GitDOX: A Linked Version Controlled Online XML Editor for Manuscript Transcription

Shuo Zhang and Amir Zeldes
Department of Linguistics, Georgetown University
{ssz6, amir.zeldes}@georgetown.edu

Abstract

In this paper we present GitDOX, an open source online, schema aware XML annotation interface linked to Natural Language Processing tools, which uses the online GitHub platform as a backend for version controlled electronic corpus development. We apply this platform to the use case of transcribing and annotating Coptic manuscript data from first millennium Egypt, in a collaborative team. The architecture of the tool is meant to be generic and extensible, supporting an unbounded range of annotation schemes, while being simple to use for annotators without extensive training in computational tools or version control software.

Introduction

In this paper we describe a new, open source tool for XML annotation, which we are using to transcribe Coptic manuscripts for online publication. The Coptic, the language of Egypt in the Hellenistic period of the first millennium, is the last stage of Ancient Egyptian, the language of the hieroglyphs. Written in a modified Greek alphabet with six added letters, the language is a treasure trove of information for researchers in multiple disciplines. Historians use Coptic documents to study the ancient Mediterranean in the first millennium; for Religious Studies, the language offers some of the earliest Christian literature, documenting the inception of the Monastic movement; and for Linguists Coptic forms the endpoint of the longest continuously documented language on the planet, with over 4,000 years of attestation. Making Coptic texts available in digital format is therefore an important endeavor.

However, annotating manuscript material can be difficult: although there are now well documented standards for transcription, such as TEI XML (http://www.tei-c.org/), annotation schemes differ substantially in practice. Additionally, experts in Coptic Studies who can digitize manuscripts may need extensive training to be able to use such standards, and enforcing consistent use of tools and schemas subsequently becomes a challenge. If only a handful of annotation types are supported, a possible solution is to offer a rich text interface with dedicated buttons representing manuscript phenomena (e.g. a button to annotate a letter as illuminated). However, as the schema grows more complex and new annotations are added, the need for access to the underlying XML surfaces: the interface may not be able to keep up with user needs and can become hard to maintain without access to a permanent software team.

A second challenge in managing data is the need to coordinate multiple annotators working on the same materials, document progress and potentially compare, and even revert to older versions. These capabilities are becoming more commonplace in projects using Version Control Software (VCS) such as git, but are still completely unused by annotation interfaces. In fact, to the best of our knowledge, the present contribution forms the first example of an annotation interface directly using an online VCS as a backend, without any local file storage.

To meet the challenges above, we have endeavored to create a highly configurable XML editor called GitDOX (Git Data-storage Online Xml editor) which can be populated with an arbitrary annotation scheme and guide a collaborative team of project annotators while offering access to linked NLP technologies and keeping track of progress without any exchange of files.

System Architecture

In order to ensure consistency across users and be able to update the interface without distributing executables to annotators, an annotation interface should ideally be online and server based. This also means that annotators, who are often domain experts with limited computer and programming skills, can use the tools without ever having to install software, and can submit annotations without exchanging

XML files, e.g. over e-mail. A further advantage is allowing administrators to link and update auxiliary technologies, such as NLP tools facilitating manual annotation, or annotation correction. Only a handful of tools support online, server-based annotation, such as WebAnno (Yimam et al. 2013) for textual relations and span annotations, Arborator (Gerdes 2013) for dependency syntax, or rstWeb (Zeldes 2016) for discourse parsing.

GitDOX is a collaborative XML annotation system, implemented as a Python-CGI based web application running on a server, using SQLite as a local database and the GitHub platform itself for versioned XML file storage. We follow the static form-submit architecture of Arborator (Gerdes 2013), in which no running services are used: Python scripts are exposed via a Web server (e.g., Apache), and calling them from a browser accesses the DB to serialize HTML for the client. This means that no service needs to constantly run and 'listen' for an event while the application is being used, thereby minimizing server workload.

