D3.3 Language Resource Transformation Software

Author(s): Christian Fäth, Maxim Ionov, Christian Chiarcos

Date: 30.09.2021

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825182. The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Union.

H2020-ICT-29b

Grant Agreement No. 825182

Prêt-à-LLOD - Ready-to-use Multilingual Linked Language Data for Knowledge Services across Sectors

D3.3 Language Resource transformation Software

Deliverable Number: D3.3
Dissemination Level: Public
Delivery Date: 2021-09-30
Version: 1.0
Author(s): Christian Fäth, Maxim Ionov, Christian Chiarcos

Document History

Version Date	Changes	Authors
2021-09-10	**Initial document**	**Christian Fäth**
2021-09-20	**Draft ready**	**Christian Fäth** **Max Ionov**
2021-09-24	**Reviewed by UPM**	**Patricia Martín Chozas** **Victor Rodriguez Doncel**
2021-09-30	**Final document**	**Christian Fäth** **Max Ionov**

Table of Contents

D3.3 Language Resource Transformation Software

Within the Prêt-à-LLOD project, five major challenges when working with linguistic resources are addressed (cf. Fig. 1):

- Discovery of resources
- Data management and licensing
- Transformation of heterogeneous resources
- Interlinking resources
- Embedding resources and algorithms into complex workflows

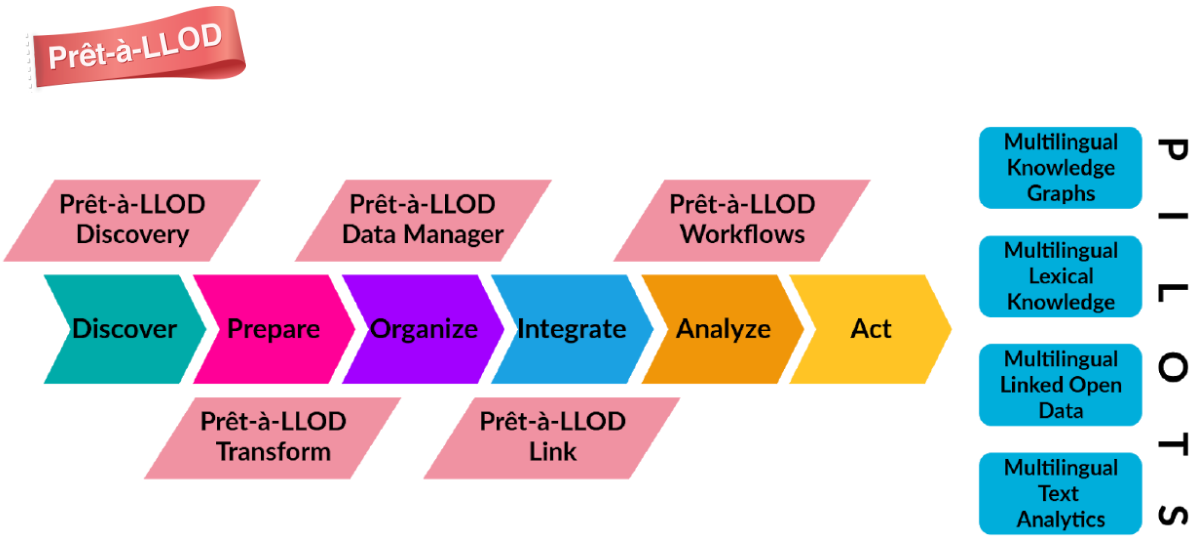

Figure 1: Prêt-à-LLOD data value chain

This short report accompanies the Prêt-à-LLOD software deliverable D3.3[1] "Resource Transformation Software". It is meant to provide a quick overview of the motivation, software architecture and its basic functionalities. It also serves as a pointer to the repositories, where both the code and more detailed user guidelines are available.

1. Motivation

Resource heterogeneity is one of the most time-consuming challenges when working with linguistic data, given the sheer wealth of different data formats and annotation schemes mixed with an ever growing set of NLP tools accepting and producing very specific input and output. More and more powerful software relying on Deep Learning requires ever larger sets of training data and combinations of corpora, lexical data, ontologies etc. which are only available in heterogeneously annotated resources.

[1] https://github.com/Pret-a-LLOD/Fintan

Linked Data to some extent addresses these issues of combining decentrally organized resources in that it introduces a common Resource Description Framework[2] (RDF) and allows ontological modeling to standardized data models for specific types of resources, such as OntoLex-Lemon (McCrae et al., 2012; Cimiano et al., 2016) for lexical data or NIF (Hellmann et al., 2013), CoNLL-RDF (Chiarcos and Fäth, 2017) and POWLA[3] (Chiarcos, 2012) for corpora. On the scope of annotation schemes, efforts of standardization have been made in the form of GOLD[4], ISOcat[5] or OLiA[6] (Chiarcos and Sukhareva, 2015). Yet, neither are all resources available as linked data, nor do all of them adhere to the same data models and annotation schemes. There can also be variations of how data models are being adapted to specific use cases.

Existing tools and conversion approaches, such as CSV2RDF (Tandy et al., 2015) or R2RML (Das et al., 2012) reflect the original data storage paradigm in RDF. Recent developments of Morph[7] achieve higher performance and more consistent access by employing relational constraints in concurrent query execution across heterogeneous databases (Chaves-Fraga et al., 2021). However, data integration into virtual knowledge graphs or generic RDF conversions alone do not necessarily produce interoperable linguistic data models and thus may require post-processing steps. This can also result in scalability issues since maintaining, querying and transforming larger RDF resources on triples stores may require powerful servers and high amounts of memory. Many open source implementations also lack parallelization which may result in extended execution time. For all these reasons, the general tendency is to tailor each converter specifically towards the source and target format, resulting in higher accuracy and performance but lower reusability.

With Fintan, the Flexible INtegrated Transformation and Annotation eNgineering platform (Fäth et al., 2020), we tackle many of these issues. Instead of a fully integrated converter suite which would either be limited to very specific source and target formats or could only produce fairly generic RDF output, we decided to develop a framework which would allow a user to easily integrate existing converters, ontologies and knowledge graphs into complex transformation workflows with parallel stream-based graph transformation[8] steps.

