Proceedings of ISA-18 Workshop at LREC2022, pages 118–126
Marseille, 20 June 2022

Towards Practical Semantic Interoperability in NLP Platforms

Julian Moreno-Schneider[1], Remi Calizzano[1], Florian Kintzel[1], Georg Rehm[1], Dimitrios Galanis[2], Ian Roberts[3]

[1] Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Germany – firstname.lastname@dfki.de • [2] ILSP, R. C. "Athena", Greece – galanisd@athenarc.gr • [3] University of Sheffield, UK – i.roberts@dcs.shef.ac.uk

Abstract

Interoperability is a necessity for the development of complex tasks that require the interconnection of several NLP services. This article presents the approaches that were adopted in three scenarios to address their respective interoperability issues. The first scenario describes the creation of a common REST API for a specific platform, the second scenario presents the interconnection of several platforms via mapping of different representation formats and the third scenario shows the complexities of interoperability through semantic schema mapping or automatic translation.

Keywords: Semantic Parsing, Semantic Mapping, NLP Platforms Interoperability

1 Introduction

For a long time, the development of NLP infrastructures or platforms that provide a larger number of NLP services was not practically feasible. In recent years this has changed with the appearance of new technologies, especially containerisation and microservices. The main advantage of these technologies is that they enable individual development as well as easy deployment and execution of NLP services, thus facilitating their re-use into one's own computing system/environment. However, individual development of tools especially when distributed among *different* organisations gives rise to interoperability (Rehm et al., 2020b) issues, e. g., the services are developed without a common data model (exchange format), with different technologies and APIs (gRPC, REST) etc. In short, the services cannot be easily combined. Achieving interoperability becomes more complex the greater the number of individual services involved.

For a further analysis of interoperability challenges encountered and potential methods of mitigation we look at three scenarios of different complexity. From simple to more challenging, these are:

- The first interoperability scenario is the situation where the services to be developed use the same format to carry out the processing and annotation of texts. An example of this type of interoperability is the European Language Grid (ELG), where a specific API has been defined that must be used by all the services that want to integrate into and be interoperable within the ELG infrastructure (see Section 2 for more details).

- In the second interoperability scenario we have two different platforms in which the services use different formats (regardless of the semantic schemas). For example, one service might use JSON as its input and output format, and another service might use XML. Interoperability between these two services (and possibly between the two platforms) can be achieved through the transformation of one format into the other, for which there are quite a few tools (SHACL[1], XSLT[2], etc.).

- In the third and most complex interoperability scenario, two services, from the same platform or from different platforms, use the same format as input and output, but the semantic schema they use is fundamentally different, i. e., the semantic information text documents are annotated with differs from one service to another. This difference not only implies that they can annotate different information, but even if they do annotate the same information, for example, an entity such as Berlin, one service uses an `ENTITY` tag and another services uses the tag `ENT` (see Listing 1). In addition, the different services can also use different categorisations to classify such entities, i. e., one service annotates Berlin as `CITY` while the other annotates it as `LOCATION`.

This article presents the work carried out so far to address the three interoperability scenarios mentioned above. The interoperability mechanisms developed in the European Language Grid (ELG) platform[3] to make its heterogeneous and growing set of tools and services compatible (common API specification) and interconnectable (Python SDK) are described in Section 2. The interoperability mechanisms implemented in the QURATOR[4] and SPEAKER[5] projects (in addition to

[1] https://www.w3.org/TR/shacl/
[2] https://www.w3schools.com/xml/xsl_intro.asp
[3] https://www.european-language-grid.eu
[4] https://qurator.ai
[5] https://www.speaker.fraunhofer.de

```
<?xml version="1.0" encoding="utf-8"?>
<text>
<Annotation_1>
  I was living in <ENTITY class="CITY">
    Berlin</ENTITY> last year
</Annotation_1>
<Annotation_2>
  I was living in <ENT class="LOCATION">
    Berlin</ENT> last year
</Annotation_2>
</text>
```

Listing 1: Example of annotations made by two services using the same format and different semantic schema

Lynx[6]) to mainly cover scenario 2, and minimally scenario 3 (Workflow Manager) are presented in Section 3. The proposed solution to solve scenario 3 as generically as possible is introduced in Section 4. Section 5 presents related work. Finally, Section 6 concludes the article and sketches directions for future work.

2 Interoperability through common API Specification (Scenario 1)

The European Language Grid (ELG) platform addresses interoperability by forcing the NLP services to use a predefined format. This corresponds to the first interoperability scenario described in Section 1.

