Proceedings of the 1st Joint SLTU and CCURL Workshop (SLTU-CCURL 2020), pages 121–125
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Scaling Language Data Import/Export with a Data Transformer Interface

Nicholas Buckeridge, Ben Foley
The University of Queensland
The Centre of Excellence for the Dynamics of Language
bucknich@gmail.com, b.foley@uq.edu.au

Abstract

This paper focuses on the technical improvement of Elpis, a language technology which assists people in the process of transcription, particularly for low-resource language documentation situations. To provide better support for the diversity of file formats encountered by people working to document the world's languages, a Data Transformer interface has been developed to abstract the complexities of designing individual data import scripts. This work took place as part of a larger project of code quality improvement and the publication of template code that can be used for development of other language technologies.

Keywords: language documentation, low-resource languages, automatic speech recognition, data conversion, Python, design patterns

1. Introduction

In the development of speech recognition language technologies, supporting the import and export of the wide range of language data formats currently in use presents a challenge. The tools available for language documentation, description and analysis produce many different formats, which makes it unfeasible to write individual import and export scripts for each format. This work aims to increase the range of corpora that tools such as Elpis (Foley et al., 2018) can feasibly import and export. To develop an understanding of the range of language data formats commonly used, archives including PARADISEC (Thieberger and Barwick, 2012), ELAR[1], OpenSLR[2] and Open Speech Corpora[3] were analysed. Existing technologies such as Salt and Pepper (Druskat et al., 2016) were reviewed to determine their suitability as a conversion engine for Elpis. For reasons of maintaining support for a commonly-used language documentation format, and concerns about increasing the complexity of the Elpis codebase, we developed a Python interface which simplifies the process of converting language data formats into an intermediate format. By identifying the design parameters of relating code structure to workflow processes, code simplicity and configuration flexibility, an "abstract factory" design pattern was determined as the architecture of the work. The development of Data Transformers, using abstract data manipulation factories, has given Elpis the capability to support importing a wide variety of transcription data formats.

2. Background

2.1. Transcription

There are many motivations for transcribing spoken language. Building collections of transcribed recordings is beneficial for language documentation as a multipurpose, lasting record of a language (Himmelmann, 2006). Transcription is currently a critical requirement for a spoken language to have a digital presence. Language technologies such as mobile keyboards, speech recognition (ASR) tools, translation systems, and text-to-speech require some degree of language in text format to train or develop the systems (van Esch, 2019).

Producing transcriptions is time-consuming; on average it takes 40 hours to transcribe one hour of audio (Foley et al., 2019). Given this "transcription bottleneck", most language workers will never get to transcribe all the speech that they have recorded (Bird, 2013; Brinckmann, 2009; McDonnell et al., 2018). The overwhelming effort required results in data graveyards, extensive collections of un-annotated audio data accumulating with limited use to anyone (Himmelmann, 2006). The lack of annotations also limits the use of the recordings in language technologies such as translation systems.

Software expertise and software literacy can hinder language community members from transcribing collections of recordings themselves, as the dominant tools used in transcription tend to involve a steep learning curve. Traditionally, transcription has been done by outsider language researchers, although this is a trend which is starting to see some change, with communities such as the Seneca language group using the Kaldi speech recognition toolkit to transcribe their own recordings (Jimerson et al., 2019).

2.2. Data Formats

The wide range of recording practises, technologies and software requirements used in language documentation and transcription activities have fostered a great variety of data formats. As an example, an analysis of the files in PARADISEC, an archive that supports work on endangered languages and cultures of the Pacific and the Australian region, shows >220,000 files with 43 file types, including approximately 20 media formats and eight formats generated by transcription software (refer to Table 1). Media files dominate, and ELAN files (.eaf) make up the most substantial portion of transcription file types (see Figure 1) . The variety of formats presents a challenge when designing language technologies which rely on importing files created or processed by other tools.

