Book Section:
Roberts, I., Labropoulou, P., Galanis, D. et al. (2022) Using the European Language Grid as a consumer. In: Rehm, G., (ed.) European Language Grid: A Language Technology Platform for Multilingual Europe. Cognitive Technologies. Springer Cham, Cham, pp. 37-66. ISBN: 978-3-031-17257-1.

https://doi.org/10.1007/978-3-031-17258-8_3

Chapter 3
Using the European Language Grid as a Consumer

Ian Roberts, Penny Labropoulou, Dimitris Galanis, Rémi Calizzano, Athanasia Kolovou, Dimitris Gkoumas, Andis Lagzdiņš, and Stelios Piperidis

Abstract This chapter describes the European Language Grid cloud platform from the point of view of a *consumer* who wishes to access language resources or make use of language technology tools and services. Three aspects are discussed: 1. the web-based user interface (UI) for casual and non-technical users, 2. the underlying REST APIs that drive the UI but can also be called directly by third parties to integrate ELG functionality in their own tools, and 3. the Python Software Development Kit (SDK) that we have developed to simplify access to these APIs from Python code. The chapter concludes with a preview of the upcoming payment module that will enable the sale of commercial LT services and resources through ELG, and a discussion of how ELG compares and relates to other similar platforms and initiatives.

1 Introduction

The European Language Grid (ELG) platform (Rehm et al. 2021) provides access to Language Technology (LT) tools and services, both basic Natural Language Processing (NLP) tools and end-to-end applications, as well as data resources, such as structured and unstructured datasets and corpora, Machine Learning models, lexica, ontologies, terminologies, etc. Chapters 7 (p. 131 ff.) and 8 (p. 151 ff.) present the current state of LT services as well as datasets and language resources included in the ELG platform respectively.

Ian Roberts
University of Sheffield, UK, i.roberts@sheffield.ac.uk

Penny Labropoulou · Dimitris Galanis · Athanasia Kolovou · Dimitris Gkoumas · Stelios Piperidis
Institute for Language and Speech Processing, R. C. "Athena", Greece, penny@athenarc.gr, galanisd@athenarc.gr, akolovou@athenarc.gr, dgkoumas@athenarc.gr, spip@athenarc.gr

Rémi Calizzano
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Germany, remi.calizzano@dfki.de

Andis Lagzdiņš
Tilde, Latvia, andis.lagzdins@tilde.lv

G. Rehm (ed.), *European Language Grid*, Cognitive Technologies,
https://doi.org/10.1007/978-3-031-17258-8_3

ELG enables consumers of Language Technology to browse through the ELG catalogue and have an overview of its contents, search for specific resources and select as well as view the features of a resource through its formal description (metadata record). Users can download resources hosted in the ELG cloud infrastructure in accordance with their licensing conditions, or, in the case of external resources, be re-directed to the location where they can be downloaded from or accessed. They can also try out services in order to assess whether they comply with their needs; for this to happen, the services must comply with the ELG technical interoperability specifications, which are outlined in Chapter 4. Furthermore, ELG includes a catalogue of commercial companies and academic and research organisations that are active in the LT domain and of EU and national projects that have funded the development and maintenance of LRTs (see Chapter 9); LRTs, actors and projects are interlinked offering a comprehensive image of the LT landscape in Europe.

Different types of users have different requirements and different levels of technical expertise, and the ELG platform provides a variety of access methods to address these; all the principal functionality of the ELG is offered through both web-based user interfaces (UIs, see Section 2) for interactive use and Application Programming Interfaces (APIs, see Section 3) for programmatic access. In addition, the ELG team supports the advanced needs of LT integrators with dedicated tools and helpers; most notably a Software Development Kit (SDK) for Python (see Section 4), which is currently the most widely used programming language in the LT community.

Supporting consumers to easily discover resources is of utmost importance, especially when a catalogue contains many entries, as in the case of ELG (over 13,000 metadata records for LRTs and 1,800 related entities at the time of writing and constantly increasing). Best practices and recommendations (Wu et al. 2019; Wilkinson et al. 2016) have been taken into account in the design and implementation of the ELG catalogue pages and interaction mechanisms with the consumers.

At present all functionality of the ELG platform is offered free of charge. All users can view the catalogue and metadata descriptions as well as download open access resources. In order to download resources with restrictive licences and try out ELG-compatible services, users must register in the platform, as described in Section 5. It should be noted that while the ELG platform does not currently charge fees for access to any resources or services, restrictions may apply with regard to the intended use(s) of the resource (e. g., available only for non-commercial use), request for explicit consent to licensing conditions, etc. Resources available with commercial licences are described in the ELG catalogue but for now re-directed to the providers for further information. A prototype billing module, described in Section 6, has been implemented and will be fully launched following the setup of the ELG legal entity (see Chapter 13). Finally, in Section 7 we compare the ELG platform to other similar services and initiatives, from the point of view of the service or resource consumer. A similar comparison from the point of view of the provider can be found in Chapter 4.

2 Web-based Interface

The ELG platform targets a diverse set of user types with different needs and levels of technical expertise. The primary access route for non-technical users is via the web user interface (UI), which prioritises user-friendliness and ease of use alongside raw performance considerations. The catalogue UI includes two main pages: the *catalogue page*, which offers access to the catalogue contents, and the *view pages* for each metadata record or resource (LT, LR, organisation, project).

2.1 Viewing the Catalogue

After ELG's homepage, the dedicated catalogue page (Figure 1) is the primary entry point through which users have access to the ELG platform contents and functions. Users can browse through the entire catalogue to find entries that might interest them. They can also look for specific entries, using the free text search bar, filtering the catalogue with one or more facets, or combining these two modes.

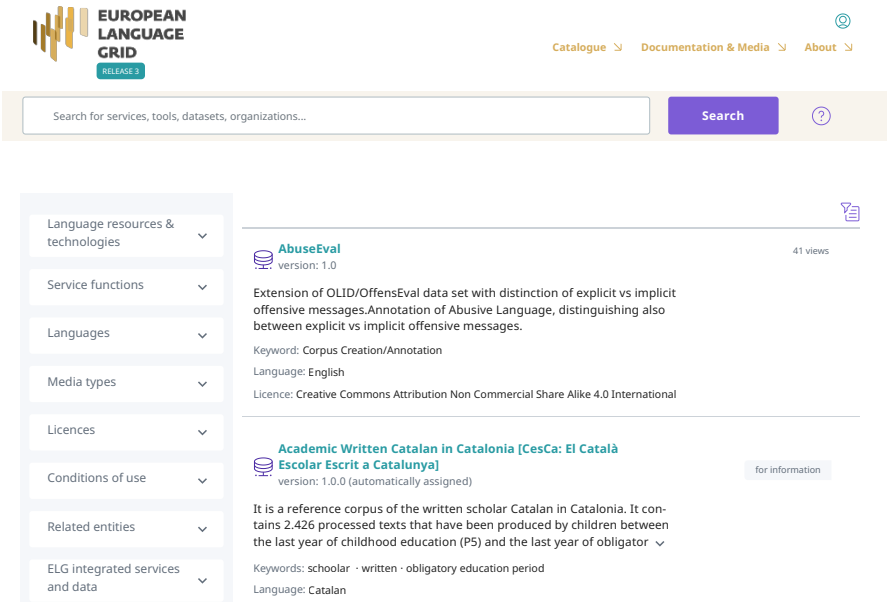

Fig. 1 Browse/Search page of the ELG catalogue

The main section of the catalogue page shows all published entries sorted by name in alphabetical order. Users can also sort the entries according to the update date of the metadata record, so that they can view the most recently added entries

first. The catalogue shows only the most recent version of each entry if multiple versions are registered. The snippet informs the users of additional older entries, which can be viewed and accessed through the view page of the newest version (see Section 2.3). This allows users to always keep up to date with the most recent version of a service, but also access older versions when needed, for instance, when reproducing previously published experiments.

Each entry is shown with an informative snippet, designed to serve as a preview of the full metadata record and to help users decide whether they want to explore the entry further. Following well-established practices in catalogues, each entry is represented by its name, an excerpt of its description, a set of metadata tags, and popularity indicators. The set of metadata tags has been carefully selected to accommodate consumer requirements, as identified in a user survey conducted during the ELG design and specification phase (Melnika et al. 2019) and subsequently enriched based on user feedback. All types of entries include their free-text keywords. Entries representing LRTs additionally include the resource type (represented with an icon), language(s), and licence(s). The popularity indicators, displayed at the right hand side of the snippet, consist of counts of visits of the view page of all versions of an entry, counts of downloads (for ELG-hosted resources only) and number of calls (for ELG-compatible services only; again for all versions of the entry). Finally, dedicated badges are shown for resources hosted in ELG and ELG-compatible services, as well as for a subset of the metadata records that have been imported from other catalogues with minimal metadata (see Chapter 6).