2.1 Introduction to ELG

The ELG platform aims to offer multiple services that will support and boost the Language Technologies (LT) sector and activities in Europe, see Rehm et al. (2020a) for a short overview and Rehm (2022) for an exhaustive description that covers all the details.

Its primary goal is to provide a scalable system dedicated to the distribution and deployment of Language Resources and Technologies (LRT).[7] ELG offers access to thousands of commercial and non-commercial LTs and ancillary data LRs for all European languages and more. These include processing and generation services, tools, applications for written and spoken language, models, corpora, lexicons, ontologies, term lists, computational grammars, etc. Moreover, resources integrated in the ELG cloud infrastructure are directly deployable and/or downloadable.

ELG aims to act as a living observatory of LT, consolidating existing and legacy tools, services, LRs, and information about them, as well as newly emerging ones.

2.2 ELG Language Technology Services

Among the more than 12,000 resources available, at the time of writing the ELG catalogue counts more than 800 functional services deployed in the ELG infrastructure. Figure 1 shows an overview of the ELG platform architecture. An ELG service is a LT tool running inside the ELG Kubernetes cluster and takes the form of a Docker image exposing an ELG-compatible endpoint. An ELG service has to be compatible with the specifications (see Section 2.3) defined by the ELG team which aim to facilitate the deployment of the services but also standardise the LT tools. Currently, ELG supports the integration of tools/services that fall into one of the following broad categories:

- Information Extraction (IE) & text analysis: Services that take text input and produce standoff *annotations* over that text.
- Text-to-text: Services (most notably Machine Translation, but also summarisation, anonymisation, etc.) that take text and return new text that is derived from the input
- Text classification: Services that take text input and classify it somehow (e. g., language identification, "fake news" detection, etc.)
- Speech recognition: Services that accept audio and return a text transcription
- Text-to-speech: Services that take text and return audio

These broad categories cover the vast majority of NLP tasks and the respective specification can be easily extended if required.

2.3 Internal LT Service API Specification

ELG services are accessible from outside the ELG cluster via the LT execution server as shown in Figure 1. The communication between the LT execution server and each NLP/LT service is done using an internal application programming interface called Internal LT Service API. The respective specification details the API that the LT tool containers need to implement in order to be runnable as functional services within the ELG infrastructure. It consists of three request messages and four response messages that ELG functional services have to use as input and output of the LT tool. The three request messages are: Text request, Structured text request, and Audio request. The four response messages are: Annotations response, Classification response, Texts response, and Audio response. These seven messages have been created having two constraints in mind: being permissive to cover as many NLP use cases as possible and being specific to force similar message structures for similar services. They are described in detail in the ELG documentation.[8] Table 1 shows the number of services per service category presented in the previous section with, for each service category, the request and response messages used.

[6] https://lynx-project.eu

[7] https://live.european-language-grid.eu/catalogue

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

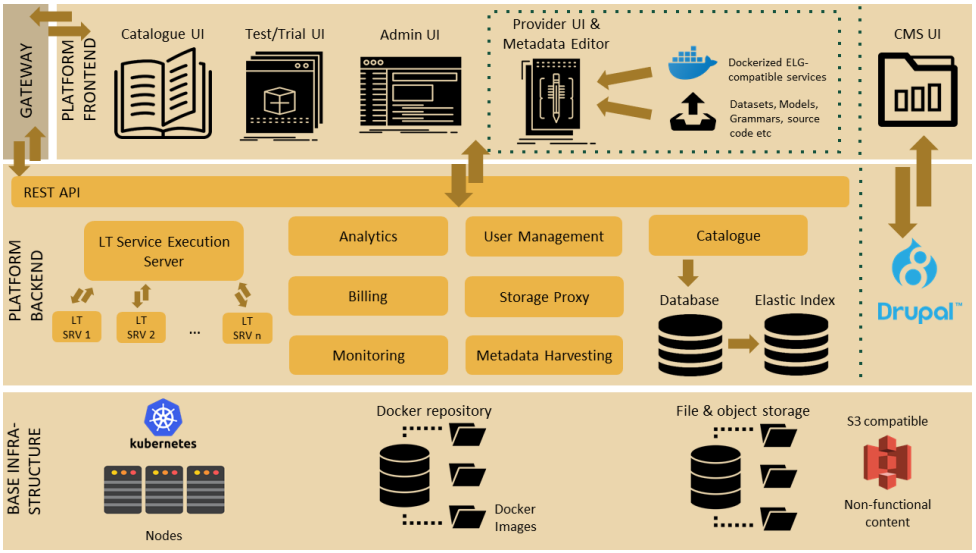

Figure 1: Architecture of the European Language Grid (ELG)