Standards have emerged to improve the interoperability of language tools. In recent years, there have been proposals to standardise formats such as XIGT (Goodman et al.,

[1] https://elar.soas.ac.uk
[2] https://www.openslr.org
[3] https://github.com/jrmeyer/open-speech-corpora

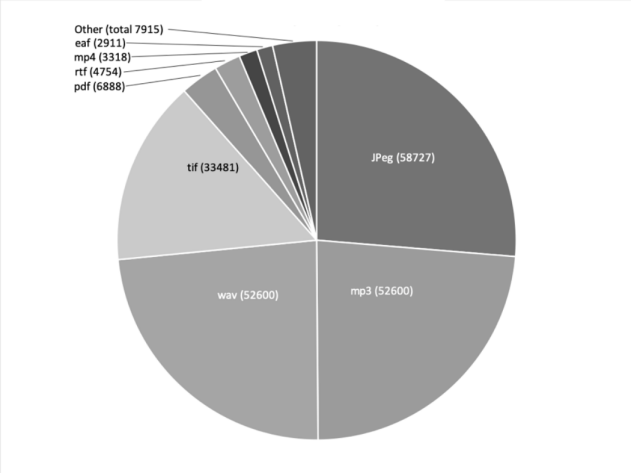

Figure 1: PARADISEC file types

2015) for Interlinear Glossed Text; and to extend XML processing tools such as XPATH (Bird et al., 2006). Framework specifications such as EXMARaLDA span working with individual transcriptions through to corpus management (Schmidt and Wörner, 2009). Wider adoption of tools such as SayMore[4], and proposals like Holton and Thieberger's collections management tool, recently released as Digame[5] would make ingesting material into archives a more reliable and quicker process, and improve downstream processes such as ASR which rely on access to language corpora. Corpus conversion tools such as Salt and Pepper map between different language data formats using graph data structures (Druskat et al., 2016).

2.3. Corpus Formats

A selection of recording collections was sourced from archives and online repositories to facilitate the design of the Data Transformer interface. Source diversity was important to ensure that the interface design was generalised, rather than fitting too specifically to one archive. Smaller specialised repositories such as OpenSRL and Josh Meyer's Open Speech Corpora tended to publish files in simple structures, usually in one ZIP file, as opposed to large and highly organised repositories. These collections were typically internally organised for application with specific ASR tools, which can make pre-processing these corpora more complicated. Highly organised repositories such as PARADISEC or ELAR store copious amounts of data grouped into collections and tend to have more uniformity across the whole collection. Each file in these archives is paired with metadata and has permissions controls. Since some of the permissions restrictions do not allow open access, or use a request-for-access rule, these were more difficult to download. There are no single collection download mechanisms for these archives as sometimes permissions would differ per-file in the same collection.

[4]https://software.sil.org/saymore
[5]https://go.coedl.net/digame

File type	Number of files
jpg	58727
mp3	52600
wav	52600
tif	33481
pdf	6888
rtf	4754
mp4	3318
eaf	2911
mxf	2327
txt	1378
webm	1144
mov	880
JPG	597
tiff	355
xml	203
mpg	155
qua	140
trs	120
png	113
TextGrid	102
pfsx	76
flextext	60
docx	55
cha	39
tab	39
lbl	30
dv	21
csv	14
fwbackup	12
img	8
MP4	8
MTS	8
xlsx	7
avi	5
ixt	4
m4v	3
md5	3
TIF	3
kml	2
doc	1
EAF	1
mpeg	1
xhtml	1

Table 1: PARADISEC file formats

2.4. Elpis

Elpis is being developed to provide an accessible interface to speech recognition tools, to accelerate the process of transcription (Foley et al., 2018). Early work focussed on writing a suite of Python scripts to assist in working with the Kaldi speech recognition toolkit. Scripts were written to clean and normalise audio and text training data; prepare the intermediary file formats which Kaldi requires; to move files into the directories required by Kaldi; and to run a Kaldi recipe to train the ASR models. This early work was operated by typing instructions into a command-line interface to run the Python scripts, and supported working with ELAN, Transcriber and plain text file formats. Subsequent development resulted in the design and development

