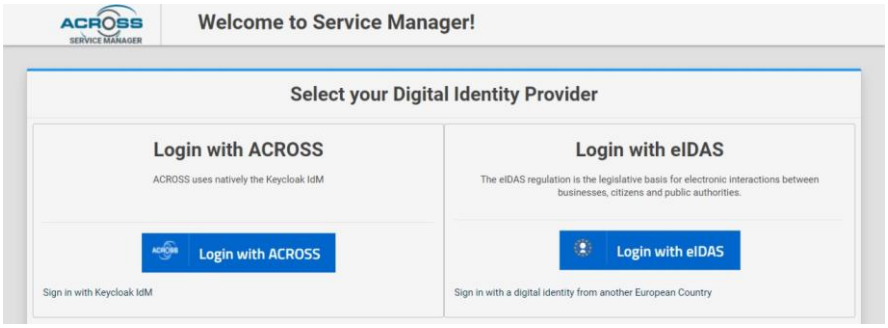

Figure 19 - Authentication page of Service Manager

In particular for eIDAS authentication, or national identity schemes, a dedicated adapter should be implemented, brokered by Keycloak.

Once authenticated, the user according to the roles assigned by means of Keycloak[18] the user can access to several sections. In particular the user can view the list of already inserted services by having a first look about their basic information (name, status, description...), or to be redirected to the "detail" page (Figure 20).

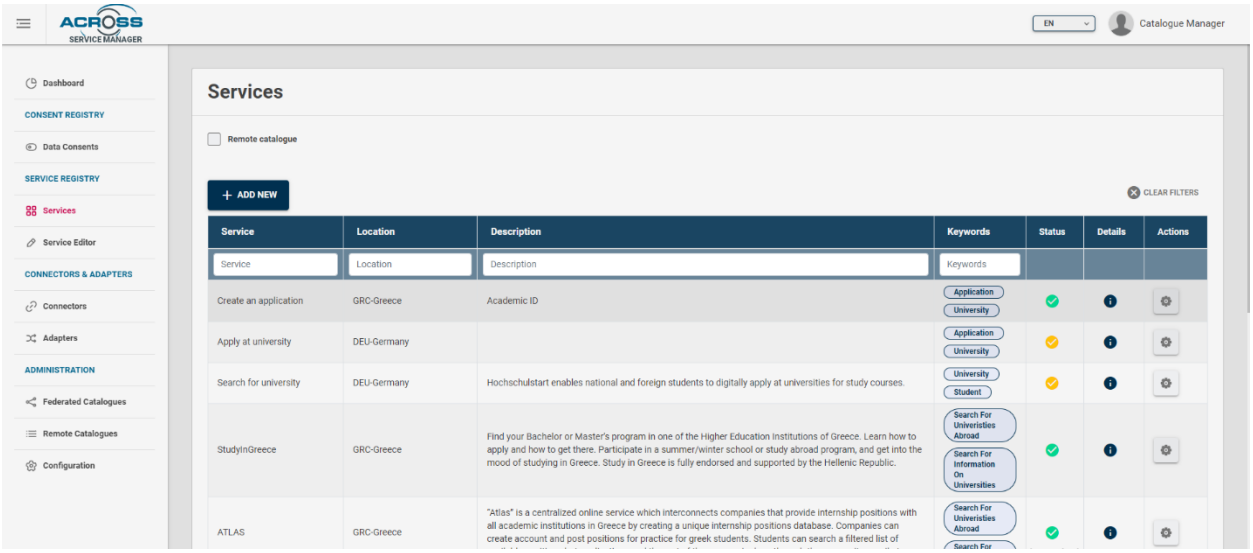

Figure 20 - Service list page

From "Actions" the user can perform several actions in accordance with the status of service (Figure 21):

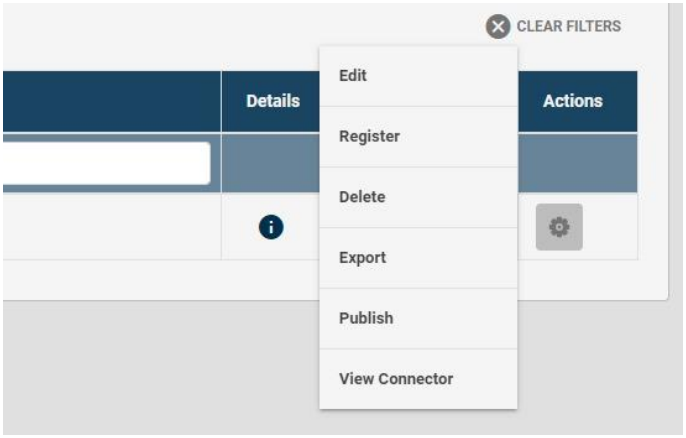

Figure 21 - Available actions in accordance to the status of service

- *Edit*. User can modify or complete service description, by entering in edit mode page

[18] https://www.keycloak.org/docs/latest/server_admin/

- *Register*. This action changes the status of service description into "completed". Once completed the service is searchable, by means of APIs exposed by the Service Catalogue, by ACROSS components.
- *Delete.* User can delete a service description. The action can be performed if the service is in the status of "UnderDevelopment" or after a de-registration.
- *Export.* User can export the description of Service by selecting different formats (Figure 22): JSON, JSON-LD, CPSV-AP Model (json-ld) and Single Digital Gateway Search Service model[19]
- *Publish.* By this action the Service Catalogue provides the availability to customize and manage multi publish actions. It lets to publish externally the service description at all or some information.
- *View Connector.* It lets to switch to the related connector adapter page.

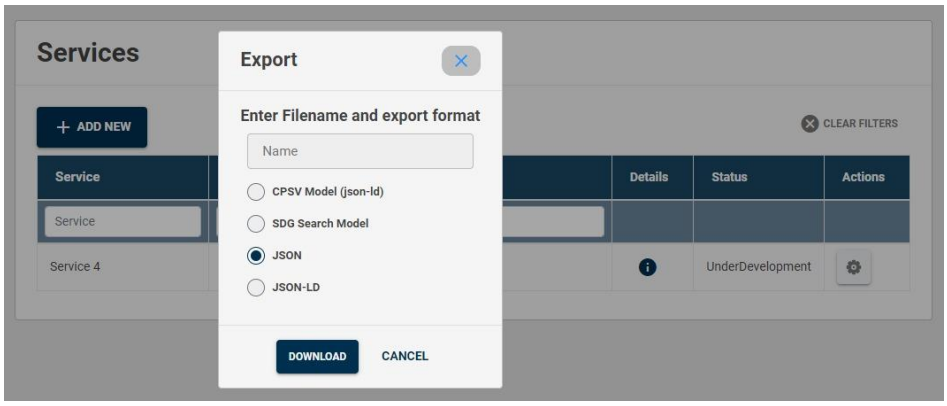

Figure 22 - Export dialog

The user can also choose a remote catalogue as source of services (see Figure 23 and section 2.3.2). If the “Remote catalogue” checkbox is checked and a remote catalogue source is selected, only services with status COMPLETED will be shown.

[19] https://github.com/catalogue-of-services-isa/SDG-search-service-model

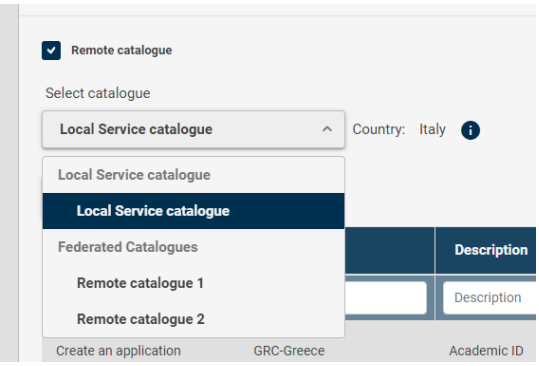

Figure 23 - Federated catalogue to select to see remote registered services.

From the list page the user can add a new service description by clicking on "Add new" button. The user is redirected to Service Editor page. It is composed in several tabs in relation to the Service model views described in section 2.2 and detailed in Annex I.

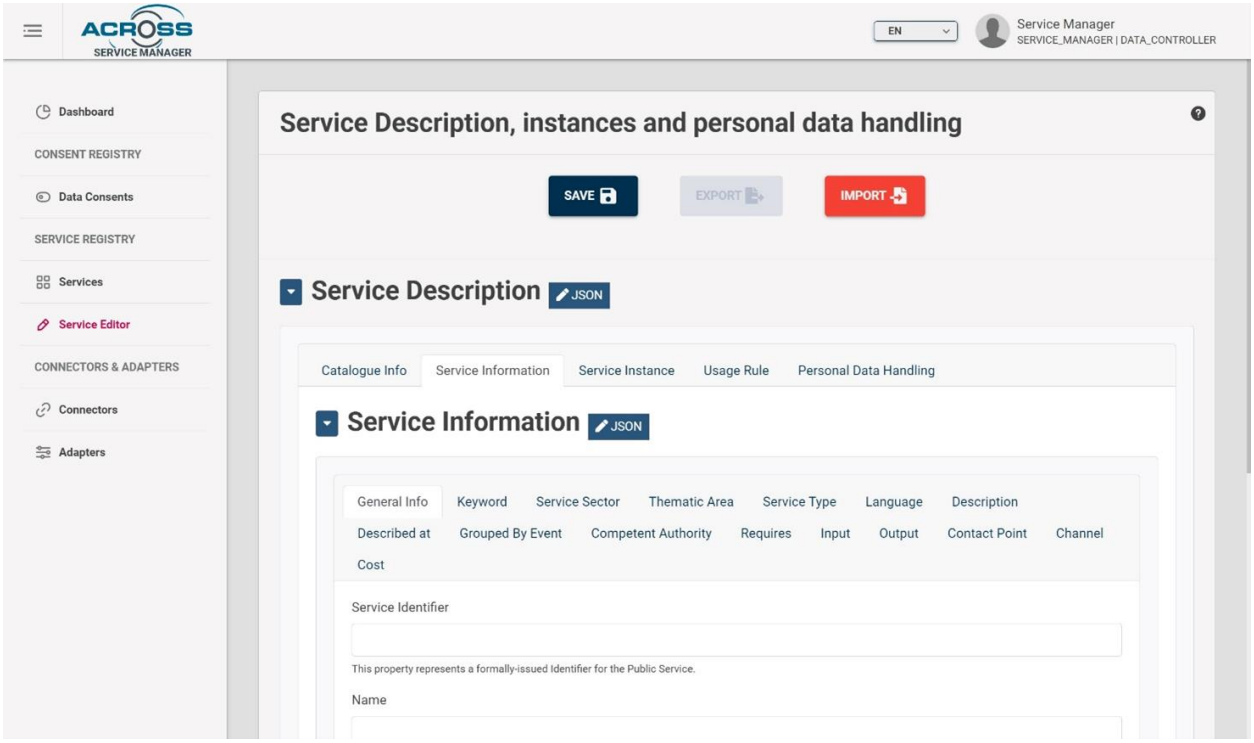

Figure 24 - Service Editor page

Each property is documented with a description and by clicking the "?" button in the in the top right-hand corner a guide is provided. The user can import existing standard service models or non-standard/legacy descriptions by selecting the suitable registered service model adapter (Figure 25).

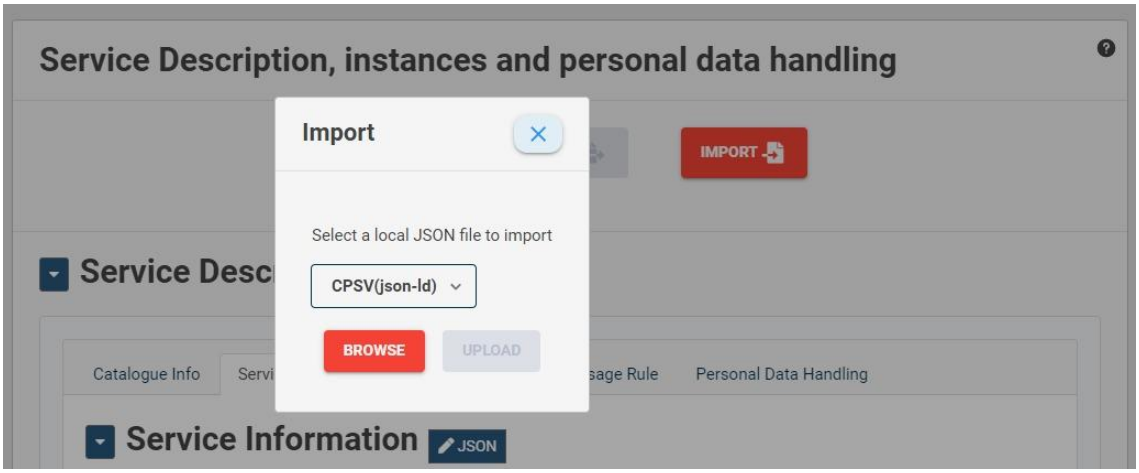

Figure 25 - Import dialog

The "Connectors" (Figure 26) and “Adapters” (Figure 27) sections provide quick information about the registered connectors and their status and logs. From these sections it is possible to edit their metadata or register new ones (Figure 28).