Category	No. of Services	Request Message	Response Message
IE & text analysis	273	Text	Annotations
Text-to-text	405	Text	Texts
Text classification	31	Text	Classification
Speech recognition	35	Audio	Texts
Text-to-speech	23	Text	Audio

Table 1: Number of ELG services per service category with the request and response type of messages used

```python
from elg import Service

# initialise LT service using its ID
lt = Service.from_id(17471)
# call LT service
result = lt("I was living in Berlin last
    year.")
print(result) # print the Annotations
    response message returned by the
    service
```

Listing 2: Calling an ELG service using Python

2.4 Calling ELG Services using the Python SDK

The ELG Python SDK provides access to most ELG functionalities through Python.[9] Among its other functionalities, the Python SDK enables users to call functional services available in ELG.

The `Service` class of the Python SDK corresponds to an ELG functional 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. A service that is initialised in Python can be called easily. Listing 2 shows the code for calling a named entity recogniser[10]. The Python SDK handles the creation of the input message, the update of the identification token, the communication with the API exposed by the LT execution server, etc.

2.5 Combining ELG Services

ELG provides access to a large variety of heterogeneous services from information extraction to text-to-speech generation (see Section 2). However, thanks to the internal LT service API specification, all the ELG services use the same message structures. Those specifications facilitate the combination of services, i. e., allows to determine if two services are compatible (can run one after the other without conversion) and, if not, it facilitates the required mapping. In the vast majority of cases, text-to-speech services returning an audio response can be used as input of a service accepting an audio request. This is also true for the texts response messages which can likely be converted into text request messages. The rare cases where the conversion is not possible happen when one of the fields that accepts arbitrary content of the response message is not empty. For the services returning an annotations or a classification response, the output cannot easily be converted into an input message for another service. These services are often used last. For the ELG services, the combination of the following categories is theoretically feasible based on the type of messages used (see Table 1): Text-to-text or Speech recognition with IE & text analysis, Text classification, or Text-to-speech, and Text-to-speech with Speech recognition.

The automatic conversion of an audio or a texts response to respectively an audio or a text request is done

[9] https://gitlab.com/european-language-grid/platform/python-client

[10] Cogito Discover Named Entity Recognizer, https://live.european-language-grid.eu/catalogue/tool-service/17471

```
from elg import Pipeline

# initialise the pipeline using LT
    services IDs
pipeline = Pipeline.from_ids([9183,
    4842])
# call the LT services
results = pipeline("I was living in
    Berlin last year.")
print(results[-1]) # print the Audio
    response message containing the
    audio of the sentence translated in
    German
```

Listing 3: Combining ELG services using the Python SDK and the Pipeline class

using a logical mapping between the fields of each message. For example, the `content` field of the texts response is mapped to the `content` field of the text request. Again, this mapping is possible thanks to the specifications defined by the ELG team. This mapping is not perfect and only the conversion of certain types of messages is possible.

The Python SDK introduced in Section 2.4 provides a `Pipeline` class to make it easy for users to interoperate ELG services. Listing 3 shows how to use the `Pipeline` class to run a machine translation service[11] following by a text-to-speech service[12]. This pipeline returns the German audio of an English sentence by combining two services automatically.

More example combinations using the `Pipeline` class of the Python SDK can be found in Annex A.

3 Intra-Platform Interoperability through Format Mapping (Scenario 2)

As presented in the introduction, interoperability scenario 2 occurs when two (or more) services (from the same or different platforms) use different formats (JSON, XML, RDF, etc.) for input and/or output of the information processed/to be processed.

The distinctive feature of this scenario is that the number of formats is limited, which allows the manual implementation of interoperability between all of them. This means that format conversions are accomplished through manually defined rules for each conversion step. In the following, we present all the implementations made to achieve interoperability in this scenario through a workflow manager (Moreno-Schneider et al., 2020).

3.1 Supported Platforms

The first format considered for integration has been the format used in the Lynx platform (Moreno-Schneider et al., 2021), a domain-specific platform for the generation and use of a Legal Knowledge Graph used and populated through various NLP services (Named Entity Recognition, Summarisation, Machine Translation, etc.), which were integrated and combined through a workflow manager developed in the project (based on Camunda[13]). We decided to use the Lynx format as the first one to be integrated into our workflow manager for a simple reason: the Lynx project used an internal Linked Data information format also based on NIF (Hellmann et al., 2013), which simplifies its integration into our format.

The second format integrated is the one used in the ELG platform (described in Section 2).

The third format is the one developed in the project SPEAKER[14], a platform for speech assistants for the German language, so the interaction with the platform is limited to short interactions (dialogues). The project has developed an API that defines input and output messages and their content, including internal services that should not be consulted externally.

