openMINTED
Open Mining Infrastructure for Text & Data

Platform Architectural Specification

July 31, 2017

Deliverable Code: D6.3

Version: 2.0

Dissemination level: Public

The aim of this document is to present a detailed description of the OpenMinTeD platform. The document focuses on the description and communication among the different services designed by the partners and aims at driving and coordinating system development.

H2020-EINFRA-2014-2015 / H2020-EINFRA-2014-2
Topic: EINFRA-1-2014
Managing, preserving and computing with big research data
Research & Innovation action
Grant Agreement 654021

Document Description

D6.3 – Platform Architectural Specification

WP6 – Platform Design and Implementation
WP participating organizations: ARC, University of Manchester, UKP-TUDA, INRA, OU, CNIO, USFD, GRNET

Contractual Delivery Date: 31/07/2017	**Actual Delivery Date:** 10/05/2018
Nature: Report	**Version:** 2.0 Final
Public Deliverable	

Preparation slip

	Name	Organization	Date
From	Antonis Lempesis	ARC	31/01/2018
	Dimitris Galanis	ARC	22/11/2017
	Mark Greenwood	USFD	30/08/2017
	Vaggelis Floros	GRNET	01/09/2017
	Nicolas Liampotis	GRNET	01/09/2017
	Jacob Carter	University of Manchester	01/10/2017
Edited by	Stefania Martziou	ARC	05/02/2018
	Antonis Lempesis	ARC	06/02/2018
Reviewed by	Byron Georgantopoulos	GRNET	18/05/2018
	Panagiotis Zervas	AK	30/05/2018
Approved by	Androniki Pavlidou	ARC	31/05/2018
For delivery	Mike Hatzopoulos	ARC	4/06/2018

Document change record

Issue	Item	Reason for Change	Author	Organization

V0.1	Draft version	Initial version sent for comments/additions	Antonis Lempesis	ARC
V1.0	First version	Incorporating reviewers' comments	Antonis Lempesis	ARC
V2.0	Final version	Finalizing	Antonis Lempesis	ARC

Table of Contents

Table of Figures

Disclaimer

OpenMinTeD is a project funded by the European Union (Grant Agreement No 654021).

Acronyms

OAI-PMH	Open Archives Initiative - Protocol for Metadata Harvesting
TDM	Text and Data Mining
AAI	Authentication & Authorization Infrastructure
REST	Representational State Transfer
JAR	Java Archive
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
ID	Identifier
XML	Extensible Markup Language
JSON	JavaScript Object Notation
JMS	Java Messaging Service
XSD	XML Schema Description
CRUD	Create, Read, Update and Delete
DOI	Digital Object Identifier
API	Application Programming Interface
OMTD	OpenMinTeD
SAML	Security Assertion Markup Language
LDAP	Lightweight Directory Access Protocol
SQL	Structured Query Language
IdP	Identity Provider
SP	Service Provider
CRUD	Create Read Update Delete
UIMA	Unstructured Information Management Architecture
GATE	General Architecture for Text Engineering

Publishable Summary

OpenMinTeD aspires to enable the creation of an infrastructure that fosters and facilitates the use of text and data mining technologies in the scientific publications world and beyond, by both application domain users and text-mining experts.

OpenMinTeD builds upon existing tools and text mining platforms, rendering them discoverable, through appropriate registries, and interoperable, through an existing standards-based interoperability layer.

This document presents a detailed description of the OpenMinTeD platform: the services that comprise it, their functionality and architecture, and the interactions between these services.

1. Introduction

OpenMinTeD aspires to enable the creation of an infrastructure that fosters and facilitates the use of text and data mining technologies in the scientific publications world and beyond, by both application domain users and text-mining experts.

OpenMinTeD builds upon existing tools and text mining platforms, rendering them discoverable through appropriate registries, and interoperable, through an interoperability layer building largely on existing standards.

OpenMinTeD supports awareness of the benefits and training of text mining users and developers alike and demonstrates the merits of the approach through several use cases identified by scholars and practitioners from different scientific areas, ranging from life sciences (bioinformatics, biochemistry, etc.) to food and agriculture and social sciences and humanities related literature.

The goal of the project is to establish an open and sustainable TDM platform and infrastructure where researchers can collaboratively create, discover, share and re-use knowledge from a wide range of text-based scientific related sources in a seamless way to advance research, promote interdisciplinary open science, and ultimately support evidence-based decision making.

1.1 Purpose of this document

The aim of this document is to present a detailed description of the OpenMinTeD platform. The document focuses on the description of the functionality of the various services, their architecture, and the communication between them, and aims at driving and coordinating system development. The target audience of this document includes software designers responsible for developing OpenMinTeD services as well as anyone interested in understanding the high-level software structure of the system.

1.2 Document layout

This deliverable is structured as follows: in the second section we present the overview of the platform, with a brief description of the main services the platform provides to its users. In the third section we present the services in the *enabling layer*, i.e. the services that provide support to every other service in the platform. In the fourth section we present the services in the *data layer*, which are responsible for storing, retrieving and managing the data that the platform is using and are also responsible for interacting with all the external to the project services and data sources. In the next, section we present the *service layer*, i.e. the layer containing the main services of OpenMinTeD that are providing the main functionality to the end users. Finally, in the fifth section we are describing in brief the user interface of the platform (for more details, see D6.4 - Platform UI specification).

2. Overview

OpenMinTeD will offer a **Registry** service for storing, browsing, downloading, searching and managing various resources such as publications, processing components (e.g. a named entity tagger or sentence splitter) and language resources (e.g. machine learning models, lexica, thesauri). These resources will be imported/registered in OpenMinTeD by using a set of specifications/protocols (e.g. Maven[1]) and they will be documented with high quality metadata. The **Workflow Editor** service of the platform will guide users (via an appropriate UI) in creating interoperable workflows of TDM components, which will be executed by the **Workflow** service in a cloud infrastructure (or on a local machine). OpenMinTeD users will be able to annotate the publications (texts) using the **Annotation Editor** service to create datasets which can be used in workflows; e.g. for evaluation purposes. Below, in **Error! Reference source not found.**, an overview of the OpenMinTeD architecture is presented. In the next sections, the layers and components of the OpenMinTeD platform are presented in detail.