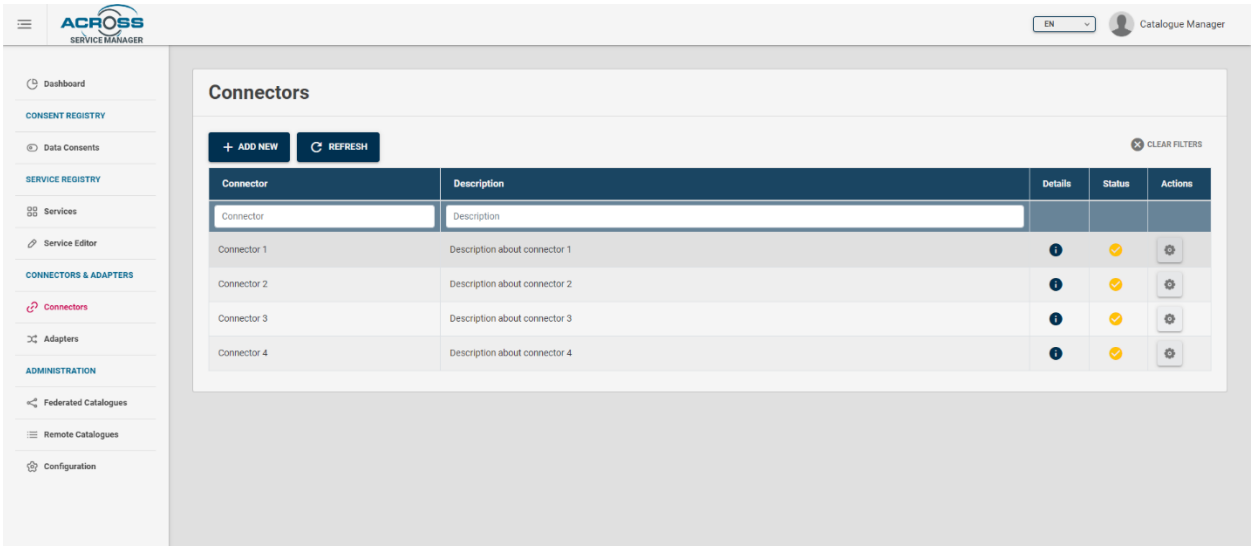

Figure 26 - Connectors list page

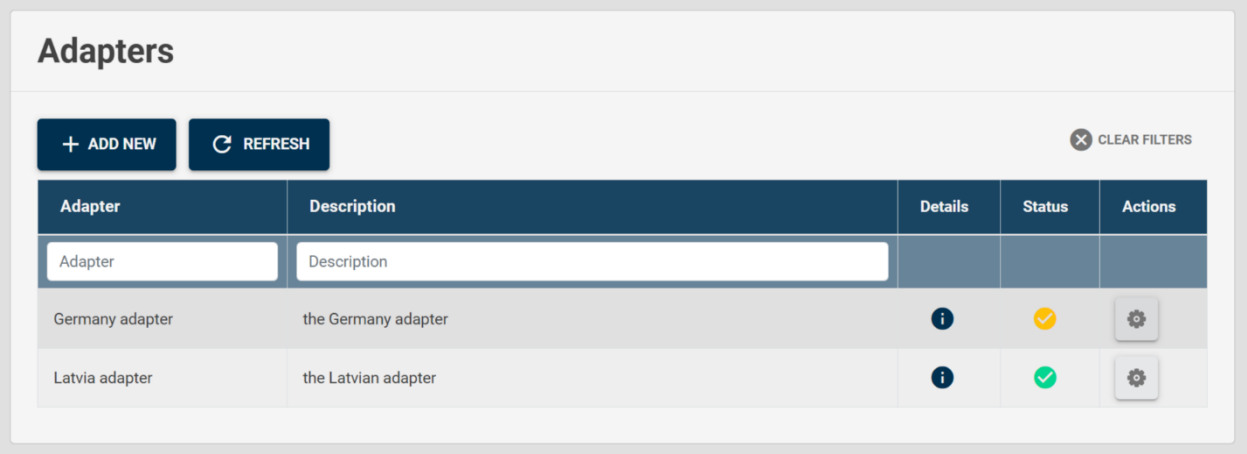

Figure 27 - Adapters list page

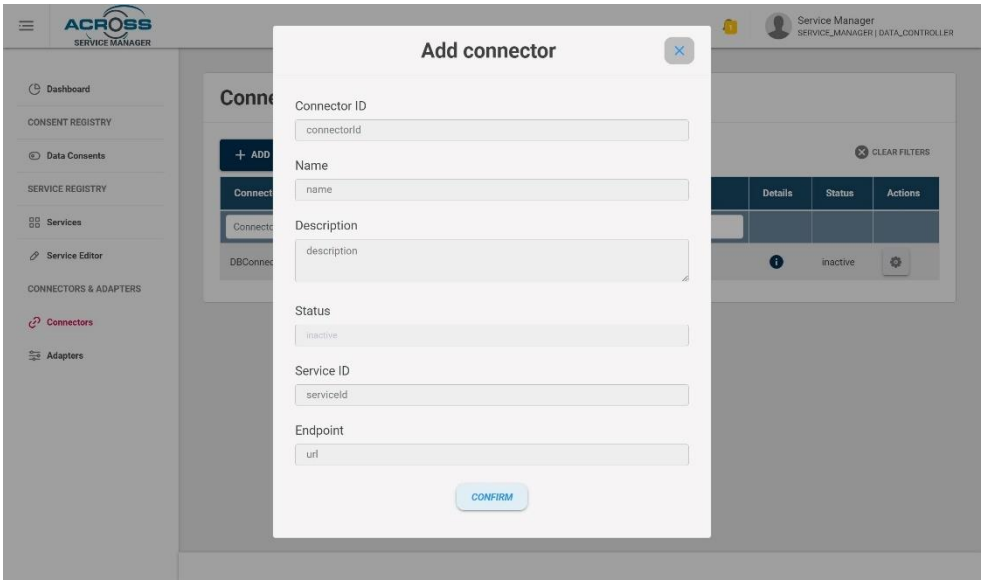

Figure 28 - Connector metadata entry

The "Data Consents" page provides, if the authenticated user has "data-controller" role, a registry of collected consents. It is a front-end client of the APIs provided by the Consent Manager component of Data Governance & Data Ownership layer of ACROSS platform.

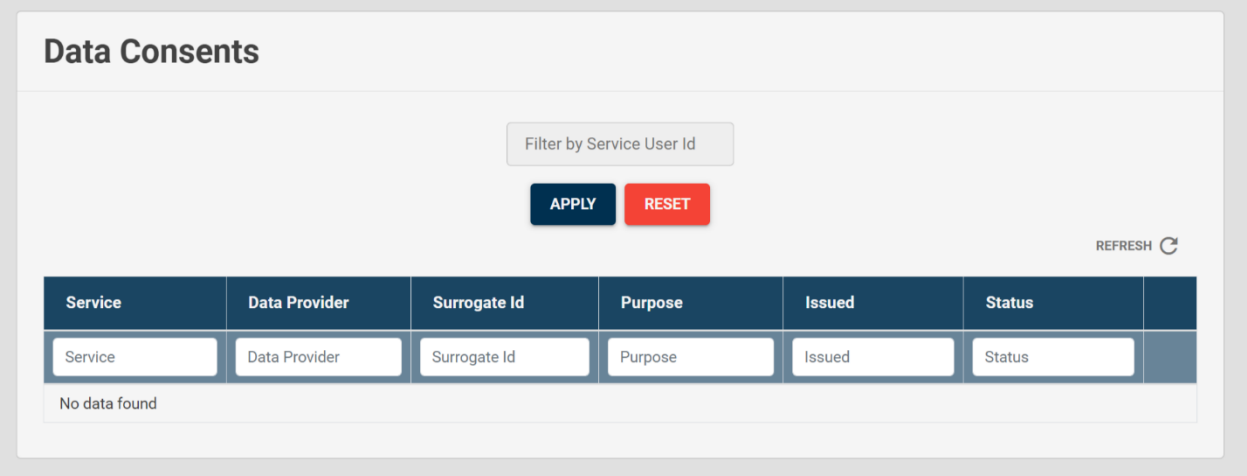

Figure 29 - Consent Registry page

Finally, the section "Dashboards" provides and extensible page of graphical dashboard cards providing some summaries about the inserted services (Figure 30) and available federated catalogues.

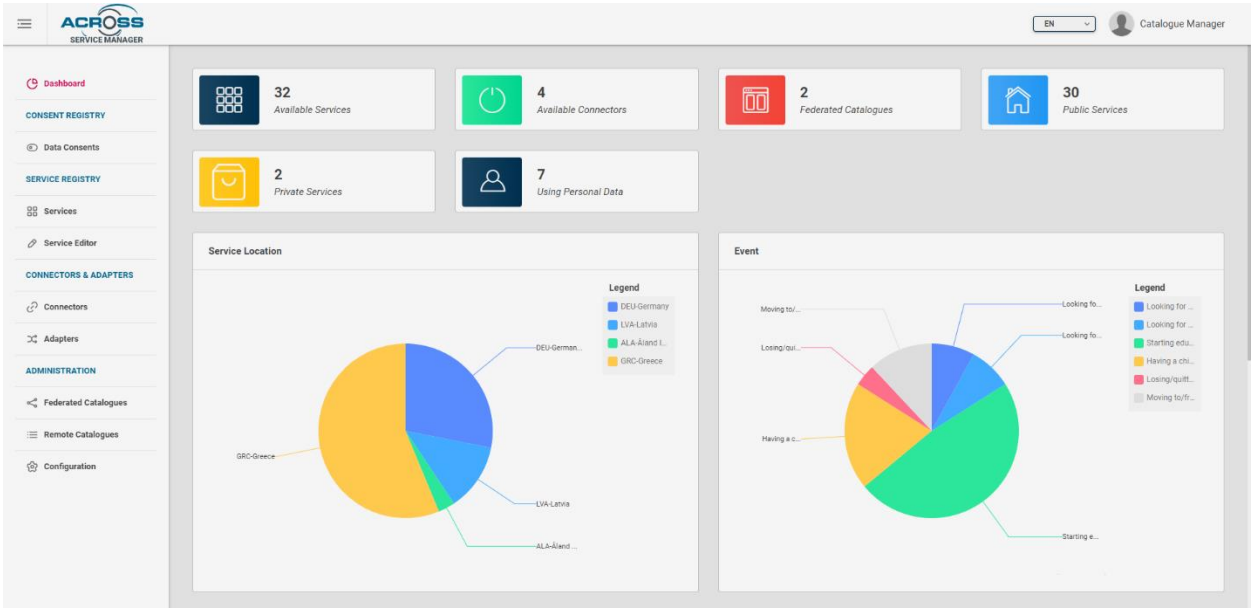

Figure 30 - Dashboard page

2.3.2 Federated Catalogues Management

Each Service Catalogue instance can be configured to manage “Federated” remote catalogues. Federating a catalogue means enabling the possibility of retrieving from the frontend or by means of API (see Annex II - Service Catalogue APIs) the list of its published services (Figure 31).

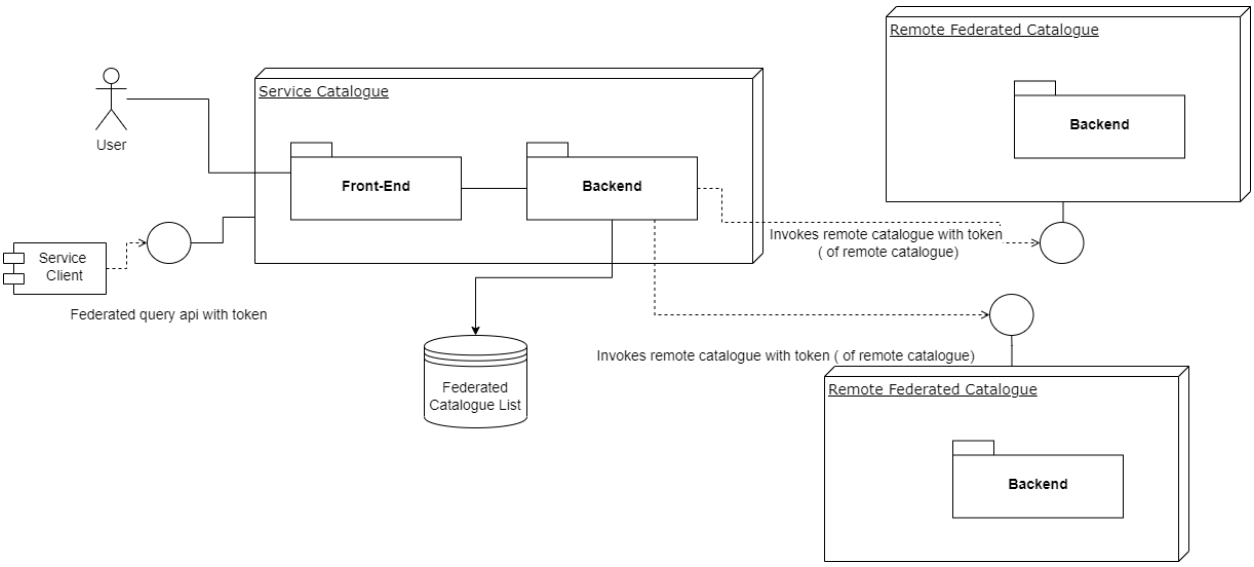

Figure 31- Federated Service Catalogue interaction

A federation is an action available only for a user with “Catalogue Manager” role. This type of user can view the “Administration Section” of the Service Catalogue.

