Enabling Annotation of Historical Corpora in an Asynchronous Collaborative Environment [*]

Enrique Manjavacas Arévalo
Universiteit Antwerpen
R.112, Rodestraat 14
Antwerp 2000, Belgium
enrique.manjavacas@uantwerpen.be

Peter Petré
Universiteit Antwerpen
R.229, Rodestraat 14
Antwerp 2000, Belgium
peter.petre@uantwerpen.be

ABSTRACT

Current research in Corpus Linguistics and other related disciplines within the multi-disciplanary field of Digital Humanities, involves computer-aided manual processing of large text corpora. Typically, corpus instances are retrieved with the help of concoordancers and textual search engines and subsequently labeled by hand before being submitted to quantitative or qualitative analysis. While well-established systems already cover the data retrieval aspect, less attention has been paid to the annotation process in terms of both software facilities and best practices. However, with the current increase in size and scope of research projects we envisage new needs for coordinating and managing interdependent analysis (meta-)data created being simultaneously created by different reasearchers. Current ad-hoc solutions typically involve general-purpose Real-Time Editing (RTE) and cloud storage software, whose functionality are arguably suboptimal for research purposes. In the present paper we will discuss potential problems related to coordinating annotations in large-scale projects as well as the potential benefits that can be derived from an application-based approach to annotation data management. Furthermore, we discuss some of the design decisions that arise in the process of developing a principled solution. Finally, we showcase our current implementation of a collaborative asynchronous system in the context of a Historical Linguistics research project involving several parallel studies and multiple researchers.

Categories and Subject Descriptors

J.5 [**Computer Applications**]: Arts and Humanities; I.7.1 [**Document and Text Processing**]: Document and Text Editing

Keywords

Corpus Annotations, Asynchronous Collaboration, Groupware

1. INTRODUCTION

Current corpus-based research practices entail interacting with a single ground-truth collection of documents —i.e. a corpus —, to which analysis metadata, such as annotations, are added.

In a context in which the responsability for the design of the research is not shared amongst several researchers, decisions as to how to create, design and manage annotation metadata are rather unproblematic. However, with the current increase in size and scope of research projects in the Digital Humanities?, we expect new issues to arise from the need for synchronization of concurrent research activities which could otherwise lead to inconsistency in the annotation metadata. In general, inconsistency may arise for reasons that are both intrinsic and extrinsic to the proper research activity —i.e. having to do, respectively, with interpretational disagreements as to the content and scope of annotations, and problems related to concurrent modification of the annotation metadata by several researchers. In particular, we are confronted with the following problems:

- Unifying label conventions and tagging systems, which may not be given as closed set tagging systems prior to research or may subject to revision renegotiation.
- Synchronizing the introduction and removal of labels into dynamic tag systems, guaranteeing that these updates do not lead to an inconsistent tag system.
- Ensuring that every involved researcher is notified of parallel work by other members of her team, allowing for in-place discussion of issues interferring with their own analyses.
- Allowing changes to existing annotations to be back-traceable and documented, providing a means for resetting to previous versions and for logging the motivations that led to the change (in the form of change logs, threaded discussions etc.).

Furthermore, a principled solution to collaborative corpus-based research is not only desirable for the sake of the aforementioned problems, but it also provides a number of associated advantages which have the potential to improve the quality of the research. For instance, managing annotation metadata through a specialized application implies having available these metadata in a unified and structured shape, easily allowing for the following features:

- Easy export and re-formatting functionality.

[*]This research was supported by ...

DATECH 2016 Poznań, Poland

- Storage of metadata in a database or indexing through a search engine.

- Advance querying of the metadata.

- Visualizing the metadata both as time snapshot and evolution over time.

- Enable meta-research on research practicies thanks to the availability of structured metadata derived from the researcher activities (annotation updates, user behaviour, etc.).

Although to our knowledge no prior studies have addressed these issues, both in terms of established practices and available tools[1], it can be assumed that a large part of current research requiring support for a collaborative environment typically resorts to general-purpose Real-Time document Editing (RTE) software in the best case[2], or to cloud-storage services —such as Dropbox [4] —, which provide versioning but do not support proper collaborative editing of documents, as specified, for instance, in [5].

While RTE software solutions represent powerful and thoroughly tested tools, there are also a number of drawbacks associated with their use in research. For instance, their functionality may still be too general and overshoot the specific needs of researchers, more specialized than those of common end-users. In particular, it is usually cumbersome to cross-reference newly added annotations with the corpus positions and corpus context in their scope. Furthermore, the fact that data is managed in servers owned by the service providers may not always be compatible with legal constraints around corpus data. Finally, RTE software solutions tend to prioritize user experience, which means that responsive and non-blocking Graphic User Interfaces (GUIs) are privileged in detriment of other features, such as support for document versioning, which stand in a direct trade-off with user experience.