2.2 Searching the Catalogue

Search of the catalogue is supported in two different modes, which can be combined in order to refine search queries and support users in easily finding entries of interest: free text search (Section 2.2.1) and faceted search (Section 2.2.2).

2.2.1 Free Text Search

Users enter a word or phrase in the search box at the top of the catalogue page (see Figure 1) and click the "Search" button to submit the query. By default, the search functionality matches whole words using the OR operator. Advanced queries, utilising the Lucene query syntax[1], are supported, allowing users to search for partial or exact matches, words or phrases, etc. Only certain metadata elements have been indexed to make them searchable; these include a resource's name(s), short name(s), keywords and a subset of technical elements appropriate for each entry type and deemed important as a search criterion. For example, for all LRTs, additional in-

[1] https://www.lucenetutorial.com/lucene-query-syntax.html

dexed elements are the “resource type”, “language” and “licence”; for LT tools/services, “service function” is also added to the search elements.

In addition, to improve recall of search results, for those metadata elements that take values from controlled vocabularies, i. e., “service function”,“intended LT application”, and “language”, the query is expanded with the use of synonyms. Synonyms for the first two elements are derived from a taxonomy of LT activities[2], which provides the values. For alternative names of languages, besides the official ones included in the ISO 639-3 standard for language codes[3] (International Organization for Standardization 2007), we exploit open access vocabularies published as linked data, i. e., the Glottolog list of languoids (families, languages, dialects)[4], the lexvo ontology of languages[5], and the WALS list of languages[6]; all these vocabularies are offered through Glottolog.

2.2.2 Faceted Search

Users can filter the catalogue or previous search results by selecting values from the list of facets (Figure 2) on the left side of the catalogue page (Figure 1). For facets with a long list of values, such as languages and licences, the facet values are broken down into subsections or a search bar is included to refine the list.

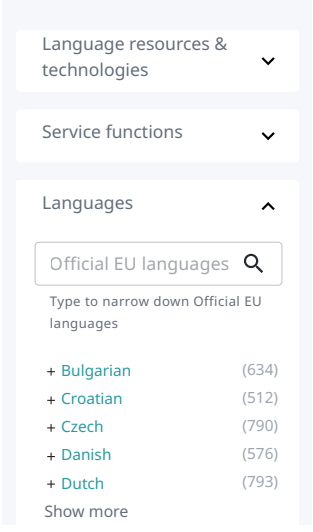

Fig. 2 Faceted search in the ELG catalogue

[2] Part of the OMTD-SHARE ontology, see http://w3id.org/meta-share/omtd-share.

[3] https://iso639-3.sil.org/code_tables/639/data

[4] https://glottolog.org

[5] http://lexvo.org/ontology

[6] https://wals.info/languoid

The facets were selected in the initial phase of the ELG development based on user preferences collected through a survey conducted for the technical platform specifications (Melnika et al. 2019). Important criteria for users searching for are language coverage (62%), licence and access conditions (59%) and availability of open source code (56%). Later on, more facets have been added to reflect updates in the metadata schema and improve search capabilities (Wu et al. 2019).

There are two facets, based on the *resource type* and *entity type* elements, that create dedicated subsets of the catalogue contents. The values are taken from the respective elements of the ELG metadata schema, but are tuned to current LT approaches. Thus, with regard to LRTs, users can view specific catalogues of tools and services, corpora, lexical/conceptual resources, models, grammars and other language descriptions. In the ELG schema the last three are subclasses of the *language description* type, but we opted to treat them as separate resource types primarily to improve the visibility of models; these are what define the state of the art for many NLP tasks and are likely to be particularly popular, so need to be easily discoverable. The two catalogues of organisations and projects are a valuable asset for boosting and activating interactions within and across the LT community (including match-making in the ELG marketplace) and eventually also for monitoring funding outcomes.

LRTs can be further filtered using the facet *ELG integrated services and data* to restrict the catalogue view to the ELG-compatible services and resources hosted in ELG, for users who wish to take advantage of the "try out" functionality offered by ELG for services or of the direct download of resources uploaded in ELG.

The facet *languages* shows the language coverage of the LRTs in the ELG catalogue, i. e., the languages of the contents of data resources and the ones that tools/services cater for. Given the scope of ELG, the official EU languages are presented in a separate group shown at the top of the facet. The encoding of language values in the catalogue follows the BCP 47 recommendations (Phillips and Davis 2009), i. e., it allows for users adding a tag consisting of subtags for language, region, script and language variants, but for simplicity of the UI the facet browser includes only the values of the *language* subtag. Moreover, it includes only one of the known names of a language; e. g., for "Catalan; Valencian", only the first name is shown. For languages and language varieties without an ISO 639 code, we show the name associated with the respective Glottocode[7] if it has one.

The facets *intended LT application* and *service function* are used for classifying LRTs and related entities with concepts specific to the LT community; consumers can search for services that perform specific functions (e. g., dependency parsers, Machine Translation tools), but also for corpora or models that have been created or can be used for a a specific application (e. g., bilingual or multilingual corpora to be used for building machine translation models), as well as for organisations and projects active in an LT area; the values of these two elements are both taken from the taxonomy of LT areas[8], and free text values that have been added by users.

[7] https://glottolog.org/meta/glossary

[8] http://w3id.org/meta-share/omtd-share/

Licensing and access conditions are among the search criteria most requested by users: *licences* gives the detailed list of licences used for LRTs in the catalogue.[9] The more coarse-grained facet *Conditions of use* groups licences by the general types of conditions they impose (e. g., "no commercial use", "share-alike"), intended for users with little knowledge of legal terms. Users are still advised to carefully read the licence specified on the view page of each LRT for all terms and conditions.

The *media types* facet was introduced at a later stage when the number of multimodal resources included in the catalogue increased. As for languages, this refers to the media type of the contents of resources or the media type of the input/output of tools, and can be used to quickly search not only for text-related applications and resources, but also for audio, video and image ones.

The ELG catalogue includes both entries added by individuals and entries aggregated from other catalogues.[10] Thus, the facet *source* refers to the source of the metadata record. It includes the name of the catalogue from which the record has been imported or the value "ELG/ELE" for records originating in ELG or added by the collaborating project European Language Equality (ELE)[11] through processes described in Chapter 6.

2.3 Viewing Metadata Records and Resources

By clicking the title of an entry on the catalogue page, users can view its full description. Figures 3 and 4 show the view page of a tool/service and a corpus respectively. Specific view pages have been implemented for all LRT types published in ELG. Their design takes into account user preferences and requirements, design and accessibility considerations and the ELG metadata schema. They allow users to access detailed information about an item, test it, if it is a service integrated in the platform, and, finally, obtain and use it for their purposes.

Even though the types of information shown on the view pages differ for each category, we apply a consistent visual look and feel for all of them. The information on the view page of each item comes from the respective metadata record. Taking into consideration the specificities and richness of the metadata schema, but also user-friendliness, the information is layered along specific sections of the page. Thus, view pages share a common layout that consists of a header, a right-hand sidebar, a main content area and a bottom content region; the positioning of the elements on the page and the formatting of the text is carefully thought through to draw users' attention to the most important information.

The header shows the name and version of the resource, its resource type and optionally important flags (e. g., to indicate that a certain service is deployed in ELG).

[9] Chapter 6 discusses why this element was made mandatory.

[10] See Chapters 4 and 6 for more information on the respective modes of population.