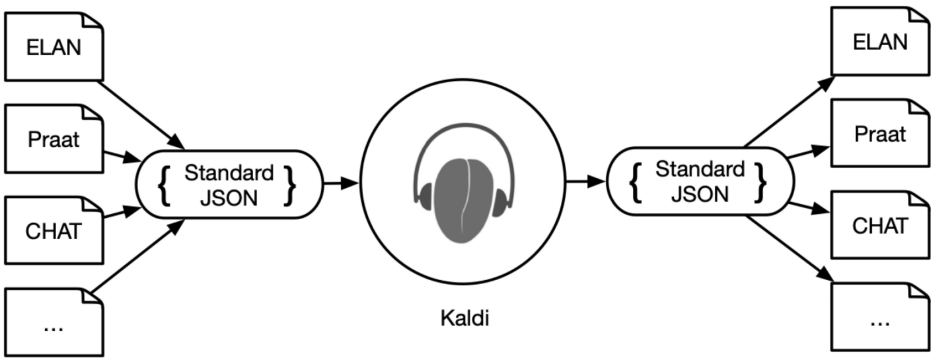

Figure 2: Data conversion mental model

of a graphical user interface (GUI) to provide a way of running Kaldi for people who had no experience with using command-line interfaces, however the GUI only imports ELAN files. Support of multiple file formats was lost due to limited development time preventing the implementation of import options.

3. Method

3.1. Approach

This work began by researching existing approaches to converting language data formats, and investigating methods which language tools use to import and export data. The initial plan to broaden Elpis' support for more file formats was to incorporate existing conversion technology, Salt and Pepper, into the Elpis pipeline. Two options were considered, firstly for Elpis to interact with Pepper via command-line calls; secondly to use a Python-to-Java bridge library for Elpis to interface with Pepper. Early tests showed that support for ELAN files with linked media was not complete[6]. Given that linked media in ELAN was commonly found in the language documentation contexts for which Elpis was originally designed, in addition to concerns about the complexity of writing wrappers around Pepper, a decision was made instead to develop a Data Transformer interface using Python. The mental model representing the data conversion process of inputs and outputs which Pepper uses was maintained in the design of the transformer interface, using Elpis' existing "Kaldi JSON" object structure as the intermediate data format (see Figure 2).

3.2. Data Analysis

After acquiring sample data sets, the data structures, formats and metadata were investigated to gain an understanding of the required features of each collection and data format. In general, collections from OpenSLR and Open ASR lacked standardisation. Another challenge facing these individual repositories was discerning the modalities of the data, with some corpora being collections of speech recordings and text, while others were image and text collections. Extreme cases of specialisation were found in the listings, including corpora that had been prepared for use directly in a machine learning toolkit. These over-specialised corpora (for example, most collections from OpenSRL) included complex directory structures and configuration files that would be required for specific speech recognition tools.

[6]https://github.com/korpling/pepperModules-ElanModules

3.3. Language Technology Architecture

Elpis is built according to an architecture of a sequence of software layers which interact via a programming interface (API). The API allows the user interface to be decoupled from the processing scripts and speech recognition toolkit, a design which enables the current speech recognition toolkit to be swapped out for another with minimal disruption to the user interface, or the development of different user interfaces for different user groups. A bare-bones version of these software layers has been published as an open-source project "Language Technology API pattern"[7], a template for other language technology projects.

3.4. Design

Designing the Data Transformer interface required adhering to the Elpis philosophy that the codebase and user interface should directly reflect the workflow process; that the code should be simple enough for a novice to understand; while allowing flexibility in configuration if necessary.

A key requirement for the transformer design was that the specification (or description) of a data format should be separate from its instantiation as a data transformer. To help implement this, "design patterns" were used. Design patterns are generic solutions to problems that match a pattern (Sommerville, 2011; Shvets, 2019). To choose a design pattern, the properties of a problem first need to be identified. For the design of the Data Transformer API, these were:

- there is one specification object per format;
- each specification can create multiple importers or exporters for that format; and
- each importer/exporter can be individually configured.

These properties fit the "abstract factory" creational design pattern. The purpose of an abstract factory is to provide an interface to create a family of related objects without specifying the concrete classes (Shvets, 2019). This pattern was implemented for the Data Transformer API.

