EUROPEAN LANGUAGE GRID

D2.5

ELG platform (second release)

Authors:	Penny Labropoulou, Dimitris Galanis, Miltos Deligiannis, Katerina Gkirtzou, Leon Voukoutis, Athanasia Kolovou, Dimitris Gkoumas, Stelios Piperidis (ILSP), Florian Kintzel, Remi Calizzano, Georg Rehm (DFKI), Ian Roberts, Kalina Bontcheva (USFD), Andis Lagzdiņš, Jūlija Meļņika (TILDE)

Dissemination Level:	Public
Date:	31-03-2021

About this document

Project	ELG – European Language Grid
Grant agreement no.	825627 – Horizon 2020, ICT 2018-2020 – Innovation Action
Coordinator	Dr. Georg Rehm (DFKI)
Start date, duration	01-01-2019, 42 months (GA amendment version: AMD-825627-7)
Deliverable number	D2.5
Deliverable title	ELG platform (second release)
Type	Other
Number of pages	38
Status and version	Final – Version 1.0
Dissemination level	Public
Date of delivery	Contractual: 31-03-2021 – Actual: 31-03-2021
WP number and title	WP2: Grid Platform – Language Grid
Task number and title	Task 2.3: Implementation, integration and initial population of the ELG platform & Task 2.4: Development and integration of APIs between technical components of the core ELG system
Authors	Penny Labropoulou, Dimitris Galanis, Miltos Deligiannis, Katerina Gkirtzou, Leon Voukoutis, Athanasia Kolovou, Dimitris Gkoumas, Stelios Piperidis (ILSP), Florian Kintzel, Remi Calizzano, Georg Rehm (DFKI), Ian Roberts, Kalina Bontcheva (USFD), Andis Lagzdiņš, Jūlija Meļņika (TILDE)
Reviewers	Katrin Marheinecke (DFKI), Gerhard Backfried, Katja Prinz (HENS)
Consortium	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Institute for Language and Speech Processing (ILSP), Greece
	University of Sheffield (USFD), United Kingdom
	Charles University (CUNI), Czech Republic
	Evaluations and Language Resources Distribution Agency (ELDA), France
	Tilde SIA (TILDE), Latvia
	HENSOLDT Analytics GmbH (HENS), Austria
	Expert System Iberia SL (EXPSYS), Spain
	University of Edinburgh (UEDIN), United Kingdom
EC project officers	Philippe Gelin, Alexandru Ceausu
For copies of reports and other ELG-related information, please contact:	DFKI GmbH European Language Grid (ELG) Alt-Moabit 91c D-10559 Berlin Germany Dr. Georg Rehm, DFKI GmbH georg.rehm@dfki.de Phone: +49 (0)30 23895-1833 Fax: +49 (0)30 23895-1810 http://european-language-grid.eu

Table of Contents

List of Figures

List of Tables

List of Terms

Functional content	Language processing tools and services that can be executed either locally or in a cloud infrastructure.
Language Data Resource, Language Data	A Language Resource composed of data, as opposed to a Language Technology tool or service.
Language Resource	A resource composed of linguistic material used in the construction, improvement or evaluation of language processing applications, but also, in a broader sense, in language and language-mediated research studies and applications; examples include data sets of various types, such as textual, multimodal or multimedia corpora, lexical data, grammars, language models, etc. in machine readable form. The term is often used in the bibliography and related initiatives with a broader meaning, encompassing also the (a) tools and services used for the processing and management of data sets, and (b) standards, guidelines and similar documents that support the research, development and evaluation of LT (cf. http://languagelog.ldc.upenn.edu/myl/ldc/LR_background.html). In this document, we use the term as first defined in the META-SHARE metadata model, i.e., including both data resources and Language Technology tools/services to avoid confusion with previous metadata descriptions. The alternative term “Language Resource/Technology” is also used.
Language Technology tool or service	A tool, service, or piece of software that performs language processing or any Language Technology related operation.
Non-functional content	Any non-executable type of resource (data resource or source code of tools/services) that can be included in the European Language Grid.

List of Abbreviations and Acronyms

API	Application Programming Interface
ASR	Automatic Speech Recognition
CD	Continuous Deployment
CI	Continuous Integration
CMS	Content Management System
CRUD	Create, Read, Update and Delete
DoA	Description of Action
DMP	Data Management Plan
ELG	European Language Grid
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IE	Information Extraction
IRI	Internationalised Resource Identifier
JSON	JavaScript Object Notation
JWT	JSON Web Token
K8s	Kubernetes
LR	Language Resource
LRT	Language Resource and Technology
LT	Language Technology

MT	Machine Translation
MVP	Minimum Viable Product
NCC	National Competence Centre
NLP	Natural Language Processing
OWL	Web Ontology Language
R1	Release 1
R2	Release 2
REST	Representational State Transfer
SDK	Software Development Kit
SPA	Single Page Application
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	YAML Ain't Markup Language

Abstract

This document introduces the interim release (Release 2, R2) of the ELG platform. We present the platform's architecture, its main building blocks, components and functionalities as well as the software and application frameworks used. The document includes an overview of the base infrastructure, the platform backend (with a detailed presentation of the catalogue application and the execution component for the LT processing services), and a short account of the catalogue frontend. A detailed presentation of the ELG catalogue graphical user interface is included in D3.3. The current contents of the catalogue and the ELG metadata model are also briefly described. Finally, the ELG user manual, an online document serving all ELG platform users (consumers, providers, validators of Language Resources and Technologies and administrators of the ELG Platform), which was initiated in Release 1, has been updated to reflect the current changes and additions. ELG Release 2 is available at https://live.european-language-grid.eu.

1 Introduction

Deliverable 2.5, consists of two parts:

- the software of Release 2 (R2) of the ELG platform[1], i.e., the modules implementing the main functionalities foreseen for this release, and the current platform contents, namely a set of functional ready-to-deploy Language Technology (LT) services and Language Resources (LRs) together with their metadata descriptions, as well as metadata entries for LT-related actors and activities;
- this report, which provides an overview of the current state of the platform, its architectural design and main components, the software and application frameworks used for their implementation, the application programming interfaces (APIs), as well as the operations and supported user interactions.

ELG R2 includes the main building blocks required for the operation of the platform:

- the user management component with all user roles defined for ELG,
- the components that support the uploading, documenting, storing, managing and downloading of resources with their documentation (metadata records),
- the components that enable the browse and search features of the catalogue and expose the metadata records to users,
- the components that are responsible for the execution of the LT services,
- and the APIs required for interacting with other layers (front-end, execution of functional services).

This report is organised as follows. Section 2 provides an overview of the ELG platform, its architecture and the development roadmap. Section 3 presents the base infrastructure on which the platform is deployed. Section 4 presents the models and policies that influence the design and operations of the ELG platform; these include the metadata model and the publication lifecycle of metadata records as well as the user management model. Section 5 focuses on the ELG platform backend, which comprises three major components: the ELG platform catalogue (in short, ELG catalogue), the ELG (platform) language processing backend services and the ELG

[1] The platform code is maintained at https://gitlab.com/european-language-grid/platform, as it is under constant development throughout the project duration. We currently discuss the appropriate software licence to apply to the code base, so that it can be shared with an open permissive licence enabling contributions from third parties.

(platform) management and support backend services. Section 6 presents the components that support access to the ELG Platform.

Finally, the ELG user manual (Figure 1), with instructions for all users (content consumers, providers and administrators) of the ELG platform (see D2.1 and D3.1), forms an integral part of this deliverable, albeit as an online document at https://european-language-grid.readthedocs.io/en/release1.1.2/. At the time of writing it includes the following contents:

- introduction and overview of the catalogue contents
- instructions for consumers (search and view of the catalogue and catalogue entries, testing of LT services and download of resources)
- instructions for providers, with subsections for functional LT services (LT tools and services in an ELG-compliant form ready to be deployed in the platform) and data resources (i.e., corpora, models, lexica, terminologies, etc.); step-by-step instructions for the creation and management of metadata records, with examples and links to additional material are included in this section
- instructions for validators & administrators
- annexes with useful links to more detailed information material (e.g., the specifications of the ELG API for LT service execution, instructions on dockerisation, etc.) and the use of the Python SDK.

The ELG manual is available online at https://european-language-grid.readthedocs.io as a living document that we continuously update following the evolution of the platform.

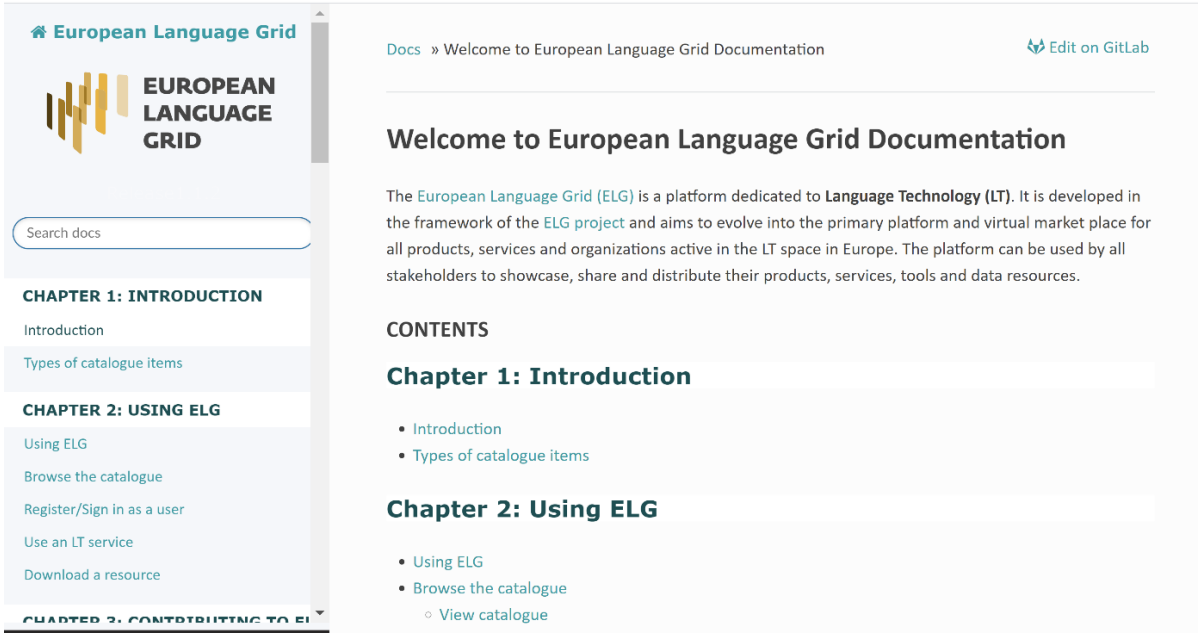

Figure 1: ELG user manual

2 The ELG Platform

2.1 Overview

The European Language Grid (ELG) platform [Rehm et al. 2020] aims to become the primary platform for Language Technology (LT) in Europe; it is developed to be a scalable cloud infrastructure that provides access to hundreds of commercial and non-commercial LTs for all European languages, including both functional (LT tools/services deployed within ELG) and non-functional (data sets, lexica, models, etc.) resources. Once the

platform is fully operational, the European LT community will be able to upload their technologies and data sets into the ELG in an easy and efficient way, to deploy them through the Grid, and to connect them with other resources. In addition, the ELG platform offers information for and about the LT domain and activities, such as information on projects, stakeholders, application and service types.

The ELG project and platform targets a wide range of users with different requirements and expectations (cf. Deliverables D2.1, D2.3 and D7.1 for more information):

- **content providers** and **developers** and **integrators**, i.e., providers and consumers of LT resources,
- **information providers** and **information seekers**, i.e., providers and consumers of LT-related (meta)-information,
- **citizens**, i.e., individuals that are interested in LT (essentially a subset of information consumers),
- **ELG platform and content administrators**, i.e., the ELG technical team that maintains and monitors the day-to-day operation and performance of the platform.

Users may identify themselves with one or more of these categories when interacting with the system, with different needs each time. In the rest of the document, we refer to them as *consumers*, *providers* and *platform administrators*. For ELG R2, the focus has been on the full set of user categories, with particular emphasis on:

- content consumers, leading to the development of the main functionalities that allow them to view and search for resources, to try out functional services and to download resources stored in the platform be it through the platform's GUIs or programmatically through the platforms APIs
- content providers, leading to the developments of the main functionalities that allow them to appropriately document and upload their language resources, tools/services as well as related entities (like research projects and organisations at different levels of granularity).

ELG platform and content administrators are also served by the current release at different levels, through modules implemented in order to support the required operations including monitoring of the use of the base infrastructure and the services hosted and deployed in ELG. Components required for the population of the platform, and deployment of services have also been implemented, as detailed in the following sections.