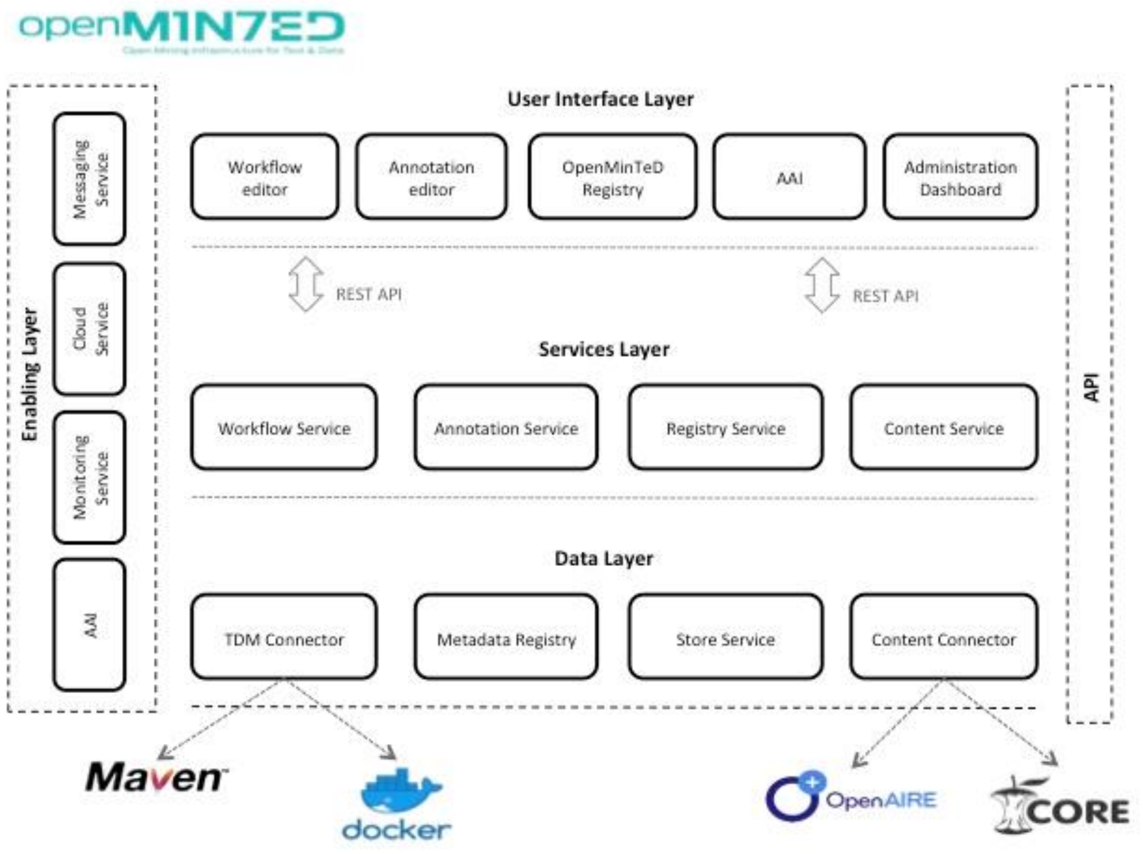


Figure 1 OpenMinTeD Architecture

[1] https://maven.apache.org/

2.1 Corpus

In the context of OpenMinTeD, a corpus is either 1) a collection of publications along with their metadata or 2) the result of a workflow execution: the publications, the generated annotations (one file per publication), again along with relevant metadata. Each corpus is assigned an identifier by the platform and is immutable; i.e. its contents cannot be modified allowing the reproducibility and validation of an experiment. In OpenMinTeD, a corpus is stored as an archive in the Store Service. Each archive contains:

- A file with the metadata of the corpus itself.
- The metadata of each (raw or annotated) publication.
- The full text of each publication or the annotated file, and
- Optionally, the abstracts and the licenses of the publications

Since the corpora are immutable and the OpenMinTeD users or services are not allowed to modify their contents, a corpus resulting from an OpenMinTeD workflow (i.e. an annotated corpus) must be a replica of the input corpus containing one or more extra archives with annotations. This does not lead to a waste of storage space, since the Store Service contains optimizations that allow it not to store duplicate files.

The Corpus is the unit of work for the OpenMinTeD workflows. Every workflow that is executed by the Workflow service accepts as input a corpus (that may already contain annotated publications) and the output is again an annotated corpus, i.e. a corpus containing annotations on the input publications. Even if, for ease of use or for demonstration purposes, a user executes a workflow by uploading a few publications, in the background a new corpus is created and provided as input to the workflow. This allows for a simple and more uniform design of the workflow service.

2.2 ID Broker

In the previous versions of this deliverable we described a Broker Service whose list of functionalities included the generation of persistent identifiers. In the course of the project it was decided that OpenMinTeD would not be providing persistent identifiers, because this is out of the scope of the project. As a result of this decision, the Broker service was removed from the platform and its remaining functionality (mainly the resolution of externally generated identifiers, like CrossRef or DataCite) will be spread among the services that need it.

The identifiers generated by OpenMinTeD and assigned to the various resources are unique and persistent but without the formal guarantees accompanying persistent identifiers, like DOI. The only exception is the identifiers assigned to the annotated corpora that are produced by the platform. These corpora can be uploaded to Zenodo[2] (this depends on the user's wishes) and be assigned a DOI. However, these DOIs are assigned and managed by Zenodo and not the OpenMinTeD platform.

[2] https://zenodo.org/

3. *Enabling Layer*

The enabling layer of the OpenMinTeD platform contains all the services that perform the management of the infrastructure or provide functionalities used by every other service in the platform. These services are the following:

- **AAI service:** It provides authentication and authorization functionality
- **Monitoring service**: It monitors the use of the hardware resources and can be used to generate usage statistics and provide support for resource allocation decisions. It also monitors the state of the platform's services and notifies the system administrators when a service is no longer available.
- **Cloud service:** It manages the hardware resources and is also responsible for executing the TDM components and tools of each workflow
- **Messaging service:** It provides asynchronous communication between the platform's services.

In the next sections, each service's functionality will be described in detail and its architecture and implementation details will be presented.

3.1 AAI

3.1.1 Functionality

The AAI activity in OpenMinTeD started in Month 12 of the project (May 2016). During the first months of the project, we worked together with the AARC[3] project in order to identify the requirements of the scientific communities. This work resulted in a set of guiding principles:

- Users should be able to access the OpenMinTeD Services using the credentials they have from their Home Organizations using eduGAIN when possible, but alternative methods should be available.
- OpenMinTeD should expect to receive at least an identifier that uniquely identifies the user and that is stable across different sessions by the same user, coming from within the scope of the authentication source.
- Within the OpenMinTeD environment, a user should have one persistent non-reassignable non-targeted unique identifier.
- OpenMinTeD should define a set of minimum mandatory attributes, without which a user cannot access the OpenMinTeD platform.
- OpenMinTeD should attempt to retrieve these attributes from the user's Home Organization. If this is not possible, then an alternative process should exist in order to acquire and verify the missing user attributes.
- There should be a distinction (Level of Assurance/LoA) between self-asserted attributes and the attributes provided by the Home Organization.
- Access to the various services should be granted based on the OpenMinTeD roles assigned to the user.

[3] https://aarc-project.eu

- OpenMinTeD Services should not have to deal with the complexity of multiple IdPs/Federations/Attribute Authorities/technologies. This complexity should be handled centrally and should be hidden from the OpenMinTeD Services.

Based on these principles and following the guidelines from the AARC project, the OpenMinTeD AAI adopted an architecture centered around a multi-protocol proxy that interconnects Identity Providers (IdPs) residing outside of the OpenMinTeD platform with Service Providers (SPs) internal to the platform. The first prototype of the OpenMinTeD AAI provided support for IdPs and SPs compliant with SAML 2.0[4], which is the dominant authentication and authorization protocol for research and education identity federations. In November 2016, the OpenMinTeD AAI joined the eduGAIN[5] interfederation as a Service Provider under the REFEDS Research and Scholarship (R&S)[6] entity category in order to ensure sufficient attribute release, as well as the uniqueness and non-reassignability of user identifiers retrieved from institutional IdPs. Complementary to this, the OpenMinTeD AAI added support for OpenID Connect[7]/OAuth 2.0[8] IdPs, allowing users without an institutional account to use their Google, Facebook, LinkedIn and ORCID accounts in order to access OpenMinTeD services. By the end of Q2 2017, the OpenMinTeD AAI introduced support for OpenID Connect relying parties, thereby allowing integration with a wider range of services built on top of modern web standards (OAuth 2.0, REST and JSON) and, at the same time, enabling federated access for non-browser based resources, such as CLI tools and APIs in a standardized way.

Apart from serving as an authentication proxy, the OpenMinTeD AAI is also responsible for the initial registration of users requiring access to OpenMinTeD services. During the registration process, a user is assigned a personal OpenMinTeD ID, which is a persistent, non-reassignable, non-targeted, opaque, and globally unique identifier that allows services to consistently identify users when accessing the platform. The generated OpenMinTeD ID must be accompanied with a minimum set of attributes, including name, email, and affiliation information, which the AAI SP Proxy attempts to retrieve from the user's IdP. If this is not possible (e.g. due to insufficient attribute release policies), users are requested to supply the missing user attributes themselves during registration. Furthermore, the OpenMinTeD AAI allows users to link additional institutional or social identities to their personal OpenMinTeD ID. Thus, a user is able to access OpenMinTeD resources with the same OpenMinTeD ID, using any of the login credentials they have linked.

Finally, the OpenMinTeD AAI can be used to manage additional information about registered users, including their roles within the OpenMinTeD platform. Any role information associated with a user is made available to services upon authentication in the form of entitlements. In the case of SAML 2.0,

[4] https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
[5] https://www.geant.org/Services/Trust_identity_and_security/eduGAIN
[6] https://refeds.org/category/research-and-scholarship
[7] http://openid.net/specs/openid-connect-core-1_0.html
[8] https://tools.ietf.org/html/rfc6749

entitlements are expressed as eduPersonEntitlement[9] (ePE) attribute values, whereas in OpenID Connect via the edu_person_entitlements[10] claim. Services are then responsible for controlling access to resources and assigning authorisation privileges based on the received role information. It should be noted that the OpenMinTeD AAI supports aggregating user information from externally managed Attribute Authorities using connectors based on popular standards, such as SAML 2.0 attribute queries, REST APIs, LDAP/SQL queries.