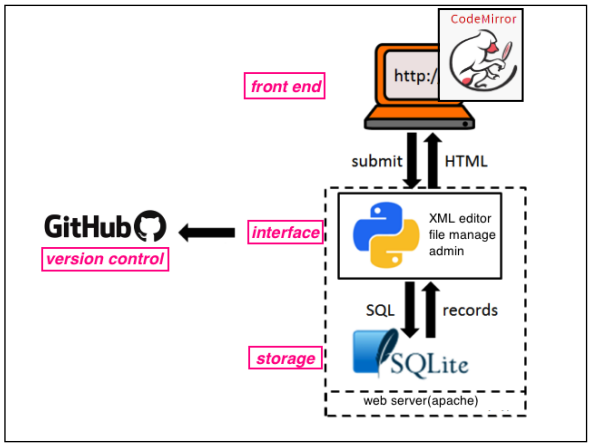

Figure 1. GitDOX software architecture.

For user interaction, we adapted the CodeMirror (https://codemirror.net/) module as our browser-embedded XML editor component. CodeMirror is an open source text editor implemented in JavaScript (see Figure 2). Using CodeMirror enables us to build the system with maximum configurability while requiring minimum server-side load. Concretely, using the CodeMirror module has the following advantages:

- Syntax and error high-lighting: by defining a set of valid XML grammars and tags, modified to the needs of specific annotation projects, CodeMirror provides syntax and error highlighting within the XML editor. This visual cue is especially crucial for annotators with limited training in XML technology, who would otherwise be prone to make mistakes during their annotation process.
- Auto complete: CodeMirror offers auto complete functionality in conjunction with a defined set of valid XML tags and hierarchy. It is worth mentioning that the defined set of valid XML tags and hierarchy can be easily modified and extended by supplying a simple text file containing such information. Therefore, the administrator is able to adapt the tool for use in any annotation project with its own XML definitions beyond the Coptic use case[1]. The auto-complete feature is also very useful for non-technically oriented users.

```
19 ⲁ|ⲫ_<hi rend="1 space right">·</hi>_ⲛ̄|ⲉⲟⲩⲟ_
20 <hi rend="ekthetic">ⲁ</hi>ⲉ_ⲉ|ⲡⲉ|ⲉⲟⲩⲟ_
21 ⲥⲉ|ⲥⲉⲟⲩⲟⲣ̄ⲣ̄<cb n="2"/>
22 <pb xml:id="YB308"/>
23 <note note="number ⲧⲏ visible in upper left
24 ⲉⲧ|ⲛⲁ|ⲥⲟⲟⲩ
25 ⲧⲛ̄_ⲧⲟⲟⲧ|ⲟⲩ
26
```

Figure 2. XML highlighted in CodeMirror .

- Customizable schema: by relying on auto complete suggestions and error highlighting to create the correct schema, we both avoid a complicated interface with many buttons, and make extending the schema with new elements trivial: there is virtually no development work required to add, remove or change the behavior of an annotation element or attribute.
- Line numbers: when this option is turned on, the line numbers in the editor interface help facilitate references to positions in text when discussing with reviewing editors and in commit messages (see Version Control). This feature is particularly desirable for our use case, since we are preserving line breaks and line numbers following original manuscript layouts.
- Lightweight: CodeMirror is a purely client side JavaScript-based module, therefore reducing server load to a minimum.

In addition to the XML document editing and annotation, we also provide support for adding metadata associated with each document (Figure 3 bottom). The system has built-in support for suggesting pre-defined sets of valid metadata fields, whereas the users also have the option to add new fields based on their needs.

The system aims to facilitate the workflow of collaborative annotation projects for users of various levels of technical abilities and administrative rights. There are three levels of system users with different levels of admin privileges: annotators, committers, and administrators. Annotators only have access to content editing, including setting the status of a document (under review, requesting assistance, etc.) and temporarily saving their work to the database (but not actually creating a time stamped, version controlled datum). Committers, in ad-

[1] For discussions of manuscript encoding and TEL XML for Coptic, see Schroeder & Zeldes 2016.

dition to annotator privileges, use their GitHub account credentials and can make commits to remote repositories (see below). Administrators have full admin privileges (including adding or removing users, configuring the system, etc.). Initially, each document is assigned to an annotator and the status is set to 'editing'. When an annotator finishes their work and the status is set to 'review', a senior editor (usually a committer or administrator) will review their work. Before data is committed by a committer, no files are saved: the current content of a document is just stored temporarily as a string in the database, which can be edited by any user with editing permissions. Figure 4 shows the sample workflow of a collaborative annotation project using the tool.