3.2 Format Mapping

There is a very large number of formats to represent text (e. g., TXT, XML, JSON, RDF, etc.), which makes it impossible to develop all potentially necessary converters to cover all possible mappings between formats (TXT→XML, TXT→XML, RDF→JSON, etc.); note, though, that all of these different formats can potentially include a sheer endless number of specific individual approaches how to represent text.

To address this problem we decided to use an intermediate unified format, so that we do not need converters for each format-pairs, but only for the format-pairs including the intermediate unified format. We simplify the necessary work to scale the interoperability solution in future scenarios with new or different formats.

Considering that we are processing text semantically, the unified format we use not only allows easy handling of semantic information, but has been specifically defined for this purpose, i. e., Resource Description Framework (RDF[15]). RDF is used in Linked Data to represent semantic information, especially for ontologies and knowledge bases.

In order to better understand this conversion, we are going to use an XML example document (shown in Listing 4) annotated with two named entities (`Berlin` and `New York`). This document is converted into an RDF document containing exactly the same information (shown in Listing 5).

In the example, we can see that URIs (Unified Resource Identifier) are assigned to each annotation, i. e.,

[11] HelsinkiNLP – OPUS-MT (eng-gmw): English-German machine translation https://live.european-language-grid.eu/catalogue/tool-service/9183

[12] MaryTTS – German male (dfki-pavoque-neutral-hsmm) https://live.european-language-grid.eu/catalogue/tool-service/4842

[13] https://camunda.com

[14] https://www.speaker.fraunhofer.de

[15] https://www.w3.org/RDF/

```
<?xml version="1.0"?>
<text>
  I was living in <ENTITY class="
    LOCATION">Berlin</ENTITY> last year,
     but then I moved to <ENTITY class="
    LOCATION">New York</ENTITY>.
</text>
```

Listing 4: Example XML document annotated with semantic information (named entities)

piece of information (whole text, annotation, etc.). The two named entities in the example are converted into specific annotations, http://ex/#char=16,21 and http://ex/#char=54,61, which are completed with semantic information (properties and values). This annotation generation process has been manually defined for the specific XML format.
Furthermore, crucially, in the RDF format there is a lot more explicit information that was not directly present in the XML version, such as *beginIndex* and *endIndex*. This helps the inference of information in semantic systems, such as knowledge bases or graphs.

3.3 Workflow Manager

Our workflow manager is currently primarily used in the scenario of digital content curation (Bourgonje et al., 2017; Bourgonje et al., 2016) but its development was started with regard to the legal domain (Moreno-Schneider and Rehm, 2018; Rehm et al., 2019). One of its main objectives is the management of containerised tools, which need to interact with each other in a flexible and efficient way, even if they have been designed and developed independently. That forces the need for interoperability mechanisms because the services use different formats. Regarding the communication between components, for the moment, the workflow manager allows the usage of REST API (Richardson et al., 2013) and gRPC (Giretti, 2022) based services.
The workflow manager uses a unified intermediate format to represent information internally, and it uses RDF together with NIF (Hellmann et al., 2013). Besides, the different formats that are currently supported for translation in the workflow manager are:

- **Lynx Document:** This document has been defined and implemented in the Lynx project (Moreno-Schneider et al., 2021) based on Linked Data and NIF (Hellmann et al., 2013) whose main objective is the semantic representation of documents and information in the legal domain. This document can be represented in any Linked Data format, such as RDF, JSON-LD or XML.
- **ELG API Specification:** The specification made in ELG has been described in Section 2.3. As mentioned, messages in ELG use JSON format.
- **SPEAKER API:** The API defined in SPEAKER is focused on virtual assistants and not so much on actual language processing, although part of the specification can be adapted (`DialogueRequest` and `DialogueReponse`). The messages in SPEAKER are defined in the Protobuf (Varda, 2008) files using the gRPC protocol.

4 Semantic Interoperability between NLP Services (Scenario 3)

The last scenario pursues the interoperability of NLP services that use different semantic schemas. The term semantic schema refers to the way information is annotated in a document and to what semantic information it is related, i. e., how it is represented in a document that a word is a named entity and of what type.[16]

4.1 Semantic Interoperability Challenge

In scenario 1, we showed the possibility to perform a semantic mapping between the ELG services that are compatible (see Section 2.5). This is possible because all ELG services use the same limited set of messages, which is why we can define rules to map the fields of two different messages. However, in the context presented in the second scenario where the services are from different platforms, it is not possible to define rules as there are theoretically as many rules as the number of pairs of services because each service uses its own semantic schema.
This issue also applies to some ELG services, because as explained in Section 2.3, the ELG messages although based on a common format specification, this specification does not enforce a specific semantic schema for the annotations.
For example, the annotation types used in the `annotations` field of the Annotations response message[17] can differ from one service to another, and we find the same difference in the semantic schemas presented in Listing 1 where the two different annotation types `CITY` or `LOCATION` are used to represent the same entity.
Because of the various semantic schemas used by NLP services, which also are not formally described or documented it is impossible to create universal rules to semantically map messages from the same or different platforms using a different semantic schema.