[TODO: Brief description of the research project]

In the present paper we sketch an application-based approach addressing the aforemention issues in a way that fits better the concrete needs of the practicioner. More concretely, we propose to address the problem of managing concurrent annotation metadata by multiple researchers in the terms of collaborative working systems. Therefore, in Sections 2.1 and 2.2, we will review two major and not necessarily competing approaches to concurrent multi-user document editing (Real-time document Editing systems and Version Control Systems). Next, in Section 2.3, we argue that a hybrid approach may be better suit for the issues established in the current Section. In Section 3, we describe the research infrastructure that we have implemented following the design principles stated in Section 2. Finally, in Section 4 we shall highlight lines of further research before concluding in 5.

2. COLLABORATIVE EDITING

Collaborative editing refers to the practice by which dispersed users are able to concurrently modify shared artifacts with the guarantee that changes by different users will not overwrite each other —see also [1]. Currently, collaborative software has become ubiquitous thanks to Web 2.0 services, such as Google's web-based suite Google Docs [7] or Microsoft's Office Online [12].

We note, however, that other approaches are available beyond the RTE framework. In particular, Version Control Systems (VCS) —also known as Revision Control Systems (RCS), or Concurrent Versioning Systems (CVS) —, represent a solution to collaborative editing that is often overlooked due arguably to its less user-friendly nature and more complex document updates procedures.

In general, RTE and VCS can be seen as alternative solutions to collaborative editing [1], differing with respect to, respectively, whether edit operations are non-blocking synchronous operations, and immediatly reflected on the shared data, or asynchronous —i.e. updates are first piped through a working copy which is only sporadically merged with the source document. In the next two Sections (2.1 and 2.2) we summarily describe RTE and VCS systems.

2.1 Real-Time document Editing

RTE software systems aim at solving the synchronous flavour of the collaborative editing problem, which can be describe in terms of the following two-step procedure [9]: first, local changes are immeditaely reflected in the local copy of the document and, secondly, the changes are propagated to other users across the network. More concretely, RTE systems can be defined by the following characteristics [15]: (i) the user responsible for a change is able to observe it *real-time*; (ii) collaboration is allowed to take place in a *distributed* way across different networks —which involves a layer of non-deterministic communication latency; (iii) the system allows for *unconstrained*, non-locking edit operations —i.e. any user can edit the shared data at any time.

Thus, one major technical puzzle a RTE system has to solve consists of the fact that changes are propragated across networks with different communication latencies, therefore not reaching all end-points in the same order. For example, given the string "efecte" and concurrent edit operations *Ins(2, "f")* —i.e. insert character "f" at position 2 —and *Del(5)* —i.e. delete character at position 5: "e" —, the spell-corrected string "effect" can only be obtained if *Del(5)* is executed before *Ins(2, "f")*. But if the operations were issued by different users and reached a third user in different order due to different communication latencies in the network, the document seen by the third user would reflect an inconsistent state.

A common solution to consistency maintenance is Operational Transformation (OT) [14], a technique that consists in a rule-based transformation of edit operations which amends the position of an edit operation against a previous edit. In the previous example, the edit sequence *[Ins(2, "f"), Del(5)]* should be transformed as follows:

$$OT([Ins(2, \text{“f”}), Del(5)]) \Rightarrow [Ins(2, \text{“f”}), Del(6)]$$

to ensure consistency with respect to the alternative edit sequence *[Del(5), Ins(2, "f")]*.

Finally, OT-based RTE systems work with linear datastructures, typically lists of characters, but it is possible to work with list of elements at other levels, such as words, paragraphs or XML nodes [9, 14]. For instance, in a RTE annotation application, one might want to consider labels (annotation keys and values) instead of characters as the

[1] [TODO: look for (tentative) studies in this direction.]

[2] See, for instance, [13] and [16], for examples of reports on using Google Docs for research.

atomic targets of edit operations.

2.2 Version Control System

VCSs have a large tradition originating in the 1970s as groupware for enabling collaborative software development and more recently have become widely adopted as a result of integration with a social network environment (GitHub, BitBucket). Current VCSs follow the copy-modify-merge model, also known as "optimistic", to collaborative document editing, in opposition to the "pessimistic" lock-modify-unlock model in use among early VCSs such as SCCS, RCS or CVS [11]. In order to guarantee consistency maintenence the lock-modify-model forces users to first acquire a lock for the target file and release it once changes are finished. However, the lock model proved to be too restrictive in practice.