2.2 ELG Platform Architecture

An overview of the ELG platform architecture is shown in Figure 2. The platform consists of three main layers: the **base infrastructure**, the **platform backend** and the **platform frontend** (user interface).

The **base infrastructure** (Section 3) is the layer on which all ELG software components are deployed and run; it includes the supporting tools that facilitate development and management of the ELG platform software.

The **platform backend** layer (Section 5) consists of all the components that empower ELG, i.e. the core components (such as the catalogue database and index), the component for processing LT services and the platform support and management components (e.g., the user management module).

The **platform frontend** layer (Section 6.1) consists of (a) the static pages maintained in the Content Management System (CMS), which aim to provide information on the project and the LT domain and activities, and (b) the platform user interfaces including the catalogue UI (i.e., the browse/search page with all the metadata records, the view pages of the metadata records), the ELG interactive metadata editor for registering resources, the users dashboard, etc. The catalogue UI consumes REST services exposed by the ELG platform backend (e.g., catalogue application, LT Service execution server).

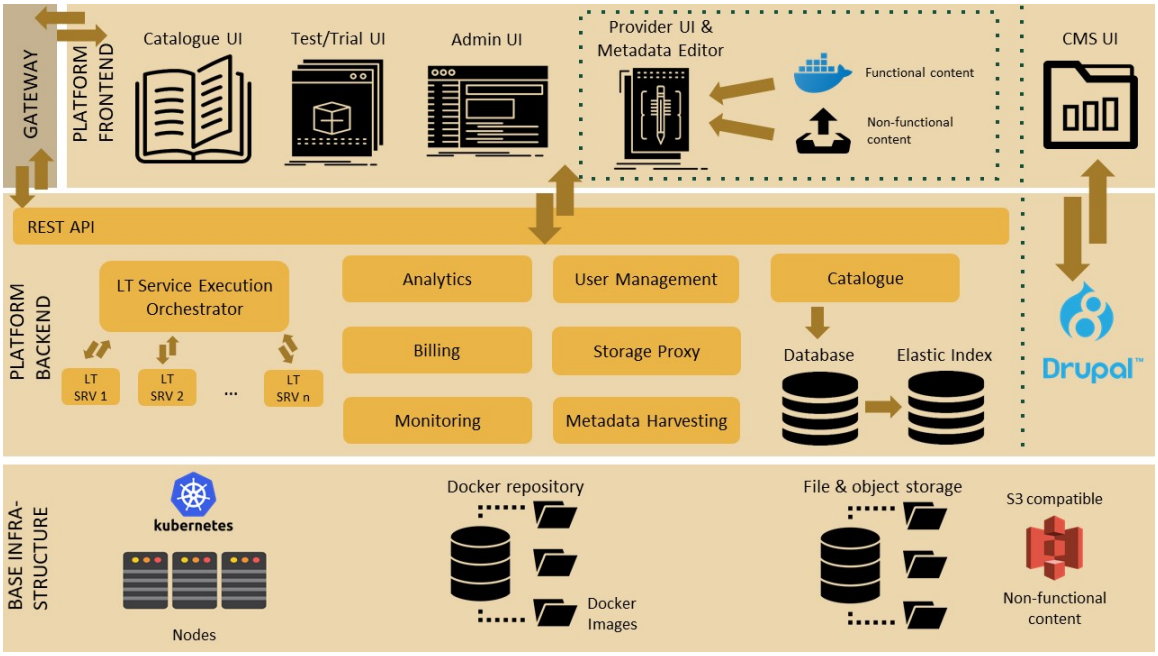

Figure 2: ELG architecture

This report focuses on the platform backend, i.e., the platform components, the ELG catalogue, the service execution layer and the supporting components. The CMS and its contents are described in D3.3.

2.3 Development Roadmap

The platform is developed and delivered in three major releases (Table 1). Two additional pre-releases had been made available before R2: a Minimum Viable Product version, presented at META-FORUM 2019 (see D7.5) and an alpha release made available on request to interested users – especially those interested in the ELG Open Call – in February 2020 (M14).

ELG platform Release 1 (R1) (D2.4; delivered M16)	• Backend components required for the operation of the catalogue:
	  ○ simple user management component,
	  ○ components supporting documentation/uploading, storing/downloading of all resource types (tools and services, data sets, etc.)
	  ○ APIs required for interacting with other layers
	• First version of the guidelines on its use and provision of resources, instructions for containerisation and invoking remotely accessible web services
	• Limited sets of tools and services and LRs

ELG platform Release 2 (R2) (D2.5; due M27)	• Updated version of the platform including the components and APIs required for running language processing services (containerised services stored in the ELG and web services via REST APIs) directly in ELG • Updated version of the guidelines on its use and provision of resources, instructions for containerisation and invoking remotely accessible web services • Updated catalogue with resources from ELG partners
ELG platform Release 3 (R3) (D2.6; due M38)	• Updated version of the platform including management and maintenance of the platform: monitoring of the platform, monitoring of remotely offered services, platform usage analytics, prototype version of user billing and payment services • Updated catalogue with resources from ELG pilots and collaborating initiatives

Table 1: ELG platform release plan

3 ELG Base Infrastructure

The base infrastructure is the layer on which the ELG platform (backend and UI) runs. It also includes tools for software development and consists of the following parts: two **kubernetes[2] clusters** (one for development and a second one for production), **source code/docker[3] repositories** and a **file and object storage module**.

More detailed information regarding the ELG base infrastructure can be found in the dedicated infrastructure deliverables (D1.1-D1.4).

3.1 Kubernetes Cluster

The ELG platform software can be divided into two parts:

- the core components, e.g., the catalogue database, the metadata index, the catalogue application, etc., which are developed and maintained exclusively by the ELG technical team, and
- the set of LT tools/services that are integrated into ELG, i.e., made available through the ELG platform, and that have been developed by consortium partners or external providers.

All components and services mentioned above run in ELG as Docker containers in a Kubernetes (in short k8s) cluster. This facilitates management and deployment (see D1.3 and D2.2). The components that run in the cluster are grouped in k8s namespaces based on their functionalities. A k8s namespace is a virtual sub-cluster, which can be used to restrict access to the respective containers that run within it. For instance, we have an "elg-core" namespace for core components such as Keycloak, which manages user authentication and nginx, the web server that acts as gateway/proxy. In the "elg-backend" namespace, we deployed the PostgreSQL database, the indexer and the REST-based catalogue application. LT services are deployed by default in the "elg-srv" namespace; however, separate dedicated namespaces are allocated for LT services for which restricted access is required.

Currently, ELG uses two clusters: the first cluster hosts the production instance[4] of the platform, while the second one[5] (which is deployed on more limited hardware resources) is used for development and testing. Both clusters run on VMs/hardware of SysEleven[6], a Berlin-based cloud service provider. SysEleven is ELG's subcontractor and was chosen in a selection process at the beginning of the project (see D1.1).

Deployment of an application in a Kubernetes cluster requires a set of configuration files (in YAML[7] format) that specify the required information, such as the Docker image location, the number of replicas that will be deployed, the ports that will be exposed to the rest of the cluster, and the deployed service's name. The latter can be used for access by other k8s applications/services. In ELG we have to manage several software components and we also need different deployments of the platform, such as a single-node deployment for development/testing, deployments in SysEleven. For these we use Helm[8], a k8s package manager that automates deployments. Helm uses (generic) templates for describing k8s resources. For each different installation/deployment, the templates are filled with the respective configuration data by running a script. The generated configuration files are then submitted to the cluster via the respective k8s API. The required scripts, helm templates and configuration files that instantiate a specific ELG deployment are kept in a separate branch in ELG's GitLab

[2] https://kubernetes.io
[3] https://www.docker.com
[4] https://live.european-language-grid.eu/catalogue/#/
[5] The ELG development cluster is accessible to consortium members only by using IP whitelisting.
[6] https://www.syseleven.de
[7] https://yaml.org
[8] https://helm.sh

repository[9], i.e., we have separate branches for production and development. For automating deployments, a continuous integration/continuous deployment (CI/CD) pipeline was created. When a source file in a branch is changed, the pipeline is triggered. It downloads the latest helm templates and scripts from GitLab and runs the scripts that inject the configuration. Finally, it deploys the final resource files to the respective k8s cluster.

All ELG services (e.g., CMS, LT service execution, catalogue backend and UI) are exposed to the public internet via an Nginx web server[10] (an ingress controller). For each service the appropriate code is inserted to the respective k8s config files that specifies the mapping between a publicly accessible URL/endpoint to the respective internal k8s backend service name (and port). Nginx is also deployed as a Docker container.

3.2 Management of Source Code and Docker Images

The platform's source code is hosted in an "organisation" account on GitLab (https://gitlab.com/european-language-grid), a code repository and development platform (for Git projects), similar to GitHub and BitBucket. GitLab was chosen because it provides numerous features for free, such as an unlimited number of private and public repositories, a built-in Docker registry for storing images and built-in CI/CD functionalities; these CI/CD pipelines are different from the ones described in Section 3.1, which are used for deploying to our k8s clusters. The GitLab CI/CD is used, for example, in order to create the images that host core components of ELG as well as images for LT services. In addition to GitLab's Docker registry, we make use of other registries, such as DockerHub[11] or Azure Container Registry[12].

All code stored within the ELG organisation account on GitLab is maintained by the ELG consortium which will ensure proper versioning of all artifacts. Additionally, though, the ELG is based on a large number of LT services, which are all packaged as Docker images, yet created and maintained by different developers, groups, or companies and hosted at these different registries (and locations). Since the images hosted in these Docker registries are not immutable and their retention policy is controlled by their respective owner, the responsibility for maintaining their versioning and availability falls to the providers. So there exist scenarios where the ELG cannot guarantee the availability of a given LT service without the support of the service provider; these are:

- Un-availability of an external image: The ELG caches downloaded images only temporarily but does not offer an additional permanent storage for images.
- Implementation of specific limitations for an image, e.g., reliance on an external proprietary licence.
- The availability of services which in turn call other LT services not hosted on the ELG infrastructure.

To mitigate scenario 1, we are investigating a combination of several solutions, such as (a) the deployment of a private/internal Docker registry for storing the images (b) keeping a backup of the services' images. For scenarios 2 and 3, they can be detected by the automatic testing of LT services that is already in place on the ELG, but they will need to be addressed by the individual LT service providers.

3.3 Storage

For the storage of files, e.g., language resources and software delivered as source code, etc., we use an S3-compatible Object Storage solution that is hosted by SysEleven. S3-compatible Object Storage provides functionalities that facilitate access to, and management of the data, organised in "buckets" with different access policies, thus ensuring access control and security. In the current implementation we have created a 'public' and a

[9] https://gitlab.com/european-language-grid
[10] https://www.nginx.com
[11] https://hub.docker.com
[12] https://azure.microsoft.com/en-us/services/container-registry/

'private' bucket. In the former, ELG keeps all publicly accessible files (e.g., logos of organisations); no authentication/authorisation is required for them. In the 'private' bucket ELG keeps files (e.g., a zip of a dataset) which are not accessible to all users, i.e., their download is constrained by the licensing terms of each dataset.

Access restrictions to an S3 object (e.g., in the case of private buckets) are applied based on the owner of the S3 account; only via this account is it possible to control the creation and management of storage buckets, to provide temporary access to requested objects if and when needed. A user interacts with the S3 Storage through the S3 Storage Proxy and the authorisation module which is part of the catalogue backend. It ensures that users have access only to the resources they own or have been granted access to. The S3 Proxy and the catalogue backend module interact with S3 based on the S3 API using the owner account.

4 Background: Metadata Model, User Model and Publication Policies

This section presents the conceptual and operational modules that influence the design of the ELG platform. We present an account of the metadata model, the publication lifecycle of ELG and the user categories.

4.1 Metadata Model

The **ELG metadata model** (or ELG-SHARE[13]) is used for the description of all entities of interest to the ELG target users (for a detailed description, see D2.3 and [Labropoulou et al. 2020]). Essentially an application profile of the META-SHARE schema, it constitutes the backbone of the ELG catalogue, which brings together language processing services and tools, LRs (data sets of different types and media, models, lexica, terminologies, etc.) as well as actors and activities related to LT (Figure 3).