[11] https://european-language-equality.eu

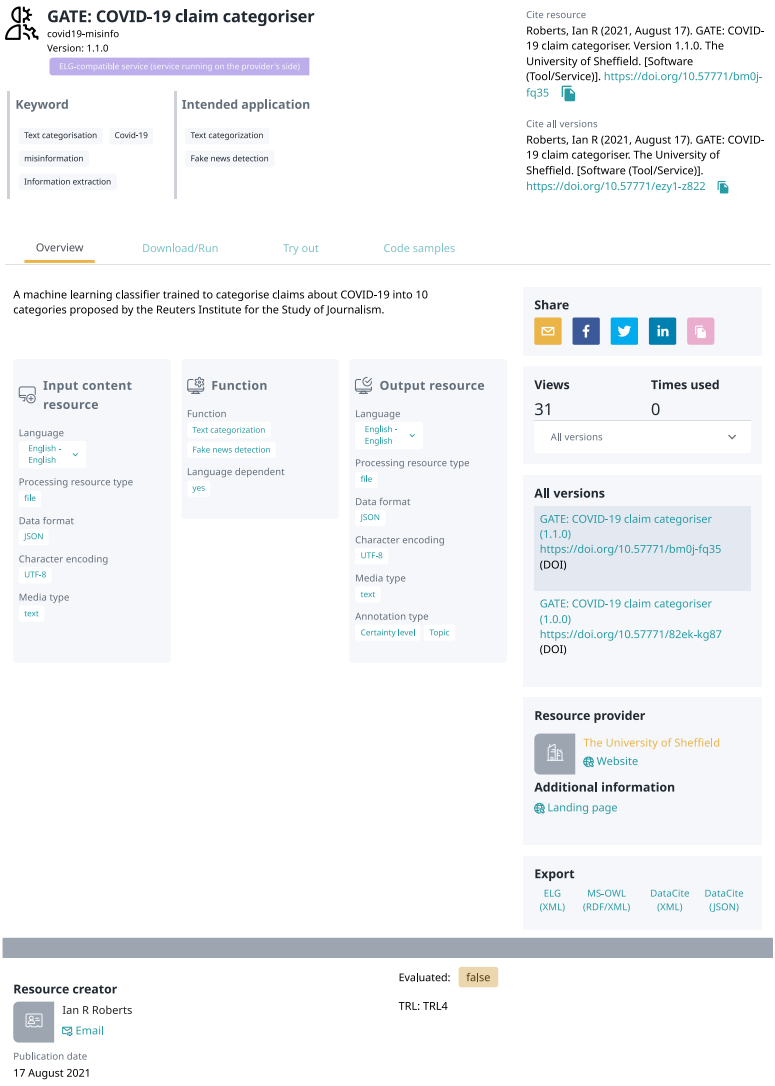

Fig. 3 View page of an ELG-compatible service

At the top of the right-hand sidebar, the button “Claim” may appear for some of the metadata records; these are records with minimal metadata that have been imported through automatic harvesting and bulk collection procedures (see Chapter 6). The claiming process enables interested users, i. e., the rightful owners of these LRTs, to ask to curate and enrich them. The same area provides for all records information on how they can be cited, according to data and software citation principles (Smith

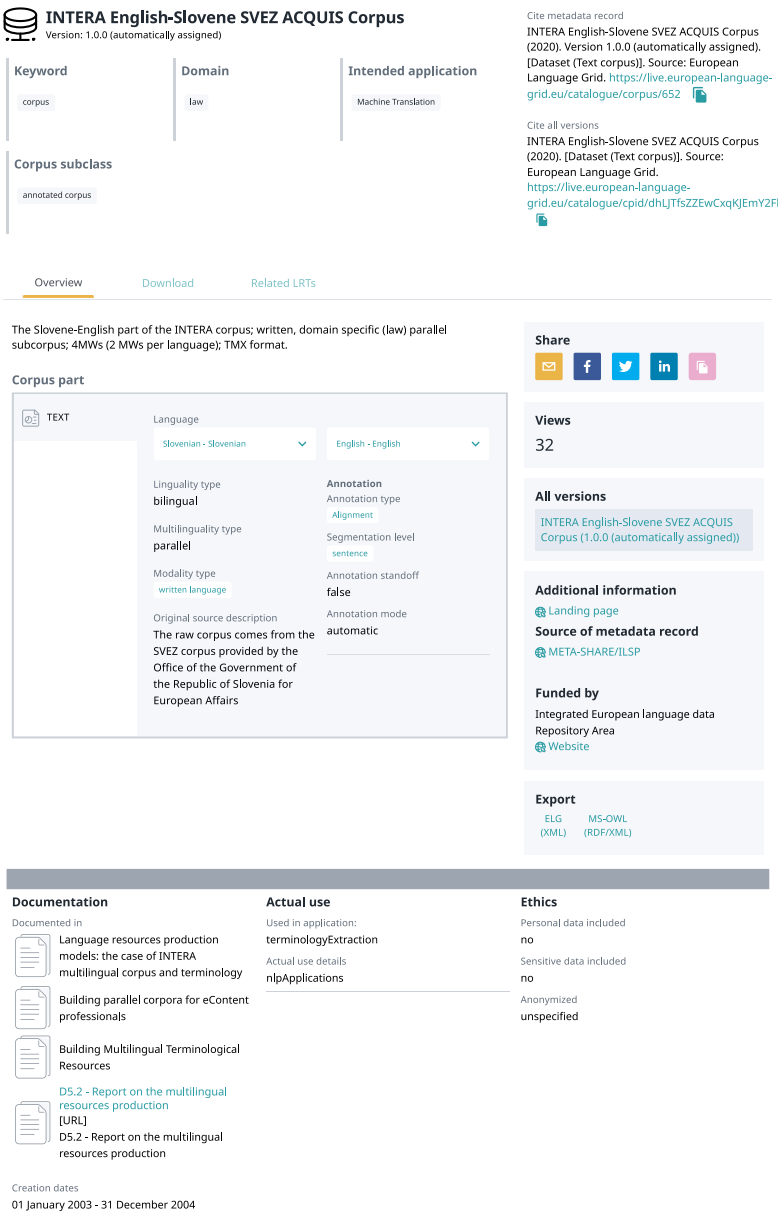

Fig. 4 View page of a corpus

et al. 2016; Data Citation Synthesis Group 2014) and DataCite guidelines[12]. They also have the option to share the URL link of the page by email or through social media and export the metadata record as an XML file in the ELG-compliant schema. Statistics of resource usage are shown both for the particular resource version and for all versions (if there are multiple versions). Links to other versions of the same resource are also displayed here.

In the content area, tabs split information into smaller views and enable users to navigate to offered functionalities of the platform. The first tab provides an overview of the main features of the entry that help users decide if the resource fits their needs. In terms of layout it is similar across resource types, but the information types (metadata elements) differ. Compare, for instance, Figures 3 and 4 that show the overview tab for a service and a corpus. The top shows a free text description for all record types, followed by a section for classification information (keywords, domain, service function, etc.) and an area for technical metadata, e. g., the media type(s) and language(s) of a corpus, the input and output data formats for a service, etc. The bottom section contains hyperlinks to useful documents, creation details, etc. and is again specific to resource types.

Depending on the resource type, the "Download" or "Download/Run" tab presents information related to the distribution of the resource, such as the licence under which it can be accessed, a technical description of its content files (e. g., size and format for data resources), and access to the resource itself – a direct download link if the resource is uploaded into ELG (see Section 3.2), otherwise a redirect to the resource on its provider's site. Figure 5 shows the tab for a corpus hosted in ELG.

A third tab appears if the item is related to other items, e. g., a project with the LRTs this project has funded, an organisation with the LRTs it has created and the projects it is involved in.

Finally, ELG-compatible services have two more tabs that enable users to try out the service (see Section 2.5) and inform them how to use it via the command line or Python SDK (see Section 4).

2.4 Consumer's Grid

Individuals can browse the catalogue, view detailed metadata cards and download open access resources without any registration. To access restricted resources and run ELG-compatible services, they must be registered with an ELG account and also logged in. For registered users, ELG offers a dashboard ("grid") for managing and performing actions on catalogue items depending on their rights (see Chapter 2 for more information on user roles and rights). As for view pages, the grid follows a similar layout which is customised for each user type.

The consumer's grid (Figure 6) allows registered users to monitor their usage of daily quotas, view details on downloads of LRTs they performed and of the services