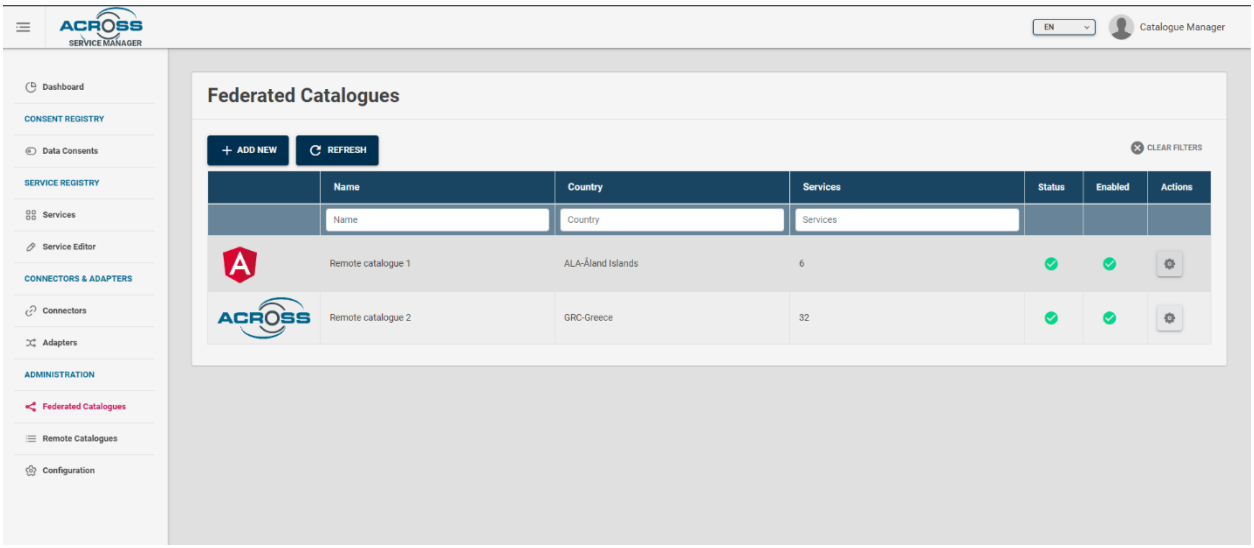

Figure 32 - Federated Catalogues Management section

Federating a remote catalogue can be done into ways:

1. Providing the needed information in Federated Catalogue section (Figure 32,Figure 33)
2. Import a list of available remote catalogues from a published JSON dataset or invoking an already federated remote catalogue (Figure 34)

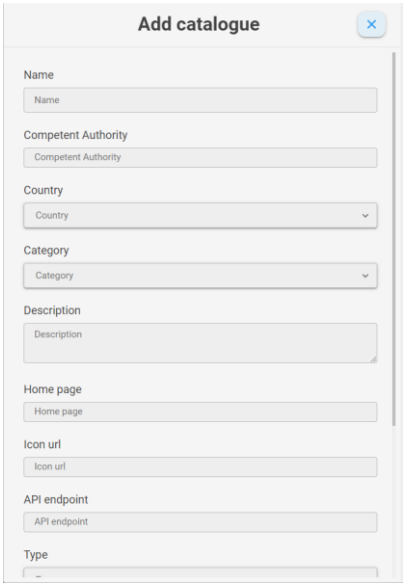

Figure 33 - Catalogue Federation form

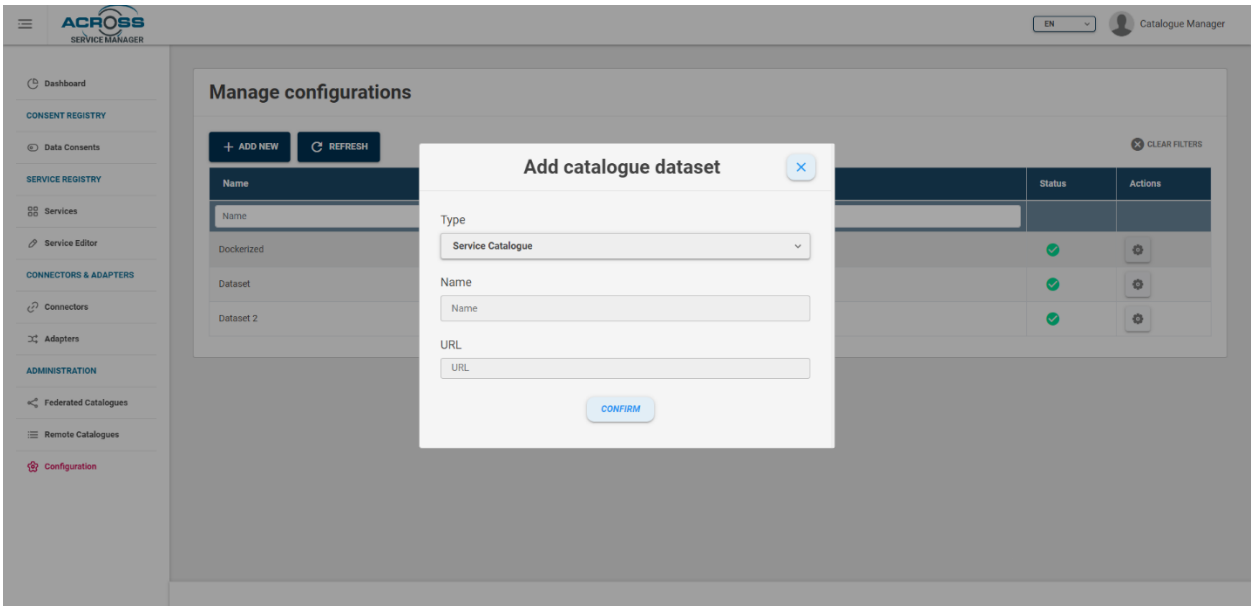

Figure 34 - Import form for remote catalogue datasets

2.4 Service Adaptation and baseline technologies

As defined in section 2, Service adapters are single instances of adaptation of services (public & private) for its use in ACROSS Platform useful to adapt heterogeneous service definitions into a common one and encapsulate generic communications-related logic required to use services and also to include logic that is quite specific for a given service. In particular connectors components make resources like database tables, stored procedures, domain objects, or files accessible to clients with a minimal amount of coding.

Each service registered in the Service Catalogue and included in User Journey processes should assure that the service invocation methods are compliant to the defined technical requirements of ACROSS Platform. If this technical requirement is not assured a ***service connector adapter*** is needed.

In the following section a description of technologies that can be adopted and extended to implement the custom implementation of a Service Adapter (see section 2.4.3). The common approach is to implement, regardless the adopted technology, the specific adapter/connector, in order to provide a complete set of standards to integrate existing legacy application systems, company-developed host applications, and third-party vendor applications.

Specifically, the Service connector adapter is a specific instance/implementation of several *Enterprise Integration Patterns[16]*:

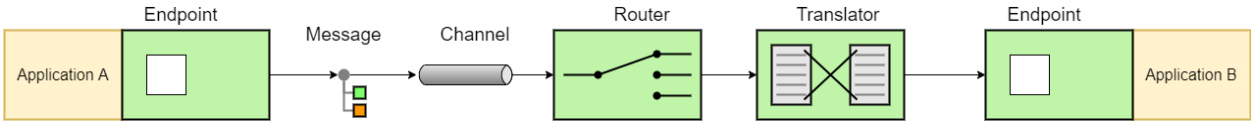

Figure 35 - Combination of several Enterprise Integration Patterns for Service Adaptation

By specializing this patterns combination for the specific need of service invocation methods compliancy, we have for a service adaptation (Figure 36):

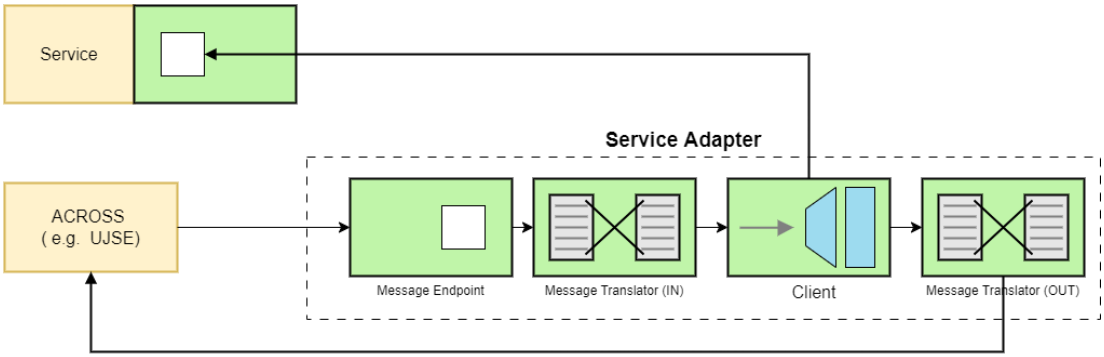

Figure 36- Service Adapter combination pattern for service invocation and adaptation in ACROSS Platform (e.g. from UJSE)

This adaptation basically includes the following steps:

1. Input adaptation: (if requested) input message is adapted
2. To "wrap" the Service invocation:
   - Authentication (if requested)
   - Invocation: the actual invocation to a service, via a specific client implementation
3. Output adaptation: (if requested) input message is adapted .

Steps 1 and 3 transforms the input/out message into the target message structure as requested in the ACROSS Platform.

A service adaptation is per service invocation method exposing an API endpoint to be invoked instead of the actual service endpoint by following the two optional actions:

1. Define a "Connector" in the Service Catalogue (see Figure 37 and section 2.3.1) and associate it to the service

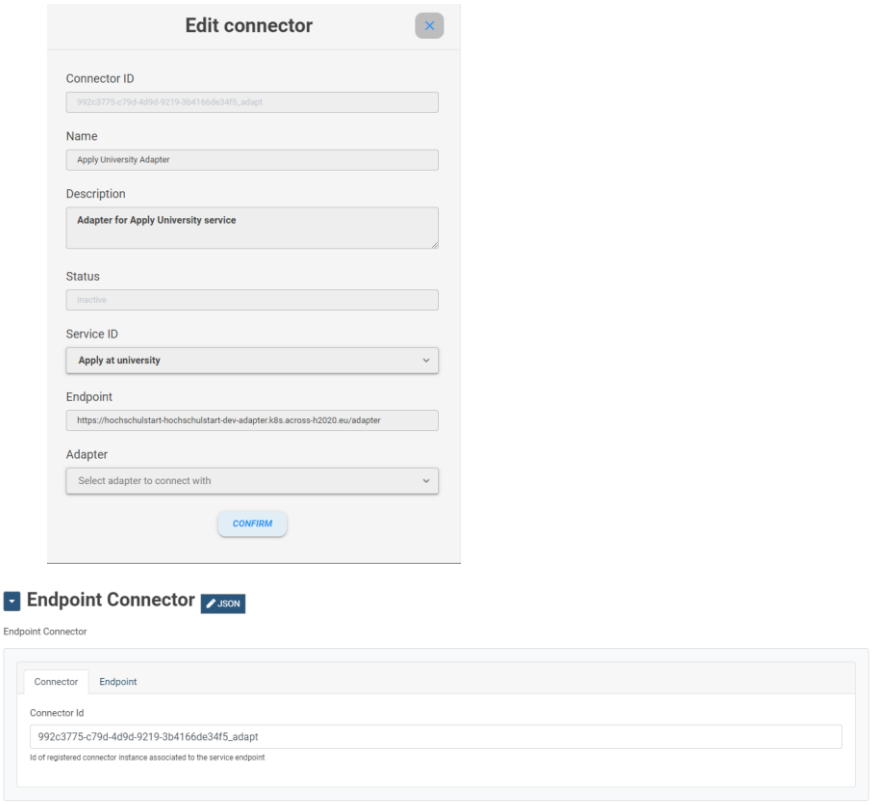

Figure 37 - Linking of a Connector to a specific registered Service

2. Or to put directly in *Endpoint.Url* the connector endpoint of the registered Service model description (see Figure 38 and section 2.3.1)

Figure 38 - Service model description metadata about endpoint invocation (Service Instance Class)

The choice of a specific technology will be influenced to the type of integration pattern to implement and to the technological background of the legacy system to integrate. Hence, the following technologies will be considered as some of the potential adoptions and the related metadata will be stored in the Service Catalogue to invoke the running instance of service adaptations (if needed) for the selected services included in the user journey processes.

2.4.1 Data Model Mapper

The Data Model Mapper (DMM) tool[10] enables to convert several file types (e.g. CSV, Json) to different defined data models. The files in input can contain either rows, JSON objects or other structured data, each of them representing an object to be mapped to an entity, according to the selected Data Model.

In particular, it performs following steps:

1. *Parsing:*
   - Parse input file, by converting it into a row/object stream.
2. *Streaming:*
   - Each row/object coming from the stream is converted to an intermediate object.
3. *Mapping:*
   - By using the input JSON Map, convert the intermediate object to an entity, according to a specific target Data Model.
4. *Validation and report:*
   - Validate resulting object against the JSON schema corresponding to a target Data Model.
   - Produce a report file with validated and unvalidated objects.

- *Writing: Context broker or File*
  - Validated objects can be sent to a configured context broker to the configured context broker[20] and/or to a local file.

The tool is developed in Node.js[21] and can be used in an "As a Service" Mode.

2.4.2 Data Connectors

Data Connectors basically covers four application integration approaches from Enterprise Integration Patterns . The four integration approaches include File Transfer, Shared Database, Messaging, and Remote Procedure Invocation:

- File Transfer: One application produces data files for others to consume, and vice versa.
- Shared Database: Applications can store and share the information in a common database.

[20] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/Context+Broker
[21] https://nodejs.org/

- Messaging: An application connects to a shared messaging system, exchanges data, and invokes behaviours using messages.
- Remote Procedure Invocation: An application exposes its APIs so that they can be invoked remotely by other applications.

In the following section some easy to use integration frameworks are provided: Apache Camel Integration, Spring Integration. They are available in a JVM environment and offer a standardized, domain-specific language to integrate applications. Anyway, the used approach and integration patterns does not constrain the use of these specific frameworks at the expense of others.

2.4.2.1 Apache Camel

Apache Camel[11] is an integration framework, which implements all Enterprise Integration Patterns for easy integration of different applications using the required patterns. We can use Java, Spring XML, Scala, or Groovy. Almost all technologies are available, for example, HTTP, FTP, JPA, RMI, JMS, JMX, LDAP, JMS, EJB, and many more. Apache Camel is also used with Apache ServiceMix, Apache ActiveMQ, and Apache CXF in service-oriented architecture projects.

An Apache Camel can be deployed in a web container like Tomcat, in a JEE Application Server, and as a standalone application and in general as a microservice as well.

The Camel architecture consists of three components − Integration Engine and Router, Processors, and Components. This is illustrated in the following Figure 39:

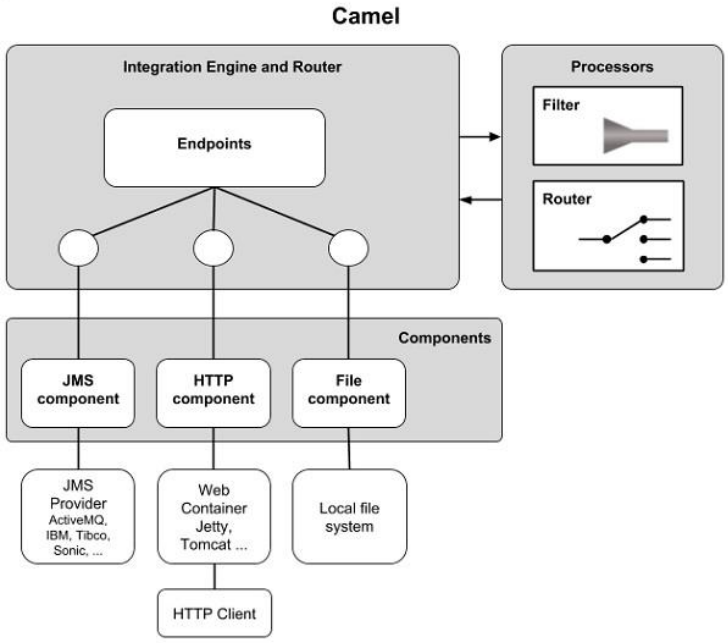

Figure 39 - Main components of Apache Camel

Apache Camel Architecture consists of a Camel Context that contains a collection of Component instances. A Component is a factory of Endpoint instances. We can explicitly configure Component instances in Java code, or they can be auto-discovered using URIs.