---

[2] https://www.w3.org/2001/sw/wiki/RDF

[3] POWLA has recently become one of the most relevant formats for representing corpora with more complex semantic or syntactic structures. (Cimiano et al., 2020)

[4] General Ontology of Linguistic Description (Farrar, S. and Langendoen 2003)

[5] A terminology repository which provided human-readable and XML-encoded information about linguistic data categories and language resource metadata via persistent URIs. Currently it is no longer maintained (Kemps-Sniders et al. 2008)

[6] The Ontologies of Linguistic Annotation (OLiA) are a set of interoperable RDF/OWL models for various annotation schemes which are all linked to a central reference model, which enables a user to transform annotations between schemes.

[7] Morph is a suite of open source tools for generating Virtual Knowledge Graphs over heterogeneous data sources https://morph.oeg.fi.upm.es/

[8] as originally introduced with CoNLL-RDF

2. Fintan - Flexible INtegrated Transformation and Annotation eNgineering

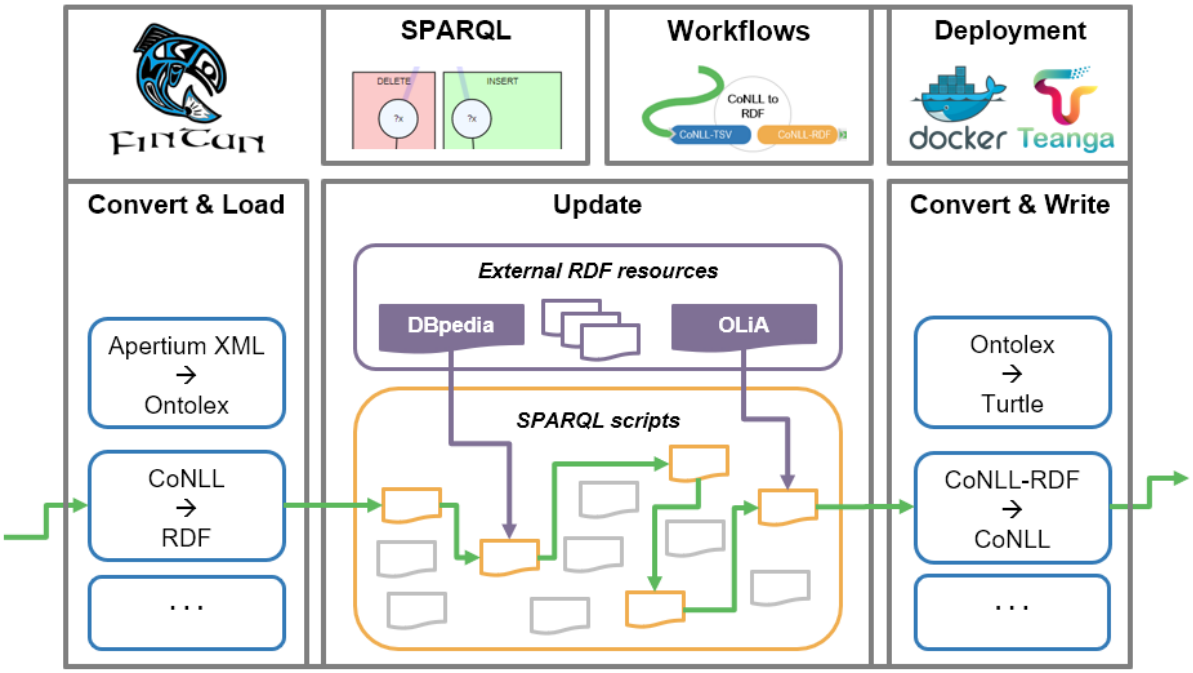

Figure 2: Fintan platform

The Fintan platform is an effort of combining existing converter frameworks with stream-based graph transformation and a workflow management engine in order to create integrated transformation pipelines for various input and output formats. By making data conversion modular, we increase the reusability of granular transformation steps. By choosing a stream-based approach, specifically for processing RDF data, we also address scalability issues typically arising with large scale datasets on triple stores. The Fintan platform, as shown in Fig. 2, encompasses:

- An interoperable pool of processing components including:
  - External converter tools
  - Stream-based graph processing for RDF
  - Serializer tools and data writers
- A development environment for SPARQL[9] updates and transformation workflows
- A means of deploying specific converter pipelines as integrated Docker[10] containers

In this section we will first describe the general design principles of Fintan and how the software operates as a whole. We then discuss the software architecture and functionalities and how they reflect the design principles.

[9] https://www.w3.org/TR/sparql11-query/
[10] http://www.docker.com/

2.1 General design principles

One major design principle of Fintan is **granularity**. This applies not only to breaking down individual processing steps but also to the data itself.

In Fintan, every means of processing data is wrapped into a specific **component**. Each component can perform various acts of data transformation, which may vary depending on the specific use case. Therefore, each component can be instanced with a specific configuration which may consist of parameters or full transformation scripts. These instances also define the requirements of how data must be modeled in order to be processable. Examples of such components include:

- A CSV transformer based on Tarql[11] accepts modified SPARQL queries to directly generate RDF from tabular source formats.
- An XSLT transformer based on the Saxon HE library[12] accepts XSL scripts to transform XML input data to RDF or other formats. The type of output is purely determined by the XSL script.
- An RDF updater can transform or combine existing RDF resources, e.g. transform the annotation schemes in a corpus or dictionary by side-loading OLiA and applying SPARQL updates (see also Fig. 2).

This granularity of processing steps adds to another design principle: **reusability**. Since components are very generic, they can be used for a multitude of input and output formats. In addition, their specific instances are defined by transformation scripts which could also be applied to other types of converters: if both, an instance of the XSL transformer, as well as an instance of the CSV transformer could convert a dictionary to OntoLex-Lemon, a subsequent change of annotation models could be performed by the same instance of an RDF updater. Furthermore, dockerized converter pipelines can be deployed to the Teanga[13] platform, thus enabling RDF-based NLP modules to directly feed on generic resource types, further increasing scope and applicability for long-term use by a wider audience.

In addition, this granularity also improves **extensibility**. Fintan not only allows to customize instances of components by using scripts and configurations, but features a generic Core API which allows to write custom components (e.g. by wrapping existing converters) and directly insert them into workflows, if they follow the general design principles. Within the Prêt-à-LLOD project, the Fintan platform allows all project partners to contribute their existing converters as modules or to make adjustments to pipelines on their own.

