Handling Disturbance and Awareness of Concurrent Updates in a Collaborative Editor

Weihai Yu, Gérald Oster, Claudia-Lavinia Ignat

▶ **To cite this version:**

Weihai Yu, Gérald Oster, Claudia-Lavinia Ignat. Handling Disturbance and Awareness of Concurrent Updates in a Collaborative Editor. CDVE 2017 - 14th International Conference on Cooperative Design, Visualization, and Engineering, Sep 2017, Mallorca, Spain. hal-01652656

HAL Id: hal-01652656
https://inria.hal.science/hal-01652656

Submitted on 30 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Handling Disturbance and Awareness of Concurrent Updates in a Collaborative Editor

Weihai Yu[1], Gérald Oster[2,3,4], and Claudia-Lavinia Ignat[2,3,4]

[1] University of Tromsø - The Arctic University of Norway
[2] Inria, Villers-lès-Nancy, F-54600, France
[3] Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54506, France
[4] CNRS, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54506, France

Abstract. When people work collaboratively on a shared document, they have two contradictory requirements on their editors that may affect the efficiency of their work. On the one hand, they would like to know what other people are currently doing on a particular part of the document. On the other hand, they would like to focus their attention on their own current work, with as little disturbance from the concurrent activities as possible. We present some features that help the user handle disturbance and awareness of concurrent updates. While collaboratively editing a shared document with other people, a user can create a focus region. The user can concentrate on the work in the region without being interfered with the concurrent updates of the other people. Occasionally, the user can preview the concurrent updates and select a number of these updates to be integrated into the local copy. We have implemented a collaborative editing subsystem in the GNU Emacs[5] text editor with the described features.

Keywords: Data replication, concurrency control, CRDT.

1 Introduction

When people work collaboratively on a shared document or artifact, two contradictory aspects may affect the efficiency of their work. On the one hand, a person would like to know what the other people are currently doing with the document, so she can avoid duplicating the same work or working unnecessarily on conflicting ideas. On the other hand, the person would like to be as concentrated as possible on her current work without the interference of concurrent work of other people, particularly when her current work requires deep thinking and involvement.

One commonly applied practice is that people work independently (offline) on their own copies of the document. They are able to concentrate completely on their work, but have no idea what the other people are currently doing with the document. They must use some additional communication channels, such as emails, to exchange their copies of the document and get updated about the other people's work. Then, they can use some particular features of their editors, such as the "track changes" feature of Microsoft Word, or third-part tools, such as `diff`, to find out the particular updates made

[5] https://www.gnu.org/software/emacs

by the other people. They should also be coordinated, again with additional communication channels, for other tasks, such as who is responsible for integrating the concurrent updates in the following rounds. Without careful planning and coordination, they may risk unnecessarily duplicating their efforts on overlapping or conflicting work.

Another possible practice, which is more and more applied lately, is that people work simultaneously (online) on the shared document with a real-time collaborative editor, such as Google Drive[6]. People can see immediately what other people are doing with the document, so they can avoid duplicating their efforts. If several people are working simultaneously on the same area of the document, even if for non-overlapping or non-conflicting purposes, they can be easily distracted by concurrent updates of the other people. Furthermore, allowing concurrent partial updates to be integrated in local copies may cause other inconvenience such as compilation errors of software programs.

A middle ground is that people work on their own copies in isolation and are aware of other people's work at some specific time, such as at the time of saving the document or committing the updates to a shared repository.

We present yet another possibility: people can work collaboratively on the same document in such a way that they can limit the disturbance from other people's concurrent updates and at the same time have an overview of the concurrent work, all these in a controlled manner.

We have implemented support for collaborative editing in GNU Emacs, using CRDT (Commutative Replicated Data Types) and supporting selective undo [16]. In this paper, we present an experimental feature that lets the user handle the disturbance and awareness of concurrent updates.

This paper is organized as follows. Section 2 gives a short review on our CRDT approach to real-time collaborative editing [16]. Section 3 presents the new features for handling disturbance and awareness through an example. Section 4 describes briefly how we implemented the features in Emacs. Section 5 discusses related work. Finally, Section 6 concludes.

2 Document view and CRDT model

With a collaborative editor, a document is concurrently updated from a number of peers at different sites. Every peer consists of a view of the document, a model, a log of operation history and several queues (Figure 1).