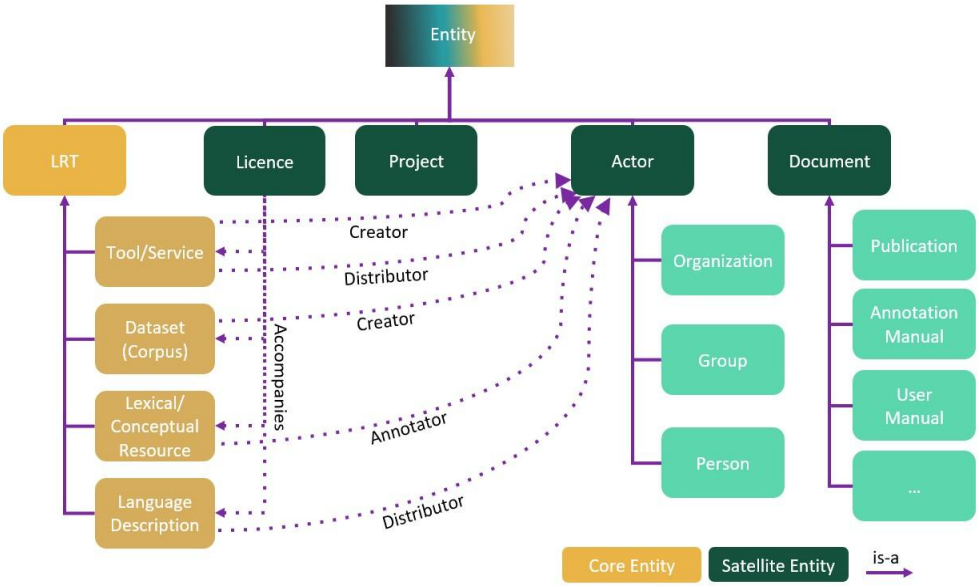

Figure 3: Overview of the ELG-SHARE entities

The model caters for the description of ELG core entities, i.e.,

[13] The ELG metadata model builds upon, extends and updates META-SHARE and its application profiles. Modifications have been made (a) in the contents (e.g., addition of elements for GDPR, improvement of the description of LT services and models), in response to the project requirements and more recent developments in the metadata area at large, (b) in the implementation, which now combines the XSD approach used for META-SHARE profiles with the deployment of two ontologies, namely META-SHARE (https://w3id.org/meta-share/meta-share/) and OMTD-SHARE (https://w3id.org/meta-share/omtd-share/), for the definition of the elements and values. For a detailed description, see Deliverable D2.3 and [Labropoulou et al. 2020].

- *LT tools/services*, covering all software that performs language processing and/or any LT-related operation (e.g., basic processing tools, applications, web services etc. that perform annotation, machine translation systems, speech recognisers, etc.).
- *Corpora* (data sets), defined for our purposes as structured collections of pieces of data (textual, audio, video, multimodal/multimedia, etc.), typically of considerable size and selected according to criteria external to the data (e.g., size, type of language, type of producer or expected audience, etc.) to represent as comprehensively as possible the object of study.
- *Lexical/conceptual resources*, i.e., resources (such as terminological glossaries, word lists, semantic lexica, ontologies, gazetteers, etc.) organised on the basis of lexical or conceptual units (lexical items, terms, concepts, phrases, etc.) with their supplementary information (e.g., grammatical, semantic, statistical information, etc.).
- *Language descriptions*, i.e., resources aiming to describe a language or some aspect(s) of a language via a systematic documentation of linguistic structures (e.g., computational grammars, statistical and machine learning-computed language models).

The ELG model also provides for entities involved in the production and usage of LTs/LRs and, in general, LT activities, i.e., *actors* (further distinguished into *organizations*, *groups*, *persons*), *documents* (e.g., user manuals, publications, etc.), *projects* and *licences/terms of use*. The model includes a large number of metadata elements grouped along three key concepts: *resource type*, *media type* and *distribution*. The *resource type* element distinguishes LRTs in the four classes presented above. *Media type* refers to the form/physical medium of a data resource (or of its parts, in the case of multimodal resources), i.e., text, audio, image, video and numerical text (used for biometrical, geospatial and other numerical data). Finally, *distribution*, following the DCAT vocabulary[14], refers to the physical form of the resource that can be distributed and deployed by consumers; for instance, software resources may be distributed as web services, executable files or source code files, while data resources may be distributed as PDF, CSV or plain text files, or through a user interface. Administrative and descriptive metadata are mostly common to all LRTs, while technical metadata differ across resource and media types as well as distributions. Figure 4 provides an example with part of the metadata model.

The abundance of information in the schema makes the task of creating metadata records quite tedious. To ensure flexibility and uptake, we distinguish metadata elements *into mandatory*, *recommended* and *optional*. Minimal metadata records with only the mandatory and strongly recommended elements are, thus, acceptable. The criteria used in ELG to determine the optionality status of elements include: required for discovery, especially features considered of high interest to ELG consumers (see D2.1, D3.1); considered indispensable for accessing the resources and, in the case of functional services, ensuring proper deployment through the platform; supporting usage of resources; deemed valuable for experiments and projects and essential for achieving interoperability with existing metadata schemas used in the wider LT and neighbouring communities.

The design and the implementation of the model have been completed in R1 (with the exception of the billing module). Nevertheless, we have made updates and changes for R2 and anticipate other updates also for upcoming releases of the platform, taking into account user feedback, technical requirements of the platform (as its development progresses) and new requirements emerging from pilot and collaborating projects.

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

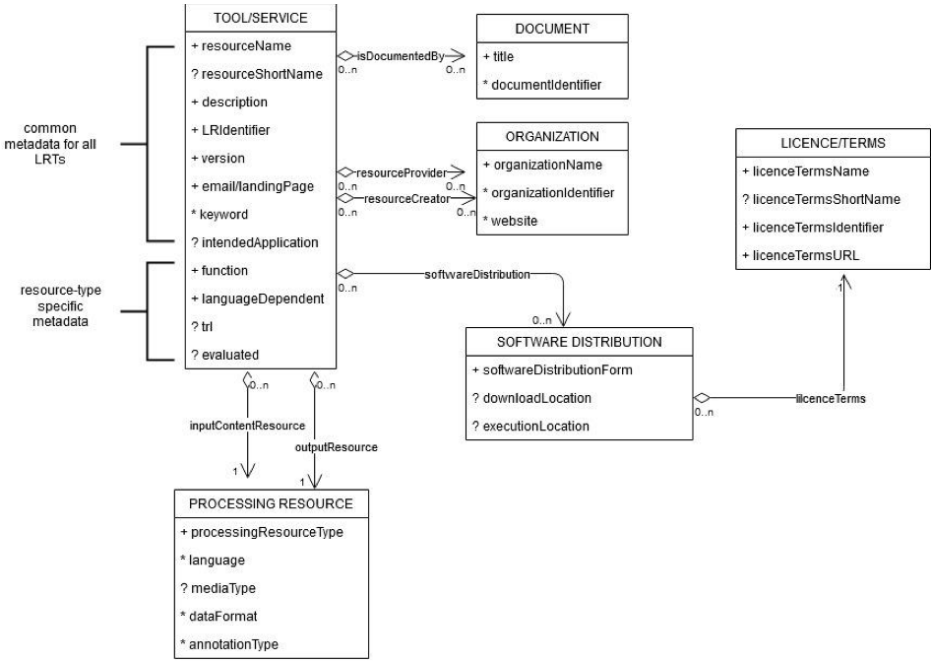

Figure 4: Excerpt of the ELG metadata model (focusing upon tools/services)

The model is implemented in the form of an XML Schema Definition (XSD)[15] . Its elements are linked to entities from two ontologies, namely the META-SHARE[16] ontology, which includes the majority of elements and controlled vocabularies, and the OMTD-SHARE ontology[17], reserved for the controlled vocabularies of LT categories (also referred to as LT taxonomy), data formats and methods. Each metadata element and value has an identifier which contains the Internationalised Resource Identifier (IRI) of the corresponding entity. This approach contributes to the FAIRness[18] of the metadata model, facilitates linking to metadata records in other catalogues, and supports import/export in the JSON-LD serialisation format, which increases the visibility of ELG metadata records overall. The use of XSD enables us to transform the metadata schema easily into an entity-relationship model[19], thus facilitating its documentation and conversion into the relational database used in the ELG catalogue backend. Ontology entities were automatically converted into XML elements, as used in the XSD. This approach enables an easy update of the schema alongside the evolution of the ontology. All relations, labels and definitions are copied into the XML elements with the same script, feed the display labels in the landing pages of the metadata records, and are also utilised in the metadata editor to display labels, and helptips.

4.2 Publication Lifecycle of Metadata Records and Publication Process

The publication lifecycle of a metadata record (Figure 5) draws upon the principles of META-SHARE, and is divided into the following states:

- **new item**: A provider creates an item, by creating a metadata record through the interactive editor or by uploading a metadata file (see Section 5.1.5) and, optionally, content files.

[15] The ELG metadata schema is available at https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema, accompanied with documentation, and metadata record templates for all resource and media types.
[16] https://w3id.org/meta-share/meta-share/
[17] http://w3id.org/meta-share/omtd-share/
[18] The FAIR Guiding Principles for scientific data management and stewardship aim to improve the findability, accessibility, interoperability, and reuse of digital assets. The principles emphasise machine-actionability (i.e., the capacity of computational systems to find, access, interoperate, and reuse data with none or minimal human intervention) because humans increasingly rely on computational support to deal with data as a result of the increase in volume, complexity, and creation speed of data, see https://www.go-fair.org/fair-principles/.
[19] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

- **draft:** Used as the initial state for metadata records created with the interactive editor. At this state, providers do not have to fill in all mandatory elements; only compliance as to the data type of the elements is checked (e.g., elements that take URL values must be filled in with the accepted pattern).
- **syntactically valid:** The metadata record complies with the ELG metadata schema and all mandatory elements are filled in. The provider can still continue to edit it until satisfied with the description and can then submit it for publication.
- **submitted for publication:** As soon as the provider submits it for publication, the record becomes no longer editable and is assigned for validation.
- **under validation:** Depending on the item type and the source (see Table 3: Example endpoints of LT processing services), the item is validated by designated users at the metadata, technical and legal level. The validation aims to check the consistency of the description and, where required, the technical compliance of the item to the ELG specifications; it doesn't include any qualitative evaluation. The validation is currently performed by the ELG consortium members.
- **(approved and) published:** for finalised metadata records (i.e., approved by validators); published records are available in the public catalogue.

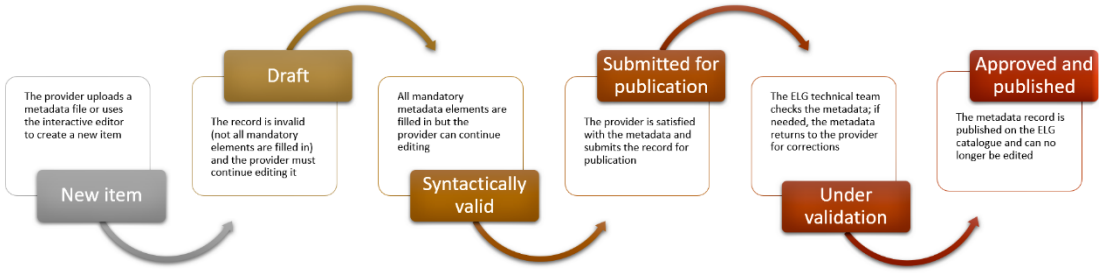

Figure 5: ELG publication lifecycle

Published records cannot be changed and cannot be deleted[20] in order to ensure reproducibility of scientific results. Instead, new versions of the same resource can be added, linked to previous version(s) and marked as replacing them, so that users are redirected to the most recent version.

Type of records	Validation type		
	Metadata	**Technical**	**Legal**
Harvested metadata	N/A	N/A	N/A
Metadata records uploaded by ELG admin	N/A	N/A	N/A
"Metadata only" records[21]	Yes	N/A	N/A
ELG compatible services	Yes	Yes	Yes
LRTs uploaded (hosted) in ELG	Yes	Yes	Yes

Table 2: Validation types by source and type of metadata record

[20] Published metadata records can be unpublished and deleted in exceptional cases upon request. Such cases mainly have to do with the reporting of violation of IRP rights and other legal issues, and technical malfunctions.
[21] "Metadata only" records are records for projects, organisations but also for LRTs that are not uploaded in ELG, i.e., they are offered just for information purposes.

Metadata records cannot be reverted to previous states, with the exception of

- records submitted for publication: when the validators find some issues, they are returned to the "syntactically valid" state in order to be editable again by the provider of the metadata records;
- published records but only in exceptional cases and only by administrators. Normally, published records must not be changed, as mentioned above. However, metadata records that have been imported in the ELG catalogue through specific operations managed by the ELG system administrators (such as the bulk import of organisations with minimal information described in Section 5.1.5.3) can be returned to previous states so that they can be edited by assigned individuals.