[11] http://tarql.github.io/
[12] http://saxon.sourceforge.net/
[13] Teanga is a workflow management software for integrated, dockerized NLP and Linked Data services developed in Task 3.3 of the Prêt-à-LLOD project initially presented by Ziad et al. (2018): https://github.com/Pret-a-LLOD/teanga

In this deliverable we include sample pipelines e.g. for converting Universal Dependencies[14] to CoNLL-RDF and Apertium (Forcada et al. 2011) dictionaries to OntoLex-Lemon. We also include steps for transforming linguistic annotations relying on OLiA. Yet, Fintan is vocabulary independent and not limited regarding the input and output formats it can consume or produce. Instead, it provides a way to consistently model, integrate and recombine steps of data conversion.

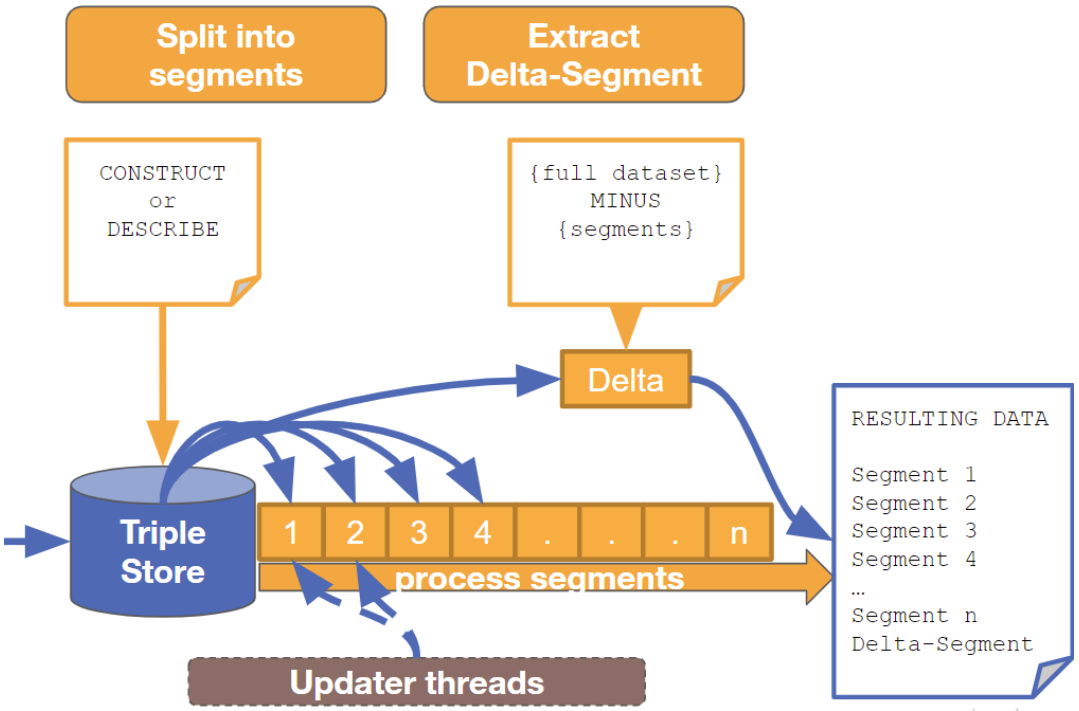

Figure 3: Splitting and processing unsegmented RDF data

Another form of **granularity** lies within the data itself. As described above, processing large RDF resources in triple stores can require powerful servers, large amounts of system memory or even proprietary implementations optimized for parallelized access. In Fintan, we make use of an inherent property of most linguistic resources: they can be divided into small, mostly self-contained segments. In corpora, these may be sentences, tokens or spans, while in Lexica, these may be lexicon entries or translation sets. Fintan therefore provides the following functionalities (cf. Fig. 3):

- **loading and splitting** streams of serialized RDF data into digestible segments
- **stream-processing** the resulting segments in **parallel** (`RDFUpdater` module)
- **writing** them back in typical RDF serializations or exporting to other formats

With this approach we can also tackle the aforementioned **scalability** issues, to a certain extent, by allowing to stream (reducing memory overhead) and parallelize (improving processing performance) wherever the data can be segmented.