A peer concurrently receives local operations generated by the local user and remote updates sent from other peers. Local user operations take immediate effect in the view. The peer stores executed view operations in Q_v and received remote updates in Q_{in}. During a synchronization cycle, it integrates the operations stored in Q_v and Q_{in} into the model and shows the effects of integrated remote updates in the view. The peer stores the integrated local operations in Q_{out}. It also records the integrated local and remote operations in the log. Later, it broadcasts the operations in Q_{out} to other peers.

Every peer has a unique peer identifier *pid*. An operation originated at a peer has a peer update number *pun* that is incremented with every integrated local operation.

[6] https://drive.google.com

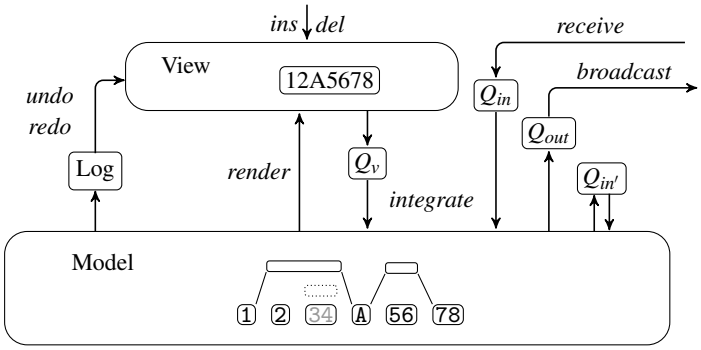

Fig. 1. View, model and operations

Therefore, we can uniquely identify an operation with the pair (pid, pun). We use op^{pid}_{pun} to denote an operation op identified with (pid, pun).

A view is mainly a string of characters. A user can insert or delete sub-strings in the view, and undo earlier integrated local or remote operations selected from the log.

A model materializes editing operations and relations among them. It consists of layers of linked nodes that encapsulate characters. Conceptually, characters have unique *identifiers* that are totally ordered. For two characters c_l and c_r, if $c_l.id < c_r.id$, then c_l appears to the left of c_r.

Nodes at the lowest layer of a model represent insertions and contain inserted characters. Nodes at higher layers represent deletions. That is, a higher-layer node (outer node) deletes the corresponding characters in the lower-layer nodes (inner nodes).

A node contains the identifiers of its leftmost and rightmost characters. The identifiers of the other characters (i.e. not at the edges of the node) are not explicitly represented in the model. An insertion node also contains a string of characters.

New operations may split existing nodes. Nodes of the same operation share an *op* element as the operation's descriptor. A descriptor contains, among other things, information about undos, which influences the visibility of the characters.

There are three types of links among nodes: *l*-*r* links maintain the left-right character order; op_l-op_r links connect nodes of the same operations; *i*-*o* links maintain the inner-outer relations. The outermost nodes and the nodes inside the same outer node are linked with *l*-*r* links. When the view and the model are synchronized, the view equals to the concatenation of all visible characters of the outermost nodes through the *l*-*r* links.

Figure 2 shows an example with three peers. The upper part shows the operations generated at the peers. The lower part shows the model snapshots at Peer 2. Nodes of the same deletion are aligned horizontally. Nodes with dotted border are invisible. Characters in light gray are invisible in the view.

Different peers can update the model concurrently. The model data structure is a CRDT that has the following convergence property: when all peers have applied the same set of updates, the states of the model at all peers converge.

We refer the interested readers to [16] for more details on the data structure and algorithms on the model.