In addition, administrators have the right to revert all records to previous states, with the exception of the "draft" state. Once a record is syntactically valid, it can no longer return to the draft state.

The publication procedure for resources and the user model of ELG (Section 4.3) determines which users can access metadata records in each of the above states.

Metadata records can be submitted to the ELG platform only by users who have registered in the platform and given the appropriate authorisation; these can be individuals who wish to share their own LRTs or act as representatives of an organisation to which they are affiliated and wish to upload LRTs developed by that organisation. The registration procedure for users includes consent to the ELG terms of use, whereby they commit to provide resources for which they have the legal right to share them and that they are technically "safe". The provision of resources by registered users who are logged in allows for giving credit to the specific providers, but also entrusting them with all responsibility for any issues that may rise with regard to them.

Authenticated and authorised users must provide at least a metadata description of their resource according to the ELG schema. They can also upload the resource itself, in which case it will be stored and preserved according to the Data Management Plan (D5.1).

Further requirements apply specifically to LT services in order to be integrated as ready-to-deploy in the ELG platform (see Section 5.1.5.2). For their publication, they go through a validation process that aims to ensure technical validity of the service itself. The process is performed by the ELG technical team, and includes checking the technical metadata required for the import and deployment of the service, and the addition of ELG-specific information that will enable the execution of the service in the ELG platform. Before publishing a tool/service, the validator checks that (a) it follows the ELG technical specifications; (b) its technical metadata are as required; (c) it can be executed in the ELG platform. Similarly for data resources uploaded in ELG, the validators make sure that the content files are conformant to the description presented in the metadata record and that they contain no malicious parts.

The ELG catalogue can be populated with metadata records harvested from other sources, following specific agreements (see D5.1). In this case, the resources may remain in their source repository or be transferred to the ELG platform storage, depending on the negotiations with each source. In any case, ELG validators can check the metadata records before or after their import in the catalogue and decide to publish them.

Finally, information (metadata records) for LT actors and activities (e.g., LT companies, research organisations active in the LT areas, projects, etc.) is collected through their relations with the core entities populating the catalogue as well as through other processes (e.g., collection from NCCs, surveys, etc.). These records can then be displayed in the public catalogue.

4.3 User Categories

For the second release of the ELG platform, the user management model comprises a set of broad user categories and roles with access policies defined in response to the functionalities of the current release:

- **Unregistered users**: Users who are not registered with ELG and who are, hence, not logged in, can browse the catalogue, search for resources and view their descriptions; they can download resources with open licences; however, they are not allowed to register a resource, or run a published LT service; they do not have access to draft, syntactically valid or submitted resources.
- **Registered users**: These are the users with accounts in the ELG user database. Each registered user can be assigned one or more of the following roles:
    - **Consumer:** In addition to the access rights of unregistered users, they can also try out LT services with the GUIs provided in the platform, or in command mode.
    - **Provider:** Providers have all the access rights of a consumer and are also able to register resources. They have editing rights for the resources they have registered until they submit them for publication.
    - **Validator:** Users who validate resources and confirm their publication. They have the same rights as a consumer and advanced rights for the resources they validate. There are three types of validators:
        - The legal validator assesses the legal standing of an LR before it can be published.
        - The metadata validator validates the content of “metadata only” records (cf. footnote 21) before they are published.
        - The technical validator ensures that any ELG hosted resource or functional service functions as intended via the ELG infrastructure.
    - **Content manager**: The ELG team that is responsible for the smooth transition of a record submitted for publication to its published state, including the assignment of validators for ELG hosted resources and functional services; content managers have full access to all resources.
    - **Administrator:** The ELG team that develops and maintains the platform; administrators have full access to all resources and functionalities of the platform.

When users register at the ELG platform, they are automatically assigned the “consumer” role. Roles with more privileges are assigned only by technical administrators. Users can request to obtain the "provider" status through their user profile. Currently, there is no procedure for requesting the validator role, since this is restricted to the ELG technical team and managed by the administrators within the consortium.

5 ELG Platform Backend

The ELG platform backend comprises three major components: the ELG platform catalogue (in short, ELG catalogue), the ELG (platform) language processing backend services and the ELG (platform) management and support backend services. They are described in detail in the following subsections.

5.1 ELG Catalogue

In the following subsection, we present the catalogue backend components, i.e., the catalogue application, the database, the indexing and search mechanism, the upload and download mechanism and procedure for content files stored in ELG, and, finally, discuss the population of the catalogue from the technical point of view.

5.1.1 Catalogue Application

The catalogue backend application is built with Django (version 3.0.9) and the Django REST Framework[22]. It mainly acts as a RESTful backend providing all necessary functionalities for database management and user access control to the various aspects of the ELG platform.

The backend defines a set of REST API endpoints used for handling requests from the catalogue UI (described below in Section 6.1.1) and other platform modules, such as the LT service execution server (described below in Section 5.2.3). The REST endpoints are controlled by the user authorisation policies, in order to distinguish between those that are accessible by everyone (e.g., the retrieval of a published metadata record) and those that require user authentication and a specific user role (e.g., users can create metadata records if they are authenticated and have been assigned the "Provider" role).

A key functionality of the catalogue backend is the management of metadata records through CRUD, i.e., create, read, update and delete operations. Currently, the defined REST API endpoints handle requests for

- creating metadata records, by uploading XML files adhering to the ELG schema or by using the interactive metadata editor;
- updating metadata records via the interactive metadata editor;
- retrieving a metadata record in JSON format, to be rendered by the frontend (catalogue UI);
- downloading a resource in accordance with its licensing terms;
- authorizing service execution (see Section 5.2.3);
- collecting usage information (see Section 5.3.2.1).

5.1.2 Database

A PostgreSQL database is used for the persistent storage of metadata records, which are imported in the database with one of the two initial states of the publication lifecycle described in Section 4.2, depending on the mode of import. The database can be populated through three different functionalities: the interactive editor, the upload of metadata records and harvesting of metadata from other repositories (see Section 5.1.5).

If a record is inserted into the database through the upload functionality, its initial state is the "syntactically valid" status. This happens because, during the upload, the catalogue backend checks whether the metadata description adheres to the ELG model. If not, the record is rejected and the user is informed of the XML validation errors, else the upload is successful and the record is saved with the status "syntactically valid".

When a record is created with the editor (see also Section 6.1.2), the initial state can be "draft" or "syntactically valid". The user can save the record as "draft", whereby the backend bypasses the validation of mandatory fields and only validates the types of metadata fields. A record can also be saved as "syntactically valid"; in this case, as in the upload of metadata records, if the record does not adhere to the ELG model, it is rejected and the validation errors are returned in JSON format;otherwise, the record is saved successfully in the database.

Harvested records are imported in the database after automatic validation against the ELG schema with the "published" state. Since they are already published in other repositories, and, thus, considered trustworthy, there is no reason to submit them to the human validation process.

In the first two cases the curator assigned to the record is the user that created or uploaded it. In the case of harvesting, the curator is the ELG system itself.

[22] https://www.django-rest-framework.org

5.1.3 Indexing and Search

Published metadata records can be searched through the catalogue UI, on a subset of the metadata information indexed by Elasticsearch[23], either by search queries or using faceted search. This subset includes

- for all entities (LRTs, projects, organisations): name or title, short name or title, description, keywords;
- in addition,
    - for organisations specifically: country of registration, name of parent organisation (if it exists);
    - for LRTs: resource type (i.e., Tool/Service, Corpus, Lexical/Conceptual Resource, Language Description), languages, licences, condition of use, access rights, intended application, version;
    - and for LT tools/services, also the service function.

The backend returns results in JSON format, when queried by the catalogue UI component.

By default, the search functionality matches whole words in the fields indexed for full text search, using the OR operator. In addition, query expressions, passed in the search parameter, are also supported. Such expressions utilise the Lucene Query Syntax[24] and can be used for advanced queries such as exact phrase matches (e.g., "bilingual corpus"), fuzzy or proximity search, term boosting, etc.

Apart from the catalogue UI, which is available to all users for the published metadata records, providers can manage their metadata records via the dashboard UI. The dashboard UI (see Section 6.1 and D3.3) also includes the search functionality on a subset of the metadata record information indexed by Elasticsearch, either by free text search or by using faceted filters. In this case, the following elements are used: name or title, publication status of the metadata record, entity type (whether LRT, project, organisation, etc.), creation and last update dates, and, for LRTs, their version and whether they have data uploaded at the ELG infrastructure.

As for the catalogue UI, the backend returns results in JSON, when queried by the dashboard UI component. Advanced search functionality with the Lucene Query Syntax is not supported for the dashboard index yet.

5.1.4 Upload/Download and Storage of Non-Functional Content

Non-functional content uploaded to the platform (together with a metadata record) is stored in a private S3 storage bucket, as described in section 3.3. Data uploads are performed through an S3 Storage proxy (see Figure 2). The proxy communicates with the catalogue backend which authorises the upload. This way, we allow only owners of a metadata record to upload the data to be attached to this record.

A similar process, albeit without the S3 proxy, is used for managing access granted to users when downloading datasets. More specifically, upon receiving a download request from the catalogue UI, the backend checks the user's access permissions depending on the user role and the status of the metadata record. If the metadata record is published, the download is authorised only for users that conform to the requirements imposed by the licensing terms of the specific dataset. For example, openly licensed data can be directly downloaded by users without further interactions. On the other hand, for data licensed upon condition of explicit consent, the user is shown the licence document and has to agree to the terms; the backend decides whether this request is eligible taking this consent into account: if the user accepts the terms, it returns a temporary pre-signed S3 download URL to the catalogue UI; otherwise, an appropriate "permission denied" response is returned.

[23] https://www.elastic.co
[24] https://lucene.apache.org/core/2_9_4/queryparsersyntax.html

If the metadata record is not published, only its curator, the assigned validators and administrators can download the data; thus, the catalogue backend allows the download of the resource directly, without checking any of the requirements imposed by the accompanied licensing terms.

The catalogue currently accepts compressed files in .zip format and stores information about the size, date of upload and a hash digest of the object that reflects changes to its contents; an ETag header is used for including the hash digest to the http request/responses). Every uploaded resource, related to a metadata record, is stored with a unique key, consisting of the metadata record's storage object identifier (a UUID assigned to each metadata record upon creation) and a filename.

5.1.5 Catalogue Population

5.1.5.1 Registration of Language Resources and Technologies

Individuals who want to make their LRTs available through ELG first have to register and ask for "provider" status through their profile. As soon as their application request is approved by an administrator, they will be able to register LRTs via the ELG interactive editor or the "Upload metadata file" functionality.

The interactive editor (see also Section 6.1 and D3.3) supports users in creating new metadata records, as well as editing/updating existing ones. It supports the minimal ELG schema, i.e., includes the mandatory and recommended fields. Validation is performed both by rules at the front-end side and the backend one, which check the appropriate datatype and mandatoriness for each element. The interactive editor communicates with the catalogue backend with JSON messages via the respective REST API. The JSON messages are generated based on the Django database objects (i.e., objects used to interact with the database) and the respective Django REST Framework (DRF) serialisers that generate JSON from these objects.

In the "upload metadata files" functionality, users are prompted to upload a file with the description of their resource in XML format; the XML file must comply with at least the minimal version of the ELG schema. In order to facilitate the registration process, pre-filled metadata templates are available at the ELG GitLab repository[25]. The XML file is then validated against the ELG XSD and also against a number of rules implemented as part of the import procedure, which check syntactic and partial semantic integrity of the description. Validation errors are reported to the user for correction. If the file is valid, it is imported and assigned the status "syntactically valid" (cf. Section 4.2).

We have also started implementing a publicly available REST endpoint toolkit for metadata validation, which can be quite useful in the process of uploading metadata files. Providers who wish to upload metadata files, can validate their XML files via this endpoint in order to ensure that they comply with the ELG metadata schema, before proceeding to the metadata upload step. Thus, for instance, users who wish to create metadata records for testing purposes can do so without having to login and register mockup records. The XML Validator loads the current XSD schema. Changes to the XSD schema require that the validator component is updated. This is an action performed only by ELG administrators.

This functionality is publicly available for single XML files or zipped archives, for batch validation at https://live.european-language-grid.eu/catalogue/#/validate-xml. The validation result is returned to the user as JSON (Figure 6).

