READ

RECOGNITION & ENRICHMENT
OF ARCHIVAL DOCUMENTS

D5.2
Language Toolkit and Resources P2

Maximilian Bryan, Nathanael Philipp
ASV

Distribution: http://read.transkribus.eu/

READ
H2020 Project 674943

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943

Project ref no.	H2020 674943
Project acronym	READ
Project full title	Recognition and Enrichment of Archival Documents
Instrument	H2020-EINFRA-2015-1
Thematic priority	EINFRA-9-2015 - e-Infrastructures for virtual research environments (VRE)
Start date/duration	01 January 2016 / 42 Months

Distribution	Public
Contract. date of delivery	31.12.2017
Actual date of delivery	27.12.2017
Date of last update	27.11.2017
Deliverable number	D5.2
Deliverable title	Language Toolkit and Resources P2
Type	Report
Status & version	In process
Contributing WP(s)	WP5
Responsible beneficiary	ASV
Other contributors	
Internal reviewers	URO, UPVLC
Author(s)	Maximilian Bryan, Nathanael Philipp
EC project officer	Martin MAJEK
Keywords	Language Toolkit, Dictionary, XML, Crawling

Contents

1 Executive Summary

This deliverable describes the current state of the language resources toolkit that is to be developed in WP5.

This task deals with developing a toolkit for various data processing steps. The processed data is then used by modules such as HTR, Layout Analysis, Document Understanding as well as Table and Forms processing. Sources that provide data are typically text documents that come in different formats such as XML, PDF, websites or Docx. This year has seen improvements at dealing with all these different source types. The focus was on working on XML, especially TEI, since most data was in this format. The challenge when working with TEI is that, although there is a standard for this format, there stil exists a high variability in the format. Thus, a tool has been developed that is flexible enough to work with these different types of formats.

2 Language resources

Language resource are used to support other tools like HTR. E.g., HTR benefits from a dictionary of the words that have to be recognized in an image (see D7.11). With an incomplete dictionary, the results would be far worse than with a rich one. To provide these helping resources, other sources have to be processed. These other sources come in a wide variate of formats like XML, PDF, websites or docx that have their upsides and downsides. This section describes the challenges when working with these resource types.

2.1 Docx

.docx is a format by Microsoft and is used by Microsoft Word. *.docx* differs from the also famous *.doc* format in that it is open and not proprietary. Thus, third party or non-Microsoft tools respectively exist to deal with files in that format. Since our tool is developed in Java, the library *Apache POI* is used, which is developed especially for dealing with tools in Microsoft formats. Using that tool, we are able to extract text from *.docx* documents. A downside when dealing with that format is that it is not possible to say how many pages a document has or, consequently, on which page a word or sentence is: The raw format contains the text and its markup. When a document is being displayed, Microsoft Word builds the document pages depending on font size and other properties. The creation of a new page is determined at that moment. Thus, only when a *.docx* document is being displayed, it is known how many pages it has.

2.2 PDF

Similar to *.docx*, *PDF* is a format for documents. Since it is standardized, there exist libraries for dealing with that format. We are using *Apache PDFBox*. Contrary to *.docx*, PDF documents are saved with an already defined number of pages. Thus when extracting text, we explicitly can extract text and then tell from with page of the document that text comes from.

2.3 XML

The most flexible and thus most challenging format to handle is XML. Technically speaking, *.docx* is an XML format as well, but it is standardized. Another XML format, which has been implemented in our tool, is TEI. TEI differs from other formats in that sense, that its markup is less used to determine how to display the document (as is it the case with *.docx*). It is rather used to add semantic annotations to the text:

```
1 [...]
2 <p>
3   <lb break="yes"/>
4   Curia balliuorum de Abirden tenta in pretorio eiusdem 3<hi
        rend="superscript">o</hi> die
5   <lb break="yes"/>
6   mensis nouembris Anno quo supra
7  </p>
8 [...]
```

Listing 1: Examplary snippet from TEI-XML

When extracting text from that kind of documents, one has to know how to interpret different tags and to keep in mind that, although different documents are all using TEI, they can use different tags for the same thing. Listing 1 contains an examplary snippet from a TEI document. It shows a paragraph, marked with the tag *p*. The following list contains all the found tags used for line breaks:

- p
- l
- head
- pb
- pb, with the attribute *break* as shown in Listing 1

When extracting text from TEI documents, some page tags contain a hint which indicates the image file corresponding to the current document page. All the until now seen formats define page breaks by the tag *pb*. This tag often, but not always, contains an attribute called *n* which refers to the page number inside of the document. Sometimes, this number refers to the page number the document itself has, that is the number that is printed on the page. In other cases, this number refers to the number of the image file, which can include the front page of the document, e.g. the book cover. In the latter case, the page numbers in the XML file and the page numbers on the pages do not correspond.

3 Architecture

The architecture of the language resources toolkit has been overhauled this year. The first year, all the source code and dependencies have been included in one Java/Maven project. This was easy for developing the requirements, but some of the tools that use the toolkit need a smaller dependency and it is not appropriate to include all the depencies the toolkit needs. Thus, we changed the structure to make it more modular and so that other tools can include only that part of the toolkit they need.

The current structure is as follows:

- TranskribusCrawler
- TranskribusDictionaries
- TranskribusTokenizer
- TranskribusExtractors
  - TranskribusCoreExtractor
  - TranskribusDocxExtractor
  - TranskribusPDFExtractor
  - TranskribusXMLExtractor
  - TranskribusGenericExtractor

The crawler needs external libraries that are not needed by other modules. The tokenizer is used by different tools and so it is desired to keep that module and its dependencies as small as possible. The extractors are seperated as well so that each one has its own external libraries needed. The module *TranskribusCoreExtractor* defines interfaces that are implemented by the modules *TranskribusDocxExtractor*, *TranskribusPDFExtractor* and *TranskribusXMLExtractor*. The *TranskribusGenericExtractor* is used to extract text from any kind of file. It checks the file extension and then calls the corresponding extractor. It is also used to extract files from *.zip* archives and then it does the text extraction recursively.

4 Future work

When extracting text from different resources, other tools need text from pages being written into text files, whose file names correspond with the image files that contain that text (D7.20). The file names used in the TEI-XML sometimes are different or are not given at all. In the future, there has to be a heuristic implementation of how to align these two things. Also, when more users are using the system and do have different text resource, the toolkit has to be further enhanced to satisfy these new requirements.