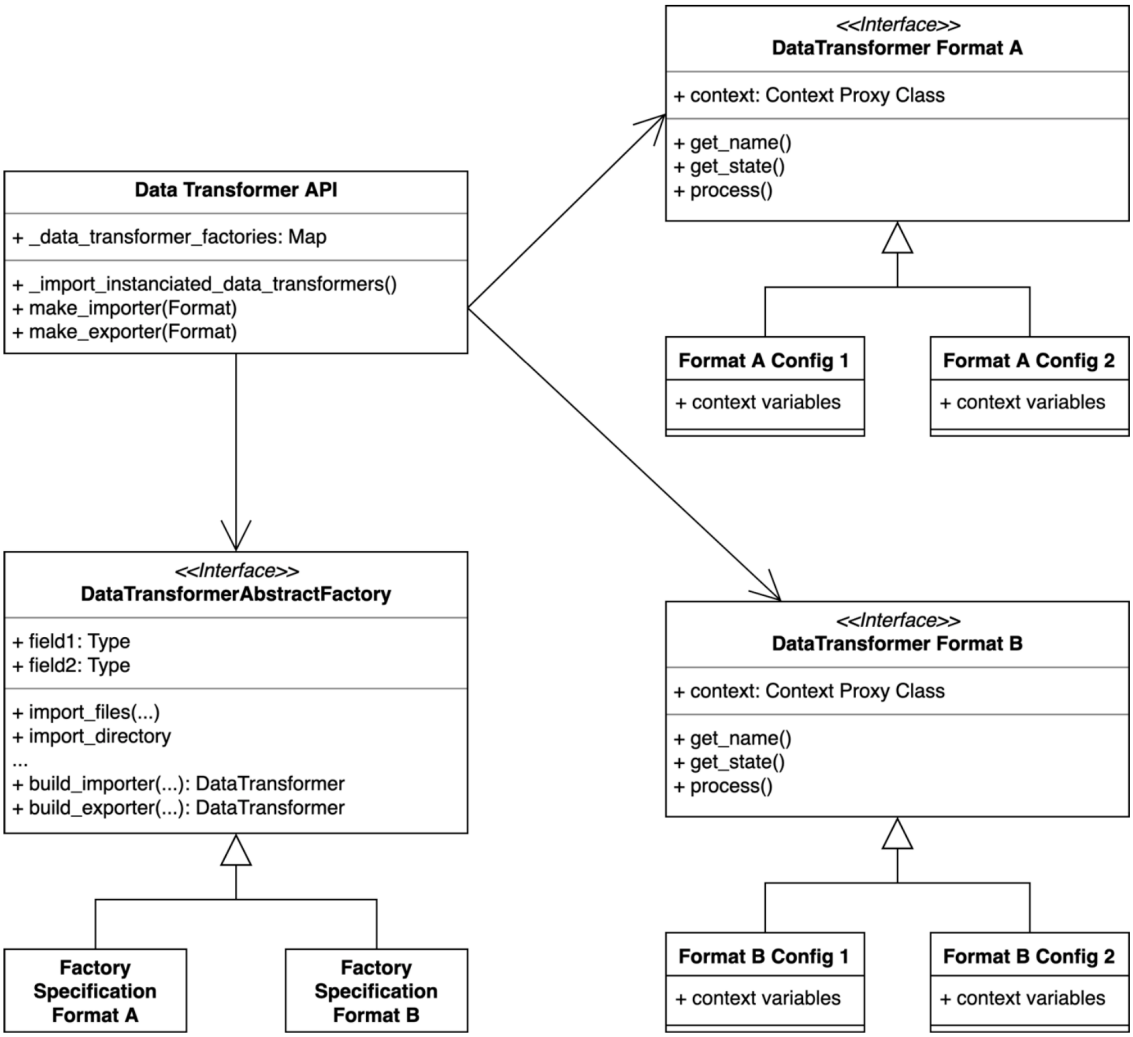

Figure 3: The transformer architecture

3.5. Data Transformer Architecture

The Data Transformer API component of the transformer architecture represents the Python *transformer* module. Import/export formats are specified by extending the *DataTransformerAbstractFactory*, represented here by Factory Specification Formats A and B. If two formats A and B are specified by extending the *DataTransformerAbstractFactory*, then as the properties indicate, concrete *DataTransformers* of the relevant format can be instantiated at any time.