[25] https://gitlab.com/european-language-grid/platform/ELG-SHARE-schema

s


```
{
  "info": "file 'corpus1.xml' valid"
}

{
    "info": "file 'tool-service1.xml' not valid",
    "errors": [
        {
            "line": 93,
            "error": "Element 'SoftwareDistributionForm': [facet 'enumeration'] The value
'http://w3id.org/meta-share/meta-share/docker' is not an element of the set
{'http://w3id.org/meta-share/meta-share/dockerImage', 'http://w3id.org/meta-share/meta-
share/executableCode', 'http://w3id.org/meta-share/meta-share/library',
'http://w3id.org/meta-share/meta-share/other', 'http://w3id.org/meta-share/meta-
share/plugin', 'http://w3id.org/meta-share/meta-share/sourceAndExecutableCode',
'http://w3id.org/meta-share/meta-share/sourceCode', 'http://w3id.org/meta-share/meta-
share/unspecified', 'http://w3id.org/meta-share/meta-share/webService',
'http://w3id.org/meta-share/meta-share/workflowFile'}."
        }
    ]
}
```


Figure 6: Validation output

All metatada records that are registered in ELG undergo the publication lifecycle described in Section 4.2.

5.1.5.2 Registration and Publication of ELG-compliant LT Services

For R2, the process of registering an LT service to ELG is as follows:

1. The LT provider creates a Docker image that contains an LT tool exposed via an ELG-compliant REST service; the Docker image has to be uploaded to a Docker registry (e.g., GitLab, DockerHub). In some cases two images are required; one that contains the LT tool and one that implements the ELG API and calls the LT tool via its own custom API . For more information see section 5.2.3.
2. The provider creates a metadata record and submits it for publication (see Section 4.2).
3. The administrators assign the metadata record to a validator; the record is now visible to the assigned validators.
4. The LT service is deployed to the k8s cluster by creating a configuration file[26] and uploading it to the respective GitLab repository. The CI/CD pipeline responsible for ELG deployments (Section 3) will automatically deploy the new service. If requested (by the LT provider), before creating the YAML file, a separate dedicated namespace is created for the LT service[27]. A user with the “technical validator” role assigns to the LT service:
5. The k8s REST endpoint that will be used for invoking it.[28]
6. An id for the service that will be used to call the LT Service via the respective REST API (Section 5.2.1).

[26] https://gitlab.com/european-language-grid/platform/infra.git
[27] A separate namespace is usually allocated for a set of services.
[28] The endpoint follows this template http://{k8s service name for the registered LT service}.{k8s namespace for the registered LT service}. svc.cluster.local{the path where the REST service is running at}.

7. The validator can use the try-out UIs or the command line to test the service; integration issues are identified and solved in collaboration with the LT provider; as a communication medium we use dedicated Slack[29] channels. This iterative process is continued until the tool/service is correctly integrated. This procedure requires access to the k8s cluster for the validator.

When the LT service works as expected, the validators approve the metadata record and this is published at the public catalogue and, thus, available to all ELG users.

5.1.5.3 Registration of other Entities

The population of the ELG catalogue with metadata records for the LRTs generates additional metadata records for related entities, i.e., entities that are mentioned in these metadata records, such as organisations and individuals that have developed the resource, projects that have funded them, licences with which they are distributed, etc. These records contain minimal information, i.e., only what is required by the ELG schema to describe a related entity, i.e., a name/title, and, optionally, an identifier and information that could uniquely describe it (e.g., email for persons, website for organisations, a URL with the text for licences, etc.).

This process often leads to the generation of duplicate entries for related entities. To tackle this, the ELG catalogue application offers a retrieval mechanism that matches records of entities already imported in the database. The criteria for the retrieval are based, in priority order, on (a) the ELG internal identifiers, and (b) on the metadata elements that have been declared in the ELG model as unique identifiers for each entity type (e.g., the website for organisations and the email address for persons).

Furthermore, we have exploited existing sources with openly available metadata records, such as the SPDX list of licences[30] and the CORDIS dataset of projects[31], in order to pre-populate the ELG catalogue with entries for the related entity types. This (a) facilitates the registration process of the LRTs, and (b) provides additional information without requesting them to be entered by the metadata providers.

In R2, we have initiated another form for populating the ELG catalogue with bulk lists of metadata records with minimal information. This has been used for the population of the catalogue with companies that use or develop LT applications, thus enabling us to quickly create a “yellow pages” for organisations active in the broader LT area. For this, we merged lists of organisations from various sources, together with minimal information on them – mainly contact data and key terms that describe their LT-related activities. This list, divided into sublists by country, was checked by the respective National Competence Centers (NCCs). The revised list, with more than 900 records, was uploaded into the ELG catalogue. The records that have been added in this way can be claimed by interested individuals that work for these organisations in order to be further enriched (Figure 7). When a person claims a metadata record, the ELG administrators are notified and can approve or reject the claim, taking into account the professional email of the user; if the claim is approved, the metadata record is unpublished and assigned to the user for further editing. Once the user finishes the editing, the record is submitted for publication and goes through the normal publication procedure.

[29] https://slack.com
[30] https://spdx.org/licenses/
[31] https://data.europa.eu/euodp/en/data/dataset/cordisH2020projects

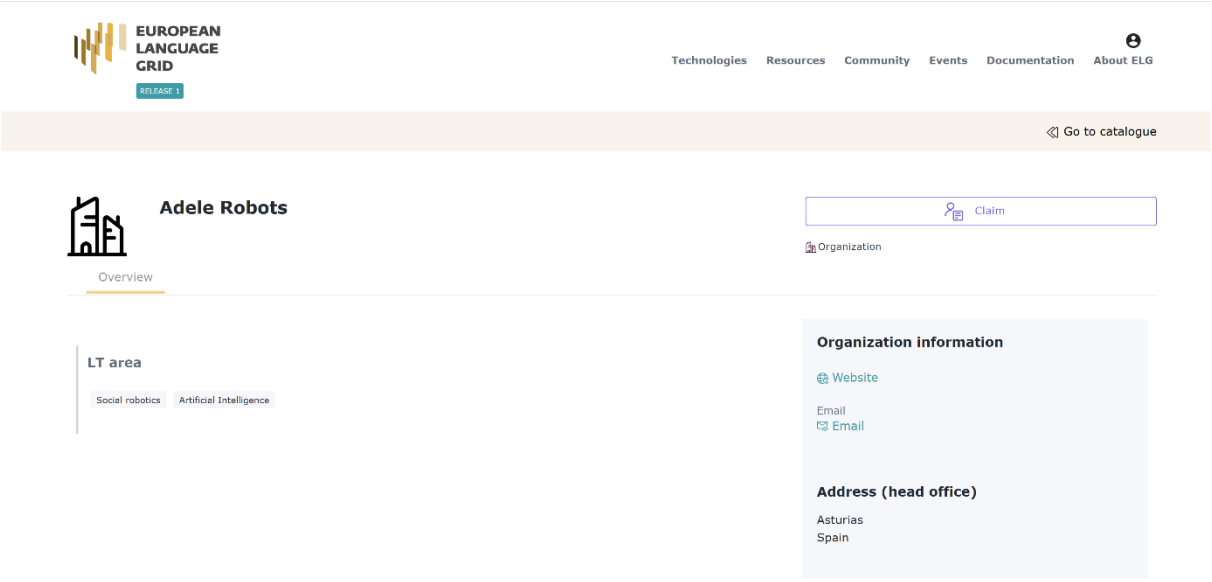

Figure 7: Claim of metadata records

5.1.5.4 Harvesting

As an additional channel for metadata population, the ELG catalogue implements an OAI-PMH client for harvesting metadata from other infrastructures. The process of harvesting first requires the registration of a third-party provider as an "OAI-PMH Provider" into the ELG catalogue. As soon as communication is established, the third-party provider shares their OAI-PMH endpoint, which ELG will call at regular intervals (currently once a week) in order to harvest their exposed metadata. The ELG harvester accepts metadata records compliant with the ELG metadata schema; the ELG team supports the providers in the process of conversion of their original schema into the ELG one. Currently, we accept OAI-PMH harvested metadata from ELRC-SHARE and LINDAT-CLARIAH-CZ (see D5.2 for more details). The harvested metadata get the "published" status and are immediately visible at the ELG public catalogue.

5.1.5.5 Current Contents

At the time of writing, the ELG catalogue includes the following set of public metadata records:

- 254 tools/services: these are all ready-to-deploy services provided by the ELG consortium partners (see D4.2 for a detailed report)
- 2993 data resources, which consist of 2244 data sets (corpora), 742 lexical/conceptual resources, 7 language descriptions (grammars and models) (see D5.2 for more information)
- 25 projects, including the Open Call I projects (see D6.2),
- 922 organisations.

5.2 LT Processing Services Execution Backend

One of the main goals of ELG is to provide a substantial number of LT services through the platform. To accomplish full integration and deployment of the services, we rely on containerisation and the specification of generic LT processing APIs. For accessing the LT services from the ELG catalogue, from the command line or by using any programming language, a common public REST API has been made available (Section 5.2.1). The LT service execution is powered by Knative, a platform installed on top of K8s that provides scale-down-to-zero and autoscaling (Section 5.2.2).

5.2.1 Internal LT APIs

The LT tools we currently work with and integrate into ELG broadly fall into one of the following categories: Information Extraction (IE), Text Classification (TC), Machine Translation (MT), Automatic Speech Recognition (ASR) and Text to Speech Generation (TTS). For each category a specific API was defined (Task 2.5) with the aim of standardizing the invocation of these services and their integration into the ELG platform. This API specification and the adoption of Docker images for packaging the LT tools solves several interoperability issues and facilitates their deployment in the ELG platform. The ELG requirements for integrating an LT service are as follows.

Expose an ELG compatible endpoint: An application that exposes an HTTP endpoint for the provided LT tool should be created. The application should consume (via the HTTP endpoint) requests that follow the ELG JSON format, call the LT tool and produce responses, again in the ELG JSON format. For instance, for an IE tool, its HTTP endpoint should accept POST requests in the JSON-based format presented in Figure 8.

```
{
  "type":"text",
  "params":{...}, /* optional */
  "content":"The content, as a string inline",
  // mimeType optional - this is the default if omitted
  "mimeType":"text/plain",
  "features":{ /* arbitrary JSON metadata about this content, optional */ },
  "annotations":{ /* optional */
    "<annotation type>":[
      {
        "start":number,
        "end":number,
        "features":{ /* arbitrary JSON */ }
      }
    ]
  }
}
```

Figure 8: JSON input

Most parts of the request are optional, only “type” and “content” are required. The “features” and “annotations” fields may be useful for services that can build on partially-annotated output from other services. The “start” and “end” of each annotation specify the position of the annotation within the text. The JSON response for an IE tool should be in the format presented in Figure 9. The response contains sets of annotations and, for each annotation, the start and end offsets are included together with any other available features.

Detailed documentation and examples for all the different cases of requests and responses is available through the ELG documentation[32] and in D4.1. Further API classes are under development following discussions with the pilot projects managed and supported in WP6.

Dockerisation: The application should be dockerised and the image uploaded into a Docker Registry, such as the GitLab, DockerHub or Azure container registry. Open-source tools would typically be deployed to a public registry, but for closed-source tools the ELG cluster can be configured with credentials to fetch the image from a private container registry. Three different options for the provision of LT tools are supported:

[32] https://european-language-grid.readthedocs.io/en/release1.1.2/all/A3_API/LTInternalAPI.html

- **LT tools packaged in one image:** One Docker image is created that contains the application that exposes the ELG-compatible endpoint.
- **LT tools running remotely outside the ELG infrastructure:** For these tools a proxy image is created that exposes one (or more) ELG-compatible endpoints; the container communicates with the actual LT service that runs externally, i.e., outside the ELG infrastructure.
- **LT tools requiring an adapter:** For tools that already offer an application that exposes a non ELG-compatible endpoint (HTTP-based or other), a second adapter image should be created that exposes an ELG-compatible endpoint and that acts as proxy to the container that hosts the actual LT tool.

```
{
  "response":{
    "type":"annotations",
    "warnings":[...], /* optional */
    "features":{...}, /* optional */
    "annotations":{
      "<annotation type>":[
        {
          "start":number,
          "end":number,
          "features":{ /* arbitrary JSON */ }
        }
      ]
    }
  }
}
```

Figure 9: JSON output

5.2.2 Kubernetes and Knative

All LT services are packaged as Docker images that follow the ELG specifications. The LT containers as well as the core components of the platform (e.g., the catalogue backend) run in a k8s cluster within predefined namespaces (e.g., for services, the “elg-srv” namespace). For security and isolation reasons, an LT provider can request a dedicated namespace with restricted access for its services. For instance, LT services provided by Expert Systems run in the “elg-srv-expsys” namespace, with private registry credentials managed using the standard k8s “secret” mechanism. When an LT service is deployed using an adapter, the LT service and the adapter containers are deployed in the same k8s pod[33].

