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-disciplinary field of Digital Humanities, involves computer-aided manual processing of large text corpora. Typically, corpus instances are retrieved with the help of concordancers and textual search engines and subsequently labeled by hand before being submitted to quantitative or qualitative analysis. While well-established software solutions already exist for the corpus data retrieval aspect, less attention has been paid to the annotation process in terms of both software facilities and best practices, especially in the context of collaborative research. However, with the increase in size and scope of research projects we envisage new needs for synchronizing interdependent analysis metadata by different researchers. Current ad-hoc solutions to collaborative corpus analysis and annotation typically involve general-purpose Real-Time Editing (RTE) and cloud storage software, whose functionality are arguably sub-optimal for research purposes. In the present paper we discuss potential problems related to coordinating annotations in large-scale projects, alternative solutions as well as the potential benefits that can be derived from a dedicated approach to annotation data management. As a proof of concept, we also 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 Annotation, Asynchronous Collaboration, Groupware, User Interfaces

*This research was supported by ...

DATECH 2016 Poznań, Poland

1. INTRODUCTION

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

In a context in which the responsibility for the design of the research is not shared amongst several researchers, decisions as to how to create, design and manage annotation metadata may be considered unproblematic. However, with the current increase in size and scope of research projects in Digital Humanities and related fields, 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.

Although to our knowledge no prior studies have addressed the issue —both in terms of established practices and software solutions—, it can be assumed that a large part of current research requiring support for a collaborative environment typically resorts to (i) general-purpose Real-Time document Editing (RTE) software in the best case[1], (ii) cloud-storage services such as DropBox, which provide versioning but do not support proper collaborative editing of documents[2], or (iii) simply avoiding direct collaboration by implementing ad-hoc heuristics —for instance, splitting the total annotation load in non-overlapping chunks to be annotated by single researchers.

While RTE software solutions represent powerful and thoroughly tested tools, there are also a number of drawbacks associated with their use in research. First, their functionality may still be too general and overshoot the specific needs of researchers, more specialized than those of common end-users. For instance, 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.

[1]See, for instance, [10] and [13], for examples of reports on using Google Docs for research.
[2]For instance, DropBox explicitly recommends users to modify shared documents outside the scope of the synchronization functionality to avoid the creation of conflicting copies —see https://www.dropbox.com/en/help/36 .

Beyond these drawbacks, general metadata inconsistency issues still have to be explicitly addressed. In general, inconsistency may arise for reasons that are both intrinsic and extrinsic to the 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 system, which may not be given as closed set tagging system 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 interfering with their own analyses.
- Allowing changes to existing annotations to be backtraceable 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 synchronization issues, 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.
- Storage of metadata in a database, which can be further indexed with a search engine enabling advanced querying of the metadata.
- Visualization of metadata both as time snapshot and evolution over time.
- Enabling meta-research on research practices through structured metadata derived from the annotation activities.

In the present paper we sketch an application-based approach addressing the aforementioned issues in a way that fits better the concrete needs of the practitioner. 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, alternative approaches to concurrent multi-user document editing —RTE and VCS systems. Next, in Section 2.3, we argue that VCS asynchronous features are better suited for enabling the creation of synchronized corpus metadata. In Section 3, we describe the research infrastructure that we have implemented following the design principles discussed 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 or Microsoft's Office Online.

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 immediately 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 flavor of the collaborative editing problem, which can be describe in terms of the following two-step procedure [7]: first, local changes are immediately 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 [12]: (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 propagated 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) [11], a technique that consists of 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 data-structures, typically lists of characters, but it is possible to

work with list of elements at other levels, such as words, paragraphs or XML nodes [7, 11]. For instance, in a RTE annotation application, one might want to consider labels (annotation keys and values) instead of characters as the 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 [9]. In order to guarantee consistency maintenance 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 flavor 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 propagation 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 sacrificed 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 successful 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 [4]. 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 follows: 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 split 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 commit. Moreover, the asynchronous nature of VCS commits allows to attach metadata to changes before commiting. This is in opposition to RTE document edits that are triggered automatically after any modification and propagated to other users.

Besides enabling responsibility 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 RTE systems 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

[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 Directed Acyclic Graph (DAG).

metadata, since it favors explicitness 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 possibility 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, leading to better reproducibility as well as facilitating data aggregation across research projects. Moreover, annotation metadata can also be utilized in order to better understand the dynamics of the annotation process enabling meta-research.