The optimistic flavour of VCS represents an asynchronous collaboration model based on a two-step procedure of local editing and merging with the shared document.

In principle, this two-step procedure resembles the one described in Section 2.1 for RTE systems. However, they essentially differ in that in VCS local changes are not merged immediately (synchronously) into the shared document (as opposed to edit prograpagation within the RTE framework); instead they are "commited" at a later stage —i.e. asynchronously. By separating edit and merge logic real-time responsiveness is sacrified but in exchange two crucial features are gained: (i) the concept of a *working copy*, (ii) the fact that edits are meaningful —in the sense that succesful merges can be deemed as intended *versions* of the document. We now turn to these aspects in more depth.

2.2.1 Working copy

A working copy is a local copy of the shared document that allows users to modify the document (in their local environment) despite parallel work by other users [3]. The concept of working copy has the potential for VCS implementations that are completely distributed —known as DVCS (Distributed Version Control Systems).

The general collaborative workflow with working copies can be described as folllows: When a user wants to update the shared data to reflect the local changes in her working copy (commits), a "push" operation is issued. Similarly, other users can update their local working copies to reflect the current state of the shared data —an operation known as "pull" in some implementations of this model. In both cases a conflict between the local and shared copies may arise if (i) the shared copy was successfully updated in a particular way by another user before the local changes are pushed, (ii) or there have been local changes incompatible with the shared copy before the latter is pulled. VCS provides a merging mechanism to solve these potential version divergences known as three-way merging, which involves all three document copies (both conflicting copies and their shared ancestor) [1]. In general, VCSs provide automatic merging in cases of "syntactically" unambiguous and it is commonly resorted to manual merging in cases of "semantic" ambiguity, in which conflicting edit intentions may be present.

A crucial aspect of working copies is that they allow for more complex and flexible workflows than the previously depicted case of a single working copy. In particular, users may create multiple working copies, each of which can focus on a certain aspect of the project, and delay merging to a later stage. By this token, the workflow can be seen as a tree of branches (working copies representing parallel versions of the shared data) that are being splitted and merged with the trunk (the shared data)[3].

2.2.2 Versioning

Versioning is a powerful feature for backtracking changes and restoring project state to a previous one. Versions can be generally defined as the state of a document in a given time point. In the context of VCSs, a version refers to the project state resulting from a successful merge. Moreover, the asynchronous nature of VCS commits allows attaching metadata to commits before merging. This is in opposition to RTE document updates that are triggered automatically after any modification and propagated to other users.

Besides enabling responsability traceability, version metadata provide grounds to build useful collaborative features with respect to the following two aspects: (i) edits to a document become meaningful allowing for the specification of the motivations behind the edit as well as to link to metadata in previous versions in a structured way —while structuring of the metadata is not enforced by default, it can be easily introduced by following particular internal project conventions; (ii) the resulting tree-shaped edit history can be easily traversed back, choosing which particular development branch should be followed, and therefore enabling more specific state restoration.

Finally, as a consequence of the concept of working copies, project state can be again restored to a certain version independently of concurrent actions by other users, allowing for parallel versions of the same document. In fact, another popular feature of VCS implementations is "forking" (creating a parallel version of the project, which may eventually be merged again with the original trunk).

2.3 Advantages of Asynchronous Collaboration for Corpus Annotations

Having outlined RTE and VCS approaches to document collaborative editing, we now proceed to discuss their suitability and aptness to handling synchronous corpus metadata creation.

We note that RTE is conceptually simpler (although its implementation is non trivial and constitutes a topic of current research): from a user perspective, little overhead is added to the actual annotation process. This conceptual simplicity —together with high GUI responsiveness —, a better user experience, which explains why RTEs are more widely used by average users; however, it also means that the synchronization logic is done in the background beyond user influence. VCS is, in comparison, more involved and requires grounded knowledge of the underlying synchronization mechanisms, but it also offers a more powerful collaboration model that supports complex collaborative workflows and better version metadata handling.

In our view, the asynchronous copy-modify-merge approach provides a better fit for creating and managing research metadata, since it favours expliciteness with respect to new edit operations (version metadata) and flexibility (branching, as a result of working copies not only as a solution to availability despite network partitioning, but also as a possi-

[3] Strictly speaking, the graph structure resulting from the collaborative model of copy-edit-merge with branching does not correspond to a tree due to the presence of merges but rather to a DAG (directed acyclic graph).

bility to develop parallel streaks of research), both properties essential to research activities.