[12] https://datacite.org/cite-your-data.html

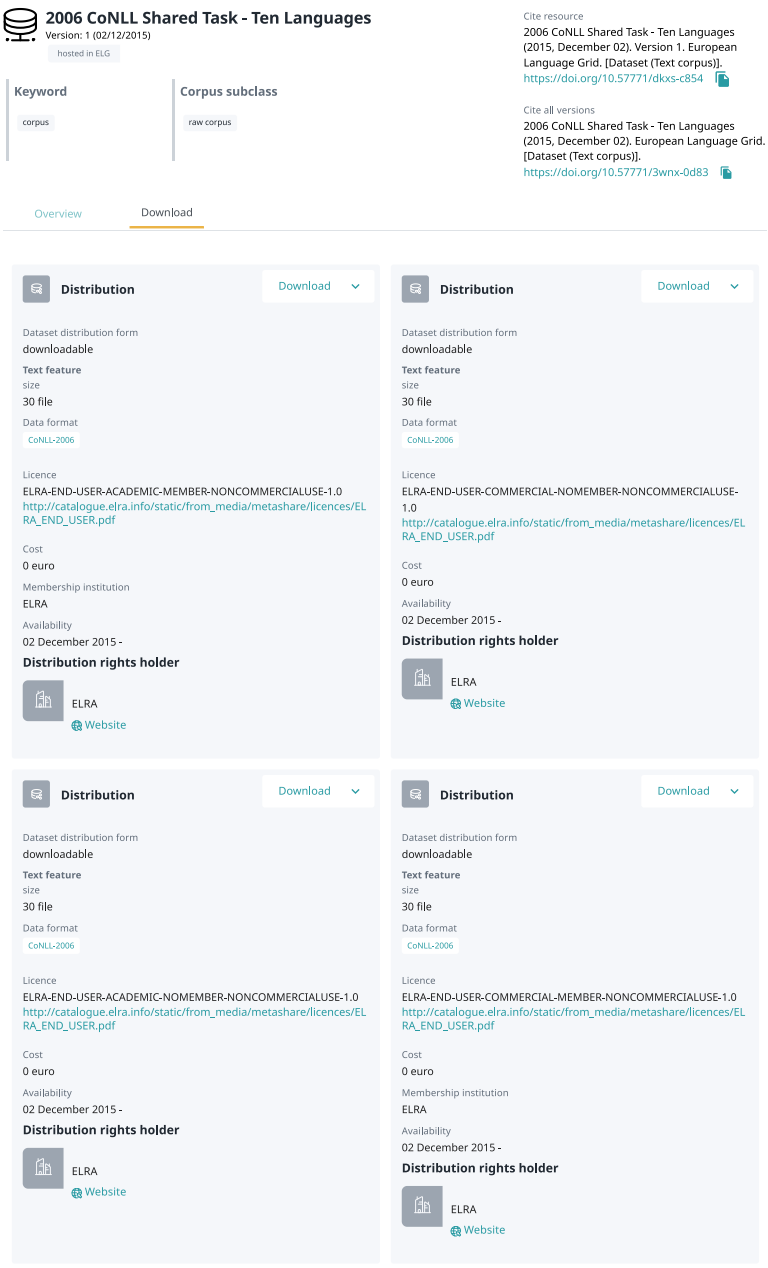

Fig. 5 Download tab for a corpus

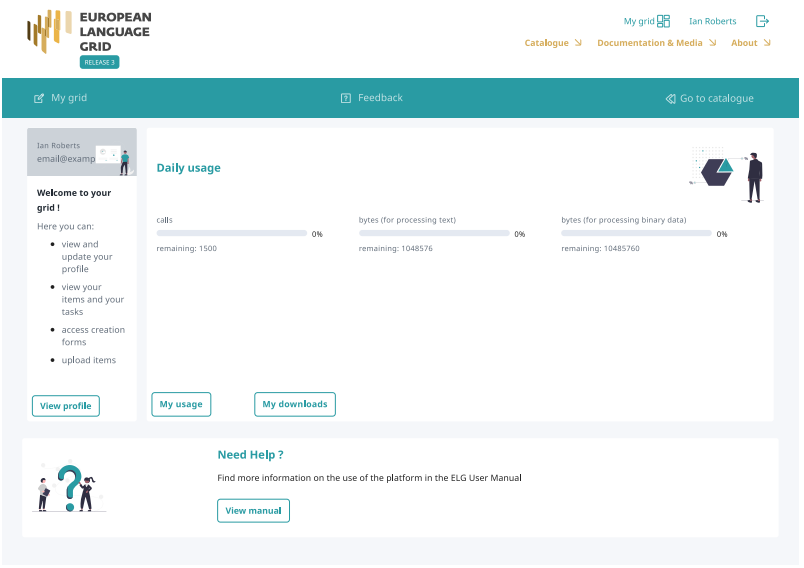

Fig. 6 Consumer's grid (see Figure 4 in Chapter 4, p. 73, for the Provider's grid)

they have deployed. Additional elements of the "My grid" section that are relevant only to *provider* users are discussed in Chapter 4.

2.5 Try out UIs for Language Technology Services

One of the key benefits of having an LT service fully integrated in ELG is that users have access to a "try out" UI from which they can test the service directly using their web browser. ELG provides standard trial UIs[13] covering all principal service types:

- *Information Extraction (IE) & text analysis* services take text input and produce standoff *annotations* over that text.
  In addition to this generic text analysis UI there is also a specific one for dependency parsers that renders CoNLL-U style annotations as a tree structure.[14]
- *Text-to-text* services (most notably Machine Translation, but also summarisation, anonymisation, etc.) take text and return new text that is derived from the input.
- *Text classification* services take text input and classify it somehow (e. g., language identification, "fake news" detection, etc.)
- *Speech recognition* services accept audio and return a text transcription.

[13] Service providers whose tools do not fit one of the above UIs are free to provide their own.

[14] https://universaldependencies.org/format.html

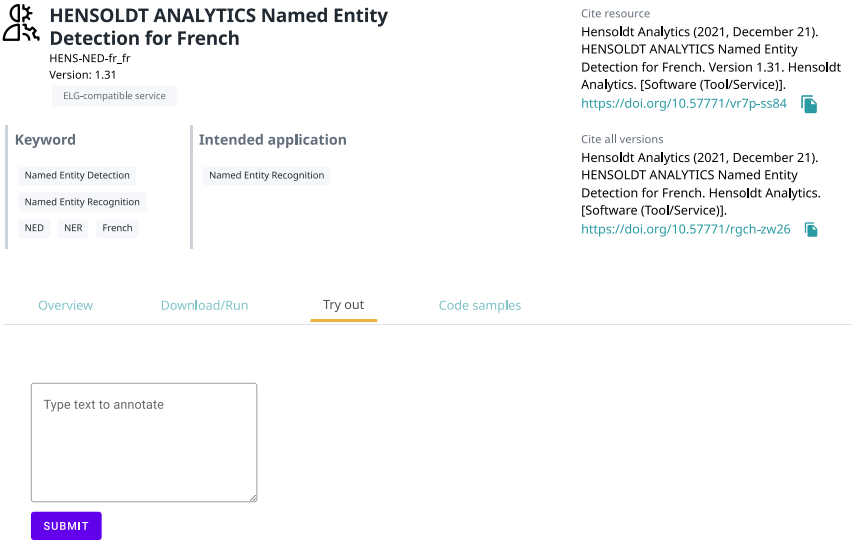

Fig. 7 An example "try out" UI for a named entity service

- *Audio annotation* services take audio and return standoff annotations over particular time segments of the audio stream.
- *Text-to-speech* services take text and return audio.
- *Image OCR* (optical character recognition) services take image data and return text extracted from the image.

The trial UIs for services are available to any user who has logged in to the ELG portal. The UI appears in the "Try out" tab when viewing a service in the catalogue; Figure 7 shows an example for a simple service that only requires plain text. However, some services can be much more complex, requiring additional parameters or providing snippets of sample data that users can test the service with – if a service declares these kinds of items in its metadata record, then the try out UI will automatically adapt, as shown in Figure 8. This service – also see Chapter 18 – declares two optional parameters and offers a selection of samples in different languages.

The UIs have been designed to render all of the main service response types in a user-friendly way, for example, annotations over text are shown as colour highlights (Figure 9), translated text is displayed alongside the original, audio can be played directly in the browser, etc.

Fig. 8 A more complex "try out" UI for the Text2TCS service

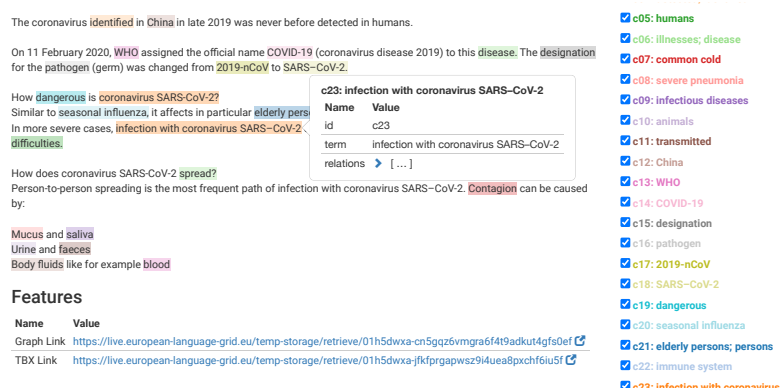

Fig. 9 Example result for the Text2TCS service showing rendered text annotations

3 Public REST APIs

The web user interfaces described above are built on top of a set of REST APIs, and the same APIs can also be called directly by third parties, allowing ELG functionality to be accessed programmatically and embedded into other tools. The current public APIs break down into three principal groups: 1. accessing/using the catalogue (Section 3.1), 2. accessing and downloading ELG-hosted data resources (Section 3.2), 3. calling ELG-hosted LT services (Section 3.3).

All APIs are HTTPS-based and use JSON as the primary data representation format. Where authentication is required, this is performed using OAuth2 access tokens issued by the ELG user management layer (see Section 5).

3.1 Accessing and Using the Catalogue

The ELG catalogue is a Python web application based on the Django REST Framework.[15] It offers a number of services as REST APIs, including the following ones which are useful for consumers: 1. searching the catalogue, 2. authorising the download of a resource or access of any resource or page, 3. retrieving the metadata description of a resource.

3.2 Downloading a Resource

ELG allows providers to upload and store the actual contents of their LRTs within the platform (data files for corpora, source code for software, etc.), and the catalogue offers an API to allow consumers to download this data subject to licensing terms.