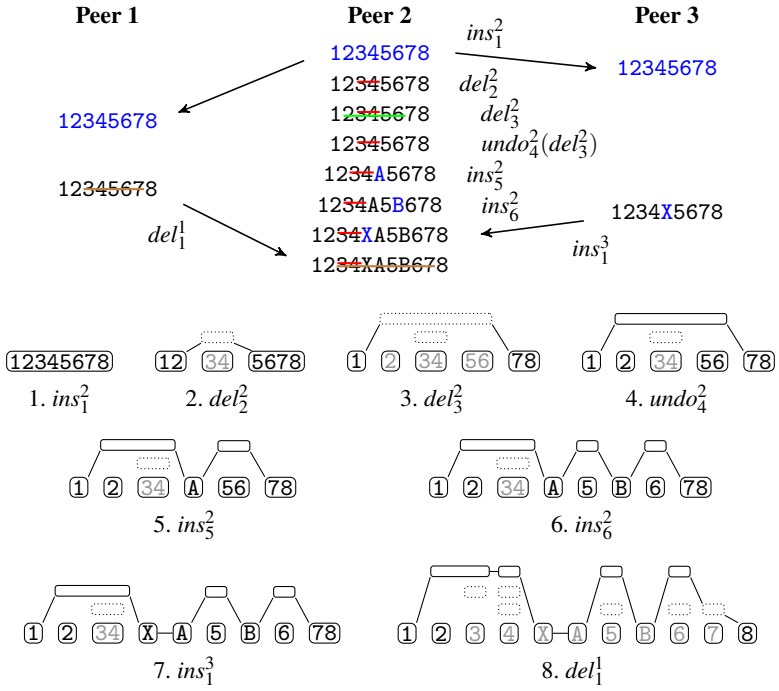

Fig. 2. Examples of model updates

Materialization of editing operations in the model makes a number of tasks easier, including support for selective undo and handling of operation dependencies [16], as well as user friendliness features like display of operation history and selection of operations to undo or redo, and preview of concurrent remote operations as described in the following sections.

3 Handling disturbance and awareness of concurrent updates

Emacs has a feature called "narrowing" that allows the user to focus on a specific region of the document, making the rest temporarily inaccessible. We think it is appropriate to augment the narrowing feature with non-disturbance control for collaborative editing, because when a user explicitly applies the narrowing, she signals the need to be focused and concentrated. No disturbance from the concurrent updates of the other users is clearly part of such concentration.

When a user narrows to a region while collaborating with other people, the editor starts a *focus region* and enters a non-disturbance mode. No concurrent updates in the region are immediately integrated.

While working in a focus region, the user may occasionally want to know what the other people are working on in this region. By knowing other people's work, she may

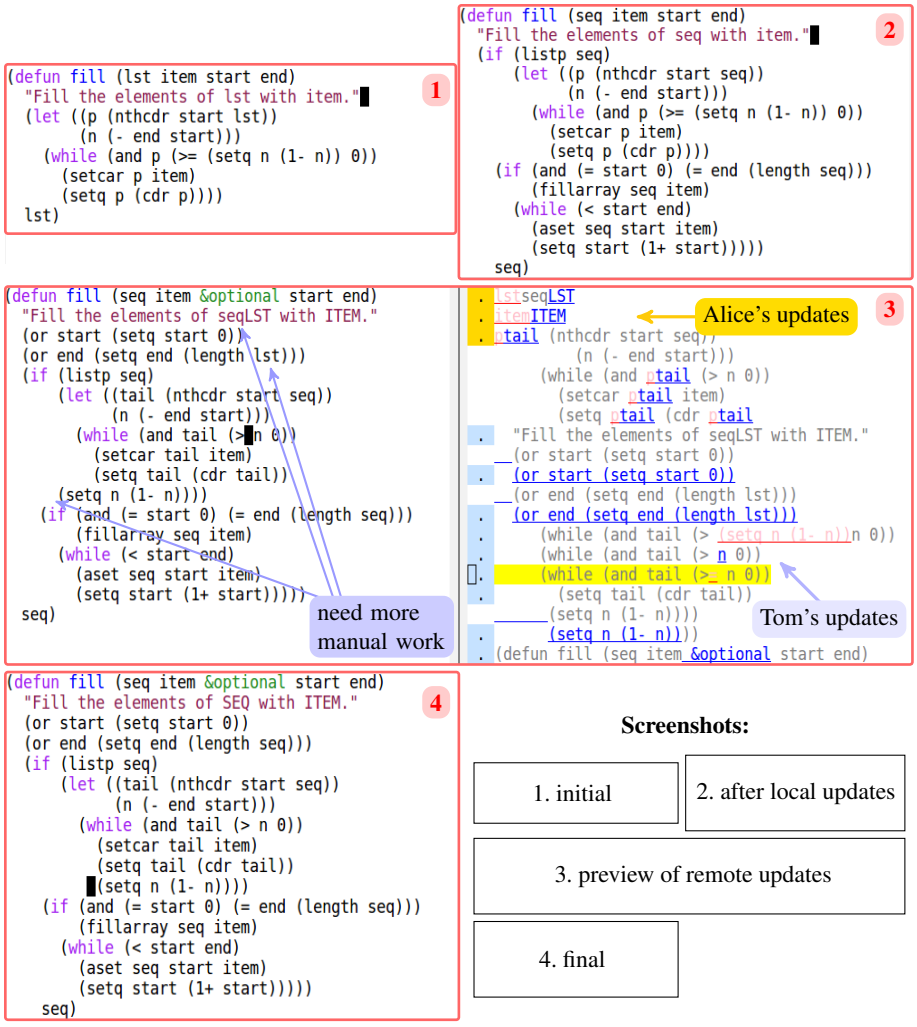

Fig. 3. An example of collaborative editing with awareness of concurrent updates

be able to avoid duplicated work. She may even selectively integrate some concurrent updates to adapt early to compatible updates performed by other people.