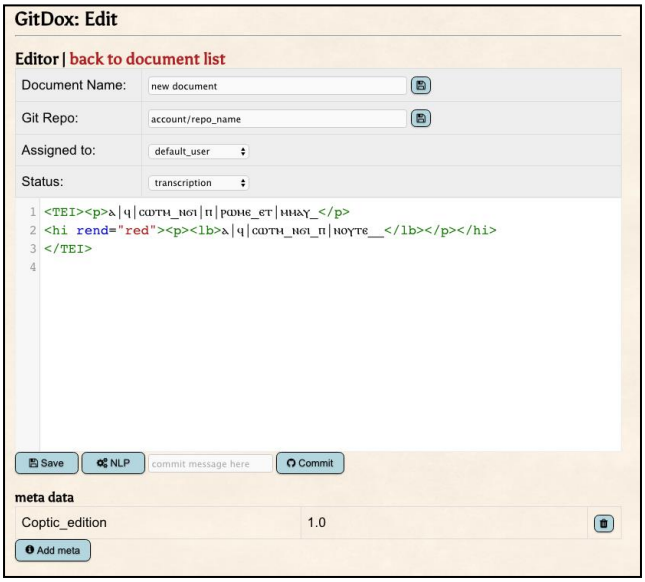

Figure 3. Example editor interface, with NLP and GitHub functionalities (commit message and commit button)

Version Control

GitDOX uses GitHub directly as a version control platform. However, users may want to save intermediate steps of their work frequently, e.g. if they leave their computer, close their browser, or just to ensure no data loss occurs. Pushing all such save operations to a GitHub repo would quickly fill the commit history, and require a cumbersome proliferation of commit messages describing changes at each save. During the annotation phase, all edited contents are therefore stored in a SQLite database and no actual files are serialized to the disk.

In order to reduce the complexity of configuring and using GitHub for end-users, GitDOX implements a GUI-based commit functionality for pushing temporarily serialized local files to the GitHub remote repository. Crucially, this functionality does not require any setup of Git local working directory by any users. Instead, it utilizes the GitHub API to directly push a temporary file to the remote repo (either to create a new file or update an existing file on the remote repository). The current implementation uses github3.py (https://github.com/sigmavirus24/github3.py), a python wrapper for the GitHub API.

When an annotated document is ready for committing, GitDOX pushes a temporarily serialized file (automatically deleted after the commit) directly to a remote repository on GitHub, along with a commit message entered by the user (Figure 3). This makes it possible for users with no experience in using VCS to have a free, state-of-the-art, and well-maintained complete functionality of version control and history tracking without the need to build an ad-hoc solution. It is also impossible to create merging conflicts, since there are no forks or branches: using the commit button overwrites the previous version of the file, which is still recoverable from GitHub. And even if users have not committed their data, they can still access their latest work from the browser on any computer after saving, since the database is on the server.

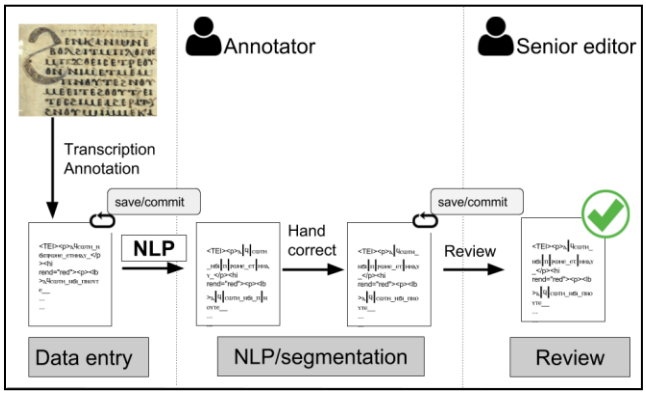

Figure 4. Example workflow of annotation and committing. Coptic manuscript image: Österreichische Nationalbibliothek, http://data.onb.ac.at/rec/RZ00002466