3.1.2 Architecture

Figure 2 provides a high-level view of the AAI architecture illustrating the key functional components, the interfaces they expose, and the interactions between them.

[9] http://software.internet2.edu/eduperson/internet2-mace-dir-eduperson-201602.html#eduPersonEntitlement

[10] There is currently no standard OpenID Connect claim to express entitlements. However, the REFEDS OpenID Connect for Research and Education Working Group (OIDCre) is already investigating the standardisation of new claims for expressing the attributes defined in the eduPerson schema, including the eduPersonEntitlement.

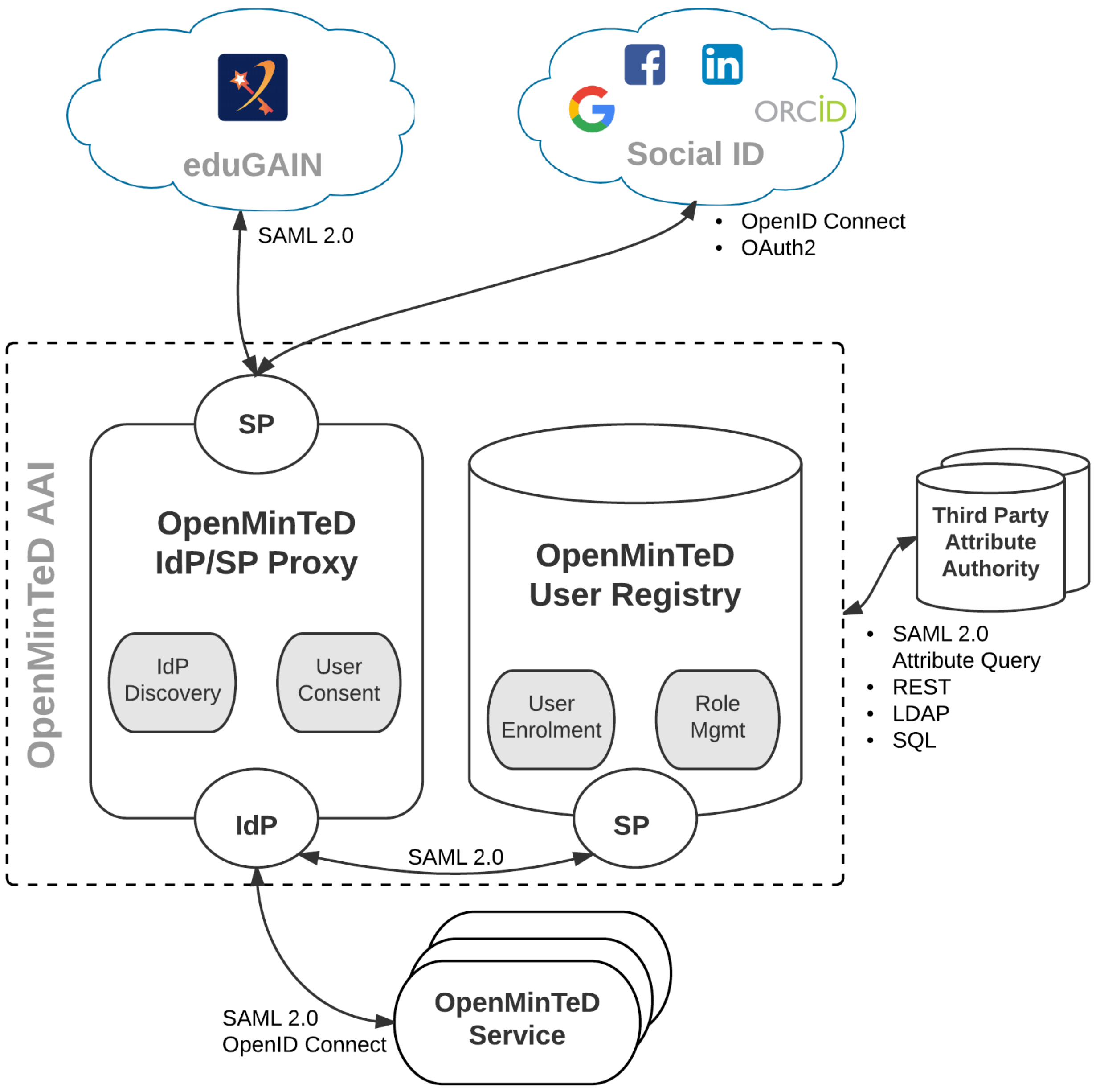


Figure 2 AAI Service

The core of the OpenMinTeD AAI is the **IdP/SP Proxy** component, which acts as a bridge between the OpenMinTeD services and external authentication providers. This separation between the internal services and the external authentication sources/identity providers allows the service developers to focus on the service features and abstract away the complexity of multiple IdPs, Federations, Attribute Authorities and different authentication and authorization technologies. This complexity is "outsourced" and handled centrally by the proxy. Services need to establish trust with just one entity, the IdP/SP proxy. Typically, services will have one static configuration for the IdP/SP proxy. Having one configured IdP, removes also the requirement from the services to operate their own IdP Discovery Service, which is a common requirement for services supporting federated access. Furthermore, all internal services will get consistent and harmonized user identifiers and attributes, regardless of the home organization or the research community that the user belongs to. Finally, this separation allows the change management of the internal services to be independent of the change management cycles at the home organizations IdPs. IdPs establish trust with one entity, the operator of the IdP/SP proxy, and they are not impacted by the change operations of each individual service.

The **User Registry** component supports the management of the full life cycle of OpenMinTeD user accounts. This includes the initial user registration, account linking, role management, delegation of administration of roles to authorized users and the configuration of custom enrolment flows for roles via an intuitive web interface. This functionality is also exposed through a REST API[11].

The implementation of the OpenMinTeD AAI architecture is based on the RCIAM[12] Identity and Access Management solution.

3.1.3 Interactions

As already mentioned, the OpenMinTeD AAI acts as an identity provider proxy that interacts with many external authentication providers using various protocols, such as SAML 2.0, OpenID Connect and OAuth 2.0:

- eduGAIN Identity Providers: To allow users to authenticate using the login credentials from their university or research institute. Through eduGAIN, OpenMinTeD services that are behind the AAI SP Proxy can become available to more than 2000 IdPs from over 40 Identity Federations with little or no administrative involvement.
- ORCID: To allow users to authenticate using their ORCID identifier. Information about the user can be obtained through the ORCID API[13] following a two-legged OAuth 2.0 authorization flow. Note that the Public ORCID API only provides access to the user's public profile information.
- Social Identity Providers: To enable researchers who do not have an account at a home organization federated through eduGAIN to be able to access OpenMinTeD services using their existing social media identities. To this end, the following social network providers are supported:
    - Facebook: Authentication and authorization is based on the OAuth 2.0 protocol. Access to user profile information is provided through the /{user-id}[14] Graph API endpoint, following the OAuth 2.0 login flow.
    - Google: Supports authentication and authorization through APIs that conform to the OpenID Connect specification. Thus, information about the user is retrieved from the UserInfo[15] endpoint in OpenID Connect format by requesting the openid scope.
    - LinkedIn: Relies on the OAuth 2.0 protocol for enabling authenticated access to its REST APIs that provide access to member data. More specifically, following a three-legged

[11] https://spaces.internet2.edu/display/COmanage/REST+API
[12] https://github.com/rciam
[13] https://orcid.org/organizations/integrators/API
[14] https://developers.facebook.com/docs/graph-api/reference/user
[15] https://developers.google.com/identity/protocols/OpenIDConnect#obtainuserinfo

OAuth 2.0 flow, LinkedIn user profile information can be accessed through the /people/~ REST API[16] endpoint.

3.2 Cloud Service

3.2.1 Overview

The Cloud layer of OpenMinTeD is in an overarching service responsible for managing the computing resources of the platform. The fundamental functionality provided is the allocation of new virtual machines with configurable amounts of CPU power, RAM, as well as the allocation of disk space necessary for the workloads and data managed by the OpenMinTeD platform.

An important aspect within the context of the OpenMinTeD platform is that the cloud service is supporting the execution of TDM workflows by providing the necessary computing power for the TDM components and the storage for the processed data. Since the consortium decided that Docker[17] will be used extensively to distribute and execute the code of TDM tools and workflow components, the cloud service provides a Docker oriented API, in which the unit of work is a Docker image instead of an arbitrary executable file.