[14] https://universaldependencies.org/

2.2 Architecture and Implementation

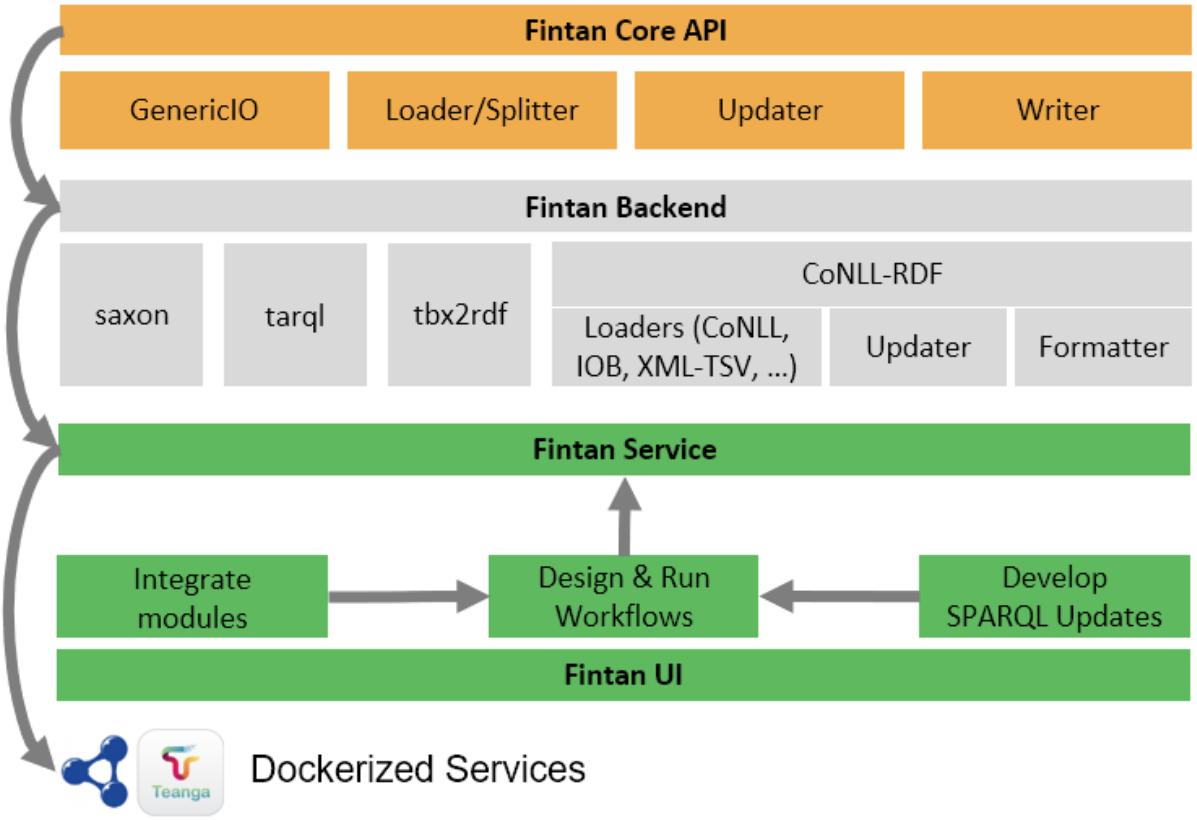

Figure 4: Modular architecture

Bearing in mind these design principles, we designed a modular architecture which aids a decentralized development process. With Fintan inheriting some of the core functionalities from CoNLL-RDF, we decided to keep the general design paradigms intact and stay within the Java-based environment including the Apache Jena API[15] for graph transformation. However, since CoNLL-RDF is designed as a self-contained tool which is focused on TSV-based input formats, Fintan establishes an additional abstraction layer.

The Fintan architecture comprises four interdependent layers which are addressed in the subsequent sections (cf. Fig. 4). All layers, as well as a pointer to the full technical documentation[16] are available in the Prêt-à-LLOD repository[17].

2.2.1 Fintan Core API[18]

The **Fintan Core API** is designed to provide the minimal abstraction layer and functional core classes for using stream-based graph processing and running Fintan-compliant

[15] http://jena.apache.org/
[16] https://github.com/acoli-repo/fintan-doc
[17] https://github.com/Pret-a-LLOD/Fintan
[18] https://github.com/acoli-repo/fintan-core

pipelines. Since it has a minimal amount of dependencies it can easily be built and imported as a Maven[19] dependency by independent projects.

We define `FintanStreamComponent` as the primary class to host a data transformation component. Each component can be run as an independent thread and allows for an indefinite amount of input and output stream slots which are designed to correspond to a specific graph for data processing. Like with RDF graphs, there is a **default stream slot** and multiple **named stream slots**, which should follow a valid URI schema. Hereby Fintan distinguishes between two types of streams:

- **Generic streams**: this can be any kind of stream supplied by an input file. Typically, this will be a machine-readable, formal representation of data, such as XML, CSV formats or RDF serializations (RDF/XML, Turtle etc.) Depending on their internal order and structure, they could already be considered segmented, as long as they can be easily split by a delimiter, such as an empty line between segments.
- **Segmented RDF**: is a stream of pre-loaded RDF models. Each model should correspond to a self-contained segment of the underlying data which can be processed independently.

Depending on the types of streams they accept as input or output, Fintan distinguishes four core classes corresponding to types of operation for components:

- **`GenericIO`** components read and write generic streams of data. They function as a baseline abstraction layer which can incorporate almost any existing transformation tool.
- **`Loader`** components read any kind of input data and write back a segmented stream of RDF data for stream-based graph transformation. In the Core API, there is a default Loader which reads RDF serializations which are already segmented by a configurable segment delimiter. For completely unstructured RDF serializations, an additional **`Splitter`** component can be used to construct a segmentation.
- **`Updater`** components read streams of segmented RDF data, process multiple data segments in parallel and then write back the transformed RDF segments. They are defined by a set of SPARQL scripts and additional resources to be permanently side-loaded for specific processing steps, e.g. inferring annotation schemes by using OLiA or entity linking to DBpedia[20] (Auer et al., 2007) entries .
- **`Writer`** components read segmented RDF data and write back generic output, e.g. RDF serializations (e.g. Turtle[21], RDF/XML) or export the data to other output formats.

Finally, the **`FintanManager`** class provides methods for constructing pipelines from JSON configuration files, as well as a basic CLI. The Core API is designed to host and run any

[19] Apache Maven is a tool for software project management. Software modules can be deployed to a central repository by independent developers and imported into a project by a dependency structure. https://maven.apache.org/
[20] The RDF representation of Wikipedia: https://wiki.dbpedia.org/
[21] https://www.w3.org/TR/turtle/

component extending one of the four core classes. Importing the Core API to another project and creating a custom component as a subclass, while adhering to the general design principles, will immediately allow it to be integrated into Fintan pipelines.

2.2.2 Fintan backend[22]

While the aforementioned components are central functionalities provided by the Fintan Core API, many of the `Loader` and `Writer` components can be external programs published in other repositories. In order for them to become fully Fintan-compliant, they must be subclasses of one of the four core classes and make the source code available as a Maven artifact.

By using Maven dependencies, they can be directly imported into Fintan as **External API modules**. On the other hand, Fintan can be directly used within their respective toolsets. The primary example for this is CoNLL-RDF, which is both the spiritual predecessor of Fintan as well as the primary set of components for handling CoNLL and related corpus formats (e.g. Chiarcos and Glaser, 2020) within Fintan. There also is an additional component for transforming XML data using XSLT (based on the Saxon HE library) as well as a Tarql-based component for processing many kinds of CSV data. In addition, we are currently working on an integration of TBX2RDF[23] (Cimiano et al., 2015) to support terminological data and to facilitate the application of Fintan in Terme-à-LLOD (Buono et al., 2020).

The **Fintan backend repository** serves as a wrapper for compliant API modules and contains a script which checks out and builds all verified dependencies or submodules. It can be run as a stand-alone CLI tool.

All External API modules available in the Fintan backend can also be built into workflows using the **Fintan Service** and **Fintan UI**. Using these modules, it is possible to integrate Fintan in larger workflows alongside closed-source services or architectures which are incompatible with the Fintan Java API. An example of the latter would be using Fintan workflows encapsulated in Docker containers as workflow components in the Teanga framework. This scenario and, in general, Fintan Service and Fintan UI, are described in the next sections.

2.2.3 Fintan Service[24]

In order to add alternative methods of interacting with Fintan and facilitate its ease of integration into larger workflows, we have implemented the **Fintan Service**, an OpenAPI-compatible[25] web service that provides a REST API for executing Fintan pipelines. To simplify the deployment process, we provide Docker containers encapsulating the service. There are two flavours in which this service is distributed:

[22] https://github.com/acoli-repo/fintan-backend
[23] https://github.com/cimiano/tbx2rdf
[24] https://github.com/acoli-repo/fintan-service
[25] https://spec.openapis.org/oas/latest.html

1. A container that provides an API capable of running any pipeline and provides endpoints for uploading new pipelines and data required for running them.
2. Containers, tailored for each particular task. In this scenario, the API in the container provides the only endpoint, for running the pipeline for which it was built. This scenario is designed for the ease of use in larger workflows, such as those built with Teanga.