LRT data is stored in a storage service compatible with the API of Amazon S3. Access by consumers is mediated by a Storage Proxy.[16] The proxy defers to a data management module within the catalogue application (see Section 6) to determine, based on authentication information provided by the user who attempts the download, whether that user has the permission to download the requested resource. Factors considered in making a decision include whether the resource is open access to all requesters (authenticated or not), if it requires authentication, or if the user must explicitly accept the terms of the licence prior to download.

3.3 Language Technology Service Public API

One of the great strengths of ELG is its use of a single harmonised set of APIs for all ELG-compatible LT services regardless of provider. This differs from other API aggregator platforms such as RapidAPI[17], where each service provider defines their own API and the caller must adapt their code for each different service.

For each LT service the platform provides two endpoints at which the service can be called, which implement synchronous and asynchronous modes of operation. These endpoints are implemented in the LT Service Execution Server. The endpoint URLs can be found in the `service_info` section of the metadata record JSON structure returned by the catalogue API.

The synchronous mode simply consists of a single API call in which the caller will `POST` the data to be processed and receive the results via the response to the same request. The asynchronous mode accepts the same type of request but instead of blocking the caller until the results are ready it returns a polling URL, which the

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

[16] https://gitlab.com/european-language-grid/platform/s3proxy

[17] https://rapidapi.com

caller must repeatedly poll for status updates. This requires more HTTP requests but for long-running services (or those that take some time to scale up from idle) the asynchronous mode is more resilient to connection failures or intermediary proxy timeouts between the client and the ELG platform.

Any query parameters appended to the URL will be passed through to the service and may affect its behaviour – each service declares the parameters that it supports in its metadata. All available versions of a given service are exposed at the same endpoint, the `?version=...` parameter is used to select between them, with the latest version used by default if no parameter is given.

The `POST` data must have an appropriate `Content-Type` header for the service in question; services that take text (such as text analysis or MT services) expect "text/plain"[18], services that take audio (such as speech recognition) expect "audio/x-wav" or "audio/mpeg", and services that take images expect the "image/png", "image/jpeg", etc. A few services expect their input to be "structured text" that has been pre-segmented by the caller, for these the request must be presented in an ELG-defined JSON format. The response will be in JSON, in one of a variety of formats depending on the data type:

- *Standoff annotations* are represented in a style inspired by the format used by Twitter, each *type* of annotation mapping to a JSON array of objects referencing the start and end locations of the annotation (characters for text, fractional seconds for audio), and an optional set of *features*.
- *Classifications* of the whole input have their own format giving an ordered list of classes, each with an optional score.
- *New texts* such as translations of text or transcriptions of audio are returned in a structured format referred to as a "texts" response (note texts is plural). This is described in more detail below.
- *Audio* responses such as text-to-speech are still represented in JSON. Short snippets of audio can be returned inline in base 64 encoding, longer audio will typically be stored at a short-lived temporary URL for the caller to download via a separate HTTPS request.

The full specification of these response types can be found in the ELG documentation.[19] The "texts" response type is the most complex one as it is able to encode a nested tree structure of texts, where each node in the tree can be either a leaf node containing a single string of content, or a branch node containing another level of texts. The vast majority of services currently using this response format produce one of the three basic forms shown in Listing 1: a single text, a flat list of segments or alternatives, or a two-level list where each segment has a set of alternatives.

The property `role` is used to distinguish the cases. Not all services populate this property but it is encouraged; conventionally a role of "sentence", "paragraph" or "segment" denotes segments of text that are all part of the same transcript or translation, and "alternative" denotes different translations or transcriptions of the same

[18] UTF-8 encoding is the default but can be overridden by adding the `charset=...` parameter.

[19] https://european-language-grid.readthedocs.io/en/stable/all/A3_API/LTPublicAPI.html

```
// A single text
{
  "response":{
    "type":"texts",
    "texts":[
      {"content":"This is some text"}
    ]
  }
}

// A flat list of segments or alternatives
{
  "response":{
    "type":"texts",
    "texts":[
      {"content":"First sentence", "role":"sentence"},
      {"content":"Second sentence", "role":"sentence"},
    ]
  }
}

// A two level list of segments that each have a number of alternatives
{
  "response":{
    "type":"texts",
    "texts":[
      {
        "role":"sentence",
        "texts":[
          {"content":"Translation one", "role":"alternative"},
          {"content":"First translation", "role":"alternative"}
        ]
      },
      ...
    ]
  }
}
```

Listing 1 The three most common types of "texts" response

input segment. In the case of alternatives, each entry may also have a "score" representing the relative quality of the different options.

For errors (and also for warning messages), ELG, being a multilingual platform, uses a format designed to be amenable to internationalisation (i18n). Each message is represented as a JSON object with three properties "code", "text" and "params" (see Listing 2). The property "code" is the primary identifier for the error; there is a list of standard message codes provided in the ELG documentation but providers are free to create their own codes if the standard messages do not adequately cover their needs. The property "text" is a string for the message text in English, and it may include numbered placeholders `{0}`, `{1}`, etc. If the message has placeholders,

```
{
  "code":"elg.request.type.unsupported",
  "text":"Request type {0} not supported by this service",
  "params":["audio"]
}
```

Listing 2 An example "status message" object from the ELG API, designed to be easily translated into many languages.

```
POST https://live.european-language-grid.eu/i18n/resolve?lang=fr
Content-Type: application/json

[
  {
    "code":"elg.request.type.unsupported",
    "text":"Request type {0} not supported by this service",
    "params":["audio"]
  }
]

// response
Content-Type: application/json

["La demande du type audio n'est pas supportée par ce service"]
```

Listing 3 Resolving a status message to a translated string

the corresponding values are given in the "params" array (as a zero-based index, so 0 refers to the first item, 1 to the second, etc.). The error message may also include an optional "detail" object providing more technical details about the error.

The standard ELG message codes have translations into a number of different languages (twelve at the time of writing, with more in the pipeline), and ELG provides a special API endpoint that accepts an array of errors and an ISO 639 language code, and returns an array of message strings in the requested language (if available) with all placeholders filled in. If the requested message code is not available in that language the endpoint falls back to English, and if the message code is not known at all then the "text" fallback from the original error is used instead.

Listing 3 shows an example of calling the "resolver" API; the `?lang=...` parameter specifies the desired language. If it is not provided then the resolver will respect any `Accept-Language` HTTP header on the request.[20] If no language is requested by the parameter or the header then messages will be returned in English by default.

Some long running services will return more meaningful progress updates as they work through their various stages of processing, and these updates will be passed back to the caller if they use the asynchronous API mode – requests to the polling

[20] For browser-based clients this will typically result in the messages being returned in the user's preferred browsing language.

URL for a given job will return the latest progress update if the process is not yet complete. These updates are represented as i18n message objects in the same way as the errors and warnings described above, and they can be resolved to strings using the same resolver API endpoint.

4 Python SDK for Users

ELG provides many APIs to access the catalogue and search for specific resources, to download corpora hosted in ELG, to call services or many other uses (see Section 3). This provides ELG users with a lot of flexibility in the way they want to interact with the platform, however, the basic APIs are rather low level. For example, the search endpoint is paginated and returns only 20 results per call, which means that multiple API calls are needed to obtain more than 20 results. Similarly, calling a service via the public LT service API in the asynchronous mode requires multiple API calls to be made at the correct times and in the correct sequence to perform what is, from the user's perspective, a single action.

In order to simplify interactions with the platform, we developed a Python SDK that operates on top of the various ELG APIs and provides simple methods to easily interact with ELG and consume the resources in Python. We chose Python as the language for this first ELG SDK as it is probably the most widely-used programming language within the LT community.

The SDK is included in the ELG Pypi package which can be installed using the `pip` command familiar to any Python programmer. The basic SDK for consumer use is installed using `pip install elg`. The SDK provides access to most ELG functions through Python. It provides access to the catalogue with methods that allow users to search the catalogue and look for corpora, services, and organisations. The SDK enables users to call ELG-compatible services, and even to combine them using a simple pipeline mechanism.