3.2.2 Workflow execution backend

The workflow execution backend is responsible for orchestrating/executing the TDM workflows designed and submitted by the OpenMinTeD platform using the Registry and the Galaxy workflow engine[18]. An overview of the workflow architecture is depicted in **Error! Reference source not found.**.

[16] https://developer.linkedin.com/docs/rest-api
[17] https://www.docker.com/
[18] https://galaxyproject.org/

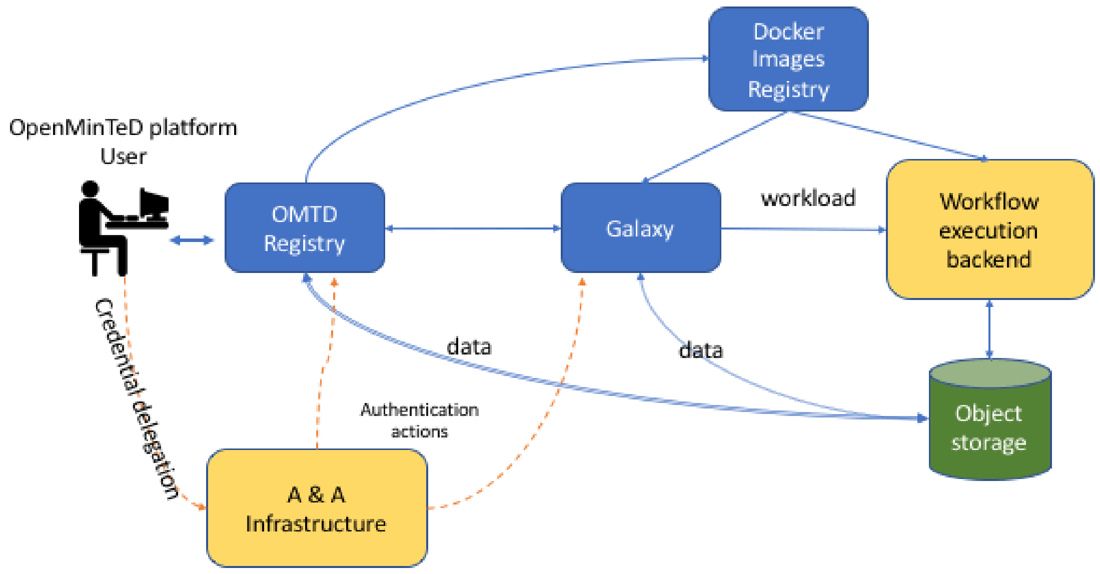


Figure 3 Workflow architecture overview

The modus operandi of workflow execution within this architecture is the following: The OpenMinTeD platform user connects to the Registry service in order to search existing Corpora, upload her/his own data, register TDM applications, design workflows and submit them for execution to the workflow engine. Authentication actions rely on the Authentication and Authorization Infrastructure supporting the platform.

In OMTD, TDM components or whole consolidated TDM applications are wrapped as Docker images and are registered on the OpenMinTeD platform-specific Docker Registry; the respective metadata for them are stored in the metadata registry. We support various TDM components such as

- UIMA-based and GATE-based TDM components which are automatically pulled from the respective Maven repos and dockerized by our platform
- TDM software not compatible to the UIMA or GATE frameworks that have already been dockerized[19] by their authors.
- Web services that are not deployed in the OpenMinTeD platform, but are hosted by third parties.

In all cases the Docker images should follow a specification that facilitates the integration and execution in OMTD.[20] The registered TDM components (e.g. UIMA-based, GATE-based) are available in the Galaxy workflow editor and can be used to create workflows by

- Connecting them and
- Setting component parameters (mandatory or not)

The TDM components are made available to the Galaxy server by generating automatically the respective XML files from the OMTD-SHARE descriptors and by uploading the XML files in the appropriate Galaxy folder.

Galaxy is the workflow engine in OMTD, responsible for managing the execution of TDM applications and workflows. The actual execution takes place in the Workflow execution backend of the architecture; consult "D6.8 – OpenMinTeD Platform services distribution specification" for more details. Finally, an

[19] Consult "D5.4 - Interoperability Standards and Specification" for more information
[20] https://github.com/openminted/omtd-docker-specification

object storage is providing a permanent cloud-based, large-capacity storage for storing input data and output results of the workflow execution.

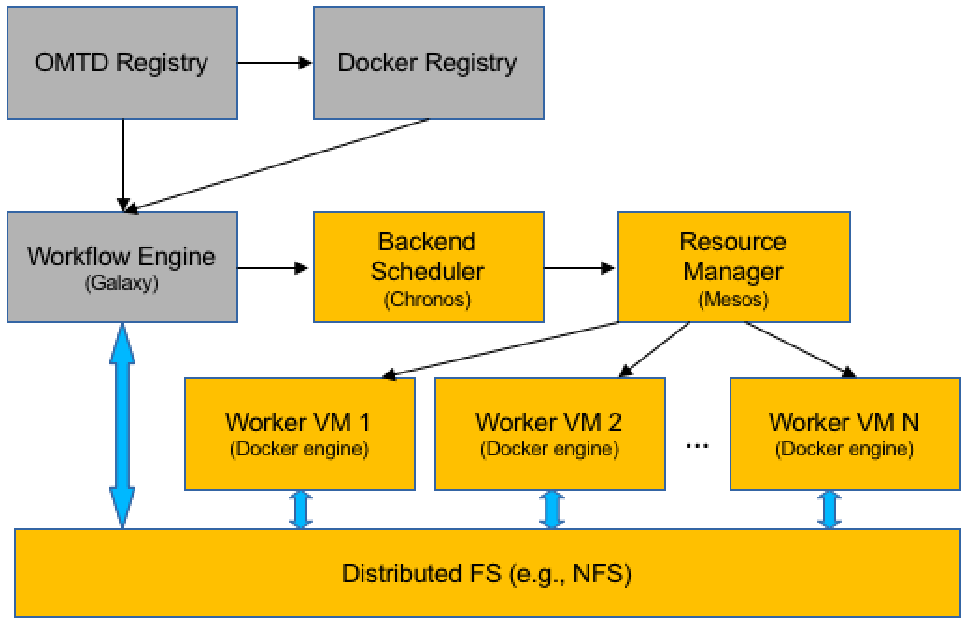

OMTD Registry
Docker Registry
Workflow Engine (Galaxy)
Backend Scheduler (Chronos)
Resource Manager (Mesos)
Worker VM 1 (Docker engine)
Worker VM 2 (Docker engine)
...
Worker VM N (Docker engine)
Distributed FS (e.g., NFS)

Figure 4 Workflow backend architecture

The main components of the backend architecture are (**Error! Reference source not found.**):

Backend scheduler: Responsible for receiving the workflows, managing priorities, and negotiating with the resource manager for the actual resources (worker VMs) needed for running each step of the application. The scheduler component is currently implemented using Chronos[21].

Resource manager: Is aware of the resources available by the cloud infrastructure. Allows pooling of VMs, network resources, storage etc. Communicates with the scheduler allocating resources required every time by a specific workflow. Currently, the resource manager is implemented using Apache Mesos[22].

Worker VMs: Are responsible for the actual execution of each step (component) of the TDM workflow. Workers have access to OMTD private registry that stores the Docker images (for the TDM components/apps). A shared file system is used for accessing input data and for storing the results of the dockerized application execution. The number of Worker VMs is not fixed; the infrastructure administrator can manually throttle it in order to adjust to specific demands. The size of the VM pool can be potentially managed in an automatic manner using a daemon application that continuously monitors the workload on the VMs and adds or removes VMs depending on the current workload.

[21] https://mesos.hithub.io/chronos/
[22] http://mesos.apache.org

Shared File System: Worker VMs and Galaxy need to have a shared file system for sharing data. For this reason, a distributed File System needs to be deployed using NFS. An NFS server is responsible for providing large storage to the rest of the VMs and for managing the sharing of data among them. Permanent data are stored at the object storage component and are staged back and forth to the distributed FS by Galaxy.

As mentioned above, the current implementation of the workflow backend in OpenMinTeD relies on the Chronos/Mesos stack for the scheduling/resource management functionalities respectively. Technical details for the architecture deployment implementation can be found in project deliverables D6.8 – “Platform services distribution specification” and D8.2 – “Infrastructure operation report”.