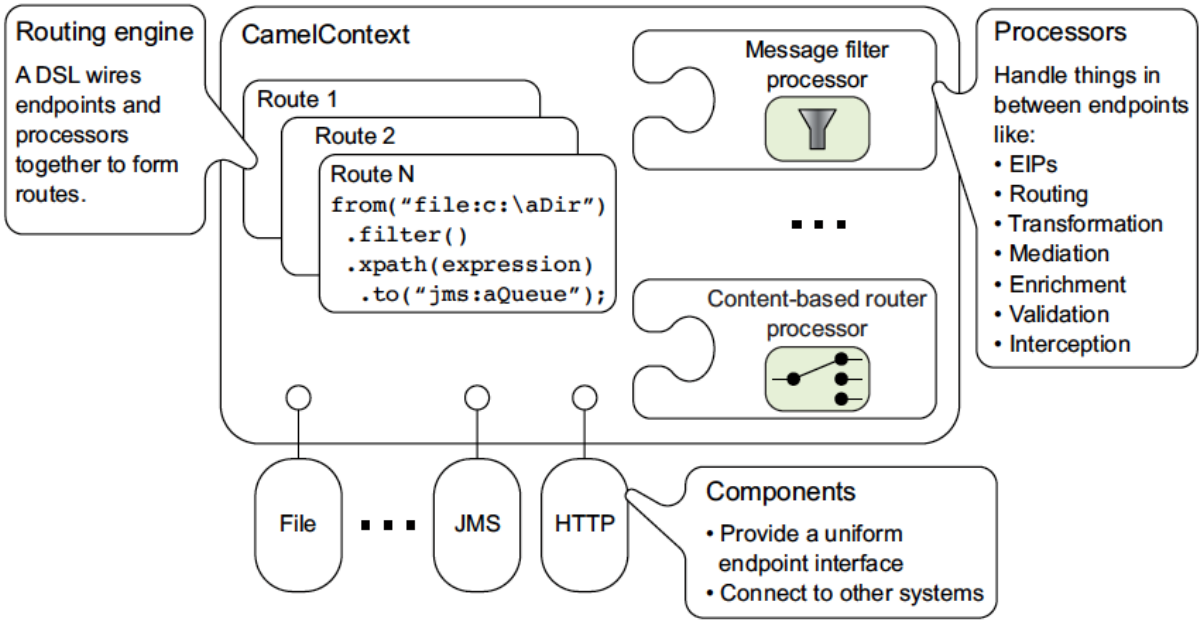

Figure 40 – Camel Context (figure from[11])

An Endpoint is either a URI or URL in a web application or a Destination in a JMS system. We can communicate with an endpoint either by sending messages to it or consuming messages from it. We can then create a Producer or Consumer on an Endpoint to exchange messages with it.

Overall, the architecture of Camel is simple. Camel Context represents the Camel runtime system, and it wires different concepts such as routes, components, or endpoints. Additionally, processors handle routing and transformations between parameters, while endpoints integrate disparate systems.

2.4.2.2 Spring Integration

Spring Integration[12] is an open source framework for enterprise application integration. It is built on top of Spring framework. Hence it is very easy to adopt Spring Integration in the projects which are already using the Spring framework.

In fact, Spring Integration provides an extension of the Spring programming model to support the well known Enterprise Integration Patterns. It enables lightweight messaging within Spring-based applications and supports integration with external systems through declarative adapters. Those adapters provide a higher level of abstraction over Spring's support for remoting, messaging, and scheduling.

As an extension of the Spring programming model, Spring Integration provides a wide variety of configuration options, including annotations, XML with namespace support, XML with generic "bean" elements, and direct usage of the underlying API. That API is based upon well-defined strategy interfaces and non-invasive, delegating adapters.

The basic concepts of a Spring Integration message-driven architecture are: message, message channel and message endpoint (Figure 41):

- A *message* is sent to an *endpoint*
- *Endpoints* are connected among them through *MessageChannels*
- An *endpoint* can receive *messages* from a *MessageChannel*

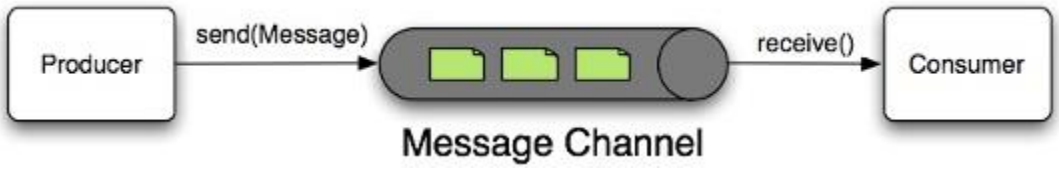

Figure 41 - Spring Integration message-driven architecture (figure from [12])

Below is a list of the available (see Figure 42 as example) message endpoints:

- *Channel adapter*: Connects the application to an external system (unidirectional).
- *Gateway*: Connects the application to an external system (bidirectional).
- *Service Activator*: Can invoke an operation on a service object.
- *Transformer*: Converts the content of a message.
- *Filter*: Determines if a message can continue its way to the output channel.
- *Router*: Decides to which channel the message will be sent.
- *Splitter*: Splits the message in several parts.
- *Aggregator*: Combines several messages into a single one.

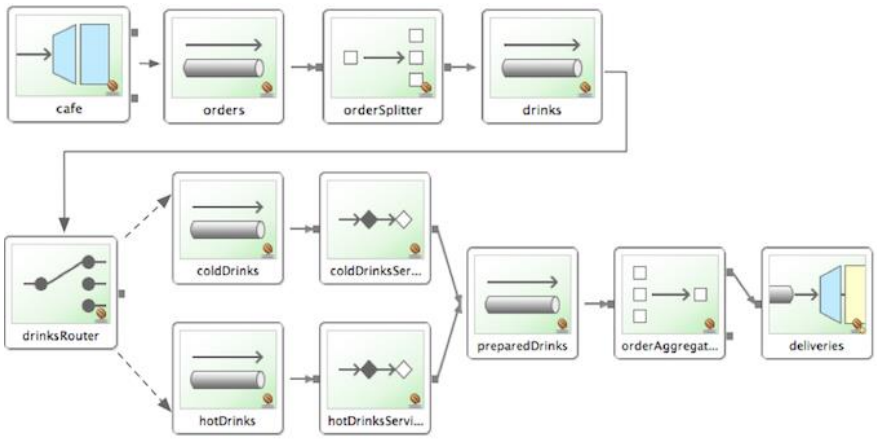

Figure 42 - Spring Integration sample from [12]

2.4.2.3 IDS Data Connector

In International Data Spaces (IDS) Reference Architecture Model (RAM)[13], the Connector is one of main technological building blocks. It is a dedicated software component allowing a Consumer and a Provider to exchange, share and process digital content. At the same time, the Connector ensures that the data sovereignty of the Data Owner is always guaranteed.

It is a configurable component, providing several system services enabling secure bidirectional communication, enforcement of content usage policies, system monitoring, and logging of content transactions for clearing purposes. The functional range of a generic Connector may be extended by custom software (Data Apps), allowing data processing, visualization, persistence, etc. The Connector provides metadata to the Data Consumer Connector as specified in the Connector's self-description. For example, technical interface description, authentication mechanism, exposed data sources, and associated data usage. Following the peer-to-peer network concept, the data is transferred between the Connectors of the Data Provider and the Data Consumer (Figure 43).

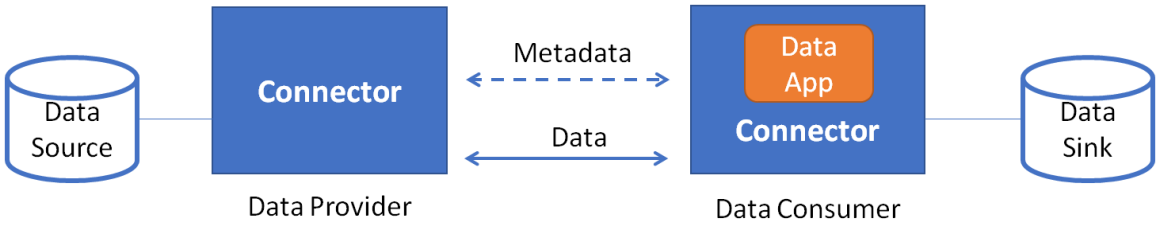

Figure 43 - IDS connector interactions

In the implementation of Service Adapter the open-source IDS connector TRUE (TRUsted Engineering) Connector[22] (Figure 44) will be leveraged in order to fit the specific Service Adapter needs. In particular the trivial Data APP application in order to adapt data on top of the Execution Core Container.

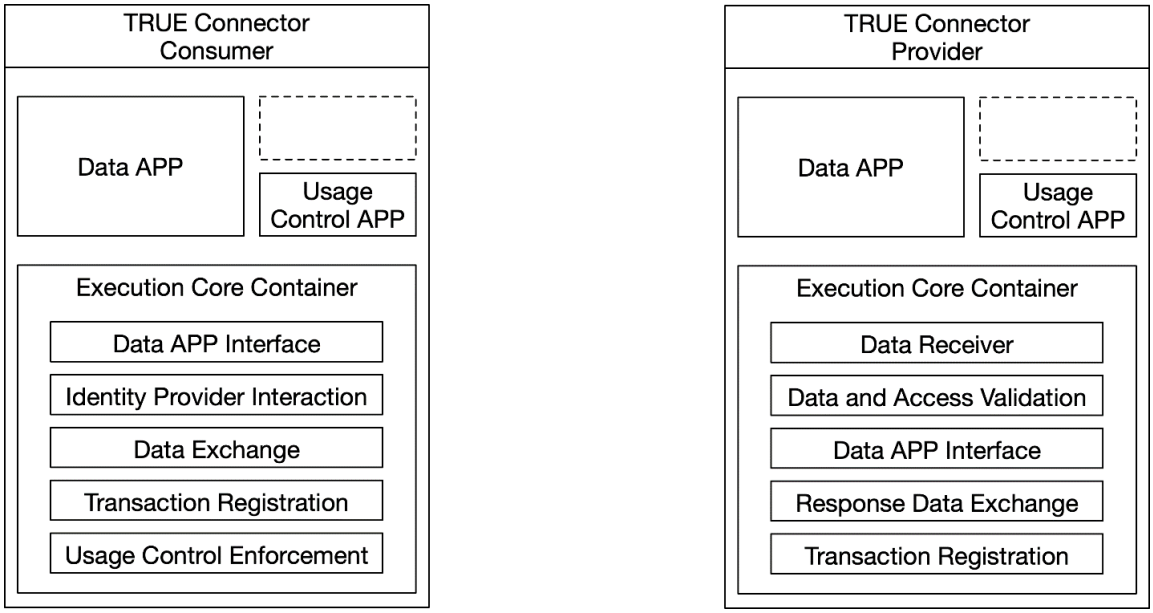

Figure 44 - TRUE Connector Architecture[14]

Considering the flows of interactions, the scenario depicted in Figure 45 shows how the Consumer connector accesses data from the Provider connector. Specifically, the Consumer receives a request to its Data App endpoint, it forwards the request to its internal Execution Core Container (ECC) which oversees establishing a secure communication with the Provider Execution Core Container to access the data. The Consumer's ECC receives the message from its Data APP, interacts with an external Identity Provider to retrieve the token of the Consumer and send the appropriate IDS message to the Provider's ECC using one of the provided communication

[22] https://github.com/Engineering-Research-and-Development/true-connector

channels. The Provider's ECC receives the message and validates the token against the Identity provider, then it retrieves the actual data from its Data APP (/data endpoint), and returns it to the Consumer's ECC who, finally, processes the response, applies the usage control policies and forward the data to the original requester.

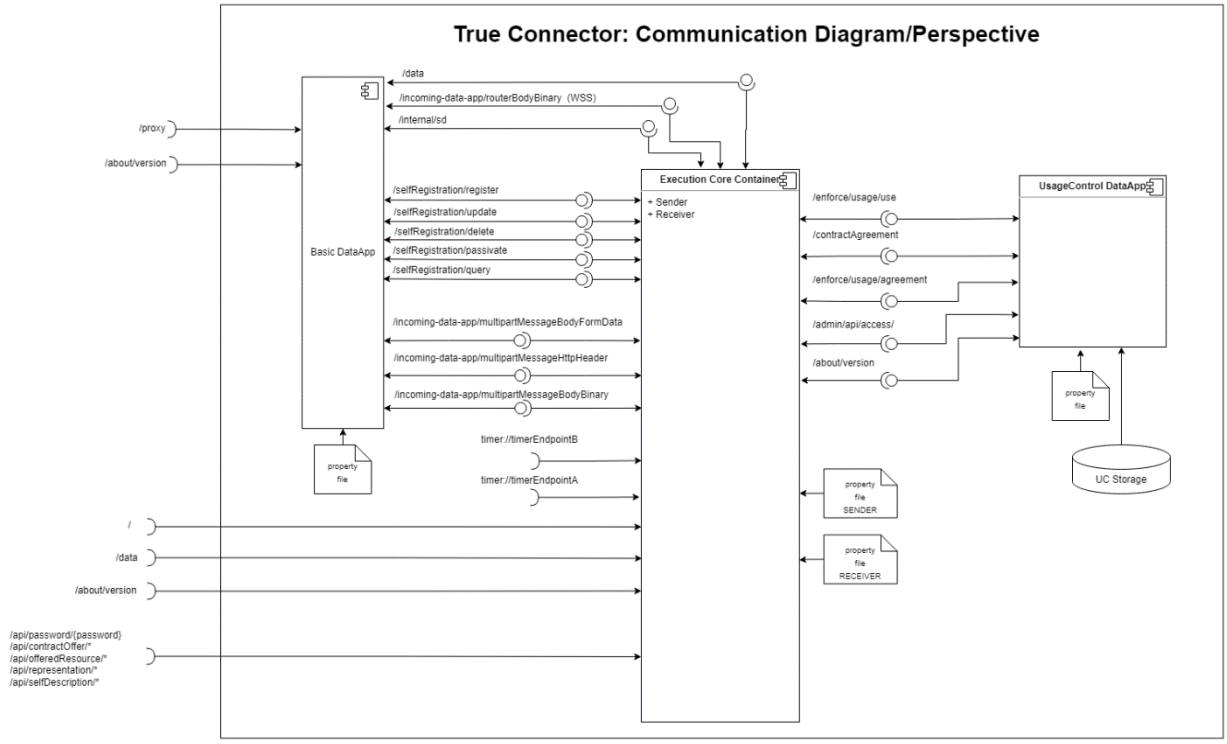

Figure 45 - TRUE Connector Communication Diagram[14]

2.4.3 Service Connector and adapter examples

The next sections provide several examples and suggestions on how to build a Service Connector and related data adaptation from a Microservice Skeleton built upon Spring Boot and Apache Camel that wraps the routes of service invocation and related IN/OUT adaptation in order to be invoked by ACROSS Platform components, in particular the User Journey Service Engine (UJSE) [17][18]

The example of Service Connector microservice used in the following sections can be found in [19].