When the API has a request to build a new data transformer, the API will attempt to find that format's factory if it exists. Then the factory uses a base *DataTransformer* class and in the build process, attaches the specification behaviour as per the factory's specification. The factory constructs a *process()* method specialised for the format. It also attaches the bound functions as a Python object attribute by name to the data transformer being built. This flexibility feature is in case an expert user wishes to call the bound function directly, but is not recommended for regular users.

[7]https://github.com/CoEDL/LT-API-pattern

```
from elpis.transformer import
    DataTransformerAbstractFactory
elan=DataTransformerAbstractFactory('Elan')
```

Example code 1: Elan factory

In Example code 1, the variable *elan* is a new factory. The factory constructor takes one argument, the name of the data transformer. *DataTransformerAbstractFactory* has informative methods that change the behaviour of the produced *DataTransformer* object. Building on the example shown in Figure 4, an implementer can inspect audio extension set, and import/export capabilities.

4. Future Work

4.1. Multi-threading Optimisation

During this work the observation was made that Elpis has limited support for multiprocessor computer resources. Performance optimization of Elpis hasn't been a high priority in its development, which has led to the sporadic use of multi-threading. Because of this, it was unclear if it was

safe to include parallel-processing techniques into the design of the data transformers. Future work would improve Elpis' support for multi-threading and update the data transformer interface to make use of multiprocessor computer resources.

5. Conclusion

The development of Data Transformers, using abstract data manipulation factories, has given Elpis the capability to support importing a wide variety of transcription data formats. The software architecture uses "abstract factory" design patterns to ensure the implementation covers a range of known corpora and is scalable for unseen formats. Factory specification methods follow good design practice with extensive documentation and unit testing. A data transformer to import Elan files has been fully implemented to demonstrate the process of using the abstract factory methods. Through this work, Elpis is now in a position to be developed quickly to accommodate the requirements of more language workers and their diverse data formats.

6. Bibliographical References

Bird, S., Chen, Y., Davidson, S. B., Lee, H., and Zheng, Y. (2006). Designing and evaluating an XPath dialect for linguistic queries. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 52–52. IEEE.

Bird, S. (2013). Androids in Amazonia: recording an endangered language. *The Conversation*, 21-May-2019, M. Ketchell.

Brinckmann, C. (2009). Transcription bottleneck of speech corpus exploitation.

Druskat, S., Gast, V., and Krause, T. (2016). An Interoperable Generic Software Tool Set for Multi-layer Linguistic Corpora. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Foley, B., Arnold, J. T., Coto-Solano, R., Durantin, G., Ellison, T. M., van Esch, D., Heath, S., Kratochvil, F., Maxwell-Smith, Z., Nash, D., et al. (2018). Building Speech Recognition Systems for Language Documentation: The CoEDL Endangered Language Pipeline and Inference System (ELPIS). In *The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages*, pages 205–209.

Foley, B., Durantin, G., Ajayan, A., and Wiles, J. (2019). Transcription Survey.

Goodman, M. W., Crowgey, J., Xia, F., and Bender, E. M. (2015). Xigt: extensible interlinear glossed text for natural language processing. *Language Resources and Evaluation*, 49(2):455–485.

Himmelmann, N. P. (2006). Language documentation: What is it and what is it good for. *Essentials of language documentation*, 178(1).

Jimerson, R., Hatcher, R., Ptucha, R., and Prudhommeaux, E. (2019). Speech technology for supporting community-based endangered language documentation.

McDonnell, B., Berez-Kroeker, A. L., and Holton, G. (2018). Reflections on Language Documentation 20 Years after Himmelmann 1998.

Schmidt, T. and Wörner, K. (2009). EXMARaLDA–Creating, analysing and sharing spoken language corpora for pragmatic research. *Pragmatics. Quarterly Publication of the International Pragmatics Association (IPrA)*, 19(4):565–582.

Shvets, A. (2019). Dive Into Design Patterns.

Sommerville, I. (2011). *Software engineering*. Pearson, Boston, 9th ed., international ed.. edition.

Thieberger, N. and Barwick, L. (2012). Keeping records of language diversity in melanesia: the pacific and regional archive for digital sources in endangered cultures (PARADISEC). *Melanesian languages on the edge of Asia: Challenges for the 21st Century*, pages 239–53.

van Esch, D. (2019). Building Language Technologies for Everyone.