Digital Critical Edition of Apocryphal Literature: Sharing the Pipeline

Violeta Seretan

Institut romand des sciences bibliques, Université de Lausanne
Violeta.Seretan@unil.ch

Abstract
The emerging field of Digital Scholarly Editing, concerned with the application of the digital paradigm to textual criticism, offers a range of software solutions for assisting critical editors in their task of documenting textual variation. But how to go from a disparate set of tools and resources to an integrated pipeline, one in which the data travels seamlessly from one format to another while meeting the requirements of each component? In this paper, we explain how we build and share an integrated processing pipeline that takes us from manuscript transcriptions in TEI XML format to a graph representation of textual variation, which constitutes the basis for the editorial work and the creation of the actual edition. With Docker Compose as the only technical prerequisite, running the pipeline is only one command away: the environments needed to run each software component are set up automatically, the processing begins, and at the end, a web server is launched which displays the automatically-built variant graphs ready for manual analysis using a dedicated online tool, Stemmaweb. This is an example of how technological advances are exploited to alleviate the technical burden put on editors.

keywords
digital scholarly editing; critical editing; textual variation; digital humanities workflows; software encapsulation; Docker; Docker Compose

INTRODUCTION

Within a digital paradigm, the process of textual criticism can be decomposed into the following main steps: transcription of manuscripts, collation, analysis, and edition [Andrews, 2013]. Collation, in turn, is decomposed into tokenisation, normalisation, and alignment [The Interedition Development Group].

We briefly describe each step (Section I), before introducing the building blocks of the processing pipeline from a technical point of view (Section II), then the pipeline itself (Section III).

The software and all the described material, including sample input and output data, are freely available online at https://github.com/seretan/IRSB-integrator.

I COMPUTER-ASSISTED TEXTUAL CRITICISM: MAIN STEPS

1.1 Transcription

The editing process starts with the full transcription of each witness (instance of a text found in a manuscript), performed manually using the TEI encoding guidelines (cf. Macé et al [2019]).[1]

1.2 Tokenisation

Once the witnesses are available in digital format, each text is automatically split into units (tokens) in order to enable comparison across versions. The splitting takes into account the

[1] Automatic transcription shows promising results, thanks in particular to the advances made by machine-learning approaches to Handwritten Text Transcription. However, the current performance does not warrant the exclusion of the human expert intervention.

XML context. For instance, in the case of substitution, encoded using the TEI *subst* tag, tokens are collected from the *add* rather than the *del* child, if the change was performed by a scribe; otherwise, if is was made by a corrector, the whole XML encoding is considered as a single token.

1.3 Normalisation

Optionally, language-dependent and language-independent transformation of tokens is performed to facilitate comparison (removing spiritus in Greek is an example of the first type; for an example of the second type, see Section 2.1 below). The original and the transformed forms coexist.

1.4 Alignment

The witnesses are automatically compared ("collated") and an alignment table is produced, similar to, but more explicit and more comprehensive than, the collations manually produced using Excel tables in a traditional way. Each table row lists the token variants across witnesses, with empty cells denoting absence of the compared token from a given witness.

1.5 Analysis

The collation output represents the textual variation: rows with non-identical content correspond to loci critici, which the editor analyses in order to come up with a directionality ("polarisation") hypothesis (i.e., which variant was copied from which, cf. the primary/secondary variant distinction).[2]

1.6 Edition

Accordingly, primary variants are marked as lemmas (selected readings) and are included in the edited text, while secondary variants (or variant readings) are listed in the critical apparatus. The edited text and the critical apparatus, together with complementary material, constitute the content of the edition, which is then formatted according to specific guidelines and published.

II BUILDING BLOCKS OF THE PIPELINE

Critical editions of ancient texts, with translation and critical notes, take years of dedicated work (depending on the length of the text, the number of manuscripts, etc). The editing process can never be automatised, because editing is an art [Huygens, 2000]. However, parts of this process can be done by the machine, or can be assisted by the machine. In particular, steps 1.2 to 1.4 can be performed entirely automatically, although the output is perfectible, the collation results requiring manual revision.

In this section, we present the technical solutions we adopted for these steps, with a focus on interoperability, i.e., on how we cope with the input/output format requirements of each tool as part of building an integrated pipeline.

2.1 Tokenisation and normalisation: tpen2tei

The tpen2tei Python module (available at https://github.com/DHUniWien/tpen2tei) takes as input a list of TEI XML files and generates a JSON file in a format suitable to CollateX collation software (i.e., a structured text file containing a list of witnesses, each containing a list of tokens). Every token is a complex entry with multiple fields, among which 't' (the display form

2 According the so-called "external" criticism, this analysis must be consistent with the transmission hypotheses put forward by the editor, possibly with the help of stemmatological analysis software (which witness was copied from which). The so-called "internal" criticism is based on the literary analysis of the text.

of the token), ‘n’ (its normalised version), and ‘lit’ (the literal version, i.e., its actual XML encoding).
For manageability reasons, large texts are processed in chunks, which are delimited by the TEI *milestone* tag. Therefore, the program expects as additional input a list of milestones to process, and for each milestone it creates a JSON file for CollateX, as described above.
The language-independent normalisation of tokens (Section 1.3) is also performed here, since this is where the XML structure decoding takes place. For instance, in order to normalise unclear text encoded using a TEI *choice* tag, we select the content of the *supplied* child tag with the highest value of the *certainty* attribute; competing *supplied* children are ignored.
The language-dependent normalisation is performed by an external configuration module. As part of this normalisation, an external data file containing abbreviation-expansion pairs can be specified, which will be used to automatically normalise abbreviations (e.g., πρς → πατρος).
The syntax of the command for executing this module is shown in Figure 1.

```
python3 teixml2collatex.py input-folder/ output-folder/ -c configuration-module
```

Figure 1. Command for executing tpen2tei.