2.4.3.1 Customization of Service Connector Skeleton

The customization of Service Connector Skeleton provides the following steps:

1. Define and implement in class `AdapterRoute` the rest() routes that wrap and adapt the remote service api to adapt. For each Route three Processors are defined to implement the ad-hoc service invocation and IN/OUT adaptation:

```
rest("/applications").description("POST REST Service adaptation").id("api-
route").produces(MediaType.APPLICATION_JSON)
            .consumes(MediaType.APPLICATION_JSON).bindingMode(RestBindingMode.auto)

            .post()
                .type(ApplicationPostRequest.class)
                .enableCORS(true)
                .outType(OutBean.class)
                .route()
                /* to invoke the POST input adaptation */
                .process(new Processor() {

                   …..

                })
                /* to invoke the remote service */
                .process(new Processor() {

                  ……

                    })
                /* to invoke the output adaptation */
                .process(new Processor() {

                 …….

                })
                .endRest();
```

2. Create a class that implements Adapter interfaces:

```
public interface Adapter {

    public abstract ArrayList<OutBean> adaptOut(Object body);
    public abstract Object adaptIn(Object body);


}
```

That implements the needed IN/OUT message adaptation (see in the example the class `AdapterApplicationImpl`)

3. Implement the Service Client class to invoke the actual service in the second step (see as example the class `ClientService`) together the rest messages pojos as defined as example in

the swagger definition of service to be invoked. The implemented service connector can be deployed and invoked as microservice :

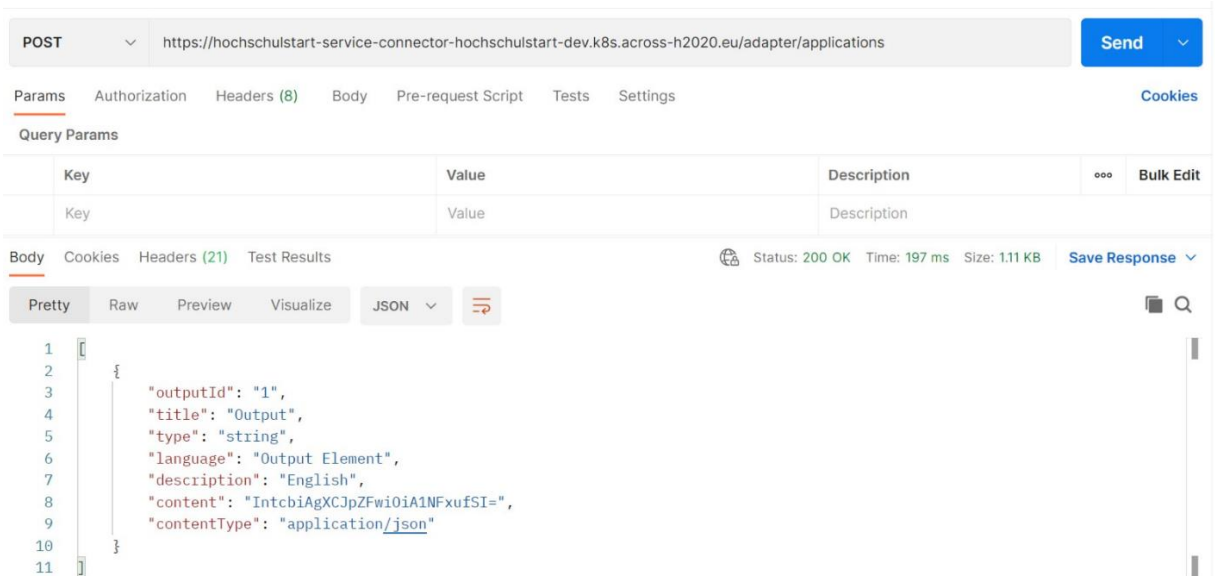

Figure 46 - POST invocation of the Service endpoint, proxied by the Service Adapter

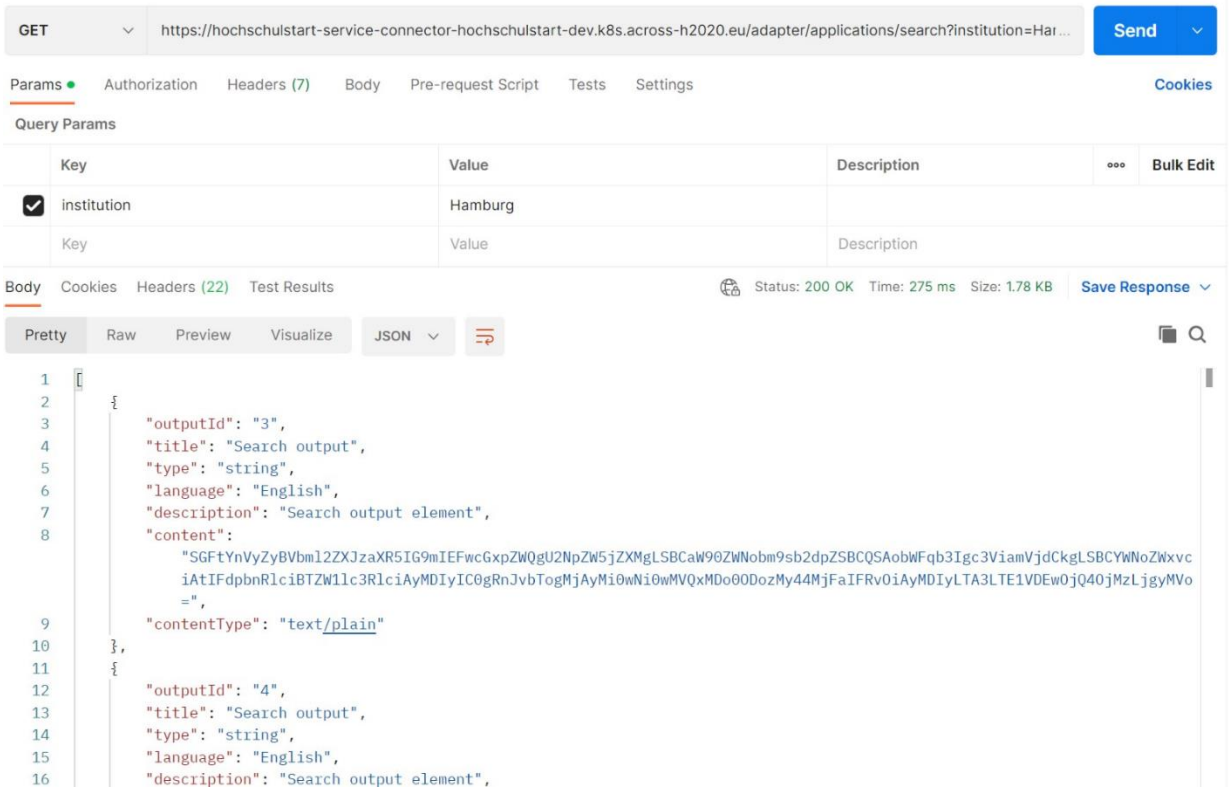

Figure 47 - GET invocation of the Service endpoint, proxied by the Service Adapter

2.4.3.2 Customization of Service Connector Skeleton by using a Data Model Mapper

As introduced in section 2.4.1 the Data Model Mapper (DMM) is a tool that can be used to transform a source file (csv, json or geojson) into a json file, with a different schema structure. It can be used to implement the adaptOut method in the Adapter class mentioned before. The functionality offered by the tools lets to implement the adaptOut method of the Adapter interface as described in the previous section 2.4.3.1 “Customization of Service Connector Skeleton”. In particular, the methods in the adapter class that implements the adapter interface (step 2 in 2.4.3.1) must call, in an as service mode, the DMM API to retrieve the adapted output instead of doing a hardcoded adaptation.

The specific data adaptation must be firstly registered in the DMM by registering the mapping configuration and a “datamodel” schema representing the output model:

```
{
    "name": "search",
    "id": "search",
    "map": {
        "outputId": [
            "static:",
            "id"
        ],
        "title": "static:Search output #",
        "type": "static:string",
        "language": "static:English",
        "description": [
            "static:Institution: ",
            "institution"
        ],
        "content": "encode:base64:[\"institution\", \"static: - \", \"name\", \"static: - \",
\"degree\", \"static: - \", \"semester\", \"static: - From: \", \"applicationPeriodFrom\", \"s
tatic: To: \", \"applicationPeriodTo\"]",
        "contentType": "static:text/plain",
        "entitySourceId": "static:search",
        "targetDataModel": "DataModelTemp"
    },
    "dataModel": {
        "schema": "http://json-schema.org/schema#",
        "type": "object",
        "title": "DataModelTemp",
        "properties": {
            "outputId": {
                "type": "string"
            },
            "title": {
                "type": "string"
            },
            "type": {
                "type": "string"
            },
```

```
            "language": {
                "type": "string"
            },
            "description": {
                "type": "string"
            },
            "content": {
                "type": "string"
            },
            "contentType": {
                "type": "string"
            }
        }
    }
}
```

```
{
    "name": "application",
    "id": "application",
    "map": {
        "outputId": "static:Id",
        "title": "static:Application Identifier",
        "type": "static:string",
        "language": "static:English",
        "description": "static:Identifier obtained as a result of the application just perform
ed.",
        "content": "encode:base64:[\"id\"]",
        "contentType": "static:text/plain",
        "targetDataModel": "DataModelTemp"
    },
    "dataModel": {
        "schema": "http://json-schema.org/schema#",
        "type": "object",
        "title": "DataModelTemp",
        "properties": {
            "outputId": {
                "type": "string"
            },
            "title": {
                "type": "string"
            },
            "type": {
                "type": "string"
            },
            "language": {
                "type": "string"
            },
            "description": {
                "type": "string"
            },
            "content": {
                "type": "string"
            },
            "contentType": {
```

```
                "type": "string"
            }
        }
    }
}
```

The resulted adapters configuration (ID and DMM endpoint) will be used to configure the service connector in the adaptation classes. This information can be registered in the Service Catalogue by defining a connector and an adapter class (see section 2.3.1):

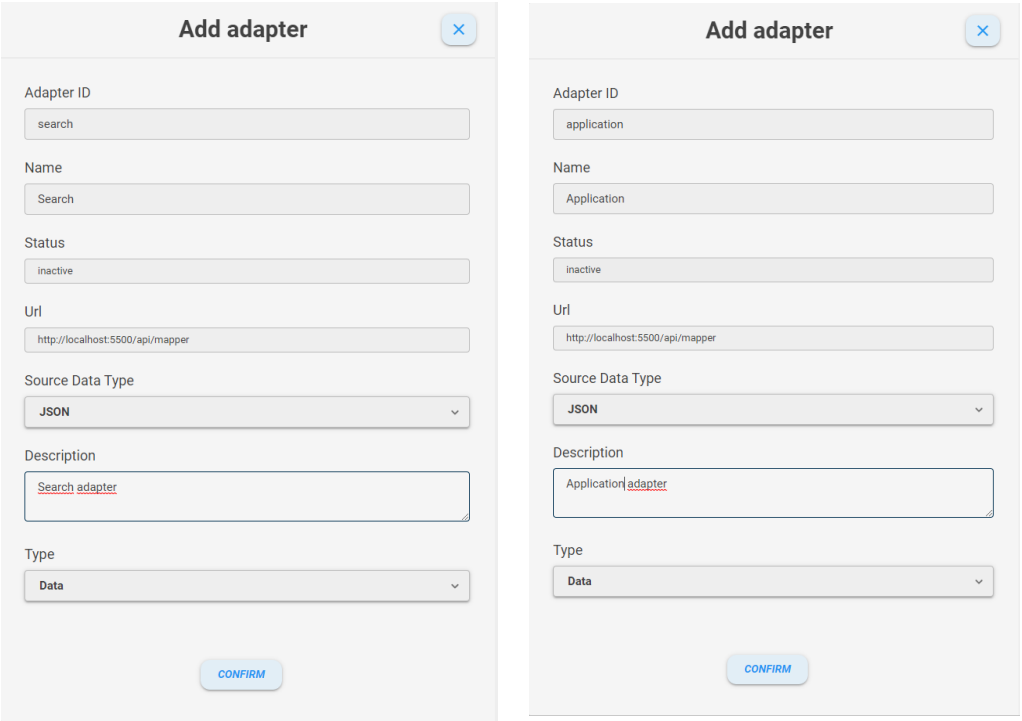

Figure 48 - Registration of the adapters in the Service Catalogue

The service connector must be configured (in `application.properties`) by providing the information about the data model mapper API and the references of the specific mappers:

```
#data model mapper url
remoteAdapterURL=http://localhost:5500/api/map/transform
#remote adapter id, defined in the Service Catalogue instance
remoteAdapterApplicationID=application
remoteAdapterSearchID=search
```

So that the `adaptOut` method of Adapter implementation class should be like the following:

```
public ArrayList<OutBean> adaptOut(Object body) throws IOException, InterruptedException {

    ObjectMapper mapper = new ObjectMapper();

    HttpClient client = HttpClient.newBuilder()
            .followRedirects(HttpClient.Redirect.NORMAL)
            .build();

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(remoteAdapterURL))
            .POST(BodyPublishers.ofString("{\"sourceDataType\" : \"json\",\"sourceData\": "+body+", \"adapterID\" : \""+remoteAdapterApplicationID+"\"}"))
            .setHeader(name:"Content-Type", value:"application/json")
            .build();

    HttpResponse<String> response;// = new HttpResponse<ArrayList<OutBean>>();

    response = client.send(request, HttpResponse.BodyHandlers.ofString());

    ArrayList<OutBean> mapped = mapper.readValue(response.body(), valueType:ArrayList.class);

    return mapped;
}
```

Like the previous section the invocation of the Service Connector will produce the same output for GET and POST service invocation (Figure 46 and Figure 47).