In abiding by the asynchronous approach towards collaborative research, we also intend to contribute to an incipient trend of expansion of VCSs from the Software Development community to other areas of group-based data creation —in particular, scientific research[4].

Furthermore, as pointed out in Section 1, we consider that a significant increase in research quality can be obtained as a result of rich and structured versioning metadata, which leads to better reproducibility as well as it facilitates aggregation across research projects. In many cases, annotation metadata can be utilized in order to better understand the phenomenon at stake.

3. APPLICATION-BASED APPROACH TOWARDS THE MANAGEMENT OF ANNOTATION DATA

In order to make use of the asynchronous collaborative features described in Section 2.2, we propose an application design which we have tailored to the specific needs of ongoing collaborative corpus-based research —see Section 1.

While we acknowledge the maturity and availability of open source VCS implementations such as “git”, “mercurial”, or “subversion”, we have instead chosen for an in-house implementation of the VCS features that are needed (branching, merging and versioning). The motivation is twofold. On the one hand, by relying on our own implementation we are able to reduce the complexity of current off-the-shelf VCS tools —although at the price of losing features, and adapt to the concrete needs (which are comparably less extensive than in software projects). On the other hand, given that the creation of annotation metadata in corpus-based research is tightly linked to the retrieval of corpus data from a (possibly remote) centralized corpus engine, rolling out a dedicated application allows us to more easily connect the VCS module with the Corpus Query Engine[5].

Figure 1 visualizes the general architecture of the application. Our application follows a client-server architecture implemented on top of current web standards —WebSockets, LocalStorage —as well as a NoSQL database. The client is a browser-based interface offering both a Corpus Query Interface and an Annotation Interface which interact with the local server through HTTP. At the server side, there are two components responsible respectively for the dynamic (annotation metadata) and the static (corpus data) application storage layer. We plan to release the application as an open-source project once developement reaches alpha state.

In the following sections, we describe the three layers of our application: the query engine module (Section 3.1), the version controlled metadata storage (Section 3.2), and the client interface (Section 3.3). Finally, in Section 4, we point at future research developements that arise from our infrastructure design.

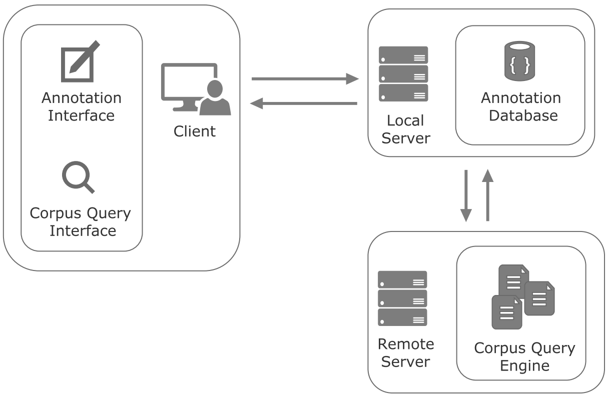

Figure 1: Client-server architecture of the app

3.1 Corpus Query Engine

A concoordancer or corpus query engine provides the entry point to the static layer of the application: the corpus data. Currently, we are using the Java-based corpus retrieval engine BlackLab[6], which is an extension to the full-text search engine Apache Lucene and provides ample and flexible possibilities for indexing corpus documents alongside rich metadata. As shown in Figure 1, the corpus query engine may be hosted at a separate server. User queries are forwarded to the remote server according to a REST Application Programming Interface (API) specified by the corpus query engine server application. For instance, a server implementation of BlackLab is available[7] allowing for client queries to be forwarded as

```
http://server/webservice/corpus/resource?parameters
```

where:

server/webservice is the URL of the web service,

corpus is the corpus to be queried,

resource is the type of information to be retrieved (hits, entire documents, etc..),

parameters is a sequence of URL encoded query strings specifying a query expression, whether to sort the results, whether to filter the results according to metadata, etc...

Provided a server implementation for a given corpus query engine, our application can be easily extended[8] in order to work with that corpus query engine as the backend. This way we aim to foster for more **reusability** corpus software development.

[4] This trend has pointed out by Brian Doll, CEO of the online network for software version control GitHub in an recent interview [2].

[5] However, we note that these two application levels represent logically separated concerns. As we will see, our implementation takes this into account by enforcing decoupling and modularity in the design of the application. This is in contrast to current corpus application developers, among which end-to-end monolithic designs are preferred —see for instance CQPWeb [8], ANNIS [17].

[6] See `https://github.com/INL/BlackLab`.

[7] See `https://github.com/INL/BlackLab-server`.