To view the history of the changes to any file, one can simply visit the repository on the GitHub website and will be able to see the current and past versions of the file, as well as the annotators who made the changes. In designing this simple GUI interface for GitHub, we take advantage of GitHub as our powerful backend for VCS, while minimizing the complexity needed for annotators to use Git, as well as for administrators to configure Git's local working directories for users on the server: since there is no file storage on the server, admins only need to configure the target repository for versioned files.

Linked NLP

Another major advantage of the online interface is the ability to maintain online APIs for semi-automatic facilitation

of annotation tasks. In our case, we have implemented a pilot interface to an NLP component automatically segmenting Coptic text on multiple levels (Figure 3, "NLP").

Word segmentation is one of the most time consuming tasks for Coptic manuscript entry, since Coptic word morphology is rather complex: much like in Hebrew or Arabic, word forms are spelled together in so-called bound groups (Layton 2011: 12-20), which correspond to prosodically stressed feet. As a result, prepositions, articles and clitic pronouns are all spelled together with nouns or verbs. To make matters worse, nouns and verbs themselves can contain affixes or incorporating compounds, which are also spelled together and must be split apart for analysis. The following example illustrates some of the issues:

ϩⲛⲛⲉϥϩⲃⲏⲩⲉ ⲙⲙⲛⲧⲣⲉϥϩⲉⲧⲃϧⲩⲭⲏ
hn-nef-hbēue m-mnṯ.ref.hetḇ.psyxē
in-his-deeds of-ness.er.kill.soul
'in his deeds of soul-killing'
(Besa, Letter to Aphthonia)

In this example, the first bound group must be split into three units: 'in', 'his', and 'deeds'. The second bound group, 'of soul-killing', must be split into the preposition 'of' and 'soul-killing', while 'soul-killing' itself must be analyzed to extract derivational affixes, as well as the incorporated lexical items 'kill' and 'soul'. This is important for a variety of research questions, such as the study of Greek loanwords in Coptic: in the example above, only the segment /psyxē/, 'soul', is a Greek loanword ('psyche'), which can only be recognized once segmentation has occurred.

In order to facilitate segmentation for annotators, we have linked an automatic Coptic tokenizer to our interface, using a REST API call (cf. Fielding 2000). The linked tokenizer achieves a best segmentation accuracy of 90.21% in a 10-fold cross validation evaluation and is described in (Zeldes & Schroeder 2016). Annotators can receive segmentation suggestions from the API at the click of a button, which sends the plain text of their transcription to an NLP service via a REST service call, and refreshes the CodeMirror editor with the resulting segmentation. This means that users do not have to consult an external tool and copy and paste results, but can perceive the interface as offering the NLP functionality directly. We hope to extend this functionality to other useful NLP functions in the future.

Conclusion

In this paper we presented GitDOX, an online XML editor linked to NLP services and using GitHub as a backend for version control of annotation projects. By completely avoiding file storage on the server and relying on GitHub as a repository, we get a mature VCS solution 'for free' and also hide some of the complexity of VCS management from users. The online interface means that users can concentrate on the annotation task at hand, and administrators don't need to worry about file management or e-mail communications within the project.