2.4.3.3 Adoption of IDS Connector with Service Connector Skeleton for trusted service connection

As described in the section 2.4.2.3 an IDS connector allows a Consumer and a Provider to exchange, share and process digital content. In Service Adapter context the adoption and integration of IDS connector introduce a trusted layer, from an operation and governance point of view (as stated by the IDS specifications[13]) between the Service Connector, acting as a proxy invocation from the ACROSS platform and the actual external service (Figure 49)

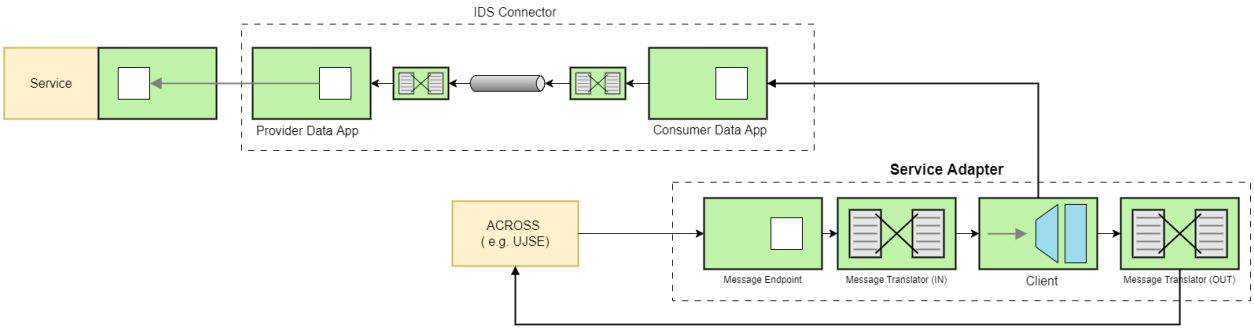

Figure 49 - Service Adapter customization to invoke Data App component of IDS connector as extension of Service Adapter integration pattern of Figure 36.

Operatively, all the steps described in the previous two sections apply, in addition a customization of “Data App” of the IDS TRUE Connector solution[14] is necessary in the Provider side (the invoked service) while in the Consumer side, the Service Connector client will be in charge to invoke the Consumer Data App for a trusted service invocation through IDS Data Connectors.

In particular, in Provider Data App `DataAppMessageHandler` interface should be implemented and instantiated to invoke the actual service, in a similar way as done in step 3 of the previous section in 2.4.3.1. The Service Connector will be in charge to invoke `/proxy` endpoint (see Figure 45) of Consumer Data App

```
DataApp URL: https://localhost:8084/proxy
"Forward-To": "https://ecc-provider:8889/data",
```

where “Forward-To” is the reference of IDS provider data connector to forward the request:

```
{
    "multipart": "form",
    "Forward-To": "https://ecc-provider:8889/data",
    "messageType": "ArtifactRequestMessage" ,
    "requestedArtifact": "http://w3id.org/engrd/connector/artifact/1" ,
    "payload" : {
        <specific payload of the service request to forward to the provider
connector>
     }
}
```

Further information on how customizing Data App component can be found on [20].

3 Conclusions

This report has documented the overall result of activities performed in Task 4.2 " Public & Private sector offerings management tool" considering the relations with the other activities involved in the definition and implementation of ACROSS Platform and related components. The report has described the architecture of Data harmonization and connectors layer to support service adaptation and how each component is involved in the flow of according to the type of adaptation performed. Service catalogue, related service model, components and functionality have been described in line with the second step evolution. The report provides some references of technologies available and related examples to build ad hoc services adapters to be published and used in ACROSS Platform.

The service model has the goal to collect the multi-view of information of a service to be used in ACROSS Platform for the delivery of a user journey service. This model will continue to evolve accordingly to the evolution of ACROSS Platform components. This evolution will contribute into the continuous extension of Service Catalogue functionality and its internal model. The layered design and implementation of Service Catalogue and the adopted technologies allow the service model evolution with a minimal amount of effort to address future verticalization requirements.

The documented solution is available in a public repository with an open-source licence[6][19]

4 References

[1] ACROSS, D5.2 - System Architecture & Implementation Plan – Final

[2] Core Public Service Vocabulary Application Profile 3.1.0 (https://joinup.ec.europa.eu/collection/semic-support-centre/solution/core-public-service-vocabulary-application-profile/release/310)

[3] ACROSS, D3.1 - Design of the ACROSS Data Governance framework for data sovereignty – Initial

[4] Regulation (EU) 2018/1724 of the European Parliament and of the Council of 2 October 2018 establishing a single digital gateway to provide access to information, to procedures and to assistance and problem-solving services and amending Regulation (EU) No 1024/2012 (https://eur-lex.europa.eu/eli/reg/2018/1724/oj)

[5] Data Privacy Vocabulary (DPV) https://dpvcg.github.io/dpv/

[6] Service Catalogue source code: https://github.com/OPSILab/Service-Catalogue

[7] Service Catalogue documentation: https://service-catalogue.readthedocs.io/en/latest/

[8] APIs for CPSV-AP based Catalogue of Services: https://joinup.ec.europa.eu/sites/default/files/news/2019-09/ISA2_APIs%20for%20CPSV-AP%20based%20Catalogue%20of%20Services.pdf

[9] Usage Control in IDS https://internationaldataspaces.org/wp-content/uploads/dlm_uploads/IDSA-Position-Paper-Usage-Control-in-the-IDS-V3..pdf

[10] Data Model Mapper : https://github.com/OPSILab/data-model-mapper

[11] APACHE CAMEL: https://camel.apache.org/manual/index.html

[12] SPRING Integration: https://docs.spring.io/spring-integration/docs/6.0.0/reference/pdf/spring-integration-reference.pdf

[13] IDS Reference architecture Model v3 - *https://internationaldataspaces.org/wp-content/uploads/IDS-Reference-Architecture-Model-3.0-2019.pdf*

[14] TRUE Connector:https://github.com/Engineering-Research-and-Development/true-connector

[15] ACROSS, D6.2 Use Case Evaluation and Impact Assessment -Initial

[16] Enterprise Integration Patterns https://www.enterpriseintegrationpatterns.com/

[17] ACROSS, D3.2 - Design of the ACROSS Data Governance framework for data sovereignty- Final

[18] ACROSS, D3.5 -Implementation of the ACROSS Data Governance framework for data sovereignty – Final

[19] Service Connector App: https://github.com/OPSILab/service-connector-app

[20] Customizing Base Data App https://github.com/Engineering-Research-and-Development/true-connector-basic_data_app#customizingdataapp

5 Annex I - ACROSS Service Model

The following sections provides an overview description of the ACROSS Service Model. Detailed description can be found in Service Model Documentation[7].

5.1 Service Model class diagram

The following Figure 50 provides the complete class diagram structure of Service Model adopted in Service Catalogue. Each class in the class diagram is described in the following sections

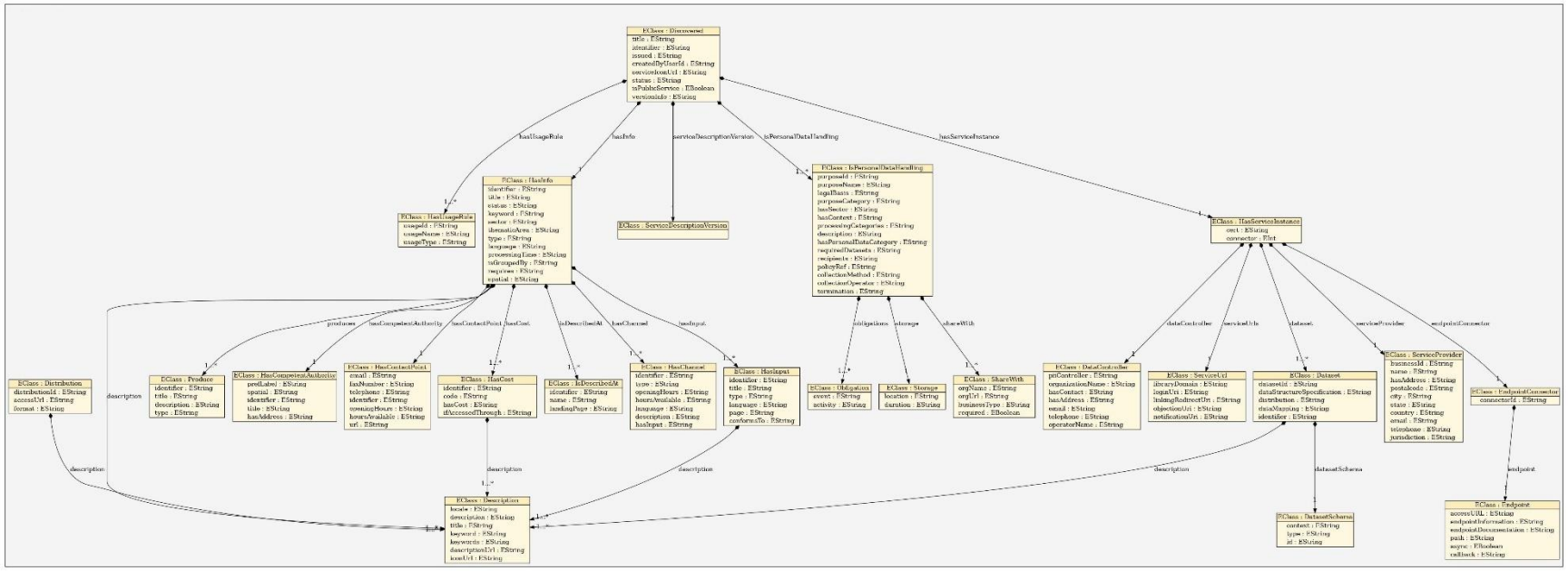

Figure 50 - Complete class diagram of Service Model

5.2 Service Basic Info

Each service to be registered in the Service Catalogue must provide some basic information.

Table 1 Basic Info properties of Service Model class

Property	Type	Description
title	String (1..1)	Service Name
identifier	String (1..1)	Id of service or service URI if exists. This identifier will be used by the Service Catalogue to identify it and could be the same identifier provided in the information section 5.3.
issued	String (0..1)	When Service entry was created (system log data)
createdByUserId	String (0..1)	User Id (if any)of Service Editor
serviceDescriptionVersion	String (0..1)	Service description version number
serviceIconUrl	String (0..1)	URL pointing to service's icon (if available)
status	String (0..1)	Status of Service ["Completed", "Deprecated", "UnderDevelopment", "WithDrawn"]
isPublicService	Boolean (1..1)	If service is public or not
hasInfo	Object (1..1)	Object describing Service information section. See updated description on readthedocs Service Model Documentation[7]
hasServiceInstance	Object (1..1)	Object describing Service information section. See updated description on readthedocs Service Model Documentation[7]

isPersonalDataHandling	Object (1..n)	Object describing Personal data handling. See updated description on readthedocs Service Model Documentation[7]
hasUsageRule	Object (1..n)	Object describing Service usage Rules. See updated description on readthedocs Service Model Documentation[7]

5.3 Service Model JSON Schema

In the following the first level of JSON schema adopted by the Service Catalogue in line with the initial Service Model.

```
{
  "title": "ServiceModel",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "title": {
      "type": "string",
      "title": "Service Name",
      "description": "Service Name"
    },
    "identifier": {
      "type": "string",
      "title": "Service Identifier",
      "description": "Service URI if exists"
    },
    "issued": {
      "type": "string",
      "title": "Issued at",
      "description": "Timestamp of the Service creation"
    },
    "createdByUserId": {
      "type": "string",
      "title": "Created by User Id",
      "description": "User Id of Service Editor (e.g. Data Controller)."
    },
    "versionInfo": {
      "type": "string",
      "title": "Service Description version",
      "description": "Service description version number.",
    },
    "serviceIconUrl": {
      "type": "string",
      "title": "Service Icon Url",
      "description": "URL pointing to Service Icon file"
```

```
    },
    "status": {
      "type": "string",
      "title": "Service Description status",
      "description": "Status of Service Description (Allowed values: *Completed*, *Deprecate
d*, *UnderDevelopment*, *Withdrawn*)",
      "default": "UnderDevelopment",
      "enum": ["Completed", "Deprecated", "UnderDevelopment", "WithDrawn"],
      "options": { "enum_titles": ["Completed", "Deprecated", "Under Development", "Withdraw
n"] }
    },
    "isPublicService": {
      "type": "boolean",
      "title": "Public Service",
      "description": "if public service or not",
      "default": "true"
    },

    "hasInfo": {
      "$ref": "./service-cpsv-entry.json"
    },
    "hasServiceInstance": {
      "$ref": "./service_instance.json"
    },
    "hasUsageRule": {
      "type": "array",
      "title": "Usage Rule",
      "format": "tabs",
      "description": "It collects contract and usage rules for data sharing",
      "items": {
        "$ref": "./usage_rule.json"
      }
    },
    "isPersonalDataHandling": {
      "type": "array",
      "title": "Personal Data Handling",
      "format": "tabs",
      "description": "It collects the different legal basis and requirements for personal da
ta processing according to EU data protection Rules (Art. 6 GDPR). It describes describe dif
ferent situations where a company or an organisation is allowed to collect or reuse your per
sonal information: contract, legal obligation, vital interest, public interest, legitimate i
nterest and consent",
      "items": {
        "$ref": "./isPersonalDataHandling.json"
      }
    }
  }
}
```