Figure 3 illustrates with an example the steps involved in working with a focus region. Suppose three programmers, Bob, Alice and Tom, work concurrently on function `fill()` (in Emacs Lisp), which initially fills a segment of a list with a given item (Step 1). The lower right part of the figure shows the layout of the screenshots of the different steps performed by Bob.

Bob would like to generalize `fill()` to sequences (in Emacs Lisp, sequence is a general type that includes list, vector etc.). He first narrows down to the function as a

focus region and works in the region without the disturbance of the concurrent updates generated by Alice and Tom.

When Bob is almost done (Step 2), he takes a preview of concurrent updates (Step 3). In the preview, every concurrent update is preceded with a leading dot. The color around the leading dots indicates who performed the updates. Notice that we support composite updates. For example, a text substitution consists of a deletion and an insertion; a global substitution consists of multiple deletions and insertions. The character strings of the updates are underlined. The insertion part of an update is displayed in blue and the deletion part is in pink. The surrounding text of the updates is also displayed. This helps the user recognize the context of the updates. Moreover, the editor keeps a mapping from the displayed updates to their positions in the document. The user can traverse up and down the updates in the preview. The current update is highlighted in yellow in the preview. The cursor in the main document, i.e. the `fill()` body in our example, moves accordingly. This helps the user further to localize the particular updates.

Bob noticed that Alice had capitalized two words in the doc-string and renamed variable `p` to `tail`. Meanwhile, Tom had refactored the code. He added two lines to deal with the default start and end positions in the list and moved the assignment of variable `n` from the loop condition to the end of the loop body.

Bob can traverse through the concurrent updates and selectively integrate some of them into the main document. In this particular example, he found that all updates performed by Alice and Tom were compatible with his work and opted to integrate all of them, which resulted in the `fill()` function as shown in Step 3. (Alternatively, Bob may reject some of the concurrent updates. For example, he may reject the renaming of variable `p`.)

Bob still had to do some manual work to get the final version as shown in Step 4.

4 Implementation

Emacs is a widely used open-source text editor. Emacs is also a run-time environment of Emacs Lisp. This make Emacs very suitable for customization and extension. We have earlier implemented the view and the CRDT model described in Section 2 in Emacs Lisp [16]. We have also implemented support for selective undo and selection of updates through a history view.

A focus region is marked with the identifiers (see Section 2) of the leftmost and the rightmost characters of the region. To show a preview of the concurrent remote updates, the peer integrates the updates stored in Q_{in} and displays those in the focus region in the history view. When the user accepts and rejects a number of selected updates, the peer marks the accepted ones and undoes the rejected ones.

When the user exits the preview, the peer handles the the accepted and rejected (i.e. undone) updates in the same way as it normally handles remote updates. For the updates that are neither accepted nor rejected (i.e. the ones which the user wants to deal with when she exits the focus region), the peer undoes them and inserts them in a new queue $Q_{in'}$ (Figure 1).

The next time the peer shows a preview of concurrent updates, it redoes the updates in $Q_{in'}$ and displays them, together with the new concurrent updates in Q_{in}.

When the user exits the focus region, she must accept or reject all updates in $Q_{in'}$.

5 Related Work

Most of the existing collaborative editing work is based on operational transformation (OT) [2, 11, 12]. A local operation is performed immediately on the local replica of a document; a remote operation is transformed and integrated in the local site based on the positions of the existing and concurrent operations. A critical issue with OT is that, it is hard to design correct operation transformation functions that is generally applicable to various integration algorithms [6]. One common way to relax certain required conditions for transformation functions is to enforce a global total order in which operations are transformed and integrated at all sites [14]. As a consequence, OT approaches practically require the involvement of central servers.

Lately, there appear a new family of approaches to collaborative editing based on commutative replication data types (CRDT) [1, 7–9, 13, 15, 16]. With CRDT, characters of concurrent insertions are ordered based on the underlying data structure. One benefit of CDRT is that different sites can integrate operations in different orders. Thus central servers are not necessary.