3.3 Messaging Service

A messaging service is a service that allows other services to communicate indirectly by exchanging messages. In the messaging paradigm, a message is a piece of information (e.g. an event, a request for an action, a notification that an action was completed, etc.) that is generated by one service (the *producer*) and is eventually received by another service (the *consumer*). A message can either have only one recipient, in which case a *queue* is used where the producer is adding messages and the consumer is removing them, or multiple recipients, in which case the producer is sending a message under a topic and all interested consumers subscribe to this topic and receive any new messages. The main feature of a messaging service is that it allows decoupled communication between the consumers and producers; the producer is not aware and not interested in which services will eventually receive the messages. This allows for the creation of simpler services, with fewer software dependencies and thus a much simpler overall architecture.

Since all the core services of the OpenMinTeD platform are developed in Java, the natural decision was to use the Java Messaging Service specification and more precisely the Apache ActiveMQ [23] implementation.

[23] http://activemq.apache.org/

4. Data Layer

The Data Layer of the OpenMinTeD platform contains the services that manage the content (individual publications or corpora), TDM tools and components and TDM related resources (e.g. a lexicon, a machine learning model, etc.). These services are responsible for managing the metadata of all these entities (Metadata registry), storing the actual data of the entities that are stored locally in the platform (Store service), acquiring the content from external sources (Content service) and acquiring information about TDM tools and resources again from external sources.

Each one of these services has a limited scope and its functionality is not strictly limited to OpenMinTeD. As such, with minor modifications or extensions, these services could be used from other projects that require similar functionality.

In the next sections, each service's functionality will be described in detail and its architecture and implementation details will be presented.

4.1 Store Service

4.1.1 Archive

An *archive* in the OpenMinTeD platform is an organized collection of files and potentially other archives and is completely analogous to a directory of a typical file system. Each archive is described by a set of metadata, including an identifier, a name, access rights, etc. and is managed by the Store service.

4.1.2 Functionality

The Store service of OpenMinTeD is responsible for storing, retrieving and managing archives. An archive may contain a TDM resource (a linguistic resource, a lexicon, etc.), the contents of a corpus (e.g. publications in PDF format) and any other piece of data that must be stored in the OpenMinTeD platform. The service allows its users to either upload a single file (in which case an archive that contains this file is created automatically by the service) or incrementally create an archive and append files or other sub-archives to it.

The Store service supports two different deployment scenarios. The first one is used in the OpenMinTeD platform and involves using Pithos+, the cloud based storage service provided by GRNET[24]. The second deployment scenario is used when the OpenMinTeD services are deployed in local sites, without access to GRNET services, in which case the local filesystem is used to store the data. However, the service is designed in such a way that more deployment options could be easily implemented in the future (e.g. use Amazon S3 storage or any other cloud based solution).

4.1.3 Implementation

The Store service is implemented in Java 8 and consists of two Maven projects; the *omtd-store-api* and *omtd-store-rest*.

[24] https://okeanos.grnet.gr/services/pithos/

- ***omtd-store-api:*** It contains the *StoreService* interface that has two implementations, *StoreServiceLocal* and *StoreServicePITHOS;* each of them is based on a different file system connector. In particular, the former implementation offers access to the local file system (e.g. hard drive of a VM) based on the functionalities that are provided by the respective built-in JAVA APIs. The latter provides access to *Pithos+* cloud storage using a REST-based client implementation available at a GitHub[25] repository maintained by GRNET.
- ***omtd-store-rest:*** It provides a REST API for the *StoreService.* The implementation is based on the Spring[26] framework and the respective Controller(s). The REST service is deployed to the Apache Tomcat Servlet container and it is offered as a standalone executable JAR file. The JAR is built using Spring-Boot [27] framework/library, which facilitates the creation of standalone applications; e.g. by offering the embedding of a Tomcat container. *omtd-store-rest* provides access to either *StoreServiceLocal or StoreServicePITHOS*; this is specified in a configuration filethat is loaded at startup. The same configuration file also stores the parameters for connecting to Pithos+ service; e.g. URL endpoint, token, UUID. The Store Service was designed to run on a separate server (from the Registry), which makes the OMTD platform more flexible and robust.

The Store Service provides operations such as:

- Upload an archive
- Delete an archive
- Store a file in an archive
- Create an archive
- Create an archive in another archive

Each archive is assigned a unique identifier.

4.1.4 Interactions

- Metadata Registry service, to store the metadata of OMTD Resources.
- Pithos+, the cloud based storage service provided by GRNET

4.2 Metadata Registry

4.2.1 Functionality

The OpenMinTeD platform manages a large and diverse set of resources: TDM tools and services, TDM linguistic resources, lexica, publication corpora, workflows, etc. These resources must be easily discoverable and in order to use them correctly, a set of metadata must be used to describe them. The Metadata Registry is a general-purpose service whose responsibility is to store and manage the metadata of the various OpenMinTeD resources.

[25] https://github.com/grnet/e-science/tree/master/pithosfs
[26] https://projects.spring.io/spring-framework/
[27] https://projects.spring.io/spring-boot/

The diversity of the resources dictates that the metadata registry must be resource type agnostic: administrators of the metadata registry must be able to add/modify and delete resource types while the service is running without the need to modify the source code of the registry or the underlying database schema. Another reason for this versatility is that the OpenMinTeD platform is under continuous development, with new features, services and resource types being added regularly and the metadata registry should be able to easily cope with these changes.

Metadata are traditionally described using XML but in the recent years JSON is gaining popularity among developers. The metadata registry is able to store metadata in either XML or JSON format and provide the same functionality. Another feature of the service is that it supports notions from the relational databases: primary keys, unique constraints, and even referential integrity between the values of different resource types. Finally, the metadata registry is able to notify interested users about changes in the contents by creating topics in JMS and posting events for every insertion/modification or deletion of a resource.

4.2.2 Architecture

4.2.2.1 Resource Type

In order to start storing and managing resources of a certain type (e.g. TDM components, lexica or workflows), a structure called ResourceType must be created and submitted to the metadata registry. The ResourceType structure contains the following information:

- **Name.** The name of the resource type. Which must be unique among all the resource types.
- **Payload type.** The format of the resource (XML or JSON). While the metadata registry supports both formats, the resources of each resource type must be expressed in one format, XML or JSON.
- **Schema** (or schema URL)**.** The schema of the resources (XSD for XML, JSON Schema[28] for JSON) or the URL of the schema, if it can be found online. The schema is used when a resource is added or modified to ensure that all stored resources are valid.
- **Indexed fields** (or IndexMapper)**.** While the underlying storage or the source code make no assumptions on the schema and contents of the stored resources, it is necessary for the service to extract some information from each resource type to allow the execution of queries. One such example is the email and name of a "Person" resource, which is described by an XSD schema. While a "Person" resource may contain a lot of information about the user, the users of the metadata registry need to search persons by their name or email. This is accomplished by specifying an IndexedField, which contains information about the searchable fields (the name of the field, its type (a string, a number, etc.) and the location of the value in the resource (XPath for XML resources or JSON Path[29] for JSON resources). When new resources are added or existing

[28] http://json-schema.org/
[29] https://github.com/jayway/JsonPath

ones are modified, the registry is using this list of IndexedFields to extract the values and store them in an index to allow for efficient queries.

4.2.2.2 Resource

The contents of the metadata registry are defined in a structure called Resource. The users of the service are storing and retrieving Resources. Each Resource contains the following information:

- **Id.** A unique identifier, which is assigned by the registry to each new resource.
- **Resource type.** The type of the resource. Before attempting to store a resource in the registry, its resource type must have already been registered.
- **Payload.** The actual contents of the resource. It is either an XML or a JSON file, depending on the payload type of the respective resource type. When storing a resource, instead of the actual payload, a URL of the payload can be supplied.
- **Creation** and **modification date.** The dates when this resource was inserted in the registry and when it was last modified.
- **Version.** Whenever a resource is modified its version number is incremented and the previous versions is retained. Only the latest version of a resource is available through the search APIs and the users can access the previous versions only by using the version API.