4.1 Browsing the Catalogue

The SDK enables access to the ELG catalogue. It uses the same filters as the UI, i. e., we can filter for the type of resource or LT service, languages and licence; free text search can also be used. Listing 4 shows how to search for an English to French machine translation service. The SDK handles issues such as pagination automatically and returns the result as a list of entities, where each entity is a Python object that encapsulates the information about the respective ELG resource.

```
from elg import Catalog

catalog = Catalog()

# Search and get the result as a list of Python objects
results = catalog.search(
    resource = "Tool/Service", # "Corpus", "Lexical/Conceptual
                               # resource" or "Language
                               # description"
    function = "Machine Translation", # only for "Tool/Service"
    languages = ["en", "fr"],  # string or list if multiple
                               # languages
)
```

Listing 4 Example code to use the ELG catalogue

4.2 Downloading a Resource

The Python SDK has a `Corpus` class that corresponds to a corpus or data set. It can be initialised using the identifier of the resource. If the resource is stored in ELG, it can be downloaded using the `download` method of the `Corpus` class. Listing 5 shows the most simple usage and parameters are available to choose the distribution or specify the download location for example.

```
from elg import Corpus

corpus = Corpus.from_id(913) # initialise the Corpus using its ID
corpus.download()            # download corpus method
```

Listing 5 Example code to download an ELG corpus

4.3 Obtaining an Access Token

Some functions are restricted to authorised users of ELG (see Section 5). For the restricted APIs, an access token must be retrieved to identify the user behind the API call. It is possible to obtain a short-lived valid access token through the UI but this is not convenient for programmatic use.

To address this limitation, the Python SDK includes the `Authentication` class that interacts directly with the ELG OpenID Connect authentication service to obtain tokens, i. e., the access token to authenticate the API call and the refresh token which is used to refresh the access token when it expires.

```
from elg import Authentication

auth = Authentication.init()
# here the user is asked to authenticate in the browser

auth = Authentication.init(scope="offline_access")
# here we are requesting an ``offline'' token that remains valid until
# revoked, as opposed to the usual token that requires re-authentication
# after 6 hours

auth.to_json("tokens.json") # export the tokens to a json file

auth = Authentication.from_json("tokens.json")
# creation of an Authentication object from the tokens in the json file
```

Listing 6 Example of code to obtain, store, and retrieve authentication tokens

Listing 6 shows an example usage of the `Authentication` class. During the process, the user has to authenticate using their browser and paste the resulting authorisation code back to the Python program. Once the `Authentication` object is initialised, it is possible to save the tokens in a `json` file and reuse them. Obtained tokens are by default valid for only six hours. It is possible to get tokens that are valid indefinitely by setting the `scope` parameter to `offline_access`.

4.4 Calling Language Technology Services

The `Service` class of the Python SDK corresponds to an ELG LT service, and can be initialised using the identifier of the service. As users need to be authenticated to use ELG services, a login step is necessary. Alternatively, it is possible to provide an `Authentication` object or a `json` file containing the tokens during the initialisation of the service, which allows the login step to be skipped. Various ways of authenticating during the service initialisation of a service are shown in Listing 7.

A service that is initialised in Python can be called easily (see Listing 8). The Python SDK handles the creation of the input message, any necessary refreshing of the access token, the communication with the REST API, etc.

When calling a service, the input request can be provided in various formats: a plain text, a path to a text or an audio file, or a `Request` object.[21] The result is a Python object that corresponds to one of the response messages (see Section 3.3).

[21] https://european-language-grid.readthedocs.io/en/stable/all/A1_PythonSDK/notebooks/Service.html#Usage

```
from elg import Service

lt = Service.from_id(474) # login step necessary (unless tokens are cached) and the tokens will expire after 6 hours
lt = Service.from_id(474, scope="offline_access") # login step necessary (unless tokens are cached) and the tokens will never expire
lt = Service.from_id(474, auth_object=auth) # 'auth' is an Authentication object. No login step and the expiration of the tokens depends on the `auth` object
lt = Service.from_id(474, auth_file="tokens.json") # file containing existing tokens. No login step and the expiration of the tokens depends on the scope used to create them
```

Listing 7 Different ways of providing authentication during Service initialisation

```
from elg import Service

lt = Service.from_id(474) # initialise LT service using its ID
result = lt("Nikola Tesla did not live in Berlin.") # run service
print(result)
```

Listing 8 Example code for calling an ELG service

5 User Authentication

While general exploration and search in the ELG catalogue is open to all, various other operations in ELG are restricted to certain users. For example, access to the LT service public API (via the Python SDK, curl or the "try out" UIs) requires the caller to be logged in so that the platform can enforce API call quotas to limit how much data can be processed by each user per day, following the ELG licensing strategy (see Section 6). Similarly, the submission of new resources and metadata records is limited to users who are registered as *providers*; administrative tasks are restricted to the technical ELG team.

Registering a regular user account is a simple self-service procedure. The registration form is available through the sign up/sign in icon in the top right corner of the catalogue page. All registered users are assigned the *consumer* role by default. To get *provider* status, users can submit a request through their profile page. All other roles are assigned internally by the ELG administrators.

ELG uses Keycloak[22], a user management, authentication and authorisation server based on the OAuth2 and OpenID Connect[23] standards. Keycloak supports both interactive authentication of users through the web UI, and programmatic access to the REST APIs using JSON Web Tokens. Users sign in to Keycloak, then they (or

[22] https://www.keycloak.org

[23] https://openid.net/connect/

the client tool they are using, such as the ELG Python SDK) can acquire an access token, which is a cryptographically signed "permit" that encodes their identity and permissions. API endpoints can verify the validity of the token by checking its signature, and then make access decisions based on the "claims" encoded in the token without needing to check every request directly with the authentication server.

The adoption of OpenID Connect opens up the possibility for third party applications to allow their own users to authenticate using ELG accounts, in the same way as many existing websites and applications support "sign in with Google" or "sign in with Facebook". The OpenID Connect specification allows this without compromising the protection of users' personal information. When a given user attempts to "log in with ELG" to a particular third party application for the first time, Keycloak requires the user to grant *explicit consent* before any of their data is shared with the provider, and that consent can be revoked at any time. At the time of writing the first proof of this concept is under development with one of the ELG pilot projects.

6 Licensing and Billing

ELG includes mechanisms that support the consumption of services and resources that are available without any restrictions in terms of commercial aspects. It supports the download of resources under the condition that they are offered free of charge with open access licences or with restrictive licences that require only user authentication and, optionally, accepting the licensing terms. Technical safeguards have been implemented to ensure that access to LRTs is granted in accordance with the above terms, for example, access to LRTs distributed with restricted licences is made available only to those users that fulfil the criteria specified in the licences. With regard to LT services, only the "try out" functionality is available and only for registered users. Each user has two independent daily quotas for the quantity of data processed, one for plain text and the other for binary (audio or image) data, to reflect the fact that binary formats generally require much more data than plain text.

In addition, we also designed and implemented the prototype of a billing module that will enable ELG to offer resources and services distributed with commercial licences. The module is based on the commercial platform Chargebee, which was selected because it fulfilled our requirements: it ensures security and includes various services, such as handling subscriptions, payments, pricing, taxes, emails, ensuring customer satisfaction and conformance to all EU and national laws, and offers several functionalities, such as checkout pages, self-service after the payment, cancellation, creating and managing subscription plans, subscription changes, etc. The integration of the external billing module is based on the interaction between the two platforms, ELG and Chargebee. Information about the pricing of a resource or service is formally encoded in the metadata record in ELG; administrative and execution costs may also be added and calculated on the ELG side. In the Chargebee catalogue we maintain a set of all monetised products and plans, and their prices.

The relationship between the ELG catalogue products and the Chargebee catalogue is not necessarily one-to-one; Chargebee can contain paid plans that allow the use of multiple products from the ELG catalogue, or the download of multiple resources. The relation between the two catalogues depends on the ELG business strategy. All transactions, subscription changes, logs, billing information, subscription data and similar information are stored on the Chargebee side, i. e., a database that is external to ELG. Any information needed from Chargebee can be synchronised through a webhook mechanism. For the ELG platform, this information includes the identity of the user who has performed an action through a subscription plan and/or a purchase, the action performed, the billing plan to which the user subscribed, etc. Chargebee sends this information via HTTPS POST to the ELG back end so that it can register changes in the ELG platform. The ELG back end monitors the user's quota usage and, taking into account the user's subscription plans from the Chargebee platform, decides whether to allow or block a request for running a service. A similar procedure is used for the download of a purchased resource.

7 Consumer-Related Functionalities in ELG and other Platforms

In this section we present platforms and catalogue-based systems that share features with ELG, with a special focus on functionalities for consumers.

7.1 Catalogue and Repository Functionalities

With regard to the presentation and organisation of the contents of such a digital catalogue of artefacts, the users of ELG can see all types of entities on the same page or go through quick links from the top menu to the subset that interests them. Offering such resource type-specific filtering functionalities is an approach adopted by many catalogues, for example, Hugging Face[24] has separate pages for models and datasets, Papers with Code[25] for datasets and benchmarks, some CLARIN centres distinguish between data resources and services (e. g., CLARIN-PL[26], etc.), the European AI on demand platform[27] maintains separate catalogues for AI assets, organisations, projects and educational resources. This approach is particularly useful for expert users with clear search objectives. In addition, distinguishing between separate resource types allows for the selection of different metadata elements and subgroupings of entries along the parameters most suitable to each type (e. g., grouping together services based on the tasks they perform or the degree of complexity

[24] https://huggingface.co
[25] https://paperswithcode.com
[26] https://clarin-pl.eu
[27] https://www.ai4europe.eu

of use, and datasets based on modality or language). On the other hand, the one-size-fits-all page has the benefit of allowing users to have an overview of resources and activities using the same set of filters. ELG combines the two approaches by providing quick links in the top menu and filters for the targeted pages.