4.2 Manual Semantic Mapping

The first solution to this problem used by the workflow manager consists of manually creating mappings for each pair of services. In practice, we create a correspondence between the fields of the output of the first service and the fields of the input of the second service. This solution works only when the number of services

[16] In UIMA, the semantic schema is called a typesystem.
[17] https://european-language-grid.readthedocs.io/en/stable/all/A3_API/LTInternalAPI.html#annotations-response

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix itsrdf: <http://www.w3.org/2005/11/its/rdf#> .
@prefix nif:    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix geo:   <http://www.w3.org/2003/01/geo/wgs84_pos/> .
<http://ex/#char=0,62>
        a                    nif:RFC5147String , nif:String , nif:Context ;
        nif:beginIndex       "0"^^xsd:nonNegativeInteger ;
        nif:endIndex         "62"^^xsd:nonNegativeInteger ;
        nif:isString         "I was living in Berlin last year, but then I moved to New York."^^xsd:string .
<http://ex/#char=16,21>
        a                   nif:RFC5147String , nif:String ;
        nif:anchorOf        "Berlin"^^xsd:string ;
        nif:beginIndex      "16"^^xsd:nonNegativeInteger ;
        nif:endIndex        "21"^^xsd:nonNegativeInteger ;
        nif:referenceContext  <http://ex/#char=0,62> ;
        itsrdf:taClassRef   <http://dbpedia.org/ontology/Location> .
<http://ex/#char=54,61>
        a                     nif:RFC5147String , nif:String ;
        nif:anchorOf          "New York\"^^xsd:string ;
        nif:beginIndex        "54"^^xsd:nonNegativeInteger ;
        nif:endIndex          "61"^^xsd:nonNegativeInteger ;
        itsrdf:taClassRef     <http://dbpedia.org/ontology/Location> ;
        nif:referenceContext  <http://ex/#char=0,62> .
```

Listing 5: Example RDF document using semantic NIF annotations

is rather limited which is the case for scenario 2, for example. However when the number of services is getting bigger the manual solution is no longer viable.

4.3 Automatic Semantic Mapping

A second approach is to make use of recent advances in machine learning and NLP to create a mapping for each couple of services automatically. The idea is to create the mapping rule once so that we do not have to recreate it each time two services are combined to not increase inference time. Here, we assume that the semantic schemas of the services are known in advance. This task can be called automatic semantic mapping rule generation and consists of finding the mapping rule between the semantic schemas of two NLP services. This mapping rule could be used afterwards to interoperate the two services by converting the first service's semantic schema to the semantic schema of the second service.

The mapping rule can take multiple formats like, for example, a Python method that takes the message from the first service as input and returns the same message with a different semantic schema compatible with the second service. We made preliminary experiments with GPT-3 using the OpenAI playground[18] with the `text-davinci-002` model. We tested various prompts and parameters to see if it is possible to generate automatically a Python method that maps two semantic schemas. Our preliminary results are promising because in some of the experiments we have been able to show that this is indeed possible. Listing 6 presents an example in which GPT-3 created a Python method capable to convert a simple semantic schema into another one. However, the model shows its limitations when it comes to mapping complex semantic schemas as shown in Listing 7. We performed these experiments only to demonstrate that advanced NLP models can be used to perform automatic semantic mapping rule generation and can potentially solve, or contribute to solving, the semantic interoperability challenge. We will explore and evaluate various automatic mapping methods with larger experiments in future work.

5 Related Work

5.1 NLP Platforms Semantic Schemas

Beyond the platforms we used for our experiments, there are more and more AI platforms offering NLP services via APIs such as HuggingFace (Wolf et al., 2020), Google Cloud Platform[19] (GCP), or OpenAI[20], or via tools such as NLTK (Bird et al., 2009), Spacy[21] or Stanza (Qi et al., 2020). We can divide these platforms into two categories. First, those with a limited number of services like GCP, OpenAI, and Stanza which only offer their services, and second the platforms like HuggingFace and Spacy where there is a wide variety of models and where it is possible to add more. In both categories, the platforms have defined one or more semantic schemas that the services need to use. HuggingFace has, for example, the no-

[18] https://beta.openai.com/playground/

[19] https://cloud.google.com/natural-language

[20] https://openai.com/api/

[21] https://spacy.io

```
"""
Python method that converts
[{"translation_text": "This is a plain
    text sentence."}]
to
{"type":"text", "content":"This is a
    plain text sentence."}
"""

def convert_to_text(translation_response
    ):
    return {"type": "text", "content":
    translation_response[0]["
    translation_text"]}
```

Listing 6: Example Python method generated by GPT-3 to convert a semantic schema into another. Only the docstring (text in green) is provided and the Python code is generated by the model. The generated Python method is valid and correctly converts the first semantic schema to the second one.