Both scenarios provide possibilities for running pipelines in both synchronous and asynchronous regimes. More specifically, the Fintan Service provides a user with APIs with the following endpoints:

- **`/api/run`** executes a pipeline and either returns the result (synchronous mode) or a link to the tasks queue (asynchronous mode)
- **`/api/upload`** provides an endpoint that allows the user to upload additional pipelines and data required for the pipelines to run;
- **`/api/docs`** employs Swagger UI[26] for visualising OpenAPI specification of the API.

2.2.4 Fintan UI[27]

In order to make the process of creating transformation pipelines and corresponding docker containers accessible to a wider audience, we have implemented a way to create these pipelines and containers visually. **Fintan UI** is a web application designed to be run locally. It allows creating complex pipelines in a simple drag-and-drop fashion. Despite this apparent simplicity, it requires knowledge of Fintan architecture since it is up to a user to set up all the properties for each component and write corresponding SPARQL queries for transformations and conversions.

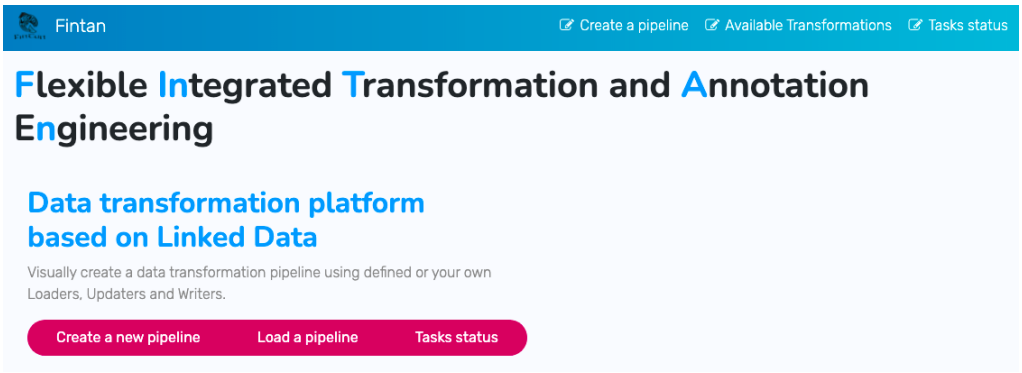

Figure 5: Fintan UI Title Page

The UI consists of 3 main parts:

- Drag-and-drop pipeline creator;
- A list of existing pipelines exported and running on a Fintan Service;
- Task progress for pipelines started asynchronously.

[26] https://swagger.io/tools/swagger-ui/
[27] https://github.com/acoli-repo/fintan-ui

A page for creating pipelines contains three groups of elements a user can put as a part of their pipeline:

- **Data import and export**: this is a set of file upload and save operations that are happening outside of Fintan and the pipeline for Fintan just contains paths to where they were uploaded and paths where to save them.
- **Data transformations**: these elements roughly correspond to Fintan classes and perform data conversion and transformation steps.
- **Resources**: these are additional resources that are used by components of the second group. Examples of this group are: SPARQL queries for `RDFUpdater` or `RDFStreamSplitterTDB`, external ontologies and mappings.

For each element, there is a set of options that can be set in a pop-up window which can be revealed by clicking on an element (see Fig. 6). These options correspond to the ones described in Fintan documentation as parameters for each class, with a notable exception: SPARQL queries are not set as options but instead are represented as resource elements connected to corresponding transformation elements (see Fig. 7).
SPARQL queries can be written directly in the UI in the option window of a corresponding resource with YASQE, a text editor with syntax highlighting and autocomplete for SPARQL.[28]