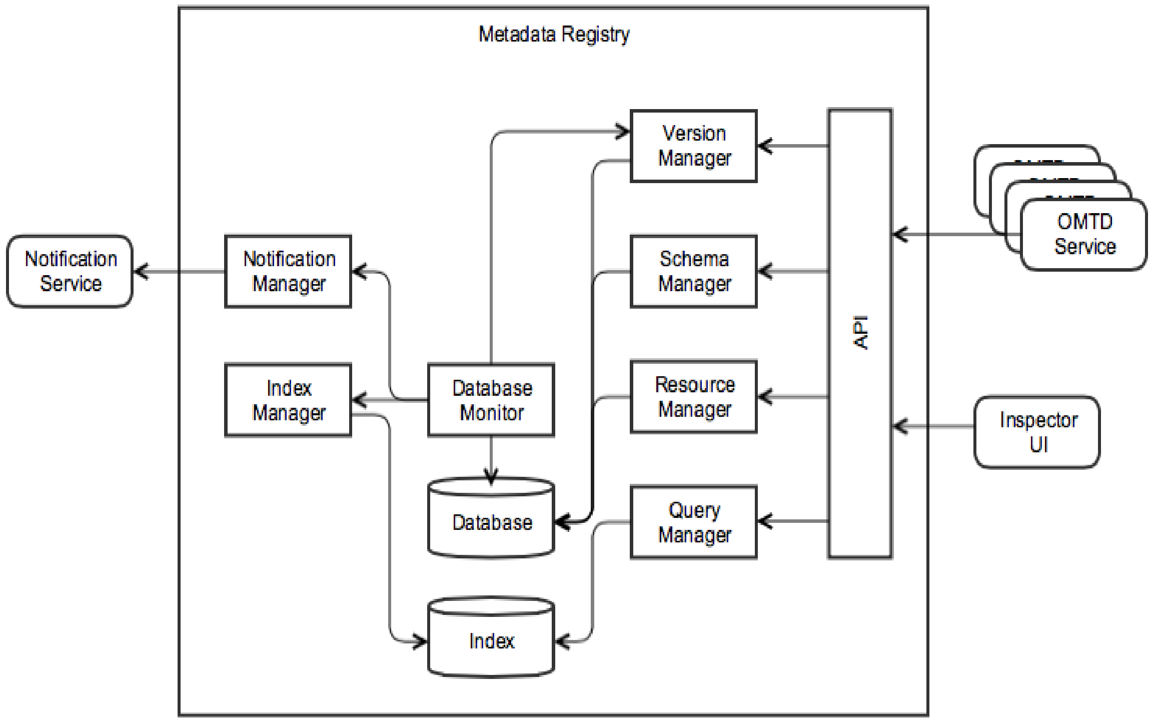


Figure 5 The Metadata Registry architecture

The Metadata Registry Service is composed by a number of different modules, each one responsible for a different task:

- **Schema Manager** is responsible for registering resource types, validating the schema provided, and preparing the database to accept resources of the new type.
- **Resource Manager** manages resources: validating new resources against the schema and the constraints (referential, primary keys and unique values), and registering them in the database. The resource manager is also used to update and delete existing resources.
- **Query Manager** performs queries at the index (based on the index fields defined in the resource type) and returns the results. While the resource manager is using the database to register the resources, the queries are performed at the index, which allows for far more efficient queries.
- **Database Monitor** monitors the database for any changes in registered resource types or resources and notifies other modules that are interested in these changes. For example, the Version Manager updates the version of resources when it receives a notification by the Database Monitor. These notifications from the database monitor to other modules allow for faster completion of the basic features of the registry (i.e. CRUD operations on resource types and resources) since the more computationally heavy operations (indexing, versioning, etc.) happen asynchronously.
- **Version Manager** manages the different versions of the resources. Whenever a resource is updated, the version manager stores the previous version and assigns a new version number to the modified resource. Using the Version Manager, users can trace the evolution of a resource from the moment it was created up to its latest version.
- **Index Manager** is responsible for updating the full text index of resources. The full text index allows the execution of queries on the resources providing results much faster than the relational database that is used to store the resource types and resources. By responding to notifications from the Database Monitor, the Index Manager keeps the underlying index synchronized with the contents of the database.
- **Notification Manager** provides the connection between the Metadata Registry and the Messaging service of the OpenMinTeD platform. Whenever a new resource type is registered, the Notification manager creates a new JMS topic (e.g. "registry.person.create" or "registry.person.update" that will be used when insertions and updates of resources of type "person" occur, respectively) and when resources are created, updated or deleted the Notification manager is sending new messages in the respective topic. The notification manager allows any other service in the platform to be asynchronously notified of the changes in the metadata registry without explicitly interacting with the metadata registry. Again, as is the case with the previous two modules, the Notification manager is notified for the changes in the resources by the database monitor.

The Metadata Registry is developed using Java 8, the underlying relational database is PostgreSQL and the full text index used is Elasticsearch**.**

4.2.3 Interactions

Since the Metadata Registry is one of the most basic and low-level services of the OpenMinTeD platform, it does not interact with almost any other service to implement its functionality. Instead, almost all services that handle metadata are using the Metadata Registry to manage them.

The only service that the Metadata Registry directly interacts with is the Notification Service, to post notifications about changes in the contents.

4.3 TDM Connector

4.3.1 Functionality

The TDM connector is the service whose responsibility is to locate and harvest metadata of TDM related resources from various external sources and transform them to the OpenMinTeD schema. A second functionality of the TDM Connector is to fetch the actual TDM resources (e.g. a lexicon or a machine learning model) when they must be stored in the OpenMinTeD platform. Although there is a wealth of information about TDM resources from various repositories and registries, for the first version of the platform we are going to focus on three external sources:

- Maven repositories, for metadata of GATE[30] and UIMA[31] TDM tools or models stored in JAR files.
- Docker registries, for OpenMinTeD compatible standalone TDM tools and web services.
- META-SHARE[32], for any kind of TDM resources like tools, services, lexica, corpora, models, etc.

In the next versions of the platform, we plan to include more sources of TDM resources, like the AgroPortal[33] or CIARDRING[34].

4.3.2 Interactions

- Maven repositories, for metadata of GATE and UIMA TDM tools or models stored in jar files.
- Docker Hub, for OpenMinTeD compatible standalone TDM tools and web services.
- META-SHARE, for any kind of TDM resources like tools, services, lexica, corpora, models, etc.

4.4 Content Connector

4.4.1 Functionality

The Content connector is the service responsible for bridging the services in the OpenMinTeD with the external content providers. OpenAIRE[35] and CORE[36] have been identified as the most suitable providers for the first phase but more may follow later, depending on the needs of the project. There are multiple

[30] https://gate.ac.uk/
[31] https://uima.apache.org/
[32] http://www.meta-net.eu/meta-share
[33] http://agroportal.lirmm.fr/
[34] http://ring.ciard.net/
[35] https://www.openaire.eu/
[36] https://core.ac.uk/

implementations and instances of the content connector, one for each external content provider. The content connector offers the following functionality:

- Performs mapping both from the OpenMinTeD metadata schema to the external provider's schema and the reverse, allowing the connector to return metadata in a common format.
- Provides search functionality by using the proprietary search API of the data provider and returning the results in a common format, usable by the Content Service.
- Provides access to the full text of the publications, allowing the Content service to build new corpora.

4.4.2 Architecture

Since there are multiple implementations of the Content connector, there is no single architecture that can describe all implementations. The only common thing between all content connectors is the API they implement and contains the following functionalities:

- **Search,** which provides faceted search functionality, used by the user interface when a user is locating the publications s/he needs to build a new corpus.
- **Fetching metadata**, which returns all the metadata of publications that belong to the new corpus
- **Fetching fulltext,** which returns the full text of a publication, again for inclusion in a new corpus

4.4.3 Interactions

- OpenAIRE and CORE, to access their search APIs and the full text of their hosted publications.

5. ***Services Layer***

The Services Layer of the OpenMinTeD platform contains the services that are providing the OpenMinTeD functionality to the end users. These services are the following:

- **Registry Service,** which stores and manages TDM-related resources.
- **Workflow Service,** which manages and monitors the execution of workflows.
- **Annotation Service**, which is responsible for storing, managing and exposing the annotations.
- **Content Service**, which is responsible for fetching publications and building corpora from external sources.

In the next subsections, we will describe the functionality and the architecture of each service in details and the interactions with the other services of the platform.

5.1 Registry Service

5.1.1 Functionality

The Registry Service stores and manages TDM-related resources: TDM tools (e.g. tokenizers, sentence splitters, POS Taggers), lexica, ontologies, annotation schemas, machine-learning models etc. It allows users to register new resources, search for them and use them to build and execute workflows/applications (using the workflow editor/service) or manage the output of workflows (annotation editor). The service is also used to manage auxiliary resources, not strictly TDM related: information about persons, organizations, projects, etc.