```
"""
Python method that converts
<?xml version="1.0" encoding="utf-8"?><
    text>I was living in <ENTITY class="
    LOCATION">Berlin</ENTITY> last year,
     but then I moved to <ENTITY class="
    LOCATION">New York</ENTITY>.</text>
to
{"response":{"type":"annotations","
    annotations":{"Place":[{"start":16,"
    end":22,"features":{"Lemma":"Berlin
    "}},{"start":54,"end":62,"features
    ":{"Lemma":"New York"}}]}}}
"""

import json
import xml.etree.ElementTree as ET

def convert_xml_to_json(xml_string):
    root = ET.fromstring(xml_string)
    annotations = []
    for child in root:
        if child.tag == "text":
            text = child.text
        elif child.tag == "ENTITY":
            annotations.append({"start":
     child.attrib["start"], "end": child
    .attrib["end"], "features": {"Lemma"
    : child.text}})
    return {"response": {"type": "
    annotations", "annotations": {"Place
    ": annotations}, "text": text}}
```

Listing 7: Example Python method generated by GPT-3 to convert a semantic schema into another. Only the docstring (text in green) is provided and the Python code is generated by the model. The generated Python method is valid Python code but does not correctly convert the XML string to the JSON one.

tion of pipeline (different from the `Pipeline` concept presented in Section 2.5). There are 17 task-specific pipelines and each of these pipelines uses its own semantic schema and message formats. For each pipeline, it is possible to use a multitude of services (called models in the HuggingFace ecosystem) as long as the service is compatible with the semantic scheme of the pipeline. A similar mechanism exists for Spacy which allows the use of different models as long as they respect the conventions imposed by the platform. All these platforms use a semantic approach similar to ELG (Section 2) and each platform has its own common API. However, all these APIs use different semantic schemas and there is no joint standard semantic schema used by these NLP platforms.

5.2 Semantic Interoperability Strategies

We only found a few works on the interoperability of NLP services. Rizzo and Troncy (2012) created a framework which unifies ten NER and disambiguation extraction tools by creating a common ontology. The different semantic schemas are manually mapped to the common ontology. Eckart de Castilho et al. (2019) combine three text annotation repositories (PubAnnotation, LAPPS Grid, and INCEpTION) in order to create one unique corpus. They show the challenges of the interoperability of different annotation types.

We can also look at the interoperability of web services in general. Nagarajan et al. (2006) and Nagarajan et al. (2007), and Sheth et al. (2008) provide three similar data interoperability strategies. Nagarajan et al. (2007) explain the different types of heterogeneities that exist in web services. The syntactic and semantic heterogeneities described in the paper correspond to a difference in the semantic schemas, and the structural and model heterogeneities correspond to the format difference. The authors propose to use a pre-defined mapping to overcome these issues. Nagarajan et al. (2006) and Sheth et al. (2008) propose two other methods, using WSDL-S[22] or XSLT[23] respectively to map the input and output to a common ontology and vice versa. These techniques are specific to web services but show that rule-based mapping is generally used to perform semantic interoperability.

We did not find any related work on automatic semantic mapping solutions.

6 Conclusions and Future Work

We propose an approach to achieve semantic interoperability in NLP services in the most generic way possible. To reach this goal, three scenarios with distinct interoperability needs are described, for which different interoperability solutions have been created.

- For scenario 1, a single format has been defined for all the services, so that they all use the same format and are directly interconnectable.

[22] https://www.w3.org/Submission/WSDL-S/

[23] https://en.wikipedia.org/wiki/XSLT

- In scenario 2, the NLP services use different formats, so a manual mapping between formats has been carried out. This mapping has been done manually because the number of services is limited. This solution does not scale.

- Scenario 3 addresses the problem of semantic mapping, that is, services use different semantic schemas to represent information. Our first solution is to also use manually defined rules, but as mentioned above, this is not scalable or generalisable. Therefore, in this scenario we introduce a novel approach: the automation of the mapping process, i. e., automatic semantic mapping. This method aims to achieve the automatic generation of mapping or conversion rules between different semantic formats without human intervention.

The first experiments with a language model (GPT-3) are promising, but also show their limitations. In terms of future work, we will focus mainly on the development of methods that allow us to successfully implement this automatic mapping.

7 Acknowledgements

This work has received funding from the German Federal Ministry of Education and Research through the project QURATOR (no. 03WKDA1A), from the German Federal Ministry for Economic Affairs and Energy through the project SPEAKER (no. 01MK19011) and from the EU's Horizon 2020 research and innovation programme through the project European Language Grid (no. 825627).