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 our 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 development reaches alpha state.

[4]This trend has been 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 development, which seems to prefer end-to-end monolithic designs —see for instance CQPWeb [6], ANNIS [14].

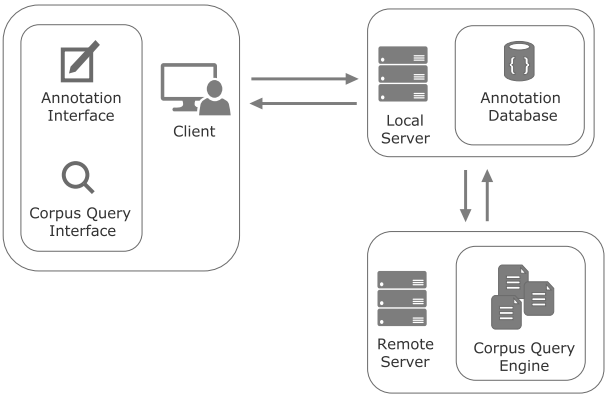

Figure 1: Client-server architecture of the app

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 developments that arise from our infrastructure design.

3.1 Corpus Query Engine

A concordancer 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], but, as we will see, our application design is not coupled to any particular backend software. 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 (REST API) specified by the corpus query engine server application. Resulting query matches are then merged with the annotations in the local server and transformed into a general representation independent from the output of the particular query engine in use. For instance, a server implementation of BlackLab is available[7], allowing for client queries to be commanded through HTTP 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...

[6]BlackLab is an extension to the full-text search engine Apache Lucene and provides ample and flexible possibilities for indexing corpus documents alongside rich metadata; see `https://github.com/INL/BlackLab`.

[7]See `https://github.com/INL/BlackLab-server`. Another corpus query engine offering similar functionality is ANNIS3.

Provided a REST API 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 back-end. This way we aim to foster more **reusability** in corpus software development.

3.2 Annotation Database

The local server is in charge of storing and retrieving the annotation metadata. Once the corpus data are 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[9] VCS features (versioning, branching, and merging) provided by the application.

3.2.1 Annotation Metadata Model

Our annotation metadata model is compatible with the CQP data model [3] and it is based on simple key-value structures and pointers to the token positions to which the metadata refers. An annotation, therefore, consists of an mapping from an annotation key to an 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 with the tokens to they which refer, our model has to be provided with a unique id for each corpus token.

Furthermore, to speed up annotation queries 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, Branching and Merging

All annotations coming into the database are assigned a version. Similarly, previous values of an annotation key are stored in the annotation version history with metadata about the author and time of the edit as well as the corpus query that retrieved the target token(s). As a further layer of consistency maintenance, users are only allowed to commit their changes to other user's annotations depending on user roles and write rights assigned by the application during setup time —i.e. if the user does not have necessary rights the update is considered to stay in conflict with the current project version and both the author of the current annotation and the author of the edit have to decide on the merge operation.

While the described work pipeline enables basic versioning capabilities, it does not exploit the concept of working copy in order to support the flexibility of VCS features as described in Section 2.2.1. In particular, it is desirable to allow parallel work —circumventing the conflicts that may thereby arise— and delay merging to a later point. Therefore, we incorporate working copies through the operation of branching in the following way. At any point, a user may create a separate branch and start modifying annotations independently of annotation activities by other researchers as well as write rights. Whenever the parallel work in the branch is finished, a global merge operation with the original trunk can be issued. This operation triggers a three-way merging algorithm in the local server which will automatically incorporate non-conflicting changes and prompt users involved in conflicts for a merge solution. As mentioned in Section 2.3, user defined metadata is attached to conflict resolutions to allow for better informed state restoration.

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

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

3.3 Web-based Client

The last component of our application is a browser-based web interface that provides access to and interaction with the search engine and the annotation database. Moreover, it is designed to provide real-time multi-user interaction, keeping users informed of parallel annotations by other users, notifying and logging annotation conflicts as well as highlighting possible inconsistencies in the tagging system.

3.3.1 Interface

Access to corpus data is provided through a query panel which with an input field and several controllers for sorting, filtering and paginating —core functionalities provided by most corpus query engines. Hits fetched from the corpus query engine are displayed as Keyword In Context (KWIC), with the words matched by the query surrounded by a user defined context window. Annotations corresponding to tokens or token spans in the KWIC hit are shown below the hit as interlinear glosses, following current practices in linguistics. Moreover, the author of the annotation is highlighted and version history is displayed in place on-demand.