5.4 Service Model JSON-LD Context

The following context definitions are used to export Service Description into the JSON-LD semantic model.

```
{
    "@context": {
        "isTypeOf": "@type",
        "id": "@id",
        "acr": "https://across-h2020.eu/ns/serviceModel",
        "adms": "http://www.w3.org/ns/adms#",
        "cpsv": "http://purl.org/vocab/cpsv#",
        "cv": "http://data.europa.eu/m8g/",
        "dcat": "http://www.w3.org/ns/dcat#",
        "dct": "http://purl.org/dc/terms/",
        "dpv": "https://w3.org/ns/dpv",
        "eli": "http://data.europa.eu/eli/ontology#",
        "foaf": "http://xmlns.com/foaf/0.1/",
        "ids": "https://w3id.org/idsa/core/",
        "locn": "http://www.w3.org/ns/locn#",
        "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
        "schema": "https://schema.org/",
        "skos": "http://www.w3.org/2004/02/skos/core#",
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "owl": "http://www.w3.org/2002/07/owl",
        "prov": "https://www.w3.org/ns/prov#",
        "vcard": "http://www.w3.org/2006/vcard/ns#",
        "Agent": "dct:Agent",
        "BusinessEvent": "cv:BusinessEvent",
        "Channel": "cv:Channel",
        "Collection": "skos:Collection",
        "Concept": "skos:Concept",
        "ContactPoint": "schema:ContactPoint",
        "Cost": "cv:Cost",
        "CriterionRequirement": "cv:CriterionRequirement",
        "Event": "cv:Event",
        "Evidence": "cv:Evidence",
        "LegalResource": "eli:LegalResource",
        "LifeEvent": "cv:LifeEvent",
        "LinguisticSystem": "dct:LinguisticSystem",
        "Location": "dct:Location",
        "OpeningHoursSpecification": "schema:OpeningHoursSpecification",
        "Output": "cv:Output",
        "Participation": "cv:Participation",
        "PublicOrganisation": "cv:PublicOrganisation",
        "PublicService": "cpsv:PublicService",
        "PublicServiceDataset": "cv:PublicServiceDataset",
        "Rule": "cpsv:Rule",
        "accessURL": {
            "@id": "dcat:accessURL",
            "@type": "@id"
```

```
        },
        "endpointInformation": {
            "@id": "ids:endpointInformation",
            "@type": "rdfs:Literal"
        },
        "endpointDocumentation": {
            "@id": "ids:endpointDocumentation",
            "@type": "rdfs:Literal"
        },
        "path": {
            "@id": "ids:path",
            "@type": "rdfs:Literal"
        },
        "createdByUserId": {
            "@id": "acr:createdByUserId",
            "@type": "rdfs:Literal"
        },
        "currency": {
            "@id": "cv:currency",
            "@type": "@id"
        },
        "connector": {
            "@id": "ids:connector",
            "@type": "@id"
        },
        "connectorEndpoint": {
            "@id": "ids:hasDefaultEndpoint",
            "@type": "@id"
        },
        "description": {
            "@id": "dct:description",
            "@type": "rdfs:Literal"
        },
        "email": {
            "@id": "schema:email",
            "@type": "rdfs:Literal"
        },
        "fax": {
            "@id": "schema:faxNumber",
            "@type": "rdfs:Literal"
        },
        "follows": {
            "@id": "cpsv:follows",
            "@type": "@id"
        },
        "follows": {
            "@id": "cpsv:follows",
            "@type": "@id"
        },
        "format": {
            "@id": "dct:format",
```

```
            "@type": "@id"
        },
        "hasAddress": {
            "@id": "cv:hasAddress",
            "@type": "@id"
        },
        "hasChannel": {
            "@id": "cv:hasChannel",
            "@type": "@id"
        },
        "hasCompetentAuthority": {
            "@id": "cv:hasCompetentAuthority",
            "@type": "@id"
        },
        "hasContactPoint": {
            "@id": "cv:hasContactPoint",
            "@type": "@id"
        },
        "hasCost": {
            "@id": "cv:hasCost",
            "@type": "@id"
        },
        "hasCriterion": {
            "@id": "cv:hasCriterion",
            "@type": "@id"
        },
        "hasDataController": {
            "@id": "dpv:hasDataController",
            "@type": "@id"
        },
        "hasInput": {
            "@id": "cpsv:hasInput",
            "@type": "@id"
        },
        "hasInfo": {
            "@id": "acr:hasInfo",
            "@type": "@id"
        },
        "hasLegalResource": {
            "@id": "cv:hasLegalResource",
            "@type": "@id"
        },
        "hasPart": {
            "@id": "dct:hasPart",
            "@type": "@id"
        },
        "hasParticipation": {
            "@id": "cv:hasParticipation",
            "@type": "@id"
        },
        "hasServiceInstance": {
```

```
            "@id": "acr:hasServiceInstance",
            "@type": "@id"
        },
        "hoursAvailable": {
            "@id": "schema:hoursAvailable",
            "@type": "@id"
        },
        "iconUrl": {
            "@id": "schema:url",
            "@type": "rdfs:Literal"
        },
        "identifier": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "ifAccessedThrough": {
            "@id": "cv:ifAccessedThrough",
            "@type": "@id"
        },
        "implements": {
            "@id": "cpsv:implements",
            "@type": "@id"
        },
        "isClassifiedBy": {
            "@id": "cv:isClassifiedBy",
            "@type": "@id"
        },
        "isDefinedBy": {
            "@id": "cv:isDefinedBy",
            "@type": "@id"
        },
        "isDescribedAt": {
            "@id": "cv:isDescribedAt",
            "@type": "@id"
        },
        "isGroupedBy": {
            "@id": "cv:isGroupedBy",
            "@type": "@id"
        },
        "isPublicService": {
            "@id": "acr:isPublicService",
            "@type": "xsd:boolean"
        },
        "issued": {
            "@id": "dct:issued",
            "@type": "rdfs:Literal"
        },
        "keyword": {
            "@id": "dcat:keyword",
            "@type": "rdfs:Literal"
        },
```

```
        "publicKey": {
            "@id": "ids:publicKey",
            "@type": "@id"
        },
        "keyType": {
            "@id": "ids:keyType",
            "@type": "rdfs:Literal"
        },
        "keyValue": {
            "@id": "ids:keyValue",
            "@type": "rdfs:Literal"
        },
        "landingPage": {
            "@id": "dcat:landingPage",
            "@type": "@id"
        },
        "language": {
            "@id": "dct:language",
            "@type": "@id"
        },
        "libraryDomain": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "loginUri": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "linkingRedirectUri": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "objectionUri": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "notificationUri": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "member": {
            "@id": "skos:member",
            "@type": "@id"
        },
        "onBehalf" : {
            "@id": "prov:actedOnBehalfOf",
            "@type": "@id"
        },
        "openingHours": {
            "@id": "schema:openingHours",
            "@type": "rdfs:Literal"
```

```
        },
        "operatorName": {
            "@id": "acr:operatorName",
            "@type": "rdfs:Literal"
        },
        "organizationName": {
            "@id": "vcard:organization-name",
            "@type": "rdfs:Literal"
        },
        "ownedBy": {
            "@id": "cv:ownedBy",
            "@type": "@id"
        },
        "page": {
            "@id": "foaf:page",
            "@type": "@id"
        },
        "playsRole": {
            "@id": "cv:playsRole",
            "@type": "@id"
        },
        "prefLabel": {
            "@id": "skos:prefLabel",
            "@type": "rdfs:Literal"
        },
        "processingTime": {
            "@id": "cv:processingTime",
            "@type": "rdfs:Literal"
        },
        "produces": {
            "@id": "cpsv:produces",
            "@type": "@id"
        },
        "publisher": {
            "@id": "dct:publisher",
            "@type": "@id"
        },
        "related": {
            "@id": "dct:relation",
            "@type": "@id"
        },
        "requires": {
            "@id": "dct:requires",
            "@type": "@id"
        },
        "role": {
            "@id": "cv:role",
            "@type": "@id"
        },
        "sector": {
            "@id": "cv:sector",
```

```
            "@type": "@id"
        },
        "serviceProvider": {
            "@id": "acr:serviceProvider",
            "@type": "@id"
        },
        "serviceUrls": {
            "@id": "acr:serviceUrls",
            "@type": "@id"
        },
        "businessId": {
            "@id": "dct:identifier",
            "@type": "rdfs:Literal"
        },
        "name": {
            "@id": "foaf:name",
            "@type": "rdfs:Literal"
        },
        "postalcode": {
            "@id": "schema:postalCode",
            "@type": "rdfs:Literal"
        },
        "city": {
            "@id": "vcard:locality",
            "@type": "rdfs:Literal"
        },
        "state": {
            "@id": "vcard:region",
            "@type": "rdfs:Literal"
        },
        "country": {
            "@id": "vcard:country-name",
            "@type": "rdfs:Literal"
        },
        "jurisdiction": {
            "@id": "eli:jurisdiction",
            "@type": "rdfs:Literal"
        },
        "spatial": {
            "@id": "dct:spatial",
            "@type": "@id"
        },
        "status": {
            "@id": "adms:status",
            "@type": "@id"
        },
        "telephone": {
            "@id": "schema:telephone",
            "@type": "rdfs:Literal"
        },
        "thematicArea": {
```

```
            "@id": "cv:thematicArea",
            "@type": "@id"
        },
        "title": {
            "@id": "dct:title",
            "@type": "rdfs:Literal"
        },
        "type": {
            "@id": "dct:type",
            "@type": "@id"
        },
        "value": {
            "@id": "cv:value",
            "@type": "xsd:double"
        },
        "versionInfo": {
            "@id": "owv:versionInfo",
            "@type": "http://www.w3.org/2001/XMLSchema#string"
        }
    }
}
```

6 Annex II - Service Catalogue APIs

Several Swagger-UI screenshot are reported below, in order to summarize the current APIs exposed by the Service Catalogue for the management of Service descriptions (Figure 51) related connectors (Figure 52) and adapters (Figure 53), and Federated catalogues management (Figure 54, Figure 55) and query (Figure 56).

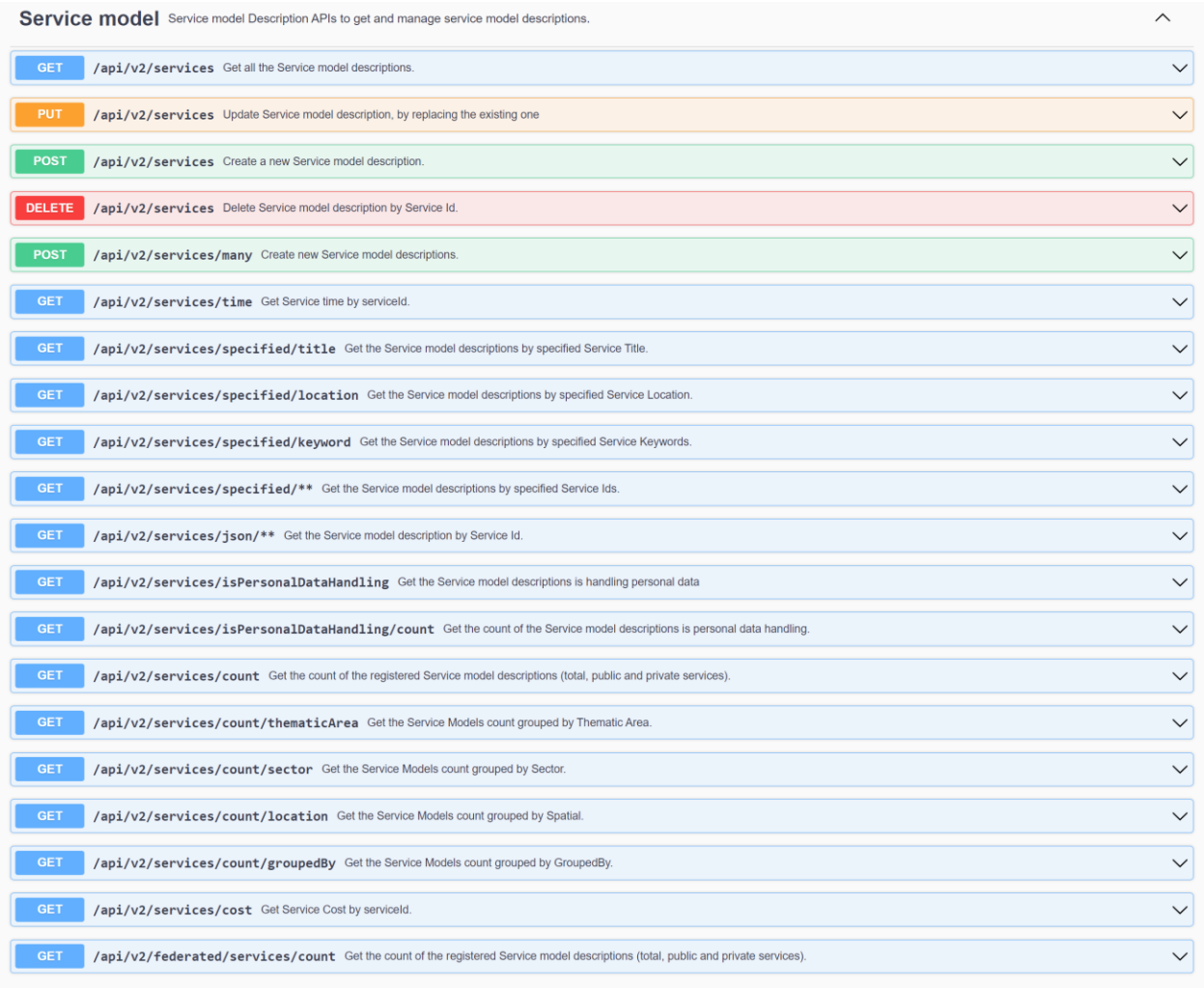

Figure 51 - Documentation of the API of Service Catalogue (Service Model)