On top of k8s, we installed Knative[34] to efficiently manage the workload of the various LT services and to avoid draining of the available hardware resources. A substantial number of LT services are planned to be integrated into the ELG platform, many of which require a significant amount of hardware resources (CPU and/or memory). It is impossible to keep all of them up and running all the time. Knative addresses this issue by offering a scale-to-zero functionality; i.e., the number of running containers (replicas) of a certain LT service is scaled down to zero for the time period that the service is not used by anyone. Also, Knative offers autoscaling functionalities, i.e., when a user initiates a processing request for a service, knative receives this request and starts the corresponding container (if replicas = 0) and forwards the request to it. Depending on the number of requests for this service and based on the configuration, it spawns additional containers. An example of such a configuration, for an MT service (French-to-English) created by Charles University, is given in Figure 10.

[33] A pod is a group of containers deployed together on the same host. Containers within a pod share a common network stack and can communicate with one another via the local loopback address 127.0.0.1
[34] https://knative.dev

```
image: "registry.GitLab.com/european-language-grid/cuni/srv-translation-t2t/fr-en"
noIngress: true
containerport: "8080"

scalability: "dynamic"
minScale: "0"
maxScale: "2"
concurrency: "500"

requests_memory: "2048Mi"
requests_cpu: "5m"
limits_memory: "2560Mi"
limits_cpu: "1000m"
```

Figure 10: Knative configuration example

The “image” field specifies the location of the LT image, “containerport” defines the port where the ELG-compliant REST service hosted in the container runs. The parameters “minScale” and “maxScale” define the minimum and maximum number of replicas that can run for this service. “Concurrency” specifies a target number of concurrent requests to be served by each replica of the LT container; a sustained load above this level will cause additional instances of the container to be spawned, subject to the configured “maxScale” limit. The last four fields specify hardware requirements/limitations for the containers that run for this LT service[35].

5.2.3 LT Service Execution Server and External LT API

The LT Service Execution Server is responsible for orchestrating LT processing. This server is different from the catalogue backend application. It implements all functionalities relevant to LT processing. In this way, we achieve modularity and a clean separation of platform functionalities. The LT Service Execution Server provides:

- a client/connector for sending HTTP processing requests to and receiving HTTP responses from the backend LT services;
- a mechanism for retrieving the k8s REST endpoint of an LT service; this endpoint is configured during the registration of the service and is kept alongside other information in the respective database table;
- a mechanism for limiting access to LT services; i.e., for each registered ELG user36, we keep a daily usage record containing the number of requests/calls that have been submitted to the LT service execution server as well as the total size of these requests (in bytes); if the user exceeds the predefined limits/quotas, execution is not allowed anymore;
- a common REST API used for calling an integrated LT service from the catalogue frontend or from the command line. Table 3 shows the current endpoints of the API37. {Domain} is the domain name of the deployed ELG platform (e.g., the production ELG domain name is “live.european-language-grid.eu”); {ltServiceID} is the id of the service that is being invoked and it is assigned during the registration process and stored in the same database table as the k8s REST endpoint for the specific LT service.
- simple error handling, i.e., if something goes wrong during processing, an appropriate message is returned to the client that initiated the process request.

---

[35] https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/
[36] When registering to the ELG platform, users must read and consent to the ELG Terms of Use, which inform them of their personal data collected when interacting with the platform, and the broader policies of ELG with regard to GDPR (General Data Protection Regulation).
[37] The REST API also includes two more endpoints for sending or creating structured requests (e.g., text split into paragraphs).

According to a **typical execution scenario**, a user visits the ELG platform, finds a tool (e.g., the ILSP Named Entity Recogniser for English) and goes to the respective resource landing page[38]. The landing page for services includes two tabs that can be used for testing the service with a limited set of calls:

- The "Try out" tab is a specially designed GUI where users can provide a sample input and see the results output by the service. Depending on the service type, they can type in or paste some text, or upload or record audio, and get the results rendered in a task-specific viewer.
- The "Code samples" tab provides the API endpoint and an example on how to call the service from a user's own code; at the moment, the example provided is a "curl" command which can be copied and executed locally; code examples in Python, Java or any other language can be added in the future.

ID	Endpoint	Type	Consumes	Produces
A	https://{domain}/execution/processText/{ltServiceID}	POST	application/JSON	application/JSON
B	https://{domain}/execution/processText/{ltServiceID}	POST	text/plain or text/html	application/JSON
C	https://{domain}/execution/processAudio/{ltServiceID}	POST	audio/x-wav or audio/wav	application/JSON
D	https://{domain}/execution/processAudio/{ltServiceID}	POST	audio/mpeg	application/JSON

Table 3: Example endpoints of LT processing services

These testing functionalities are reserved for logged in users. In both cases, i.e., when the user clicks on the "Process" button in the "Try out" GUI, or uses the curl command, a POST call, which contains the input data as payload, is sent to the same REST endpoint of the LT Service Execution Server; e.g., for the ILSP Named Entity Recogniser for English, the endpoint B (Table 3) is used. The LT execution server then communicates with the catalogue backend (via an appropriate endpoint) and checks whether this user is allowed to call this service (based on the respective daily usage/quotas). If access is granted, the catalogue returns the HTTP k8s endpoint of this service. Then, it uses the respective client/connector to generate a request to the endpoint; the request is received from Knative and forwarded to the appropriate container. If there are no available containers up and running for the specified LT service, Knative spawns a new one before forwarding the request (see Section 5.2.2). The response with the output results of the LT service container is returned back to the LT execution server which returns it to the client (try out GUI or curl) that initiated the call.

As described above, only authenticated users can access the LT service execution server; this is achieved by using gatekeeper[39], a proxy for the Keycloak Identity and Access Management server. The gatekeeper container runs[40] in front of the LT execution server container and allows only requests that contain a valid JSON Web Token (JWT) to pass through.

The LT service execution orchestrator is implemented in Java using Spring Boot. Spring Boot was selected because it (a) facilitates application configuration, (b) allows the creation of a consolidated standalone application which can be easily Dockerised and deployed, (c) provides an easy way to create REST services, and (d) provides easy integration with a large number of tools and libraries (e.g., databases or messaging middleware).

5.3 ELG platform management and support services

Apart from the ELG basic/core services (e.g., catalogue, CMS and LT services execution) there are also modules that (a) manage users and control access, (b) monitor the whole software stack, (c) gather/generate reports

[38] https://live.european-language-grid.eu/catalogue/#/resource/service/tool/480
[39] https://github.com/keycloak/keycloak-gatekeeper
[40] A separate keycloak client was created for gatekeeper, see Section 3.2 for more information.

(analytics) for the usage of the ELG platform and resources, and (d) facilitate licensing and billing. These modules are presented in the sections that follow.

5.3.1 User management

The user management and authentication module is based on Keycloak, an open-source identity and access management solution. Its adoption relieves us from the need to design and implement login and registration forms, to create and maintain mechanisms for issuing and distributing access tokens etc.

Many catalogue functionalities do not require authentication (e.g., search, view resource via the catalogue UI); however, some functionalities, such as LT service execution, are provided only for registered users. In these cases, users are redirected to a Keycloak page in order to log in to their account by providing the appropriate credentials. Once logged in, they acquire an access token (in JSON Web Token (JWT) format) and they are redirected back to the catalogue frontend; the token keeps them logged in (for some time). However, a JWT token expires after a time period, the expiration time is configurable and is specified in the keycloak server; this means that the user must re-authenticate. To avoid this situation we use refresh tokens, which are tokens that are issued to the user by Keycloak and are used by a client (e.g. frontend) to obtain in a way transparent to the user a new JWT access token when the current token expires.

A key concept in Keycloak is the realm, i.e., a dedicated domain that consists of a specific set of users, clients, security policies, access roles etc. Using Keycloak's administration console, we have created a realm dedicated to ELG, and registered the required clients under it. A client is the only way for an application (e.g., the catalogue frontend) to use and access Keycloak. We created a "catalogue frontend" client and defined the roles described above, i.e., those of Consumer, Provider, Metadata validator, Technical validator, Legal validator, Administrator and Content Manager (see Section 4.3).

Currently, as already described in previous sections, granting advanced roles for registered users of the platform is a procedure supervised by the administrators. For instance, to become a provider, a user must request this via the user profile and the request is assessed by the appropriate ELG administrative users.

User authentication is managed by Keycloak; however, user authorisation and assignment of the appropriate access level to a resource or functionality is controlled by a specific set of modules, implemented as part of the catalogue backend. These catalogue modules must know which roles are assigned to each user; however, roles are managed in Keycloak and may change at any time. For example, as described above, a user applies for the provider role; the application is processed by one of the administrators; if the request is approved, the administrator uses Keycloak's admin pages to assign the role to the user in question. This change of user roles must be communicated from the Keycloak system to the catalogue backend user module at real time, so that the user is granted the appropriate rights when he/she logs in to the system. In a similar scenario, the catalogue backend also requires for specific tasks some user information that is kept in keycloak and may change at any time (e.g., update of a user's profile). Such information are name, surname, and email and are used, for example, to send notification emails when a metadata record is submitted for publication. To solve the syncing issue we have developed a Keycloak plugin that notifies the catalogue backend via an appropriate REST service for any addition or change[41].

[41] https://gitlab.com/european-language-grid/ilsp/keycloak-events-notifier

5.3.2 Monitoring and Analytics

5.3.2.1 Monitoring of the LT processing services

Monitoring of the LT services consists of testing the ELG services on a regular basis, collecting the results of the tests, and, finally, displaying everything on a monitoring interface. These three different steps correspond to three separate services running on separate pods inside the Kubernetes cluster.

The service testing is carried out by a Flask server which calls every service using the Python SDK (Section 6.2). It collects three metrics (success or failure, response or error message, request time) for each service call and pushes them to the Prometheus[42] server. Prometheus gathers metrics (from the services monitoring but also metrics from Kubernetes) and stores them in databases. To visualise the metrics being collected, we finally use Grafana[43]. Grafana is accessible from outside the Kubernetes cluster[44] in order to make monitoring metrics accessible to the ELG team, and, in the near future, to give providers access to metrics specific to their services.

Two dashboards are currently available. One provides an overview of the current status (working or not) of the ELG services, and a second one, that is service-specific, visualises metrics related to this service (Figure 11).

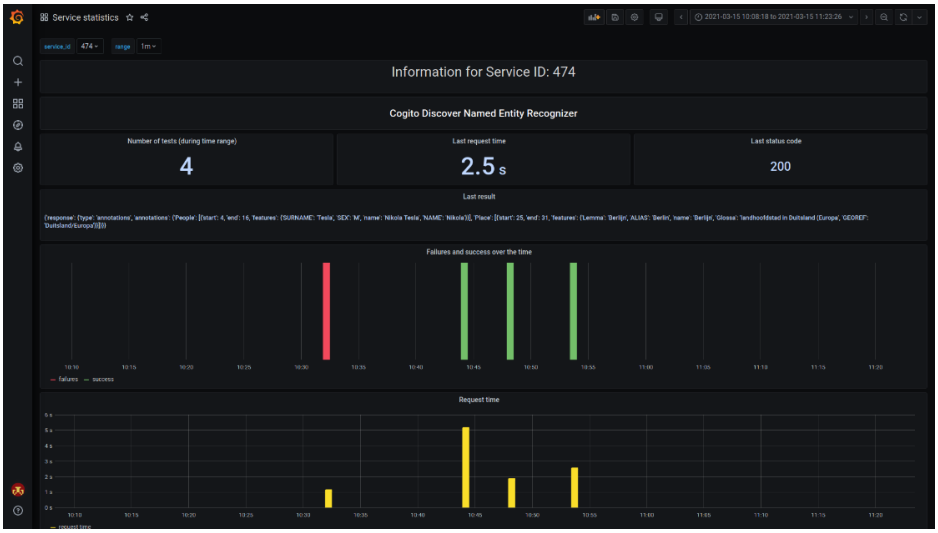

Figure 11: Screenshot of the service-specific Grafana dashboard

5.3.2.2 Catalogue usage analytics

The catalogue backend implements a statistics/analytics module for collecting usage information. Currently, the information collected includes the overall number of views for a metadata record and number of downloads/times used for a resource. User-specific actions are also tracked; these pertain to the downloads and service executions performed by a user. The statistics/analytics module will be extended in R3, if and as required.

For user downloads the information collected includes:

- Username (if logged in);
- Licence attached to the downloaded resource;
- date/time of download;
- relevant metadata record;
- calling client type.