3.3.2 Real-time Multi-user Interactivity

As a means to provide users with a way of controlling the amount of live feedback shown in the client interface, we use the publish-subscribe notification pattern together with a particular metadata to which we refer as "projects".

The pub/sub pattern refers to a messaging pattern where messages are not directly forwarded to specific receivers but instead are assigned a class to which other users can subscribe in order to retrieve the message. In our application, users can subscribe to and unsubscribe from events related to annotations within a particular "project".

Projects, respectively, are metadata that underlie annotations and group them into disjoint sets. Besides providing the message classes for the pub/sub notification pattern, structuring annotations around projects represents an important asset that helps organizing the application logic at several levels. For instance, users can create new projects to address particular research questions and assign other users particular write rights. Furthermore, given that each annotation is provided with project metadata, annotation metadata queries can be done in a more specific way. Finally, our application implements the branching functionality described in Section 3.2.2 around projects, such that as a result of branching, a sibling project is created to which other users can subscribe.

In order to manage the complexity associated with the multi-client interface in our application, we leverage HTML5 standards such as the WebSocket protocol for bi-directional client-server communication and the LocalStorage specification for in-session storage. Furthermore, we utilize a modern user interface framework ReactJS, which uses functional reactive programming features to ease the synchronization

of application state with the backend model— for instance, the client view of the annotation database with its current status.

4. RELATED WORK, LIMITATIONS AND FUTURE WORK

Our implementation of an asynchronous collaborative annotation interface is still in development and certain aspects are subject to improvement. In particular we would like to highlight the following two points.

First, while the simplicity of our annotation data model provides an easily maintainable and implementable abstraction, it does not, however, reflect current advances made by other researchers with respect to corpus data modeling. For instance, authors related to the ANNIS project have presented a generic, multi-layer, graph-based data model known as RelANNIS [8] that aims at covering an extensive set of annotation types (including sub-token, token, span and tree annotations). Similarly, members of the CQP development team have recently released a working paper describing the data model —known as Ziggurat [5]— to be used in a future release of the CQP engine. Similarly to RelANNIS, Ziggurat describes a multi-layer data model anchored at the token level that allows for the incorporation of tree and graph annotations. In the future, our annotation data model should be extended to be compatible with their specifications (in particular, tree and graph structures).

Secondly, decoupling annotation application logic from the corpus query engine implies that users cannot reflect the on-going body of annotations in their queries. This is, however, in pair with the status of current corpus query engines which do not allow for incremental indexing of new metadata. In the short-term, a less cumbersome solution can be implemented in terms of on-demand and off-line incremental indexing —i.e. asynchronously triggering corpus reindexing to include the newly produced annotation metadata from the annotation database.

5. CONCLUSIONS

In the present paper, we have discussed the necessity of specifically addressing issues arising in a research context where annotation metadata are being created by multiple independent researchers. We have pointed out that current solutions either do not directly support multi-user synchronization —cloud-storage software, ad-hoc heuristics—, or when they do, they are not fully suited for handling inconsistencies in collaboratively created analysis metadata —RTE systems. As a precondition to the design of an appropriate collaborative tool, we have discussed the principles behind RTE and VCS systems, highlighting to what extent they represent respectively the synchronous and asynchronous alternatives to the collaborative document editing problem. As a result of our discussion we concluded that research metadata curation in a collaborative environment is better covered with the asynchronous features of versioning, branching and merging. Finally, we have showcased our particular implementation of a asynchronous collaborative application, which encompasses (i) a remote corpus query engine, (ii) a version controled annotation storage and (iii) an interactive multi-user interface for collaborative annotation. Finally, our design pledges for modularity and reusability in corpus software development.

6. ACKNOWLEDGMENTS

The first author wants to thank Chris Emmery for useful discussions on previous drafts of this work.

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, Stack Exchange, and TopCoder, 2013.

[3] O. Christ, B. M. Schulze, A. Hofmann, and E. Koenig. The IMS Corpus Workbench: Corpus Query Processor (CQP): User's Manual. Technical report, University of Stuttgart: Institute for Natural Language Processing, Stuttgart, Germany, 1999.

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

[5] 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.

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

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

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