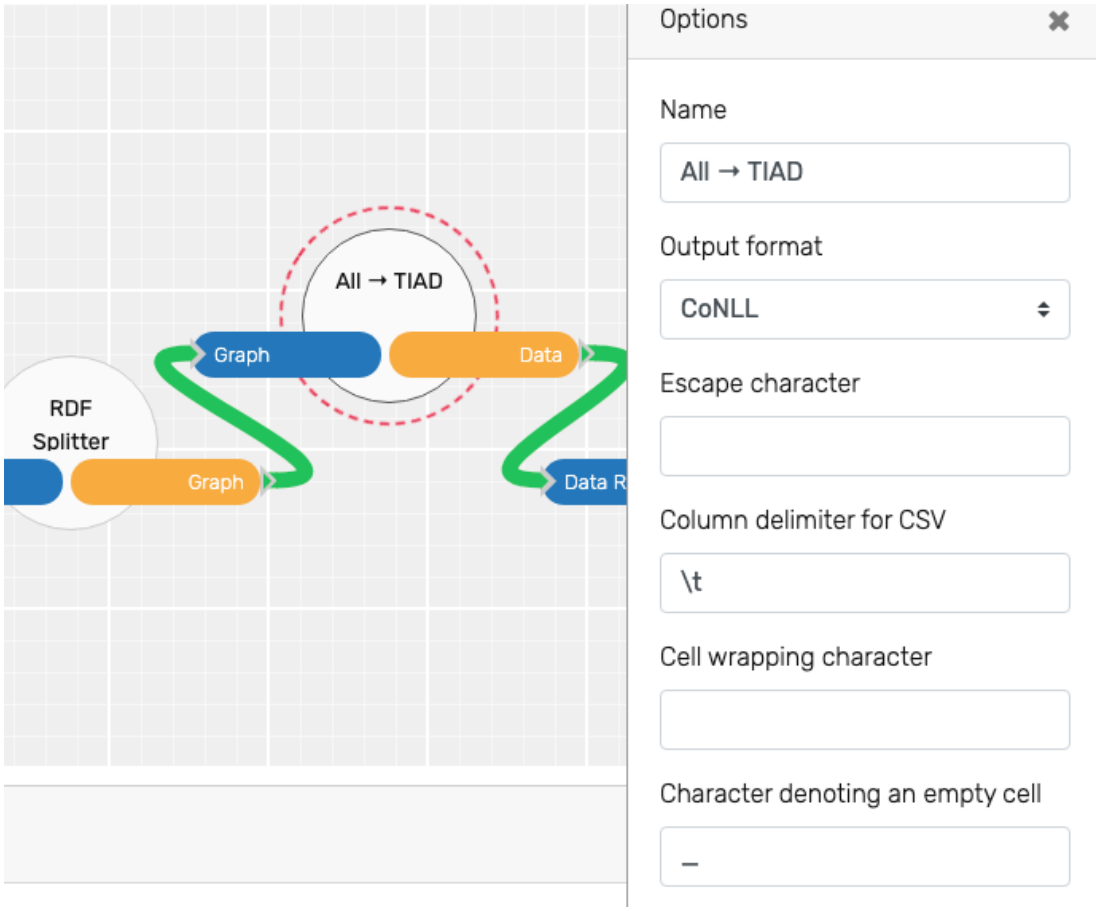

Figure 6: Options for an element

[28] https://triply.cc/docs/yasgui

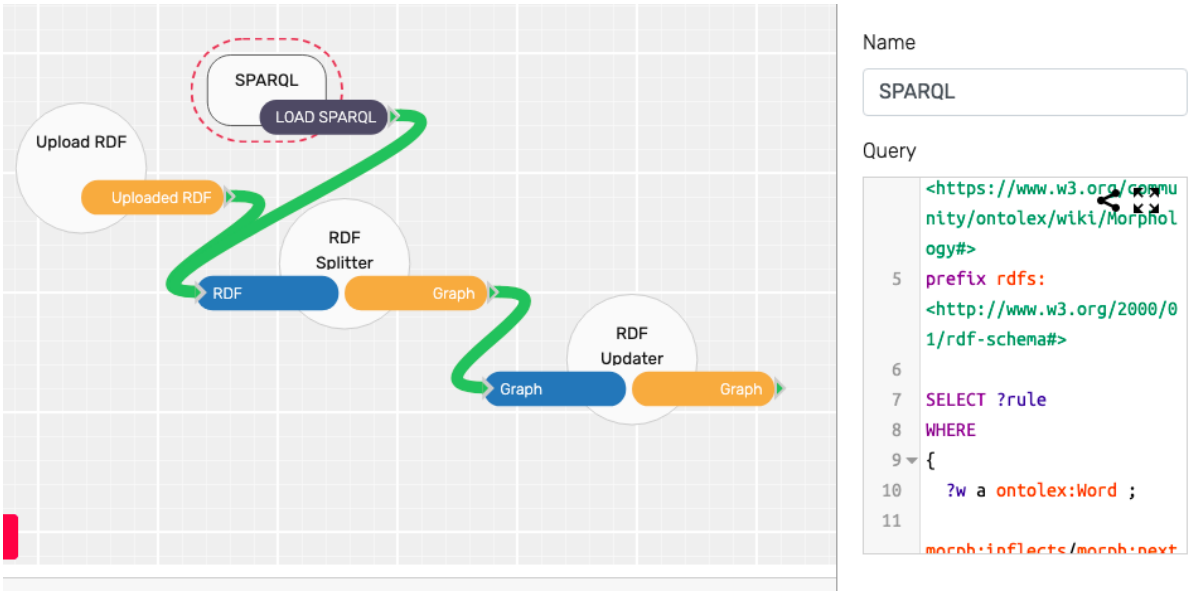

Figure 7: Options for an element

After the pipeline is created, a transformation workflow can be created by clicking "**Generate**". Here a user can choose whether they want to …

- download everything they need to build a Docker container with this pipeline running,
- run the pipeline on Fintan service running locally,
- or just download the JSON for the pipeline.

The list of all transformation workflows available on the Fintan service can be accessed on the page "**Available transformations**". Tasks that were started asynchronously can be viewed on the third page, "**Tasks status**".

In addition to building workflows in Fintan UI, the development of graph transformation steps can be aided by **SparqViz**, a tool for creating visualizations of SPARQL queries and updates using GraphViz and the underlying dot format. A REST API accepts a SPARQL query and outputs a dot file and an SVG image. An exemplary lightweight editor website is included with the latest stand-alone version.[29]

2.3 Example Pipeline

Fintan features a lot of configuration options and modes of operation. A detailed description of each of the processing components and example pipelines can be found in the Fintan repositories. Therefore, in this report we will not repeat the technical documentation but instead showcase the core functionalities with one reasonably complex example pipeline.

In the last report D3.2 and on the International Semantic Web Conference (Gracia et al., 2020), we already presented a pipeline for converting Apertium dictionaries to

[29] https://github.com/acoli-repo/sparqviz/

OntoLex-Lemon to be used in the scope of WP4 as a part of Semalytix' Pharos® software which provides international customers from the pharmaceutical industry with actionable real-world evidence (RWE). In addition, a TSV representation of the translation sets is used for the Translation Inference Across Dictionaries (TIAD, Gracia et al., 2019) shared task. The conversion pipeline was partially implemented with Fintan technology but still relied on external processing steps.