[42] https://prometheus.io
[43] https://grafana.com
[44] https://live.european-language-grid.eu/monitoring/

Service execution statistics include

- username;
- service name;
- execution start date/time;
- execution end date/time;
- number of bytes processed;
- execution status (succeeded/failed);
- calling client type.

Statistics tied to individual published metadata records include:

- count of views;
- count of downloads of the resource attached to the record.

5.3.3 Licensing and Billing

Once fully operational, the ELG platform will also provide access to resources distributed with commercial licences. This requires a billing strategy for the ELG platform and for the LRTs it makes available, as well as metadata records that indicate their licensing terms. Appropriate technical safeguards must be implemented to ensure that access to LRTs is granted in accordance with the licensing terms; for instance, access to LRTs distributed with restricted licences must be made available only to those users that fulfil the criteria specified in the licences. The ELG licensing and billing module is currently under design and will be presented in D2.6.

In R2 we use the policy adopted in R1, i.e., focusing on LRTs and functionalities that can be provided without restrictions:

- for LT services, only the trial functionality is available and only for registered users (with quotas),
- for data resources, the first round of collection concentrated on resources that use open licences (e.g., Creative Commons) and that are available free of charge.

6 ELG platform access methods

As described in previous sections, the ELG platform exposes several REST APIs that offer functionalities for (a) browsing and searching the catalogue, (b) creating, updating and retrieving metadata records, (c) executing services, (d) downloading resources, etc. We provide two ways to our users for accessing the ELG platform, a web-based UI and a Python package. Both ways make use of the offered REST APIs, however, the web UI offers access to all functionalities implemented in the backend, while the python package offers only a subset of them (e.g., dashboard and metadata editing functionalities are not offered via SDK).

6.1 Access through the website

The ELG frontend consists of

- the “website” pages, maintained in the Drupal CMS and providing information on the LT domain and activities, the ELG project, etc., and
- the platform UIs that enable users to interact with the platform components, i.e., the ELG catalogue (e.g., search, view metadata, upload), the ELG metadata editor for documenting and registering content, the user dashboard that supports ELG users in managing their metadata records and resources.

Deliverable D3.3, which is dedicated to the ELG User Interfaces, provides a detailed account for the ELG platform frontend together with a description of the ELG website and the choices made for implementing it. In order to give an overview of the ELG platform that is as complete as possible, we provide an abridged version of the platform's different interfaces.

6.1.1 Catalogue UI

The catalogue UI is implemented as a Single Page Application (SPA), which provides fast page updates instead of unnecessary full page reloads. All functionalities are built using React, a JavaScript library for the development of user interfaces; the implementation details are presented in D3.2 and D3.3.

In ELG R2 the following functionalities are offered via the catalogue UI:

- **Browse**: The browse/search page (Figure 12) provides a list of all published entries of the ELG catalogue (e.g., LT tools/services, corpora, language descriptions, organisations, etc.) accompanied with a snippet of the description of each entry, a set of tags that have been identified as useful for consumers (languages, keywords and licences), popularity indicators (number of views and downloads/usage) and specific flags that indicate whether the resource is uploaded at ELG or an ELG-compatible (functional) service.

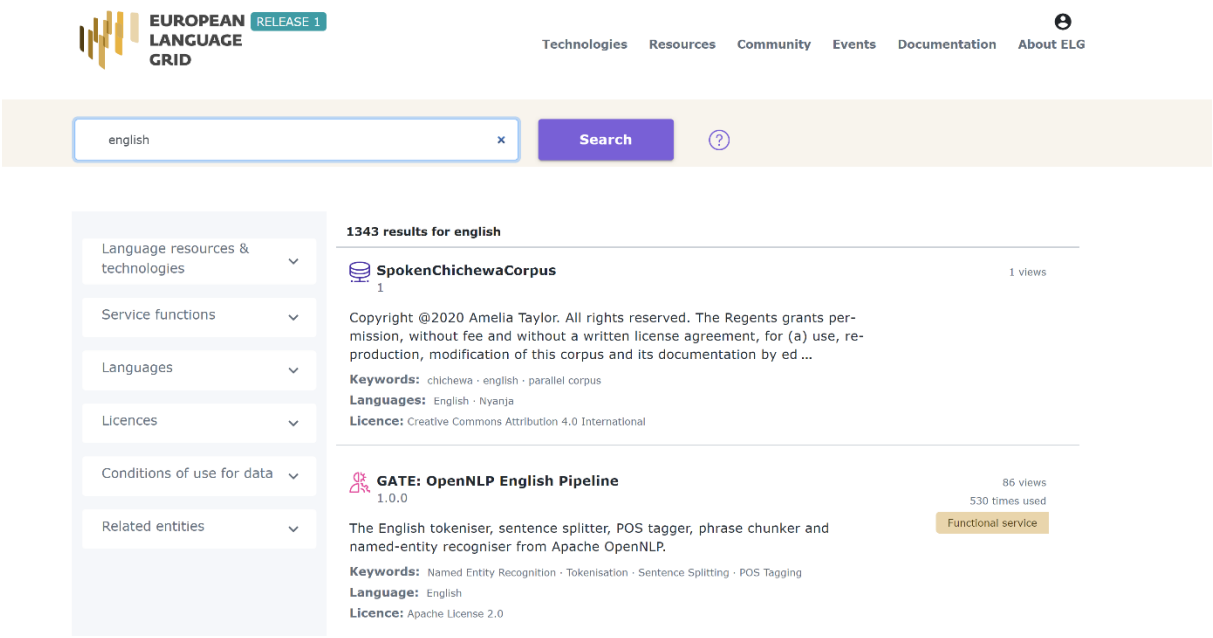

Figure 12: Browse/Search page of the ELG catalogue

- **Search**: A user enters a keyword or phrase (Figure 13) in a search box and the application narrows down the results dynamically; the minimum number of characters a user has to type to refine the search results is set to four. The search is performed on the indexed metadata elements (Section 5.1.3).

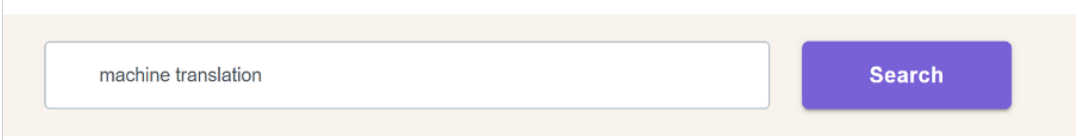

Figure 13: Keyword search

- **Faceted search**: Users can filter the full catalogue contents or previous search results by selecting values from the facets on the left side of the browse/search page (Figure 12). The facets have been selected based on user preferences (D3.2). Facets and free text search can be combined in order to refine search queries and support users in easily finding the resources they desire. Faceted search provides a long list of sub-sections (facets) that can be explored gradually.
- **View published metadata entries**: By clicking the title of an entry, users can view their full description; Figure 14 displays the landing page of a tool/service. Landing pages have been implemented for all LRT types as well as for Organisations and Projects. The design of the landing pages takes into consideration user preferences (see D3.1), design and accessibility considerations and the ELG metadata schema. As described in more detail in D3.2 and D3.3, the information is grouped in semantically similar sets and organised in tabs and specific areas of the page layout.

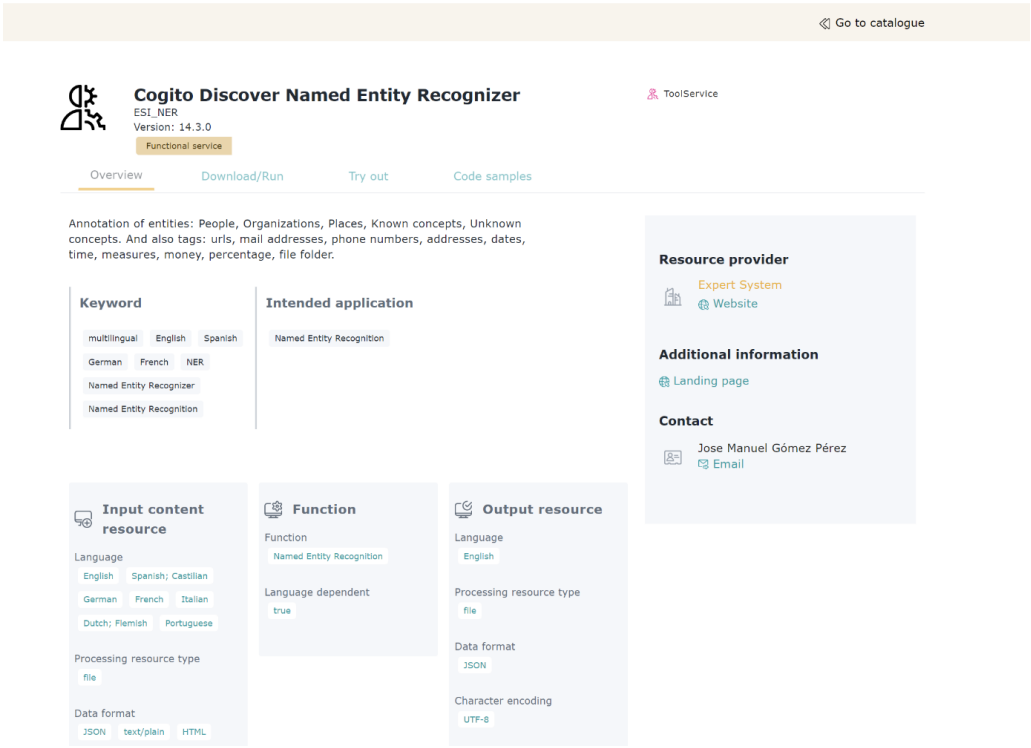

Figure 14: Landing page of a tool/service

- **Download resources**: All users are able to download resources directly from the ELG storage system in accordance with their licensing terms. Downloading is offered via the Download tab, which displays all distributions of a resource and information on the licensing (and in the future also billing) conditions for each of them.
- **Try out LT services**: An important and rather unique feature that the ELG platform provides to the users is that they can test the ELG-compliant LT services via pages called "Try out" UIs. This feature is currently reserved for registered users (see Section 5.3.1). The "Try out" UIs are separate HTML pages that use JavaScript code and are embedded as iframes[45] in the catalogue UI. They are responsible for sending and receiving data to and from the LT execution server and for visualizing the processing execution results[46]. Figure 15 shows an example. Of course, the "Try out" tab is provided only for those services that are integrated into the ELG platform; this information is provided by the ELG backend. The ELG

[45] https://www.w3schools.com/tags/tag_iframe.asp
[46] The "Try out" UIs run in containers that are deployed in the same cluster as all the other ELG components.

backend also provides the LT service execution endpoint while the authentication token that is required is retrieved from the user's active keycloak session.

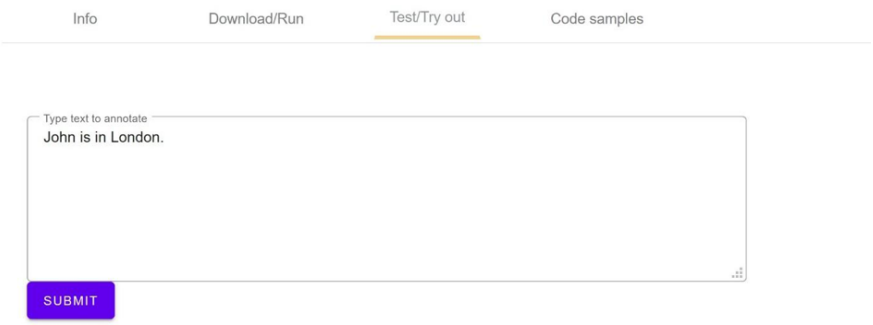

Figure 15: Try out GUI for Machine Translation services

- **Upload metadata descriptions**: Users with the "provider" role can upload metadata descriptions of their resources. For the current release, they can upload XML files compliant with the ELG schema. If the metadata file (in XML format) is valid, the user receives a "success" message; if it's invalid, the validation errors are aggregated and returned in a single message.
- **Describe items with the interactive editor**: For the ELG metadata editor, we have designed and implemented forms that enable users to formally describe their resources in compliance with the ELG metadata schema (Figure 16). Since the schema is rather complex, we attempt to meet the needs of our users by hiding this complexity and present them with a full-fledged GUI. As a result, we refrain from using long forms that could overwhelm users; instead, we have adopted a more interactive design based on the organisation of the elements along a natural workflow split across multiple steps. The design of the editor takes into account user needs and preferences (see D3.3).