The milestone and abbreviation files are read from the current working directory (‘milestones.csv’ and ‘abbs.csv’, respectively).
Because our project uses a custom tokenisation method that considers “rich” tokens possibly containing XML encoding as opposed to simple text, we published our version of the code in a forked repository, namely, https://github.com/seretan/tpen2tei, branch xmlrich_tokenization.

2.2 Alignment: CollateX

The Java tool CollateX (available at https://collatex.net/) supports a variety of input and output formats. For our purposes, we use JSON for both input and output. The structure of the input file was presented is Section 2.1. The output file contains the witness index in the form of an identifier list (the sigla), followed by the collation table proper (rows with as many cells as there are witnesses, cf. Section 1.4).
In our work, we used the newest version of the tool, 1.8 – in development; source code available at https://github.com/interedition/collatex – because it fixes some issues[3] present in the official version, 1.7.1. The commands for compiling and running CollateX 1.8 are shown in Figure 2.

```
mvn -Dmaven.test.skip=true package
java -jar -Dnashorn.args="--no-deprecation-warning" collatex-tools-1.8-SNAPSHOT.jar input-file.json -t -f json > output-file.json
```

Figure 2. Commands for compiling and running CollateX 1.8.

2.3 Analysis: Stemmarest and Stemmaweb

The web tool Stemmaweb (https://github.com/tla/stemmaweb), developed in Perl (using the Catalyst framework) and Javascript (using JQuery), is the final component of the pipeline. Initially conceived as a stemmatological tool including, *inter alia*, collation revision functionalities, it is currently being transformed into a scholarly editing tool proper [Andrews, 2019]. Stemmaweb adopts a graph-based textual variation model, and the goal is “to express the entire edition—edited text, variants, emendations, and annotations—in graphical form, from which a print-style critical apparatus can be automatically generated” [Andrews, 2019]. Therefore, it relies on a graph database for its Java back end, the Stemmarest tool (https://github.com/DHUniWien/tradition_repo).
Once Stemmaweb is running, the user can upload collation files and start the manual analysis and the actual editorial work. Setting up the tools, however, is not a straightforward task.

3 https://github.com/interedition/collatex/issues/44

Stemmarest requires the installation of an Apache Tomcat web server, the source compilation, and the set up of the web application (‘stemmarest.war’). Launching Stemmaweb is relatively easier (Figure 3).

```
perl script/n4jtestdb.pl
perl script/stemmaweb_server.pl
```

Figure 3. Commands for running Stemmaweb.

III PIPELINE

With today’s technology, this technical hassle can be greatly reduced. What if, instead of installing each of the software tools one by one and making sure they work together, the user would simply enter one command and get the end results with no further intervention?

This is precisely what we achieved using the Docker encapsulation tool.

Our pipeline is defined in a Docker Compose file, ‘docker-compose.yml’ (Annex 1). As it can be seen, it contains three services: Stemmarest, Stemmaweb, and Integrator. The third is the main service. It interacts with the first two and performs all the processing, including tokenisation, normalisation and collation (cf. Section II). Each service is defined in a specific “dockerfile” (‘Dockerfile-SR-IRSB’, ‘Dockerfile-SW-IRSB’, ‘integratorDockerfile’) that is used to create an “image” for the service. The three images are publicly stored in the Docker central repository, Docker Hub.

Sharing the pipeline means that the user only needs the ‘docker-compose.yml’ file (about 20 lines of code). Then with Docker Compose installed and one command, everything is set up automatically: the images are downloaded and used for starting the three services, which communicate with each other; the main service accesses the current working directory and starts processing the XML files therein; the list of milestones is automatically identified; and finally, collations are generated and uploaded directly into the Stemmarest back end. All the user has to do is open a web browser and connect to the Stemmaweb instance running at http://localhost:3000/ in order to start editing.

```
docker-compose up
```

Figure 4. Command for running the pipeline.

Acknowledgment

The work described in this paper has been supported by the Swiss National Science Foundation through grant no. 157961 awarded to Prof. Frédéric Amsler.

References

Andrews T. L. The third way: philology and critical edition in the digital age. *Variants*, 2013;10:61-73.

Andrews T. L. The letters of Ioannes Tzimiskes in the Chronicle of Matt‘eos Urhayec‘i, in Outtier B., Horn C. B., Ostrovsky A. and Lourié B. (eds.), *Armenia between Byzantium and the Orient*. Brill (Leiden), 2019;259-287.

Huygens, R. *Ars edendi. A Practical Introduction to Editing Medieval Latin Texts*. Brepols (Turnhout), 2000.

Macé C., Rouquette M., Seretan V., Amsler F., Andrist P. and Antonelli C. Critical digital editions of Christian apocryphal literature in Latin and Greek: Transcription and collation of the Acts of Barnabas. *Storie e Linguaggi*. 2019;5(1):125-145.

The Interedition Development Group. *The Gothenburg Model*. https://collatex.net/doc/.

ANNEX 1

Pipeline definition via Docker Compose (‘docker-compose.yml’).

```
version: '3.2'

services:
  stemmarest:
    image: irsb/stemmarest
    ports:
      - 8080:8080

  stemmaweb:
    image: irsb/stemmaweb
    ports:
      - 3000:3000
    depends_on:
      - stemmarest

  integrator:
    image: irsb/integrator
    depends_on:
      - stemmarest
    volumes:
      - type: bind
        source: .
        target: /home
```