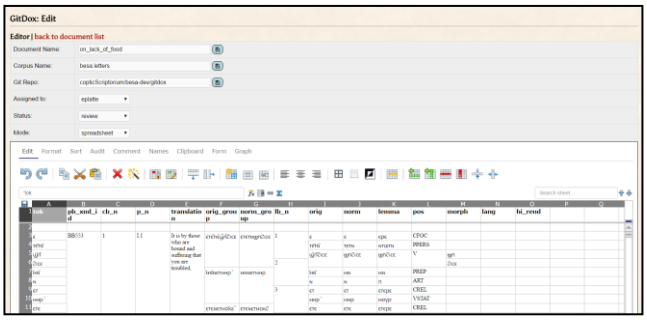

Figure 5. Example screenshot of the GitDOX editor in spreadsheet mode

Because documents are managed online and can be assigned a status and an assignee, project management is relatively simple: senior editors can assign tasks to users, check progress, and set the GitHub repository address for each document. Earlier versions can be recovered and compared (i.e. 'diffed') using GitHub itself, and commit messages, which appear to come from the users' own accounts, keep track of changes. Updates to the annotation schema or the NLP components can happen in the background and are automatically applied to all users as soon as they log in – there is never a need to update software for clients or disseminate schema files.

Currently we are applying the GitDOX editor in our Coptic manuscript annotation project with positive feedback from annotators. The project has so far digitized 96 manuscript parts (176 pages) with earlier software, which are now being ported to the new GitDox interface. All new documents are being transcribed in the system.

The GitDOX editor is general and easily configurable to fit any annotation projects with different XML tag and hierarchy definitions beyond the current use case of Coptic manuscript annotation. In addition, it is also easy to incorporate non-XML based editors into the editor interface, allowing the user to switch between multiple annotation editor modes. Shortly before publication, we added a second embedded editor next to CodeMirror XML editor component, using EtherCalc collaborative spreadsheets (https://ethercalc.org/). As a result, it is now possible to choose spreadsheet editing mode (see Figure 5) and mark up data in an annotation grid. This also opens up the possibility of binding more advanced NLP tools to add annotation in further columns, such as POS taggers, which we are currently working on.

There are still several points for improvement which we would like to work on for the future. First, progress tracking for administrators is currently doable only by monitoring GitHub commits and document statuses. Ideally we would like to have some sort of aggregate dashboard functionality to alert administrators to progress in projects or subprojects, as well as communicating with groups of users over the system. It would be also desirable in some use cases for the user to be able to access past versions of documents from the tool instead of going to GitHub. Second, we hope to extend the tool to perform verbose XML schema validation, on top of the current schema-based syntax highlighting. Finally, we have also only begun to explore binding more advanced NLP functionality. Some targets for future expansion include online dictionary lookup directly from the editor and automatic normalization, both of which would feed naturally into our workflow.

Acknowledgments

Support for this project comes from a bilateral grant from the National Endowment for the Humanities (NEH) and the German Research Council (DFG), titled KELLIA: Koptische/Coptic Electronic Language and Literature International Alliance (HG-229371-15). We would like to thank the KELLIA project team for their support and feedback, as well as three anonymous reviewers for their valuable comments on earlier versions of this paper.

References

Fielding, Roy Thomas (2000), *Architectural Styles and the Design of Network-based Software Architectures*. PhD Thesis, University of California, Irvine.

Gerdes, Kim (2013), Collaborative Dependency Annotation. In: *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*. Prague, 88–97.

Layton, Bentley (2011), *A Coptic Grammar*. Third Edition, Revised and Expanded. (Porta linguarum orientalium 20.) Wiesbaden: Harrassowitz.

Schroeder, Caroline T. and Zeldes, Amir (2016), Raiders of the Lost Corpus. *Digital Humanities Quarterly* 10(2). Available at: http://www.digitalhumanities.org/dhq/vol/10/2/000247/000247.html.

Yimam, Seid Muhie, Iryna Gurevych, Richard Eckart de Castilho & Chris Biemann (2013), WebAnno: A Flexible, Web-based and Visually Supported System for Distributed Annotations. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Sofia, Bulgaria, 1–6.

Zeldes, Amir (2016), rstWeb - A Browser-based Annotation Interface for Rhetorical Structure Theory and Discourse Relations. In: *Proceedings of NAACL-HLT 2016 System Demonstrations*. San Diego, CA, 1–5.

Zeldes, Amir & Schroeder, Caroline T. (2016), An NLP Pipeline for Coptic. In: *Proceedings of LaTeCH 2016 - The 10th SIGHUM Workshop at the Annual Meeting of the ACL*. Berlin, 146-155.