8 Bibliographical References

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

Bourgonje, P., Schneider, J. M., Rehm, G., and Sasaki, F. (2016). Processing Document Collections to Automatically Extract Linked Data: Semantic Storytelling Technologies for Smart Curation Workflows. In Aldo Gangemi et al., editors, *Proc. of the 2nd Int. Workshop on NLG and the Sem. Web (WebNLG 2016)*, pages 13–16, Edinburgh. ACL.

Bourgonje, P., Schneider, J. M., and Rehm, G. (2017). Domain-specific Entity Spotting: Curation Technologies for Digital Humanities and Text Analytics. In Nils Reiter et al., editors, *CUTE Workshop 2017 – CRETA Unshared Task zu Entitätenreferenzen. Workshop bei DHd2017*, Berne, CH.

Eckart de Castilho, R., Ide, N., Kim, J.-D., Klie, J.-C., and Suderman, K. (2019). Towards cross-platform interoperability for machine-assisted text annotation. volume 17, page e19.

Giretti, A., (2022). *Understanding the gRPC Specification*, pages 85–102. Apress, Berkeley, CA.

Hellmann, S., Lehmann, J., Auer, S., and Brümmer, M. (2013). Integrating NLP using Linked Data. In *12th Int. Semantic Web Conf.*, Sydney, Australia.

Moreno-Schneider, J. and Rehm, G. (2018). Towards a Workflow Manager for Curation Technologies in the Legal Domain. In Georg Rehm, et al., editors, *Proceedings of the LREC 2018 Workshop on Language Resources and Technologies for the Legal Knowledge Graph*, pages 30–35, Miyazaki, Japan.

Moreno-Schneider, J., Bourgonje, P., Kintzel, F., and Rehm, G. (2020). A Workflow Manager for Complex NLP and Content Curation Pipelines. In Georg Rehm, et al., editors, *Proc. of the 1st Int. Workshop on Language Technology Platforms (IWLTP 2020)*, pages 73–80, Marseille, France. 16 May 2020.

Moreno-Schneider, J., Rehm, G., Montiel-Ponsoda, E., Rodríguez-Doncel, V., Martín-Chozas, P., Navas-Loro, M., Kaltenböck, M., Revenko, A., Karampatakis, S., Sageder, C., Gracia, J., Maganza, F., Kernerman, I., Lonke, D., Lagzdins, A., Gil, J. B., Verhoeven, P., Diaz, E. G., and Ballesteros, P. B. (2021). Lynx: A Knowledge-based AI Service Platform for Content Processing, Enrichment and Analysis for the Legal Domain. *Information Systems*, page 101966. Special Issue on Managing, Mining and Learning in the Legal Data Domain.

Nagarajan, M., Verma, K., Sheth, A. P., Miller, J., and Lathem, J. (2006). Semantic interoperability of web services-challenges and experiences. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 373–382. IEEE.

Nagarajan, M., Verma, K., Sheth, A., and Miller, J. (2007). Ontology driven data mediation in web services. volume 4, pages 104–126.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A python natural language processing toolkit for many human languages. In *Proc. of the 58th Annual Meeting of the ACL: System Demonstrations*, pages 101–108, Online. ACL.

Rehm, G., Moreno-Schneider, J., Gracia, J., Revenko, A., Mireles, V., Khvalchik, M., Kernerman, I., Lagzdins, A., Pinnis, M., Vasilevskis, A., Leitner, E., Milde, J., and enhorn, P. W. (2019). Developing and Orchestrating a Portfolio of Natural Legal Language Processing and Document Curation Services. In Nikolaos Aletras, et al., editors, *Proc. of Workshop on Natural Legal Language Processing (NLLP 2019)*, pages 55–66, Minneapolis, USA. Co-located with NAACL 2019. 7 June 2019.

Rehm, G., Berger, M., Elsholz, E., Hegele, S., Kintzel, F., Marheinecke, K., Piperidis, S., Deligiannis, M., Galanis, D., Gkirtzou, K., Labropoulou, P., Bontcheva, K., Jones, D., Roberts, I., Hajic, J., Hamrlová, J., Kačena, L., Choukri, K., Arranz, V., Vasiļjevs, A., Anvari, O., Lagzdiņš, A., Meļņika, J., Backfried, G., Dikici, E., Janosik, M., Prinz, K., Prinz, C., Stampler, S., Thomas-Aniola, D., Pérez, J. M. G., Silva, A. G., Berrío, C., Germann, U., Re-

nals, S., and Klejch, O. (2020a). European Language Grid: An Overview. In Nicoletta Calzolari, et al., editors, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, pages 3359–3373, Marseille, France. ELRA.

Rehm, G., Galanis, D., Labropoulou, P., Piperidis, S., Welß, M., Usbeck, R., Köhler, J., Deligiannis, M., Gkirtzou, K., Fischer, J., Chiarcos, C., Feldhus, N., Moreno-Schneider, J., Kintzel, F., Montiel, E., Doncel, V. R., McCrae, J. P., Laqua, D., Theile, I. P., Dittmar, C., Bontcheva, K., Roberts, I., Vasiljevs, A., and Lagzdiņš, A. (2020b). Towards an Interoperable Ecosystem of AI and LT Platforms: A Roadmap for the Implementation of Different Levels of Interoperability. In Georg Rehm, et al., editors, *Proc. of the 1st Int. Workshop on Language Technology Platforms (IWLTP 2020)*, pages 96–107, Marseille, France. 16 May 2020.

Georg Rehm, editor. (2022). *European Language Grid: A Language Technology Platform for Multilingual Europe*. Cognitive Technologies. Springer. Forthcoming.

Richardson, L., Amundsen, M., and Ruby, S. (2013). *RESTful Web APIs*. O'Reilly Media, Inc.

Rizzo, G. and Troncy, R. (2012). NERD: A framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76, Avignon, France. Association for Computational Linguistics.

Sheth, A. P., Gomadam, K., and Ranabahu, A. H. (2008). Semantics enhanced services: Meteor-s, sawsdl and sa-rest. *Bulletin of the Technical Committee on Data Engineering*, 31(3):8.

Varda, K. (2008). Protocol buffers: Google's data interchange format. Technical report, Google.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proc. of EMNLP 2020: System Demonstrations*, pages 38–45, Online. ACL.

Appendix

A ELG Python SDK: More Examples