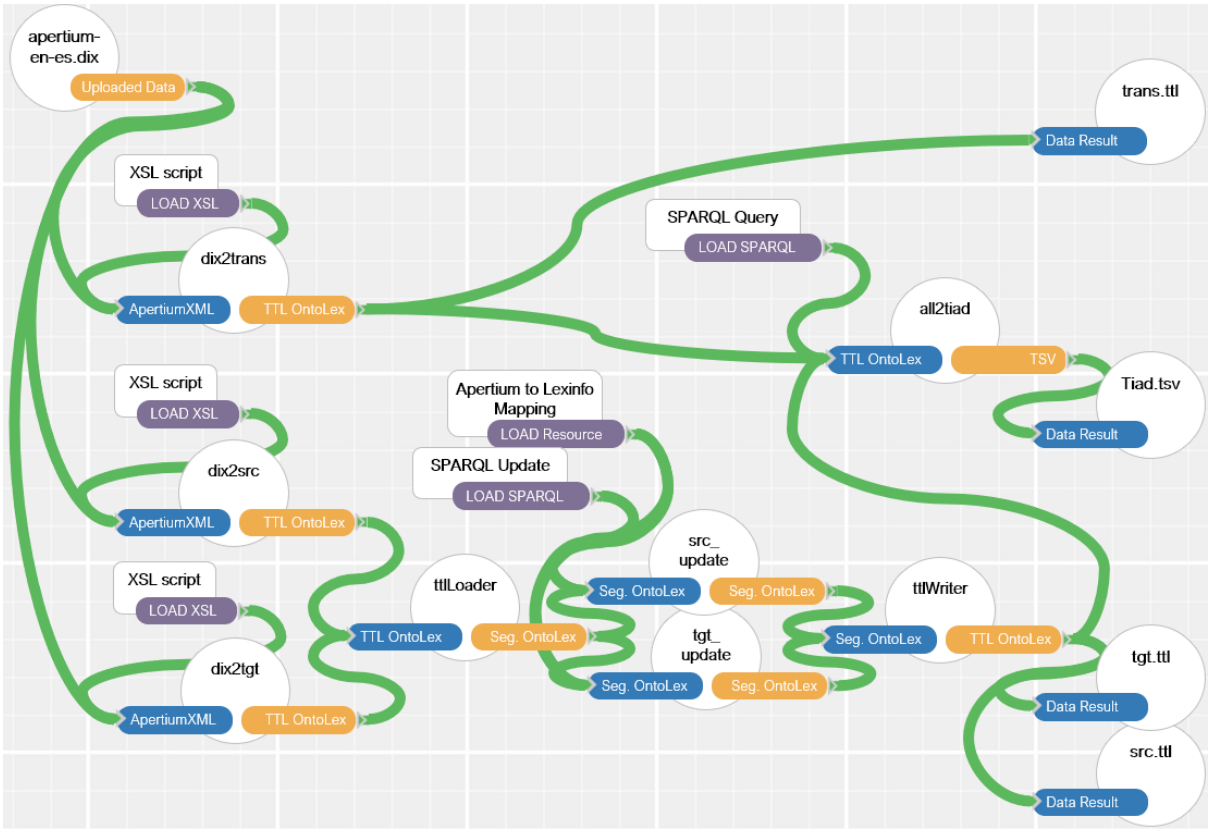

Figure 8: Apertium conversion workflow in Fintan

With the full version of Fintan, it is now possible to recreate the entire converter pipeline to run within Fintan and export it as an integrated Docker container (cf. Fig. 8). The pipeline itself consists of the following processing steps:

- Upload the original Apertium XML data and supply it to three separate instances of the `XSLTStreamTransformer` component, each defined by a distinct XSL script:
  - `dix2src` extracts a `lime:Lexicon` containing all `LexicalEntries` of the source language
  - `dix2tgt` extracts a `lime:Lexicon` containing all `LexicalEntries` of the target language
  - `dix2trans` extracts a `vartrans:TranslationSet` containing all Translations between source and target language
- The lexicons of both the source and the target language contain linguistic annotations which are proprietary to Apertium data. In order to create interoperable RDF data adhering to OntoLex-Lemon guidelines, we transform these annotations to Lexinfo (Cimiano et al., 2011) using stream-based graph processing:

    - In a first step, we employ a `ttlLoader`, an instance of the `RDFStreamLoader` component, to create a segmented RDF stream with each segment corresponding to a `LexicalEntry`. Since the output of the XSL transformation creates a Turtle serialization which separates lexical entries by empty lines, we do not need to employ complex splitting operations.
    - `src_update` and `tgt_update` are instances of the `RDFUpdater` component which side-load the Apertium to Lexinfo mapping table created and maintained by UNIZAR[30]. They consume the respective segmented RDF streams and apply a SPARQL update transforming the annotations of multiple segments in parallel.
    - Finally, the `ttlWriter`, an instance of the `RDFStreamWriter` component, writes back a Turtle serialization of the resulting transformed datasets.
- At this point, the OntoLex-Lemon conversion is finished, and the three resulting files `src.ttl`, `tgt.ttl` and `trans.ttl` can be stored to disk.
- However, all three result files are required to create a combined TSV output for the TIAD shared task. Therefore the streams are forked into an instance `all2tiad` of the `SparqlStreamTransformerTDB` component. It stores the three named input streams to graphs of the same names inside an internal, temporary database. From here an additional SPARQL script can query the combined dataset to create custom separated value exports, in this case configured to TSV.

The JSON configuration for the full pipeline is available as part of the samples section in the Fintan backend repository.[31]

3. Conclusion and Outlook

As we have shown, Fintan lays the foundation for interoperable transformation components and workflows. It is designed to host existing converter modules by implementing the Core API in a wrapper component. Fintan encourages granularity in workflow design by dividing pipelines into self-contained transformation steps and streaming specific types of data between them, increasing transparency and reusability. This is further augmented by the approach of separating actual execution from applied transformation scripts like XSL or SPARQL and rather treating them as configuration data. The stream-based transformation of segmented RDF data also allows to improve the scalability of workflows, wherever applicable. The graphical workflow manager and the means of deploying integrated pipelines as docker containers further adds to usability and interoperability, e.g. with the Teanga platform.

Yet, there is still a way to go from here. Within the scope of the Prêt-à-LLOD project, we aim to include as many existing converters and pipelines from our project partners as possible to the pool of available configurations and make them available as Docker containers. Potential examples for this include morphological data (Declerck and Racioppa, 2019) or open

[30] https://github.com/sid-unizar/apertium-lexinfo-mapping
[31] https://github.com/acoli-repo/fintan-backend/blob/master/samples/xslt/apertium/_demo-apertium-full-with-tiad.json

multilingual wordnets (Declerck and Siegel, 2019). This will allow the project to publish a suite of converters for many existing types of resources.

But even beyond the scope of the project, Fintan's granular design opens up opportunities. We have been prototyping ontological models for Fintan which allow us to address additional challenges of data conversion: The assessment of functional compatibility between processing steps in the current implementation is a mostly manual task. A semi-automated assessment would require a formal definition of input and output, at least encompassing:

- data format and serialization (XML, RDF/XML, Turtle …)
- data models (OntoLex, CoNLL-RDF …)
- annotation models (Lexinfo, UD, OLiA …)
- required properties or nodes (as a further specification of required models)
- required source and target graphs (I/O slots)
- required ontological datasets or mappings.

In addition, the optimization of processing order and detecting deadlocks would require transitive assessment of streaming properties distinguishing whether a stream is fully consumed (or supplied) as a preprocessing step, such as the side-loading of mapping tables, or if it is sequentially processed and handed through during runtime. In the implementation of the Splitter we also allow an unsegmentible delta (e.g. containing general meta information) to be supplied after all constructed segments have already been streamed out. If a subsequent component waits for such a "post-supplied" stream on a "pre-supplied" input slot, this might result in a deadlock.

Creating an ontology for these two challenges which enables users to model their modules and workflows in a consistent way and to upload them to a central repository, would allow us to go even one step further: we could aid workflow development by context sensitive proposals for compatible next processing steps. We could even go so far as to induce complex workflow configurations if a user only supplies source and target format, as we have recently showcased for a fraction of Fintan with CoNLL-Transform (Chiarcos et al., 2021).