[8] In the next section we will review the minimal assumptions that the annotation database make on the structure of the corpus data.

3.2 Annotation Database

The local server is in charge of storing and retrieving the user metadata. Once the corpus data is retrieved from the corpus query engine, the local server retrieves the corresponding user annotations and merges them with the corpus data before sending the result to the client. Additionally, it also hosts the implementation of the centralized VCS features (branching, merging and versioning)[9] provided by the application.

3.2.1 Annotation Metadata Model

Our annotation metadata model is based on simple key-value structures and pointers to the token positions to which the metadata refers. An annotation, therefore, consists of an annotation key and annotation value —e.g. for a token like “going”, we might have an annotation key “aspect” associated with an annotation value “progressive”. In order to match annotations to the tokens to they which refer, our model has to be provided with a unique id for each corpus token. Furthermore, to allow for fast we also require that these ids be ordered integers —with the order corresponding to the sequence of tokens in the corpus. Besides token annotations, our model also covers span annotations, which match annotation key-value pairs to span of adjacent tokens in the corpus, instead of single tokens.

3.2.2 Versioning

All annotations coming into the database are assigned a version. Similarly, modifications to existing annotations always store the previous annotation in a history - Versioning, - Branching, - Merging,

3.3 Web-based Client

Researchers working with large sequential datasets (such as text corpora) will most often find themselves analyzing chunks in the absence of concurrent work by other peers[10], meaning that updates will result in conflicts with lower probability than in other setups. We can take advantage of this fact in order to alleviate some aspects simulate more RT interaction. As we will see in the next Section, we also seek to improve user experience by reducing the complexity of actual implementations of VCSs to a minimal subset of neccessary features.

Annotation visualization is tailored to (KWIC with token and span annotations) Enabling real time interaction between users.

3.4 Use Cases

How our application addresses the problems specified in Section 1. Perhaps this should stay in Section 2.3.

3.5 Limitations

Data Model, refer to RelANNIS and Ziggurat [10, 6].

By decoupling annotation app from the backend search engine, we lose the power to incorporate the body of ongoing annotations. While we acknowdlege this to be an important limitation of our current architecture, it need also be pointed out that current corpus search engines are built around static corpus representations, thus not allowing for incremental indexing of new metadata.

4. FUTURE RESEARCH

Integration with other tools, being agnostic with respect to which search engine is used, providing a simple API (only requirements are that the backend allows to retrieve data via a communication protocol such as HTTP or RPC and that tokens are given an id corresponding to their position in the corpus.

5. CONCLUSIONS

6. ACKNOWLEDGEMENTS

7. REFERENCES

[1] K. Altmanninger, M. Seidl, and M. Wimmer. A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304, 2009.

[2] A. Begel, J. Bosch, and M. A. Storey. Social networking meets software development: Perspectives from GitHub, MSDN, stackExchange, and top coder, 2013.

[3] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion.* 2011.

[4] Dropbox. Dropbox.

[5] Dropbox. Dropbox Help Center. Sharing files and folders.

[6] S. Evert and A. Hardie. Ziggurat: A new data model and indexing format for large annotated text corpora. *Challenges in the Management of Large*, 2015.

[7] Google. Google Docs.

[8] A. Hardie. CQPweb - combining power, flexibility and usability in a corpus analysis tool. *International Journal of Corpus Linguistics*, 17(3):380–409, 2012.

[9] A. Imine. Coordination Model for Real-Time Collaborative Editors. pages 225–246. Springer Berlin Heidelberg, 2009.

[10] T. Krause and A. Zeldes. ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, 31(1):118–139, apr 2016.

[11] J. Loeliger and M. McCullough. *Version Control with Git.* O’Reilly, 2012.

[12] Microsoft. Office Online.

[13] I. Rowlands, D. Nicholas, B. Russel, N. Canty, Watkinson, and Anthony. Social media use in the research workflow. *Learned Publishing*, 24(3):183–195, 2011.

[14] C. Sun and E. Clarence. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68. ACM, 1998.

[15] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, 1998.

[9] See Section 2.2.1 for the distinction between centralized and distributed VCS systems.

[10] This is not to say that the analysis can be carried out outside the scope of a collaborative system since updates still have to be documented and brought to the attention of other users.

[16] M. Wood. Collaborative Lab Reports with Google Docs. *The Physics Teacher*, 49(3):158, 2011.

[17] A. Zeldes, J. Ritz, A. Lüdeling, and C. Chiarcos. ANNIS : a search tool for multi-layer annotated corpora. *Proceedings of Corpus Linguistics 2009*, 2009.