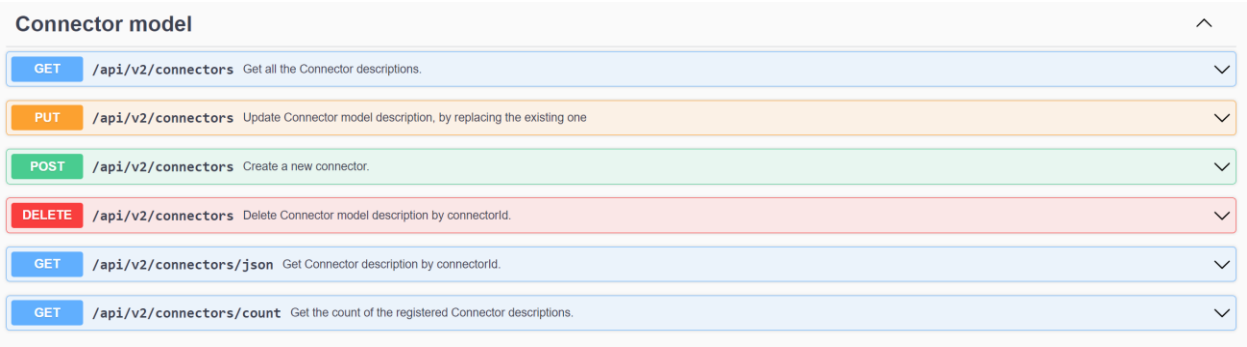

Figure 52 - Documentation of the API of Service Catalogue (Connector Model)

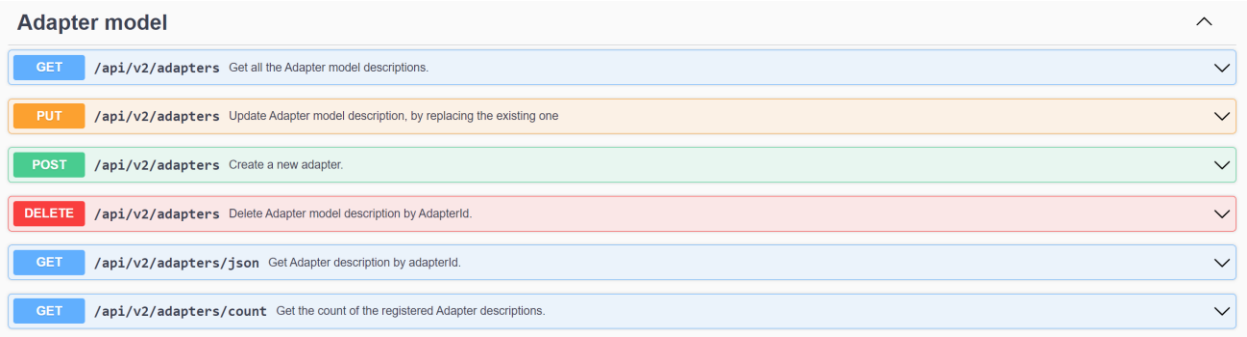

Figure 53 - Documentation of the API of Service Catalogue (Adapter Model)

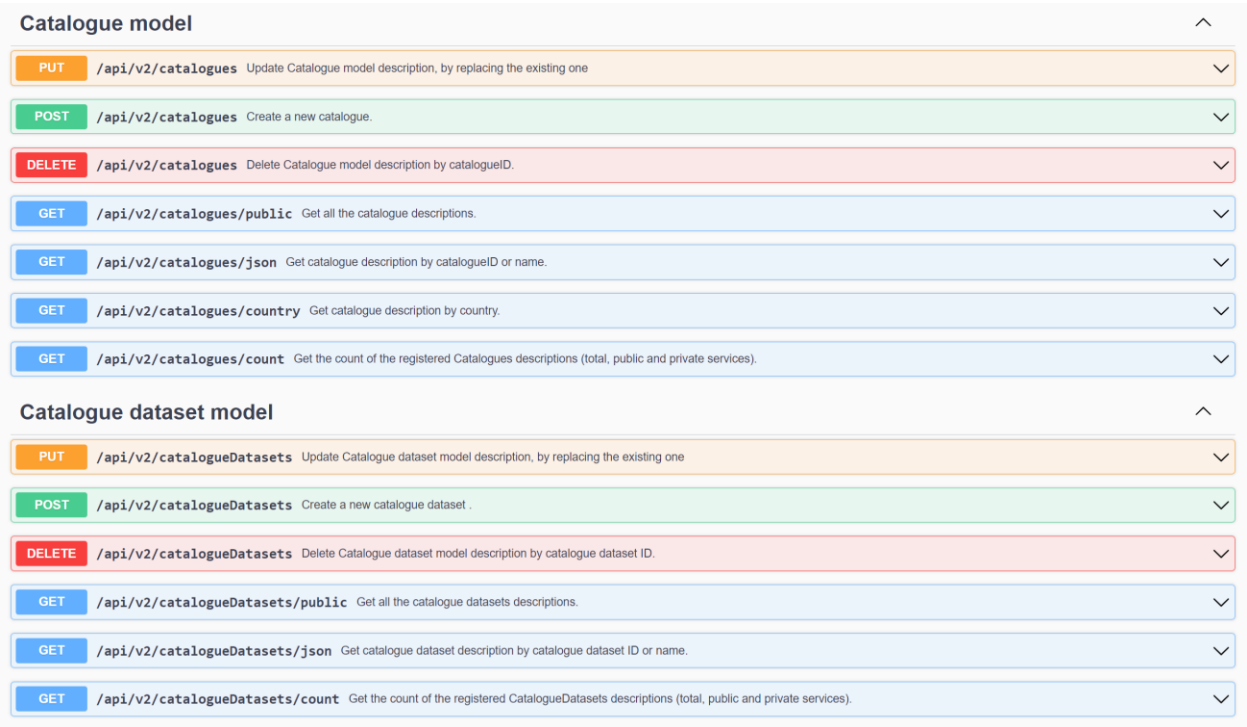

Figure 54 - Documentation of the API of Service Catalogue (Catalogue Model and Dataset)

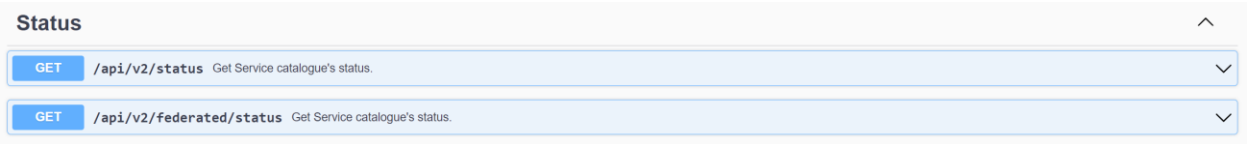

Figure 55 - Documentation of the API of Service Catalogue (Catalogue Status)

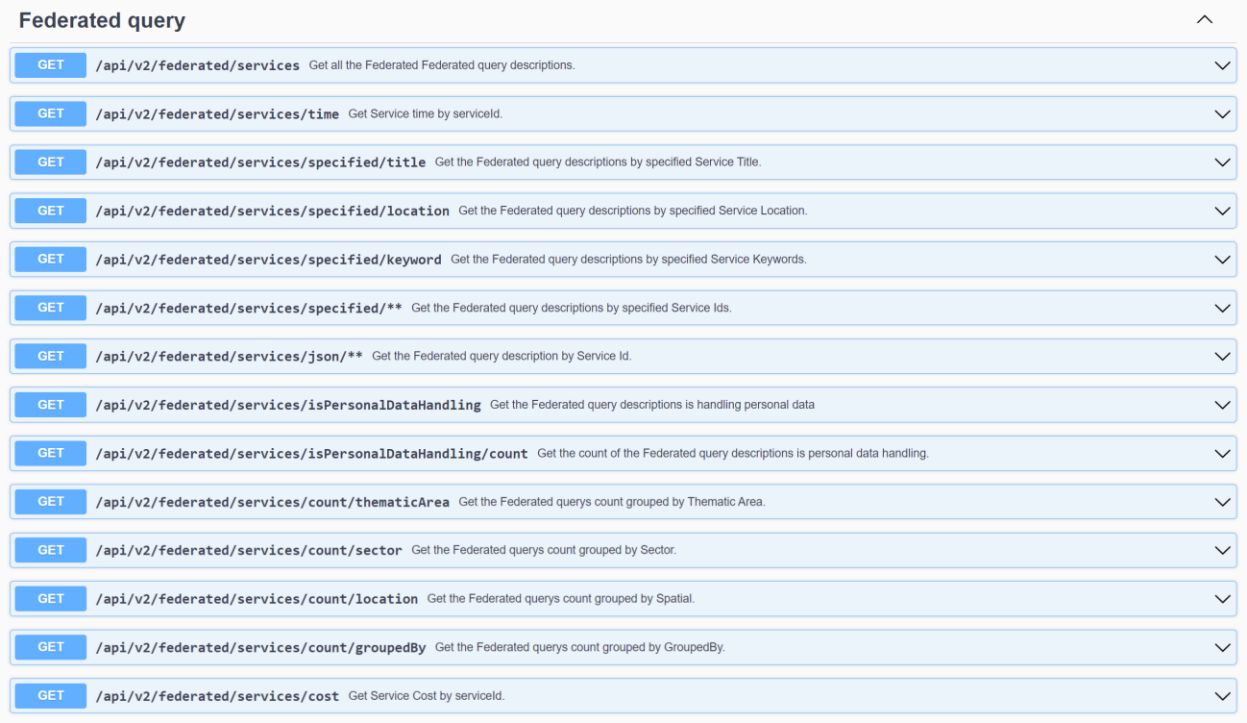

Figure 56 - Documentation of the API of Service Catalogue (Federated Query)

Service Catalogue APIs are protected by the Keycloak Oauth2 authorization server. An external client application/service that wants to interact with Service Catalogue by using the APIs, must perform one of the available OAuth2 flows (Authorization Code, Client Credentials and Password grants) against the Keycloak IdM, in order to get an Access Token and then use it in the API requests.

7 Annex III - ACROSS Requirements Mapping

Table 2 - Functional and Technical Requirement

Id	Title	Description	Type	Category
Req_01	Semantic and technical interoperability with SDG	The system should ensure an alignment of semantic and technical interoperability with SDG IT Tools	non functional	Platform architecture and interoperability
Req_09	Free access to other countries' e-services	As a user I want to be able to access other countries' services	non functional	Platform architecture and interoperability
Req_13	Interoperability with legacy systems	It has to be possible to connect the ACROSS platform with the existent PA legacy systems (e.g. databases, web services). Secure and reliable communication with the existing public administration information systems have to be provided without requiring changes in these systems. The platform should also provide tools and predefined components to facilitate the interoperability.	non functional	Platform architecture and interoperability
Req_15	Easy to use service integration and orchestration tools	In order to create cross border services the platform has to support Public and Private providing a set of tools and applications that will help them to easily implement service integration.	functional	Connectors to integrate the private and public sector offering
Req_16	Open API access	Data and services available in the ACROSS platform have to be accessible via a set of APIs using	functional	Platform architecture and interoperability

		standardized approaches(e.g. RESTful API).		
Req_17	Service Registries	ACROSS platform has to maintain registries of all available services offered by different PAs, SMEs and by the platform itself. Every service should be well-described using standard metamodels	functional	Connectors to integrate the private and public sector offering
Req_18	Cross Border Authentication	The services deployed and executed in ACROSS platform should have the possibility to be integrated, if needed, with eIDAS system. The platform can optionally support single-sign-on mechanism to simplify authentication on multiple applications and services internally to the platform.	functional	Security and Privacy
Req_19	Reliability and Integrity	The implementation of ACROSS should follow open standards and use well-known and widely accepted technologies in order to ensure integrity. The ACROSS platform has to be reliable assuring integrity of the components/tools that are part of it.	non functional	Platform architecture and interoperability
Req_20	Security access	Access to services and data has to be available to authorized users/applications only. Only audited applications are allowed to be deployed to ensure compliance with the security policies. Every security violation should be reported and the necessary actions to protect information	functional	Security and Privacy

		and applications present in the platform has to be performed.		
Req_27	Catalogue of services (public/private) data model	The Catalogue of services data model will follow the common public core vocabularies coming from ISA2 and the EIF implementation regulation and will support interoperability with SDG	functional	Platform architecture and interoperability
Req_28	Catalogue of services (public/private) objective	The catalogue of services will take care of harmonisation of the private and public services and related data enabling semantic interoperability and supporting the selected common vocabularies should be used to express the metadata.	functional	Platform architecture and interoperability
Req_29	No vendor lock-in	I want the ACROSS reference architecture to be technologically agnostic to avoid vendor lock-in.	non functional	Platform architecture and interoperability
Req_30	Open source	I want the ACROSS reference architecture to reuse already available open source solutions and only create or improve those aspects that are not covered by the existing solutions	non functional	Platform architecture and interoperability

Req_35	Usability and adaptability	The provided solutions in the platform should be user-friendly and easy to use and should be multilingual. No piece of text that might be displayed to a user shall reside in source code and solution and user should be able to select the preferred language . The implementation of the system should follow open standards and use well-known and widely accepted technologies in order to ensure ease of use.	non functional	Platform architecture and interoperability
Req_36	Minimal browser support.	The component user interface (where available e.g. dashboards, forms, etc...) should provide support for the wide range of widely used browsers.	non functional	Web&Mobile applications

The following table provides additional requirements and recommendations that are/will addressed by the Service Catalogue, selected from the user requirements coming from the initial use case evaluation [15] also reported in [1]:

Table 3 - User Requirements and Recommendations from [15]

No.	Title	Description
Req_3	Tutorials & examples tool	As user I want a place where I can access examples on how to perform services, fill in forms, and access other relevant information on services depending on the country.
Req_4	Information tool	As user I want a place where general information on migration to other countries is stored and constantly updated. It must be written in simple and understandable language.
Req_5	Connections to outer sources	As user I want to be able to view relevant informational links – national platforms, suggestions on job search

		portals, housing market, education portals, etc. while I access services.
Rec_C	Integrated application forms' features must be extended and standardized	As I service provider that I would like to integrate an application form in ACROSS platform, I want to give the same experience to the end user as with the original application form
Rec_L	Facilitating trust in the process and product	It has to include official contacts of each service owner and possibly their social media accounts for ability to reach out for support (contacting a human being, not an AI solution).
Rec_M	Reusability of Components and Technologies	As a developer I would like to download the components developed for ACROSS (like Transparency Dashboard, User Journey Engine, eIDAS proxy, ect) well documented and ready-to-use by other platforms.
Rec_N	Estimated processing time for applications done in ACROSS	As a user, I would like to have an overview of the expected processing time by the authorities for my applications, in order to simplify my time planning for the preparation of my stay abroad.
Rec_O	Overview about all costs and fees for administrative services	As a user, I would like to have an overview of the expected direct and upcoming costs, especially for mandatory official notices from authorities, in order to be able to plan my stay abroad financially.