To achieve the same features as presented in this paper in OT is non-trivial. For example, for every remote operation, to detect whether it falls inside a focus region, we must first transform the operation. To preview an operation and keep a mapping to its position in the current document, we must also transform the operation. However, for these different purposes, we must transform the operations differently. With CRDT, the effect of an integration is materialized in the data structure, which can be used for different purposes, such as for selective undo [16] and for focus regions. To detect whether a remote update falls inside a focus region, we only need to compare the character identifiers of the operation and of the region.

Awareness during distributed collaboration has got wide attention for many years, such as in the context of collaborative software development [3, 4, 10]. A recent study suggests that "developers would appreciate having access to awareness information frequently but not in real time; they have, however, diverse preferences regarding the level of detail in which such information should be made available" [3]. Without the mechanisms for collaborative editing, tools typically provide information about concurrent updates with coarse granularity, such as at a per-file level [10]. On the other hand, they may take advantage of the capabilities of software development environments to detect conflicts that are specific to programming languages or platforms [10].

In our previous work, we were able to provide similar awareness features as presented in this paper [4, 5]. Both [4] and [5] applied OT to localize concurrent remote updates in the context of local document states. In [4], we combined OT with a central version control repository. Because local and remote replicas share some common base version of the document, we need only to maintain operation histories after the base version. Thus the lengths of the operation histories are typically short and the OT algorithms do not have a significant run-time overhead. In [5], we focused on providing awareness without compromising privacy. For the awareness part, we modeled the document with a layered structure. The operation history of a document consisted of

multiple shorter sub-histories. In that way, we were able to overcome the performance drawbacks of the OT algorithms. None of [4] and [5] supported disturbance avoidance, since the focus was on awareness, rather than integration of concurrent updates.

6 Conclusion

We have presented some features for handling disturbance and awareness of concurrent updates during collaborative editing. We implemented the features as part of our collaborative editing subsystem in GNU Emacs. Our implementation benefited from the CRDT model that materializes the operations and their relations. We plan to evaluate these features in real use scenarios and make improvements based on the evaluation.

References

1. L. André, S. Martin, G. Oster, and C.-L. Ignat. Supporting adaptable granularity of changes for massive-scale collaborative editing. In *CollaborateCom*. IEEE, 2013.
2. C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *SIGMOD*, pages 399–407. ACM, 1989.
3. H. Estler, M. Nordio, C. A. Furia, and B. Meyer. Awareness and merge conflicts in distributed software development. In *ICGSE*, pages 26–35, 2014.
4. C. Ignat and G. Oster. Awareness of concurrent changes in distributed software development. In *CoopIS/DOA/ODBASE (1)*, pages 456–464, 2008.
5. C. Ignat, S. Papadopoulou, G. Oster, and M. C. Norrie. Providing awareness in multi-synchronous collaboration without compromising privacy. In *CSCW*, pages 659–668, 2008.
6. A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving correctness of transformation functions in real-time groupware. In *ECSCW*, pages 277–293, 2003.
7. G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for P2P collaborative editing. In *CSCW*, pages 259–268. ACM, 2006.
8. N. M. Preguiça, J. M. Marquès, M. Shapiro, and M. Letia. A commutative replicated data type for cooperative editing. In *ICDCS*, pages 395–403. IEEE Computer Society, 2009.
9. H.-G. Roh, M. Jeon, J. Kim, and J. Lee. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel Distrib. Comput.*, 71(3):354–368, 2011.
10. A. Sarma, D. F. Redmiles, and A. van der Hoek. Palantír: Early detection of development conflicts arising from parallel code changes. *IEEE Trans. Software Eng.*, 38(4):889–908, 2012.
11. 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 Trans. Comput.-Hum. Interact.*, 5(1):63–108, 1998.
12. D. Sun and C. Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(10):1454–1470, 2009.
13. S. Weiss, P. Urso, and P. Molli. Logoot-undo: Distributed collaborative editing system on P2P networks. *IEEE Trans. Parallel Distrib. Syst.*, 21(8):1162–1174, 2010.
14. Y. Xu and C. Sun. Conditions and patterns for achieving convergence in OT-based co-editors. *IEEE Trans. Parallel Distrib. Syst.*, 27(3):695–709, 2016.
15. W. Yu. Supporting string-wise operations and selective undo for peer-to-peer group editing. In *GROUP*, pages 226–237. ACM, 2014.
16. W. Yu, L. André, and C. Ignat. A CRDT supporting selective undo for collaborative text editing. In *DAIS*, pages 193–206, 2015.