References

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007) DBpedia: A Nucleus for a Web of Open Data. In Aberer K. et al. (eds) *The Semantic Web. ISWC 2007, ASWC 2007*. Lecture Notes in Computer Science, vol 4825, pp. 722-735. Springer, Berlin, Heidelberg

Buono, P. M., Cimiano P., Elahi, F. M., and Grimm, F. (2020). Terme-à-LLOD: Simplifying the Conversion and Hosting of Terminological Resources as Linked Data. In *Proceedings of the 7th Workshop on Linked Data in Linguistics, LDL 2020* at *LREC 2020*, pp. 2503-2510 Marseille, France.

Chaves-Fraga, D., Ruckhaus, E., Priyatna, F., Vidal, M. and Corcho, O. (2021). Enhancing Virtual Ontology Based Access over Tabular Data with Morph-CSV. *Semantic Web*., 0(0) accepted, pp. 1–34. 10.3233/SW-210432.

Chiarcos, C. (2012). POWLA: Modeling linguistic corpora in OWL/DL. In *Proceedings of the Extended Semantic Web Conference, ESWC 2012*, pp. 225–239..

Chiarcos, C. and Sukhareva, M. (2015). OLiA–ontologies of linguistic annotation. *Semantic Web*, 6(4), pp. 379–386.

Chiarcos, C., and Fäth, C. (2017). CoNLL-RDF: Linked Corpora Done in an NLP-Friendly Way. In *Language, Data, and Knowledge, LDK 2017,* pp. 74–88. Galway, Ireland.

Chiarcos, C., and Glaser, L. (2020): A Tree Extension for CoNLL-RDF. In *Proceedings of the 12th International Conference on Language Resources and Evaluation, LREC 2020*, pp. 7161–7169. Marseille, France

Chiarcos, C., Ionov, M., Glaser, L., and Fäth, C. (2021): An Ontology for CoNLL-RDF: Formal Data Structures for TSV Formats in Language Technology. In *Proceedings of the 3rd Conference on Language, Data and Knowledge, LDK 2021*. Zaragoza, Spain.

Cimiano, P., Buitelaar, P., McCrae, J., and Sintek, M. (2011). LexInfo: A declarative model for the lexicon-ontology interface. In *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 9(1), pp. 29–51.

Cimiano, P., McCrae, J., Rodriguez-Doncel, V., Gornostay, T., Gomez-Perez, A., Siemoneit, B., and Lagzdins, A. (2015). Linked Terminology: Applying Linked Data Principles to Terminological Resources. In *Proceedings of the eLex 2015 conference,* pp. 504-517. Herstmonceux Castle, United Kingdom.

Cimiano, P., McCrae, J.P., and Buitelaar, P. (2016). Lexicon Model for Ontologies: Community Report. URL https://www.w3.org/2016/05/ontolex/

Cimiano, P., Chiarcos, C., McCrae, J.P., Gracia, J. (2020). Linguistic Linked Data - Representation, Generation and Applications. Springer.

Das, S., Sundara, S., Cyganiak, R. (2012). R2RML: RDB to RDF mapping language. W3C recommendation, World Wide Web Consortium. URL https://www.w3.org/TR/csv2rdf/

Declerck, T., and Racioppa, S. (2019). Porting Multilingual Morphological Resources to OntoLex-Lemon. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2019,* pp. 233--238. Varna, Bulgaria.

Declerck, T. and Siegel, M. (2019). Porting a Crowd-Sourced German Lexical Semantics Resource to Ontolex-Lemon. In: *Proceedings of the eLex 2019 conference, Sintra, Portugal, Lexical Computing CZ s.r.o., CELGA-ILTEC 2019,* pp. 970-984. University of Coimbra, Brno.

Farrar, S. and Langendoen, D.T. (2003). A linguistic ontology for the semantic web. GLOT international 7(3), 97–100.

Fäth, C., Chiarcos, C., Ebbrecht, B., and Ionov, M. (2020): Fintan - Flexible, INtegrated Transformation and Annotation eNgineering. In *Proceedings of the 12th International Conference on Language Resources and Evaluation, LREC 2020*, pp. 7212-7221 Marseille, France.

Forcada, M. L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Sánchez-Martínez, F., Ramírez-Sánchez, G. and Tyers, F. M. (2011). Apertium: a free/open-source platform for rule-based machine translation. Machine translation, 25(2), 127-144.

Gracia, J., Kasabi, B., and Kernerman, I. (eds.) (2019): Proceedings of TIAD-2019 Shared Task – Translation Inference Across Dictionaries, co-located with the 2nd Conference on Language, Data and Knowledge, LDK 2019. Leipzig, Germany.

Gracia, J., Fäth, C., Hartung, M., Ionov, M., Bosque-Gil, J., Verissimo, S., Chiarcos, C., and Orlikowski, M. (2020). Leveraging Linguistic Linked Data for Cross-Lingual Model Transfer in the Pharmaceutical Domain. In *The Semantic Web – ISWC 2020,* pp. 499–514. Springer, Cham.

Kemps-Snijders, M., Windhouwer, M., Wittenburg, P., and Wright, S.E., ISOcat: Corralling data categories in the wild, In: *Proceedings of the 6th International Conference on Language Resources and Evaluation, LREC 2008*. Marrakech, Morocco, pp. 887–891.

McCrae, J. P., Aguado-de Cea, G., Buitelaar, P., Cimiano, P., Declerck, T., Gomez-Perez, A., Garcia, J., Hollink, L., Montiel-Ponsoda, E., Spohr, D., and Wunner, T. (2012). Interchanging Lexical Resources on the Semantic Web. In *Language Resources and Evaluation*, 46(4), pp. 701–719.

Hellmann, S., Lehmann, J., Auer, S., and Brümmer, M. (2013). Integrating NLP using linked data. In *Proceedings of the 12th International Semantic Web Conference, ISWC 2013*. Sydney, Australia, pp. 98–113.

Tandy, J., Herman, I., Kellogg, G., et al. (2015). Generating RDF from Tabular Data on the Web. W3C recommendation, World Wide Web Consortium.
URL https://www.w3.org/TR/csv2rdf/

Ziad, H., McCrae, J., and Buitelaar, P., (2018). Teanga: A Linked Data based platform for Natural Language Processing. In *Proceedings of the 11th International Conference on Language Resources and Evaluation, LREC 2018*. Miyazaki, Japan.