```
from elg import Pipeline

pipeline = Pipeline.from_ids([9395,
    9385])
results = pipeline(request_input="
    audio.mp3", request_types=["audio
    ", "text"])
print(results[-1]) # Sentiment
    Analysis response: {type='
    annotations' features={'OVERALL':
     '71.3'}}
```

Listing 8: Combining ELG services using the Python SDK and Pipeline class. We combine an ASR service and an English sentiment analysis service

```
from elg import Pipeline

pipeline = Pipeline.from_ids([9212,
    18092])
results = pipeline(
    LONG_ENGLISH_ARTICLE)
print(results[-1]) # summary of the
    English article in Ukrainian
```

Listing 9: Combining ELG services using the Python SDK and Pipeline class. We combine an English summariser and a Ukrainian to English MT service

B Parameters (GPT-3 Experiments)

```
engine="text-davinci-002",
prompt="\"\"\"\nPython method that
    converts\n[{\"translation_text\":
     \"This is a plain text sentence
    .\"}] \nto \n{\"type\":\"text\",
    \"content\":\"This is a plain
    text sentence.\"}\n\"\"\"\n\n",
temperature=0,
max_tokens=105,
top_p=0,
frequency_penalty=0,
presence_penalty=0
```

Listing 10: Parameters used in the OpenAI playground for Listing 6

```
engine="text-davinci-002",
prompt="\"\"\"\nPython method that
    converts\n<?xml version=\"1.0\"
    encoding=\"utf-8\"?><text>I was
    living in <ENTITY class=\"
    LOCATION\">Berlin</ENTITY> last
    year, but then I moved to <ENTITY
     class=\"LOCATION\">New York</
    ENTITY>.</text>\nto \n{\"response
    \":{\"type\":\"annotations\",\"
    annotations\":{\"Place\":[{\"
    start\":16,\"end\":22,\"features
    \":{\"Lemma\":\"Berlin\"}},{\"
    start\":54,\"end\":62,\"features
    \":{\"Lemma\":\"New York\"}}]}}}\
    n\"\"\"\n\n",
temperature=0,
max_tokens=150,
top_p=0,
frequency_penalty=0,
presence_penalty=0
```

Listing 11: Parameters used in the OpenAI playground for Listing 7