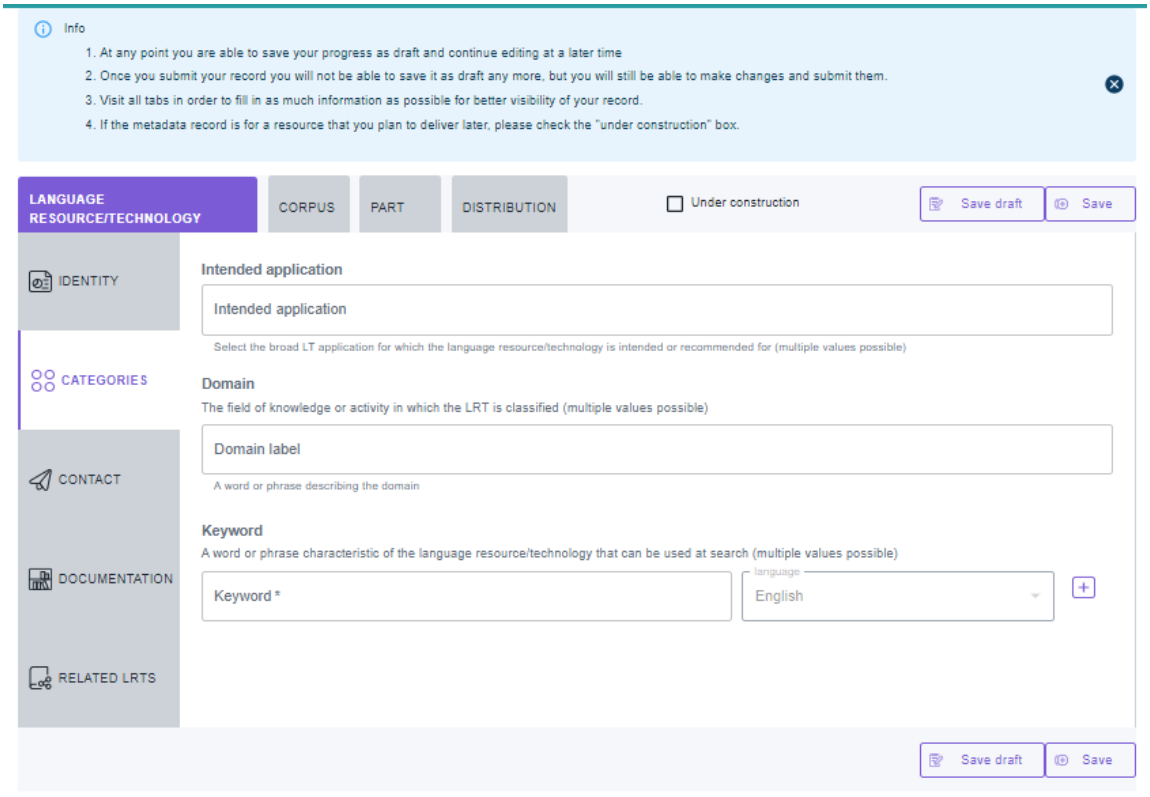

Figure 16: Editor form for corpus

Dashboard: To support users in accessing and managing the catalogue items and performing the actions that are allowed depending on their user role, we make use of a dashboard, which in ELG we call **grid** (Figure 17). Although the look and feel is the same across user roles, the actions and menu items differ depending on the user role. In this release, we have focused on the provider's grid. The grid is the central place through which providers have access to all the items they have created, as well as to the functionalities that will allow them to create new items.

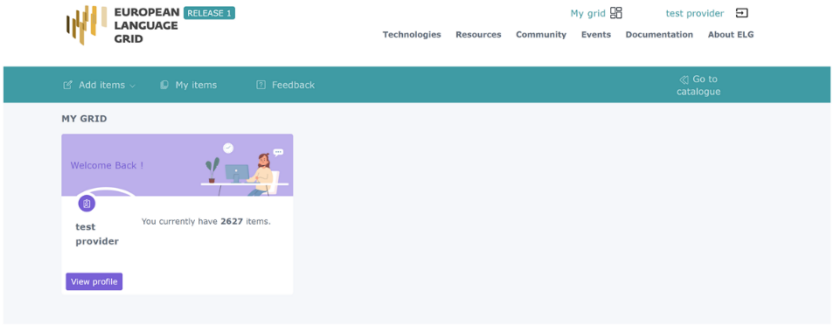

Figure 17: My grid

Users can manage the metadata records they have created through a dedicated page (My items), and, in accordance with their user rights and the publication status of the record, perform the following actions: edit or update a metadata record, submit it for publication, delete a metadata record, upload content files to a metadata record, and replace the content files that have been uploaded. The My items page (Figure 18) is a focused version of the catalogue, this time filtering records according to each user's role. This page also implements browse and search functionalities like the main catalogue page.

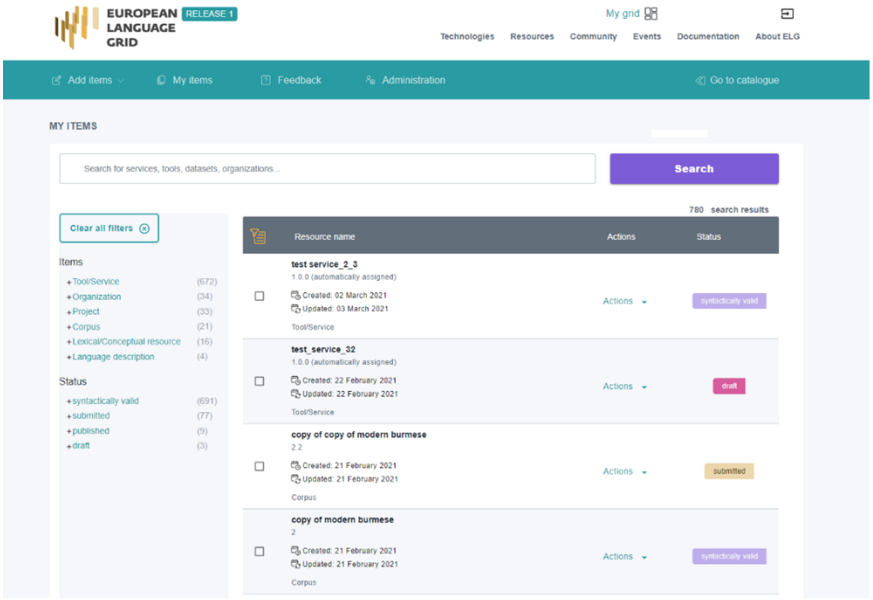

Figure 18: My items

Administration pages: Access to the administration pages, implemented with Django as a separate application, is available only for authorised users (platform administrators, content managers and all validators) via a top-level menu. Unlike platform administrators and content managers, that have access to all the administration

page facets, validators only have access to their respective validation pages; in addition, technical validators of LT services have access to the LT registration forms that are tied to records they are assigned to validate.

Validation Forms: Through their respective validation pages, validators are provided with forms that contain the relevant information and links needed for the validation of a resource.

6.1.2 Integration with the website

The current entry point for all ELG users is https://live.european-language-grid.eu. This page brings together the website part and the catalogue. Access to the catalogue is available through the "Search" button.

The catalogue UI and the ELG website are implemented as two different applications serving different purposes; however, the integration of the two systems is required for some functionalities. For instance, when a user enters a search query in the website's search field and presses the "Search" button, they are redirected to the catalogue UI where the results of the query are displayed.

In addition, a common "look and feel" is adopted by both applications. The catalogue UI follows the overall design concept of the ELG website and corporate identity to achieve this. Consequently, they both share a similar page layout, with common header and footer sections, and menu items. To integrate these sections, they both consume a REST API provided by the CMS. This API provides the information about which links should be displayed and which links should be enabled and disabled. Finally, both applications make use of common UI frameworks (Google Material design), common icons, fonts and colors.

6.2 Python SDK toolkit

Recently, we started to develop a Python SDK toolkit to allow users to use ELG through Python. The result is a pypi package[47] installable using the package installer for Python (pip). Currently only the main ELG functionalities are available on the package: catalogue browsing, corpus downloading, authentication, service calling. More functionalities will be added in the future. We chose Python for the first ELG toolkit as it is the leading coding language for NLP. The Python SDK is composed of classes that we will detail below.

6.2.1 Catalogue class

The Catalogue class uses the ELG catalogue backend API to browse the catalogue and access the metadata of the resources. Users can use the same filters available on the UI and get the result as a list of entities that can be looked at individually.

```
from elg import Catalogue
catalogue = Catalogue()
results = catalogue.search(
    resource = "Tool/Service",
    function = "Machine Translation",
    languages = ["en", "fr"],
    limit = 100,
)
print(f"Machine Translation service for English and French:\n{results[0]}")
```

Figure 19: Python SDK Catalogue code example

[47] https://pypi.org/project/elg/

6.2.2 Corpus class

The Corpus class allows users to download ELG corpora directly from Python. The Corpus can be initialised using the ELG id of a corpus or using the result of a Catalogue search.

```
from elg import Corpus, Catalogue
catalogue = Catalogue()
results = catalogue.search(
    resource = "Corpus",
    languages = ["German"],
    search="ner",
    limit = 100,
)
corpus = Corpus.from_entity(results[0])
corpus.download()
```

Figure 20: Python SDK Corpus code example

6.2.3 Authentication

In a typical scenario when users attempt to access the ELG platform via the Python SDK, they are redirected to the ELG web interface and asked to enter their login credentials; from this procedure, a JWT token is acquired which keeps them logged-in in order to be able to interact with the platform (see Section 5.3.1). The ELG Python SDK manages authentication based on the Authentication class which communicates with the Keycloak server to obtain Bearer JWT tokens for the specified user. JWT tokens expire after a time period and users have to provide again their credentials. To avoid this, we implemented utility methods that keep users authenticated in a transparent way by using the Keycloak's refresh token mechanism (see Section 5.3.1).

6.2.4 Service

The Service class of the Python SDK is very important as it allows users to call ELG services and so to integrate ELG services in Python workflows. It also uses features that simplify the use of services. For example, as mentioned previously, the JWT access tokens are updated automatically (refresh token), the input provided to the service can be either a text or a file, the call can be synchronised or not. The Service can be initialised using its ELG id or using the result of a Catalogue search.

```
from elg import Service
service = Service.from_id(474)
result = service("Nikolas Tesla lives in Berlin.")
print(f"\n{result}")
```

Figure 21: Python SDK Service code example

7 Conclusions

In this deliverable we present Release 2 of the ELG platform. This release comes with an important set of functionalities and mechanisms, demonstrating the potential capacities of ELG, once fully operational. Users can browse through the catalogue, search for specific LT services and resources, view them, test LT services and download resources with open licences through graphical UIs or a Python SDK toolkit. Providers can describe and integrate LT services or upload data resources into the platform with an interactive editor, as well as an upload functionality of metadata records. They can also manage their records via their dashboard. The catalogue has already been populated with a substantial number of LT services and data resources, stakeholders

and projects. We have designed and implemented APIs for the execution of LT services, and the interaction with the platform. The final release will enhance the platform with improvements of the existing functionalities and the addition of new ones.

8 References

Penny Labropoulou, Katerina Gkirtzou, Maria Gavriilidou, Miltos Deligiannis, Dimitris Galanis, Stelios Piperidis, Georg Rehm, Maria Berger, Valérie Mapelli, Michael Rigault, Victoria Arranz, Khalid Choukri, Gerhard Backfried, José Manuel Gómez Pérez, and Andres Garcia-Silva. Making Metadata Fit for Next Generation Language Technology Platforms: The Metadata Schema of the European Language Grid. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020), pages 3421-3430, Marseille, France, 2020. European Language Resources Association (ELRA).

Georg Rehm, Maria Berger, Ela Elsholz, Stefanie Hegele, Florian Kintzel, Katrin Marheinecke, Stelios Piperidis, Miltos Deligiannis, Dimitris Galanis, Katerina Gkirtzou, Penny Labropoulou, Kalina Bontcheva, David Jones, Ian Roberts, Jan Hajic, Jana Hamrlová, Lukáš Kačena, Khalid Choukri, Victoria Arranz, Andrejs Vasiļjevs, Orians Anvari, Andis Lagzdiņš, Jūlija Meļņika, Gerhard Backfried, Erinç Dikici, Miroslav Janosik, Katja Prinz, Christoph Prinz, Severin Stampler, Dorothea Thomas-Aniola, José Manuel Gómez Pérez, Andres Garcia Silva, Christian Berrío, Ulrich Germann, Steve Renals, and Ondrej Klejch. European Language Grid: An Overview. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Christopher Cieri, Khalid Choukri, Thierry Declerck, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, pages 3359-3373, Marseille, France, 2020. European Language Resources Association (ELRA).