With regard to search functionalities, free text search is the most popular one. In some cases, an autocomplete function (e. g., Hugging Face) is used while advanced queries are less used. Faceted search is also common, but in most cases with limited facets (e. g., European AI on Demand platform, Hugging Face, etc.). Search with programmatic modes through REST APIs is offered by many platforms on a limited set of metadata elements in the same way that ELG does.

With regard to the functionalities offered for hosted data resources, direct download of open access resources is common. A download link that can be used from outside the platform (e. g., through a command line mode, or as a URL link) is provided in most cases. The deployment of integrated services on hosted resources is a feature offered in only a few platforms (e. g., OpenMinTeD, clarin:el[28]). Machine Learning platforms, like Hugging Face, can feed hosted datasets into applications, but this is not among the objectives of the ELG platform.

7.2 Language Technology Service Execution

ELG's LT service execution functionality has been designed and implemented from scratch. Below, we compare this functionality with similar related infrastructures or frameworks and highlight the similarities and differences in various aspects, e. g., interchange format, trial/visualisation UIs and support of workflows.

The DKPro[29] family of tools and resources (Gurevych et al. 2007) consists of a growing number of projects addressing different NLP tasks and aspects, such as pre-processing, machine learning, and lexical resources. It offers a collection of tools wrapped as UIMA components (Unstructured Information Management Architecture)[30], i. e., the components implement the interfaces and specifications of the UIMA framework. A UIMA reader component should extend the `ResourceCollectionReaderBase` class and also implement the `getNext(CAS aJCas)` method. A processor must extend `JCasAnnotator_ImplBase` and, furthermore, implement `process(JCas aJCas)` and a writer extends `JCasFileWriter_ImplBase` and implements `process(JCas aJCas)`. A UIMA reader loads data from a text file and creates a Common Analysis System (CAS) object. A processor gets a CAS object, runs the wrapped NLP tool and adds the results to the CAS object. A writer gets a CAS object and serialises its content to a file in a specific format. UIMA is Java-based but it can be used to wrap non-Java tools as well. UIMA allows to programmatically define pipelines (workflows), i. e., chain a reader, various processors, a

[28] https://inventory.clarin.gr

[29] https://dkpro.github.io

[30] https://uima.apache.org

writer and run the pipeline locally; it does not run remote services as in the case of ELG. The DKPro components are interoperable because they all follow the DKPro typesystem[31], which defines which annotations can be added to a CAS object, which features an annotation can contain, how these are serialised etc. The typesystem is actually an ontology for annotations, how they are organised etc. The ELG JSON format does not follow a typesystem. Another difference with ELG is that a CAS object is serialised (by default) in XML Metadata Interchange (XMI) format[32], a standard for exchanging metadata information via XML; other formats are also supported. If the results of a DKPro pipeline are exported in an appropriate format (e. g., XMI) they can be loaded, visualised and even edited with the annotation tool INCEpTION[33] (Klie et al. 2018), which is not possible in the ELG trial UIs.

GATE[34] (Cunningham et al. 2013) is an open source toolkit capable of solving numerous text processing problem. The GATE framework is written in Java and similar to DKPro/UIMA. As with UIMA there are additional modules to support integration with non-Java tools. It allows creating, either via a UI builder or programatically, a pipeline of NLP tools for specific tasks. The completed pipeline can be saved in the XML "recipe" format XGAPP, which can, in turn, be loaded into the developer UI to process small numbers of documents and visualise the resulting annotations, run using a batch processing tool for larger scale processing, or packaged as a service on either the ELG or GATE's own GATE Cloud platform (see Chapter 7, Section 4.2, 140 ff.). Each GATE processing component gets as input a `GATE Document` which is enriched with annotations. Again, as in DKPro, GATE readers and writers load the data and write the processing results. A `GATE Document` is by default serialised to GATE XML, however, other formats are also supported. The annotations that are added in `GATE Document` do not follow a specific typesystem but follow some generic rules – each document has one or more sets of annotations, each set can contain annotations of many types, each annotation can have zero or more features, and while there is no *enforced* typesystem, all standard GATE components share a set of informal conventions for the types and features they use. This logic is very similar to the one adopted in ELG's JSON-based format. Contrary to ELG, the DKPro/UIMA and GATE tools are not dockerized (by default) and run as command line tools locally. Furthermore, the ELG services always process raw text while DKPro and UIMA components can also handle other formats such as PDF, and documents that have already been partially annotated.

GATE Cloud[35] (Tablan et al. 2013) is a platform very similar in spirit to ELG, but specifically built around the requirements of GATE-based text analysis tools. It was developed by the same team at the University of Sheffield that was responsible for the initial design of the ELG LT service execution layer and thus shares many of the same API design decisions. GATE Cloud offers a REST API accepting documents

[31] http://dkpro.github.io/dkpro-core/releases/1.8.0/docs/typesystem-reference.html

[32] https://www.omg.org/spec/XMI/2.5.1/About-XMI/

[33] https://inception-project.github.io

[34] https://gate.ac.uk

[35] https://cloud.gate.ac.uk

via HTTP post and returning annotations in the native JSON or XML formats of the GATE framework. GATE Cloud services process only text (not audio or other media types), but can accept formats such as XML, PDF (with machine-readable text) or Word documents as well as plain text. As well as the single document API, GATE Cloud also supports batch processing of larger amounts of data using on-demand processing capacity from Amazon Web Services. GATE Cloud services are defined as XGAPP "recipes" in the native GATE format, which are wrapped as Docker containers for the REST API or executed as-is by the batch processing engine. GATE Cloud has recently added support for other types of APIs such as image OCR (a service which has itself been integrated into the ELG platform).

The LAPPS Grid platform, as DKPro, is based on a typesystem, the LAPPS Web Service Exchange Vocabulary (Ide et al. 2016), "an ontology of terms for a core of linguistic objects and features exchanged among NLP tools that consume and produce linguistically annotated data. It is intended to be used for module description and input/output interchange to support service discovery, composition, and reuse in the natural language processing domain." In LAPPS Grid, as in ELG, tools are wrapped as web services, packaged as Docker images and exchange JSON messages. However, LAPPS Grid also offers workflows by using Galaxy, a workflow management system. Galaxy includes a visual editor for creating and parameterising workflows and an engine for executing these workflows. LAPPS Grid does *not* have a catalogue and each service is described with a limited set of metadata elements that are required for adding it to the Galaxy tool inventory. ELG was not designed to offer workflows, i. e., it does not include a workflow editor or a workflow execution engine. In addition, all services get as input raw text and they were not designed for playing the role of components in a workflow. However, some pipelines can be created by using external tools, e. g., the Python SDK and some code/adapters (Rehm et al. 2020; Moreno-Schneider et al. 2022). For example, using the ELG Python SDK, a Machine Translation service can be called, the result can extracted from the output JSON message and fed to an ELG NER service.

The OpenMinTeD execution service (Labropoulou et al. 2018) is also built on top of Galaxy. A large number of tools from the DKPro and GATE collections were ingested to OpenMinTeD. Several tools from other providers were also added. All tools were dockerized and are executed inside the container as command line tools, i. e., not as web services. An OpenMinTeD workflow is executed by running a series of Docker images (one after the other) in a cluster managed by Mesos[36], a framework similar to Kubernetes[37]. The workflow itself is created using the Galaxy editor. In OpenMinTeD no specific interchange format was enforced, the recommendation was to use the DKPro typesystem and XMI serialization. However, the GATE tools were using GATE XML format and several others were using their own custom format (e. g., based on JSON). In order to create a "mixed" workflow the creator had to combine the respective components with corresponding format adapters. If the results of

[36] https://mesos.apache.org

[37] https://kubernetes.io

the workflow were in XMI format, they could be visualised using WebAnno[38], a predecessor of INCEpTION.

The European AI on Demand platform[39] covers the whole European AI landscape rather than being restricted to LT or NLP. For example, computer vision is also included. The services are gRPC-based (not REST-based as in ELG) and are packaged as Docker images. The messages that they consume and produce are based on the ProtoBuf serialisation format[40] and no specific typesystem is used. The platform does not offer an execution environment. However, the worfklows that are created with the AI4EU Experiments editor[41], an editor similar to the one offered by Galaxy, are exported to a format that allows their execution in a Kubernetes cluster.