While the Metadata Registry hands the bulk of the metadata management operations (storing, retrieving and searching of metadata), it does so in a resource type agnostic way and provides an API so generic that cannot be easily used by the OpenMinTeD Registry UI or the other services. The Registry Service is building on the functionality of the Metadata Registry and provides an API that is tailored to the needs of the UI or the other services of the platform.

Another functionality of the Registry service is to harvest (on demand or at specified intervals) metadata about TDM related resources from external sources, like Maven repositories or Docker registries (for TDM components) and META-SHARE (for any kind of TDM related resources).

5.1.2 Architecture

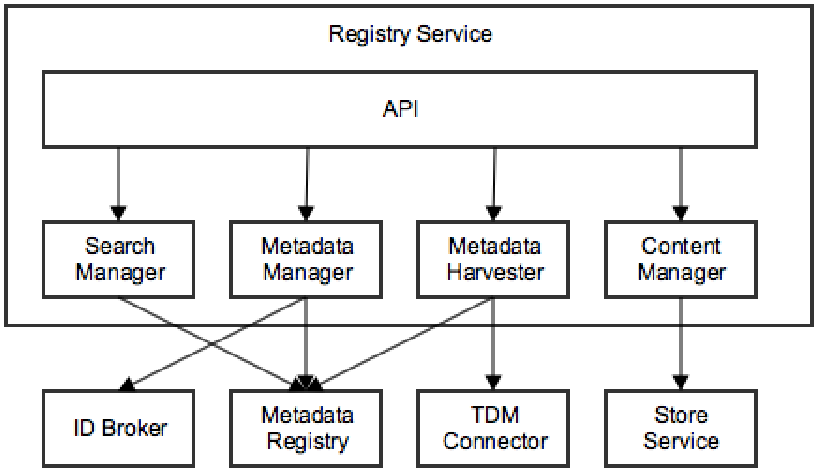


Figure 6 The Registry Service architecture

The Registry service comprises of a small number of modules, each one responsible for interacting with an OpenMinTeD service:

- **Search Manager,** which is responsible for performing queries on the metadata of the stored resources.
- **Metadata Manager,** which is responsible for retrieving, updating and storing the metadata of new and existing resources.
- **Content Manager,** which is responsible for storing and retrieving the actual TDM resources (e.g. a lexicon or a machine learning model) from the Store service.
- **Metadata Harvester,** which is responsible for harvesting metadata of TDM-related resources from external sources and storing them to the metadata registry. The harvest manager can be configured to either perform harvesting periodically or on demand.

5.1.3 Interactions

- Metadata Registry, to store the metadata of the resources.
- Object store, to store the actual contents of the resources.
- TDM Connector, to acquire metadata about TDM resources.

5.2 Content Service

5.2.1 Functionality

The Content Service is responsible for providing access to content from external sources. It allows OpenMinTeD users to perform queries for content metadata (both basic keyword search and more advanced filtering) using a common query language. The service redirects the query to multiple sources of content (OpenAIRE and CORE are examples of sources of publications), aggregates, homogenizes and transforms the results to the OpenMinTeD (OMTD-SHARE) format and returns them to the user. The service is configurable and extensible and thus able to use any external source that provides a search API.

Another responsibility of the Content Service is to provide access to the actual data (e.g. full text in case of publications). It allows users to either directly upload and describe with appropriate metadata a corpus or, by using the Content Connectors, create a new corpus containing publications from external sources (e.g. OpenAIRE or CORE) and edit the automatically generated metadata for it. The service is using the Object Store to store the contents of the corpora and the Metadata Registry to store and manage the metadata of each corpus.

When creating a new corpus, the input of the service is a user-supplied query (e.g. "the English publications published in PLoS ONE in 2016"). The service then uses the Content connectors to execute this query in all supported content sources and fetch both the full text files and the metadata of the publications that match the criteria. Using the Store Service, a new archive is created that contains the contents of the new corpus and a metadata record for the corpus is generated with information extracted from the metadata of the publications.

5.2.2 Architecture

The Content Service comprises of a number of modules, each one responsible for a specific functionality:

- **Search Manager,** which is responsible for performing searching for publication metadata in the external sources (OpenAIRE and CORE). For this functionality, it uses the Content connectors to perform the queries and retrieve the results.
- **Corpus Manager,** which performs the main functionality of the service by coordinating the rest of the modules: it downloads the contents of the new corpora from the external sources, creates new persistent identifiers, creates metadata and stores them in the registry, and also stores the contents to the Store Service. The corpus manager is using an internal database to store the status of each corpus-building request.
- **Metadata Manager,** which is responsible for creating and storing metadata for new corpora to the Registry and fetching metadata of existing corpora from the Registry.
- **Content Manager,** which is responsible for storing and retrieving the content of corpora from the Store Service.
- **Content Connector,** which (as described in the Content Connector section) is responsible for performing queries to the external content sources (OpenAIRE or CORE) and retrieving the metadata and publications of the results.

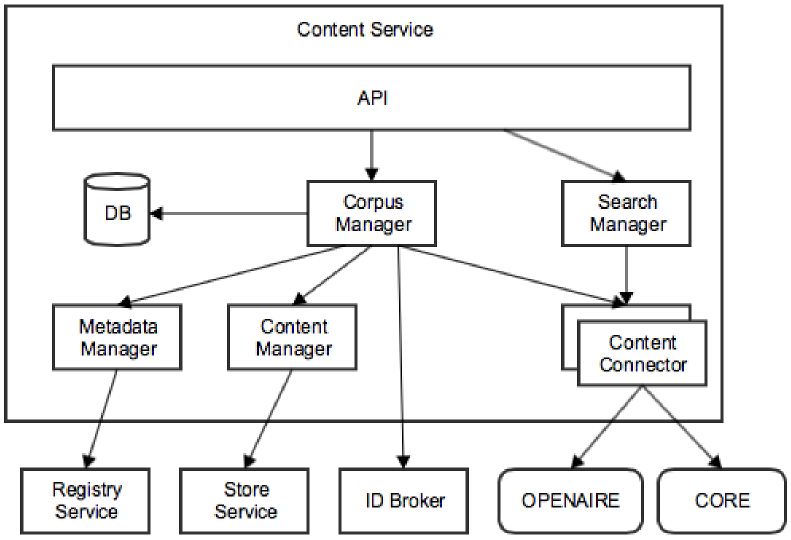


Figure 7 The Content Service architecture

5.2.3 *Interactions*

- Registry Service, to store the metadata of the corpora.
- Store Service, to store the contents of the corpora.
- Messaging Service, to provide notifications about the status of the submitted workloads.

5.3 Workflow Service

After a long and detailed process of evaluation, the OpenMinTeD consortium voted to adopt Galaxy as the platform workflow engine and workflow editor, in preference to both a number of other systems and to developing a new OpenMinTeD workflow system. Galaxy is a complete, web-based system originally designed for computational biology, which is also used by a number of other TDM infrastructures, including the LAPPS Grid project. Most projects that adopted Galaxy use the entire system as it includes not just a workflow engine and editor but also a simple component registry and data management functionalities. However, in OpenMinTeD we already have a fully functional registry (that handles resources and corpora), so we are only interested in the workflow management aspects of Galaxy. This presents us with some unique challenges as we attempt to use the workflow engine and editor in isolation from the rest of Galaxy. This non-standard use has revealed a number of limitations/issues within Galaxy, which have already been reported to the project maintainers.

5.3.1 *Functionality*

As already mentioned in the previous section, the OpenMinTeD consortium decided not to implement a new workflow execution service but instead opted to use the Galaxy Workflow Engine. However, Galaxy (as any other off-the-shelf software) cannot be used in the OpenMinTeD platform without an integration layer. The Workflow Service plays the role of the integration layer between the OpenMinTeD services and Galaxy. More specifically, the workflow service is responsible for:

- Allowing users of the OpenMinTeD Registry UI to start, stop and cancel workflow jobs.
- Managing and monitoring the execution of workflows.
- Ensuring that the correct corpora are retrieved from the OpenMinTeD Store and made available to Galaxy as well as pushing workflow output files into the Store.