Hugging Face offers a large collection of Transformer-based models for computer vision, language processing, audio processing etc. Transformers are a specific type of neural networks (Vaswani et al. 2017) that have revolutionised machine learning since they achieve state of the art results in many tasks. Hugging Face allows training of Transformer-based models via the AutoNLP API[42], which is not free of charge. While we have performed initial experiments, ELG does not offer integrated model training. In Hugging Face, training as well model deployment is based on Amazon SageMaker, which is built on top of Docker. Hugging Face users can call a model via the trial UIs/widgets that are embedded in the respective page (as in ELG). For doing the same in a programmatic way, Hugging Face offers an inference REST API along with a Python client API[43]. Similar inference functionalities are offered through the ELG REST APIs and the Python SDK. Upon request, Hugging Face also offers an inference solution delivered as a container with the Transformer model for on-premise usage.[44] It can be used via a HTTP API (as in ELG). Finally, Hugging Face has developed a Python-based library (called "transformer") that allows to download a model and either fine-tune it in a specific task or use it for inference. Such functionality is not offered by the ELG Python SDK.

8 Conclusions

The ELG platform has fully achieved all objectives it had set for serving consumers. It allows consumers to browse through the whole ELG catalogue, already populated with more than 13,000 metadata records, apply faceted filtering and exploration, search for specific resources and services, download them (if hosted in ELG) and try out more than 800 functional services, both basic processing NLP services and

[38] https://webanno.github.io/webanno/

[39] https://www.ai4europe.eu

[40] https://developers.google.com/protocol-buffers

[41] https://aiexp.ai4europe.eu

[42] https://huggingface.co/autotrain

[43] https://api-inference.HuggingFace.co/docs/python/html/quicktour.html

[44] https://HuggingFace.co/infinity

end-to-end applications. Users can also access the directory of LT-developing companies and academic organisations, find organisations active in a specific LT area, and initiate collaborations with them. The links between LRTs, organisations and projects allows users to navigate between them and have an overview of the overall European LT landscape. Consumers can access all these functionalities through user-friendly web user interfaces, or in programmatic ways, using the public REST APIs and Python SDK.

References

Cunningham, Hamish, Valentin Tablan, Angus Roberts, and Kalina Bontcheva (2013). "Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics". In: *PLOS Computational Biology* 9.2, pp. 1–16. DOI: 10.1371/journal.pcbi.1002854.

Data Citation Synthesis Group (2014). *Joint Declaration of Data Citation Principles – FORCE11*. Ed. by M. Martone. DOI: 10.25490/a97f-egyk. URL: https://doi.org/10.25490/a97f-egyk.

Gurevych, Iryna, Max Mühlhäuser, Christof Müller, Jürgen Steimle, Markus Weimer, and Torsten Zesch (2007). "Darmstadt Knowledge Processing Repository based on UIMA". In: *Proc. of the First Workshop on Unstructured Information Management Architecture (co-located with GLDV 2007)*. Tübingen, Germany, p. 89.

Ide, Nancy, Keith Suderman, Marc Verhagen, and James Pustejovsky (2016). "The Language Application Grid Web Service Exchange Vocabulary". In: *Worldwide Language Service Infrastructure*. Lecture Notes in Computer Science. Springer, pp. 18–32.

International Organization for Standardization (2007). *Codes for the representation of names of languages – Part 3: Alpha-3 code for comprehensive coverage of languages*. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/95/39534.html.

Klie, Jan-Christoph, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych (2018). "The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation". In: *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018): System Demonstrations*. Santa Fe, USA: ACL, pp. 5–9. URL: http://tubiblio.ulb.tu-darmstadt.de/106270/.

Labropoulou, Penny, Dimitris Galanis, Antonis Lempesis, Mark Greenwood, Petr Knoth, Richard Eckart de Castilho, Stavros Sachtouris, Byron Georgantopoulos, Stefania Martziou, Lucas Anastasiou, Katerina Gkirtzou, Natalia Manola, and Stelios Piperidis (2018). "OpenMinTeD: A Platform Facilitating Text Mining of Scholarly Content". In: *Proceedings of WOSP 2018 (co-located with LREC 2018)*. Miyazaki, Japan: ELRA, pp. 7–12. URL: http://lrec-conf.org/workshops/lrec2018/W24/pdf/13_W24.pdf.

Melnika, Julija, Andis Lagzdiņš, Uldis Siliņš, Raivis Skadins, and Andrejs Vasiļjevs (2019). *Deliverable D3.1 Requirements and Design Guidelines*. Project deliverable; EU project European Language Grid (ELG); Grant Agreement no. 825627 ELG. URL: https://www.european-language-grid.eu/wp-content/uploads/2021/02/ELG-Deliverable-D3.1-final.pdf.

Moreno-Schneider, Julián, Rémi Calizzano, Florian Kintzel, Georg Rehm, Dimitris Galanis, and Ian Roberts (2022). "Towards Practical Semantic Interoperability in NLP Platforms". In: *Proceedings of the 18th Joint ACL-ISO Workshop on Interoperable Semantic Annotation (ISA 2022; co-located with LREC 2022)*. Ed. by Harry Bunt. Marseille, France, pp. 118–126. URL: http://www.lrec-conf.org/proceedings/lrec2022/workshops/ISA-18/pdf/2022.isa18-1.16.pdf.

Phillips, Addison and Mark Davis (2009). *Tags for Identifying Languages*. Tech. rep. RFC 5646. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/rfc5646.

Rehm, Georg, Dimitrios Galanis, Penny Labropoulou, Stelios Piperidis, Martin Welß, Ricardo Usbeck, Joachim Köhler, Miltos Deligiannis, Katerina Gkirtzou, Johannes Fischer, Christian Chiarcos, Nils Feldhus, Julián Moreno-Schneider, Florian Kintzel, Elena Montiel, Víctor Ro-

dríguez Doncel, John P. McCrae, David Laqua, Irina Patricia Theile, Christian Dittmar, Kalina Bontcheva, Ian Roberts, Andrejs Vasiljevs, and Andis Lagzdiņš (2020). "Towards an Interoperable Ecosystem of AI and LT Platforms: A Roadmap for the Implementation of Different Levels of Interoperability". In: *Proc. of the 1st Int. Workshop on Language Technology Platforms (IWLTP 2020, co-located with LREC 2020)*. Ed. by Georg Rehm, Kalina Bontcheva, Khalid Choukri, Jan Hajic, Stelios Piperidis, and Andrejs Vasiljevs. Marseille, France, pp. 96–107. URL: https://www.aclweb.org/anthology/2020.iwltp-1.15.pdf.

Rehm, Georg, Stelios Piperidis, Kalina Bontcheva, Jan Hajic, Victoria Arranz, Andrejs Vasiļjevs, Gerhard Backfried, José Manuel Gómez Pérez, Ulrich Germann, Rémi Calizzano, Nils Feldhus, Stefanie Hegele, Florian Kintzel, Katrin Marheinecke, Julian Moreno-Schneider, Dimitris Galanis, Penny Labropoulou, Miltos Deligiannis, Katerina Gkirtzou, Athanasia Kolovou, Dimitris Gkoumas, Leon Voukoutis, Ian Roberts, Jana Hamrlová, Dusan Varis, Lukáš Kačena, Khalid Choukri, Valérie Mapelli, Mickaël Rigault, Jūlija Meļņika, Miro Janosik, Katja Prinz, Andres Garcia-Silva, Cristian Berrio, Ondrej Klejch, and Steve Renals (2021). "European Language Grid: A Joint Platform for the European Language Technology Community". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations (EACL 2021)*. Kyiv, Ukraine: ACL, pp. 221–230. URL: https://www.aclweb.org/anthology/2021.eacl-demos.26.pdf.

Smith, Arfon M., Daniel S. Katz, and Kyle E. Niemeyer (2016). "Software citation principles". In: *PeerJ Computer Science* 2. URL: https://peerj.com/articles/cs-86.

Tablan, Valentin, Ian Roberts, Hamish Cunningham, and Kalina Bontcheva (2013). "GATECloud.net: A Platform for large-scale, Open-Source Text Processing on the Cloud". In: *Philosophical Transactions of the Royal Society A: Math., Phys. and Eng. Sciences* 371.20120071.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010.

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A.C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons (2016). "The FAIR Guiding Principles for Scientific Data Management and Stewardship". In: *Scientific Data* 3. DOI: 10.1038/sdata.2016.18. URL: http://www.nature.com/articles/sdata201618.

Wu, Mingfang, Fotis Psomopoulos, Siri Jodha Khalsa, and Anita de Waard (2019). "Data Discovery Paradigms: User Requirements and Recommendations for Data Repositories". In: *Data Science Journal* 18.1. URL: http://datascience.codata.org/articles/10.5334/dsj-2019-003/.