5.3.2 Architecture

To make the platform more modular and to use Galaxy as required

- We created a Java interface that acts as an abstraction layer between the Registry and the workflow engine[37].
- We updated the aforementioned Java interface to the Galaxy API (which will eventually become an OpenMinTeD output). In particular, blend4j[38] has been adapted, so that it can communicate with the latest Galaxy server, and also has been extended to include some more functionalities.
- We implemented omtd-worfklow-api for Galaxy (omtd-workflow-service[39]) and implemented a REST interface on top of it (omtd-workflow-service-rest[40]).
- We connected the omtd-workflow-service to OMTD OpenMinTeD Message Service to facilitate integration with the rest of the platform.

5.3.3 Interactions

The current design of the workflow service limits the number of interactions between Galaxy and the rest of the OpenMinTeD platform. Those interactions that take place do so via the OpenMinTeD Messaging Service and currently include the following:

- Registry Service: to retrieve information about TDM tools and resources and to store the information about the workflow execution.
- Store Service: to retrieve information about corpora and the corpora themselves and to store the annotated corpora once a workflow execution has finished.
- Galaxy and Cloud service: Sends workflow execution requests to Galaxy which in turn uses the Chronos/Mesos cluster (Cloud service) for executing them.

5.4 Annotation Service

5.4.1 Functionality

The Annotation service is responsible for managing annotations. Its main responsibilities include the following:

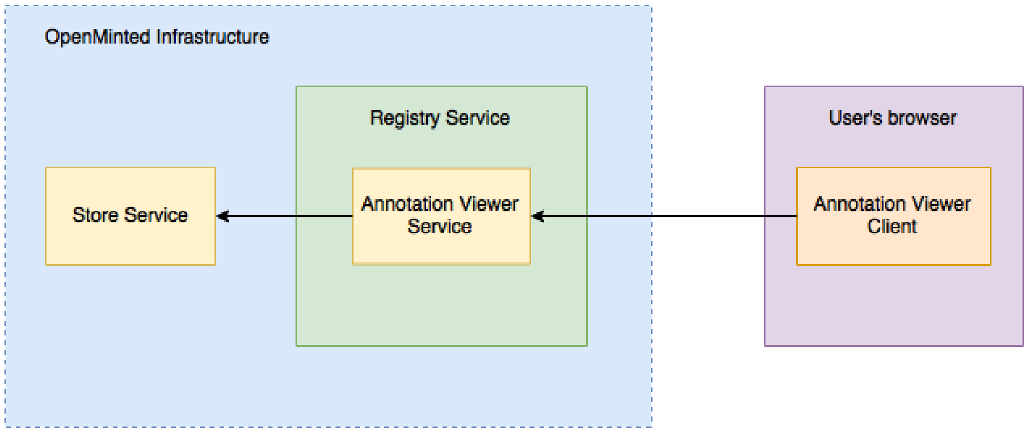
- Store and retrieve annotations from the Store service.

[37] https://github.com/openminted/omtd-workflow-api
[38] https://github.com/jmchilton/blend4j
[39] https://github.com/openminted/omtd-workflow-service
[40] https://github.com/openminted/omtd-workflow-service-rest

- Expose annotations in different formats.

5.4.2 Architecture

The Annotation Viewer service provides a REST web service API, allowing users access to annotated documents residing within the OpenMinTeD infrastructure. The Annotation Viewer, an entirely client-side web application developed for the OpenMinTeD platform, uses this service to enable users of the platform to view the results of their workflow executions. This architectural separation allows for the future development of alternative annotation viewing applications (e.g. mobile) without incurring any changes to the core OpenMinTeD services.



5.4.3 Interactions

- Metadata Registry to acquire information about executed workflows and corpora.
- Store Service to store and retrieve annotation.

6. *User Interface Layer*

This layer includes the user interfaces (UI) for interacting with the OMTD platform. Below we briefly present them; a detailed description is given in "D6.4 - Platform UI Specification".

6.1 Registry

The Registry UI is the main portal of the platform and the starting point for every user that wants to visit the OpenMinTeD platform. It offers the following functionality:

- Search and browse for registered TDM tools, components, corpora and TDM resources
- Register new TDM related resources
- Discovery of publications from the external sources (using the content service)
- Creation or uploading of new corpora
- Discovery and registration of TDM related resources and tools from external source
- User registration and account management
- Personal space for users containing "their own" data and resources that have not been made public (e.g. a closed access corpus, a private TDM resource, etc.)
- Ability to start the execution of workflows and monitor their progress.
- Links to the other portals (Workflow and Annotation editors) to edit or create new workflows or annotations.

6.2 Workflow Editor

The Workflow Editor UI of the OpenMinTeD platform is the editor provided by Galaxy; for more details on this choice, see the Workflow Service section. The Galaxy UI offers the following functionality:

- Find the components that the user is interested in (e.g. a named entity recognizer).
- Create workflows of interoperable tools by connecting them in a pipeline (or acyclic graph).
- Set the parameters of a processing component (e.g. a lexicon or a machine learning model).
- Create a new workflow by modifying an existing one.

As already mentioned (in previous sections), to make a TDM component/tool available in the Galaxy server an appropriate XML file is required. This file is automatically generated by the OMTD platform using as input the OMTD-SHARE descriptor (metadata) for the TDM component. As explained in the Galaxy documentation

"The XML File for a Galaxy tool, generally referred to as the "tool config file" or "wrapper", serves a number of purposes. First, it lays out the user interface for the tool (e.g. form fields, text, help, etc.). Second, it provides the glue that links your tool to Galaxy by telling Galaxy how to invoke it, what options to pass, and what files it will produce as output"

When new TDM components are registered to OMTD, the respective Galaxy XML files that are generated are (automatically) copied to a specific location in the Galaxy server; this location is monitored and the tools are made available to the Galaxy environment without requiring a restart.

In OMTD, to achieve robustness and task isolation, we deploy two instances of the Galaxy server. The one is responsible for workflow execution and the other is responsible for workflow editing. The latter is integrated to the Registry, i.e. a user that wishes to create/edit a workflow clicks the respective button which opens a Galaxy workflow editor window within the Registry UI. When designing is finished the window is closed and the workflow is saved to the Registry. Many of the functionalities, buttons and forms have been hidden in the Galaxy editing instance since they are not needed (e.g. upload data) or have been moved to the Registry (e.g. create/edit workflow button). As already stated, the workflows that are created in the editor are exported/saved in the Registry; they are moved to the Galaxy execution instance when this is required, e.g. when a workflow is published and is made available to all OMTD users.

6.3 Annotation Editor

The OpenMinTeD consortium decided to use WebAnno[41] as the annotation editor of the platform instead of creating a new editor. WebAnno is *"a general purpose web-based annotation tool for a wide range of linguistic annotations including various layers of morphological, syntactical, and semantic annotations".* It has been developed by UKP-TUDA and provides a large number of features, the most important of which are:

- Support for different user roles (annotators, project managers, administrators, etc.).
- Creation and management of annotation projects.
- Support for new annotations, curation and correction of existing annotations.
- Support for a large number of annotation formats.

For more information about WebAnno see deliverable D6.6-Platform UI specification.

6.4 Annotation Viewer

A bespoke annotation viewer[42] has been developed for the OpenMinTeD platform. The possibility of using WebAnno as both the annotation viewer and editor was considered, however it was finally decided that WebAnno would be too "heavyweight" as a viewer; it would expose additional unnecessary functionality in this role, thus impacting upon the end-user experience.

The annotation viewer is a web-based application, written in Java and compiled to JavaScript using the Google Web Toolkit (GWT) framework.

[41] https://webanno.github.io/webanno/
[42] https://github.com/openminted/omtd-annotation-viewer

A user of the OpenMinTeD platform will be able to browse and select an annotated document (residing inside a corpus on the platform) from the Registry UI, resulting in the document being shown in the annotation viewer.

The annotation viewer provides a number of features, including:

- Annotations overlaid onto the document text.
- List of annotations, alongside the document text, which allows inspection of each annotation's individual attributes.
- Navigation of nested annotation within the annotation list.
- Easier identification of annotations by type, as types are rendered in different colors.
- Support for multiple views within a single CAS.

7. References

[1] D4.3 – OpenMinTeD Functional Specifications
[2] D6.6 - Platform UI Specification
[3] D6.8 - OpenMinTeD Platform services distribution specification
[4] D8.4 – Infrastructure Operation Report (4th edition)